POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Systems and Methods for Big and Unstructured Data Project

Author(s): **Federica Topazio 10985760**

Academic Year: 2023-2024

Contents

Contents **i**

Introduction **1**

1 Data Wrangling **3**
1.1 Data Loading . . . 3
1.2 Data Merging . . . 4
1.3 Data Transformation . . . 4
1.4 Normalization . . . 5
1.5 Feature Extraction . . . 5
1.6 Data Export . . . 5

2 Dataset **7**
2.1 Structure . . . 7
2.2 Non-Relational Schema . . . 8
2.3 Further Choices . . . 8

3 Queries **9**
3.1 Average Rating per Genre . . . 9
3.1.1 Query 1 . . . 9
3.1.2 Outcome Table (possibly partial) . . . 10
3.2 User Rating Patterns . . . 11
3.2.1 Query 2 . . . 11
3.2.2 Outcome Table (possibly partial) . . . 12
3.3 Most Polarizing Movies . . . 13
3.3.1 Query 3 . . . 13
3.3.2 Outcome Table (possibly partial) . . . 14
3.4 Temporal Trends in Ratings . . . 15
3.4.1 Query 4 . . . 15

3.4.2 Outcome Table (possibly partial) 16
3.5 Genre Popularity Over Time 17
3.5.1 Query 5 17
3.5.2 Outcome Table (possibly partial) 18
3.6 Top Rated Movies by Year 19
3.6.1 Query 6 19
3.6.2 Outcome Table (possibly partial) 19
3.7 User Engagement Over Time 20
3.7.1 Query 7 20
3.7.2 Outcome Table (possibly partial) 21
3.8 Genre Popularity Evolution 22
3.8.1 Query 8 22
3.8.2 Outcome Table (possibly partial) 22
3.9 Time-Based Rating Patterns 23
3.9.1 Query 9 23
3.9.2 Outcome Table (possibly partial) 24
3.10 Correlation Between Rating Count and Average Rating 25
3.10.1 Query 10 25
3.10.2 Outcome Table (possibly partial) 25

4 Data Analysis and Visualization with Python 27
4.1 Average Rating per Genre 28
4.1.1 Python Code 1 28
4.1.2 Data Visualisation 1 29
4.2 User Rating Patterns 30
4.2.1 Python Code 2 30
4.2.2 Data Visualisation 2 31
4.3 Most Polarizing Movies 32
4.3.1 Python Code 3 32
4.3.2 Data Visualisation 3 34
4.4 Temporal Trends in Rating 35
4.4.1 Python Code 4 35
4.4.2 Data Visualisation 4 36
4.5 Top Rated Movies by Year 37
4.5.1 Python Code 5 37
4.5.2 Data Visualisation 5 38
4.6 Most Polarizing Movies 39

4.6.1 Python Code 6 39
4.6.2 Data Visualisation 6 40
4.7 User Engagement Over Time 41
4.7.1 Python Code 7 41
4.7.2 Data Visualisation 7 42
4.8 Genre Popularity Evolution 43
4.8.1 Python Code 8 43
4.8.2 Data Visualisation 8 45
4.9 Time-Based Rating Patterns 46
4.9.1 Python Code 9 46
4.9.2 Data Visualisation 9 47
4.10 Correlation Between Rating Count and Average Rating 48
4.10.1 Python Code 10 48
4.10.2 Data Visualisation 10 49

List of Figures **51**

Introduction

The aim of the SMBUD Project is performing data analysis on the MovieLens dataset, as a mean to gain an understanding of movies popularity, and their genre over time using the MovieLens dataset, which includes ratings and movie information, using MongoDB as the database technology.

The overarching goal is to understand various aspects of movie ratings and preferences. This includes analyzing genre popularity, user engagement, trends over time, and the relationship between the number of ratings and average ratings.

MongoDB is ideal for handling semi-structured data like movie ratings, which can vary in fields (like genres being a list). MongoDB's flexible schema accommodates the varied structure of data, MongoDB's powerful aggregation framework allows for complex queries, such as grouping, sorting, and calculating aggregates, which are essential for the type of analyses listed.

Another major factor that contributed to this choice is the scalability offered by MongoDB with large datasets, ensuring efficient processing of queries even as the dataset grows. Furthermore, data retrieval is achieved with high efficiency for tasks like identifying trends and patterns, due to MongoDB's indexing and query optimization.

1 | Data Wrangling

Data wrangling is a critical step in the data analysis process. This section outlines the data wrangling procedures applied to the *MovieLens dataset*, comprising steps such as data loading, merging, transforming, and normalization. These steps are fundamental in preparing the data for subsequent analysis, ensuring accuracy, and facilitating ease of use.

The dataset in question comprises several files related to movies, including information about movie titles, genres, user ratings, and tags. The data, sourced from different CSV files, required preprocessing to be suitable for analysis. All the data wrangling operations are in the *"dataWrangling.ipynb"* file.

1.1. Data Loading

The initial step involved loading multiple CSV files (movies.csv, ratings.csv, and others) into pandas DataFrames. This step was essential to facilitate subsequent data manipulation and analysis.

```
import pandas as pd
import numpy as np
# File paths
movies_file_path = 'movies.csv'
links_file_path = 'links.csv'
ratings_file_path = 'ratings.csv'
tags_file_path = 'tags.csv'

# Loading each file into a DataFrame
movies_df = pd.read_csv(movies_file_path)
links_df = pd.read_csv(links_file_path)
ratings_df = pd.read_csv(ratings_file_path)
tags_df = pd.read_csv(tags_file_path)

```

Listing 1.1: Data Loading

1.2. Data Merging

The ratings DataFrame, containing user ratings for movies, was merged with the movies DataFrame, which included details like movie titles and genres. This merge operation was based on the movieId column, present in both DataFrames. The result was a comprehensive dataset that combined detailed movie information with user ratings, to facilitate queries using MongoDB, since this kind of operation would have been executed for every query perfomed on the dataset.

```
    # merge the ratings and movies dataframes
    merged_df = pd.merge(ratings_df, movies_df, on='movieId')

```

Listing 1.2: Data Merging

1.3. Data Transformation

The genres column, originally a string of pipe-separated genres, was transformed into an array. This transformation improved the data structure, making it more suitable for analysis involving movie genres. Additionally, UNIX timestamps were converted into a human-readable date format. This conversion made the dataset more interpretable and laid the groundwork for any direct temporal analysis.

```
# convert genre in array
merged_df['genres'] = merged_df['genres'].apply(lambda x: x.split('|'))
    import datetime

# Convert the timestamp from UNIX time to a human-readable date format
merged_df['timestamp'] = pd.to_datetime(merged_df['timestamp'], unit='s')

```

Listing 1.3: Data Transformation

1.4. Normalization

Movie ratings were normalized on a scale from 0 to 1. This step is crucial in standardizing the data, especially useful when dealing with data from various sources or scales, and for future expansions of the dataset from possibly different sources.

```
# Normalizing the ratings on a scale from 0 to 1
merged_df['rating'] = merged_df['rating'] / 5.0

```

Listing 1.4: Normalization

1.5. Feature Extraction

The release year was extracted from the movie title and stored in a new column. This extraction not only cleaned the title column but also added a valuable feature for analysis, which will be used to draw useful insights on the data.

```
merged_df['release'] = merged_df['title'].str.extract(r'\((\d{4})\)',
    expand=False)
merged_df['title'] = merged_df['title'].str.replace(r' \(\d{4}\)', '',
    regex=True)


```

Listing 1.5: Feature Extraction

1.6. Data Export

The final, cleaned DataFrame was saved as a JSON file.

```
import json
merged_df_json_compatible = merged_df.where(pd.notnull(merged_df), None)

json_data = merged_df_json_compatible.to_dict(orient='records')

with open('merged_movies_ratings.json', 'w') as json_file:
json.dump(json_data, json_file, default=str)

```

Listing 1.6: Data Export

2 Dataset

The MovieLens Dataset describes 5-star rating and free-text tagging activity from [MovieLens](http://movielens.org), a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. These data were created by 610 users between March 29, 1996 and September 24, 2018. This dataset was generated on September 26, 2018. Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars). Hence, it has been decided to normalize this field, as well as convert the time stamp in a human-readable way (in the following format YYYY-MM-DD HH:MM:SS)., to make the data more intuitive and useful for any time-based analysis or filtering that might be required in the recommendation system. Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

2.1. Structure

This dataset is composed of 4 collections(‘links.csv‘, ‘movies.csv‘, ‘ratings.csv‘ and ‘tags.csv‘), of which only two of them will be taken into consideration. These two main entities are movies and ratings.

- **Movies Dataset (movies.csv):**
  - **movieId (int64):** Unique identifier for each movie.
  - **title (string):** Movie title, with release year.
  - **genres (string):** List of genres, in pipe-separated format.
- **Ratings Dataset (ratings.csv):**
  - **userId (int64):** Identifier for users who rated movies.
  - **movieId (int64):** References movieId in movies dataset.
  - **rating (float64):** Rating given by the user.
  - **timestamp (int64):** Time of rating in UNIX timestamp.

2.2. Non-Relational Schema

The Non-Relational Schema for MovieLens Dataset, after the operations presented in Chapter 1 have been executed, is as follows:

```
{
  "movies_ratings_collection": {
    "_id": "ObjectId (MongoDB unique identifier)",
    "userId": "int",
    "movieId": "int",
    "rating": "float",
    "timestamp": "datetime",
    "title": "string",
    "genres": "[string]",
     "release": "string"
  }
}
```

Listing 2.1: Non-Relational Schema

2.3. Further Choices

To leverage the features of MongoDB, especially for optimizing query performance, creating indices on frequently queried fields is a common practice. In the context of the **movies_ratings** collection in MongoDB, two indices have been defined:

```
db.movies_ratings.createIndex({ userId: 1 })
db.movies_ratings.createIndex({ movieId: 1 })
```

Listing 2.2: Indices

3 | Queries

In this chapter will be enlisted and explained all the queries performed on the dataset.

3.1. Average Rating per Genre

Calculate the average rating for each genre across all movies, to understand the highly rated genres within the sampled users.

3.1.1. Query 1

```
db.movies_ratings.aggregate([
    {
        $unwind: "$genres"
    },
    {
        $group: {
            _id: "$genres",
            averageRating: { $avg: "$rating" }
        }
    },
    {
        $sort: { averageRating: -1 }
    }
    ])

```

3.1.2. Outcome Table (possibly partial)

	_id	averageRating
0	Film-Noir	0.784023
1	War	0.761659
2	Documentary	0.759557
3	Crime	0.731659
4	Drama	0.731237
5	Mystery	0.726492
6	Animation	0.725987
7	IMAX	0.723667
8	Western	0.716788
9	Musical	0.712736
10	Adventure	0.701722
11	Romance	0.701302
12	Thriller	0.698741
13	Fantasy	0.698200
14	(no genres listed)	0.697872
15	Sci-Fi	0.691144
16	Action	0.689597
17	Children	0.682591
18	Comedy	0.676944
19	Horror	0.651639

Figure 3.1: Outcome Query 1

3.2. User Rating Patterns

Find users who have rated more than 300 movies, and calculate their average rating. The threshold has been selected as the half plus the 80%. The aim is to understand the behavior of active users in the dataset, and provide the most seen genre by them.

3.2.1. Query 2

```
db.movies_ratings.aggregate([
    { $unwind: "$genres" },
    {
        $group: {
            _id: { userId: "$userId", genre: "$genres" },
            totalRatingsPerGenre: { $sum: 1 },
            averageRatingPerGenre: { $avg: "$rating" }
        }
    },
    {
        $sort: { "_id.userId": 1, totalRatingsPerGenre: -1 }
    },
    {
        $group: {
            _id: "$_id.userId",
            totalRatings: { $sum: "$totalRatingsPerGenre" },
            averageRating: { $avg: "$averageRatingPerGenre"
   },
            mostRatedGenre: { $first: "$_id.genre" }
        }
    },
    { $match: { totalRatings: { $gt: 300 } } },
    {$sort : {totalRatings : -1}}
])

```

3.2.2. Outcome Table (possibly partial)

	_id	totalRatings	averageRating	mostRatedGenre
0	414	6616	0.707182	Drama
1	599	6266	0.553527	Drama
2	474	4739	0.687937	Drama
3	448	4702	0.584474	Comedy
4	380	3761	0.757581	Action

Figure 3.2: Outcome Query 2

3.3. Most Polarizing Movies

Identify movies with the greatest variance in ratings. The variance is calculated as

$$\sigma^2 = \frac{1}{N-1}\left(\frac{\sum_{i=1}^{N} x_i^2}{N} - (\bar{x})^2\right)$$

3.3.1. Query 3

```
db.movies_ratings.aggregate([
    {
        $group: {
            _id: { movieId: "$movieId", title: "$title" },
            averageRating: { $avg: "$rating" },
            count: { $sum: 1 },
            sumOfSquares: { $sum: { $pow: ["$rating", 2] } }
        }
    },
    {
        $project: {
            title: "$_id.title",
            variance: {
                $cond: [
                    { $gt: ["$count", 1] },
                    // Check if count is greater than 1
                    {
                        $divide: [
                            {
                                $subtract: [
                                    {$divide:
                                    ["$sumOfSquares",
                                    "$count"] },
                                    {$pow: ["$averageRating",
    2]}
                                ]
                            },
                            { $subtract: ["$count", 1] }
                        ]
```

```
                    },
                    0

                // Variance is zero if there is only one rating
                ]
            }
        }
    },
    { $sort: { variance: -1 } }

])
```

3.3.2. Outcome Table (possibly partial)

	_id	averageRating	variance	movie
0	{'movieId': 2068, 'movie': 'Fanny and Alexande...	0.55	0.2025	Fanny and Alexander (Fanny och Alexander)
1	{'movieId': 32892, 'movie': 'Ivan's Childhood ...	0.55	0.2025	Ivan's Childhood (a.k.a. My Name is Ivan) (Iva...
2	{'movieId': 70946, 'movie': 'Troll 2'}	0.70	0.1800	Troll 2
3	{'movieId': 3223, 'movie': 'Zed & Two Noughts,...	0.60	0.1600	Zed & Two Noughts, A
4	{'movieId': 484, 'movie': 'Lassie'}	0.60	0.1600	Lassie

Figure 3.3: Outcome Query 3

3.4. Temporal Trends in Ratings

Analyze how average movie ratings have changed over years for the *Toy Story* movie. Understand how the users' behaviour has changed over time for a specific movie.

3.4.1. Query 4

```
db.movies_ratings.aggregate([
    {
        $match: { title: "Toy Story" }
        // Filter for a specific movie
    },
    {
        $project: {
            year: { $year: { $toDate: "$timestamp" } },
            rating: 1, title: 1
        }
    },
    {
        $group: {
            _id: { year: "$year", title : "$title"},
            // Group by year
            averageRating: { $avg: "$rating" },
            numRatings: {$sum: 1}
            // Calculate the average rating
        }
    },
    { $sort: { "averageRating": 1 } }
    // Sort results chronologically
])
```

3.4.2. Outcome Table (possibly partial)

	year	averageRating
0	1996	0.830303
1	1997	0.836364
2	1998	0.600000
3	1999	0.666667
4	2000	0.800000

Figure 3.4: Outcome Query 4

3.5. Genre Popularity Over Time

Understand how the popularity of different genres has changed over time based on the number of ratings.

3.5.1. Query 5

```
db.movies_ratings.aggregate([
    {
        $project: {
            year: { $year: { $toDate: "$timestamp" } },
   genres: 1
        }
    },
    { $unwind: "$genres" },
    {
        $group: {
            _id: { year: "$year", genre: "$genres" },
            ratingCount: { $sum: 1 }
        }
    },
    { $sort: { "_id.year": 1, ratingCount: -1 } }
])


```

3.5.2. Outcome Table (possibly partial)

	year	genre	ratingCount
0	1996	Drama	2518
1	1996	Comedy	2250
2	1996	Thriller	1990
3	1996	Action	1964
4	1996	Adventure	1459

Figure 3.5: Outcome Query 5

3.6. Top Rated Movies by Year

Examine the film with the highest average rating per each year and its genre, to find an eventual pattern between the top movies and the genre to which they belong.

3.6.1. Query 6

```
db.movies_ratings.aggregate([
    {
        $group: {
            _id: { year: "$release", movieId: "$movieId", title: "$title", genres: "$genres" },
            averageRating: { $avg: "$rating" }
        }
    },
    { $sort: { "_id.year": 1, "averageRating": -1 } },
    {
        $group: {
            _id: "$_id.year",
            movie: { $first: "$_id.title" },
            genres: { $first: "$_id.genres" },
            averageRating: { $first: "$averageRating" }
        }
    },
    { $sort: { "_id": 1 } }
])
```

3.6.2. Outcome Table (possibly partial)

	release	title	genres	rating	primary_genre
0	None	Black Mirror	[(no genres listed)]	1.0	(no genres listed)
1	1902	Trip to the Moon, A (Voyage dans la lune, Le)	[Action, Adventure, Fantasy, Sci-Fi]	0.7	Action
2	1903	The Great Train Robbery	[Crime, Western]	0.5	Crime
3	1908	The Electric Hotel	[Animation, Comedy, Sci-Fi]	0.8	Animation
4	1915	Birth of a Nation, The	[Drama, War]	0.4	Drama

Figure 3.6: Outcome Query 6

3.7. User Engagement Over Time

Analyze how individual users' rating activity changes over time, in order to gain an understanding of whether it increases or not with time; considering the number of ratings given by a user per each year.

3.7.1. Query 7

```
db.movies_ratings.aggregate([
    {
        $project: {
            userId: 1,
            year: { $year: { $toDate: "$timestamp" } }
        }
    },
    {
        $group: {
            _id: { userId: "$userId", year: "$year" },
            ratingCount: { $sum: 1 }
        }
    },
    { $sort: { "_id.userId": 1, "_id.year": 1 } }
])

```

3.7.2. Outcome Table (possibly partial)

	userId	year	ratingCount
0	1	2000	696
1	2	2015	87
2	3	2011	117
3	4	1999	246
4	4	2000	192

Figure 3.7: Outcome Query 7

3.8. Genre Popularity Evolution

Track how the popularity of different genres changes over the years based on the number of movies released in each genre and their average ratings.

3.8.1. Query 8

```
        db.movies_rating.aggregate([
    { $unwind: "$genres" },
    {
        $project: {
            releaseYear: "$release",   genres: 1, rating: 1
        }
    },
    {
        $group: {
            _id: { year: "$releaseYear", genre: "$genres" },
            totalMovies: { $sum: 1 },
            averageRating: { $avg: "$rating" }
        }
    },
    { $sort: { "_id.year": 1, "_id.genre": 1 } }
])

```

3.8.2. Outcome Table (possibly partial)

	year	genre	totalMovies	averageRating
1498	2018	Horror	8	0.575000
1499	2018	Mystery	8	0.612500
1500	2018	Romance	1	0.900000
1501	2018	Sci-Fi	47	0.742553
1502	2018	Thriller	21	0.628571

Figure 3.8: Outcome Query 8

3.9. Time-Based Rating Patterns

Insights on whether movie ratings are higher or lower during certain times of the year, considering the average ratings per month.

3.9.1. Query 9

```
db.movies_ratings.aggregate([
    {
        $project: {
            month: { $month: { $toDate: "$timestamp" } },
            rating: 1
        }
    },
    {
        $group: {
            _id: "$month",
            averageRating: { $avg: "$rating" }
        }
    },
    { $sort: { _id: 1 } }
])

```

3.9.2. Outcome Table (possibly partial)

Here the proper output of the query does not address each month by its name, but it follows the numerical notation, namely from 0 to 11. However, it is presented in this way to get a better understating of the results, and data visualisation.

	month	averageRating
0	January	0.700749
1	February	0.670399
2	March	0.691104
3	April	0.722700
4	May	0.690113
5	June	0.683445
6	July	0.727986
7	August	0.671810
8	September	0.719965
9	October	0.702532
10	November	0.726984
11	December	0.702981

Figure 3.9: Outcome Query 9

3.10. Correlation Between Rating Count and Average Rating

Discover an eventual correlation between the number of ratings and the average score per movies.

3.10.1. Query 10

```
db.movies_ratings.aggregate([
    {
        $group: {
            _id: {"movieID": "$movieId", "movieTitle": "
  $title"},
            averageRating: { $avg: "$rating" },
            ratingCount: { $sum: 1 }
        }
    }
])

```

3.10.2. Outcome Table (possibly partial)

	movieID	movieTitle	averageRating	ratingCount
0	2751	From the Hip	0.800000	1
1	6786	Kiss of the Spider Woman	0.775000	4
2	5833	Dog Soldiers	0.933333	3
3	69640	Public Enemies	0.611111	9
4	7025	Midnight Clear, A	0.750000	2
...	...	...	...	...

Figure 3.10: Outcome Query 10

4 | Data Analysis and Visualization with Python

As a mean of visualizing the data using Python and before running any query, it's necessary to establish a connection to the database.

The initial step establishes a connection to a MongoDB database, specifically to the collection named "*movies*".

```
import pymongo
import pandas as pd
import matplotlib.pyplot as plt

# Connect to MongoDB
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["movies"]
```

Consequently, follows the data visualization conducted based on the provided queries (3).

4.1. Average Rating per Genre

The average rating for each genre across all movies is useful to gain a better understanding of User Preferences. It provides insights into the genres that are most appreciated by the users. This could be used in a recommendation system as it can suggest new movies to users based on their preferred genres. Indeed, such visualization clearly shows which genres are popular among users, which can be used to prioritize certain genres in the recommendation algorithms, therefore, the system might recommend more movies like that.

4.1.1. Python Code 1

```
# Execute the query
pipeline = [
    { "$unwind": "$genres" },
    { "$group": { "_id": "$genres", "averageRating": { "$avg": "$rating" } } },
    { "$sort": { "averageRating": -1 } }
]
average_ratings = list(db.movies_ratings.aggregate(pipeline))

# Convert to DataFrame
df = pd.DataFrame(average_ratings)

# Plotting
plt.figure(figsize=(10, 6))
plt.barh(df['_id'], df['averageRating'], color='skyblue')
plt.xlabel('Average Rating')
plt.ylabel('Genre')
plt.title('Average Rating per Genre')
plt.gca().invert_yaxis()  # Invert y-axis to have highest rating on top
plt.show()
```

4.1.2. Data Visualisation 1

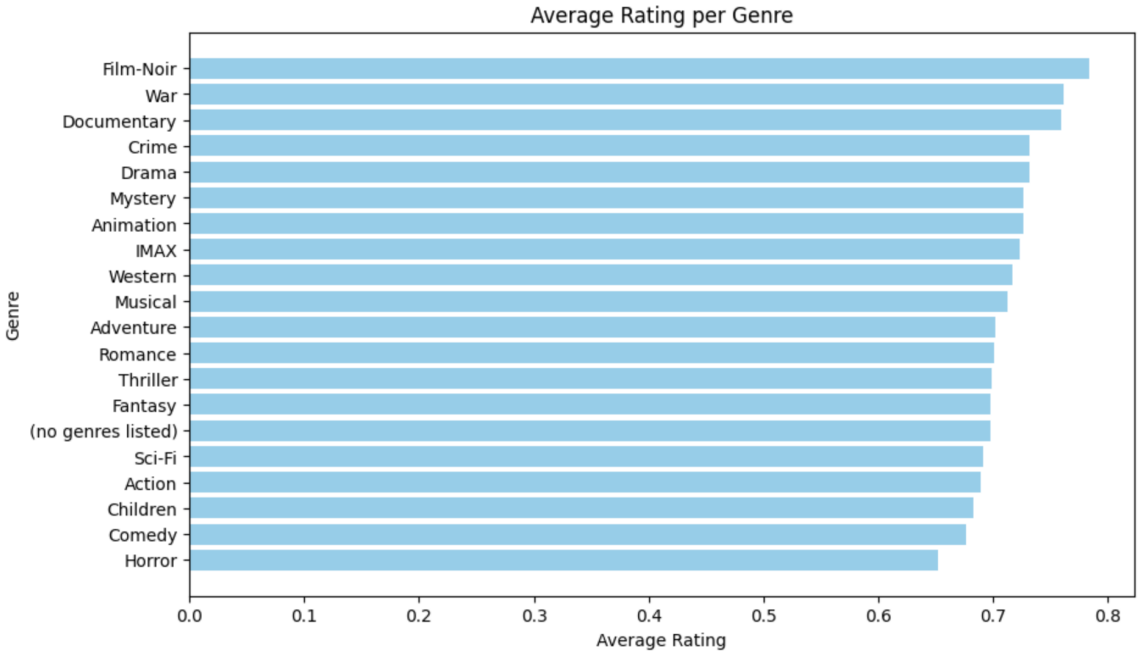

Figure 4.1: Average Rating per Genre

4.2. User Rating Patterns

In order to understand the behavior of active users (defined as the user who have rated more than 300 ratings) in the dataset, it is used a scatter plot to visualize each user's total ratings against their average rating.

This plot helps in identifying patterns in how user rating behavior varies with activity level. For instance, do more active users tend to give higher or lower ratings on average? It's useful for detecting outliers, such as users who rate a lot but have unusually high or low average ratings. The plot can also help to acknoledge the distribution of users' activity and their rating tendencies, important for understanding the dataset's overall characteristics. By analyzing which genres are rated most frequently and highly by active users, a possible recommendation system can better cater to the most engaged segment of the user base.

4.2.1. Python Code 2

```python
# Execute the query
pipeline = [
    { "$unwind": "$genres" },
    { "$group": { "_id": { "userId": "$userId", "genre": "$genres" }, "totalRatingsPerGenre": { "$sum": 1 }, "averageRatingPerGenre": { "$avg": "$rating" } } },
    { "$sort": { "_id.userId": 1, "totalRatingsPerGenre": -1 } },
    { "$group": { "_id": "$_id.userId", "totalRatings": { "$sum": "$totalRatingsPerGenre" }, "averageRating": { "$avg": "$averageRatingPerGenre" }, "mostRatedGenre": { "$first": "$_id.genre" } } },
    { "$match": { "totalRatings": { "$gt": 300 } } },
    { "$sort": { "totalRatings": -1 } }
]
user_patterns = list(db.movies_ratings.aggregate(pipeline))

# Convert to DataFrame
df_users = pd.DataFrame(user_patterns)

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(df_users['totalRatings'], df_users['averageRating'], color='blue')
plt.xlabel('Total Ratings')
plt.ylabel('Average Rating')
```

```
plt.title('User Rating Patterns for Active Users')
plt.show()
```

4.2.2. Data Visualisation 2

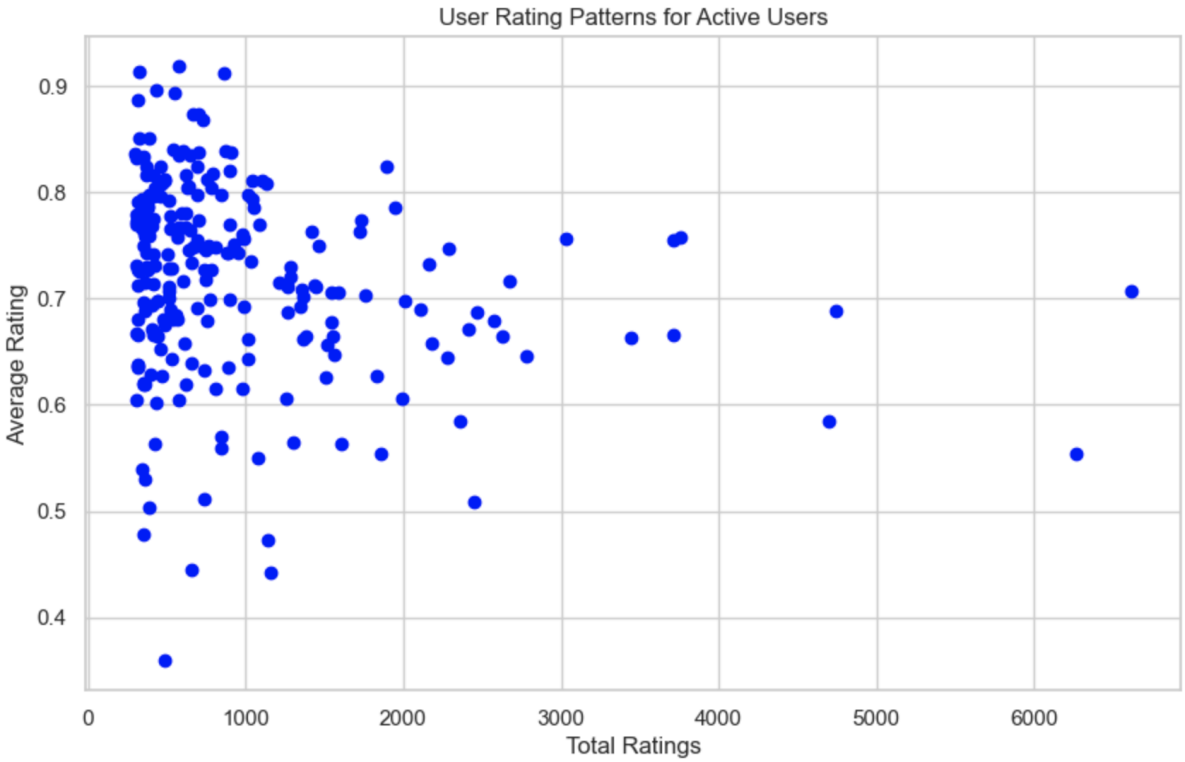

Figure 4.2: User Rating Patterns

4.3. Most Polarizing Movies

As a mean of visualisation of the most polarizing movies, the bar chart allows for clear comparison across different movies regarding their variance in ratings.

This visualization helps in understanding the general audience's opinion about a movie. It's crucial to understand not just the average rating of a movie but also how these ratings are distributed. A movie with a high average rating but a wide distribution might be polarizing, appealing strongly to certain viewers but not to others.

4.3.1. Python Code 3

```python
# MongoDB aggregation query
pipeline = [
    {
        "$group": {
            "_id": {"movieId": "$movieId", "movie": "$title"},
            "averageRating": {"$avg": "$rating"},
            "count": {"$sum": 1},
            "sumOfSquares": {"$sum": {"$pow": ["$rating", 2]}},
            "sumOfRatings": {"$sum": "$rating"}
        }
    },
    {
        "$project": {
            "averageRating": 1,
            "variance": {
                "$subtract": [
                    {"$divide": ["$sumOfSquares", "$count"]},
                    {"$pow": [{"$divide": ["$sumOfRatings", "$count"]}, 2]}
                ]
            }
        }
    },
    {"$sort": {"variance": -1}}
]

# Execute the query
result = list(db.movies_ratings.aggregate(pipeline))

# Convert to DataFrame
df = pd.DataFrame(result)

```

```python
# Add a column for movie titles (extracted from the _id field)
df['movie'] = df['_id'].apply(lambda x: x['movie'])



# Fetch the top 10 most polarizing movies
top_movies = list(db.movies.aggregate(pipeline))[:10]
top_movie_ids = [movie['_id']['movieId'] for movie in top_movies]
# Function to plot rating distribution for a specific movie
def plot_rating_distribution(movie_id, movie_title, ax):
    # Fetch ratings for the specific movie
    ratings_data = list(db.movies_ratings.find({"movieId": movie_id}, {"rating": 1}))
    ratings_df = pd.DataFrame(ratings_data)


    sns.histplot(ratings_df, kde=True, bins=10, ax=ax)
    ax.set_title(f"Rating for '{movie_title}'", fontsize=10)
    ax.set_xlabel("Rating", fontsize=8)
    ax.set_ylabel("Frequency", fontsize=8)

# Create a 2x5 subplot matrix
fig, axes = plt.subplots(2, 5, figsize=(20, 10))
fig.tight_layout(pad=4.0)

# Plot distribution for each of the top 10 movies in the matrix
for i, movie in enumerate(top_movies):
    movie_id = movie['_id']['movieId']
    movie_title = movie['_id']['movie']
    row, col = divmod(i, 2)
    plot_rating_distribution(movie_id, movie_title, axes[col, row])

plt.show()
```

4.3.2. Data Visualisation 3

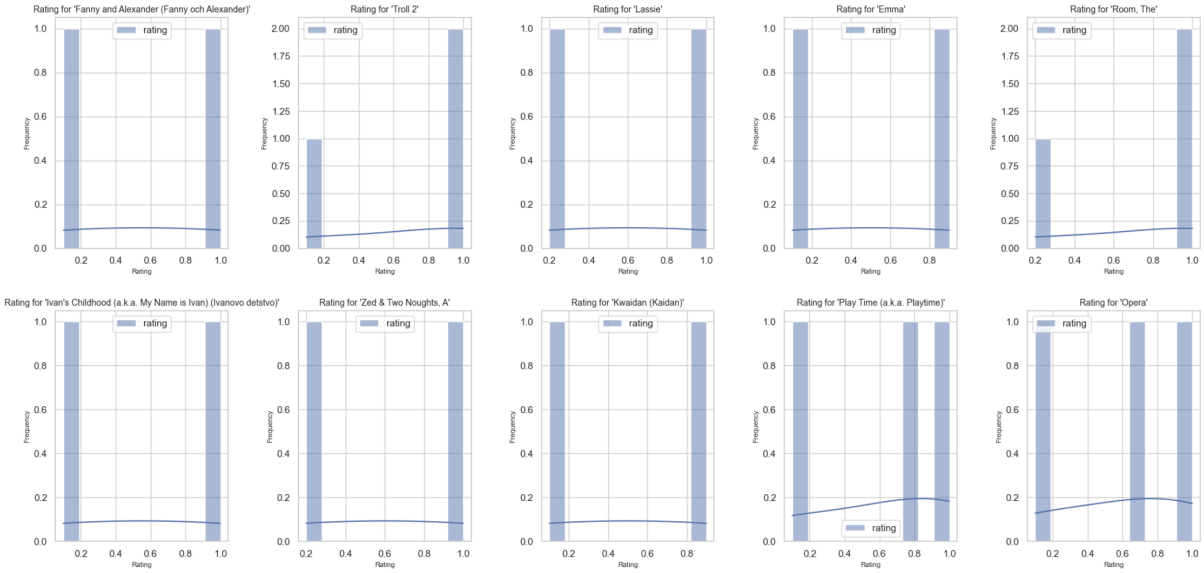

Figure 4.3: Most Polarizing Movies

4.4. Temporal Trends in Rating

The following query has the purpose of visualizing the distribution of ratings, hence the change in average movie ratings over time for a specific movie (like "Toy Story").

The provided script is tailored for analyzing and visualizing data, particularly focusing on how the popularity and reception of movies change over time. This visualization is useful to see how the movie's reception has evolved over time. An upward trend in the plot could indicate increasing popularity or a growing appreciation of the film. Understanding these temporal trends is important as it can identify timeless classics or movies that are currently trending.

4.4.1. Python Code 4

```python

# MongoDB aggregation query
pipeline = [
    { "$match": { "title": "Toy Story" } },
    { "$project": { "year": { "$year": { "$toDate": "$timestamp" } }, "rating": 1 } },
    { "$group": { "_id": { "year": "$year" }, "averageRating": { "$avg": "$rating" }, "numRatings": {"$sum": 1} } },
    { "$sort": { "_id.year": 1 } }
]

# Execute the query
temporal_trends = list(db.movies_ratings.aggregate(pipeline))

# Ensure the DataFrame creation is correctly mapping the data
df = pd.DataFrame([{ 'year': item['_id']['year'], 'averageRating': item['averageRating']} for item in temporal_trends])

# Check if the DataFrame is created correctly
df.head()

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(df['year'], df['averageRating'], marker='o')
plt.xlabel('Year')
plt.ylabel('Average Rating')
plt.title('Temporal Trends in Ratings for "Toy Story (1995)"')
plt.grid(True)
plt.show()
```

4.4.2. Data Visualisation 4

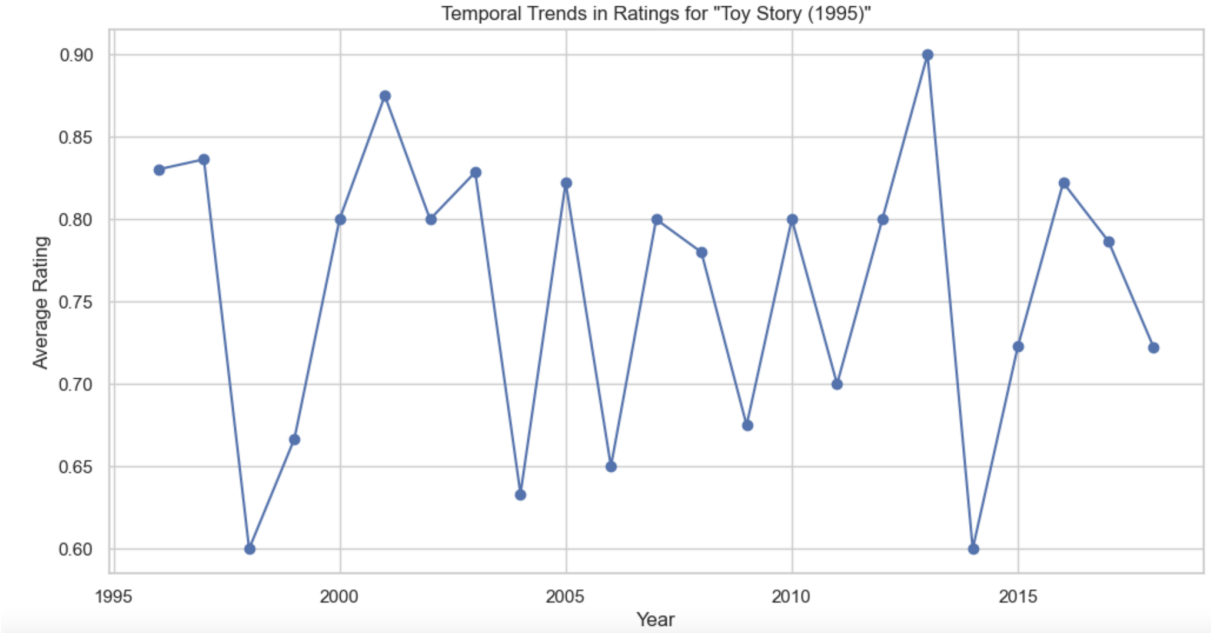

Figure 4.4: Temporal Trends in Rating

4.5. Top Rated Movies by Year

In order to find an eventual pattern between the top movies and the genre to which they belong, the data is pivoted to form a matrix with years as rows and genres as columns, with the values representing the count of ratings. The heat map displays the correlation between different genres over the years, as well as identifying trends in genre popularity over time. For instance, if certain genres consistently receive more ratings in specific years, it could indicate shifting user preferences.

The heatmap also allows for the identification of any correlations between genres, which can be leveraged to suggest similar genres to users.

4.5.1. Python Code 5

```python
# MongoDB aggregation query
pipeline = [
    {"$project": {"year": {"$year": {"$toDate": "$timestamp"}}, "genres": 1}},
    {"$unwind": "$genres"},
    {"$group": {"_id": {"year": "$year", "genre": "$genres"}, "ratingCount": {"$sum": 1}}},
    {"$sort": {"_id.year": 1, "ratingCount": -1}}
]

# Execute the query
genre_popularity = list(db.movies_ratings.aggregate(pipeline))
# Convert to DataFrame
df = pd.DataFrame([{"year": item["_id"]["year"], "genre": item["_id"]["genre"], "ratingCount": item["ratingCount"]} for item in genre_popularity])

# Pivot the data for visualization
pivot_df = df.pivot(index='year', columns='genre', values='ratingCount').fillna(0)
plt.figure(figsize=(12, 8))
sns.heatmap(pivot_df.corr(), annot = True, cmap = 'YlGnBu', vmin = -1, vmax = 1)
#sns.heatmap(pivot_df, cmap="YlGnBu", annot=False, cbar=True)
plt.title('Genre Popularity Over Time Based on Number of Ratings')
plt.xlabel('Genre')
plt.ylabel('Year')
plt.show()
```

4.5.2. Data Visualisation 5

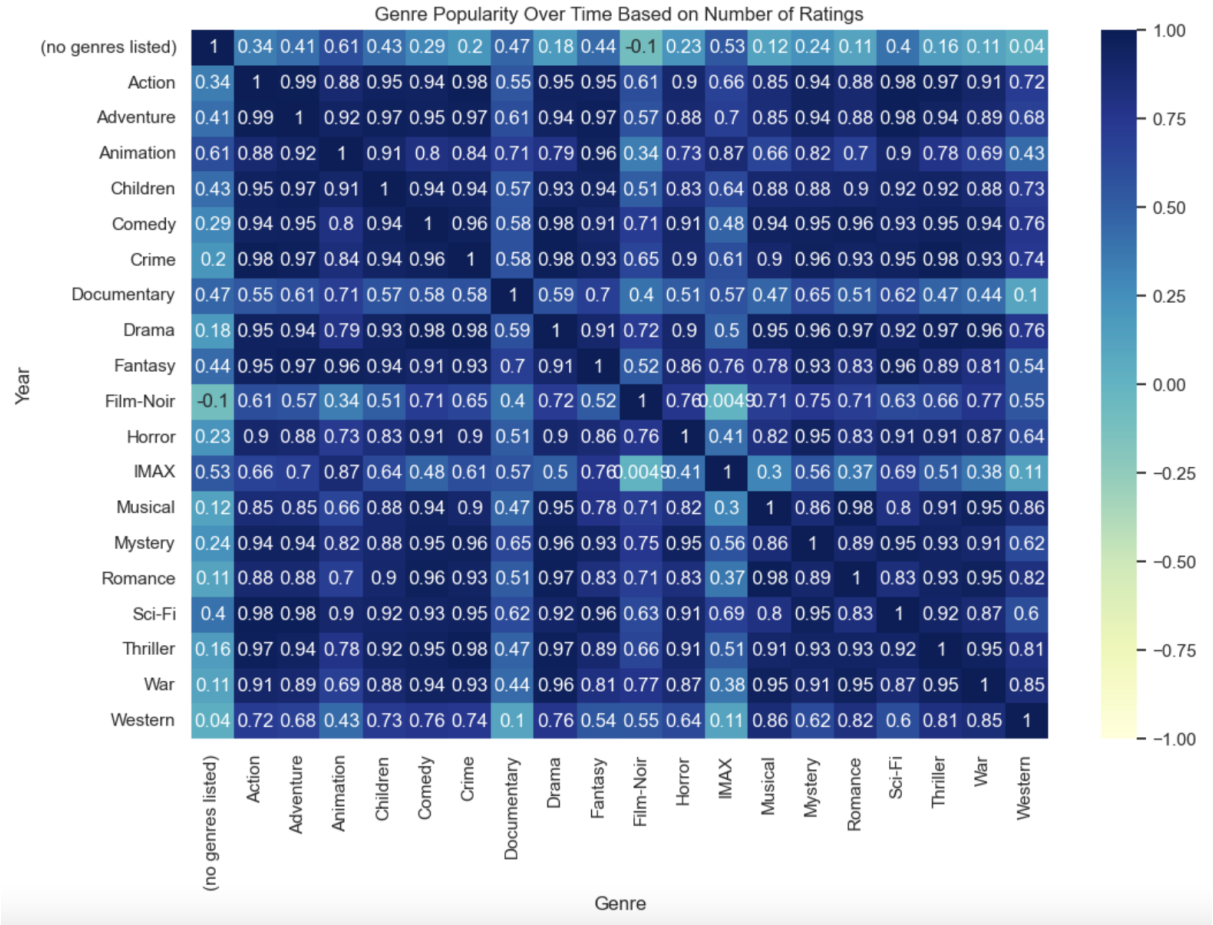

Figure 4.5: Top Rated Movies by Year

4.6. Most Polarizing Movies

To examine the film with the highest average rating per each year and its genre, to find an eventual pattern between the top movies and the genre to which they belong to. A scatter plot is used to visualize the top-rated movies over the years. Each point in the scatter plot corresponds to a top-rated movie in a particular year, with the color indicating its primary genre. This visualization allows for the easy identification of trends and patterns in movie ratings over the years, particularly in relation to genres. For instance, one might observe certain genres consistently appearing among the top-rated movies, indicating their popularity or quality.

4.6.1. Python Code 6

```python
# Perform the aggregation query
pipeline = [
    {
        "$group": {
            "_id": {"year": "$release", "movieId": "$movieId", "title": "$title", "genres": "$genres"},
            "averageRating": {"$avg": "$rating"}
        }
    },
    {"$sort": {"_id.year": 1, "averageRating": -1}},
    {
        "$group": {
            "_id": "$_id.year",
            "movie": {"$first": "$_id.title"},
            "genres": {"$first": "$_id.genres"},
            "averageRating": {"$first": "$averageRating"}
        }
    },
    {"$sort": {"_id": 1}}
]

results = list(db.movies_ratings.aggregate(pipeline))

# Convert the results into a pandas DataFrame
df = pd.DataFrame([{
    'release': r['_id'],
    'title': r['movie'],
    'genres': r['genres'],
    'rating': r['averageRating']
```

```
} for r in results])

# Convert the 'genres' list into a string of the first genre
df['primary_genre'] = df['genres'].apply(lambda x: x[0] if x else None)

# Visualizing the top-rated movies over the years
plt.figure(figsize=(15, 10))
sns.scatterplot(data=df, x='release', y='rating', hue='primary_genre',
    palette='viridis', s=100)
plt.title('Top Rated Movies by Year and Primary Genre', fontsize=16)
plt.xlabel('Year of Release', fontsize=12)

plt.xticks(rotation=45)
plt.ylabel('Average Rating', fontsize=12)
plt.legend(title='Primary Genre', bbox_to_anchor=(1.05, 1), loc='upper left
    ')
plt.tight_layout()
plt.show()
```

4.6.2. Data Visualisation 6

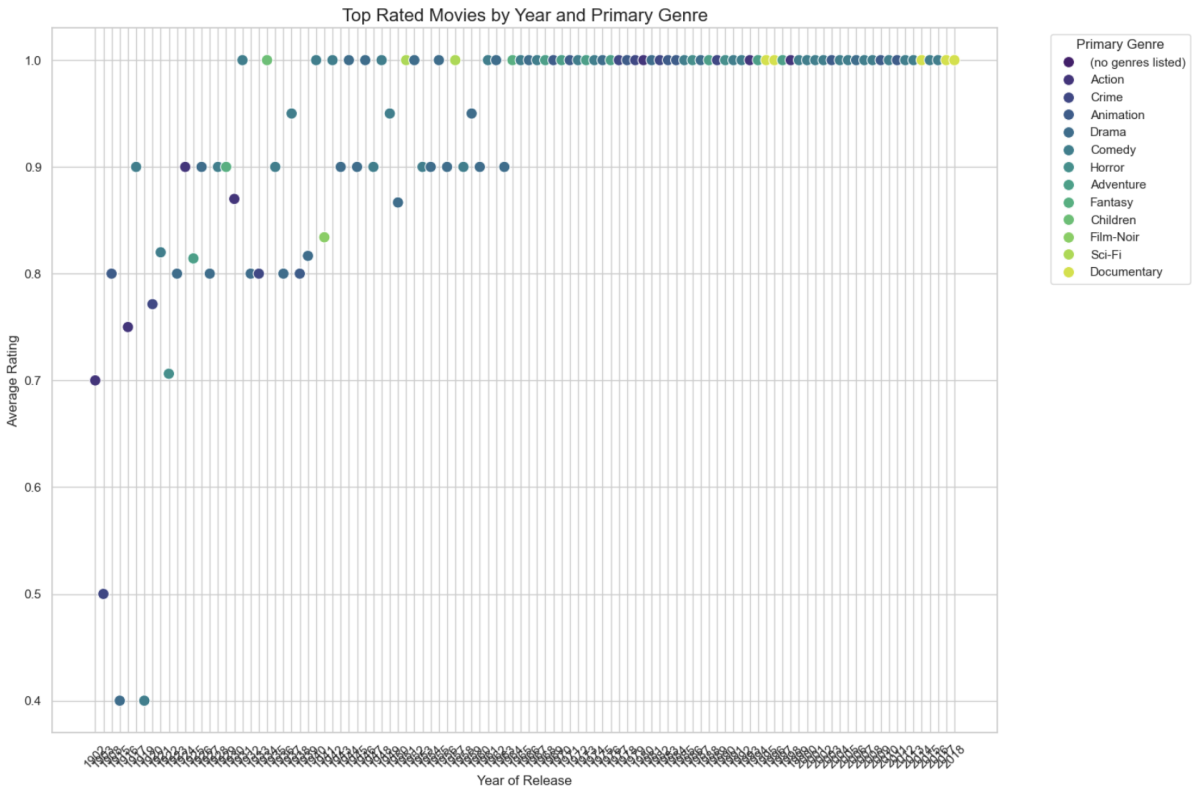

Figure 4.6: Most Polarizing Movies

4.7. User Engagement Over Time

How individual users' rating activity changes over time is an analysis crucial for understanding user behavior, considering the number of ratings given by a user per each year. The data is pivoted to create a matrix with years as rows and user IDs as columns, where each cell represents the number of ratings by a user in a particular year. A line plot is used to visualize this data, with different lines representing different users. The x-axis represents years, and the y-axis represents the number of ratings. This visualization allows us to observe and compare the rating activity of different users over time. Trends in user engagement can be identified, such as whether certain users are consistently active in rating movies, or if there are periods of increased or decreased activity.

4.7.1. Python Code 7

```python
# MongoDB aggregation query
pipeline = [
    {
        "$project": {
            "userId": 1,
            "year": {"$year": {"$toDate": "$timestamp"}}
        }
    },
    {
        "$group": {
            "_id": {"userId": "$userId", "year": "$year"},
            "ratingCount": {"$sum": 1}
        }
    },
    {"$sort": {"_id.userId": 1, "_id.year": 1}}
]

# Execute the query
result = list(db.movies.aggregate(pipeline))[:10]
# Convert the result to a DataFrame
df = pd.DataFrame([{'userId': x['_id']['userId'], 'year': x['_id']['year'],
    'ratingCount': x['ratingCount']} for x in result])

# Pivot the DataFrame for visualization
pivot_df = df.pivot(index='year', columns='userId', values='ratingCount').
    fillna(0)

# Plotting
```

```
plt.figure(figsize=(15, 8))
plt.plot(pivot_df, alpha=0.5)
plt.xlabel('Year')
plt.ylabel('Number of Ratings')
plt.title('User Engagement Over Time')

plt.show()
```

4.7.2. Data Visualisation 7

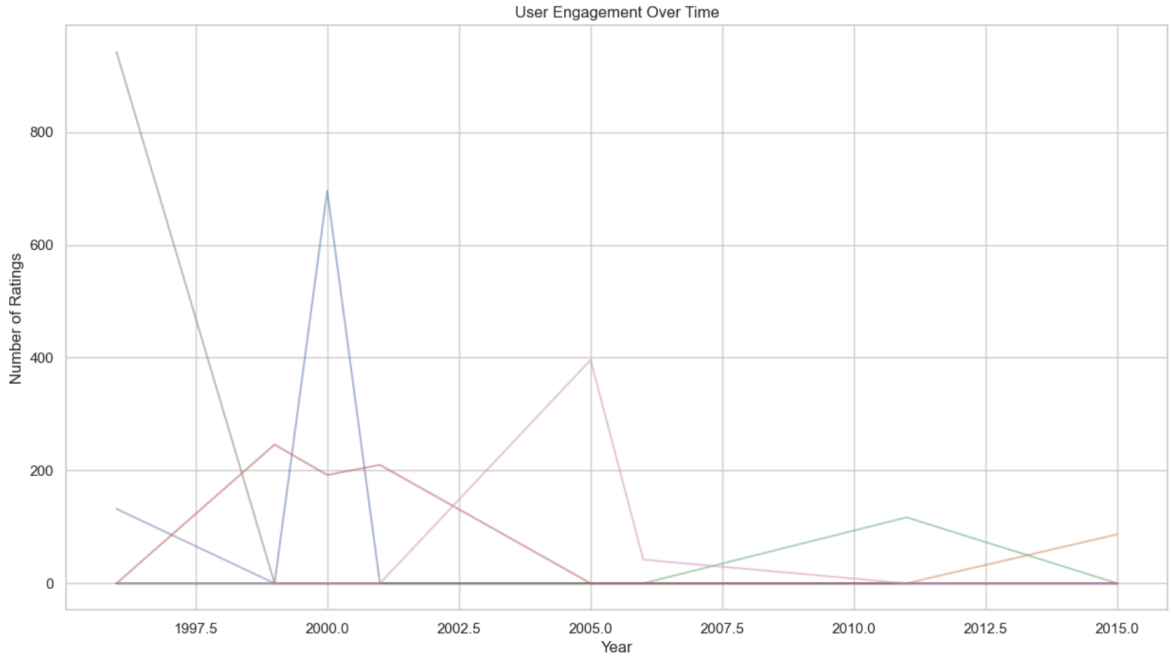

Figure 4.7: User Engagement Over Time

4.8. Genre Popularity Evolution

Keeping track of how the popularity of different genres changes over the years based on the number of movies released in each genre and their average ratings, is the type of analysis crucial for understanding trends in movie production and audience preferences.

The first plot shows the total number of movies per genre over the years.
The second plot shows the average rating per genre over the years.
In both plots, the x-axis represents the year, and the y-axis represents either the total number of movies or the average rating.

This plot provides insights into how the production of movies in different genres has changed over time. An increasing trend for a particular genre indicates growing popularity or market demand. Furthermore, it helps in understanding how audience reception (in terms of ratings) for different genres has evolved over the years. It can highlight if certain genres are consistently receiving higher ratings, indicating their quality or resonance with the audience.

4.8.1. Python Code 8

```python

# Execute the aggregation query
pipeline = [
    { "$unwind": "$genres" },
    {
        "$project": {
            "releaseYear": "$release",
            "genres": 1,
            "rating": 1
        }
    },
    {
        "$group": {
            "_id": { "year": "$releaseYear", "genre": "$genres" },
            "totalMovies": { "$sum": 1 },
            "averageRating": { "$avg": "$rating" }
        }
    },
    { "$sort": { "_id.year": 1, "_id.genre": 1 } }
]

results = list(db.movies_ratings.aggregate(pipeline))
```

```python

# Convert results to a DataFrame
df = pd.DataFrame([{
    'year': r['_id']['year'],
    'genre': r['_id']['genre'],
    'totalMovies': r['totalMovies'],
    'averageRating': r['averageRating']
} for r in results])

# Plotting
plt.figure(figsize=(20, 10))

# Plot for total movies per genre over the years
sns.lineplot(data=df, x='year', y='totalMovies', hue='genre', marker='o')
plt.title('Total Number of Movies per Genre Over the Years', fontsize=16)
plt.xlabel('Year', fontsize=12)
plt.xticks(rotation=45)
plt.ylabel('Total Movies', fontsize=12)
plt.legend(title='Genre', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show the plot
plt.show()

# Plot for average rating per genre over the years
plt.figure(figsize=(20, 10))
sns.lineplot(data=df, x='year', y='averageRating', hue='genre', marker='o')
plt.title('Average Rating per Genre Over the Years', fontsize=16)
plt.xlabel('Year', fontsize=12)


plt.xticks(rotation=45)
plt.ylabel('Average Rating', fontsize=12)
plt.legend(title='Genre', bbox_to_anchor=(1.05, 1), loc='upper left')

# Show the plot
plt.show()
```

4.8.2. Data Visualisation 8

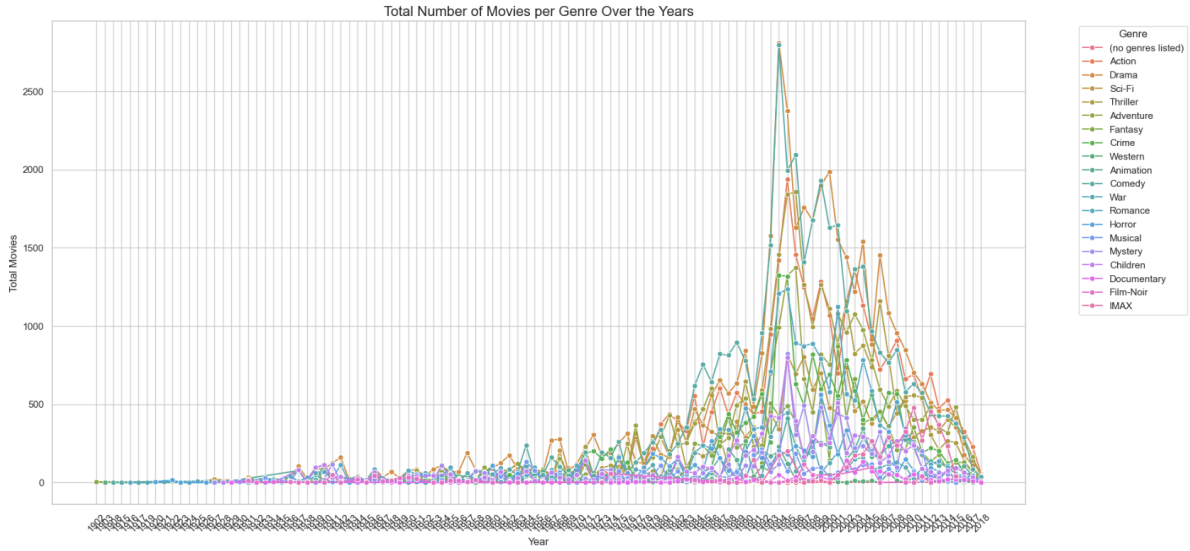

Figure 4.8: Genre Popularity Evolution 1

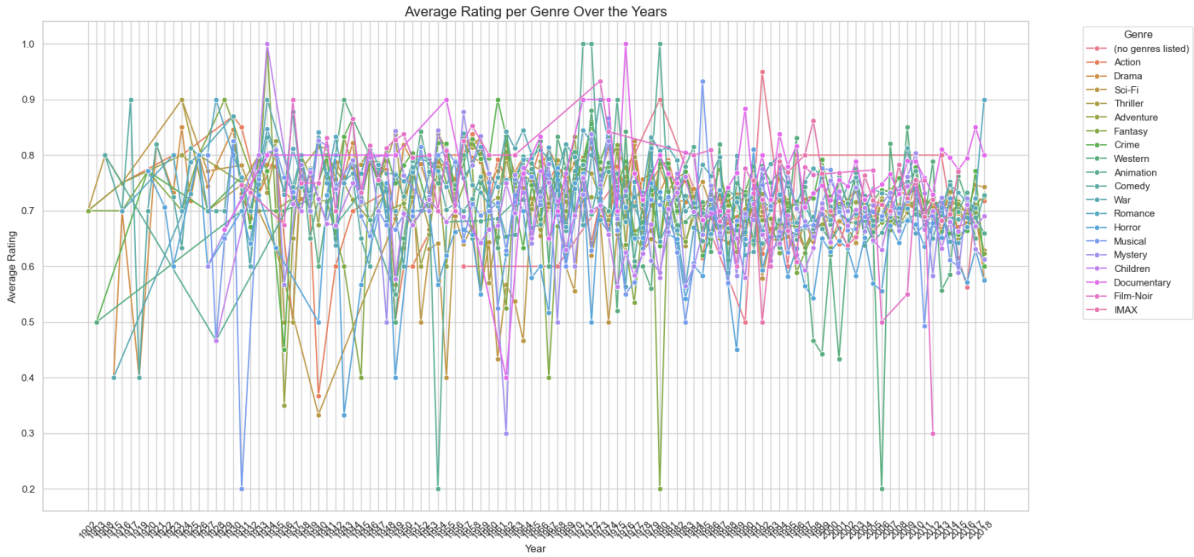

Figure 4.9: Genre Popularity Evolution 2

4.9. Time-Based Rating Patterns

Insights on whether movie ratings are higher or lower during certain times of the year, considering the average ratings per month, is an approach that helps in understanding if there are seasonal trends in movie ratings, possibly influenced by factors like holiday seasons, summer blockbusters, or award seasons.

A bar plot is used to visualize the average movie ratings by month. The x-axis represents months of the year, and the y-axis represents the average movie ratings. Such trends can indicate viewer preferences and industry patterns, such as the release of highly anticipated movies during specific periods.

4.9.1. Python Code 9

```python
# Execute the aggregation query
pipeline = [
    {
        "$project": {
            "month": { "$month": { "$toDate": "$timestamp" } },
            "rating": 1
        }
    },
    {
        "$group": {
            "_id": "$month",
            "averageRating": { "$avg": "$rating" }
        }
    },
    { "$sort": { "_id": 1 } }
]

results = list(db.movies_ratings.aggregate(pipeline))

# Convert results to a DataFrame
df = pd.DataFrame([{
    'month': r['_id'],
    'averageRating': r['averageRating']
} for r in results])

# Map month numbers to names for better readability
month_names = {1: 'January', 2: 'February', 3: 'March', 4: 'April', 5: 'May', 6: 'June',
               7: 'July', 8: 'August', 9: 'September', 10: 'October', 11: '
```

```
    November', 12: 'December'}
df['month'] = df['month'].map(month_names)

# Plotting
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='month', y='averageRating', palette='coolwarm')
plt.title('Average Movie Ratings by Month', fontsize=16)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Average Rating', fontsize=12)
plt.xticks(rotation=45)

# Show the plot
plt.show()
```

4.9.2. Data Visualisation 9

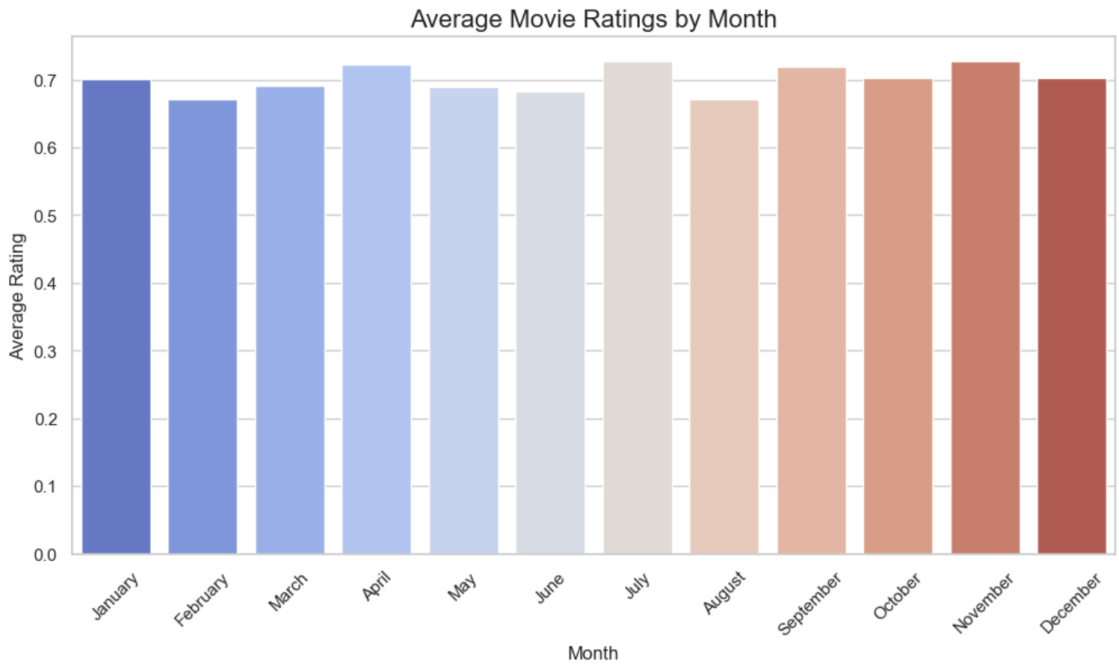

Figure 4.10: Time-Based Rating Patterns

4.10. Correlation Between Rating Count and Average Rating

Discover an eventual correlation between the number of ratings and the average score per movies. This kind of analysis can be very useful in understanding movie popularity and audience perception. Each point on the scatter plot represents a movie, positioned according to its total number of ratings and its average rating. This visualization is designed to explore the relationship between the number of ratings a movie has and its average rating. For instance, if there's a trend where movies with more ratings tend to have higher average ratings, this could indicate that popular movies are generally well-received, or it could reflect a bias where viewers are more likely to rate movies they like. Understanding the correlation between rating count and average rating can help distinguish between movies that are popular (many ratings) and those that are highly rated (high average rating). This analysis might reveal biases in user behavior, such as a tendency to rate only movies that they feel strongly about.

4.10.1. Python Code 10

```python
# Execute the aggregation query
pipeline = [
    {
        "$group": {
            "_id": {"movieID": "$movieId", "movieTitle": "$title"},
            "averageRating": { "$avg": "$rating" },
            "ratingCount": { "$sum": 1 }
        }
    }
]

results = list(db.movies_ratings.aggregate(pipeline))

# Convert results to a DataFrame
df = pd.DataFrame([{
    'movieID': r['_id']['movieID'],
    'movieTitle': r['_id']['movieTitle'],
    'averageRating': r['averageRating'],
    'ratingCount': r['ratingCount']
} for r in results])


# Setting the theme for seaborn
```

```
sns.set_theme(style="whitegrid")

# Creating a scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='ratingCount', y='averageRating')

# Adding titles and labels
plt.title('Correlation Between Rating Count and Average Rating')
plt.xlabel('Rating Count')
plt.ylabel('Average Rating')

# Display the plot
plt.show()
```

4.10.2. Data Visualisation 10

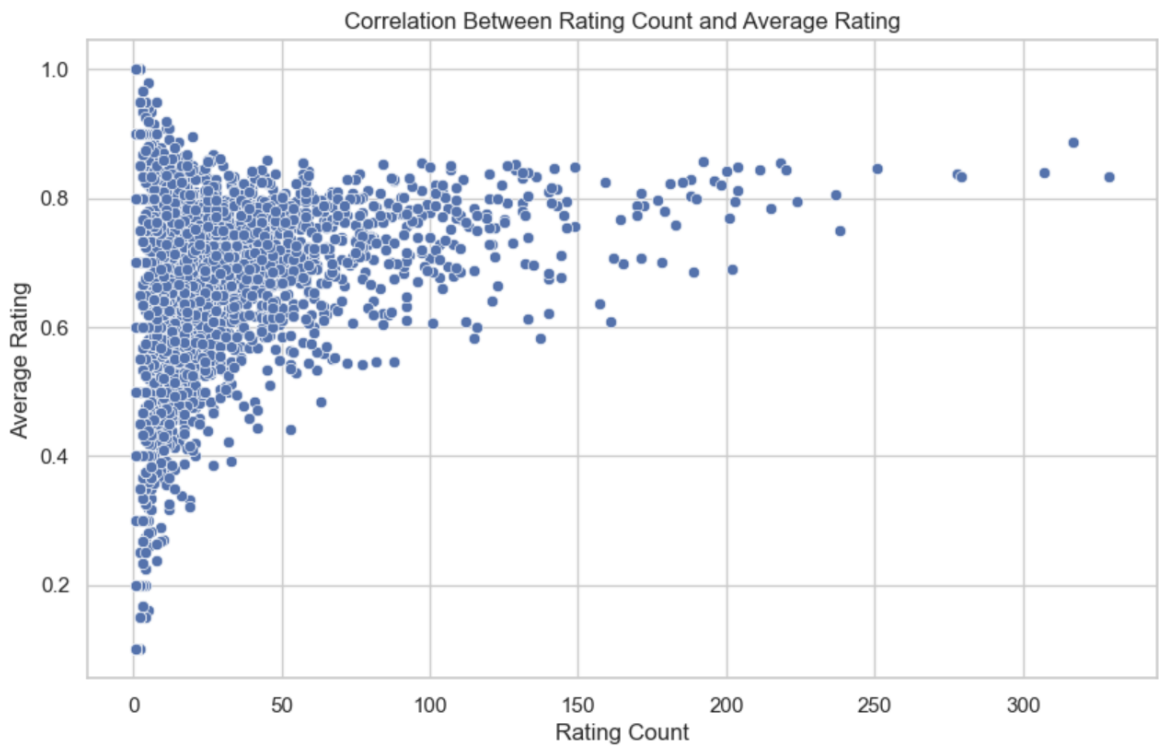

Figure 4.11: Correlation Between Rating Count and Average Rating

All the enlisted procedures are in the *"extra.ipynb"* file.

List of Figures

3.1 Outcome Query 1 10
3.2 Outcome Query 2 12
3.3 Outcome Query 3 14
3.4 Outcome Query 4 16
3.5 Outcome Query 5 18
3.6 Outcome Query 6 19
3.7 Outcome Query 7 21
3.8 Outcome Query 8 22
3.9 Outcome Query 9 24
3.10 Outcome Query 10 25

4.1 Average Rating per Genre 29
4.2 User Rating Patterns 31
4.3 Most Polarizing Movies 34
4.4 Temporal Trends in Rating 36
4.5 Top Rated Movies by Year 38
4.6 Most Polarizing Movies 40
4.7 User Engagement Over Time 42
4.8 Genre Popularity Evolution 1 45
4.9 Genre Popularity Evolution 2 45
4.10 Time-Based Rating Patterns 47
4.11 Correlation Between Rating Count and Average Rating 49