Coding Challenge Metycle

Movie Data Analysis

Author: Álvaro Romeu

Email: alvaroromeumr@gmail.com

Index

Introduction

This report summarizes the steps taken to complete the Movie Data Analysis Challenge. The objective was to import a movie dataset into a relational database, perform SQL queries for data analysis, and generate insights from the provided data. This document contains all the steps and information about the challenge; however, it is recommended to also read the Jupyter Notebook *Coding Challenge Metycle – Movie Data Analysis.ipynb* to get a better visualization of the queries and figures.

To begin the challenge, a PostgreSQL database was set up using a Docker container. The database interaction and SQL querying were performed using DBeaver.

Additionally, a Jupyter Notebook was created to visualize the tables and queries created in DBeaver, offering a comprehensive overview of the data analysis process.

Objectives

- Run SQL queries to answer specific data analysis questions.
- Generate insights and visualizations based on the analysis.

Database Setup and Data Import

To begin the data analysis process, the first step involved setting up a PostgreSQL database using a Docker container. A PostgreSQL container was launched with the default port 5432. The database connection was established in DBeaver. This setup allowed to interact with the database directly for importing data and executing queries.

After successfully connecting to the database, three tables were created: titles, ratings, and credits. The titles table included columns such as movie ID, title, type, runtime, and genre. The ratings table stored IMDb-related metrics like IMDb score and votes, while the credits table contained actor details with columns for person_id, name, character, and role. The CSV datasets were imported into the respective tables using DBeaver's import tool, ensuring proper data structure alignment. The setup process concluded with a verification of data integrity and structure, confirming the successful creation of the tables and the ability to run SQL queries for analysis.

```sql
create table titles (
    id VARCHAR primary KEY,
    title TEXT,
    type VARCHAR,
    description TEXT,
    release_year INT,
    age_certification VARCHAR,
    runtime INT,
    genres TEXT,
    production_countries TEXT,
    seasons INT,
    imdb_id VARCHAR,
    imdb_score NUMERIC,
    imdb_votes INT,
    tmdb_popularity NUMERIC,
    tmdb_score NUMERIC
);
```

Figure 1: Table titles.

```sql
create table ratings (
    id VARCHAR primary key,
    title TEXT,
    type VARCHAR,
    description TEXT,
    release_year INT,
    age_certification VARCHAR,
    runtime INT,
    imdb_id VARCHAR,
    imdb_score NUMERIC,
    imdb_votes INT
);
```

Figure 2: Table ratings.

```sql
create table credits (
    person_id VARCHAR,
    id VARCHAR,
    name VARCHAR,
    character VARCHAR,
    role VARCHAR,
    primary key (person_id, id, role)
);
```

Figure 3: Table credits

After the data import, a deduplicated list of movies and shows was created by merging the tables titles and ratings into a new table called combined_titles. This table was designed to remove any duplicate entries while keeping all the fields both tables had in common. Each movie and show appeared only once in this table, making it suitable for consistent analysis.

```sql
create table combined_titles as
select distinct
    t.id, t.title, t.type, t.description, t.release_year,
    t.age_certification, t.runtime, t.genres, t.production_countries,
    t.imdb_id, t.imdb_score, t.imdb_votes, t.tmdb_popularity, t.tmdb_score
from titles t
left join ratings r on t.id = r.id;

select * from combined_titles;
```

Figure 4: Table Combined_titles

To enrich the dataset further, another table named combined_titles_with_credits was created, which combined the information from combined_titles with the credits table. This final table included all movie details along with the actors and their roles, making it easier to analyze actor appearances and their involvement in each title.

```sql
create table combined_titles_with_credits as
select
    t.*,
    c.person_id,
    c.name as actor_name,
    c.character,
    c.role
from combined_titles t
left join credits c on t.id = c.id;
```

Figure 5: Table Combined_titles_with_credits

After creating all the tables and completing the exercises in DBeaver, I proceeded to develop a Jupyter Notebook to visualize the results more effectively.

Jupyter Notebook to visualize data

First, the necessary libraries to analyze and visualize data were imported. Then, the PostgreSQL package was installed to be able to connect the Jupyter Notebook with the PostgreSQL database created in DBeaver. After the installation, the connection with the database was established.

Below, the queries used with Python to realize the previous steps are shown:

```
# Importing libraries
import pandas as pd
from sqlalchemy import create_engine
import matplotlib.pyplot as plt
import seaborn as sns

# Installing package for PostgreSQL
pip install psycopg2-binary
```

```
Collecting psycopg2-binary
  Using cached psycopg2-binary-2.9.10.tar.gz (385 kB)
  Preparing metadata (setup.py) ... e=psycopg2_binary-2.9.10-cp312-cp312-
macosx_11_0_arm64.whl size=134383
sha256=0b31d28c984b77ded2b0a22fafe83f417bf76d3db53ccb31066df6500fa9ae38
  Stored in directory:
/Users/alvaroromeu/Library/Caches/pip/wheels/06/bc/a4/bad5bdabd4cf012a00e927d
b042e0e44d3a649596c548212be
Successfully built psycopg2-binary
Installing collected packages: psycopg2-binary
Successfully installed psycopg2-binary-2.9.10
Note: you may need to restart the kernel to use updated packages.
```

```
# Establishing connection to PostgreSQL database
engine =
create_engine('postgresql://postgres:postgres@localhost:5432/postgres')
```

Loading Tables

The next step involved loading all the tables from the database into the Jupyter Notebook to start analyzing and visualizing the data. The following queries were used to load the tables (titles, ratings, credits, combined_titles and combined_titles_with_credits) and to visualize them.

```
# Loading tables
tables = pd.read_sql("""
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public';
""", con=engine)
```

[9]:

	table_name
0	combined_titles
1	combined_titles_with_credits
2	titles
3	ratings
4	credits

Figure 6: Tables created in DBeaver transferred to Notebook

```
# Loading data from titles
titles_df = pd.read_sql("""
SELECT *
FROM titles
LIMIT 10;
""", con=engine)

titles_df
```

```
# Loading data from ratings
ratings_df = pd.read_sql("""
SELECT *
FROM ratings
LIMIT 10;
""", con=engine)

ratings_df
```

```
# Loading data from credits
credits_df= pd.read_sql("""
SELECT *
FROM credits
LIMIT 10;
""", con=engine)

credits_df
```

```
# Loading data from combined_titles
combined_titles_df= pd.read_sql("""
```

```
SELECT *
FROM combined_titles
LIMIT 10;
""", con=engine)

combined_titles_df

# Loading data from combined_titles_with_credits
combined_titles_with_credits_df = pd.read_sql("""
SELECT *
FROM combined_titles_with_credits
WHERE title IS NOT NULL
LIMIT 10;
""", con=engine)

combined_titles_with_credits_df
```

SQL Queries and Data Visualization

After all the tables were loaded, it was proceeded with the SQL queries to answer the questions about the challenge. Every question contains the query used to find the solution and below each query the answer to the question can be seen.

What are the longest and shortest run times among movies?

```
# Query for the 5 film titles with the longest running time

longest_movie = pd.read_sql("""
SELECT title, runtime
FROM combined_titles
WHERE type = 'MOVIE'
ORDER BY runtime DESC
LIMIT 5;
""", con=engine)

print("Top 5 Longest Movies:")
longest_movie
```

Top 5 Longest Movies:

	title	runtime
0	Bonnie & Clyde	240
1	A Lion in the House	225
2	Lagaan: Once Upon a Time in India	224

```
3                       Jodhaa Akbar      214
4           Kabhi Khushi Kabhie Gham      210
```

```
# Query for the 5 film titles with the shortest running time
shortest_movie = pd.read_sql("""
SELECT title, runtime
FROM combined_titles
WHERE type = 'MOVIE'
ORDER BY runtime ASC
LIMIT 5;
""", con=engine)

print("Top 5 Shortest Movies:")
shortest_movie
```

Top 5 Shortest Movies:

```
                   title  runtime
0          Time to Dance        2
1                 Silent        3
2            Sol Levante        4
3  Amsterdam to Anatolia        6
4                  Match        7
```

Is there an increase or decrease in the average runtime per release year?

```
# Query to analyze the average runtime trend over the years for movies and
shows
runtime_trend = pd.read_sql("""
SELECT release_year, type, AVG(runtime) AS avg_runtime
FROM combined_titles
GROUP BY release_year, type
ORDER BY release_year;
""", con=engine)

# Plotting the average runtime trend
plt.figure(figsize=(12,6))
sns.lineplot(data=runtime_trend, x='release_year', y='avg_runtime',
hue='type', marker='o')
plt.title("Average Runtime per Year for Movies and Shows")
plt.xlabel("Year")
plt.ylabel("Average Runtime (minutes)")
plt.grid(True)
plt.show()
```

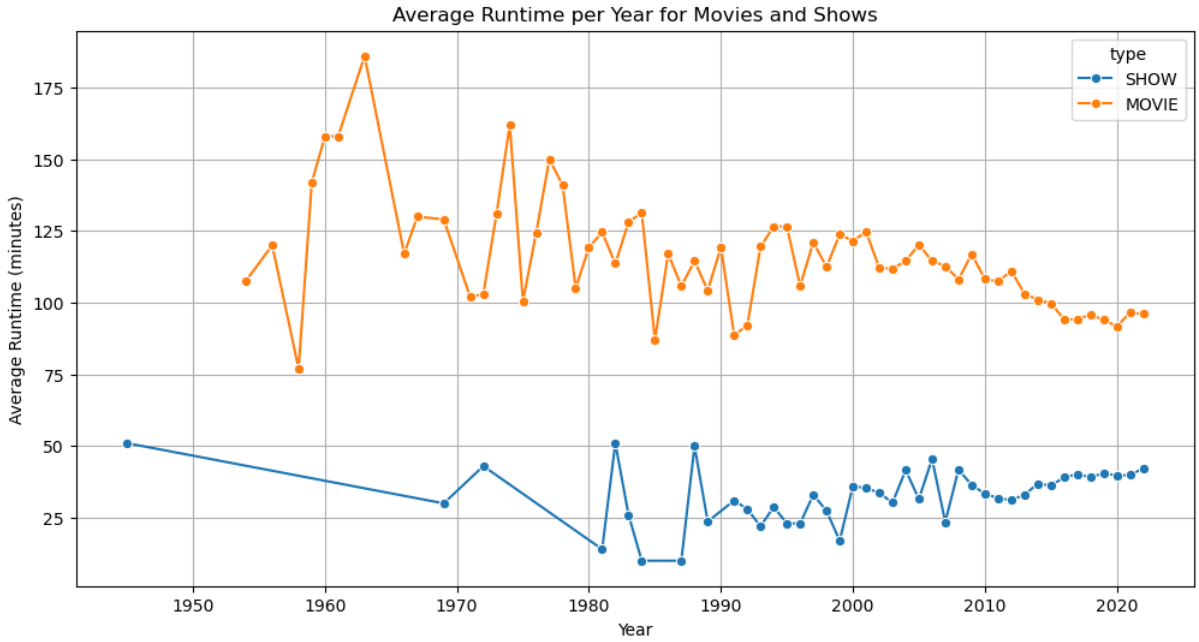


Figure 7: Average Runtime per Year for Movies and Shows

The graph shows a general trend of decreasing average length of MOVIES over time.

While SHOWS demonstrate greater variability, although with a slight decrease in recent decades. But, since 2010, there can be seen a recovery and a slight increase in the length of SHOWS.

Top 10 Longest and Shortest Titles

```
# Query to get top 10 longest titles based on title length

longest_titles = pd.read_sql("""
SELECT title, LENGTH(title) AS title_length
FROM combined_titles
WHERE title IS NOT NULL AND LENGTH(title) > 0
ORDER BY title_length DESC
LIMIT 10;
""", con=engine)

print("10 Longest Titles:")
longest_titles
```

10 Longest Titles:

	title	title_length
0	Jim & Andy: The Great Beyond - Featuring a Ver...	104
1	Steve Martin and Martin Short: An Evening You ...	83
2	One Piece: Episode of Chopper Plus: Bloom in t...	79
3	Cultivating the Seas: History and Future of th...	79
4	The Road to El Camino: Behind the Scenes of El...	75
5	Judah Friedlander: America Is the Greatest Cou...	71
6	One Piece: The Desert Princess and the Pirates...	69
7	LEGO Marvel Super Heroes: Guardians of the Gal...	69
8	Tim Minchin and the Heritage Orchestra: Live a...	69
9	The Lonely Island Presents: The Unauthorized B...	69

```
# Query to get top 10 shortest titles based on title length
shortest_titles = pd.read_sql("""
SELECT title, LENGTH(title) AS title_length
FROM combined_titles
WHERE title IS NOT NULL AND LENGTH(title) > 0
ORDER BY title_length ASC
LIMIT 10;
""", con=engine)

print("10 Shortest Titles:")
shortest_titles
```

10 Shortest Titles:

	title	title_length
0	H	1
1	3%	2
2	PK	2
3	83	2
4	IO	2
5	It	2
6	42	2
7	21	2
8	She	3
9	Don	3

Movies and Average IMDb Score by Number of Votes

```
# Query to analyze the distribution of movies and average IMDb score by
number of votes
votes_binned = pd.read_sql("""
SELECT
    CASE
```

```
        WHEN imdb_votes BETWEEN 0 AND 1000 THEN '0-1000'
        WHEN imdb_votes BETWEEN 1001 AND 5000 THEN '1001-5000'
        WHEN imdb_votes BETWEEN 5001 AND 10000 THEN '5001-10000'
        ELSE '>10000'
    END AS vote_range,
    COUNT(*) AS num_movies,
    AVG(imdb_score) AS avg_score
FROM combined_titles
GROUP BY vote_range
ORDER BY vote_range;
""", con=engine)

# Plotting the number of movies and average IMDb score by vote range
fig, ax1 = plt.subplots(figsize=(12,6))

# Bar plot for number of movies
sns.barplot(x="vote_range", y="num_movies", data=votes_binned,
color='orange', alpha=0.7, ax=ax1)
ax1.set_ylabel("Number of Movies")
ax1.set_xlabel("Range IMDb Votes")

# Line plot for average IMDb score
ax2 = ax1.twinx()
sns.lineplot(x="vote_range", y="avg_score", data=votes_binned, marker='o',
color='red', ax=ax2)
ax2.set_ylabel("Average IMDb Score")

plt.title("Movies and Average IMDb Score by Number of Votes")
plt.tight_layout()
plt.show()
```

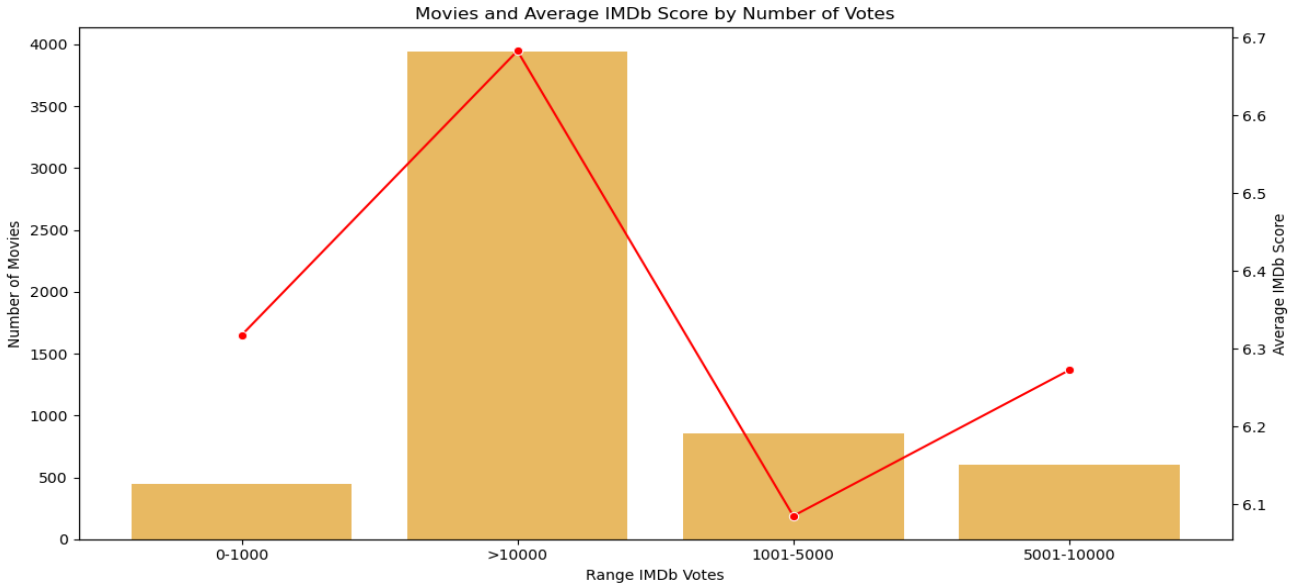


Figure 8: Movies and Average IMDb Score by Number of Votes

The graph shows that most films have more than 10,000 votes, indicating that popular films tend to accumulate a high number of votes. However, the average IMDb score does not necessarily increase with the number of votes. In fact, films with fewer votes (0-1000) have a higher average score, while those in the intermediate range (1001-5000 votes) tend to have a lower score. This may reflect that films with fewer votes may be influenced by a narrower and more specific audience, while more popular films receive a greater diversity of opinions, leading to more moderate scores.

Number of Movies per Genre

```
# Query to get the top 10 genres with most movies (splitting combined genres
correctly)

movies_per_genre = pd.read_sql("""
SELECT

      TRIM(BOTH ‘ ‘ FROM UNNEST(

            STRING_TO_ARRAY(

                  REPLACE(REPLACE(REPLACE(REPLACE(genres, '[', ''), ']', ''),
            '"', ''), '''', ''), ','

                  )

      )) AS genre,

      COUNT(*) AS num_movies

FROM combined_titles
GROUP BY genres
ORDER BY num_movies DESC;
""", con=engine)

# Bar plot for top 10 genres with most movies
plt.figure(figsize=(10, 6))
sns.barplot(y="genres", x="num_movies", data=clean_genres.head(10))
plt.title("Top 10 Genres with Most Movies")
plt.xlabel("Number of Movies")
plt.ylabel("Genres")
plt.tight_layout()
plt.show()
```

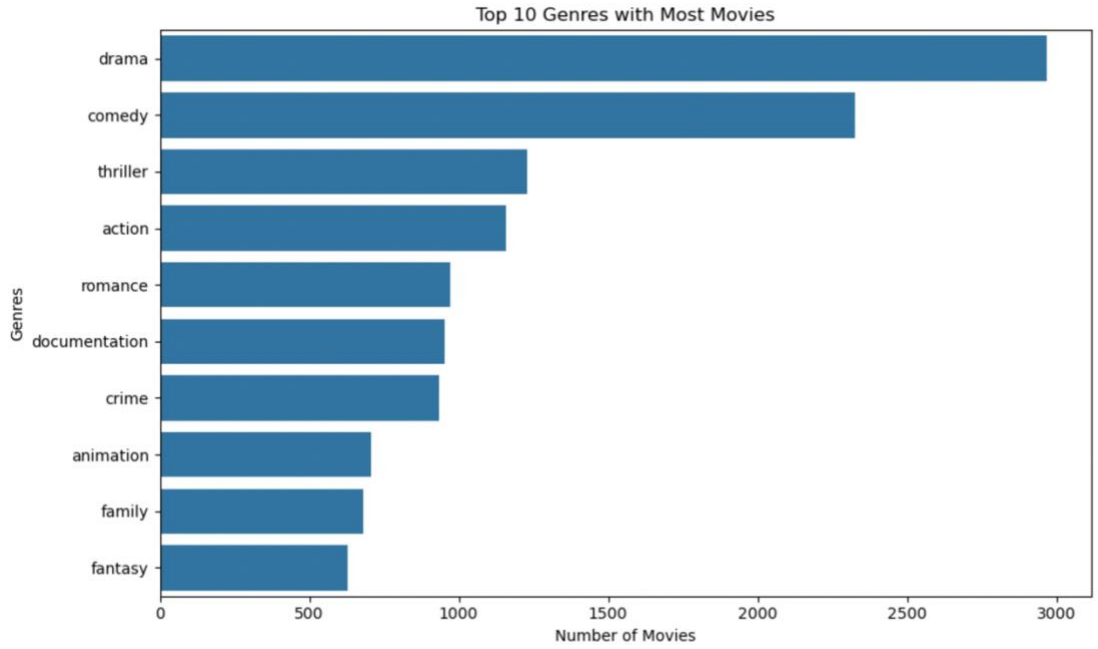


Figure 9: Top 10 Genres with Most Movies

The chart displays the top 10 individual genres with the highest number of movies in the dataset. *Drama* and *Comedy* are the most dominant genres, with significantly higher counts compared to others, indicating their popularity and prevalence in the industry. Following them, *Thriller*, *Action*, and *Romance* also appear frequently but with lower representation. The presence of *Documentation*, *Crime*, and *Animation* genres suggests a diverse range of content, while *Family* and *Fantasy* complete the list with moderate counts.

Top Actors by Number of Movies per Year

```
# Query to get top actors by number of movies per year

most_frequent_actors = pd.read_sql("""
SELECT release_year, actor_name, COUNT(*) AS num_movies
FROM combined_titles_with_credits
WHERE role = 'ACTOR'
GROUP BY release_year, actor_name
ORDER BY num_movies DESC
LIMIT 10;
""", con=engine)


print("The most frequent actors per year were:")
most_frequent_actors
```

The most frequent actors per year were:

	release_year	actor_name	num_movies
0	2019	Tiffany Haddish	6
1	2021	London Hughes	6
2	2021	Fortune Feimster	6
3	2018	Yuki Kaji	6
4	2009	Kareena Kapoor Khan	5
5	2020	Vir Das	5
6	2018	Kana Hanazawa	5
7	2016	Fred Tatasciore	5
8	2017	Ron Funches	5
9	2021	Vijay Sethupathi	5

Conclusion

This Jupyter Notebook effectively demonstrates data analysis and visualization using PostgreSQL, SQL queries, and Python libraries. Throughout this project, a movie dataset was explored by performing various queries and generating insights. The genres comedy and drama were identified as the most represented. Additionally, genres such as thriller, action, and romance were also prevalent in the dataset, but to a lesser extent than drama and comedy. When analyzing runtime, Bonnie & Clyde (240 minutes) and A Lion in the House (225 minutes) were among the longest films, while Time to Dance (2 minutes) and Silent (3 minutes) were the shortest. Over time, the average movie runtime has shown a downward trend since the early 2000s, while shows have experienced greater variability, with a slight upward trend after 2010.

Regarding title lengths, the longest titles include *"Jim & Andy: The Great Beyond"* with 104 characters, while the shortest include *"H"* with just one character. Analyzing IMDb scores against vote counts revealed that films with fewer votes (0-1000) tend to have higher average ratings compared to those with intermediate votes (1001-5000), suggesting that niche films might attract more favorable reviews from a smaller audience, while widely popular films gather more diverse ratings. Finally, the analysis of actors with the most movie appearances by year revealed Tiffany Haddish, London Hughes, and Fortune Feimster as top recurring actors in recent years, with up to 6 movies in a single year.

This project provided insights into the distribution of movie genres, runtimes, and ratings and demonstrated the importance of SQL queries and data visualization techniques for effective data storytelling. Further steps could involve predictive modeling and exploring correlations between genres, ratings, and success metrics for deeper insights.

Figures

```
create table titles (
    id VARCHAR primary KEY,
    title TEXT,
    type VARCHAR,
    description TEXT,
    release_year INT,
    age_certification VARCHAR,
    runtime INT,
    genres TEXT,
    production_countries TEXT,
    seasons INT,
    imdb_id VARCHAR,
    imdb_score NUMERIC,
    imdb_votes INT,
    tmdb_popularity NUMERIC,
    tmdb_score NUMERIC
);
```

Figure 1: Table titles.

```
create table ratings (
    id VARCHAR primary key,
    title TEXT,
    type VARCHAR,
    description TEXT,
    release_year INT,
    age_certification VARCHAR,
    runtime INT,
    imdb_id VARCHAR,
    imdb_score NUMERIC,
    imdb_votes INT
);
```

Figure 2: Table ratings.

```
create table credits (
    person_id VARCHAR,
    id VARCHAR,
    name VARCHAR,
    character VARCHAR,
    role VARCHAR,
    primary key (person_id, id, role)
);
```

Figure 3: Table credits

```
create table combined_titles as
select distinct
    t.id, t.title, t.type, t.description, t.release_year,
    t.age_certification, t.runtime, t.genres, t.production_countries,
    t.imdb_id, t.imdb_score, t.imdb_votes, t.tmdb_popularity, t.tmdb_score
from titles t
left join ratings r on t.id = r.id;


select * from combined_titles;
```

Figure 4: Table Combined_titles

```
create table combined_titles_with_credits as
select
    t.*,
    c.person_id,
    c.name as actor_name,
    c.character,
    c.role
from combined_titles t
left join credits c on t.id = c.id;
```

Figure 5: Table Combined_titles_with_credits

[9]:

	table_name
0	combined_titles
1	combined_titles_with_credits
2	titles
3	ratings
4	credits

Figure 6: Tables created in DBeaver transferred to Notebook

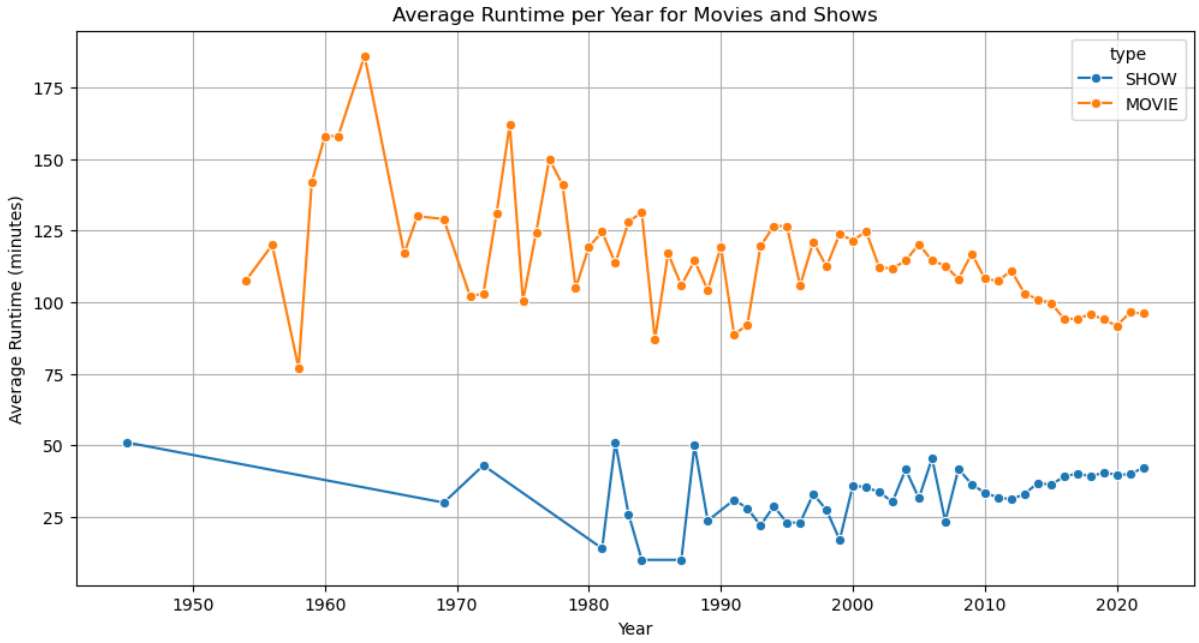


Figure 7: Average Runtime per Year for Movies and Shows

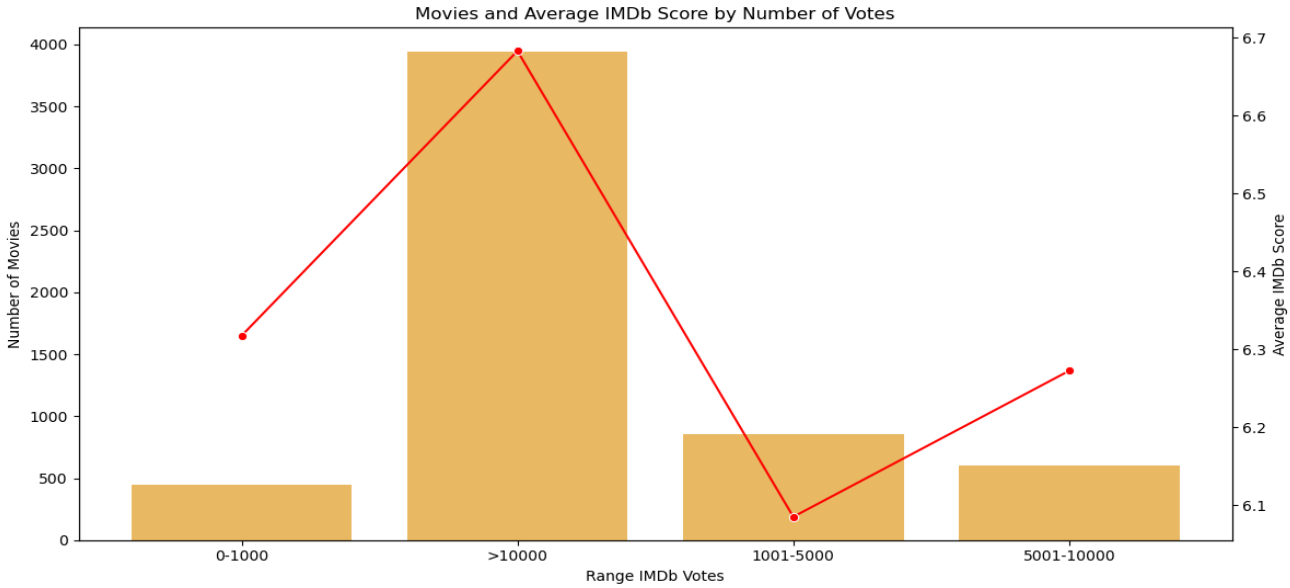


Figure 8: Movies and Average IMDb Score by Number of Votes

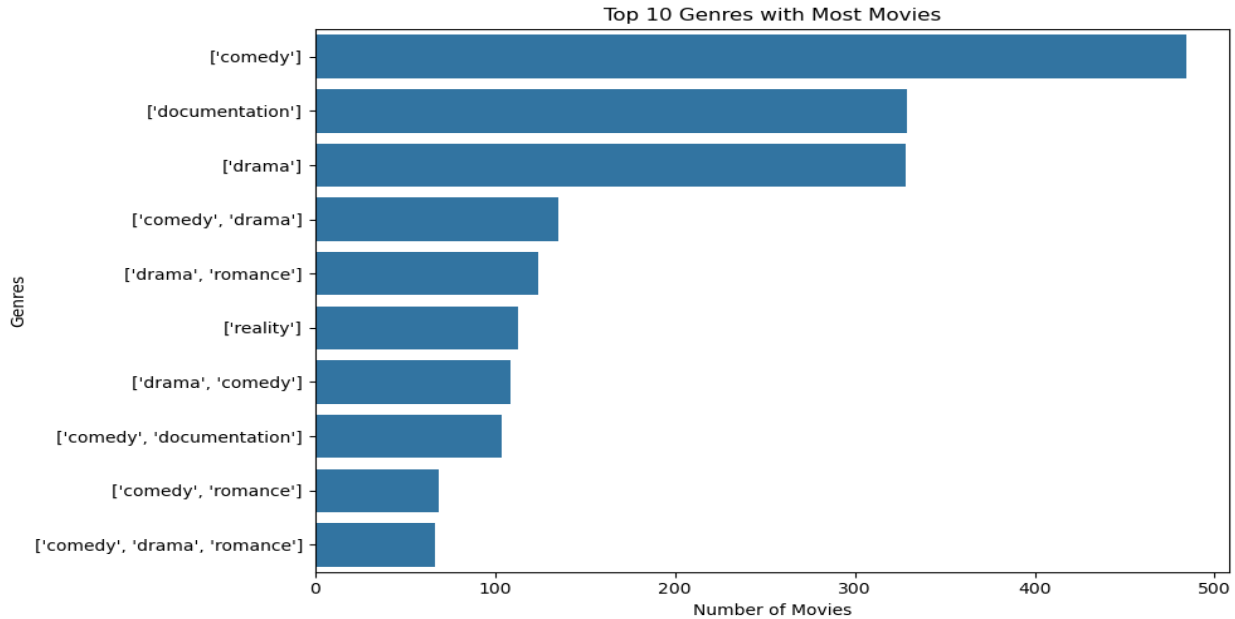
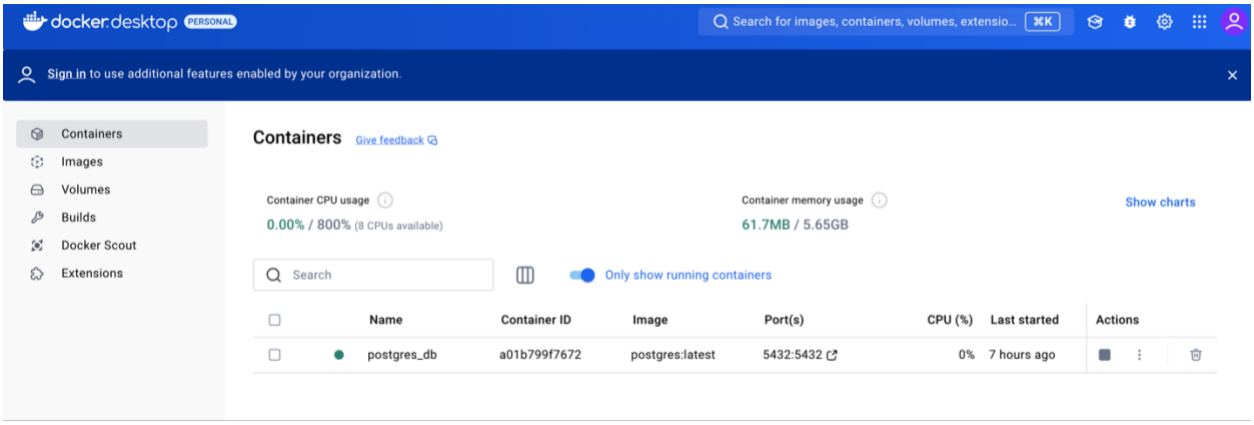
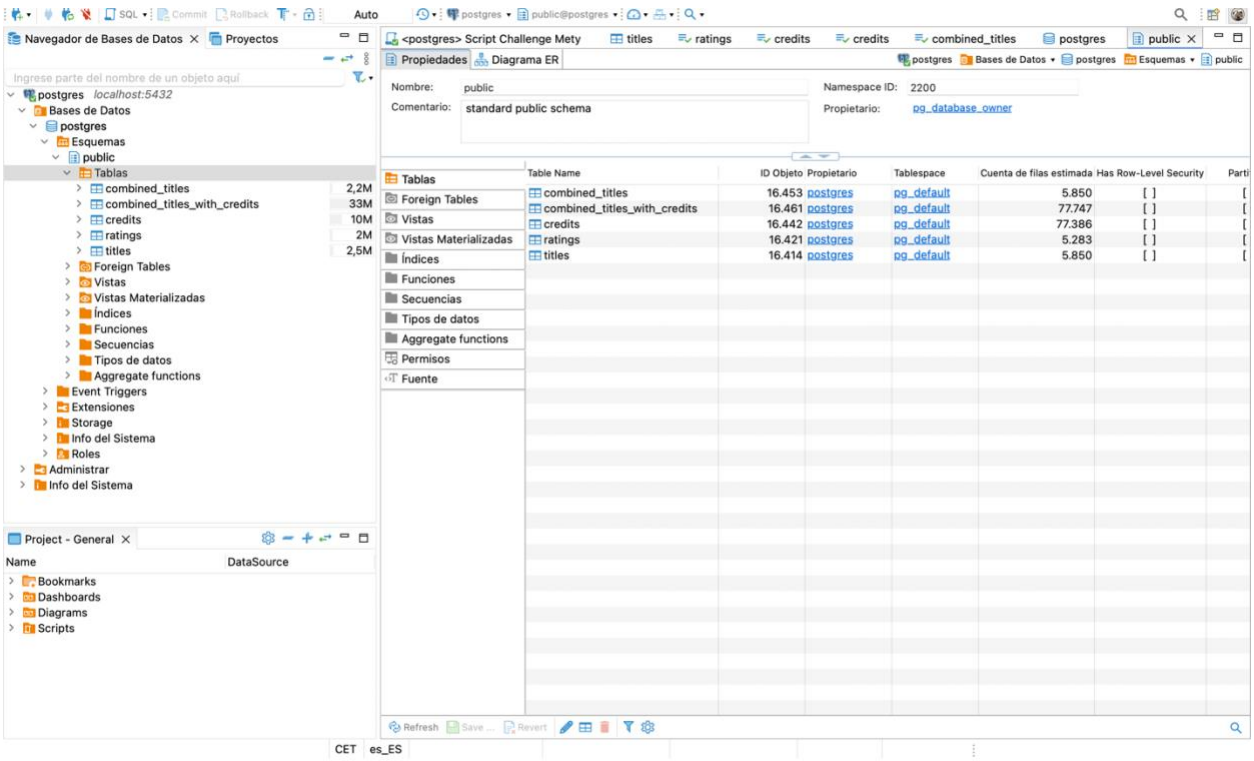


Figure 9: Top 10 Genres with Most Movies

Screenshots



Screenshot 1: Docker container



Screenshot 2: Tables Created in DBeaver