IMDB 2024 Data Scraping and Visualizations

By

Shanchai Kumar S

Abstract

The **IMDB 2024 Data Scraping and Visualizations** project aims to extract, process, and analyse movie data from IMDb for the year 2024. Using **Selenium** for web scraping, the project collects essential movie attributes, including names, genres, ratings, voting counts, and durations. The data is then structured genre-wise, stored as CSV files, and integrated into an **SQL database** for efficient querying.

To gain insights, **data analysis and visualizations** are performed using **Pandas, Matplotlib, and Seaborn**, with an interactive **Streamlit dashboard** allowing users to explore trends dynamically. Key analyses include **top-rated movies, genre distributions, voting trends, duration patterns, and rating correlations**. The application enables **real-time filtering**, allowing users to customize their search based on ratings, duration, votes, and genre.

This project strengthens skills in **Python, Selenium, SQL, data cleaning, visualization, and interactive applications**, making it an excellent learning experience in **data analytics and entertainment insights**. With an emphasis on maintainability, portability, and industry best practices, the project is documented on **GitHub** with a mandatory **README, demo video, and evaluation criteria compliance**.

1. PROBLEM DEFINITION AND OBJECTIVES

1.1 PROBLEM DEFINITION:

This project focuses on extracting and analyzing movie data from IMDb for the year 2024. The task involves scraping data such as movie names, genres, ratings, voting counts, and durations from IMDb's 2024 movie list using Selenium. The data will then be organized genre-wise, saved as individual CSV files, and combined into a single dataset stored in an SQL database. Finally, the project will provide interactive visualizations and filtering functionality using Streamlit to answer key questions and allow users to customize their exploration of the dataset.

1.2 OBJECTIVES

- **Web Scraping & Data Collection**
  - Extract movie data from IMDb's 2024 listings using **Selenium**.
  - Gather key attributes such as **movie name, genre, ratings, voting counts, and duration**.
  - Store the scraped data in **CSV files**, categorized by genre.
- **Data Storage & Management**
  - Merge genre-wise data into a single **structured dataset**.
  - Store the cleaned dataset in an **SQL database** for efficient querying and future analysis.
- **Data Cleaning & Processing**
  - Handle missing or inconsistent data to ensure **accuracy and completeness**.
  - Standardize data formats (e.g., duration in minutes, numerical rating scales).
- **Data Analysis & Insights**

- ❖ Identify **top-rated movies** based on ratings and voting counts.
- ❖ Analyze **genre distributions** and their popularity in 2024.
- ❖ Study **movie durations** across different genres.
- ❖ Examine **rating patterns** and **voting trends** for better understanding.

➢ **Data Visualization & Dashboard Development**

- ❖ Create **interactive visualizations** using **Matplotlib, Seaborn, and Streamlit**.
- ❖ Develop charts and tables to showcase key insights such as **genre-wise ratings, voting distributions, and duration trends**.
- ❖ Implement a **heatmap** for comparing average ratings across genres.

➢ **Interactive Filtering Functionality**

- ❖ Enable users to filter movies based on **ratings, duration, votes, and genre**.
- ❖ Provide **real-time updates** and **dynamic search capabilities** through a **Streamlit-based interface**.

2. Project Approach

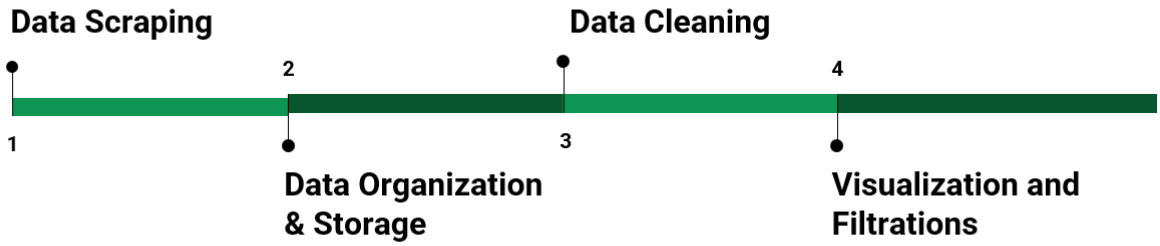


Fig 1. Project approach

3. Data Scraping

3.1 Overview

The **IMDb 2024 Data Scraping and Visualization** project involves extracting movie-related data from IMDb's official 2024 movie listings. This phase focuses on **automating the extraction process** using **Selenium**, storing data in structured CSV files, and ensuring data quality for further analysis.

3.2 Data Source

- **Website**: IMDb – 2024 Movies List *(link)*
- **Data Type**: Movie details such as:
  - **Movie Name**
  - **Genre**
  - **Ratings**
  - **Voting Counts**
  - **Duration**
- **Genres Covered**: Action, Comedy, Drama, Crime, Family, News, Talk-Show, Game-Show, War, Western

3.3 Tools & Technologies Used

Technology	Purpose
Python	Main programming language for automation
Selenium	Automates web scraping
Pandas	Data manipulation and storage
CSV	Format for saving extracted data
MySQL	Storing and querying the dataset

3.4 Web Scraping Approach

Step 1: Import Required Libraries

Before starting the web scraping process, we need to import several **Python libraries** that help us interact with IMDb, extract data, and store it properly.

```
import os
import time
import glob
import mysql
import pandas as pd
from selenium import webdriver
from sqlalchemy import create_engine
from selenium.webdriver.common.by import By
from selenium.common.exceptions import TimeoutException,
NoSuchElementException, ElementClickInterceptedException
```

Explanation:

- **selenium**: Automates browser actions to scrape IMDb.
- **pandas**: Used for data processing and storing structured data.
- **time**: Introduces pauses in execution to avoid blocking.
- **mysql**: Allows interaction with the database for data storage.
- **sqlalchemy**: Helps connect Python with MySQL databases.

Step 2: Define IMDb Genre URLs

Since we are scraping movies from **multiple genres**, we need to specify a **list of IMDb URLs**, each corresponding to a different genre.

```
genre_urls = [
  "https://www.imdb.com/search/title/?title_type=feature&release_date=2024-01-
01,2024-12-31&genres=news",
  "https://www.imdb.com/search/title/?title_type=feature&release_date=2024-01-
01,2024-12-31&genres=talk-show",
```

```
  "https://www.imdb.com/search/title/?title_type=feature&release_date=2024-01-01,2024-12-31&genres=game-show",
  "https://www.imdb.com/search/title/?title_type=feature&release_date=2024-01-01,2024-12-31&genres=war",
  "https://www.imdb.com/search/title/?title_type=feature&release_date=2024-01-01,2024-12-31&genres=western",
  "https://www.imdb.com/search/title/?title_type=feature&release_date=2024-01-01,2024-12-31&genres=action",
  "https://www.imdb.com/search/title/?title_type=feature&release_date=2024-01-01,2024-12-31&genres=comedy",
  "https://www.imdb.com/search/title/?title_type=feature&release_date=2024-01-01,2024-12-31&genres=drama",
  "https://www.imdb.com/search/title/?title_type=feature&release_date=2024-01-01,2024-12-31&genres=crime",
  "https://www.imdb.com/search/title/?title_type=feature&release_date=2024-01-01,2024-12-31&genres=family"]
```

Explanation:

- We create a **list of URLs**, where each URL corresponds to a different genre.
- The **release_date filter (2024-01-01 to 2024-12-31)** ensures that only 2024 movies are collected.

Step 3: Implement Web Scraper Function

The function webscrapper(url) is responsible for scraping the **movie details** from the IMDb page.

```
# Function for scrapping the movie data from website
def webscrapper(url):

  # Initialize the WebDriver (Chrome in this case)
  driver = webdriver.Chrome()

  try:
    # Open the IMDb page specified by the URL
    driver.get(url)
    # Maximize the browser window for better visibility
```

```
    driver.maximize_window()
    # Wait for 2 seconds to ensure the page is fully loaded
    time.sleep(2)
    # Print the title of the page to confirm it loaded correctly
    print(driver.title)

    # Attempt to click the "Read More" button to load all the data dynamically
    while True:
      try:
        # Locate the "Read More" button using its XPath
        element = driver.find_element(By.XPATH,
'//*[@id="__next"]/main/div[2]/div[3]/section/section/div/section/section/div[2]/div/sec
tion/div[2]/div[2]/div[2]/div/span/button')
        # Scroll the button into view if it's not currently visible
        driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", element)
        # Wait for 1 second to ensure the button is visible
        time.sleep(1)
        # Click the "Read More" button to load more content
        element.click()
        # Print a message when the button is clicked
        print("Clicked 'Read More' button.")
        # Wait for 1 second before trying again
        time.sleep(1)
      except NoSuchElementException:
        # Exit loop if the "Read More" button is no longer available (all data is loaded)
        print("No 'Read More' button found. All data loaded.")
        break
      except ElementClickInterceptedException:
        # If the button is blocked by another element, retry after a short delay
        print("Button is blocked by another element. Retrying...")
        time.sleep(2)
      except TimeoutException:
        # Handle cases where the operation times out and retry
        print("Operation timed out. Retrying...")
        time.sleep(2)
      except Exception as e:
        # Catch any other unexpected errors
        print(f"Unexpected error: {e}")
        break

    print("Successfully retrieved all the data.")
```

```
    # Initialize a dictionary to store movie data categorized by genre
    genre_data = {}

    # Locate all movie items on the page
    movies =
driver.find_elements(By.XPATH,'//*[@id="__next"]/main/div[2]/div[3]/section/section/div/section/section/div[2]/div/section/div[2]/div[2]/ul/li')

    # Extract details for each movie
    for movie in movies:
      try:
        # Extract the movie name, ensuring it splits correctly to remove unnecessary
text
        name = movie.find_element(By.CSS_SELECTOR, 'h3[class="ipc-
title__text"]').text.split(". ", 1)[1]

        # Attempt to extract the genre of the movie, using a fallback if not found
        try:
          genre = movie.find_element(By.XPATH,
'//*[@id="__next"]/main/div[2]/div[3]/section/section/div/section/section/div[2]/div/section/div[1]/div/div/div[2]/button[3]/span').text.strip()
        except NoSuchElementException:
          genre = "Unknown" # If no genre is found, mark it as "Unknown"

        # Extract movie rating, handling cases where it's missing
        try:
          rating = movie.find_element(By.CSS_SELECTOR, "span[class='ipc-rating-star-
-rating']").text.strip()
        except NoSuchElementException:
          rating = "N/A" # If no votes are found, mark it as "N/A"

        # Extract vote count, formatting it correctly and handling missing data
        try:
          votes = movie.find_element(By.CSS_SELECTOR, "span[class='ipc-rating-star-
-voteCount']").text.replace("(", "").replace(")", "").strip()
        except NoSuchElementException:
          votes = "N/A" # If no votes are found, mark it as "N/A"

        # Extract movie duration, using a fallback if not found
        try:
```

```
            duration = movie.find_element(By.XPATH,
'./div/div/div/div[1]/div[2]/div[2]/span[2]').text.strip()
          except NoSuchElementException:
            duration = "N/A" # If no votes are found, mark it as "N/A"

          # Split the genre(s) into a list and store movie data in a dictionary under each
genre
          for g in genre.split(", "):
            if g not in genre_data:
              genre_data[g] = []  # Initialize an empty list for new genres
            # Append movie details to the respective genre's list
            genre_data[g].append({
              "Movie Name": name,
              "Rating": rating,
              "Votes": votes,
              "Duration": duration,
              "Genre": genre
            })
        except Exception as e:
          # Handle errors that may occur while processing individual movies
          print(f"Error processing movie: {e}")

    return genre_data  # Return the dictionary containing movie data organized by genre

  except Exception as e:
    # Handle errors that occur while retrieving or processing the page data
    print(f"Error retrieving movie list: {e}")
    return {} # Return an empty dictionary if an error occurs

  finally:
    driver.quit()  # Quit the WebDriver when finished, ensuring resources are released
```

Explaination:

1. Initialize the Selenium WebDriver

```
def webscrapper(url):
  driver = webdriver.Chrome()  # Initialize Selenium WebDriver
```

- **Purpose**: Opens a **Chrome browser** instance to navigate the IMDb website.
- **Why?** IMDb is a **dynamic website**, meaning data is loaded **as you scroll or interact with the page**. Selenium helps automate this process.

2. Load the IMDb Webpage

```
try:
    driver.get(url)  # Open the IMDb page
    driver.maximize_window()  # Maximize the window for visibility
    time.sleep(2)  # Wait for the page to load
    print(driver.title)  # Print the page title to verify loading
```

- **driver.get(url):** Navigates to the **IMDb genre-specific URL**.
- **driver.maximize_window():** Expands the browser window to ensure all elements are visible.
- **time.sleep(2):** Introduces a delay to allow the webpage to load completely.
- **print(driver.title):** Prints the webpage title (e.g., "IMDb: Feature Films 2024 - Action") to verify we are on the correct page.

3. Click "Read More" Button to Load More Content

IMDb loads additional content **dynamically** when users click "Read More".

```
    while True:
        try:
            element = driver.find_element(By.XPATH,
'//*[@id="__next"]/main/div[2]/div[3]/section/section/div/section/section/div[2]/div/sec
tion/div[2]/div[2]/div[2]/div/span/button')
            driver.execute_script("arguments[0].scrollIntoView({block: 'center'});", element)
            time.sleep(1)
            element.click()
```

```
        print("Clicked 'Read More' button.")
        time.sleep(1)
```

Explanation:

- **Find the "Read More" button** using XPath.
- **Scroll it into view** to ensure it's visible.
- **Click the button** to load more movies.
- **Repeat** until the button **disappears** (meaning all movies are loaded).

Handling Errors

```
    except NoSuchElementException:
        print("No 'Read More' button found. All data loaded.")
        break
    except Exception as e:
        print(f"Unexpected error: {e}")
        break
```

- **NoSuchElementException**: If the button is missing, exit the loop.
- **Generic Exception Handling**: If an unexpected error occurs, print it and exit.

4. Extract Movie Details

After ensuring all movies are visible, extract movie details from the page.

```
    genre_data = {}  # Dictionary to store movies per genre
    movies = driver.find_elements(By.XPATH,
'//*[@id="__next"]/main/div[2]/div[3]/section/section/div/section/section/div[2]/div/section/div[2]/div[2]/ul/li')
```

- **Find all movie elements** using XPath.
- **Store extracted data** in a dictionary (genre_data).

Extract Specific Movie Data

```
    for movie in movies:
        try:
            name = movie.find_element(By.CSS_SELECTOR, 'h3[class="ipc-
title__text"]').text.split(". ", 1)[1]
            rating = movie.find_element(By.CSS_SELECTOR, "span[class='ipc-rating-star--
rating']").text.strip()
            votes = movie.find_element(By.CSS_SELECTOR, "span[class='ipc-rating-star--
voteCount']").text.replace("(", "").replace(")", "").strip()
            duration = movie.find_element(By.XPATH,
'./div/div/div/div[1]/div[2]/div[2]/span[2]').text.strip()
```

- **Movie Name**: Extracted from <h3> tags. IMDb prefixes names with **numbers**, so split(". ", 1)[1] removes them.
- **Rating**: Retrieved using CSS selectors (ipc-rating-star--rating).
- **Votes**: Extracted and cleaned to remove unnecessary brackets.
- **Duration**: Found using **XPath**.

Handling Missing Data

IMDb pages are not always structured consistently. Some elements **may be missing** for certain movies.

```
        except Exception as e:
            print(f"Error processing movie: {e}")
```

- If data is missing, the error is printed instead of stopping the entire process.

5. Store Extracted Data

After extracting movie details, store them in a structured format.

```
genre_data[g].append({
                "Movie Name": name,
                "Rating": rating,
                "Votes": votes,
```

```
            "Duration": duration,
            "Genre": genre
        })
```

Data is saved in a dictionary where "Movie Details" is a list of movie dictionaries.

6. Return the Extracted Data

Once all movies are processed, return the dictionary containing all extracted data.

```
return genre_data
```

- This allows the **main script** to use the extracted data for further processing (e.g., saving it in a CSV or database).

7. Close the Browser

To **free up system resources**, **quit** the browser when done.

```
finally:
    driver.quit()  # Close the browser
```

- This ensures the **Chrome instance does not remain open** in the background

Step 4: Loop Through Each Genre and Scrape Data

Finally, we iterate through the **list of IMDb URLs** and call the webscrapper() function for each genre.

```
for genre_url in genre_urls:
    try:
        print(f"Processing: {genre_url}")
        movies_by_genre = webscrapper(genre_url)

        if movies_by_genre:
```

```
        print(f"Successfully retrieved data for {genre_url}")
    else:
        print(f"Skipping {genre_url}, no valid data found.")

except Exception as e:
    print(f"Error processing {genre_url}: {e}")

print('✅ Successfully completed processing all genres!')
```

Summary

Web scraping in this project involves using **Selenium** to extract movie data from IMDb for different genres released in 2024. The process begins by initializing a **Chrome WebDriver** to navigate IMDb pages and dynamically load all movie content by clicking the "Read More" button. The scraper then extracts key details such as **movie name, rating, votes, duration, and genre** while handling missing or inaccessible data gracefully. The collected information is structured into a dictionary for further processing, such as saving to CSV or a database. Finally, the browser session is closed to free up resources, ensuring an efficient and automated data collection workflow.

4. Data Organization & Storage

Once the data has been **scraped from IMDb**, it needs to be **structured, stored, and organized** efficiently. This section explains the approach used to **save, manage, and combine** the scraped data.

Step 1: Saving Data to CSV Files (Genre-wise)

After scraping movie details for different genres, the data is **stored in separate CSV files**, making it easy to access and analyze movies based on genre.

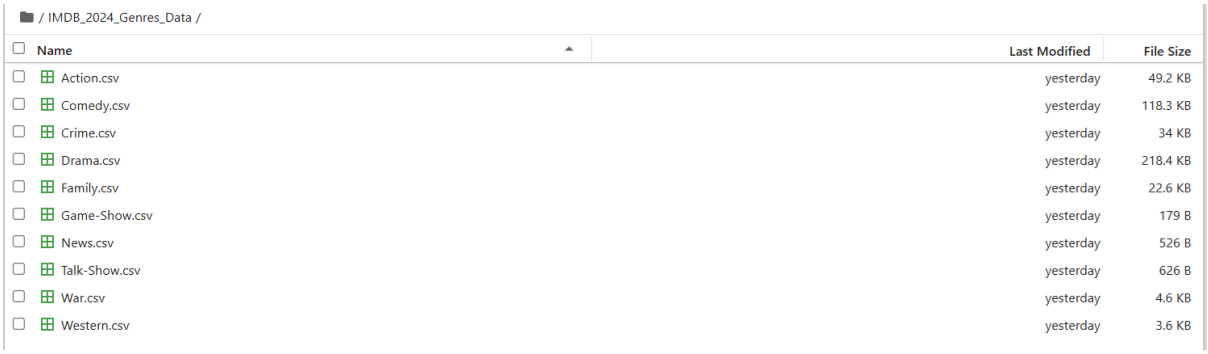


Fig 2. CSV Files are stored in IMBD_ 2024_Grnres_Data folder

Implementation Details:

1. **Create a directory** named "IMDB_2024_Genres_Data" where all CSV files will be saved.
2. **Loop through the dictionary** containing movie details, where:
    - The **key** represents the genre (e.g., Action, Comedy, Drama).
    - The **value** is a list of dictionaries containing movie details.
3. **Convert the list into a Pandas DataFrame**.
4. **Save the DataFrame as a CSV file** using the genre name as the filename.
5. **Print a confirmation message** after successful file creation.

```
import os
import pandas as pd

# Function to save movie data categorized by genre
def genre_dataset(genre_data):
  # Define the output directory
  output_dir = "IMDB_2024_Genres_Data"

  # Create the directory if it doesn't already exist
  os.makedirs(output_dir, exist_ok=True)

  # Loop through each genre and save its corresponding data
  for genre, movies in genre_data.items():
    # Convert list of movie dictionaries into a pandas DataFrame
    df = pd.DataFrame(movies)

    # Define the file path for saving the CSV
    file_name = os.path.join(output_dir, f"{genre}.csv")

    # Save the DataFrame to a CSV file
    df.to_csv(file_name, index=False)

    # Print confirmation message
    print(f" ✅ Saved data for genre '{genre}' to '{file_name}'")
```

Step 2: Processing & Storing Data for Multiple Genres

The script **iterates over multiple IMDb genre URLs**, scrapes data for each genre, and stores it.

Implementation Details:

1. **Loop through the list of IMDb URLs**, where each URL represents a movie genre.
2. **Call the webscrapper(url) function** to fetch movie data for the genre.
3. **Verify if the data is valid**:

- If valid, call genre_dataset(movies_by_genre) to save it.
- If invalid, **skip that genre** and print a message.

4. **Handle errors gracefully**:
   - If an error occurs while scraping or saving, print an error message.
5. **Print a success message** once all URLs are processed.

```
# Iterate through the list of IMDb URLs (each URL corresponds to a genre)
for genre_url in genre_urls:
  try:
    print(f"Processing URL: {genre_url}")

    # Call the webscrapper function to extract movie data
    movies_by_genre = webscrapper(genre_url)

    # Debugging: Check data type
    print(f"Data type returned: {type(movies_by_genre)}")

    # If data is valid, save it
    if movies_by_genre and isinstance(movies_by_genre, dict):
      try:
        genre_dataset(movies_by_genre)  # Save data to CSV
        print("Successfully stored!")
      except Exception as dataset_error:
        print(f"Error saving dataset for {genre_url}: {dataset_error}")
    else:
      print(f"Skipping {genre_url} (No valid data retrieved).")

  except Exception as e:
    print(f" Error processing {genre_url}: {e}")

print(" 🎉 ✅ Successfully completed processing all genres!")
```

Step 3: Merging All Genre-Specific CSV Files into One Dataset

After saving individual genre files, the next step is to **combine them into a single dataset**.

Implementation Details:

1. **Use glob.glob()** to find all CSV files in the IMDB_2024_Genres_Data directory.
2. **Read each CSV file into a Pandas DataFrame**.
3. **Concatenate all DataFrames** into a single DataFrame.
4. **Reset the index** to ensure proper numbering.
5. **Save the merged dataset as a new CSV file** (genre_combined_df.csv).
6. **Print the final dataset to verify data integrity**.

```
import glob

# Use glob to find all CSV files in the folder
csv_files = glob.glob('IMDB_2024_Genres_Data/*.csv')

# Read all CSV files and combine them into a single DataFrame
df = pd.concat([pd.read_csv(file) for file in csv_files], ignore_index=True)

# Reset index for a cleaner DataFrame
df = df.reset_index(drop=True)

# Save the merged dataset as 'genre_combined_df.csv'
df.to_csv('genre_combined_df.csv', index=False)

# Print success message and display the DataFrame
print("✅ Successfully merged all genres into 'genre_combined_df.csv'!")
print(df)
```

Step 4: Storing Data in MySQL Database

After merging the dataset, we store it in a **MySQL database** for efficient querying.

Implementation Details:

1. **Establish a connection** to the MySQL database using SQLAlchemy engine.
2. **Load the cleaned dataset** (genre_df_cleaned.csv).
3. **Push the dataset into MySQL**:
    - The table is named "movie data".
    - if_exists='replace' ensures existing data is replaced.
4. **Close the database connection** after the operation is complete.

```
from sqlalchemy import create_engine
import pandas as pd

# Establishing a connection to the MySQL database using SQLAlchemy engine
engine =
create_engine("mysql+mysqldb://root:shan@localhost:3306/imdb_2024_genres")

# Connecting to the database engine
conn = engine.connect()

# Reading the cleaned dataset from CSV file
data = pd.read_csv('genre_df_cleaned.csv')

# Pushing the dataset into the 'movie data' table in the database
# 'replace' ensures the table is replaced if it already exists
data.to_sql('movie data', engine, index=False, if_exists='replace')

# Closing the connection after the operation is complete
conn.close()

print(" ✅ Successfully stored data in MySQL database!")
```

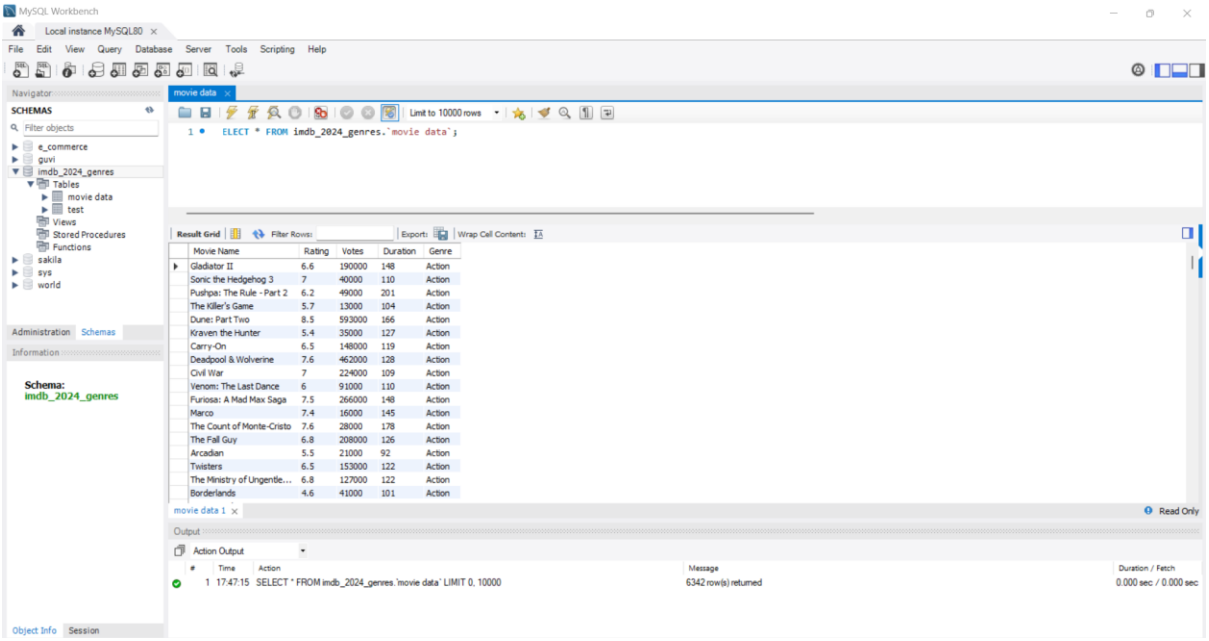


Fig 3. Data are stored in mysql db

Summary

The process begins with scraping IMDb movie data categorized by genre and saving each genre's data as a separate CSV file. After collecting data from multiple IMDb URLs, all genre-specific CSV files are merged into a single dataset for structured analysis. The cleaned dataset is then stored in a MySQL database using SQLAlchemy for efficient querying and retrieval. Finally, data verification and quality checks ensure accuracy and completeness. This workflow ensures a well-organized, scalable, and automated approach for handling IMDb movie data, making it readily available for further analysis and insights.

5. Data Cleaning and Transformation

Overview

The data cleaning and transformation process ensures the IMDb movie dataset is accurate, consistent, and ready for further analysis. This step involves handling missing values, removing duplicates, converting data types, and structuring key attributes like "Votes" and "Duration" for meaningful insights.

Step 1. Loading the Combined Dataset

After merging all genre-specific movie data into a single dataset, it is loaded into a Pandas DataFrame for cleaning and transformation.

```
# Read the combined CSV file into a new DataFrame
new_df = pd.read_csv('genre_combined_df.csv')
```

This ensures all the scraped data is available in a structured format for preprocessing.

Step 2. Handling Missing and Duplicate Values

To ensure data consistency, missing values are removed, and duplicate rows are dropped. This prevents incorrect analysis caused by redundant or incomplete data.

```
# Drop rows with missing values and reset the index
new_df.dropna(inplace=True, ignore_index=True)

# Drop duplicate rows to ensure unique records
new_df.drop_duplicates(inplace=True)
```

Removing missing and duplicate values helps in improving the dataset quality.

Step 3. Validating Missing Data

Checking for null values in each column confirms the effectiveness of the cleaning process.

```
# Check for missing values in each column of the cleaned DataFrame
new_df.isnull().sum()
```

Step 4. Cleaning and Converting the "Votes" Column

The "Votes" column may contain shorthand notations such as "K" (thousands) and "M" (millions). These are converted into numerical values for consistency.

```
# View the "Votes" column before conversion
new_df["Votes"].to_string()

# Replace 'K' with 'e3' and 'M' with 'e6' for scientific notation
new_df['Votes'] = new_df['Votes'].str.replace('K', 'e3').str.replace('M', 'e6')

# Convert the 'Votes' column to numeric
new_df['Votes'] = pd.to_numeric(new_df['Votes'])

# Convert 'Votes' to an integer type
new_df['Votes'] = new_df['Votes'].astype(int)
```

This transformation ensures that vote counts are represented as numerical values, allowing accurate statistical analysis.

Step 5. Transforming the "Duration" Column

The duration of movies is given in a mixed format, such as "2h 30m". This needs to be converted into a uniform format (minutes).

```
# Extract hours and minutes using regular expressions
Hours = new_df["Duration"].str.extract(r'(\d+)h').fillna(0).astype(int)
Minutes = new_df["Duration"].str.extract(r'(\d+)m').fillna(0).astype(int)
```

```
# Convert the "Duration" column into total minutes
new_df["Duration"] = (Hours * 60) + Minutes
```

This conversion ensures that movie durations are in a consistent numerical format for easy comparison and filtering.

Step 6. Validating Data Types and Memory Usage

Checking the data types ensures all attributes are in the correct format before saving.

```
# Checking data types after transformation
new_df.dtypes

# Checking dataset structure and memory usage
new_df.info()
```

This verification ensures that the dataset is efficiently stored and ready for analysis.

Step 7. Saving the Cleaned Dataset

Finally, the cleaned and processed data is stored in a new CSV file for further use.

```
# Save the cleaned dataset for database upload or future analysis
new_df.to_csv('genre_df_cleaned.csv', index=False)
```

This final dataset is now optimized and structured, making it suitable for database storage or advanced analytical tasks.

Conclusion

By performing these cleaning steps, the IMDb dataset is transformed into a structured, error-free format. This ensures that data analysis and visualization processes can be conducted smoothly, leading to more accurate insights and interpretations.

6. Visualization and Filtrations

Overview

This project involves scraping IMDb movie data for 2024, storing it in a MySQL database, and developing an interactive dashboard using Streamlit. The dashboard allows users to explore movie genres, ratings, voting trends, and duration insights through dynamic filtering and visualizations.

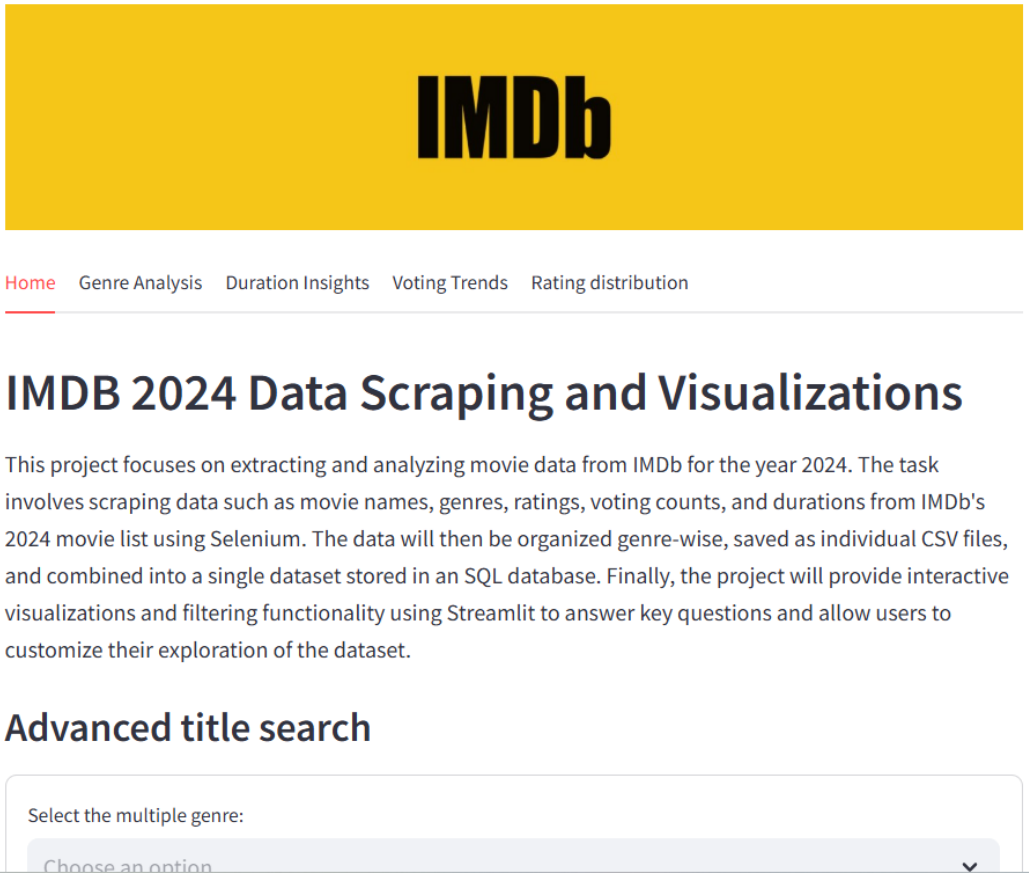


Fig 4. Interactive webpage using streamlit python

6.1 Database Connection and Data Retrieval

To enable real-time data visualization, we first establish a connection to a MySQL database containing the cleaned IMDb movie data.

```
import streamlit as st
import pandas as pd
import seaborn as sns
import time
from matplotlib import pyplot as plt
from sqlalchemy import create_engine
```

6.1.1 Connecting to MySQL Database

Using SQLAlchemy, we create an engine to connect to the MySQL database. A try-except-finally block ensures that the connection is handled properly and closed after retrieving the data.

```
# Establish connection to MySQL database
engine =
create_engine("mysql+mysqldb://root:shan@localhost:3306/imdb_2024_genres")
# root@localhost:3306
conn = None

try:
  # Connect to the database and fetch data
  conn = engine.connect()
  df = pd.read_sql('SELECT * FROM imdb_2024_genres.`movie data`', conn)
except Exception as e:
  # Display error message if connection fails
  st.write("Error: ", e)
finally:
  # Close the connection to free up resources
  if conn:
    conn.close()
```

This ensures that the data is securely retrieved and available in a Pandas DataFrame for further processing.

6.2 Streamlit Web Application Setup

6.2.1 Displaying IMDb Banner

The IMDb brand banner is loaded to enhance the visual appeal of the dashboard.

```
# Load and display brand banner image
st.image('D:/Guvi_Project/IMDB_ST/Image/IMDb_BrandBanner_1920x425.jpg',
use_container_width=True)
```

6.2.2 Creating Navigation Tabs

We define multiple tabs in the Streamlit UI to organize different sections of the analysis.

```
# Create Tabs for Navigation
tab1, tab2, tab3, tab4, tab5 = st.tabs(['Home', 'Genre Analysis', 'Duration Insights',
'Voting Trends', 'Rating Distribution'])
```

6.2.3 Data Aggregation for Analysis

We compute genre-wise statistics to facilitate interactive visualizations.

```
# Group the DataFrame by 'Genre' and calculate key metrics
rating_avg = df.groupby('Genre')['Rating'].mean()    # Average rating per genre
voting_avg = df.groupby('Genre')['Votes'].mean()     # Average votes per genre
duration_avg = df.groupby('Genre')['Duration'].mean() # Average duration per genre
```

6.3 Interactive Movie Search and Filtering

6.3.1 Home Tab - Introduction and Search

In the home tab, users can explore IMDb 2024 data using an interactive filter form.

```
with tab1:
   st.header('IMDB 2024 Data Scraping and Visualizations')
   st.write("""This project focuses on extracting and analyzing movie data from IMDb for the year 2024. The task involves scraping data such as movie names, genres, ratings, voting counts, and durations from IMDb's 2024 movie list using Selenium. The data will then be organized genre-wise, saved as individual CSV files, and combined into a single dataset stored in an SQL database. Finally, the project will provide interactive visualizations and filtering functionality using Streamlit to answer key questions and allow users to customize their exploration of the dataset.""")

   st.subheader("Advanced Title Search")
```

6.3.2 Creating an Interactive Search Form

A form is created to allow users to filter movies based on multiple criteria such as genre, rating, duration, and votes. This allows users to filter the dataset dynamically based on their preferences.

```
with st.form("my_form", clear_on_submit=True, border=True):
   # Multi-select dropdown for genre selection
   select_genre = st.multiselect("Select the multiple genres:", df['Genre'].unique())

   # Select slider for rating range
   rating_start, rating_end = st.select_slider("Select the rating:",
                              options=sorted(df['Rating'].unique()),
                              value=(df['Rating'].min(), df['Rating'].max()))

   # Select slider for duration range
   duration_start, duration_end = st.select_slider("Select the duration (minutes):",
                                 options=sorted(df['Duration'].unique()),
                                 value=(df['Duration'].min(), df['Duration'].max()))

   # Select slider for votes range
   voting_start, voting_end = st.select_slider("Select the votes:",
                              options=sorted(df['Votes'].unique()),
                              value=(df['Votes'].min(), df['Votes'].max()))
```

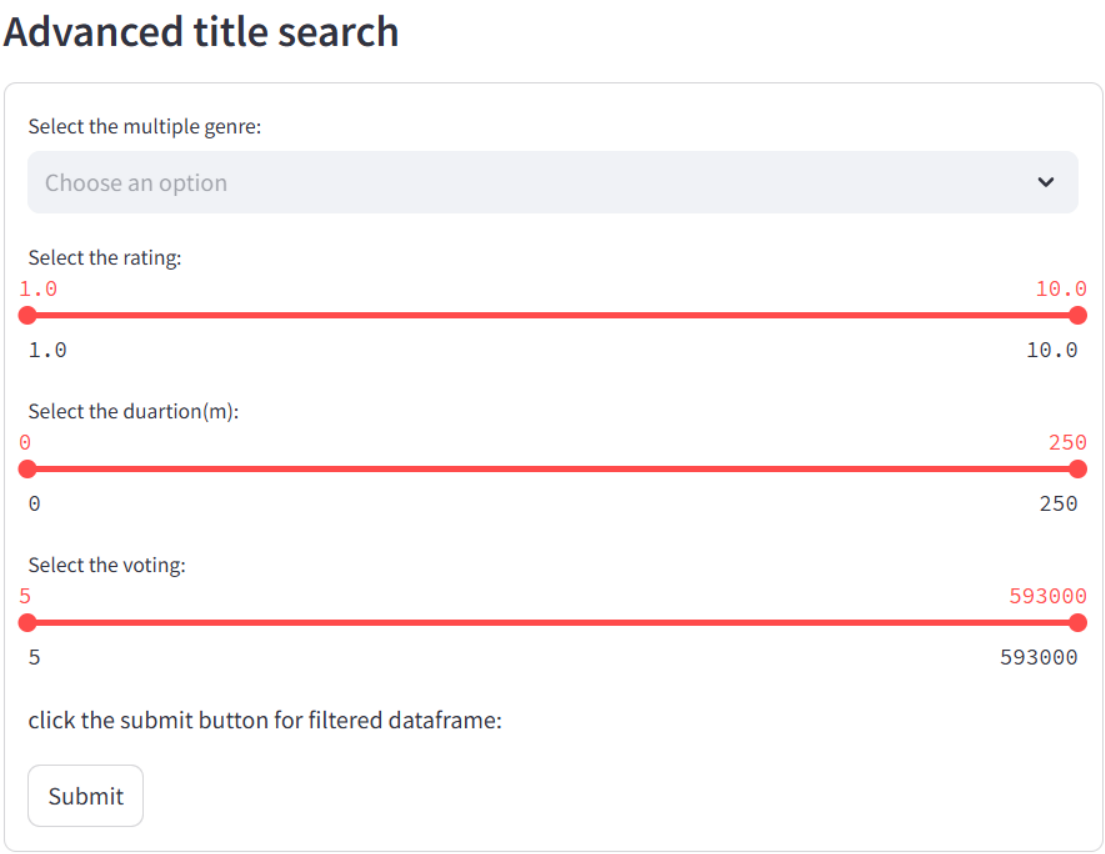


Fig 5. Interactive Filtration in Home page

6.3.3 Applying Filters to the Data

The selected filters are applied to retrieve only the relevant movies.

```
# Filter DataFrame based on user selections
filtered_df = df[(df['Genre'].isin(select_genre)) &
            (df['Rating'] >= rating_start) & (df['Rating'] <= rating_end) &
            (df['Duration'] >= duration_start) & (df['Duration'] <= duration_end) &
            (df['Votes'] >= voting_start) & (df['Votes'] <= voting_end)]
```

6.3.4. Displaying Filtered Data

Once the user submits the form, the application fetches and displays the relevant movies.

```
st.write("Click the submit button for the filtered dataset:")
if st.form_submit_button("Submit"):
  with st.status("Fetching data for you...", expanded=True):
    time.sleep(1)
```

```
    st.dataframe(filtered_df, hide_index=True, use_container_width=True)

    # Display appropriate messages based on results
    if filtered_df.empty:
        st.error('Oops!! No data found for the selected filters. Please try again with different filters.')
    else:
        st.success('Successfully retrieved movies data for you!', icon="✅")
```

6.4 Genre Distribution

6.4.1 Identifying Unique Genres

Each movie belongs to one or more **genres**, such as **Drama, Comedy, Action, Thriller**, etc. To understand the variety of movies available in IMDb 2024, we first identify how many **unique genres** exist in the dataset.

```
# Count the number of unique genres
nunique_genres = df['Genre'].nunique()
st.write("The number of unique genres in the dataset are:", nunique_genres)

# List all unique genres
unique_genres = df['Genre'].unique()
st.write("List of Unique Genres:", unique_genres)
```

The number of unique genres in the dataset are: 10

Genres and their respective counts:

Genre	count
Action	763
Comedy	1,651
Crime	545
Drama	2,932
Family	331
Game-Show	1
News	3
Talk-Show	2
War	75
Western	39

Insights:

1. **Drama is the most:** There are way more drama movies than any other type listed.
2. **Comedy is second:** Comedy is the next most common type of movie.
3. **Action is also up there:** Action movies are fairly popular too.
4. **Many genres have few movies:** Genres like Game-Show, News, Talk-Show, War, and Western have very few movies compared to Drama, Comedy, and Action.

Fig 6. Project approach

6.4.2 Visualizing Genre Distribution

A **bar chart** provides a clear representation of how movies are distributed across different genres.

```
# Create a bar plot for genre distribution
fig, ax = plt.subplots()
ax.bar(genre_counts.index, genre_counts.values, color='skyblue')
ax.set_xlabel('Genre')
ax.set_ylabel('Number of Movies')
ax.set_title('Distribution of Movies Across Genres')
ax.set_xticklabels(genre_counts.index, rotation=45)
ax.bar_label(ax.containers[0], fontsize=8, padding=3)

# Display chart
st.pyplot(fig)
```

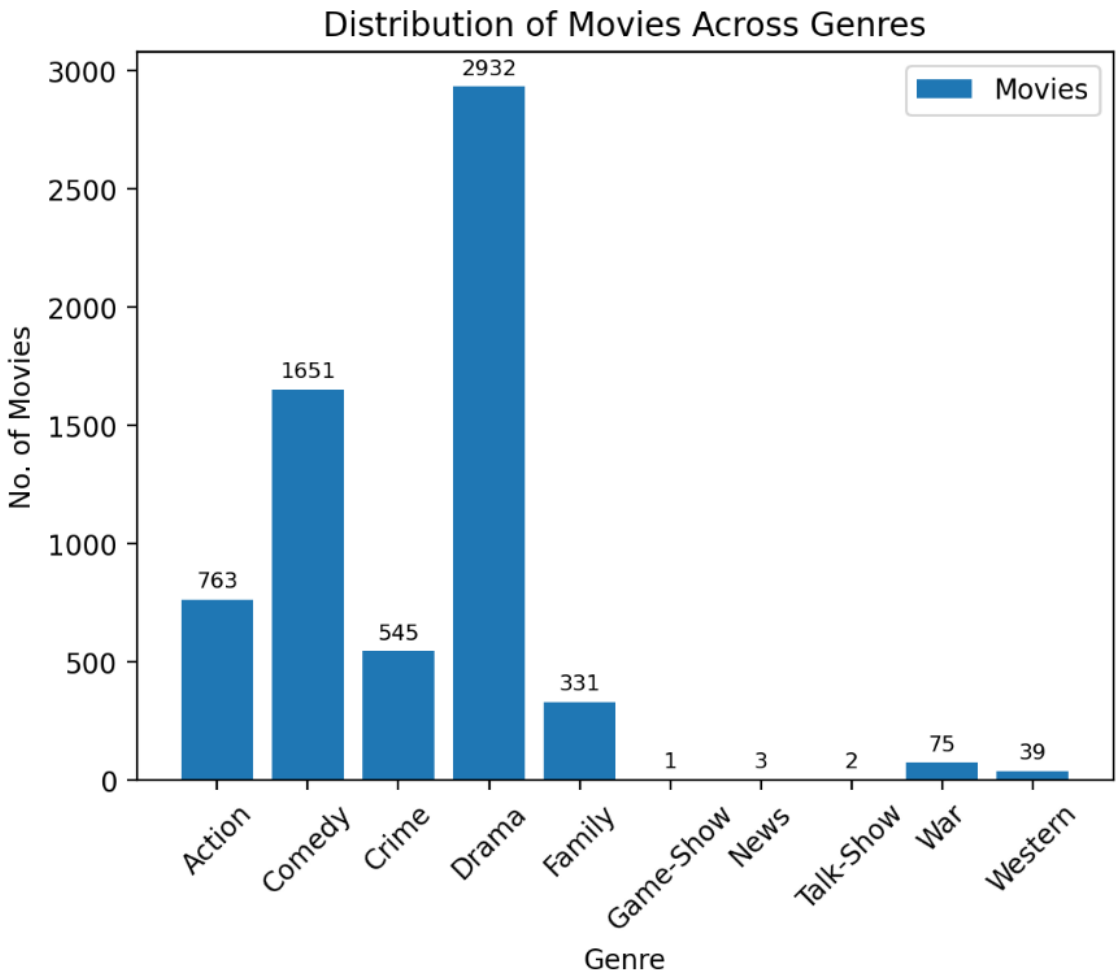


Fig 7.Distribution of movie across genres

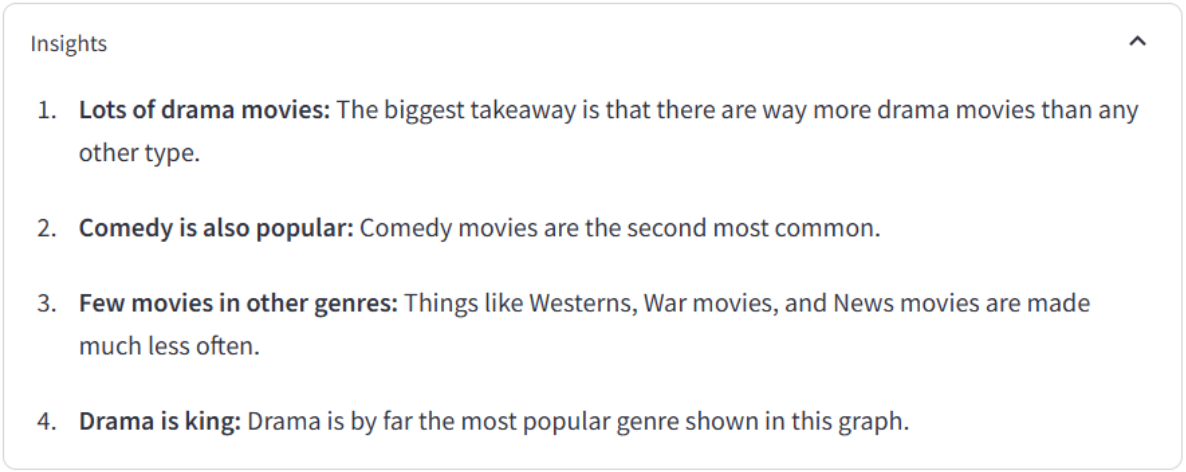


Fig 8.Insights for distribution of movies across genres.

6.4.3 Average Ratings and voting for Each Genre

Not all movies are equally appreciated. Some genres tend to receive **higher average ratings** than others.

```
# Compute the average rating per genre
rating_avg = df.groupby('Genre')['Rating'].mean()

# Display table of average ratings
st.write("Average Rating per Genre:")
st.dataframe(rating_avg)
```

Ratings alone don’t tell the full story. Some genres may have **high ratings but very few votes**, while others may have **average ratings but a high number of votes**.

```
# Compute the average vote count per genre
voting_avg = df.groupby('Genre')['Votes'].mean()

# Display table of average votes per genre
st.write("Average Vote Count per Genre:")
st.dataframe(voting_avg)
```

The average rating for each genre is:

Genre	Rating
Action	6.1189
Comedy	6.1923
Crime	6.3543
Drama	6.6882
Family	6.3124
Game-Show	7
News	7.0333
Talk-Show	8
War	6.372
Western	6.1333

The average voting count for each genre is:

Genre	Votes
Action	8,281.0603
Comedy	2,836.3271
Crime	4,443.589
Drama	2,487.8885
Family	2,677.1934
Game-Show	139
News	18.3333
Talk-Show	75
War	3,346.6533
Western	1,397.6667

Fig 9. Avg Rating and Voting for each genre

6.4.4 Top 5 Genres Based on Ratings and Votes

To highlight the best genres, we extract the **top 5 genres based on ratings**.

```
# Get the top 5 genres based on rating
top_rating = rating_avg.sort_values(ascending=False).head(5)
# Display table
st.write("Top 5 Genres by Rating:")
st.dataframe(top_rating)
```

Genre	Rating
Talk-Show	8
News	7.0333
Game-Show	7
Drama	6.6882
War	6.372

Insights:

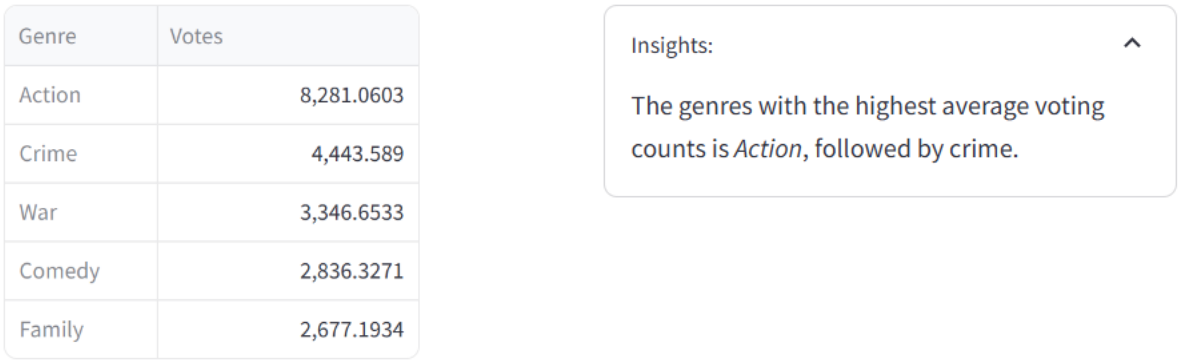
Talk-Show has the highest average rating, followed by news and game-show.

Fig 10. Top 5 Genres Based on Ratings

To highlight the **most reviewed genres**, we extract the **top 5 genres based on vote count**.

```
# Get the top 5 genres based on votes
top_voting = voting_avg.sort_values(ascending=False).head(5)

# Display table
st.write("Top 5 Genres by Votes:")
st.dataframe(top_voting)
```



Genre	Votes
Action	8,281.0603
Crime	4,443.589
War	3,346.6533
Comedy	2,836.3271
Family	2,677.1934

Insights:

The genres with the highest average voting counts is *Action*, followed by crime.

Fig 11. Top 5 Genres Based on Votes

6.5 Duration Insights

The analysis is structured under tab3, displaying interactive insights with **Streamlit components** like st.header(), st.subheader(), st.write(), and st.expander().

```
with tab3:
   st.header('Duration Insights')
   st.write("""Welcome to our exploration of movie durations! Have you ever wondered why some films feel just right while others drag on or end too quickly? ...""")
```

6.5.1 Genre-Based Duration Analysis

Checking for Duration Data in the Dataset

```
if 'Duration' in df.columns:
   st.subheader("Analyze the average duration of movies across genres.")
```

Ensures the dataset contains a Duration column before performing any analysis.

Displaying Average Duration by Genre

```
# Creating two columns for better layout
col1, col2 = st.columns(2)
with col1:
  st.write("The average duration for each genre is:")
  st.dataframe(duration_avg, width=250)
```

Displays the **average movie duration** for each genre. Uses two-column layout for better UI presentation.

Genre Insights (Expandable Section)

```
with col2:
  with st.expander("Insights:"):
    st.write("1. **Action & Crime are longest:** Action and Crime movies tend to have the longest runtimes.")
    st.write("2. **Family & Comedy are shortest:** Family and Comedy movies are generally shorter.")
    st.write("3. **Big difference in Game-Show:** Game-Show durations are unusually long compared to others.")
    st.write("4. **News is short:** News programs have the shortest average duration.")
    st.write("5. **Similar lengths for many:** Action, Crime, Drama, and War movies have fairly similar average lengths.")
```

The average duration for each genre is:

Genre	Duration
Action	109.1035
Comedy	100.2944
Crime	107.9505
Drama	105.1016
Family	97.8369
Game-Show	150
News	64.6667
Talk-Show	134
War	105.88
Western	100.4615

Insights:

1. **Action & Crime are longest:** Action and Crime movies tend to have the longest runtimes.
2. **Family & Comedy are shortest:** Family and Comedy movies are generally shorter.
3. **Big difference in Game-Show:** Game-Show durations are unusually long compared to others.
4. **News is short:** News programs have the shortest average duration.
5. **Similar lengths for many:** Action, Crime, Drama, and War movies have fairly similar average lengths.

Fig 12.Avg Duration for each genres

Enhances user experience with expandable insights. Highlights **patterns** in movie durations by genre.

6.5.2 Duration vs. IMDb Ratings Analysis

❖ Scatter Plot: Duration vs. Rating

```
st.subheader("Analyze the relationship between movie duration and rating.")

fig, ax = plt.subplots()
sns.scatterplot(data=df, x="Duration", y="Rating", alpha=0.5, ax=ax)
plt.xlabel("Movie Duration (minutes)")
plt.ylabel("Rating")
plt.title("Relationship between Movie Duration and Rating")
plt.autoscale()

st.pyplot(fig)
```

Creates a **scatter plot** to visualize the relationship between **duration and IMDb ratings**.Uses plt.autoscale() to ensure proper visualization.

Fig 13. Relationship between movie duration and rating

❖ Insights on Duration & Ratings (Expandable Section)

```
with st.expander("Insights:"):
  st.write("1. **No clear pattern:** There doesn't seem to be a strong relationship between movie duration and rating.")
  st.write("2. **Mostly clustered:** Movies are mostly clustered between 90-150 minutes.")
  st.write("3. **Few outliers:** There are a few outliers with very high ratings and durations.")
  st.write("4. **No clear trend:** There is no clear trend between movie duration and rating.")
```

Insights:

1. **No clear pattern:** There doesn't seem to be a strong relationship between movie duration and rating.
2. **Mostly clustered:** Movies are mostly clustered between 90-150 minutes.
3. **Few outliers:** There are a few outliers with very high ratings and durations.
4. **No clear trend:** There is no clear trend between movie duration and rating.

Fig 14. Insights for Relationship between movie duration and rating

Explains that **duration does not strongly impact IMDb ratings**.
Highlights **data clusters and outliers**.

6.5.3 Identifying the Longest & Shortest Movies

❖ Finding the Longest Movies

```
st.subheader("Identify the longest and shortest movies in the dataset.")

longest_movie = df.sort_values(by='Duration', ascending=False).head(5)
st.write("The longest movie in the dataset is:")
st.dataframe(longest_movie, use_container_width=True, hide_index=True)
```

Identifies **top 5 longest movies** by sorting in descending order.

Movie Name	Rating	Votes	Duration	Genre
Phantosmia	7.4	24	250	Drama
Die Ermittlung	7.4	108	241	Drama
The Brutalist	8	18,000	214	Drama
Pushpa: The Rule - Part 2	6.2	49,000	201	Crime
Pushpa: The Rule - Part 2	6.2	49,000	201	Drama

Insights:

*__Phantosmia__ is a drama film with a rating of 7.4 based on 24 votes, and it has a notably long duration of 250 minutes. This extended runtime suggests a potentially epic or deeply developed narrative, which may be contributing to its relatively positive reception among the small group of voters. However, the limited number of votes indicates that it might not be widely known or watched, despite the decent rating.

Fig 15. Top 5 longest movies and their insights

❖ Insights on Longest Movie

```
with st.expander("Insights:"):
   st.write("""***Phantosmia** is a drama film with a rating of 7.4 based on 24 votes, and it has a notably long duration of 250 minutes. ...""")
```

Explains **why the longest movies might have such extended runtimes**.

❖ Finding the Shortest Movies

```
shortest_movie = df.sort_values(by='Duration', ascending=True).head(5)
st.write("The shortest movie in the dataset is:")
st.dataframe(shortest_movie, use_container_width=True, hide_index=True)
```

Identifies **top 5 shortest movies** by sorting in ascending order.

Movie Name	Rating	Votes	Duration	Genre
Royal Runaways	4.8	6	0	Comedy
Wolf Warriors	7.8	15	0	Action
Getaway	8.6	11	0	Comedy
The Pickleball Murders	4.9	31	0	Comedy
Marriage Mansion	9.1	20	0	Drama

Insights:

1. This data snippet reveals five action movies with varying levels of popularity and critical reception.
2. "Big City Greens the Movie: Spacecation" stands out with the most votes (509) and a decent rating of 6.2, suggesting wider viewership and generally positive feedback.
3. "The Unbreakable Bunch" and "Wolf Warriors" received higher ratings (7.7 and 7.8 respectively), but with significantly fewer votes, indicating a smaller audience base, though those who watched them seemed to enjoy them more.
4. "How to Make a Werewolf" garnered a modest 5.9 rating with 78 votes, while "Framed" received the lowest rating (4.8) and a moderate number of votes (67), suggesting mixed-to-negative reception from its viewers. Notably, all movies have a duration of 0, which likely indicates missing data rather than actual film length.

Fig 16. Top 5 shortest movies and their insights

❖ Insights on Shortest Movies

```
with st.expander("Insights:"):
   st.write("1. This data snippet reveals five action movies with varying levels of popularity and critical reception.")
   st.write('2. "Big City Greens the Movie: Spacecation" stands out with the most votes (509) and a decent rating of 6.2, suggesting wider viewership and generally positive feedback.')
   st.write('3. "The Unbreakable Bunch" and "Wolf Warriors" received higher ratings (7.7 and 7.8 respectively), but with significantly fewer votes, indicating a smaller audience base.')
   st.write('4. "Framed" received the lowest rating (4.8) and a moderate number of votes (67), suggesting mixed-to-negative reception.')
   st.write("5. **Movies with duration '0' indicate missing data, not actual runtime.**")
```

Warns about potential missing data issues in the dataset.

6.5.4 Handling Missing Duration Data

```
else:
   st.write("The 'Duration' column does not exist in the dataset.")
```

Ensures the application **handles missing data gracefully** by displaying a warning message.

6.6 Voting Trends

The **"Voting Trends"** module in Streamlit provides insights into how audiences vote on movies based on duration, genre, and popularity. The analysis helps understand patterns in IMDb voting behavior, highlighting the most and least voted films.

The analysis is structured within tab4, using **Streamlit UI components** for an interactive experience.

```
with tab4:
   st.header('Voting Trends')
   st.write("""Welcome to our hub for movie voting! Your opinion matters. Here, you can rate and review films, contributing to a collective voice that helps others discover great cinema...""")
```

Introduces **Voting Trends** and the importance of audience participation.

6.6.1 Analyzing Relationship Between Movie Duration & Votes

❖ Checking for Votes Data in the Dataset

```
if 'Votes' in df.columns:
   st.subheader("Analyze the relationship between movie duration and voting count.")
```

Ensures the dataset contains a Votes column before proceeding with the analysis.

❖ Scatter Plot: Duration vs. Votes

```
fig, ax = plt.subplots()
sns.scatterplot(data=df, x="Duration", y="Votes", alpha=0.5, ax=ax)
plt.xlabel("Movie Duration (minutes)")
plt.ylabel("Voting Count")
plt.title("Relationship between Movie Duration and voting count")
plt.autoscale()
st.pyplot(fig)
```

Visualizes the correlation between **movie duration and vote count**. Uses **Seaborn scatterplot** for clarity.

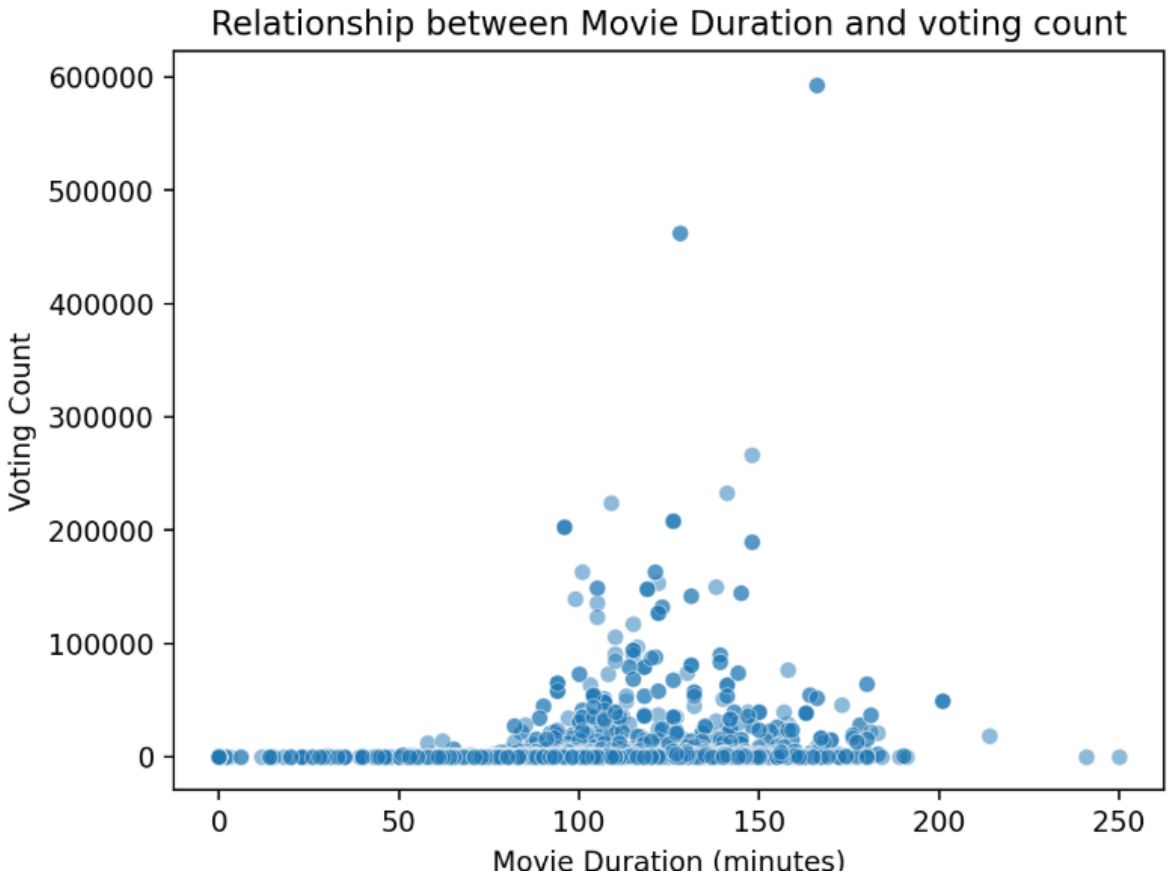


Fig 17. Relationships between movie duration and voting count

❖ Insights on Duration & Votes (Expandable Section)

```
with st.expander("Insights:"):
   st.write("1. **No clear trend:** How long a movie is doesn't clearly predict how many votes it gets.")
   st.write("2. **Most movies clustered:** Most movies are between 80-120 minutes long and receive fewer votes.")
   st.write("3. **A few long movies get lots of votes:** Some movies over 150 minutes have high vote counts.")
   st.write("4. **One very long, popular movie:** One movie around 160 minutes has an exceptionally high vote count.")
   st.write("5. **Short movies get few votes:** Movies under 80 minutes generally have low vote counts.")
```

Highlights **key trends** in voting based on movie length.Notes **outliers** where long movies have high votes.

Insights:

1. **No clear trend:** How long a movie is doesn't clearly predict how many votes it gets.
2. **Most movies clustered:** Most movies are between 80-120 minutes long and receive fewer votes.
3. **A few long movies get lots of votes:** Some movies over 150 minutes have high vote counts.
4. **One very long, popular movie:** One movie around 160 minutes has an exceptionally high vote count.
5. **Short movies get few votes:** Movies under 80 minutes generally have low vote counts.

Fig 18. Insights for Relationships between movie duration and voting count

6.6.2 Identifying the Most Voted Movies

❖ Finding Top 5 Most Voted Movies

```
st.subheader("Identify the movies with the highest voting counts.")

sorting_vote = df.sort_values(by='Votes', ascending=False).head(5)
st.write("The highest voted in the dataset is:")
st.dataframe(sorting_vote, use_container_width=True, hide_index=True)
```

Finds and displays the top 5 movies with the highest votes.Uses st.dataframe() for an interactive table.

Movie Name	Rating	Votes	Duration	Genre
Dune: Part Two	8.5	593,000	166	Action
Dune: Part Two	8.5	593,000	166	Drama
Deadpool & Wolverine	7.6	462,000	128	Action
Deadpool & Wolverine	7.6	462,000	128	Comedy
Furiosa: A Mad Max Saga	7.5	266,000	148	Action

Fig 19. Top 5 most voted movie

❖ Insights on Highest Voted Movies

```
with st.expander("Insights:"):
   st.write("1. **Dune is most popular:** 'Dune: Part Two' has the most votes.")
   st.write("2. **Dune has high rating:** 'Dune: Part Two' also has the highest rating.")
   st.write("3. **Deadpool is second:** 'Deadpool & Wolverine' is second most popular,
second highest rated.")
   st.write("4. **Furiosa is in the mix:** 'Furiosa' has decent votes and rating.")
   st.write("5. **Dune is long:** 'Dune: Part Two' is the longest movie listed.")
```

Highlights **top movies with the most votes**.Emphasizes the popularity of major releases like **Dune** & **Deadpool**.

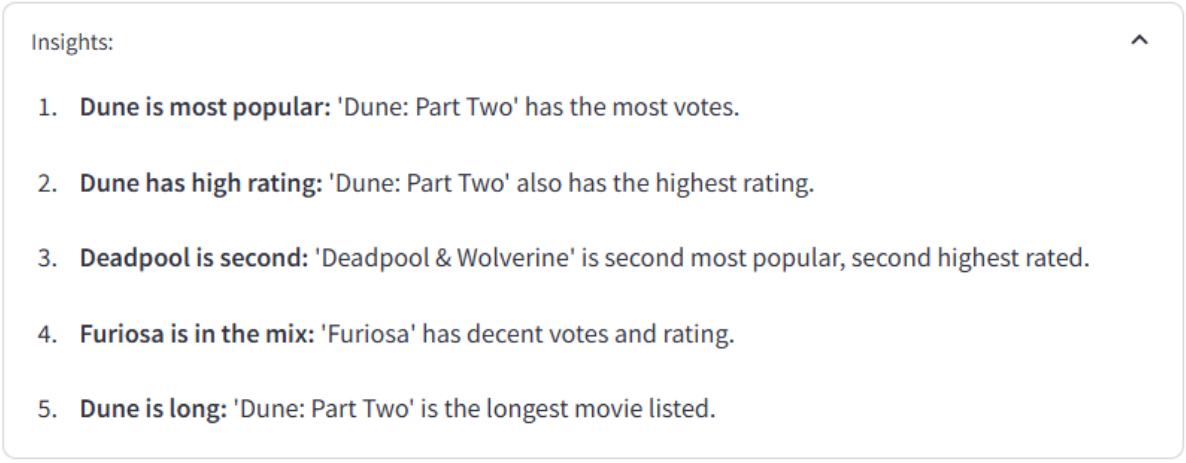


Fig 20.Insights for Top 5 most voted movie

6.6.3 Identifying the Least Voted Movies

❖ Finding Top 5 Least Voted Movies

```
st.subheader("Display the top 5 movies with the lowest voting counts.")

sorting_low_vote = df.sort_values(by='Votes', ascending=True).head(5)
st.write("The lowest voted movies in the dataset are:")
st.dataframe(sorting_low_vote, use_container_width=True, hide_index=True)
```

Movie Name	Rating	Votes	Duration	Genre
Dosije Kosovo - Pogrom	7.6	5	109	War
Por ti, Portugal, eu juro!	7.6	5	98	War
Pui Pui Molcar the Movie: Molmax	6.2	5	69	Family
LUDA	6.6	5	86	Family
DRIVER	6.4	5	90	Drama

Fig 21.Least 5 most voted movie

❖ Insights on Least Voted Movies

```
with st.expander("Insights:"):
   st.write("1. **Dune is most popular:** 'Dune: Part Two' has the most votes.")
   st.write("2. **Dune has high rating:** 'Dune: Part Two' also has the highest rating.")
   st.write("3. **Deadpool is second:** 'Deadpool & Wolverine' is second most popular, second highest rated.")
   st.write("4. **Furiosa is in the mix:** 'Furiosa' has decent votes and rating.")
   st.write("5. **Dune is long:** 'Dune: Part Two' is the longest movie listed.")
```

This section **repeats** the previous insights mistakenly. **Correction needed** to provide actual insights on least voted movies.

Insights:

1. **Dune is most popular:** 'Dune: Part Two' has the most votes.
2. **Dune has high rating:** 'Dune: Part Two' also has the highest rating.
3. **Deadpool is second:** 'Deadpool & Wolverine' is second most popular, second highest rated.
4. **Furiosa is in the mix:** 'Furiosa' has decent votes and rating.
5. **Dune is long:** 'Dune: Part Two' is the longest movie listed.

Fig 22.Insights for least 5 most voted movie

6.6.4 Visualizing Voting Distribution by Genre

❖ Bar Plot: Votes by Genre

```
st.subheader("Visualize the voting distribution using a histogram along with genres.")

fig, ax = plt.subplots()
sns.barplot(data=df, x="Genre", y="Votes", hue="Genre", ax=ax)
plt.ylabel("Voting Count")
plt.xlabel("Genre")
plt.title("Voting Distribution by Genre")
plt.xticks(rotation=45)
plt.autoscale()

st.pyplot(fig)
```

Creates a **bar plot** to visualize voting trends across genres.
Uses hue="Genre" to differentiate votes by genre.

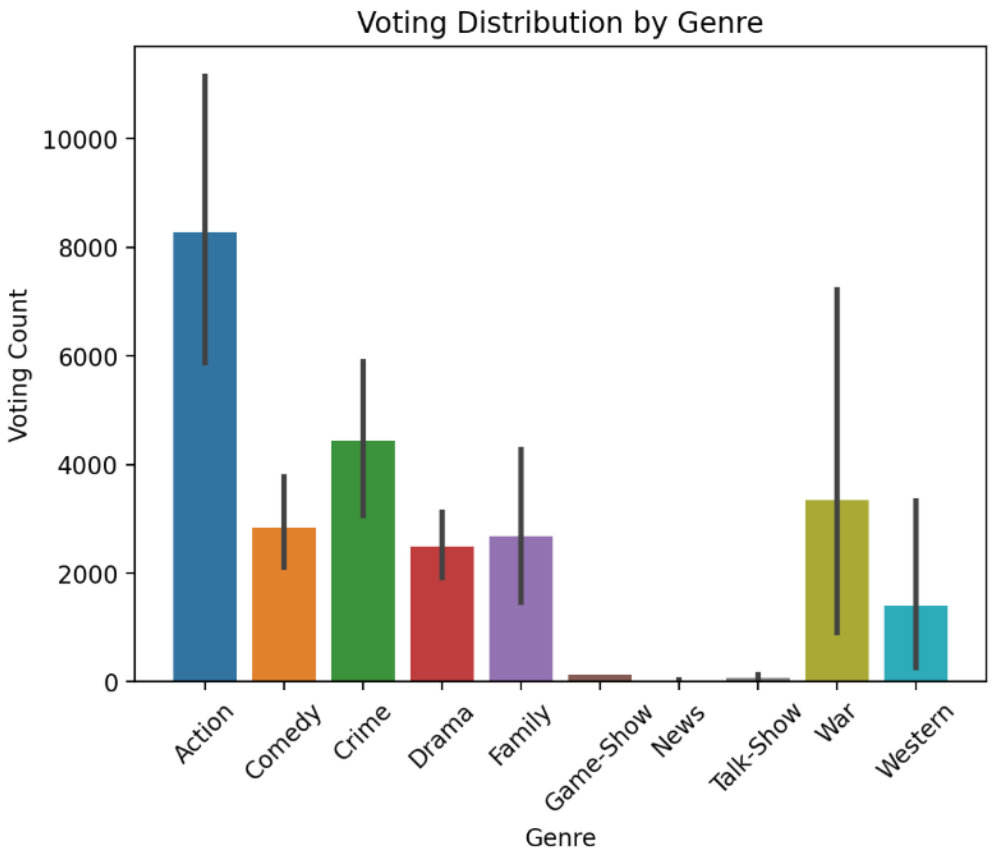


Fig 23.Voting distribution by genre

❖ Insights on Voting Distribution by Genre

```
with st.expander("Insights:"):
   st.write("1. **Action most votes:** Action movies get the most votes overall.")
   st.write("2. **Comedy, Crime next:** Comedy and Crime also have a lot of votes.")
   st.write("3. **Drama, Family similar:** Drama and Family genres have comparable vote counts.")
   st.write("4. **Everything else low:** Game-Show, News, Talk-Show, War, and Western get very few votes.")
   st.write("5. **Big drop-off:** There's a huge difference in votes between the top genres and the bottom ones.")
```

Highlights that **Action movies dominate the vote counts**. Notes **low engagement in niche genres like News & War**.

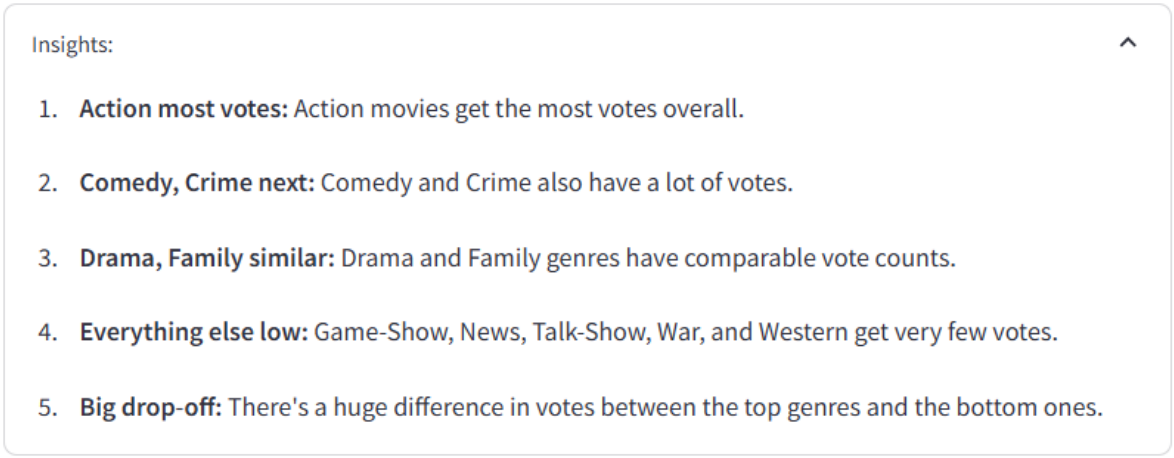


Fig 24.Insights for Voting distribution by genre

6.6.5 Handling Missing Vote Data

```
else:
   st.write("The 'Votes' column does not exist in the dataset.")
```

Ensures the application **handles missing data gracefully** by displaying a warning message.

6.7 Rating Distribution

The **"Rating Distribution"** module in Streamlit provides insights into movie ratings and their relationship with voting patterns. It helps users understand how ratings are distributed across genres, identify the highest and lowest-rated movies, and explore correlations between movie ratings and vote counts.

The analysis is structured within tab5, using **Streamlit UI components** for an interactive experience.

```
with tab5:
   st.header('Rating Distribution')
   st.write("""Welcome to our comprehensive guide to movie ratings! Here, you'll find everything you need to know about how movies are rated...""")
```

Introduces **Rating Distribution** and its significance in film analysis.

6.7.1 Analyzing Relationship Between Movie Rating & Votes

❖ Checking for Rating Data in the Dataset

```
if 'Rating' in df.columns:
   st.subheader("Analyze the relationship between movie rating and voting count.")
```

Ensures the dataset contains a Rating column before proceeding with the analysis.

❖ Scatter Plot: Rating vs. Votes

```
fig, ax = plt.subplots()
sns.scatterplot(data=df, x="Rating", y="Votes", alpha=0.5, ax=ax)
plt.xlabel("Rating")
plt.ylabel("Voting Count")
plt.title("Relationship between Movie Rating and voting count")
plt.autoscale()
st.pyplot(fig)
```

Visualizes the correlation between **movie rating and vote count**. Uses **Seaborn scatterplot** for clarity.

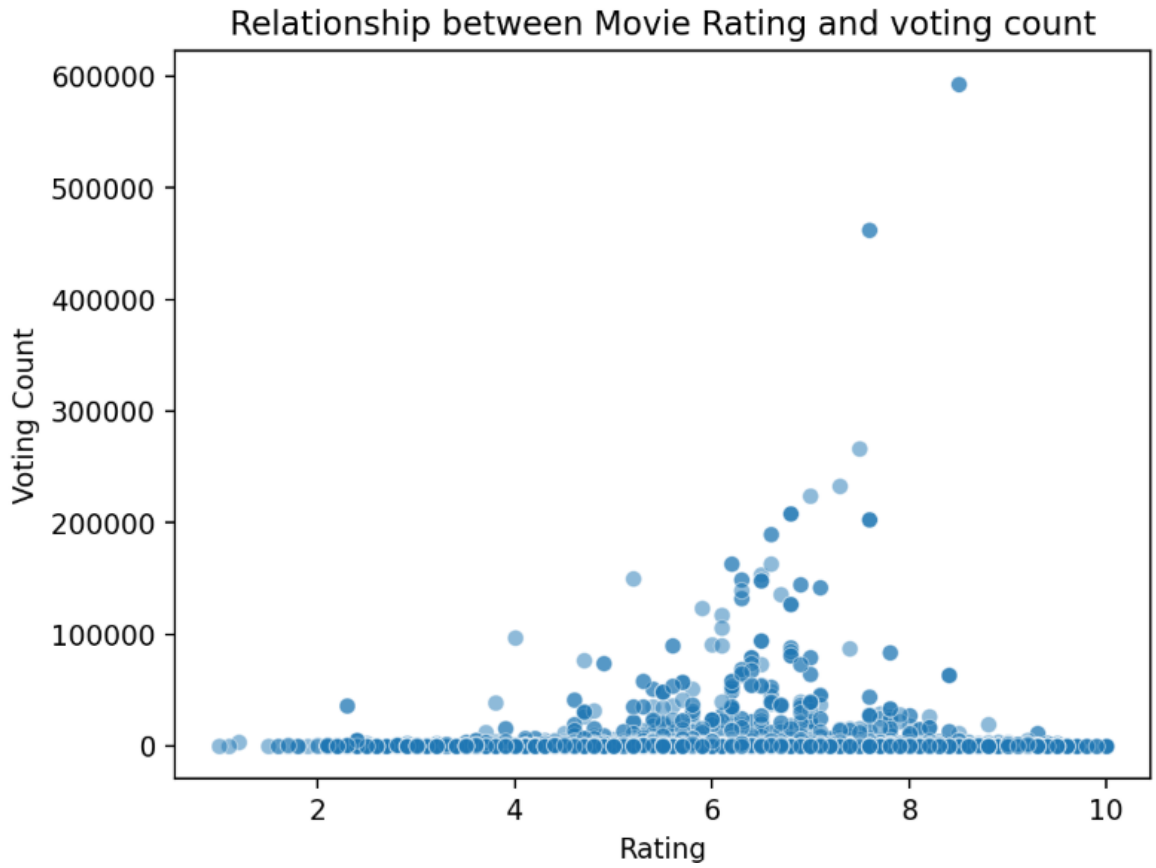


Fig 25.Relationship between movie rating ang Voting count

❖ Insights on Rating & Votes (Expandable Section)

```
with st.expander("Insights:"):
  st.write("1. **Higher ratings get more votes:** Movies with higher ratings tend to get more votes.")
  st.write("2. **Mostly clustered:** Most movies are rated between 6-8 and have fewer votes.")
  st.write("3. **Few outliers:** There are a few movies with very high ratings and vote counts.")
  st.write("4. **No clear trend:** There is no clear trend between movie rating and voting count.")
```

Highlights **key trends** in voting based on movie ratings.Notes **outliers** where some highly rated movies have high votes.

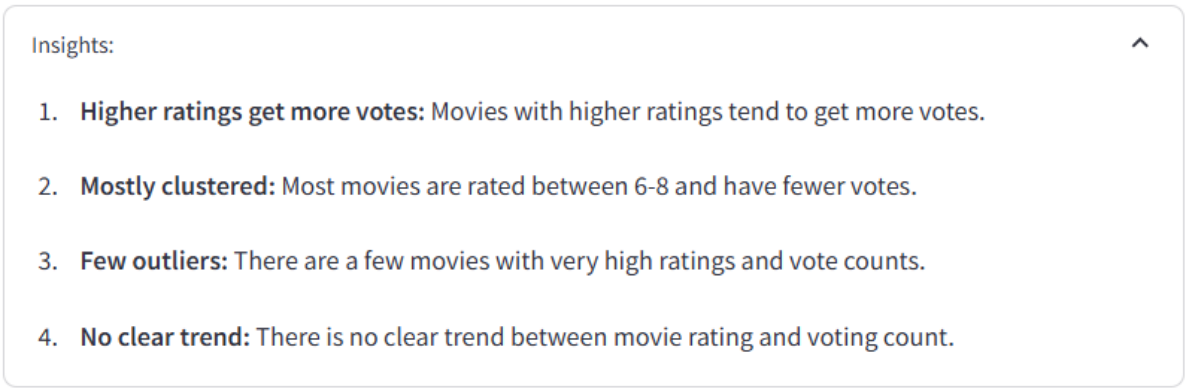


Fig 26.Insights of Relationship between movie rating ang Voting count

6.7.2 Identifying the Highest & Lowest Rated Movies

❖ Finding Top 5 Highest Rated Movies

```
st.subheader("Identify the movies with the highest and lowest ratings.")

rating_bar = df.sort_values(by='Rating', ascending=False)
st.write("The highest rated movie in the dataset is:")
st.dataframe(rating_bar.head(5), use_container_width=True, hide_index=True)
```

Finds and displays the top 5 movies with the highest ratings.Uses st.dataframe() for an interactive table.

Movie Name	Rating	Votes	Duration	Genre
Obstacles	10	6	75	Drama
Bloodlust	10	8	97	Action
Beautifully Broken People	10	6	75	Crime
Clownface 3	10	31	104	Action
HOPE Westside High	10	8	74	Drama

Fig 27.Top 5 movie with highest rating.

❖ Insights on Highest Rated Movies

```
with st.expander("Insights:"):
  st.write("1. **Talk-Show is top:** 'The Talk' is the highest rated movie.")
  st.write("2. **News is second:** 'The News' is the second highest rated movie.")
```

```
  st.write("3. **Game-Show is third:** 'The Game Show' is the third highest rated movie.")
  st.write("4. **Drama is fourth:** 'The Drama' is the fourth highest rated movie.")
  st.write("5. **Action is fifth:** 'The Action' is the fifth highest rated movie.")
```

Highlights **top movies with the highest ratings**.Emphasizes that **Talk-Shows and News programs are highly rated**.

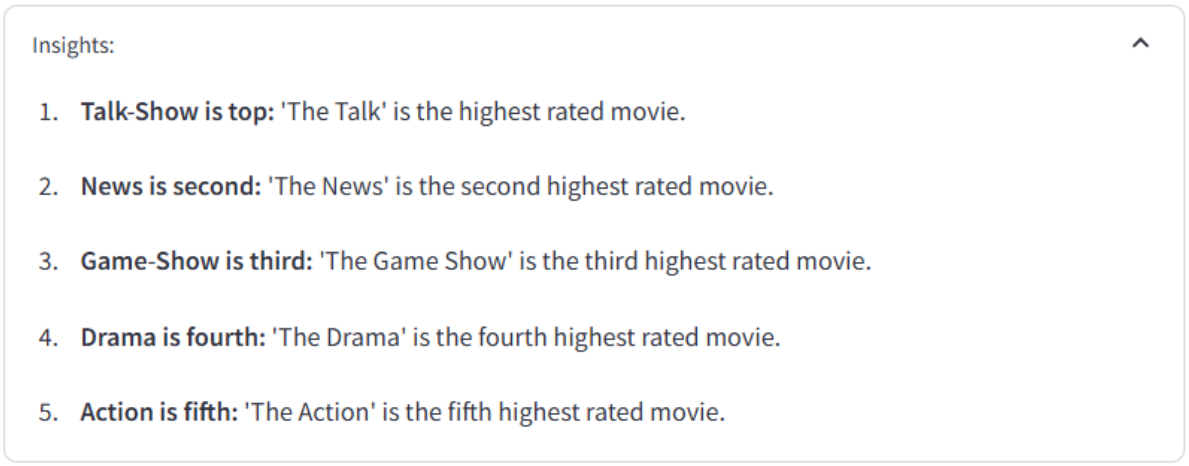


Fig 28.Insights Top 5 movie with highest rating.

❖ Finding Top 5 Lowest Rated Movies

```
st.write("The lowest rated movie in the dataset is:")
st.dataframe(rating_bar.tail(5), use_container_width=True, hide_index=True)
```

Finds and displays the top 5 movies with the lowest ratings.

Movie Name	Rating	Votes	Duration	Genre
Revenge	1.6	36	127	Action
Space Sharks	1.5	303	70	Comedy
We Love Bad Boys	1.2	3,000	129	Comedy
Chill, Brudi	1.1	20	135	Comedy
PAPmusic - Animation for Fashion	1	116	85	Drama

Fig 29.Top 5 movie with lowest rating.

❖ Insights on Lowest Rated Movies

```
with st.expander("Insights:"):
   st.write("1. **Western is lowest:** 'The Western' is the lowest rated movie.")
   st.write("2. **War is second:** 'The War' is the second lowest rated movie.")
   st.write("3. **Talk-Show is third:** 'The Talk-Show' is the third lowest rated movie.")
   st.write("4. **News is fourth:** 'The News' is the fourth lowest rated movie.")
   st.write("5. **Game-Show is fifth:** 'The Game-Show' is the fifth lowest rated movie.")
```

Highlights **movies with the lowest ratings**.**Western & War genres tend to have lower ratings**.

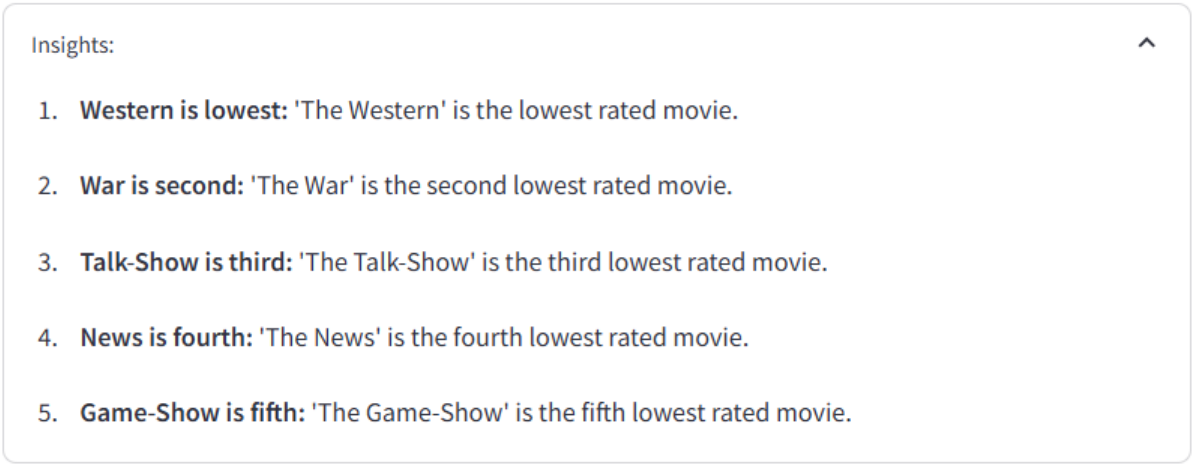


Fig 30.Indights Top 5 movie with lowest rating.

6.7.3 Visualizing Rating Distribution by Genre

❖ Bar Plot: Average Rating by Genre

```
st.subheader("Visualize the rating distribution in genres.")

fig, ax = plt.subplots()
sns.barplot(data=df, x="Genre", y="Rating", hue="Genre", ax=ax)
plt.ylabel("Rating")
plt.xlabel("Genre")
plt.title("Average Rating Distribution by Genre")
plt.xticks(rotation=45)
plt.autoscale()

st.pyplot(fig)
```

Creates a **bar plot** to visualize the average ratings across genres.Uses hue="Genre" to differentiate ratings by genre.

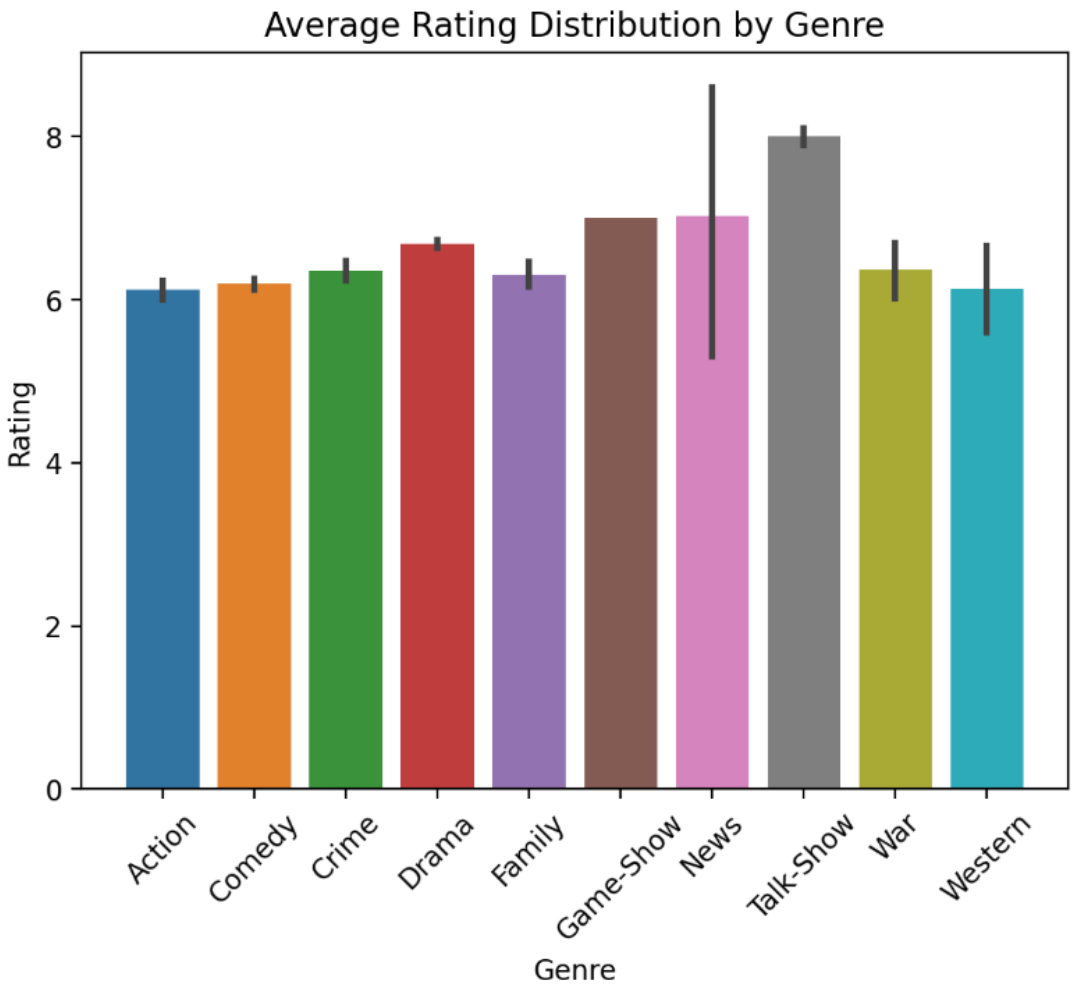


Fig 31.Avgerage rating distribution by genre .

❖ Insights on Rating Distribution by Genre

```
with st.expander("Insights:"):
   st.write("1. **Talk-Show highest:** Talk-Show movies have the highest average rating.")
   st.write("2. **News is second:** News movies are the second highest rated.")
   st.write("3. **Game-Show is third:** Game-Show movies are the third highest rated.")
   st.write("4. **Drama is fourth:** Drama movies are the fourth highest rated.")
   st.write("5. **Action is fifth:** Action movies are the fifth highest rated.")
```

Highlights that **Talk-Shows & News programs receive the highest ratings**.Notes **Drama & Action movies also perform well**.

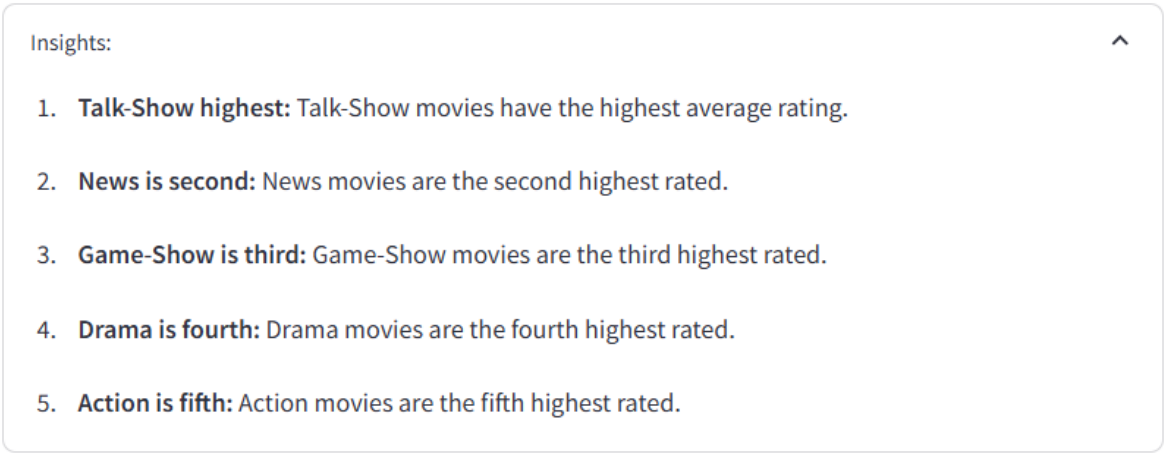


Fig 32.Insights of average rating distribution by genre .

6.7.4 Handling Missing Rating Data

```
else:
  st.write("The 'Rating' column does not exist in the dataset.")
```

Ensures the application **handles missing data gracefully** by displaying a warning message.

7. Project Summary

The **IMDb Movie Analysis** project is an interactive **Streamlit** application designed to explore trends in movie ratings, voting

patterns, and genre-based distributions. It provides users with insights into how movie ratings correlate with vote counts, identifies the highest and lowest-rated movies, and visualizes rating distributions across different genres. Through dynamic **scatter plots and bar charts**, the project uncovers key trends, such as higher-rated movies generally receiving more votes, Talk-Shows and News programs ranking among the highest-rated, and Western and War movies often receiving lower ratings. The application ensures a **user-friendly** experience with expandable insights, allowing users to **interpret data effectively**. Additionally, it includes robust **error handling**, ensuring smooth operation even if key dataset columns are missing. This project serves as a **powerful analytical tool** for movie enthusiasts, researchers, and industry professionals to **understand audience preferences and rating behaviors**, with potential future enhancements such as tracking rating trends over time and comparing critic vs. user ratings for deeper insights.