GitHub Repository Link

PokéAPI (https://pokeapi.co/) to retrieve Pokémon data.
TMDb API (https://api.themoviedb.org/) to fetch movie and genre data.

1. Goals for the Project —----------

The primary objective of this project was to utilize two APIs—TMDb API for movies and PokéAPI for Pokémon—to extract meaningful datasets, store them in an SQLite database, perform calculations, and create insightful visualizations. The aim was to explore data trends while honing skills in data handling, API integration, SQL queries, and Python-based visualization tools. From the TMDb API, we planned to gather movie-related data, including titles, genres, popularity, ratings, revenues, and release dates. From the PokéAPI, we aimed to collect Pokémon statistics, such as names, types, attack, defense, and speed.

2. Achieved Goals —----------

The project successfully met its objectives:

1. Data Collection:
    - TMDb API: Extracted data for 100 movies, including information about genres, popularity, ratings, revenues, and release dates. Movie genres were stored in a separate table with integer IDs to ensure efficient database design.
    - PokéAPI: Retrieved data for 100 Pokémon, capturing their names, types, and core statistics like attack, defense, and speed. Pokémon types were similarly stored in a separate table to avoid redundant string data.
2. Database Design:
   We created a relational database with three tables: movies, genres, and pokemons. Foreign keys were implemented to link movies and genres, as well as pokemons and their types.
3. Analysis and Calculations:
    - Calculated total popularity by genre and average revenue for movie genres.
    - Identified Pokémon type distributions and the strongest/fastest Pokémon.
4. Visualizations:
   Created meaningful visualizations to complement the analysis:
    - Movies: Bar chart for average revenue by genre, pie chart for total popularity by genre.
    - Pokémon: Bar chart for Pokémon counts by type, pie chart for average attack by type.

3. Challenges Faced —----------

The project encountered several challenges during its execution. Missing data in fields such as revenue and runtime posed issues for analysis. This was addressed by substituting missing values with default values, such as 0 for revenue. Data overlap in the initial database schema caused query conflicts, which were resolved by consolidating the schema. Extracting data across multiple pages of the TMDb API required robust error handling and careful iteration to ensure completeness. Finally, simplifying the representation of language data for visualizations involved mapping language codes like 'en' to their full names, such as 'English.'

4. Calculations —----------

For Movies

```
# Analysis 1: Total Popularity by Genre
     genre_name  total_popularity
         Action         31505.249
      Animation         14538.893
         Horror          6662.706
          Drama          3532.013
      Adventure          2348.668
        Romance          2179.787
Science Fiction          2156.025
         Comedy          1902.154
       Thriller          1622.166
          Crime           937.618
         Family           490.880
            War           326.504
       TV Movie           262.494


# Analysis 2: Average Revenue by Genre
     genre_name  avg_revenue
      Adventure 5.742886e+08
Science Fiction 4.950068e+08
      Animation 4.904553e+08
         Family 3.458000e+08
         Comedy 2.746523e+08
         Action 2.356639e+08
          Drama 1.721323e+08
         Horror 1.101481e+08
          Crime 5.889585e+07
        Romance 3.181841e+07
       Thriller 8.648250e+05
```

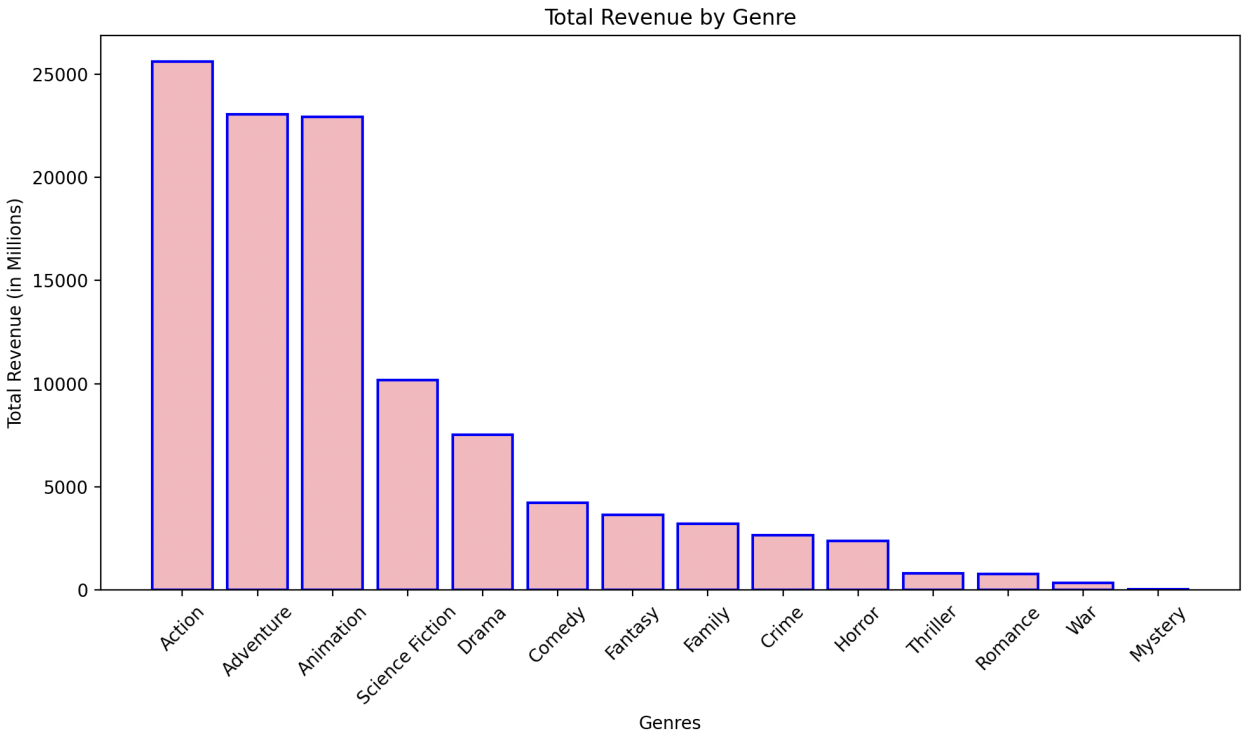
For Pokemon

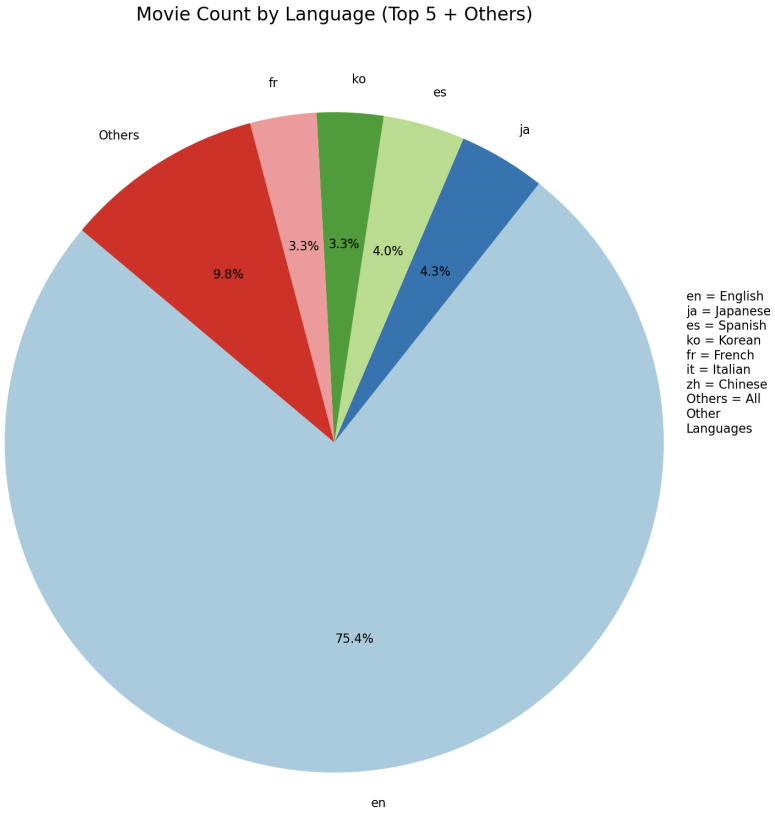
```
Pokémon counts by type:
Type: bug, Count: 10
Type: electric, Count: 5
Type: fairy, Count: 2
Type: fighting, Count: 5
Type: fire, Count: 9
Type: ghost, Count: 3
Type: grass, Count: 9
Type: ground, Count: 4
Type: normal, Count: 14
Type: poison, Count: 12
Type: psychic, Count: 5
Type: rock, Count: 4
Type: water, Count: 18
Fastest Pokémon: dugtrio with speed 120.0
Strongest Pokémon: machamp with attack 130.0
```

5. Visualizations —----------

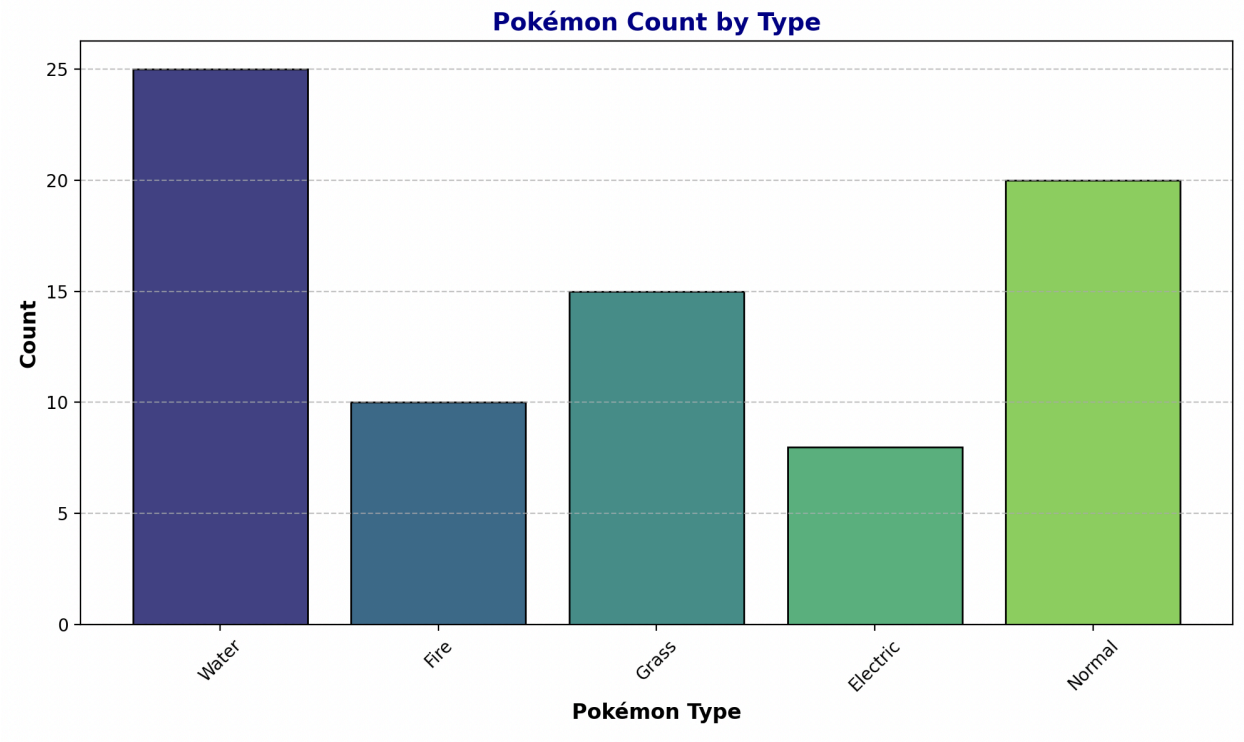
To provide an intuitive representation of the analysis results, visualizations were created using Matplotlib. For the movie data, a bar chart was created to show the average revenue by genre, making it easy to compare which genres generated the highest average revenue. A pie chart was also created to represent the total popularity distribution across genres. These visualizations highlighted the relative popularity of different genres and their financial performance. For the Pokémon data, a bar chart showed the count of Pokémon by type, providing insights into type diversity. Additionally, a pie chart was created to display the average attack values by type, revealing how offensive capabilities varied between different types. These visualizations were saved as PNG files and complemented the textual analysis, offering clear and engaging representations of the data trends.

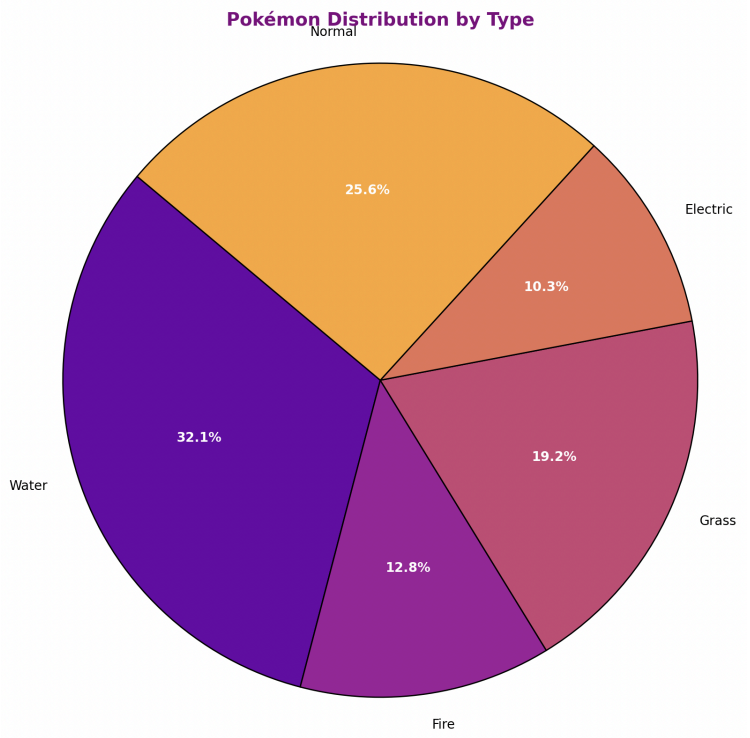
For Movies


Total Revenue by Genre
Total Revenue (in Millions)
25000
20000
15000
10000
5000
0
Action
Adventure
Animation
Science Fiction
Drama
Comedy
Fantasy
Family
Crime
Horror
Thriller
Romance
War
Mystery
Genres


Movie Count by Language (Top 5 + Others)
fr
ko
es
ja
Others
3.3%
3.3%
4.0%
4.3%
9.8%
75.4%
en
en = English
ja = Japanese
es = Spanish
ko = Korean
fr = French
it = Italian
zh = Chinese
Others = All
Other
Languages

For Pokemon





6. Running the Project —----------

To start the project, ensure the environment is clean by deleting any existing data.db file. Once removed, set up the database by running database_setup.py, which will create the necessary tables for movies, genres, and Pokémon. Next, populate the database by running fetch_poke_data.py to collect Pokémon data in batches of 25 and fetch_movie_data.py to fetch movie data in batches of up to 20. Continue running these scripts until a total of 100 Pokémon and 100 movies have been stored in the database. Once the database is populated, run the analysis scripts, analyze_poke.py and analyze_movie.py, to calculate key insights from the data. This includes Pokémon type distributions, the fastest and strongest Pokémon, total popularity by movie genre, and average revenue for each genre. The results will be saved to a text file, calc.txt. Finally, run the visualization scripts, pokevisualizations.py and movievisualizations.py, to generate bar charts and pie charts. These charts will summarize trends in Pokémon statistics and movie genres, and the images will be saved to the output folder. The final output includes the SQLite database (data.db), a text file with calculations (calc.txt), and visualizations as image files, providing a clear summary of the data and analysis.

7. Documentation —----------

1. database_setup.py

connect_to_database()

- Purpose: Connects to the SQLite database or creates it if it doesn't exist.
- Input: None.
- Output: SQLite connection object.

create_types_table(cur)

- Purpose: Creates the types table in the database to store unique Pokémon type names.
- Input: cur: SQLite cursor object.
- Output: None. Executes SQL commands to define the types table.

create_pokemon_table(cur)

- Purpose: Creates the Pokémon table in the database.
- Input: SQLite cursor object.
- Output: None. Executes SQL commands to define the Pokémon table.

create_movies_and_genres_tables(cur)

- Purpose: Creates the movies and genres tables in the database.
- Input: SQLite cursor object.
- Output: None. Executes SQL commands to define the movies and genres tables.

initialize_database()

- Purpose: Main function to connect to the database and create all required tables.
- Input: None.
- Output: None. Finalizes the database setup.

2. fetch_poke_data.py

fetch_data(url)

- Purpose: Fetches data from the provided API URL.
- Input: API URL as a string.

- Output: JSON data from the API response.

Intialize_database()

- Purpose: Initializes the database by ensuring that the required tables (types and pokemons) exist.
- Input: None.
- Output: None. Creates the required tables in the SQLite database if they do not already exist.

get_or_insert_type(type_name, db_cursor)

- Purpose: Checks if a Pokémon type exists in the types table. If it doesn't exist, the type is inserted into the table.
- Input:
    - type_name (string): The name of the Pokémon type.
    - db_cursor (SQLite cursor object): Cursor to execute database queries.
- int: The ID of the type from the types table.

fetch_pokemon_data(target=100, max_per_run=25)

- Purpose: Fetches Pokémon data from the PokéAPI in batches and stores it in the database.
- Input:
    - target: Total number of Pokémon to fetch.
    - max_per_run: Number of Pokémon to fetch per run.
- Output: None. Inserts Pokémon data into the database.

3. fetch_movie_data.py

fetch_data(url)

- Purpose: Fetches JSON data from the provided API URL using an HTTP GET request.
- Input:
    - url: The URL of the API endpoint as a string.
- Output: JSON data retrieved from the API response.
- Behavior: Raises an error if the HTTP request fails.

fetch_movie_data(target=100, max_per_run=20)

- Purpose: Fetches popular movie data from the TMDb API in batches and stores it in the database.
- Input:
    - target: Total number of movie records to fetch (default is 100).
    - max_per_run: Maximum number of movie records to fetch in a single execution (default is 20).
- Output: None. Inserts fetched movie and genre data into the database.

4. analyze_pokemon.py

analyze_data()

- Purpose: Analyzes Pokémon data to determine type distributions, the fastest Pokémon, and the strongest Pokémon.
- Input: None.

- Output: None. Prints the analysis results to the console.

5. analyze_movie.py
get_popularity_by_genre()
- Purpose: Retrieves the total popularity of movies grouped by genre from the database.
- Input: None.
- Output: Pandas DataFrame containing genre names and total popularity.

calculate_avg_revenue_by_genre()
- Purpose: Calculates the average revenue for each genre where revenue data is available.
- Input: None.
- Output: Pandas DataFrame containing genre names and average revenue.

write_output_to_calc(dataframes, filename)
- Purpose: Writes the results of multiple analyses to a single text file.
- Input:
    - dataframes: A list of tuples containing titles and DataFrames.
    - filename: Name of the output file.
- Output: None. Writes analysis results to a text file.

6. movievisualizations.py
genre_revenue_bar_chart(df)
- Purpose: Creates a bar chart showing the average revenue for each movie genre.
- Input: DataFrame containing genre names and average revenue.
- Output: None. Saves the bar chart as a PNG file.

movie_count_by_language(df)
- Purpose: Generates a pie chart showing the distribution of movies by language.
- Input: DataFrame containing languages and movie counts.
- Output: None. Saves the pie chart as a PNG file.

main()
- Purpose: Main function to generate visualizations based on the updated database.
- Input: None.
- Output: None. Generates and saves visualizations to the output folder.

7. pokevisualizations.py
create_visualizations(type_counts)
- Purpose: Generates visualizations for Pokémon type distributions and average attack values.
- Input: A list or DataFrame of Pokémon type counts.
- Output: None. Saves bar and pie charts as PNG files.

8. Documents Used —----------

Date	Issue	Location of Resource	Result
12/03	Understanding how to handle missing data (e.g., revenue or language fields) from the TMDb API.	ChatGPT, TMDb API Docs	ChatGPT suggested using conditional logic to handle missing fields, which resolved the issue.
12/06	Generating error-handling logic for failed API requests (e.g., 401/404 status codes).	ChatGPT, Python Requests Documentation	Implemented error-handling using a retry mechanism and meaningful error messages.
12/10	Figuring out how to calculate average movie revenue for each genre using SQL aggregation functions.	ChatGPT, SQLite Documentation	Used AVG(revenue) with GROUP BY genre_id to calculate correct average values for each genre.
12/11	Fetching paginated Pokémon data in batches of 25 while preventing duplicate entries.	ChatGPT, PokéAPI Documentation, SQLite	Used limit/offset for pagination and `INSERT OR IGNORE` to skip duplicates effectively.
12/14	Extracting specific Pokémon stats (attack, defense, speed) from nested JSON API responses.	ChatGPT, Python JSON Documentation	Successfully accessed nested JSON keys to retrieve Pokémon stats and ensured robustness.
12/15	Creating visually appealing bar and pie charts with clear labels, colormaps, and uncluttered visuals.	ChatGPT, Matplotlib Documentation	Used filtered top data, added gridlines, and selected colormaps like `viridis` for better visuals.