

Anna Kerhoulas , Willow Traylor , Claire Fuller

Si 201

Barbara Ericson

12/11/25

Link to repository: https://github.com/witaylor/SI201_FinalProject_RCCG.git

1. The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (5 points)

Our goal for this project was to collect data from 3 different APIs: Spotify, TMDB, and TVMaze. From each of these APIs, we planned to gather data about songs/movies/shows, their genres, popularity, and release date. For songs, we also planned to collect data about the song's artist, and for movies and shows, we planned to collect ratings data. With this data, we planned to calculate the average popularity of a form of media per month, compare the popularity of different media types, and search for a correlation between song popularity and movie/show popularity.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (5 points)

We used the Spotify, TMDB, and TVMaze APIs to gather the information we planned to about each different media type. With that data we decided to compare the most popular genre in regards to each media type. We also calculated average Spotify popularity by genre and the number of TMDB movies for different genres.

3. The problems that you faced (5 points)

Git as a collaborative platform is quite hard to get used to, we had a hard time understanding how to edit at the same time. The Spotify API was also difficult to work with and (Claire) had a hard time working with the correct query while still having enough data to work with. It was also difficult to separate duplicate string data using an ID from other tables.

Another issue I (Willow) faced was collecting shows with recent premiere dates from the TVMaze API. I tried looking through multiple pages of the API to find more recently premiered shows, but this was slow and still resulted in older shows being collected. Because of this, we shifted our goals and decided not to calculate popularity by month. Instead, we calculated and compared the most popular genres across the three media types.

4. The calculations from the data in the database (i.e. a screen shot) (5 points)

```
#Find most popular song genre
def most_popular_song_genre(conn):
    cur = conn.cursor()

    cur.execute("""
        SELECT g.genre_name, s.popularity
        FROM SpotifySongs s
        JOIN SongsToGenre sg ON s.id = sg.song_id
        JOIN SpotifyGenres g ON sg.genre_id = g.id
        WHERE s.popularity IS NOT NULL;
    """)

    genre_sums = {}
    genre_counts = {}

    for genre, pop in cur.fetchall():
        if genre not in genre_sums:
            genre_sums[genre] = 0
            genre_counts[genre] = 0
        genre_sums[genre] += pop
        genre_counts[genre] += 1

    best_genre = None
    best_avg = -1

    for genre in genre_sums:
        avg = genre_sums[genre] / genre_counts[genre]
        if avg > best_avg:
            best_avg = avg
            best_genre = genre

    return best_genre, best_avg
```

```
def calculate_tmdb_genre_counts(conn, output_file="tmdb_genre_counts.txt"):
    """
    Read Movies.genre_ids, convert to names, count occurrences.
    """
    cur = conn.cursor()
    cur.execute("SELECT genre_ids FROM Movies")
    rows = cur.fetchall()

    counts = Counter()

    for (genre_ids_str,) in rows:
        if not genre_ids_str:
            continue
        try:
            id_list = json.loads(genre_ids_str)
        except json.JSONDecodeError:
            id_list = []
        names = get_genre_names(id_list)
        for g in names:
            counts[g] += 1

    sorted_genres = sorted(counts.items(), key=lambda x: x[1], reverse=True)

    with open(output_file, "w") as f:
        for genre, count in sorted_genres:
            f.write(f"{genre}: {count}\n")

    print(f"Wrote TMDB genre counts to {output_file}")
```

1	dream pop: 91.00
2	j-rock: 88.00
3	pop soul: 87.33
4	art pop: 87.00
5	country rock: 86.00
6	pop: 85.50
7	techengue: 85.00
8	new wave: 85.00
9	neo soul: 85.00
10	madchester: 85.00
11	latin house: 85.00
12	jangle pop: 85.00
13	j-pop: 85.00
14	funk rock: 85.00
15	anime: 85.00
16	edm: 84.50
17	garage rock: 84.33
18	trap soul: 84.00
19	sertanejo universitário: 84.00
20	sertanejo: 84.00
21	rap rock: 84.00
22	quiet storm: 84.00
23	northern soul: 84.00
24	metal: 84.00

1	Unknown: 723
2	Drama: 137
3	Comedy: 124
4	Family: 70
5	Animation: 66
6	Crime: 61
7	Mystery: 23
8	Documentary: 5
9	Western: 2
10	

```

#Find most popular song genre
def most_popular_song_genre(conn):
    cur = conn.cursor()

    cur.execute("""
        SELECT Genres.genre, Songs.popularity
        FROM Genres
        JOIN Songs ON Genres.song_id = Songs.id
        WHERE Songs.popularity IS NOT NULL;
    """)

    genre_sums = {}
    genre_counts = {}

    for genre, pop in cur.fetchall():
        if genre not in genre_sums:
            genre_sums[genre] = 0
            genre_counts[genre] = 0
        genre_sums[genre] += pop
        genre_counts[genre] += 1

    best_genre = None
    best_avg = -1

    for genre in genre_sums:
        avg = genre_sums[genre] / genre_counts[genre]
        if avg > best_avg:
            best_avg = avg
            best_genre = genre

    return best_genre, best_avg

#Find most popular movie genre
def most_popular_movie_genre(conn):
    cur = conn.cursor()

    cur.execute("""
        SELECT popularity, genres
        FROM Movies
        WHERE popularity IS NOT NULL AND genres IS NOT NULL;
    """)

    genre_sums = {}
    genre_counts = {}

    for pop, genre_str in cur.fetchall():
        try:
            pop = float(pop)
        except:
            continue

        #split comma separated lists
        genres = [g.strip() for g in genre_str.split(",") if g.strip()]

        for genre in genres:
            if genre not in genre_sums:
                genre_sums[genre] = 0
                genre_counts[genre] = 0
            genre_sums[genre] += pop
            genre_counts[genre] += 1

    best_genre = None
    best_avg = -1

    for genre in genre_sums:
        avg = genre_sums[genre] / genre_counts[genre]
        if avg > best_avg:
            best_avg = avg
            best_genre = genre

    return best_genre, best_avg

#Find most popular show genre
def most_popular_show_genre(conn):
    cur = conn.cursor()

    cur.execute("""
        SELECT weight, genres
        FROM Shows
        WHERE weight IS NOT NULL AND genres IS NOT NULL;
    """)

    genre_sums = {}
    genre_counts = {}

    for pop, genre_str in cur.fetchall():
        try:
            pop = float(pop)
        except:
            continue

        #Separate comma separated lists
        genres = [g.strip() for g in genre_str.split(",") if g.strip()]

        for genre in genres:
            if genre not in genre_sums:
                genre_sums[genre] = 0
                genre_counts[genre] = 0
            genre_sums[genre] += pop
            genre_counts[genre] += 1

    best_genre = None
    best_avg = -1

    for genre in genre_sums:
        avg = genre_sums[genre] / genre_counts[genre]
        if avg > best_avg:
            best_avg = avg
            best_genre = genre

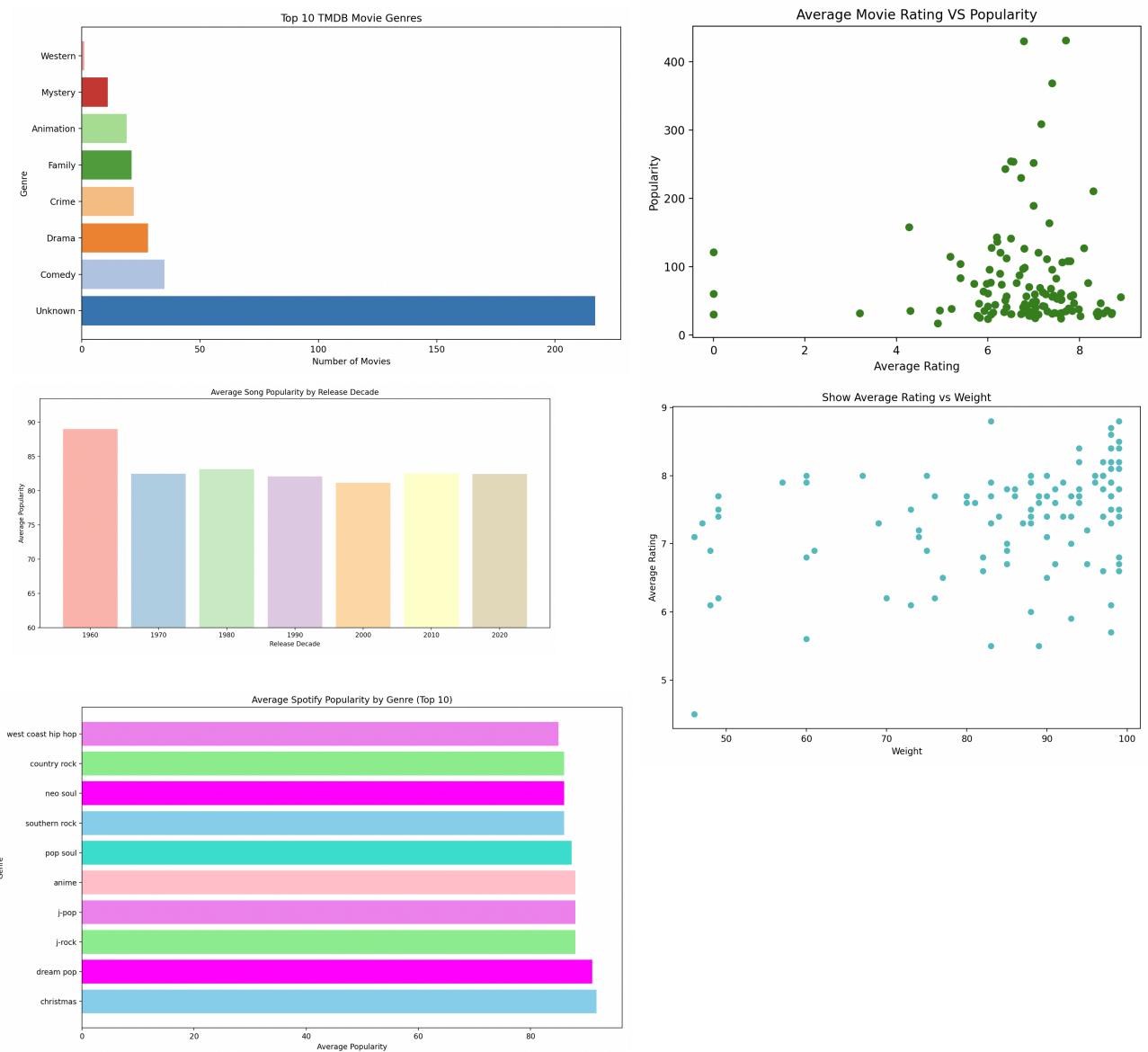
    return best_genre, best_avg

def find_most_popular_genres(conn):
    return {
        "songs": most_popular_song_genre(conn),
        "movies": most_popular_movie_genre(conn),
        "shows": most_popular_show_genre(conn)
    }

1 media_type,genre,avg_popularity
2 Song,dream pop,91.0
3 Movie,Western,69.8215
4 Show,Music,97.0
5

```

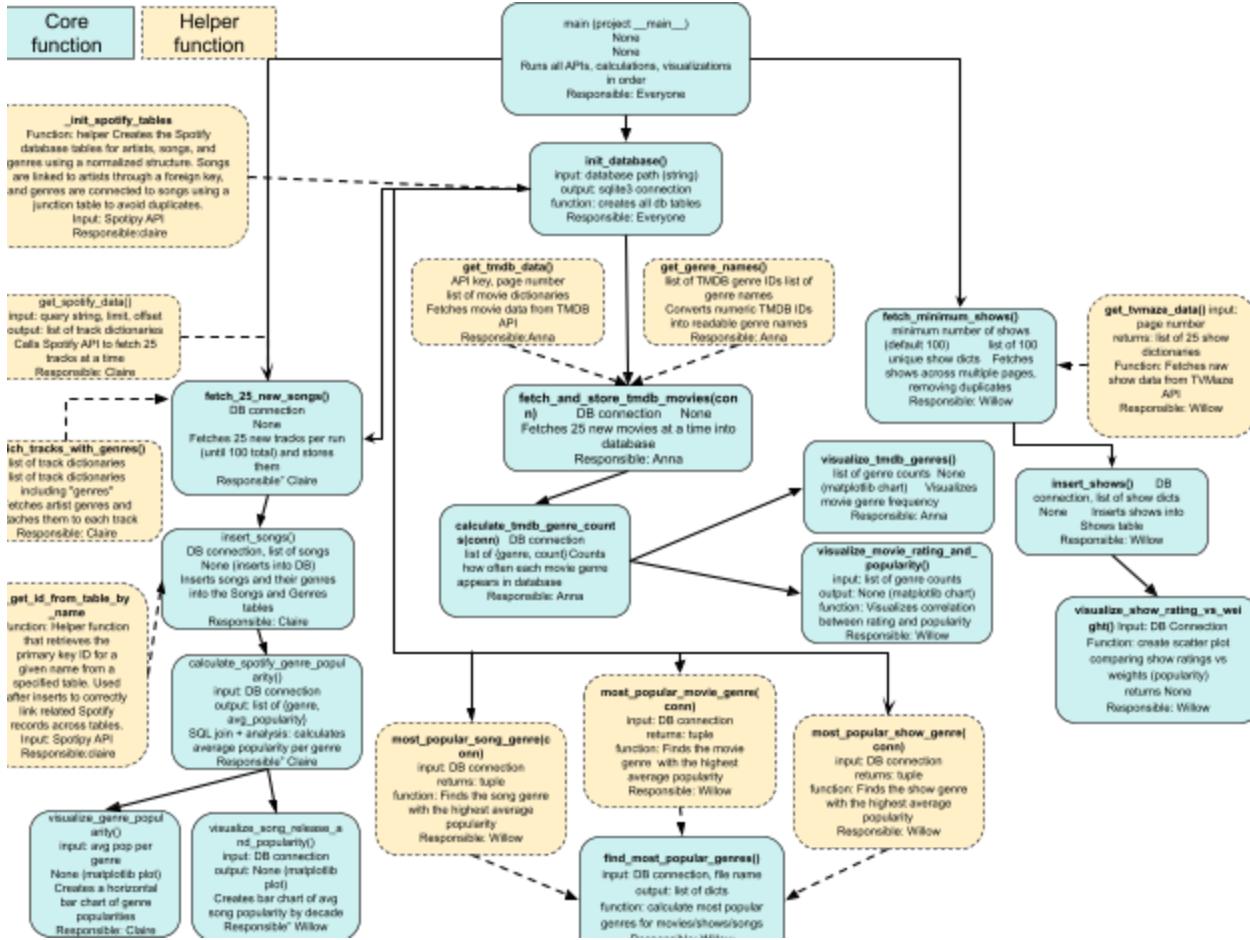
5. The visualizations that you created (i.e. screen shot or image file) (5 points)



6. Instructions for running your code (5 points)

- Run the code. 25 rows should be added to the SpotifySongs, Movies, and Shows tables.
 - View “Movie”, “Shows”, and “SpotifySongs”
 - Other Spotify Table sets aid “Spotify” Songs index with no duplicate string data
- Run the code 4 times in order to get all databases to 100 rows.

7. An updated function diagram with the names of each function, the input, and output and who was responsible for that function (10 points)



8. You must also clearly document all resources you used. The documentation should be of the following form (10 points)

Date	Issue Description	Location of Resource	Result
12/09/25	Understanding stashing changes with Github	https://git-scm.com/docs/git-stash	Successfully work on the file at the same time as other group members rather than one at a time
12/01/25	Understanding TVMaze API	https://www.tvmaze.com/api	Successfully used documentation to call API and use data
12/10/25	Getting started with spotify API	https://spotipy.readthedocs.io/en/2.22.1/	Successfully use information in spotipy API.

12/10/25	Inserting data into databases	https://www.sqlitetutorial.net/sqlite-python/insert/	Was able to successfully insert data into database
12/10/25	Using Sqlite is confusing	https://www.freecodecamp.org/news/work-with-sqlite-in-python-handbook/	Gathered more info on how to use SQLite
12/10/25	Difficulty analyzing spotipy API.	https://developer.spotify.com/documentation/web-api/reference/search	Helped understand query, limit, etc.
12/09/25	Creating charts with Matplotlib	https://matplotlib.org/stable/users/index.html	Successfully created customized visualizations

What we changed post-grading season:

- Changed our code so it added EXACTLY 25 songs, movies, and shows to the database so that we could include in the instructions to run the code 4 times in order to hit the 100 row minimum.

– Anna (TMDB)

- Changed release_date duplicate string data into release_year (an integer)
- Changed movie bar-graph to be colorful to avoid being accused of copying the in-class example

– Willow (TVMaze & extra credit visualizations)

- Changed fetch_minimum_shows function to fetch 25 shows instead of 100
- Added 2 extra credit visualizations

– Claire (Spotipy)

- Created multiple tables using integers as an index to prevent the use of duplicate string data
- Adjusted calculation and visualization based on that change in tablesets
- Fixed shared integer key