TDT4225 – Very Large, Distributed Data Volumes

Assignment 3: MongoDB

Norwegian University of Science and Technology

Fermín Martínez Cintas, Jordan Ikukele Bomolo Mia & Liang Ji Zhu

October 30, 2025

Contents

1 Introduction

In this third assignment, we worked with the *Movies Dataset* to perform a large-scale data exploration, filtering, and storage in **MongoDB**. The objective was to prepare a consistent and compact dataset suitable for advanced analytical queries while applying a distributed database environment.

Our workflow was divided into four main components:

1. Data exploration and cleaning (`filter_movies.py`)
2. Exploratory Data Analysis (EDA) (`eda_movies.ipynb`)
3. Data insertion into MongoDB (`insertion.py`)
4. Analytical queries and aggregation pipelines (`task2.ipynb`)

All work was performed collaboratively through a shared Git repository and Docker-based MongoDB deployment. Due to the large size of the datasets, we employed chunk-based processing and efficient filtering strategies to maintain performance and reliability.

2 Data Filtering and Preprocessing

Before inserting the data into MongoDB, we developed a robust preprocessing script called `filter_movies.py`. This script integrates, cleans and filters several large CSV files provided with the Movies Dataset — including `movies_metadata.csv`, `credits.csv`, `links.csv`, `keywords.csv`, and `ratings.csv`.

2.1 Objective

The purpose of this filtering step was to:

- Remove inconsistent, incomplete or redundant entries.
- Merge the five datasets into a single, coherent structure.
- Standardize formats (numeric, date, list, and JSON-like fields).
- Reduce the dataset's size to improve insertion speed and query efficiency in MongoDB.

2.2 Filtering Procedure

The cleaning pipeline in `filter_movies.py` consists of several well-defined steps designed to guarantee data integrity and consistency across all movie-related files.

1. **Load and parse source CSVs:** The script reads the .csv files using chunked reading where necessary. Fields stored as JSON strings (e.g., `genres`, `cast`, `crew`, `keywords`) are parsed with Python's `ast.literal_eval()`.

2. **Data type correction:** The column `id` is converted from string to numeric to ensure correct merging across datasets. Invalid or missing IDs are removed. Numeric fields such as `budget`, `revenue`, `vote_average`, and `vote_count` are coerced to numeric values, replacing errors with zeros or nulls.

3. **Merging datasets:** Datasets are merged on the movie `id` column:

   - `movies_metadata` + `credits` → adds cast and crew information.
   - `+ keywords` → adds thematic descriptors.

   This results in a unified DataFrame containing metadata, production information, and textual tags for each movie.

4. **Cleaning and reducing columns:** Irrelevant or redundant columns such as `poster_path`, `homepage`, `imdb_id`, and `video` are removed to simplify the dataset and reduce storage usage. Only essential analytical fields are kept: `id`, `title`, `genres`, `cast`, `crew`, `keywords`, `release_date`, `budget`, `revenue`, `vote_average`, and `vote_count`.

5. **Date and missing values:** The column `release_date` is converted to `datetime` format, and rows missing critical fields (ID or release date) are dropped. This ensures that every record corresponds to a real, valid movie.

6. **Ratings filtering:** The `ratings.csv` file is filtered to include only movies that exist in the cleaned dataset, removing irrelevant ratings. This maintains referential consistency between movies and user ratings.

7. **Export cleaned datasets:** The resulting datasets are saved as:

   - `movies_clean.csv`
   - `credits_clean.csv`
   - `links_clean.csv`
   - `ratings_clean.csv`
   - `keywords_clean.csv`
   - `keywords_exploded.csv`

   They are ready for MongoDB insertion.

3 Exploratory Data Analysis (EDA)

Before deciding how to structure and filter the dataset, we conducted an extensive exploratory data analysis (EDA) using `eda_movies.ipynb`. This notebook includes visualizations, statistics, and data quality checks that were essential in understanding the dataset's structure, anomalies, and limitations.

The EDA serves not only as a summary of the dataset, but also as a decision-making tool that informed every other step of the project: from filtering thresholds to database schema design and query optimization. We strongly recommend reviewing the notebook itself, as it contains rich commentary and reproducible code beyond what we summarize here.

Key Insights and Motivations

- **Rating Distributions:** Most movies in the dataset have a `vote_average` centered around 6–7, with a sharp cutoff below 4 and very few high scores. This peak at 0 corresponds to movies with no votes (0) or incorrectly entered records. It can be filtered out during cleanup (vote_count == 0 or vote_average == 0).
- **Temporal Trends:** By converting `release_date` to datetime and extracting the `year`, we visualized the evolution of movie production. A surge in movies after the 1990s was evident, along with a rise in incomplete or invalid release dates in earlier entries.
- **Popularity Bias:** The `vote_count` distribution followed a classic long-tail pattern (few movies with many votes). This insight helped justify filtering based on a minimum number of ratings to ensure statistical significance.
- **Votes vs. Score Stability:** A scatter plot of `vote_count` vs. `vote_average` showed that movies with more votes tend to have more stable and moderate scores. This justified the use of weighted ratings and filtering thresholds.
- **Runtime Outliers:** We observed runtime values exceeding 800 minutes and several near-zero durations. Based on this, we applied a maximum cap to remove extreme values and replaced invalid entries with `NaN`.
- **Crew and Cast Coverage:** We analyzed the coverage of cast and crew per movie, counting how many names were associated with each. This helped detect duplicate or sparse entries and select the most complete records during filtering.
- **Keywords Distribution:** Using parsed JSON fields, we evaluated the number of thematic keywords per movie. A majority of entries had between 1 and 5 keywords, providing moderate signal for semantic filtering.

Key Findings and Graphs

We illustrate below a selection of plots that were instrumental to our analysis:

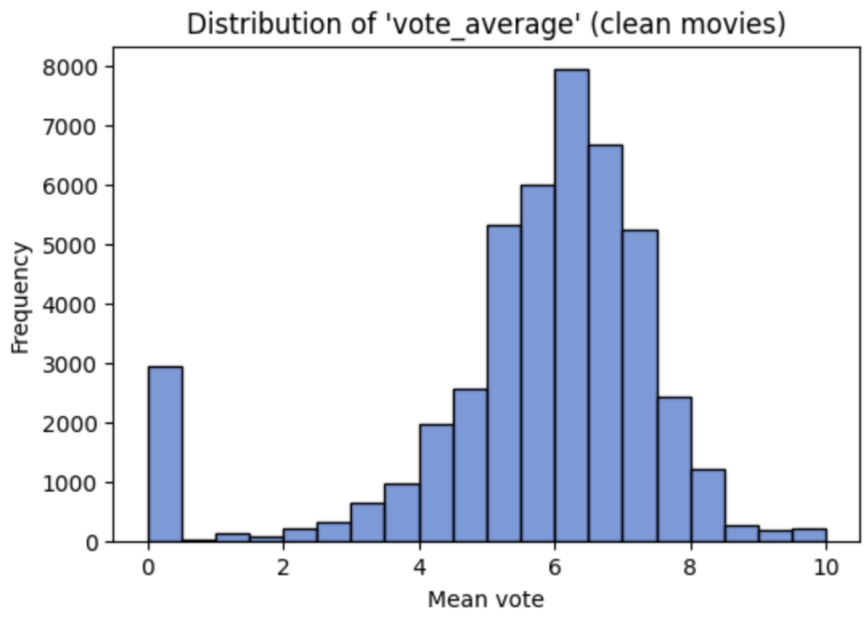


Figure 1: Distribution of `vote_average` values (clean movies only). This histogram confirmed that most movies are rated between 5 and 7, with very few extreme values.

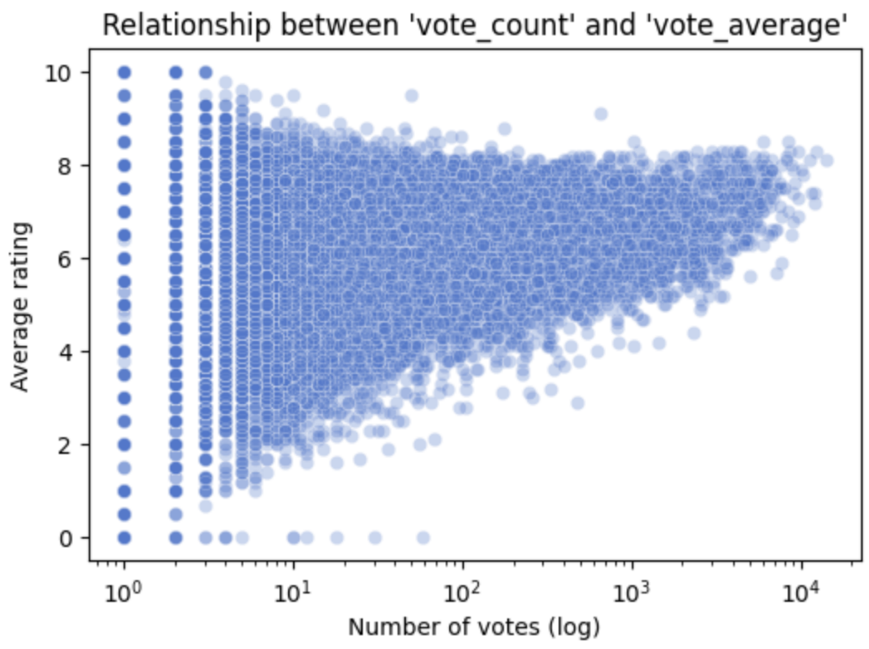


Figure 2: Scatter plot showing the relationship between `vote_count` and `vote_average`. Movies with few votes display greater variance, confirming our decision to remove films with low `vote_count`.

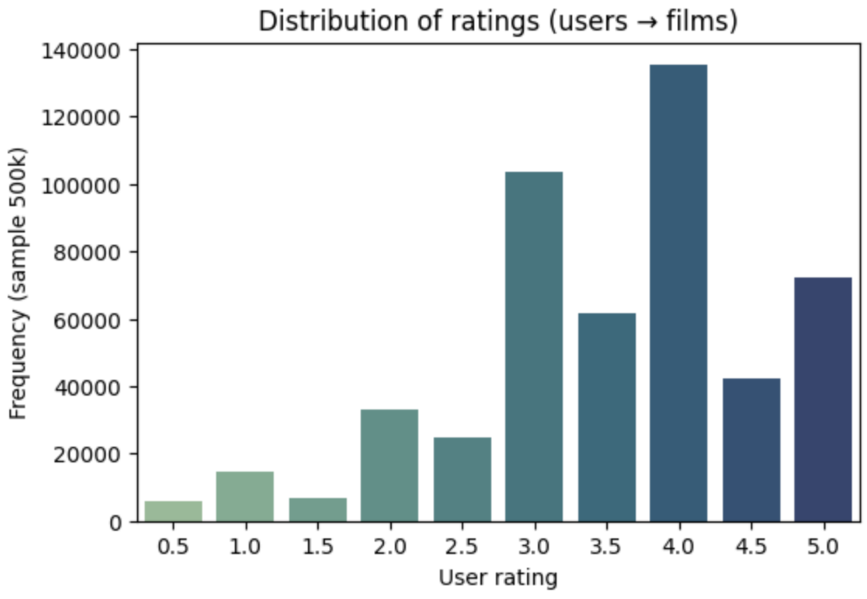


Figure 3: User rating distribution (MovieLens). Ratings are biased towards positive values, with a peak around 4.0. This influenced our normalization strategy.

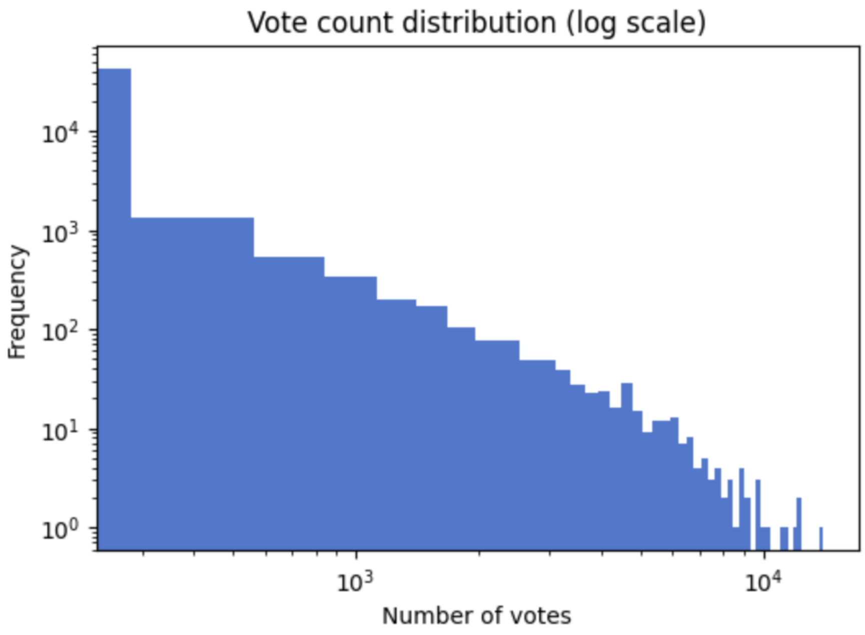


Figure 4: Log-scale histogram of `vote_count`. Most movies received fewer than 10 votes, motivating our threshold-based filtering to improve statistical reliability.

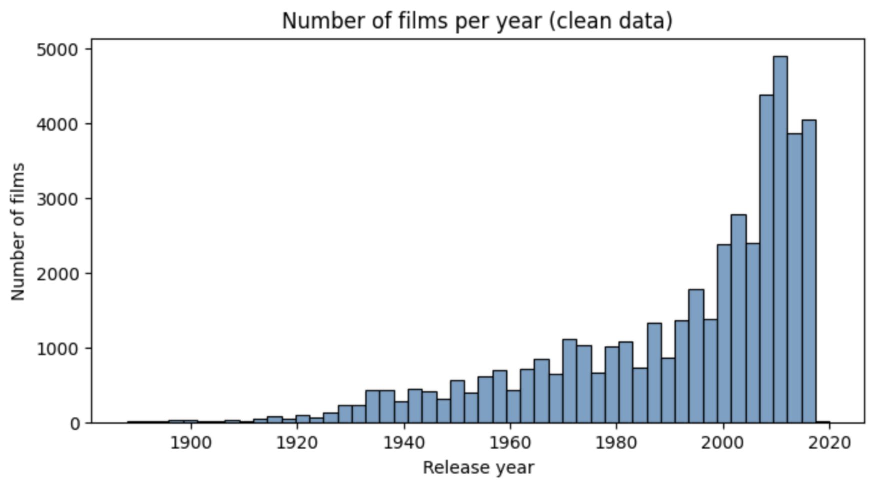


Figure 5: Number of movies by release year. A sharp increase is observed from the 1990s onwards, partially due to better data collection. Early years include many incorrect or missing values.

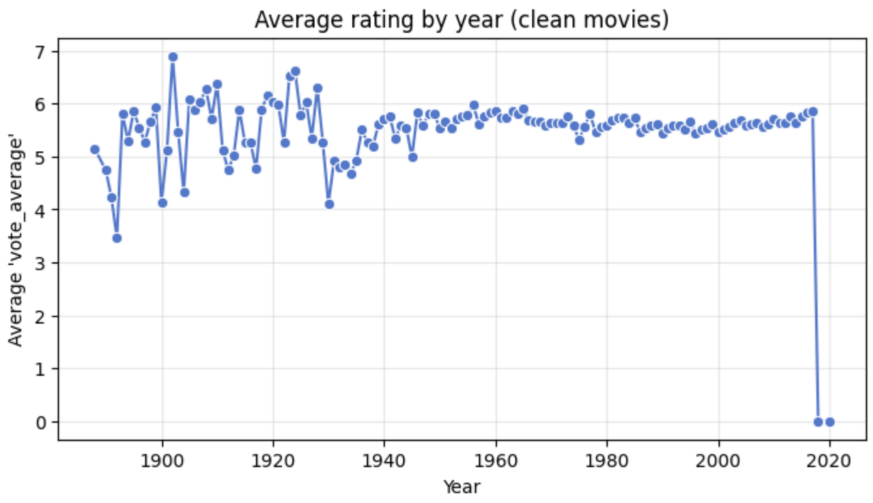


Figure 6: Average rating by release year. Ratings remain relatively stable over time, except for the earliest years (possibly due to low sample size).

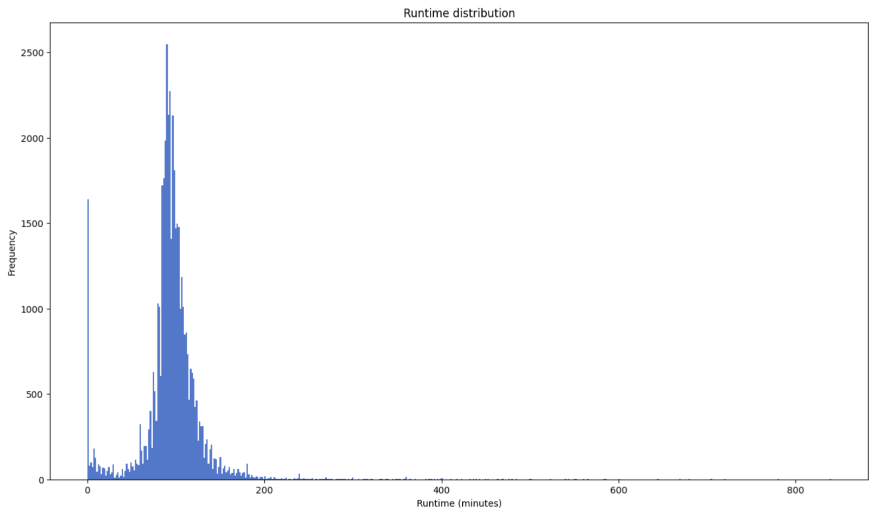


Figure 7: Cleaned runtime distribution. After removing invalid values, most movies last between 80 and 130 minutes.

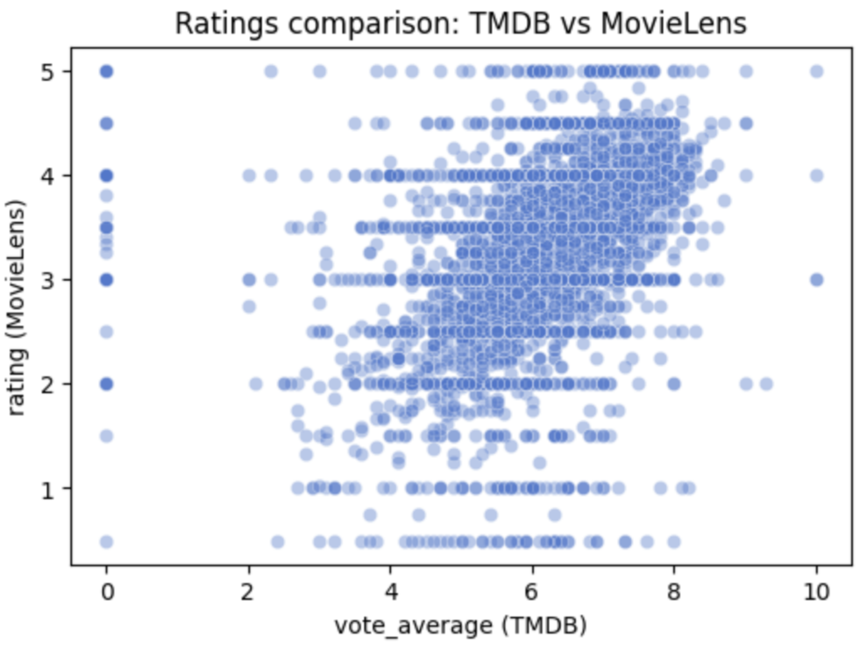


Figure 8: Comparison between `vote_average` (TMDB) and `rating` (MovieLens). While correlated, the two rating sources show slight scale and variance differences.

Further Commentary

While this report summarizes the main insights, it only scratches the surface. The `eda_movies.ipynb` notebook contains:

- All raw and cleaned metrics.
- Additional visualizations (e.g. crew/cast coverage).
- Intermediate code and rationale behind each cleaning rule.
- Interactive filtering trials and threshold testing.

This notebook was our primary analysis and decision-making environment. We strongly recommend inspecting it to fully appreciate the work done and the logic behind each design decision in the pipeline.

4 Filtering Criteria and Justification

Based on the anomalies and patterns discovered during our EDA, we implemented a detailed and rigorous filtering strategy in `filter_movies.py`. This was not a generic cleaning script: each rule was designed with a specific anomaly in mind, ensuring that only high-quality, analytically relevant records were kept. We detail below the filtering rules applied, alongside a rationale for each.

Usage

The filtering script can be executed from the command line to preprocess the raw MovieLens–TMDB dataset before insertion into MongoDB. All input CSV files must be placed in the input directory (default: `data/`), and the cleaned outputs will be written to the output directory (default: `data_clean/`).

```
python filter_movies.py --in-dir data --out-dir data_clean \
    --min-votes 50 --year-min 1900 --year-max 2025 \
    --ratings-keep last --explode-keywords
```

Main command-line options:

- `--in-dir`, `--out-dir`: Input/output directories.
- `--min-votes`: Minimum number of votes required to keep a movie.
- `--year-min`, `--year-max`: Year range filter for valid release dates.
- `--ratings-keep`: How to handle multiple ratings per user–movie pair (`first`, `last`, or `all`).
- `--keep-null-tmdb`: Keep link rows with missing TMDB identifiers.
- `--explode-keywords`: Additionally export an exploded movie–keyword table.

Cleaning Rules Applied

- **Enforcing numeric IDs:** Movies with non-numeric or missing IDs were discarded to ensure correct merging between datasets and indexing in MongoDB.
- **Dropping null or duplicated titles:** Since titles are essential identifiers for human readability, all rows missing a `title` or with duplicated `ids` were dropped.
- **Vote-based filtering:**
  - Movies with `vote_count` lower than a given number of minimum votes (--min-votes) would be excluded to avoid statistical noise, by default it was set to 0.

  - Movies with `vote_average = 0` were also excluded (only if --min-votes > 0), as these are likely unrated or placeholder records.

- **Year filtering:** Only movies with release years between **1888** and **2100** were kept. This range covers the entire known history of cinema while excluding malformed entries.

- **Runtime validation:** Runtimes below 0 or above **873 minutes** (based on the longest movie in the history: *Resan (The Journey)*) were removed or set to `NaN`.

- **Vote average validation:** Entries with `vote_average` outside the range [0, 10] were discarded.

- **Release date validation:** Release dates were parsed using `pd.to_datetime` with error coercion. Movies with null or unparsable dates were discarded.

- **Parsing and normalization of JSON-like columns:** Fields such as `genres`, `production_companies`, `production_countries`, `spoken_languages` and `belongs_to_collection` were parsed safely and re-serialized to ensure uniformity.

- **Deduplication of credits:** From the `credits.csv` file, for each `id`, we retained the row with the largest combined number of `cast + crew`, ensuring the most complete record per movie.

- **Ratings deduplication:** In `ratings.csv`, when multiple ratings existed for the same `(userId, movieId)` pair, we kept only the **latest one**. This was handled in a chunk-wise, memory-efficient manner.

- **Keyword normalization and optional explosion:** The `keywords.csv` entries were normalized to lowercase and stripped. We also generated an `exploded` format (one row per keyword–movie pair) to facilitate further semantic filtering.

Filtering Summary Table

Column	**Filtering Rule**	**Justification**
`id`	Must be numeric and unique	Ensure join consistency and database indexing
`title`	Must not be null	Required for human interpretability
`vote_count`	Must be > --min-votes	Avoid noisy, low-signal records
`vote_average`	Must be in [0, 10] and $\neq 0$	Ensure score reliability and remove dummy values
`release_date`	Must be parseable and within 1888–2100	Remove malformed or implausible entries
`runtime`	Must be in [0, 873] or set to `NaN`	Remove extreme or invalid durations
`credits`	Keep row with max (cast + crew)	Maximize coverage per movie
`ratings`	Keep latest per (userId, movieId)	Avoid duplication and reflect final user opinion
`JSON fields`	Must be parseable and re-serialized	Ensure structural consistency for genres, keywords, etc.

Table 1: Summary of Filtering Criteria Applied in `filter_movies.py`.

5 Data Insertion in MongoDB

Data insertion was performed using the `insertion.py` script, which provides a robust and fully parameterized pipeline for populating MongoDB collections from cleaned CSV files. The script performs defensive parsing (for numbers, dates, JSON fields, and timestamps) and uses batch insertion with fault tolerance to handle duplicate keys and partial write errors gracefully.

Each dataset is mapped to a dedicated collection:

- `movies`: TMDB metadata, using the TMDB `id` as `_id`.
- `credits`: cast and crew lists, keyed by `movie_id`.
- `links`: MovieLens–TMDB mappings.
- `ratings`: user ratings (over 26M rows, inserted in large batches).
- `keywords` and `keywords_exploded`: keyword metadata, kept for potential future relationships.

The script accepts several command-line options, including batch sizes, maximum allowed runtime (`--max-runtime`), and optional collection reset (`--reset`) before insertion. All numeric, date, and JSON fields are validated and normalized (e.g., runtimes capped at 873 minutes, timestamps converted to UTC).

After insertion, compact indexes are created to optimize joins, lookups, and text queries:

- **Movies:** unique on `id`, plus indexes on `title`, `release_date`, and a text index on `overview` and `tagline`.
- **Keywords:** unique on `movie_id`, and text index on `keyword`.
- **Credits:** unique on `movie_id`.
- **Links:** unique on `movieId`, and index on `tmdbId`.
- **Ratings:** indexes on `movieId`, `userId`, their combination, and `timestamp`.

This indexing strategy ensures efficient query performance for filtering, joining, and text-based searches, while maintaining referential consistency across collections.

Note: Although the two keyword collections (`keywords` and `keywords_exploded`) were inserted and indexed, they were **not used in the analytical queries**. Nevertheless, keeping them in the database is useful for several reasons:

- Preserving the complete TMDB vocabulary associated with each movie;
- Enabling potential future joins or content-based filtering (for example, recommendation systems or semantic searches);
- Illustrating how to manage both nested-list and exploded one-to-many data representations in MongoDB.

In summary, they were not required for the current queries but provide valuable extensions for future analyses.

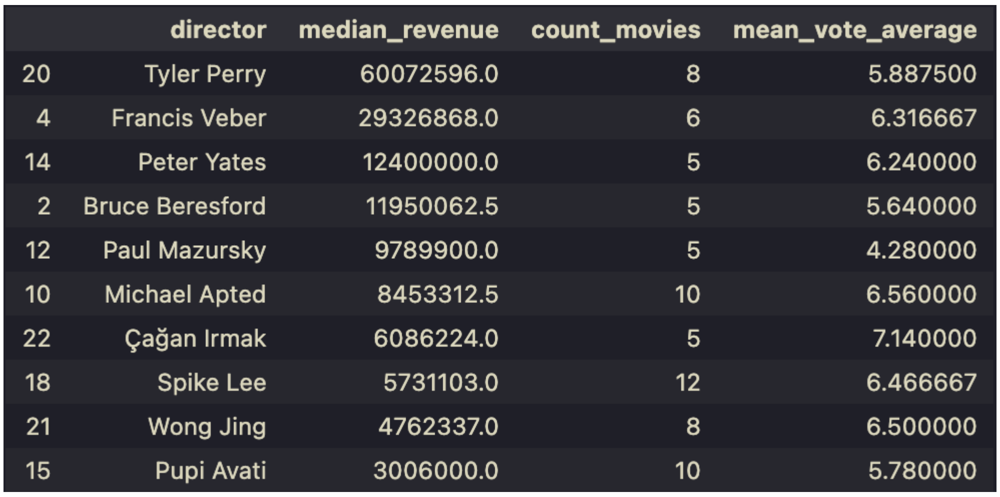
6 Analytical Queries and Aggregations

In this section, we present the analytical queries executed over the MongoDB collections. Each query explores a different analytical dimension of the movies dataset, using aggregation pipelines and custom logic. The code implementations and explanations are detailed in `task2.ipynb`.

Below, we summarize the objective and result of each query, along with the output captured from our execution.

Query 1 – Top Directors by Median Revenue

We selected directors with at least 5 directed movies and computed their median revenue and average rating. This helps identify consistent, commercially successful directors with a sufficient number of works.



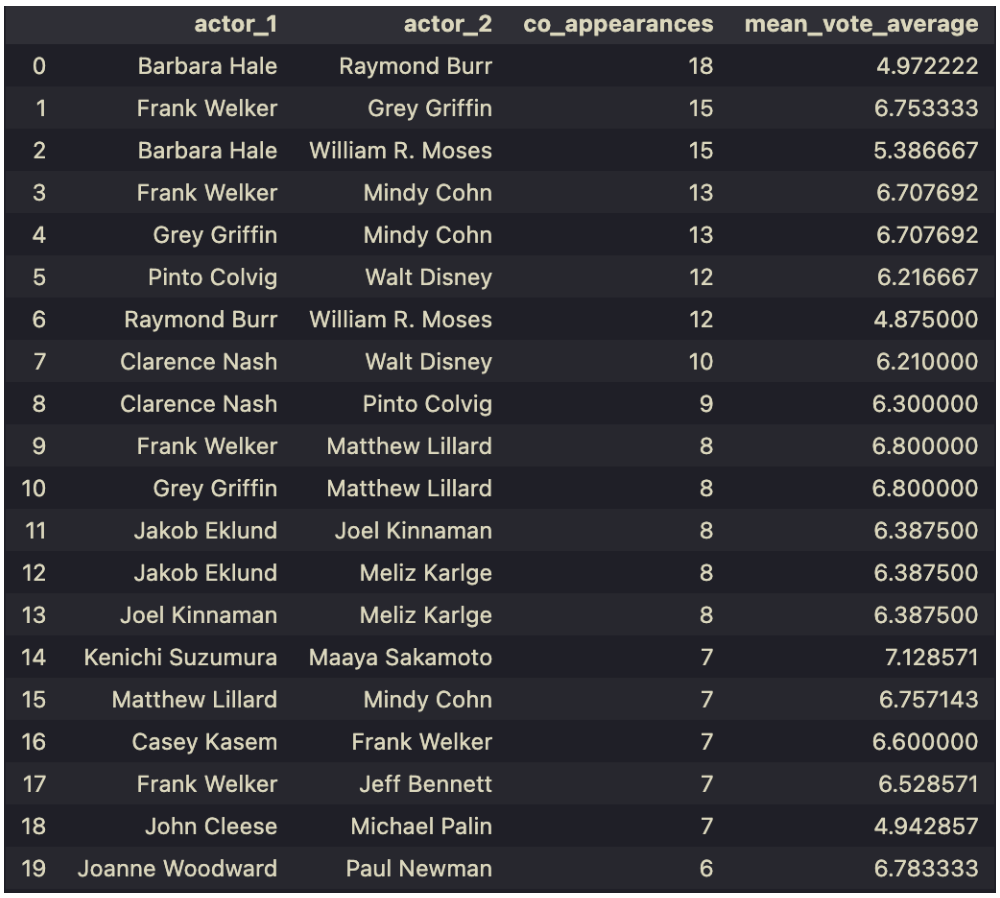
	director	median_revenue	count_movies	mean_vote_average
20	Tyler Perry	60072596.0	8	5.887500
4	Francis Veber	29326868.0	6	6.316667
14	Peter Yates	12400000.0	5	6.240000
2	Bruce Beresford	11950062.5	5	5.640000
12	Paul Mazursky	9789900.0	5	4.280000
10	Michael Apted	8453312.5	10	6.560000
22	Çağan Irmak	6086224.0	5	7.140000
18	Spike Lee	5731103.0	12	6.466667
21	Wong Jing	4762337.0	8	6.500000
15	Pupi Avati	3006000.0	10	5.780000

Figure 9: Top 10 directors $\geq$ 5 movies) by median revenue.

Notably, Tyler Perry and Francis Veber lead the ranking with high revenue consistency, despite varying average ratings.

Query 2 – Actor Pairs with Most Co-Appearances

This query identifies pairs of actors that have co-starred in at least 3 movies together. We computed both their number of shared appearances and the average rating of those films.



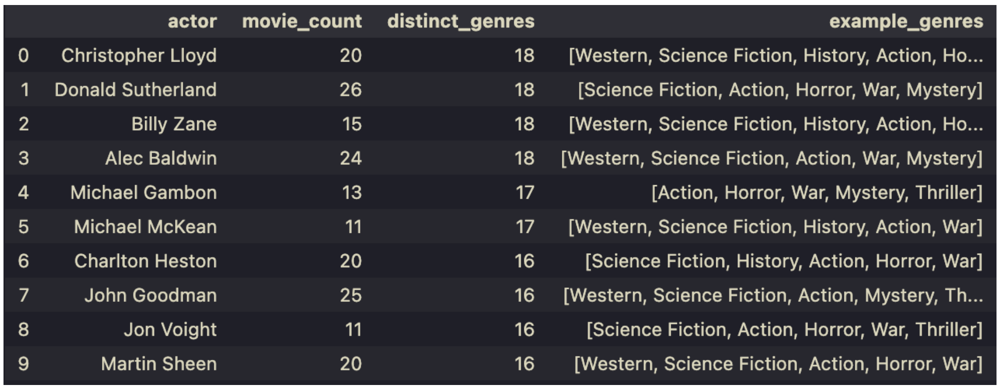
	actor_1	actor_2	co_appearances	mean_vote_average
0	Barbara Hale	Raymond Burr	18	4.972222
1	Frank Welker	Grey Griffin	15	6.753333
2	Barbara Hale	William R. Moses	15	5.386667
3	Frank Welker	Mindy Cohn	13	6.707692
4	Grey Griffin	Mindy Cohn	13	6.707692
5	Pinto Colvig	Walt Disney	12	6.216667
6	Raymond Burr	William R. Moses	12	4.875000
7	Clarence Nash	Walt Disney	10	6.210000
8	Clarence Nash	Pinto Colvig	9	6.300000
9	Frank Welker	Matthew Lillard	8	6.800000
10	Grey Griffin	Matthew Lillard	8	6.800000
11	Jakob Eklund	Joel Kinnaman	8	6.387500
12	Jakob Eklund	Meliz Karlge	8	6.387500
13	Joel Kinnaman	Meliz Karlge	8	6.387500
14	Kenichi Suzumura	Maaya Sakamoto	7	7.128571
15	Matthew Lillard	Mindy Cohn	7	6.757143
16	Casey Kasem	Frank Welker	7	6.600000
17	Frank Welker	Jeff Bennett	7	6.528571
18	John Cleese	Michael Palin	7	4.942857
19	Joanne Woodward	Paul Newman	6	6.783333

Figure 10: Top actor pairs by number of co-starring movies ($\geq$ 3).

The results highlight frequent collaborators, which can indicate strong actor partnerships or recurring roles in franchises.

Query 3 – Genre Breadth per Actor

We retrieved actors with at least 10 credited movies and measured how many distinct genres they have appeared in. This serves as a proxy for actor versatility across different film categories.



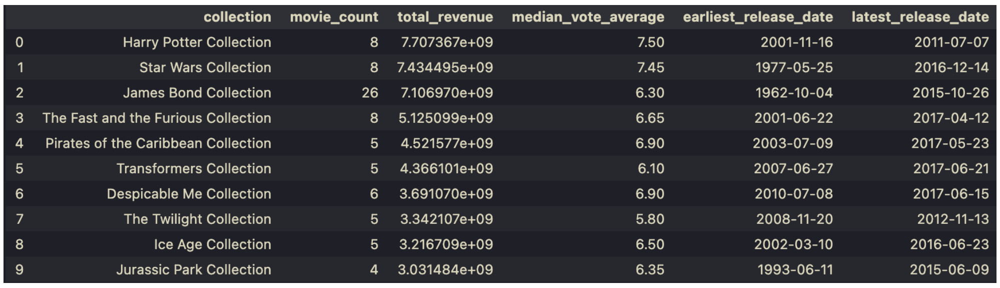
	actor	movie_count	distinct_genres	example_genres
0	Christopher Lloyd	20	18	[Western, Science Fiction, History, Action, Ho...
1	Donald Sutherland	26	18	[Science Fiction, Action, Horror, War, Mystery]
2	Billy Zane	15	18	[Western, Science Fiction, History, Action, Ho...
3	Alec Baldwin	24	18	[Western, Science Fiction, Action, War, Mystery]
4	Michael Gambon	13	17	[Action, Horror, War, Mystery, Thriller]
5	Michael McKean	11	17	[Western, Science Fiction, History, Action, War]
6	Charlton Heston	20	16	[Science Fiction, History, Action, Horror, War]
7	John Goodman	25	16	[Western, Science Fiction, Action, Mystery, Th...
8	Jon Voight	11	16	[Science Fiction, Action, Horror, War, Thriller]
9	Martin Sheen	20	16	[Western, Science Fiction, Action, Horror, War]

Figure 11: Top 10 actors by genre diversity ($\geq$ 10 movies).

Actors like Christopher LLoyd stand out with appearances across a wide range of genres.

Query 4 – Top Revenue Collections

This query evaluates movie collections (`belongs_to_collection`) with at least 3 entries. We ranked them by total revenue and reported additional metrics such as average rating and date range.



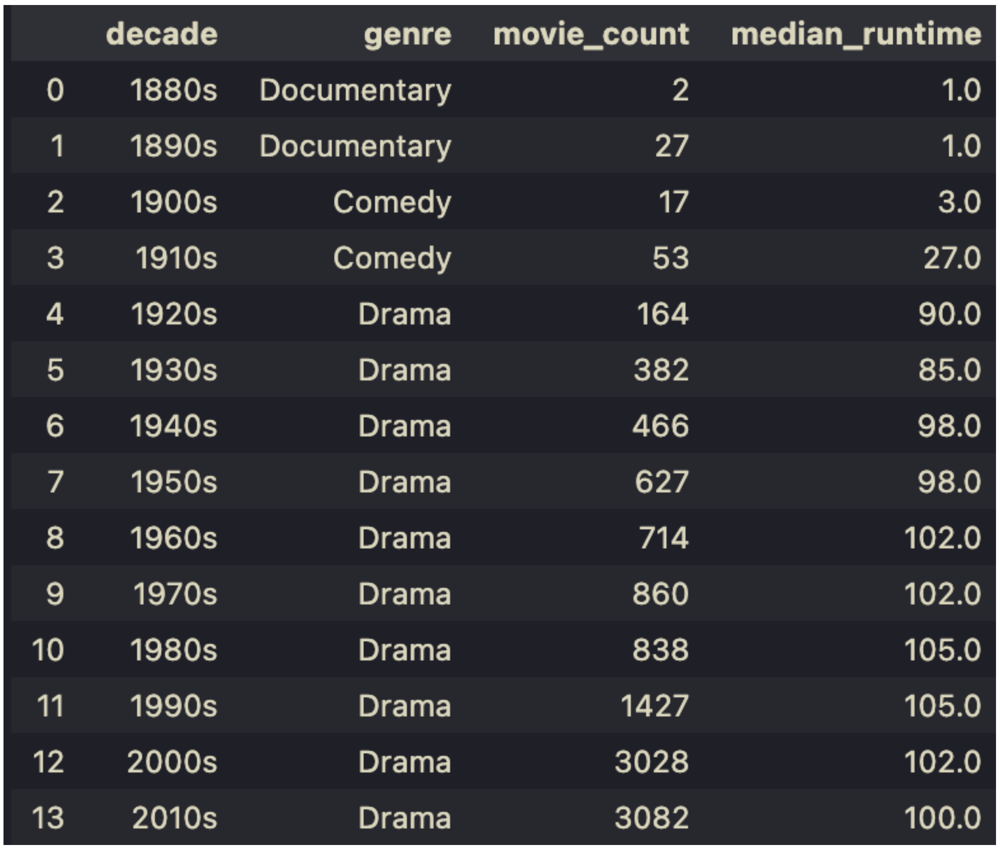
	collection	movie_count	total_revenue	median_vote_average	earliest_release_date	latest_release_date
0	Harry Potter Collection	8	7.707367e+09	7.50	2001-11-16	2011-07-07
1	Star Wars Collection	8	7.434495e+09	7.45	1977-05-25	2016-12-14
2	James Bond Collection	26	7.106970e+09	6.30	1962-10-04	2015-10-26
3	The Fast and the Furious Collection	8	5.125099e+09	6.65	2001-06-22	2017-04-12
4	Pirates of the Caribbean Collection	5	4.521577e+09	6.90	2003-07-09	2017-05-23
5	Transformers Collection	5	4.366101e+09	6.10	2007-06-27	2017-06-21
6	Despicable Me Collection	6	3.691070e+09	6.90	2010-07-08	2017-06-15
7	The Twilight Collection	5	3.342107e+09	5.80	2008-11-20	2012-11-13
8	Ice Age Collection	5	3.216709e+09	6.50	2002-03-10	2016-06-23
9	Jurassic Park Collection	4	3.031484e+09	6.35	1993-06-11	2015-06-09

Figure 12: Top 10 movie collections by total revenue.

Collections like Harry Potter dominate due to their box office impact.

Query 5 – Runtime by Decade and Genre

We computed the median runtime and number of movies per decade and primary genre (first genre listed). This provides insight into how movie lengths vary over time and across genres.



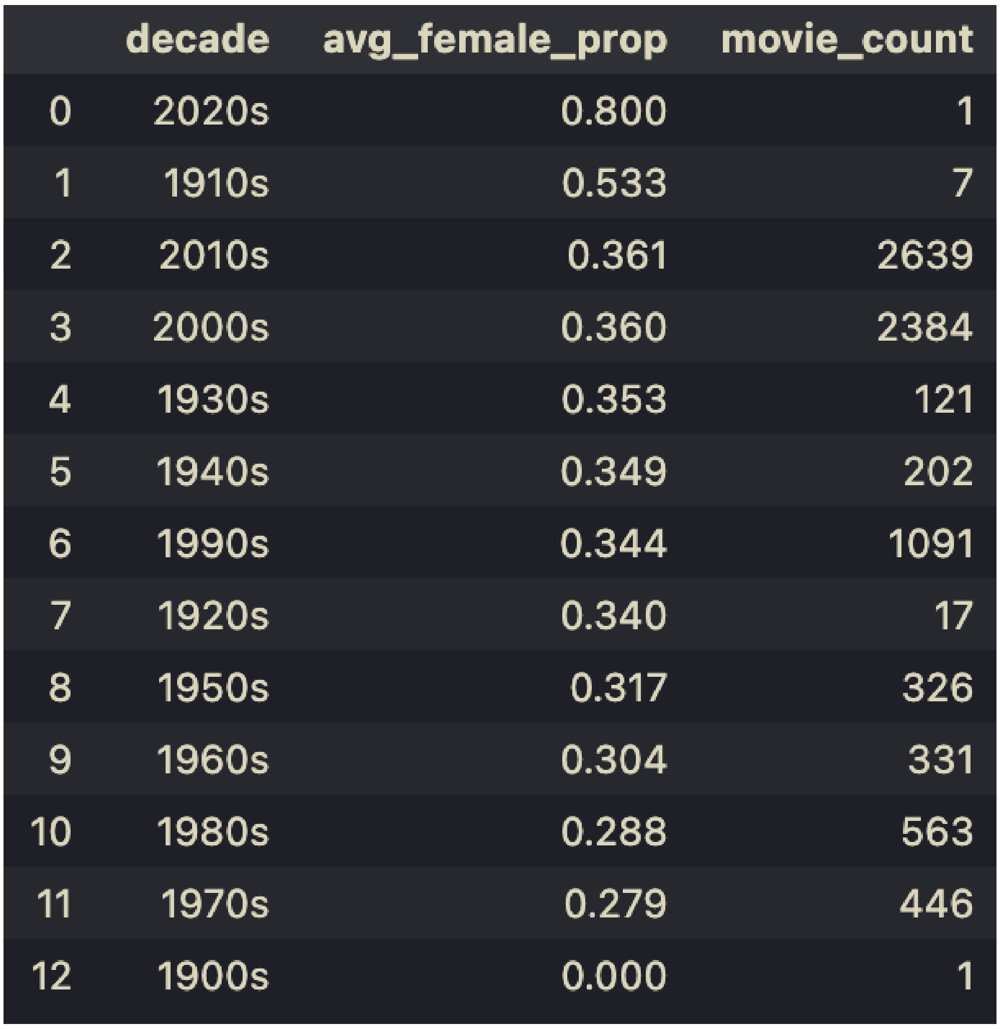
	decade	genre	movie_count	median_runtime
0	1880s	Documentary	2	1.0
1	1890s	Documentary	27	1.0
2	1900s	Comedy	17	3.0
3	1910s	Comedy	53	27.0
4	1920s	Drama	164	90.0
5	1930s	Drama	382	85.0
6	1940s	Drama	466	98.0
7	1950s	Drama	627	98.0
8	1960s	Drama	714	102.0
9	1970s	Drama	860	102.0
10	1980s	Drama	838	105.0
11	1990s	Drama	1427	105.0
12	2000s	Drama	3028	102.0
13	2010s	Drama	3082	100.0

Figure 13: Median runtime and movie count by decade and genre.

Notably, Drama show consistent length increases in modern decades.

Query 6 – Female Cast Proportion by Decade

For each movie's top 5 billed cast members, we computed the proportion of female actors. This was then aggregated by decade to observe gender representation trends.



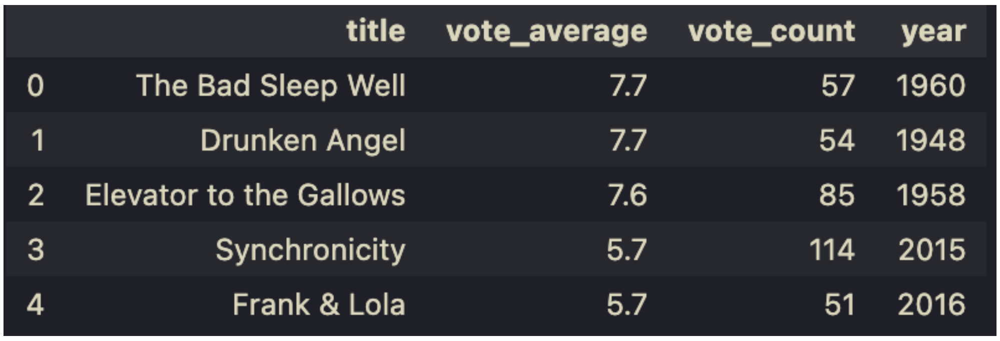
	decade	avg_female_prop	movie_count
0	2020s	0.800	1
1	1910s	0.533	7
2	2010s	0.361	2639
3	2000s	0.360	2384
4	1930s	0.353	121
5	1940s	0.349	202
6	1990s	0.344	1091
7	1920s	0.340	17
8	1950s	0.317	326
9	1960s	0.304	331
10	1980s	0.288	563
11	1970s	0.279	446
12	1900s	0.000	1

Figure 14: Average proportion of female actors (top 5 billed) per decade.

Recent decades show a slow but steady increase in female representation in leading roles.

Query 7 – Noir Text Search in Overviews

We performed a text search over `overview` and `tagline` fields, filtering for movies containing the terms "noir" or "neo-noir" with at least 50 votes.



	title	vote_average	vote_count	year
0	The Bad Sleep Well	7.7	57	1960
1	Drunken Angel	7.7	54	1948
2	Elevator to the Gallows	7.6	85	1958
3	Synchronicity	5.7	114	2015
4	Frank & Lola	5.7	51	2016

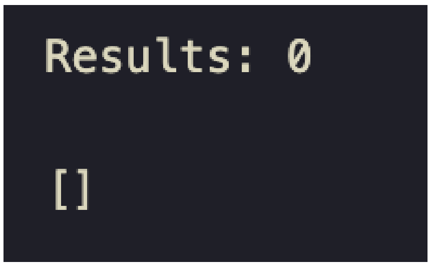
Figure 15: Top 20 noir or neo-noir movies by rating.

The results include both classic and modern noir entries, confirming the relevance of keyword-based filtering.

Query 8 – Director–Actor Collaboration Success

This query aimed to identify director–actor pairs who have worked together on at least three movies with significant audience engagement ($\geq$ 100 votes each). For each pair, we would compute their number of collaborations, average rating, and mean revenue. Unfortunately, after enforcing the conditions specified in the assignment, **no pairs in the dataset satisfy all criteria simultaneously**.

This highlights the sparsity of high-vote, recurring collaborations in our current dataset. It may also suggest that filtering thresholds (like 'vote count $\geq$ 100' and '$\geq$ 3 collaborations') are too strict to yield results from te cleaned corpus.



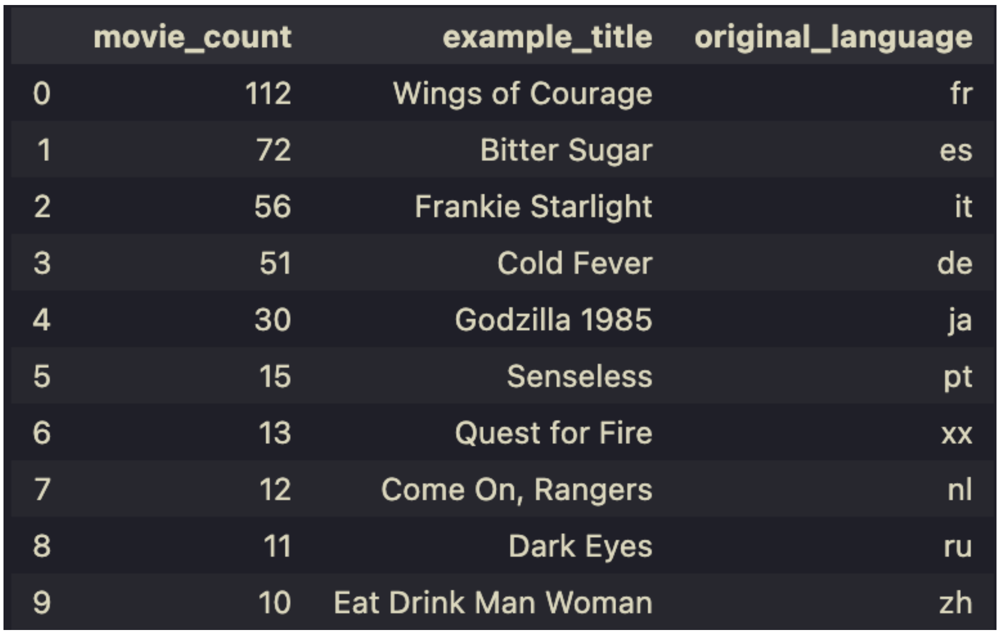
```
Results: 0

[]
```

Figure 16: Probe result without the 3-collaboration condition. No valid pairs were found under full constraints.

Query 9 – Foreign Language Movies with U.S. Ties

Among non-English movies, we identified those with production companies or countries associated with the United States. The query ranks the most frequent original languages under these conditions.



	movie_count	example_title	original_language
0	112	Wings of Courage	fr
1	72	Bitter Sugar	es
2	56	Frankie Starlight	it
3	51	Cold Fever	de
4	30	Godzilla 1985	ja
5	15	Senseless	pt
6	13	Quest for Fire	xx
7	12	Come On, Rangers	nl
8	11	Dark Eyes	ru
9	10	Eat Drink Man Woman	zh

Figure 17: Top 10 original languages ($\neq$ English) with U.S. production involvement.

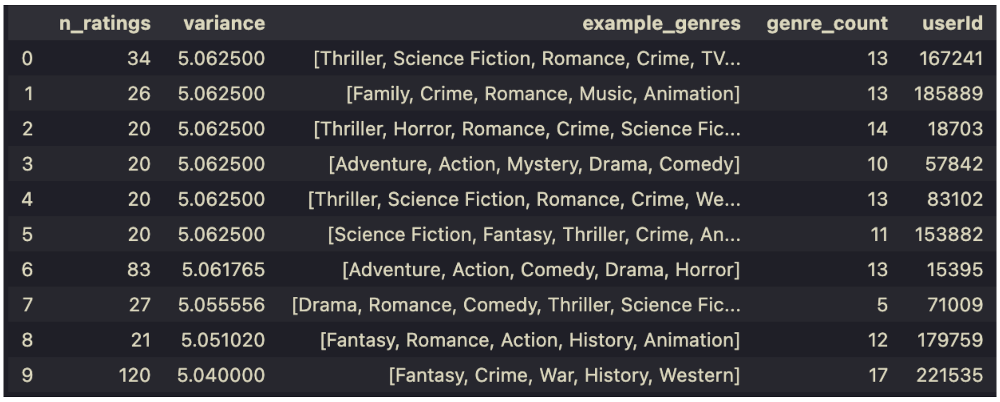
Languages like Spanish and French dominate among non-English U.S.-influenced films.

Query 10 – Genre Diversity and Rating Variance per User

For users with at least 20 ratings, we computed:

- Total ratings count
- Population variance of ratings
- Number of distinct genres rated

We then ranked the top 10 most genre-diverse users and the top 10 highest-variance users.



	n_ratings	variance	example_genres	genre_count	userId
0	34	5.062500	[Thriller, Science Fiction, Romance, Crime, TV...	13	167241
1	26	5.062500	[Family, Crime, Romance, Music, Animation]	13	185889
2	20	5.062500	[Thriller, Horror, Romance, Crime, Science Fic...	14	18703
3	20	5.062500	[Adventure, Action, Mystery, Drama, Comedy]	10	57842
4	20	5.062500	[Thriller, Science Fiction, Romance, Crime, We...	13	83102
5	20	5.062500	[Science Fiction, Fantasy, Thriller, Crime, An...	11	153882
6	83	5.061765	[Adventure, Action, Comedy, Drama, Horror]	13	15395
7	27	5.055556	[Drama, Romance, Comedy, Thriller, Science Fic...	5	71009
8	21	5.051020	[Fantasy, Romance, Action, History, Animation]	12	179759
9	120	5.040000	[Fantasy, Crime, War, History, Western]	17	221535

Figure 18: Top users by genre diversity and rating variance ($\geq$ 20 ratings).

These users demonstrate either exploratory behavior or extreme preference polarization.

7 Discussion

This assignment highlighted how document-oriented design changes the way analytical questions are expressed. Many operations that would be short SQL `JOIN`s become multi-stage pipelines requiring careful control of payload size (`$project`), fan-out (`$unwind`), and join selectivity (`$lookup` with sub-pipelines). Indexing the relevant keys and, when appropriate, materializing compact intermediate results were essential to maintain performance at scale.

An important change from Assignment 2 was the move from a virtual machine to a **Docker**-based workflow. At the beginning there was a learning curve understanding images, containers, and bind-mounted volumes, as well as configuring the VS Code dev container correctly. Once the setup was established, Docker proved more flexible and efficient than the previous VM: dependencies were isolated, environments were identical across the team, and rebuilding from scratch became straightforward. This increased reliability and reduced friction relative to the VM-based workflow used previously.

Collaboration benefited from the shared database with per-user credentials, allowing independent execution without overwriting each other's work. The combination of notebooks for analysis and scripts for repeatable pipelines made the process transparent and easy to audit.

8 Feedback

The assignment offered practical exposure to large-scale data processing in a NoSQL environment. The shift from relational to document-based thinking was valuable, particularly in understanding when denormalization and materialization are appropriate. The Docker-based setup ultimately improved productivity and reproducibility compared to the earlier VM approach; after an initial adaptation period, it provided a cleaner and more controlled development experience.

For future iterations, smaller pre-cleaned samples or early guidance on aggregation performance patterns could help students focus faster on query design. Overall, the assignment met its objectives and provided a realistic view of analytical work on very large, nested datasets using MongoDB.