Final Report

Evie Katmanivong & Lyla Loomis

REPO LINK:
https://github.com/lloomiss/SI206-final-project

Goals

Original

Our goal was to compare the difference in ratings between anime enthusiasts and as opposed to a more general audience. We decided to do this by comparing ratings across MyAnimeList (representing anime enthusiasts) and Rotten Tomatoes (representing a more general audience). We were also curious to see if critic reviews influenced the ratings of normal audience members.

Planned Resources

MyAnimeList Official API
RottenTomatoes Website (no specified page)

Planned Extracted Data

From MyAnimeList's Official API

For each anime:

- Score (Float/10)
- Number of seasons
- Title
- Genre(s)
- Studio
- Number of episodes
- Runtime or release date
- Number of reviews

From Rotten Tomatoes Website:

For each anime:

- Title
- Avg. Tomatometer (percentage)
- Avg. Popcornmeter (percentage)

Achieved Goals

We were able to study the relationship between rating averages by genre and platform as well as for the current top 5 anime trending right now. We calculated averages and sorted them based on the criteria above. Furthermore, we were able to visualize the difference in ratings between my anime list, the tomatometer, and the popcornmeter for the top most reviewed anime and the least reviewed anime

Resources Used

Jikan (MyAnimeList's unofficial API)
RottenTomatoes Website (TV Shows with filter anime, top to bottom rated tomatometer)

Extracted Data

From Jikan API

For each anime:

- Score (Float/10)
- Title
- Genre(s)
- Number of episodes
- Runtime or release date
- Number of reviews

From Rotten Tomatoes Website:

For each anime:

- Title
- Avg. Tomatometer (percentage)
- Avg. Popcornmeter (percentage)

Problems

MAL API vs. Jikan API (11/20/2024) – resolved

At the beginning of our final project, we decided that we were going to use MyAnimeList's (MAL) official API as well as RottenTomatoes' official website to draw our comparisons. During office hours on 11/20, we realized that gaining access to the official MAL API would be more difficult than previously thought. We were required to state our purpose for access, provide a link to our app/website, and provide specifications on our uses. Given that we are not developing an application/website with this data, we did not have a link to provide. Unfortunately, we could not bypass providing a link to our "app/website", so we decided to pivot to using the unofficial MAL API known as "Jikan".

Retrieving Data with Title as Pathway Parameter (11/25/2024) – resolved

Getting Full Data

We noticed that Rotten Tomatoes was limited in its number of anime to choose from. Due to this, we decided that it would be best for us to pull information from Rotten Tomatoes, and then use a common characteristic to get more information on a specific anime through Rotten Tomatoes. An issue that we ran into using the titles to gather information from the Jikan API was that using the call ‘'https://api.jikan.moe/v4/anime?q={RT_title}' would result in the retrieval of information equivalent to a search query on the MyAnimeList for the specified title.

Initially, we decided that it would be best to use the first result in the search query to fill out the information on the desired anime, however, we quickly realized that some of the animes from RottenTomatoes were not documented on MyAnimeList. Therefore, if we were to use the first search query, it was possible that we would be retrieving information for the wrong anime. To combat this, we decided to iterate through all of the animes that were in the search results and check if the titles matched.

Different Title Languages (11/26/2024) – resolved

The default title on Rotten Tomatoes had instances where the title was in a different language than the default title on MyAnimeList. Despite the difference between default titles, we were able to extract the Japanese equivalent of the default title for MyAnimeList and used that to confirm that the two data entries were what we were looking for.

Different Scoring Systems – (11/26/2024) — resolved

We ran into an issue where the scoring systems were different with Jikan's API scoring system being out of 10, and Rotten Tomatoes had a rating system out of 100. To combat this, we standardized the score on a scale of 10 – effectively translating the scoring values from Rotten Tomatoes to the same scoring format as Jikan API.

Calculation File

```python
import json
import re
import unittest
import sqlite3
import json
import os
import time
import matplotlib.pyplot as plt
import numpy as np
import csv


def set_up_database(db_name):
    path = os.path.dirname(os.path.abspath(__file__))
    conn = sqlite3.connect(path + "/" + db_name)
    cur = conn.cursor()
    return cur, conn


def combined_select_n_CSV(conn, cur):
    query = 'SELECT * FROM combined_anime_data'
    cur.execute(query)
    rows = cur.fetchall()
    column_names = [description[0] for description in cur.description]

    csv_file = 'combined_anime_data_output.csv'  # Output CSV file name
    with open(csv_file, 'w', newline='') as file:
        writer = csv.writer(file)
        # Optional: Write the column headers
        writer.writerow(column_names)
        # Write the data rows
        writer.writerows(rows)
    conn.close()

    print(f"Data has been written to {csv_file}")
    return csv_file
```

```python
def avg_rating_by_genre(csv_file):
    # genres with more than 1000 entries on myanimelist
    genres = ("Drama", "Action", "Adventure", "Sci-Fi",  "Fantasy",
"Comedy", "Sports", "Supernatural", "Mystery", "Romance", "Slice of Life")
    genre_ids = (2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13)

    genre_mapping = dict(zip(genres, genre_ids))

    genre_data = {genre_id: {'MAL_total': 0, 'RT_total': 0, 'count': 0}
for genre_id in genre_ids}

    with open(csv_file, newline='', encoding='latin1') as csvfile:  # Use
'latin1' encoding here
        reader = csv.reader(csvfile)
        header = next(reader)  # Skip the header

        for row in reader:
            genre_name = row[3]  # Assuming genre name is in the 4th
column
            genre_id = genre_mapping.get(genre_name)
            if genre_id in genre_data:
                try:
                    mal_score = float(row[2]) if row[2] else None
                except ValueError:  # In case it cannot convert to float
                    mal_score = None

                try:
                    rt_popcorn = float(row[9]) if row[9] else None
                except ValueError:  # In case it cannot convert to float
                    rt_popcorn = None

                if mal_score is not None:
                    genre_data[genre_id]['MAL_total'] += mal_score
                if rt_popcorn is not None:
                    genre_data[genre_id]['RT_total'] += rt_popcorn
                genre_data[genre_id]['count'] += 1

    # Calculate averages
    for genre_id in genre_ids:
        if genre_data[genre_id]['count'] > 0:
            genre_data[genre_id]['MAL_avg'] =
genre_data[genre_id]['MAL_total'] / genre_data[genre_id]['count']
            genre_data[genre_id]['RT_avg'] =
genre_data[genre_id]['RT_total'] / genre_data[genre_id]['count']
        else:
            genre_data[genre_id]['MAL_avg'] = 0
            genre_data[genre_id]['RT_avg'] = 0

    score_means = {
        'MyAnimeList Score': [genre_data[genre_id]['MAL_avg'] for genre_id
in genre_ids],
        'Rotten Tomatoes Popcornmeter': [genre_data[genre_id]['RT_avg']
for genre_id in genre_ids]
    }

    x = np.arange(len(genres))  # the label locations
    width = 0.35  # the width of the bars
    multiplier = 0

    fig, ax = plt.subplots(layout='constrained')

    colors = {'MyAnimeList Score': '#2e51a2', 'Rotten Tomatoes
Popcornmeter': '#fa320a'}

    for attribute, measurement in score_means.items():
        offset = width * multiplier
        rects = ax.bar(x + offset, measurement, width, label=attribute,
color=colors[attribute])
        ax.bar_label(rects, padding=3)
        multiplier += 1

    # Add some text for labels, title and custom x-axis tick labels, etc.
    ax.set_ylabel('Average Rating (out of 10)')
    ax.set_title('Average Rating by Platform and Genre')
    ax.set_xticks(x + width / 2, genres)
    ax.legend(loc='upper left', ncols=2)
    ax.set_ylim(0, 11)  # Since scores are out of 10

    plt.show()
```

```
def top_5_results():
    anime = ("Uzumaki", "Pluto", "Solo Leveling", "Delicious in Dungeon",
"Frieren: Beyond Journey's End")
    scores = {
        'Score': (5.89, 8.46, 8.27, 8.61, 9.32),
        'Tomatometer': (8, 10, 10, 10, 10),
        'Popcornmeter': (7, 9.5, 8.4, 9.6, 9.5)
    }

    x = np.arange(len(anime))  # the label locations
    width = 0.15  # the width of the bars
    multiplier = 0

    fig, ax = plt.subplots(layout='constrained')

    colors = {'Score': '#2e51a2', 'Tomatometer': '#fa320a',
'Popcornmeter': '#f9d320'}

    for attribute, measurement in scores.items():
        offset = width * multiplier
        rects = ax.bar(x + offset, measurement, width, label=attribute,
color=colors[attribute])
        ax.bar_label(rects, padding=3)
        multiplier += 1

    # Add some text for labels, title and custom x-axis tick labels, etc.
    ax.set_ylabel('Rating (out of 10)')
    ax.set_title('Current Top 5 Anime Ratings')
    ax.set_xticks(x + width / 2, anime)
    ax.legend(loc='upper left', ncols=2)
    ax.set_ylim(0, 11)  # Since scores are out of 10

    plt.show()

def top_5_most_reviewed_anime(csv_file):
    anime_data = []

    # Step 1: Read the CSV file and extract the relevant information
    try:
        with open(csv_file, newline='', encoding='latin1') as csvfile:  # 
Try 'utf-8' if 'latin1' does not work
            reader = csv.reader(csvfile)
            header = next(reader)  # Skip the header
            print(f"CSV Header: {header}")

            for row in reader:
                # Print each row for debugging
                print(f"CSV Row: {row}")

                try:
                    anime_id = row[0]
                    score = float(row[1]) if row[1] else None
                    genre_id = int(row[2]) if row[2] else None
                    num_episodes = int(row[3]) if row[3] else None
                    release_date = row[4]
                    num_reviews = int(row[5]) if row[5] else 0
                    tomatometer = float(row[6]) if row[6] else None
                    popcornmeter = float(row[7]) if row[7] else None

                    anime_data.append({
                        'anime_id': anime_id,
                        'score': score,
                        'genre_id': genre_id,
                        'num_episodes': num_episodes,
                        'release_date': release_date,
                        'num_reviews': num_reviews,
                        'tomatometer': tomatometer,
                        'popcornmeter': popcornmeter
                    })
                except ValueError as e:
                    print(f"ValueError: {e} in row {row}")
                    continue  # In case of conversion errors, skip the row

        # Debug print to verify data is being read correctly
        print(f"Total anime entries read: {len(anime_data)}")

        if not anime_data:
            print("No anime data available.")
            return
        # Step 2: Sort the anime data by num_reviews in descending order
        anime_data_sorted = sorted(anime_data, key=lambda x:
x['num_reviews'], reverse=True)

        # Step 3: Select the top 5 most reviewed anime
        top_5_anime = anime_data_sorted[:5]

        # Debug print to verify top 5 data
        for idx, anime in enumerate(top_5_anime):
            print(f"{idx + 1}: {anime['anime_id']} with
{anime['num_reviews']} reviews")

        # Step 4: Create the bar graph with the extracted data
        anime_titles = [anime['anime_id'] for anime in top_5_anime]
        scores = {
            'MAL score': [anime['score'] for anime in top_5_anime],
            'Tomatometer': [anime['tomatometer'] for anime in
top_5_anime],
            'Popcornmeter': [anime['popcornmeter'] for anime in
top_5_anime]
        }

        x = np.arange(len(anime_titles))  # the label locations
        width = 0.15  # the width of the bars
        multiplier = 0

        fig, ax = plt.subplots(layout='constrained')

        colors = {'MAL score': '#2e51a2', 'Tomatometer': '#fa320a',
'Popcornmeter': '#f9d320'}

        for attribute, measurement in scores.items():
            offset = width * multiplier
            rects = ax.bar(x + offset, measurement, width,
label=attribute, color=colors[attribute])
            ax.bar_label(rects, padding=3)
            multiplier += 1
        # Add some text for labels, title and custom x-axis tick labels,
etc.
        ax.set_ylabel('Rating (out of 10)')
        ax.set_title('Top 5 Most Reviewed Anime Ratings')
        ax.set_xticks(x + width / 2, anime_titles)
        ax.legend(loc='upper left', ncols=2)
        ax.set_ylim(0, 11)  # Since scores are out of 10

        plt.show()

    except FileNotFoundError as e:
        print(f"FileNotFoundError: {e}")
    except Exception as e:
        print(f"An error occurred: {e}")

def main():
    try:
        #cur, conn = set_up_database('anime.db')
        #csv_file = combined_select_n_CSV(conn, cur)
        #avg_rating_by_genre('combined_anime_data_output.csv')
        #top_5_results()
        top_5_most_reviewed_anime('combined_anime_data_output.csv')
    except Exception as e:
        print(e)


if __name__ == "__main__":
    main()
```

Visualizations

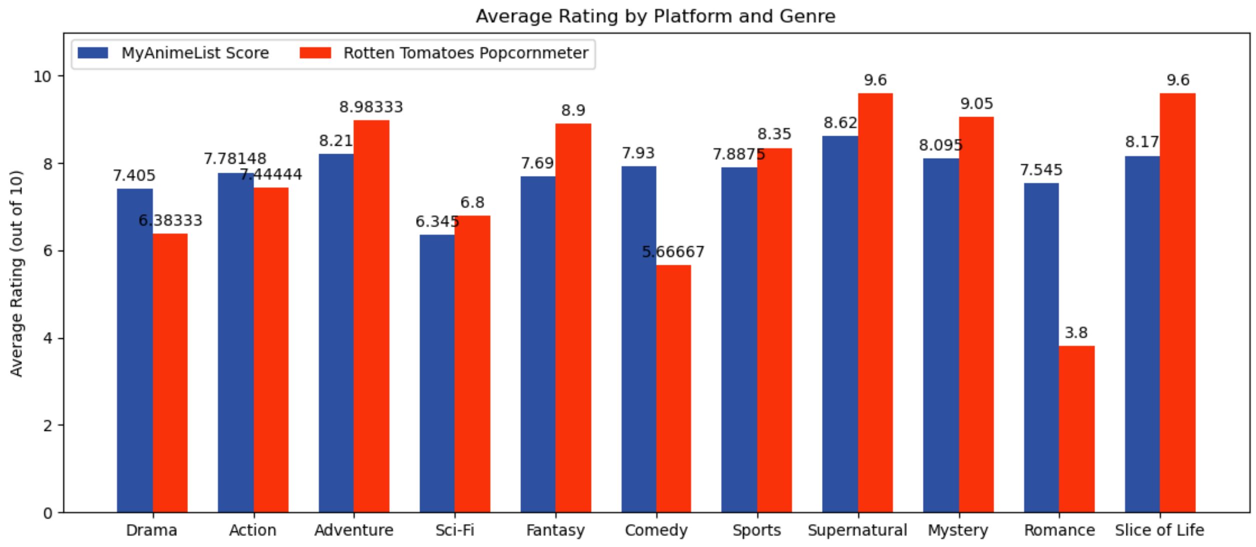


Figure 1

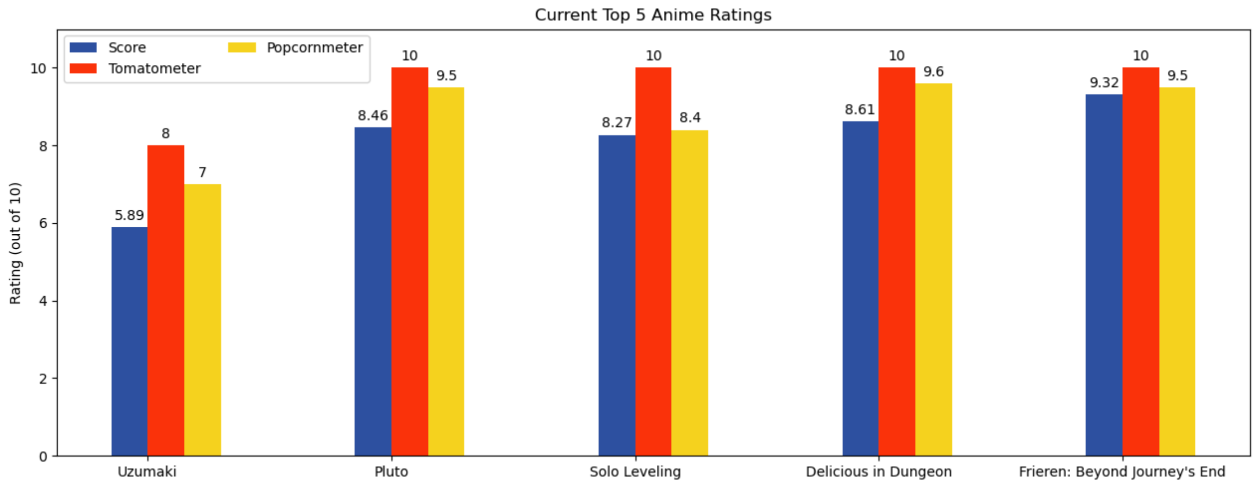
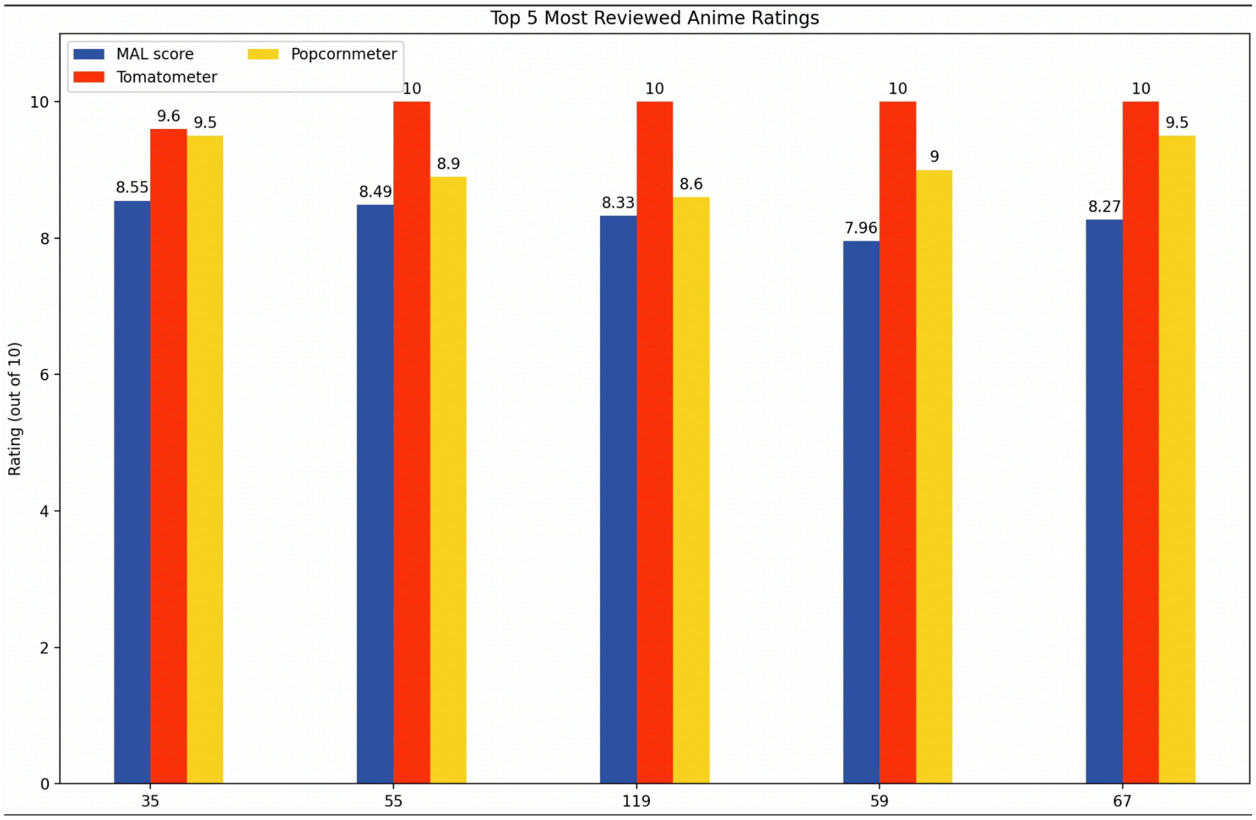


Figure 2


Top 5 Most Reviewed Anime Ratings
MAL score
Tomatometer
Popcornmeter
Rating (out of 10)
0
2
4
6
8
10
8.55
9.6
9.5
8.49
10
8.9
8.33
10
8.6
7.96
10
9
8.27
10
9.5
35
55
119
59
67

Instructions for Running Code

1. Download all files in the zip file
2. Run SI_206_FinalProject.py (if you have anime.db from the zip file, no need to run)
   a. This will establish the database with 6 tables.
   b. If you don't have the anime.db file, the program will have to be run multiple times to get the full db structure, as it only uploads 25 lines at a time.
3. Run calculations.py
   a. This will set up a connection to the database and allow you to calculate things like average scores based off CSV data
   b. Comment out graphs that you don't want to display and comment out the csv builder functions while you're making the graphs.

Code Documentation

SI_206_FinalProject.py

tomato_extract(tag) → tuple

This is a helper function for get_RT_info that returns a tuple with information from a given tag in HTML. In the tuple, it will return the anime's title, tomatometer, and popcornmeter. For those missing a title or meter, the default value will be "NULL."

Input:

- Type: bs4.element.Tag
- Description: This is an HTML Tag object from the BeautifulSoup library that represents an item on Rotten Tomatoes. The tag is expected to include nested tags for the title, tomatometer score, and popcornmeter score of the item.

output:

- If valid data is extracted, it returns a tuple: (lower_title, tomatometer, popcornmeter).
    - lower_title (str): The title of the item in lowercase. Returns "NULL" if not found.
    - tomatometer (float or str): The tomatometer score scaled between 0 and 10. Returns "NULL" if not found or not a percentage.
    - popcornmeter (float or str): The popcornmeter score scaled between 0 and 10. Returns "NULL" if not found or not a percentage.
- If all values are "NULL", it returns None.

get_RT_info(soup, start, stop) → list

Input:

- Parameter soup:
    - Type: bs4.BeautifulSoup
    - Description: A BeautifulSoup object representing the parsed HTML content from which data will be extracted.
- Parameter start:
    - Type: int
    - Description: An integer specifying the starting index from which to begin extracting data tags.

- Parameter stop:
    - Type: int
    - Description: An integer specifying the stopping index up to which data tags will be extracted (not inclusive).

Output:

- Type: list
- Description: A list of tuples, where each tuple contains:
    - Title (str): The title of the Rotten Tomatoes item, converted to lowercase.
    - Tomatometer Score (float or str): The tomatometer score scaled between 0 and 10. "NULL" if not available.
    - Popcornmeter Score (float or str): The popcornmeter score scaled between 0 and 10. "NULL" if not available.
- Each tuple in the list represents an individual item extracted from Rotten Tomatoes within the specified range.

Purpose:

The function get_RT_info extracts information about items from Rotten Tomatoes using the parsed HTML content. It collects data on items within a specified range and returns a list of extracted information.

get_MAL_info(RT_data_list) → list

Input:

- Parameter RT_data_list:
    - Type: list
    - Description: A list of tuples, where each tuple contains:
        - Title (str): The title of the anime from Rotten Tomatoes, converted to lowercase.
        - Tomatometer Score (float or str): The tomatometer score from RT, scaled between 0 and 10.
        - Popcornmeter Score (float or str): The popcornmeter score from RT, scaled between 0 and 10.
    - The tuples in the list represent individual anime items extracted from Rotten Tomatoes.

Output

- Return Value:
    - Type: list
    - Description: A list of tuples, where each tuple contains:
        - Title (str): The title of the anime (from RT).
        - Season Score (float or str): The MAL score of the anime. "null" if not available.
        - Genre (str): The primary genre of the anime. "null" if not available.
        - Number of Episodes (int or str): The number of episodes of the anime. "null" if not available.
        - Release Date (str): The release date of the anime. "null" if not available.
        - Number of Reviews (int or str): The number of reviews/scored_by on MAL. "null" if not available.
    - Each tuple in the list represents an individual anime item with detailed information from MyAnimeList.

set_up_database(db_file) → cur, conn

Input:

The input for this function should be the name of the db file.

Output:

This function sets up the connection and the handle for the database (anime.db)

Purpose:

This function creates and sets up the cursor & connection for the database we’ll be using for the rest of the project.

create_tables (cur, conn) → None

Input:

This function takes in a database cursor and connection

Output:

This function does not return any value

Purpose:

This function creates the Rotten Tomatoes table in the database titled Anime, implementing 3 columns, the Anime ID, Tomatometer, and the Popcornmeter. For animes in the RT_data_list, the function inserts the corresponding values into the table.

This function also creates a table named ‘Genres’ to track the anime genres, with each genre having a corresponding ID

This function also creates the MyAnimeList Table (named MAL), storing an anime_id, title, score, genre_id, number of episodes, release date, number of reviews

This function also creates a table named ‘progress_tracker’, this table keeps track of how many rows have been processed. This table plays a pivotal role in ensuring that only 25 rows of data are inserted into our respective tables at a time.

get_rows_processed(cur) → int:

Input:

The input for this function is the cursor (handler) of a database file (intended to be anime.db)

Output:

This function returns the number of rows that have been processed from the table that keeps track of how many rows have been uploaded.

Purpose:

The purpose of this function is to provide a tracking number to indicate how many rows have been processed. It pulls directly from the function’s respective table in the db file so the program knows how far down the list it is. That way, the program won’t keep trying to upload data from the very beginning, which could lead to duplicate rows.

update_rows_processed(cur, conn, rows_processed) → None

Input:

The input for this function is the cursor and connection for a database (intended to be from anime.db) and the number of rows that have been processed (from the function get_rows_processed)

Output:

This function does not output any data. Rather, it keeps track of how many rows have been uploaded to the database in its own table.

Purpose:

The purpose of this function is to change the progress tracking table to accurately reflect how many rows have been processed in our database.

upload_batch(RT_info, MAL_info, cur, conn) → int

Input:

The input for this function is a list (RT_info, intended to be list returned by from get_RT_info), another list (MAL_info, intended to be list returned by get_MAL_info), and the cursor and connection to a database (intended to be anime.db)

Output:

No output. This function commits changes to the db file after uploading rows of a certain batch size. Ours is limited to 25 at a time, so each time you run the program, 25 (or fewer) rows should be added to the db file's tables.

Purpose:

The purpose of this function is to insert data into tables RT_meters and MAL and return the number of rows that have been uploaded to the database. All of the commits to the database in the current iteration of running the program are made here. By returning the number of rows uploaded, the program can keep track of where it is in the list of items to upload, avoiding duplicate or skipped data.

set_up_combined_tables(cur, conn) → None

Input:

The expected input in the cursor and the connection to a database (intended to be anime.db)

Output:

This function does not have an output

Purpose:

The purpose of this function is to create a combined table, using JOIN to merge the data from database tables MAL and RT_meters by using their anime_ID

main():

Gets our functions to work together.

calculations.py

set_up_database(db_name):

Input:

name of a database file (intended to be anime.db)

Output:

cursor(handle) and connection to the database

Purpose:

This function establishes a connection to the database (intended to be anime.db), and creates a handler object

combine_select_n_CSV(conn, cur):

Input:

connection and cursor (returned from set_up_database)

Output:

Nothing

Purpose:

This function fetches all the data from the table combined_anime_data and writes it into a csv.

avg_rating_by_genre(csv_file):

Input:

csv_file

Output:

Plotted bar graph

top_5_results():

Input:

None, calculations are handled manually within the function

Output:

Plotted bar graph

Def main():

Pieces function together

Documentation of Resources Used

U-M GPT

- Used for debugging

get_MAL_info

- Had difficulties with incrementing through the JSON
    - “Can you help me access the English title equivalent”
    -

create_combined_table

- “Why are we getting a UNIQUE error”
- “How can we combine on a single column of values?”

Jikan REST API v4 Documentation

- Information on request limits
- JSON formatting notes
- API-specific functions

W3 Schools

- Teaching ourselves autoincrement