SI 206 - Final Project Report

Carolina Mondragon, Mia Goldstein, Dash Kwan

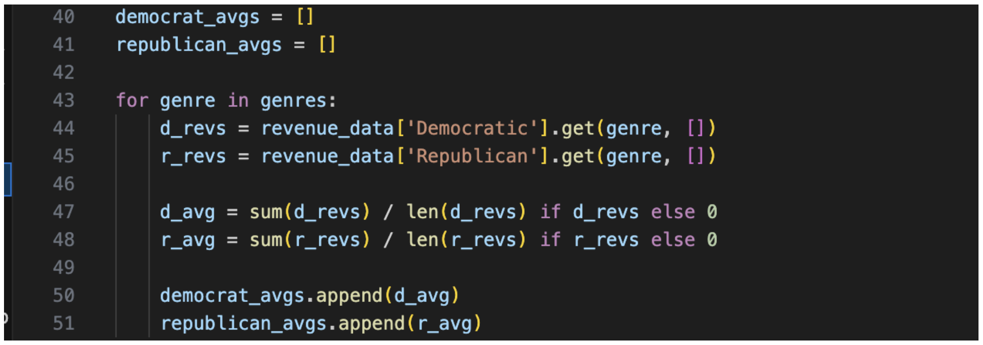
Link: https://github.com/cmondrag18/api-web-proj.git

Initial Goal: We sought to compare the highest grossing movies in specific genres before and after election years, planning to gather an election data API. However, after thorough research we could not find a proper election data API, so we changed our project approach. We had not discovered any election APIs at the time of developing the project idea, but planned to use TMDB as our movie API source.

Finalized Project Goals: After working with movie APIs in class (TMDB), we were interested in applying the box office data to gauge popularity and success of specific movies per year. We also used the Spotify API to track song popularity in each year and compare that data to the movies. Additionally, we tied them all together by using election data results online and fetching certain parts of the website to track election winners in the last 4 elections. We fused all of this information together and created an analysis on the impact of the election on pop culture popularity and reception.
Websites/APIs used: TMDB, Spotify API, Last.fm API, The American Presidency Project

Problems: We faced several problems throughout the project. The first problem came during the development of the project, when we could not find a proper election API. Either the API required a license, or some form of payment, so we had to adjust our project. It was a challenge to search for another API/element to analyze in the project, so we decided to apply our analysis to media in general and selected the Spotify API. Additionally, we had to accommodate with limited data, for example we could not obtain specific data that would label for us the highest grossing movies in a specific genre by year, so we adjusted accordingly. Finally, we had to handle some errors with our databases, ensuring that each table met the requirements and were properly formatted with sufficient data.

Calculations:



```python
democrat_avgs = []
republican_avgs = []

for genre in genres:
    d_revs = revenue_data['Democratic'].get(genre, [])
    r_revs = revenue_data['Republican'].get(genre, [])

    d_avg = sum(d_revs) / len(d_revs) if d_revs else 0
    r_avg = sum(r_revs) / len(r_revs) if r_revs else 0

    democrat_avgs.append(d_avg)
    republican_avgs.append(r_avg)
```

In the file mia_visualizations.py, we calculated the average revenue per genre during election years when either the Democratic or Republican party won. Specifically, we grouped revenue data from the box_office_movies table by genre and winning party using SQL joins between box_office_movies, movie_genre_id, and election_results. For each of the 11 genres, we computed the average revenue using a simple sum divided by count logic (sum(revenue) / number of entries), which was then visualized in a grouped bar chart comparing Democratic and Republican years.

<u>Text File Output</u> avg_revenue_by_genre_and_party:

```
Average Revenue by Genre and Winning Party (2012, 2016, 2020, 2024)

Sci-Fi:
  Democratic Wins: $942,059,810
  Republican Wins: $1,246,560,030

Romance:
  Democratic Wins: $438,869,034
  Republican Wins: $650,549,696

Drama:
  Democratic Wins: $771,231,423
  Republican Wins: $966,550,600

Comedy:
  Democratic Wins: $598,480,232
  Republican Wins: $1,363,731,908

Documentary:
  Democratic Wins: $21,889,354
  Republican Wins: $62,000,000
```

Action:
  Democratic Wins: $507,119,058
  Republican Wins: $0

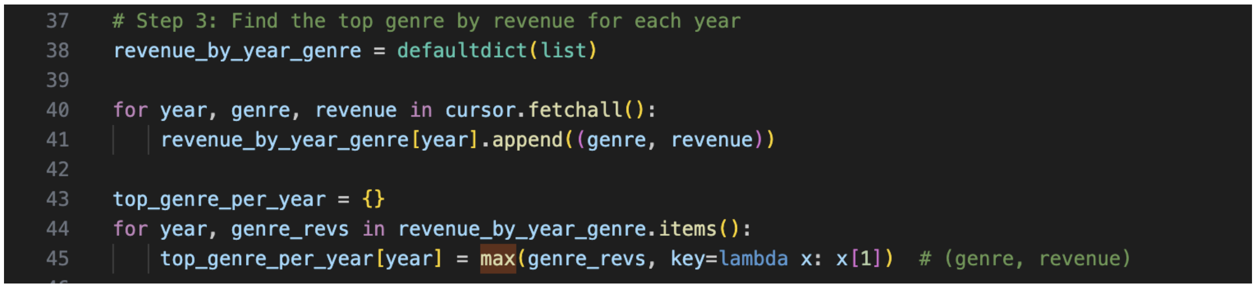
Animation:
  Democratic Wins: $0
  Republican Wins: $0

Adventure:
  Democratic Wins: $251,410,631
  Republican Wins: $0

Fantasy:
  Democratic Wins: $1,021,103,568
  Republican Wins: $0

Thriller:
  Democratic Wins: $1,108,594,176
  Republican Wins: $410,016,366

Mystery:
  Democratic Wins: $236,177,234
  Republican Wins: $263,877,148

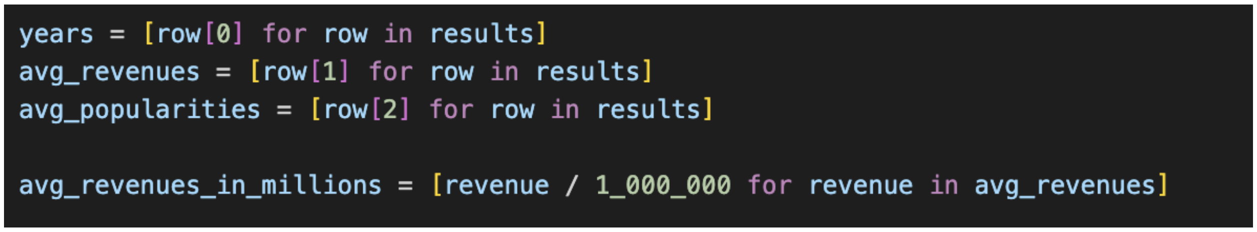


```python
# Step 3: Find the top genre by revenue for each year
revenue_by_year_genre = defaultdict(list)

for year, genre, revenue in cursor.fetchall():
    revenue_by_year_genre[year].append((genre, revenue))

top_genre_per_year = {}
for year, genre_revs in revenue_by_year_genre.items():
    top_genre_per_year[year] = max(genre_revs, key=lambda x: x[1])  # (genre, revenue)
```

In mia_visualizations2.py, we identified the top-grossing genre during each winning party's election year—specifically, 2012, 2016, 2020, and 2024. To determine this, we queried the maximum revenue (MAX(b.revenue)) for each genre in each election year, using a join between box_office_movies, movie_genre_id, and election_results and filtering by years where the party won (election_results.winner = 'Yes'). From the results, we selected the genre with the highest revenue per year, which was used to generate a color-coded bar chart by political party.

<u>Text File Output</u> top_genre_by_year:
Top-Grossing Genre by Election Year (2012, 2016, 2020, 2024)

2012 (Democratic win): Sci-Fi - $1,518,815,515
2016 (Republican win): Sci-Fi - $1,155,046,416
2020 (Democratic win): Action - $507,119,058
2024 (Republican win): Comedy - $1,698,863,816



```
years = [row[0] for row in results]
avg_revenues = [row[1] for row in results]
avg_popularities = [row[2] for row in results]

avg_revenues_in_millions = [revenue / 1_000_000 for revenue in avg_revenues]
```

From “dash_visualizations.py”, which calculates the average revenues and popularities of the total box office revenue and song popularity for each year.

<u>Text File Output:</u>
Average Movie Revenue Per Election Year:

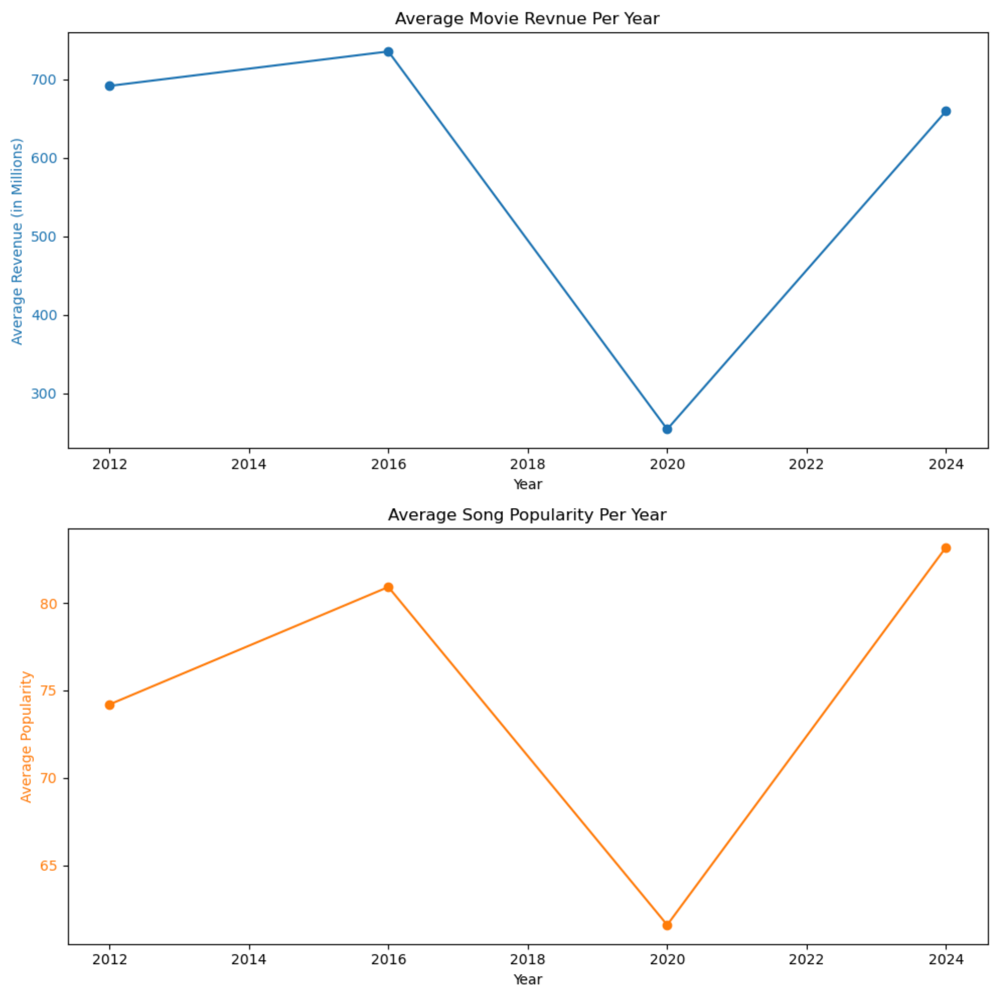
2012: $690.3 Million
2016: $735.6 Million
2020: $250.1 Million
2024: $657.4 Million

Average Song Popularity Per Election Year:

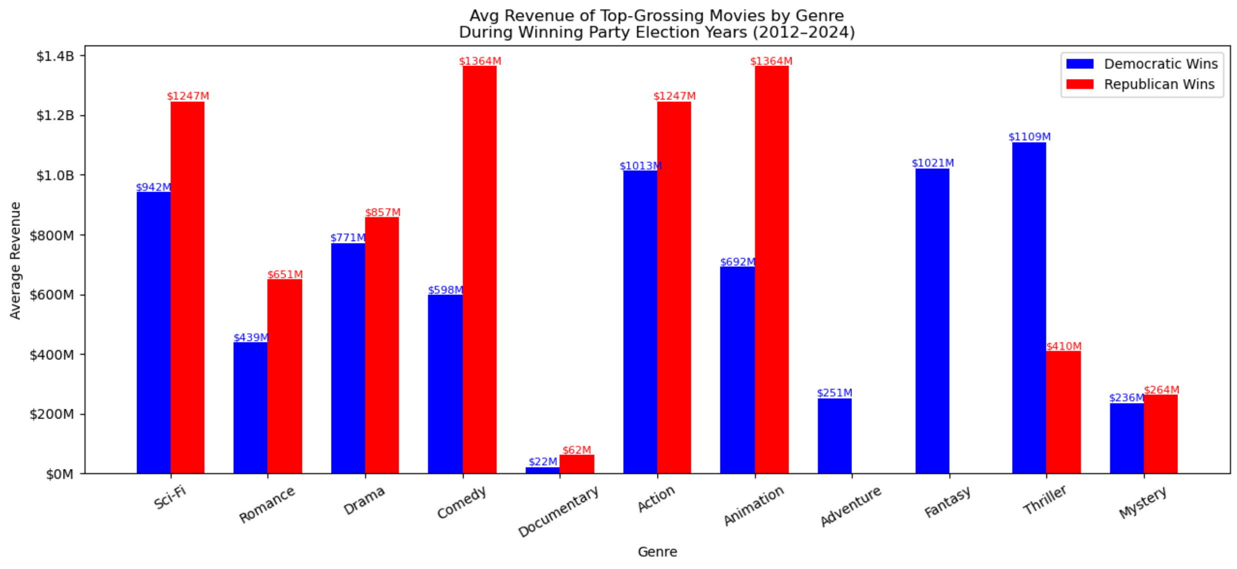
2012: 74.18
2016: 80.81

2020: 61.48
2024: 83.14

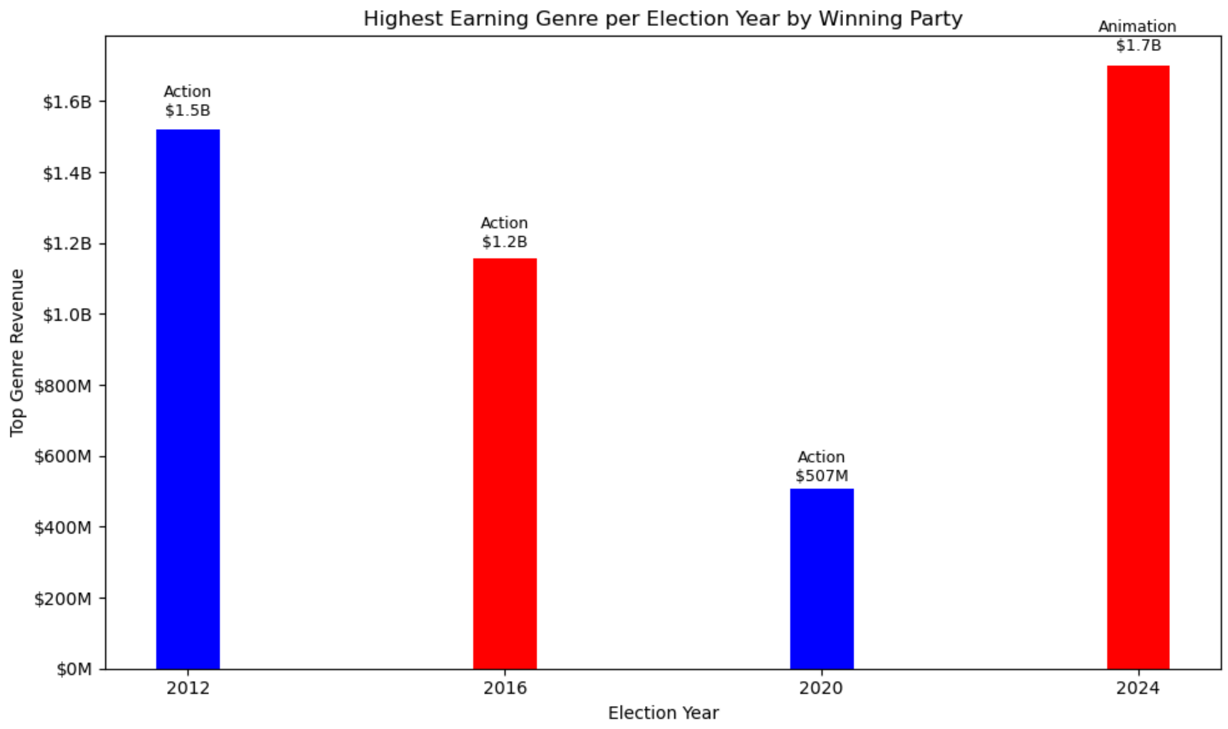
Visualizations:



Above image compares the average box office revenue per year versus the average popularity of the top 25 songs of the year via the Spotify API. Popularity is a scale designed by Spotify and given to us in the API.



The above bar chart compares the average revenue of top-grossing movies by genre during winning party election years (2012, 2016, 2020, 2024), broken down by Democratic wins (blue) and Republican wins (red).



The above bar chart shows the highest earning movie genre for each U.S. presidential election year (2012, 2016, 2020, 2024), with bars colored by the winning party—blue for Democratic wins and red for Republican wins. The results of 2020 are likely different from the other years due to COVID-19 pandemic.

Instructions for running our code: To run our code, start by generating the data files. First, run tmdbfile.py to collect movie data from the TMDb API, then election_web_scraping.py to scrape U.S. election results, and finally spotify_and_lastfm_api.py to gather top song data from Spotify and Last.fm. Once those JSON files are created, run sql_processing.py to build and populate the database. This script will create seven tables: tracks, music_genres, election_results, music_election, movie_genre_id, box_office_movies, and movie_election. After the data is in place, you can run the visualization files (mia_visualizations.py, mia_visualizations2.py, and dash_visualizations.py) to generate the graphs shown in our final project.

Documentation:

File: "sql_processing.py" Mia Goldstein Carolina Mondragon-Tadiotto

This file includes no specific functions, but imports json, sqlite3, datetime, and pandas to create the SQL tables. "Json_files" represent the json files gathered from the Spotify API, which are used with our "election_web_scrapping.json" which accesses the election data from the election data website. From here, we created separate tables that organize the data.

File: "tmdbfile.py" Mia Goldstein

This script uses The Movie Database (TMDb) API to collect and cache data about the highest-revenue movies by genre and year, focusing on 11 genres across 15 election-related years (year of the election, year before, and year after). The data is stored in a JSON cache file to avoid repeated API calls.

The get_json_content(filename) function reads a specified JSON file and returns its contents as a dictionary. If the file doesn't exist or fails to load, it safely returns an empty dictionary. Conversely, save_cache(cache, filename) writes a given dictionary to a specified file in JSON format, formatting it for readability and alerting the user if the process fails.

The fetch_genre_ids() function sends a request to the TMDb API and prints out a list of all available genre names along with their corresponding genre IDs. (Not called in main, was only used for testing).

To gather actual movie data, discover_movies_by_genre_year(genre_id, year) sends a request to TMDb's discover endpoint using a genre ID and a release year, sorted by revenue in descending order. It returns a list of movie summaries from the API results. The get_movie_details(movie_id) function takes a TMDb movie ID and fetches detailed information about that movie—this includes its full revenue, release date, and title, all of which are important for identifying top performers.

The main logic of the script resides in the update_cache() function. It loops through every combination of genre and year and checks whether a corresponding entry is already in the local cache. If not, it fetches movie results from TMDb using discover_movies_by_genre_year(), then calls get_movie_details() on each result to find the movie with the highest revenue. Once the top movie is identified, it stores that movie's title, release year, and revenue in the cache under a key formatted as Genre-Year (e.g., "Comedy-2012"). The function returns the final cache dictionary after processing all combinations.

Finally, the main() function calls update_cache() and prints a summary of all the movies that were cached, displaying each genre-year pair along with the movie title and its box office revenue. It also prints the total number of genre-year combinations cached, which is expected to be 165 (11 genres × 15 years). The script is meant to be run directly, and main() is called within the if __name__ == "__main__": block to execute the full caching process.

File: "spotify_and_lastfm_api.py" Carolina Mondragon-Tadiotto
First, get_tracks_from_playlist(playlist_id), takes in ID of spotify playlist and returns top 50 tracks. Second, get_track_details(track_id, artists) takes in ID and artists list and first artist's genre using Last.fm API, returns detailed info about specific track in organized dictionary. Third, get_and_export_by_year(year, playlist_id) returns JSON file of tracks by specific year. Lastly, process_all_years() returns JSON file of all years and their information.

File: "election_web_scraping.py" Carolina Mondragon-Tadiotto
Takes in a list of URLs from 2012 to 2024 to loop through, saving the political party, president, vice president, and other election details to be used later. Then returns all of this information into a JSON.

File: "mia_visualizations.py" Mia Goldstein
This script creates a bar chart comparing the average revenue of top-grossing movies across 11 genres during U.S. presidential election years where either the Democratic or

Republican party won (2012, 2016, 2020, 2024). The script connects to a SQLite database (sql_processing_clean.db) and executes a SQL query joining the box_office_movies, movie_genre_id, and election_results tables to retrieve revenue data by genre and winning party. It aggregates the revenues by party, calculates the average revenue per genre for Democratic and Republican wins, and visualizes the results using side-by-side bars. The Y-axis is formatted to display values in millions or billions for readability, and each bar is labeled with its corresponding revenue. This visualization helps highlight which genres perform better in box office revenue during years when either party wins a presidential election.

File: “mia_visualizations2.py” Mia Goldstein
This script creates a bar chart displaying the single highest-earning movie genre for each U.S. presidential election year in which a candidate won (2012, 2016, 2020, 2024). It connects to the sql_processing_clean.db database and joins the box_office_movies, movie_genre_id, and election_results tables to determine the top genre by maximum revenue for each year. Only 11 specified genres are included in the analysis. For each election year, the genre with the highest box office revenue is identified, and the corresponding revenue value is plotted. The bars are colored by the winning political party—blue for Democratic wins and red for Republican wins. Revenue amounts are formatted in millions or billions, and each bar is labeled with the genre name and its revenue. This visualization highlights genre dominance during winning election years and how they may align with party victories.

File: “dash_visualizations.py” Dashiell Kwan
Created visualizations for two different line graphs, one compared the average revenue for the last 4 election years and the other comparing the average popularity for the last 4 election years.

Resources:

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
April 9th, 2025	n/a	The American Presidency Project 2024	n/a

April 9th, 2025	n/a	The American Presidency Project 2020	n/a
April 9th, 2025	n/a	The American Presidency Project 2016	n/a
April 9th, 2025	n/a	The American Presidency Project 2012	n/a
April 8th, 2025	API used to gather genre information given an artist name	Last.fm API	n/a
April 8th, 2025	API used to gather tracks from playlists, along with other relevant information about the graph	Spotify API	Last.fm API was used to help gather genre information since Spotify was having trouble gathering the genre information for all artists (many had NAs when only using Spotify)
April 8th, 2025	API used to gather movie data, specifically their revenue and genre	TMDB API	n/a