Final Solution Report

1. Introduction

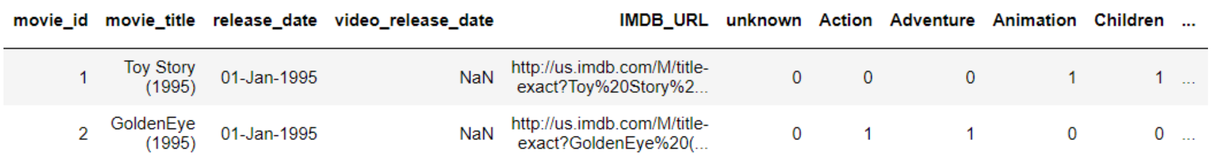
Our objective is to develop a Movie Recommender System. This type of system is a specialized form of information filtering that suggests items or content to users based on their interests, preferences, or past behavior. Recommender systems are widely used across various domains, including e-commerce, entertainment, social media, and online content platforms.

In this context, our goal is to predict the most suitable films for a specific user. To achieve this, we utilize a dataset that includes user profiles, their ratings for various films, and detailed information about the films.

The development of our solution involves two key steps. First, we conduct thorough data preprocessing, which includes removing unnecessary columns and performing appropriate encodings. Second, we train a suitable model tailored to our task. This two-step process ensures that our Movie Recommender System is both efficient and effective in predicting films that align with a user's preferences.

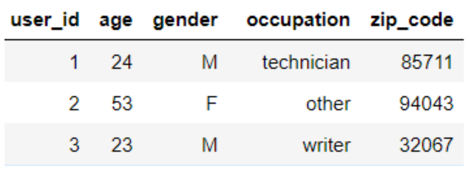
2. Data analysis

Let's delve deeper into our collected datasets from the first notebook. We have three datasets: one with movies and their associated information, another with users and their demographic data, and a third with a multitude of users, movies, their ratings, and the time of rating. The movie dataset, for instance, looks like this(not full, because there are more genres):



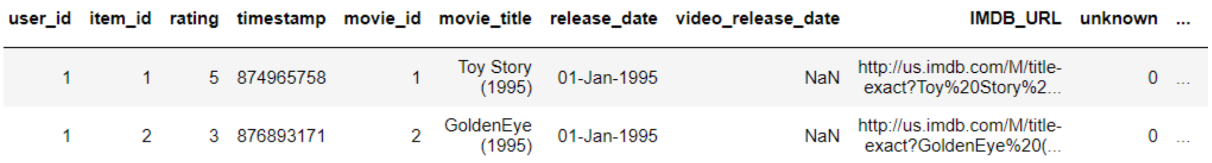
movie_id	movie_title	release_date	video_release_date	IMDB_URL	unknown	Action	Adventure	Animation	Children	...
1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	0	0	1	1	...
2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	1	1	0	0	...

The users dataset looks that way:



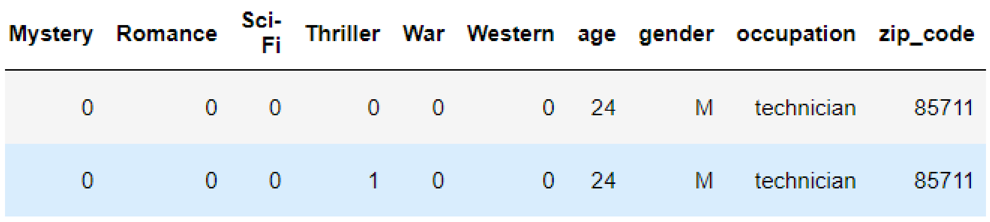
user_id	age	gender	occupation	zip_code
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067

After merging all the datasets, we obtain a comprehensive view as follows:



user_id	item_id	rating	timestamp	movie_id	movie_title	release_date	video_release_date	IMDB_URL	unknown	...
1	1	5	874965758	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	...
1	2	3	876893171	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	...

(part 1)



Mystery	Romance	Sci-Fi	Thriller	War	Western	age	gender	occupation	zip_code
0	0	0	0	0	0	24	M	technician	85711
0	0	0	1	0	0	24	M	technician	85711

(part 2)

After combining all the data, we can now proceed to the preprocessing stage. As observed, certain columns contain null values:

```
video_release_date    100000
IMDB_URL                  13
```

We've streamlined our dataset by removing columns that aren't beneficial for our analysis. The "item_id" column was used for merging data and isn't useful for our model, so we've removed it. The "timestamp" column, which indicates when a user rated a film, isn't crucial for our purposes, so it's been dropped. The "video_release_date" column, which contained all null values, has been discarded. The "IMDB_URL" column, while helpful for locating a film, doesn't contribute to our model, so we've omitted it. We've also removed the "movie_title" column. While it's useful for human readability, it doesn't contribute to our machine learning model. Instead, we can use the already created "movie_id" which serves as a unique identifier for each movie. Lastly, the "zip_code" column was removed because it contains unique information that doesn't correlate well in machine learning. After these adjustments, our dataset no longer contains any null values.

In terms of encoding, we need to transform our textual data into a numerical format that our model can understand. For the "occupation" column, which has more than 20 unique values, we use ordinal encoding to avoid overloading the dataset with too many new columns. For the "gender" column, which only has two unique values, we apply one-hot encoding, creating a new binary column for each gender. Lastly, we replace the "release_date" column with a "release_year" column, simplifying the date information to just the year. These steps ensure our dataset is appropriately formatted for machine learning analysis. This is how it looks like:

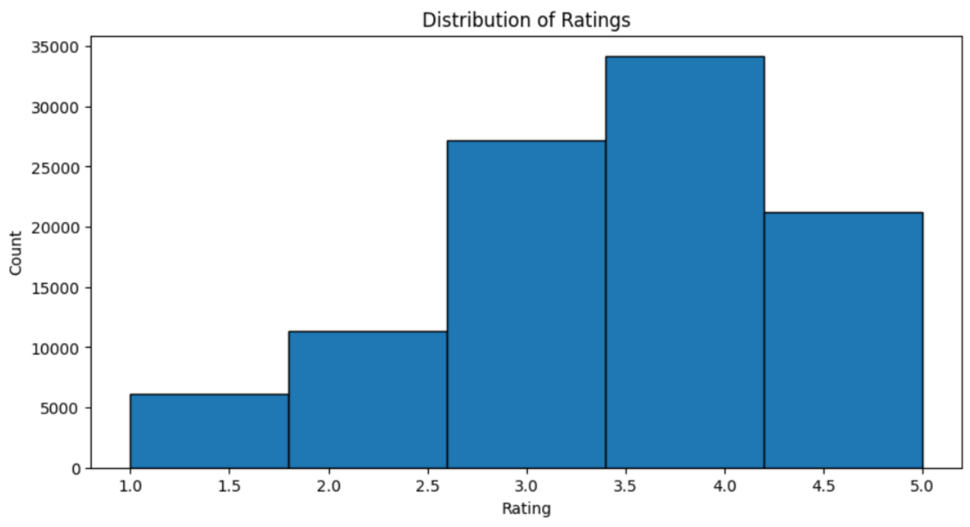
user_id	rating	movie_id	unknown	Action	Adventure	Animation	Children	Comedy	Crime	...
1	5	1	0	0	0	1	1	1	0	...
1	3	2	0	1	1	0	0	0	0	...

(part 1)

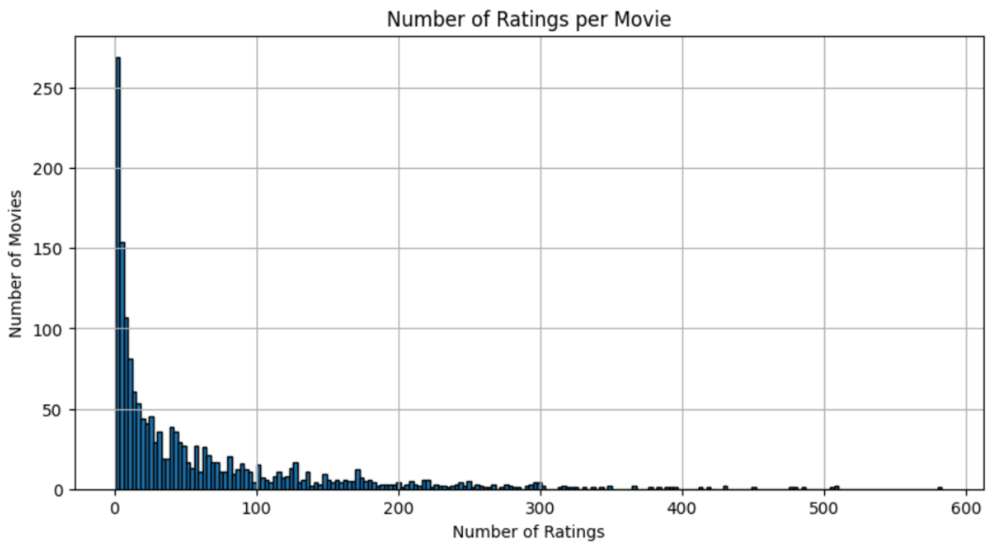
Romance	Sci-Fi	Thriller	War	Western	age	occupation	gender_F	gender_M	release_year
0	0	0	0	0	24	19.0	0	1	1995
0	0	1	0	0	24	19.0	0	1	1995

(part 2)

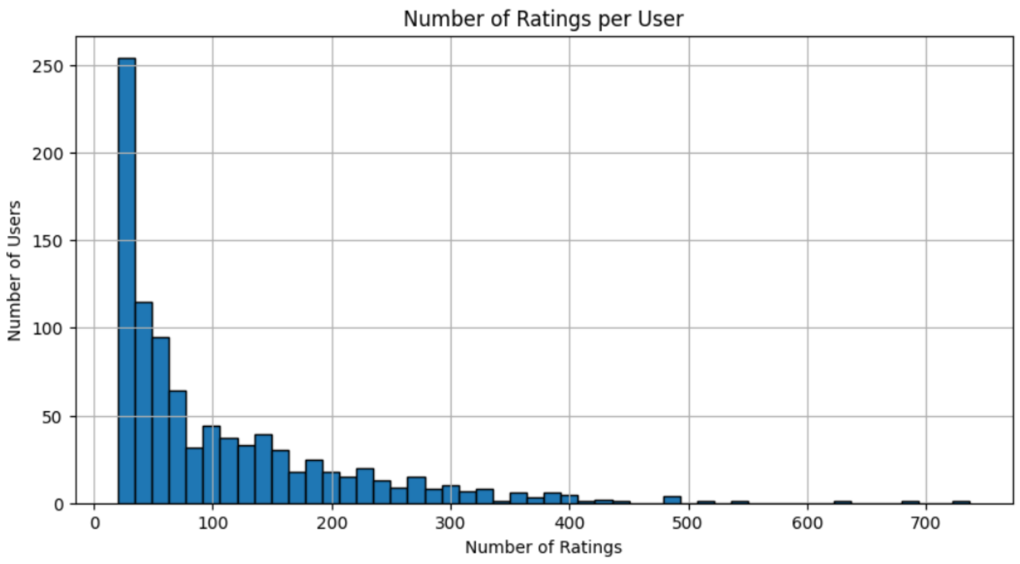
We can extract some valuable insights from our data for our analysis. One aspect that could be intriguing is the general trend of ratings given by users. For instance, the following graph illustrates the distribution of ratings across all films, as rated by all users:



We observe that a rating of 4 is the most common, followed by ratings of 3 and 5. Ratings of 1 and 2 are considerably less frequent in comparison to the others. Also on this graph we can see number of ratings per movie:



We observe that a significant number of films (over 250) have received very few ratings, which is not ideal for our analysis. In general, most films have fewer than 100 ratings. Only a small proportion of films have received more than 100 ratings. This distribution indicates that our dataset is skewed towards films with fewer ratings. On the next graph we see the distribution of the number of ratings per user:

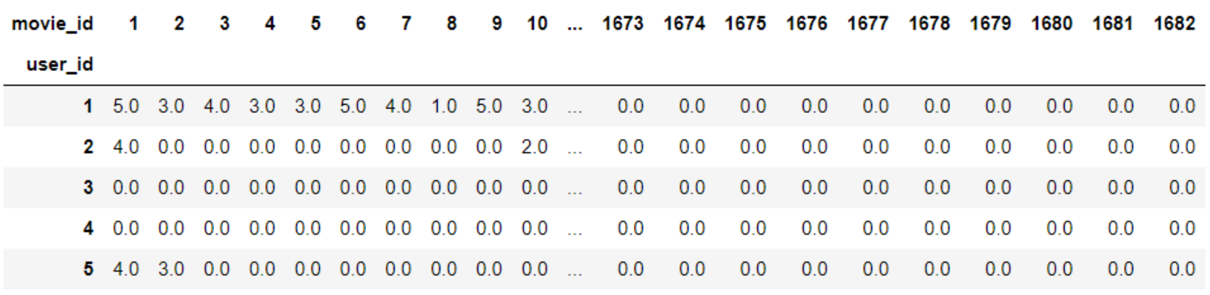


The graph reveals that a large number of users have approximately 20 ratings. The majority of users have a rating count that doesn't exceed 100. A slightly smaller group falls within the 100 to 200 rating range, and an even smaller group ranges from 200 to 300 ratings. Only a relatively small number of users have a rating count that exceeds 300. This distribution provides a good collection for creating recommendation ML model.

Now we are now set to proceed to the exciting phase of building our model.

3. Model implementation and training

Our goal is to identify movies that a user might find interesting. To achieve this, we'll construct a model that recommends movies similar to a selected movie, based on the ratings of all users. We'll create a matrix, referred to as the "user_item_matrix", which displays the rating given by each user to each movie. If a user hasn't watched a particular movie, we'll assign a rating of 0. This matrix, which forms the bedrock of our recommendation system, is structured as follows:



movie_id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676	1677	1678	1679	1680	1681	1682
user_id																					
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	4.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

After creation of the user_item_matrix, we then transpose this matrix to create the movie_user_matrix. This is done because we want to recommend movies based on their similarity to a selected movie. By transposing the matrix, we can now compare movies (which are now represented by rows) based on the ratings they received from all users.

Next, we create a mapping from movie_id to matrix row index. This will be useful later when we want to locate a specific movie in our matrix.

We then convert our DataFrame of movie features to a scipy sparse matrix using the csr_matrix function. This is a space-efficient way to represent our matrix, especially considering that our matrix will likely be very sparse (i.e., contain a lot of zeros) because not all users will have watched all movies.

We then define our model using the NearestNeighbors function from the sklearn library. We use the 'cosine' metric because it is a good measure of similarity when dealing with high-dimensional data. We use the 'brute' algorithm because it computes the distances between all pairs of points in the dataset, which is what we want for our recommendation system. We set n_neighbors=10 because we want to recommend the 5-10 most similar movies, and n_jobs=-1 to use all processors to speed up the fitting process.

After defining our model, we fit it to our data using the fit method.

Finally, we define a function movie_recommender_engine that takes a movie_id, the matrix, the fitted model, and the number of recommendations (n_recs) as inputs. This function calculates the distances and indices of the n_recs nearest neighbors to the selected movie in the matrix. It then returns the indices of the recommended movies. Note that we add 1 to the indices because Python indexing starts at 0, but our movie_id starts at 1. We also exclude the first index because it is the movie_id itself.

It's important to note that we don't partition the dataset into training and testing sets. This is because our task requires comprehensive information about all users within the system. Additionally, our data doesn't contain predefined target values; instead, we are tasked with uncovering these values ourselves.

4.Model advantages and disadvantages

The advantages of this method include:

1. Personalized Recommendations: Offers tailored suggestions based on user behavior, leading to highly customized experiences.
2. Diverse Content Discovery: Capable of recommending a wide range of items, helping users discover content they might not find on their own. It gives diverse content discovery the edge over content-based filtering.
3. Community Wisdom: Leverages the collective preferences of users, often leading to more accurate recommendations than individual or content-based analysis alone.
4. Dynamic Adaptation: The model continuously gets updated with user interactions, keeping the recommendations relevant and up-to-date.

However, there are also some disadvantages to consider:

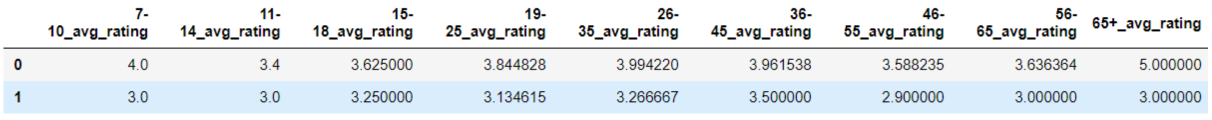
1. Cold Start Problem: New users or new items in the system that have no ratings are hard to recommend because there's no historical data to base the recommendations on.

2. Scalability: As the number of users and items grows, the computational complexity of the model also increases, making it harder to generate recommendations quickly.
3. Sparsity: The user-item interactions matrix is usually very sparse, meaning most users have not interacted with most items. This can make it hard to find similar items or users and can lead to less accurate recommendations.
4. Popularity Bias: The model tends to recommend popular items, which can lead to a lack of diversity in the recommendations and make it harder for less popular or new items to be recommended.

5. Recommendation metric creation

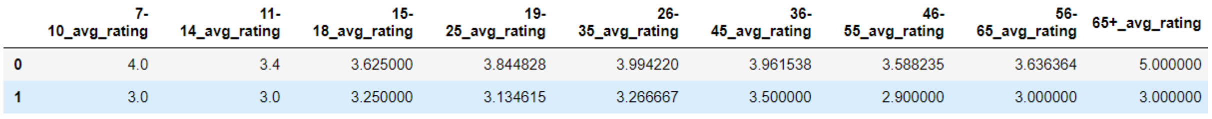
While it's possible to find metrics for our problem on the internet, I thought it would be more engaging to craft them myself. Although we could leverage this metric to identify top-rated movies without resorting to machine learning models (which would contradict the problem statement), I acknowledge that my metric may be somewhat slow and inefficient for a large dataset. However, in this instance, I'm keen on utilizing it.

We'll begin by incorporating the average movie rating for each gender, age group, and occupation. This approach can provide us with a portion of the rating when a user falls into a specific group. Here's a glimpse of our newly added columns:



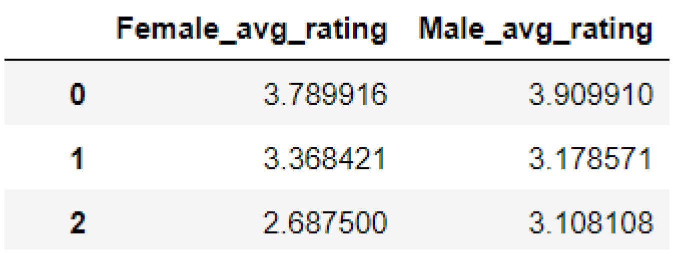
	7-10_avg_rating	11-14_avg_rating	15-18_avg_rating	19-25_avg_rating	26-35_avg_rating	36-45_avg_rating	46-55_avg_rating	56-65_avg_rating	65+_avg_rating
0	4.0	3.4	3.625000	3.844828	3.994220	3.961538	3.588235	3.636364	5.000000
1	3.0	3.0	3.250000	3.134615	3.266667	3.500000	2.900000	3.000000	3.000000

(age group columns)



	7-10_avg_rating	11-14_avg_rating	15-18_avg_rating	19-25_avg_rating	26-35_avg_rating	36-45_avg_rating	46-55_avg_rating	56-65_avg_rating	65+_avg_rating
0	4.0	3.4	3.625000	3.844828	3.994220	3.961538	3.588235	3.636364	5.000000
1	3.0	3.0	3.250000	3.134615	3.266667	3.500000	2.900000	3.000000	3.000000

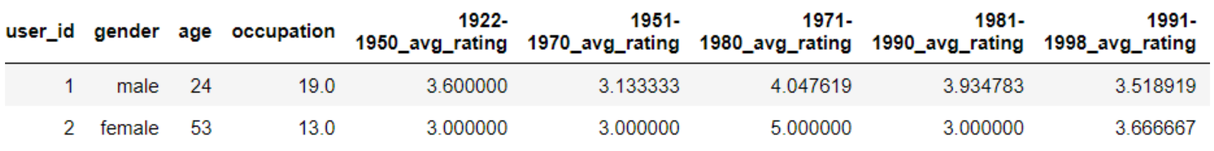
(occupation group columns)



	Female_avg_rating	Male_avg_rating
0	3.789916	3.909910
1	3.368421	3.178571
2	2.687500	3.108108

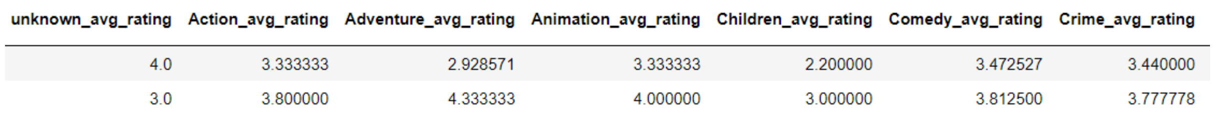
(gender group columns)

Next, we construct a new dataset that includes the user_id, along with their average movie ratings for each release year group and each genre. Here’s a snapshot of how our enriched dataset appears:



user_id	gender	age	occupation	1922-1950_avg_rating	1951-1970_avg_rating	1971-1980_avg_rating	1981-1990_avg_rating	1991-1998_avg_rating
1	male	24	19.0	3.600000	3.133333	4.047619	3.934783	3.518919
2	female	53	13.0	3.000000	3.000000	5.000000	3.000000	3.666667

(release year ratings)



unknown_avg_rating	Action_avg_rating	Adventure_avg_rating	Animation_avg_rating	Children_avg_rating	Comedy_avg_rating	Crime_avg_rating
4.0	3.333333	2.928571	3.333333	2.200000	3.472527	3.440000
3.0	3.800000	4.333333	4.000000	3.000000	3.812500	3.777778

(all genres ratings)

At this point, I’ve devised a function that estimates a movie’s rating for a specific user. This function considers both the user’s individual preferences and the preferences of other users, thereby offering a personalized prediction of the movie ranking. The final score for a given user is computed using the following formula:

movie rating = 0.3 * genre part + 0.2 * gender part + 0.2 * age part + 0.2 * profession part + 0.1 * year part

As you can see, the rating is composed of 40% of the user’s personal preferences, while the remaining 60% represents the average profile of users who share similar demographics with our user. This approach ensures a balanced blend of personal taste and community wisdom in our recommendations.

Final metric will be calculated during the evaluation part.

6. Evaluation

In our approach, we generate personalized movie recommendations for each user in our dataset. Our model identifies 5 movies that align with the user’s preferences and viewing history.
Here’s how it works: we begin by identifying the top 4 favorite movies (as rated by users) and suggest 20 good movies (5 similar movies for each movie). We then eliminate watched and repeated movies and randomly select 5 of the most recent ones. If the number of movies is less than 5, we simply supplement with a random movie from the unwatched ones.

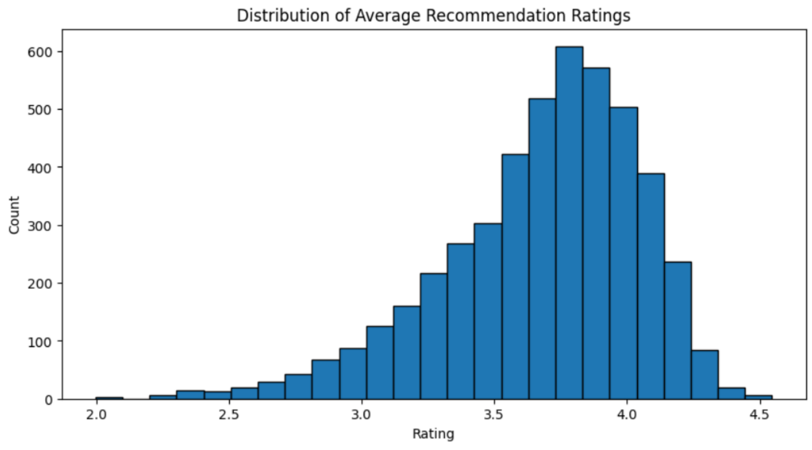
Here’s a snapshot of how our model generates recommendations for a specific user:

```
Film suggested for user_9: ['Return of the Jedi (1983)', 'Top Gun (1986)', 'Speed (1994)', 'Nightmare on Elm Street, A (1984)',
'Interview with the Vampire (1994)']

Film suggested for user_90: ['Piano, The (1993)', 'Back to the Future (1985)', 'Bullets Over Broadway (1994)', 'Empire Strikes
Back, The (1980)', 'Princess Bride, The (1987)']

Film suggested for user_900: ['Heat (1995)', 'Pulp Fiction (1994)', 'Silence of the Lambs, The (1991)', 'Star Wars (1977)', 'On
ce Upon a Time in the West (1969)']
```

To evaluate the quality of our recommendations, we first calculate the average rating of our pre-recommended films. We also calculate the average rating of the 5 best and worst films for all users and calculate the average of them. The effectiveness of our recommendations is assessed using the following formula: goodness_score = (average_recommendation_rating - worst_recommendation_rating) / (best_recommendation_rating - worst_recommendation_rating)
This formula gives us a normalized score that reflects the quality of our recommendations relative to the best and worst possible recommendations. Our final metrics looks following way:

```
Average recommendation rating: 3.6770703346535907
Recommendations goodness_score: 0.7215713411986324
```

The following graph provides a visual representation of the distribution of our average recommendation ratings:



7. Results

In wrapping up, we’ve made significant progress in the field of movie recommendations. The additional resources and code have played a crucial role in shaping our approach. Furthermore, we’ve succeeded in refining and enhancing numerous aspects of our features and code.

While our results might not be perfect, they are certainly noteworthy: achieving a quality score of 72% is a considerable accomplishment. Moreover, the data shows that most movies received a subjective rating of over 3.5, which is quite impressive.

Even though other machine learning models might offer better outcomes for this problem, our current methods have proven to be effective. It’s crucial to recognize that our evaluation metric, while useful, isn’t flawless and has room for enhancement. It considers a wide range of preferences, including both the individual’s personal tastes and the broader preferences of the users. However, it’s essential to bear in mind that these preferences are fluid and can evolve over time.