# Machine Learning

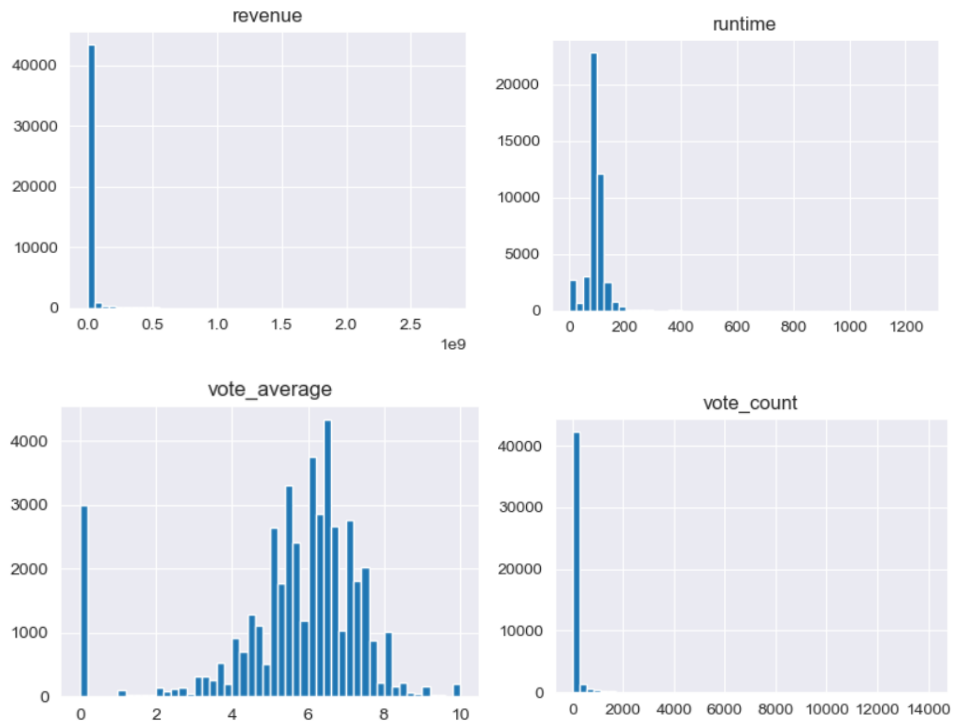
## Final Project : Movie Recommendation System

### Movie EDA

For EDA and Feature Engineering of the file `movies_metadata.csv`, a separate notebook was created.

First, thelibraries were imported, and then the data was analyzed.

Using histograms, the distribution of the numerical columns in the `movie` dataframe (`revenue`, `runtime`, `vote_average`, `vote_count`) was examined.




After that, helper JSON functions were used:

- parse_json_safe:
  This function converts a text into a Python list or dictionary, but also handles errors safely to prevent issues.
- extract_single_value:
  This function is used to extract values from a Python dictionary.
- extract_text_value:
  This function is used when the input text is either a list of dictionaries or a single dictionary.

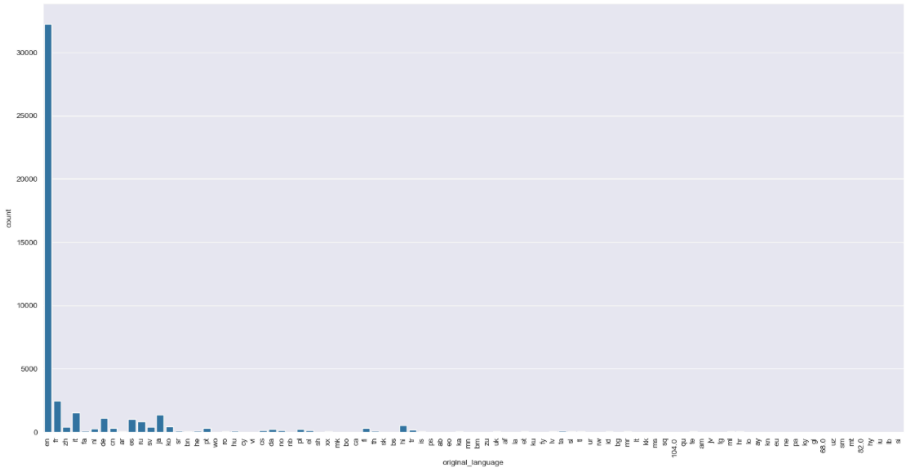
Using the `extract_single_value` function, the keys `name`, `id`, `poster_path`, and `backdrop_path` were extracted from the column `belongs_to_collection`.
Using the `extract_text_value` function, the keys were extracted as follows:

- `name` and `id` from `genres`,
- `name` and `iso_3166_1` from `production_countries`,
- `name` and `id` from `production_companies`.

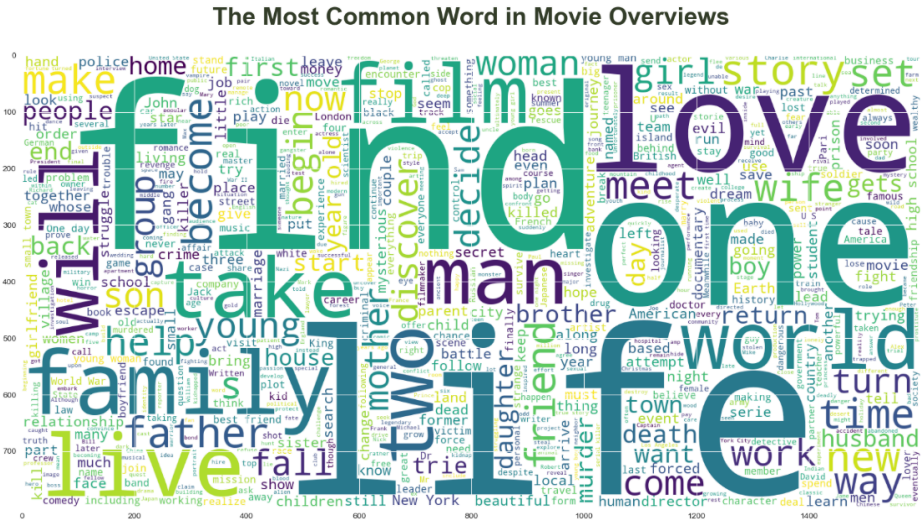
Each of these was converted into a new column in the `movie` dataframe.
Then, the columns `production_countries`, `production_companies`, `genres`, `spoken_languages`, `homepage`, and `belongs_to_collection` were dropped from the dataframe.

The following plots were created:

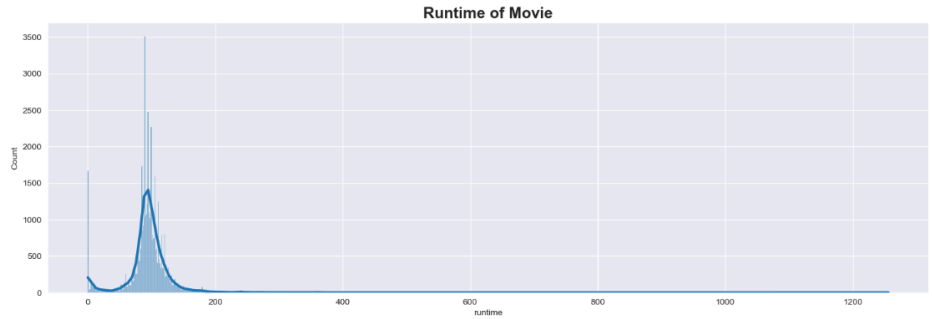
- Plot of the distribution of movie languages.



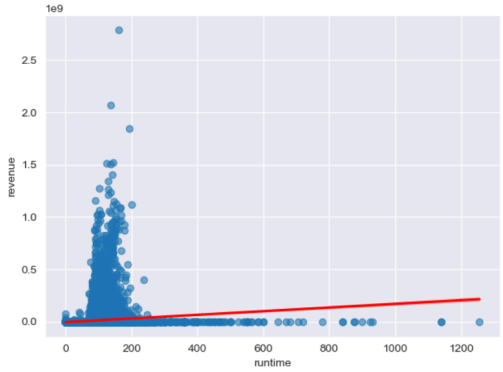

- Plot of the most frequent words in movie overviews.



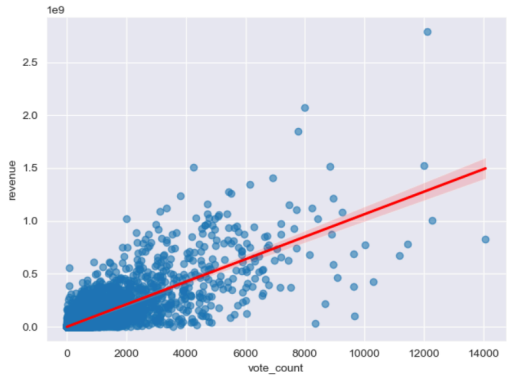
- Plot of movie runtime distribution.



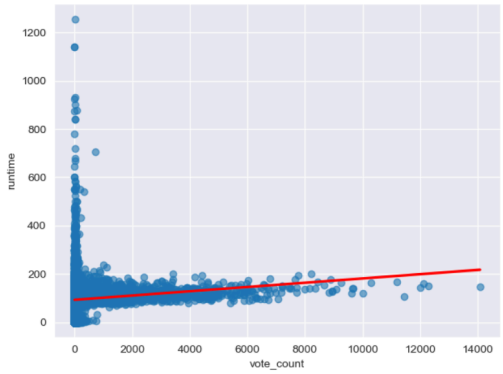

- Plot of the relationship between movie runtime and revenue.



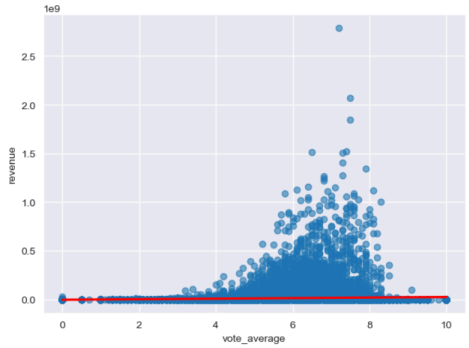

- Plot of the relationship between vote count and revenue.



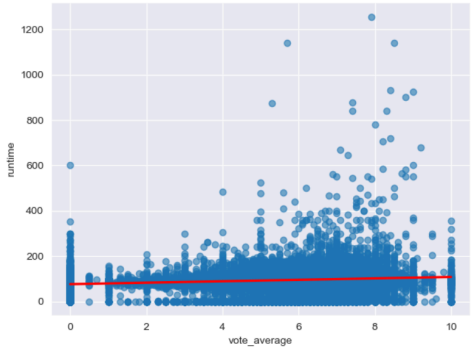

- Plot of the relationship between vote count and movie runtime.



- Plot of the relationship between vote average and revenue.




- Plot of the relationship between vote average and movie runtime.

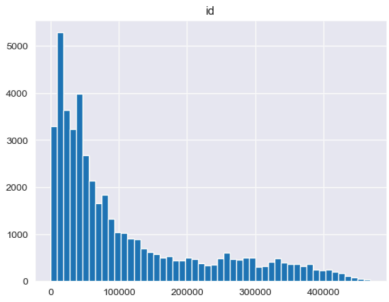



Finally, the file `movies_metadata.csv` was saved with the new features.

## Keywords EDA

First, the libraries were imported, and then the data was analyzed.

Using histograms, the distribution of the numerical column (`id`) in the `keywords` dataframe was examined.




The function **`extract_text_value`** was used to process the input text, which contains a list of dictionaries.

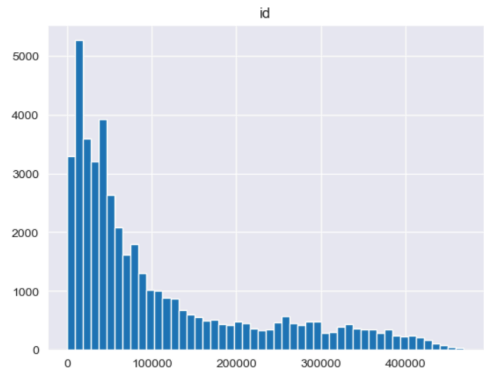
From the column **`keywords`**, the keys `name` and `id` were extracted, and each was converted into a new column in the `keywords` dataframe.

Finally, the `keywords` column was dropped, and the `keywords` dataframe was saved as a CSV file.

## Credits EDA

First, the libraries were imported, and the data was analyzed.

Using histograms, the distribution of the numerical column (`id`) in the `credits` dataframe was examined.




The function **`extract_text_value`** was used to process the input text, which contains a list of dictionaries.

From the column **`crew`**, the keys `name`, `department`, `gender`, `job`, and `id` were extracted, and each was converted into a new column in the `credits` dataframe.

From the column **`cast`**, the keys `name`, `order`, `gender`, `credit_id`, `id`, `profile_path`, and `character` were extracted, and each was converted into a new column in the `credits` dataframe.

Finally, the columns `cast` and `crew` were dropped, and the `credits` dataframe was saved as a CSV file.

## Splitting

First, all the libraries were imported, and the datasets were analyzed.

A dictionary was created from the `links` dataframe to establish a relationship between the columns **`tmdbId`** and **`movieId`**, so that each movie could be linked to its corresponding identifier in TMDB.

In the next step, only the rows where `movieId` could be mapped to `tmdbId` were retained, and a new column **`tmdbId`** was generated from this mapping.

After that, the process of sorting and separating the data for each user's latest interaction was designed. In this step, the **`timestamp`** column was converted into datetime format, and the entire dataframe was sorted by `userId` and `timestamp`.

Finally, the **train** and **test** dataframes were created using an 80-20 split.

## Baselines(Weighted Rating)

Here, two key values were defined for the **weighted rating formula**:

- **C**: the mean rating of all movies.
- **m**: the 80th percentile of vote counts (i.e., the minimum number of votes required for a movie to be considered valid).

In the next step, two more values were defined:

- **V**: the number of votes for a movie.
- **R**: the average rating of a movie.

These values together form the formula.

For each movie, the weighted average was calculated and added as a new column.
The set of all mapped TMDB IDs was stored in **`candidate_tmdb`**, and from `movie_small`, only the movies included in `candidate_tmdb` were selected and stored in **`pop_candidate`**.
Finally, all the data in `pop_candidate` was sorted based on **`wr`**.

## Content-Based Recommender

In this section, the goal is to build a **content-based recommendation system**.Here, `candidate_tmdb`, which is the list of candidate TMDB IDs, is used for filtering.

Using **TF-IDF**, for each movie a string `cb_text` is constructed, which includes:

- the **overview** and **tagline**,
- **genre tokens**,
- **keywords**,
- the names of the **cast** and **crew** (up to the first 10 members).

Each of these elements is given its own specific prefix.

For item vectorization, the vocabulary size is limited to **50k**, and the output is the matrix **`X_item`**.
For quick access between TMDB IDs and matrix rows, two mappings are used:

- **`item_index`**: maps `tmdb_id → row in X_item`.
- **`index_item`**: the inverse of `item_index`.

In building user profiles:

- Training data is restricted to the indexed items.
- The mean rating of each user is calculated for centering.

- An item × user matrix is created and multiplied by **`X_item`**.

Now, user profiles are computed as a **matrix multiplication**, which is equivalent to a weighted sum of the TF-IDF vectors of the items.
Afterward, each user vector is **L2-normalized**.

The function for building a single-user profile works by aggregating the TF-IDF vectors of all items the user has interacted with.

The **`explain_recommendation`** function takes a target movie's genres and cast, and compares them with what the user has already seen.

- If there is a common genre, the first shared genre is included in the explanation.
- Otherwise, if there are shared actors, the overlapping actors are used.

Finally, the **`recommend_cb`** function is implemented.

### Collaborative Filtering

in the first part, **`n_item`** specifies the number of items available in the dataset.
A dataframe is then created for use with the **Surprise** library, which contains `user_item`, `item_idx`, and `rating`.

Using **Reader**, the model identifies the range of ratings, and with **Dataset**, the data is converted into the Surprise format.

Next, the ratings are normalized to a range between 0 and 1.
If a vector is empty or if the difference between the minimum and maximum values is too small, a zero array is returned.

Using **KNNBasic**, a collaborative filtering (CF) algorithm based on item similarity is built, where similarity is measured with **Pearson similarity**.

With the function **`cf_item_item_score`**, all predicted scores for a user based on the item–item approach are returned, and the predicted score for each item is obtained.

In the next stage, a **Matrix Factorization** algorithm is constructed. Using the **SVD** model, the predicted ratings for all items are generated and then normalized.

Finally, the function **`recommend_mf`** returns the top **k items** recommended for a user.

### Hybrid Model

In the first part, a helper function for **Collaborative Filtering (CF)** is created, which selects and executes one of the two CF methods.

In the `recommend_hybrid` function, the parameter `alpha=0.5` is used. This is because the **Content-Based (CB)** model has significantly higher accuracy than the CF model, and `alpha` represents the weight of CF in the hybrid model.

For each user, `cb_score` and `cf_score` are calculated.
If a user is not present in either model, the function `recommend_popular` provides general popular items to the user.

First, the CB and CF scores are normalized (if they are already normalized, no further normalization is applied).
Then, the scores are combined to build the **hybrid model**, and the hybrid scores are sorted to generate the top **k recommended items**.

## Evaluation and Experiment Design

The evaluation metrics are defined as follows:

- **Precision@k** measures the percentage of recommended items that are actually of interest to the user.

- **Recall@k** measures the percentage of a user's true items that are covered by the system's predictions.
- **NDCG@k** (Normalized Discounted Cumulative Gain) considers the rank of correct items; higher-ranked correct items receive higher scores.
- **Hit Rate@k** assigns a score of 1 if at least one correct item is in the recommendation list, and 0 otherwise.
- **MAP@k** (Mean Average Precision) refers to the average cumulative precision, giving higher scores if correct items appear earlier in the recommendation list.

After defining the metrics, the sets `train_seen` and `test_truth` are created.

Prediction functions for **CB**, **CF (MF)**, **CF (item-item)**, and **Hybrid** models were implemented:

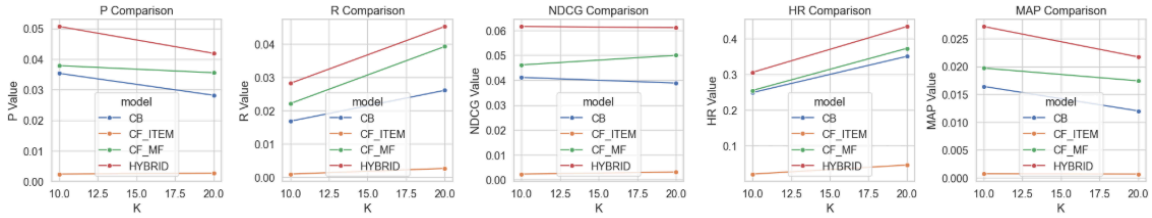
- The **CB** function selects similar movies based on their content (overview, genre, cast).
- The **CF MF** function uses Matrix Factorization to compute similarity scores between items.
- The **CF item-item** function finds similarity between movies using item–item relationships.
- The **Hybrid** function combines CB and CF scores with a weight parameter `alpha`.

The models are stored in a dictionary to allow easy switching between them.

**Bootstrap** was used to compute a 95% confidence interval for the mean of each metric.

Finally, for each model, each value of **k = [10, 20]**, and each user, the metric values are calculated. Then, the mean is taken, and confidence intervals are constructed.

| | model | k | P | P_ci95 | R | R_ci95 | NDCG | NDCG_ci95 | HR | HR_ci95 | MAP | MAP_ci95 | Coverage | Novelty | ILD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | CB | 10 | 0.035320 | [0.030, 0.041] | 0.016852 | [0.014, 0.020] | 0.041112 | [0.034, 0.048] | 0.248882 | [0.216, 0.281] | 0.016427 | [0.013, 0.020] | 0.205131 | 5.948777 | 0.915060 |
| 1 | CF_ITEM | 10 | 0.002683 | [0.001, 0.004] | 0.000992 | [0.000, 0.002] | 0.002488 | [0.001, 0.004] | 0.020864 | [0.010, 0.032] | 0.000812 | [0.000, 0.002] | 0.300154 | 7.216246 | 0.981701 |
| 2 | CF_MF | 10 | 0.037854 | [0.032, 0.043] | 0.022163 | [0.018, 0.027] | 0.046141 | [0.038, 0.054] | 0.254844 | [0.226, 0.289] | 0.019716 | [0.016, 0.023] | 0.013323 | 2.857396 | 0.975198 |
| 3 | HYBRID | 10 | 0.050671 | [0.044, 0.058] | 0.028247 | [0.024, 0.034] | 0.061566 | [0.053, 0.071] | 0.305514 | [0.271, 0.338] | 0.027203 | [0.022, 0.032] | 0.023783 | 2.907601 | 0.974195 |
| 4 | CB | 20 | 0.028167 | [0.025, 0.032] | 0.026129 | [0.022, 0.031] | 0.038853 | [0.034, 0.044] | 0.350224 | [0.316, 0.384] | 0.012030 | [0.010, 0.015] | 0.376129 | 6.200527 | 0.945399 |
| 5 | CF_ITEM | 20 | 0.002534 | [0.002, 0.004] | 0.002625 | [0.001, 0.004] | 0.003015 | [0.002, 0.004] | 0.041729 | [0.027, 0.057] | 0.000679 | [0.000, 0.001] | 0.399582 | 7.230559 | 0.981889 |
| 6 | CF_MF | 20 | 0.035544 | [0.031, 0.041] | 0.039307 | [0.033, 0.046] | 0.049984 | [0.043, 0.058] | 0.372578 | [0.335, 0.408] | 0.017436 | [0.014, 0.021] | 0.019489 | 2.974452 | 0.979848 |
| 7 | HYBRID | 20 | 0.041878 | [0.037, 0.047] | 0.045329 | [0.039, 0.053] | 0.061116 | [0.054, 0.069] | 0.433681 | [0.395, 0.472] | 0.021736 | [0.018, 0.026] | 0.033583 | 3.063142 | 0.979235 |




After computing the metrics, **Ablation experiments** were performed to assess the importance of features:

1. The first ablation experiment removes the **cast** feature, using only the **overview**, **tagline**, and **genre**.

| | model | k | P | P_ci95 | R | R_ci95 | NDCG | NDCG_ci95 | HR | HR_ci95 | MAP | MAP_ci95 | Coverage | Novelty | ILD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | CB_no_cast | 10 | 0.004471 | [0.003, 0.006] | 0.001354 | [0.001, 0.002] | 0.005606 | [0.003, 0.008] | 0.041729 | [0.027, 0.057] | 0.002039 | [0.001, 0.003] | 0.093963 | 7.343092 | 0.976129 |
| 1 | CB_no_cast | 20 | 0.003577 | [0.003, 0.005] | 0.002739 | [0.002, 0.004] | 0.005153 | [0.004, 0.007] | 0.065574 | [0.051, 0.085] | 0.001392 | [0.001, 0.002] | 0.137257 | 7.459615 | 0.976097 |

2. The second ablation experiment replaces **TF-IDF** with **Count Vectorizer**, aiming to compare the performance of Count Vectorizer against TF-IDF.

| | model | k | P | P_ci95 | R | R_ci95 | NDCG | NDCG_ci95 | HR | HR_ci95 | MAP | MAP_ci95 | Coverage | Novelty | ILD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | CB_count | 10 | 0.004769 | [0.003, 0.007] | 0.001469 | [0.001, 0.002] | 0.005757 | [0.004, 0.008] | 0.044709 | [0.029, 0.061] | 0.002056 | [0.001, 0.003] | 0.057161 | 7.374737 | 0.962342 |
| 1 | CB_count | 20 | 0.004545 | [0.004, 0.006] | 0.002645 | [0.002, 0.004] | 0.005605 | [0.004, 0.007] | 0.080477 | [0.061, 0.100] | 0.001423 | [0.001, 0.002] | 0.082319 | 7.427162 | 0.962642 |

## Streamlit

First, the models are loaded.

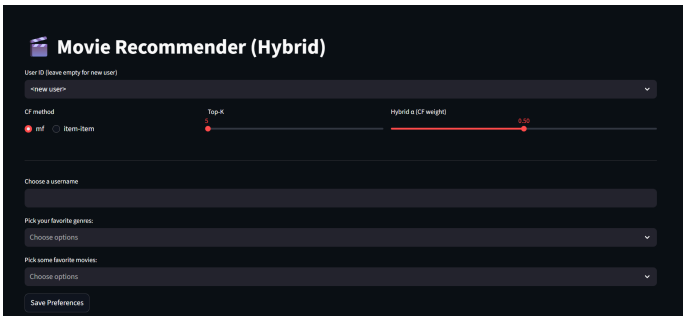
Four recommendation functions were implemented:

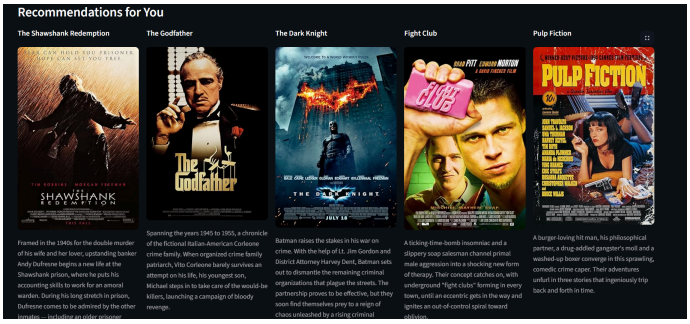
1. `preds_cb`: Based on **Content-Based** filtering. This function computes the inner product of the user feature vector with the item similarity matrix, sorts the resulting vector by similarity, removes movies the user has already seen, and returns the final recommendations.

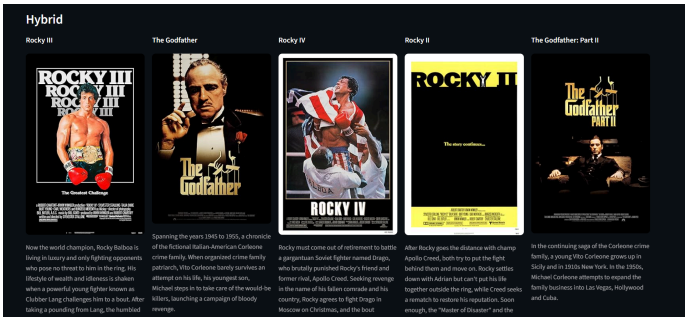
2. `preds_cf`: Based on **Collaborative Filtering**. It makes recommendations using collective user behavior and leverages two algorithms: **Matrix Factorization** and **similarity-based CF**. The algorithm constructs a score vector, sorts items based on these scores, removes already-seen movies, and returns the top recommendations.
3. `preds_hybrid`: Builds a **hybrid model** by combining CB and CF scores in a weighted manner.
4. `recommend_popular`: Recommends items based on the **weighted rating (wr)**.

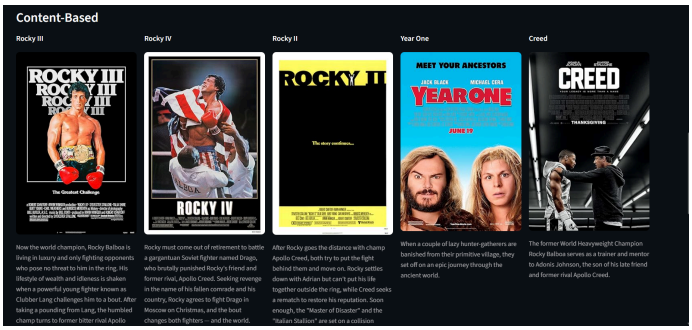
In the **UI section**, for the 671 users whose movie ratings were available, three models are displayed: **Hybrid**, **Content-Based**, and **Collaborative Filtering**.

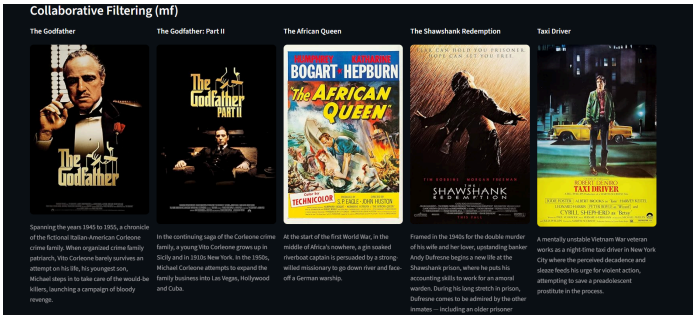
- For **new users**, the **Weighted Rating (WR)** model is shown.
- If a user registers, recommendations are made based on their preferred genres and favorite movies.
- Additionally, all users have access to **movie search** and **genre-based recommendations**.

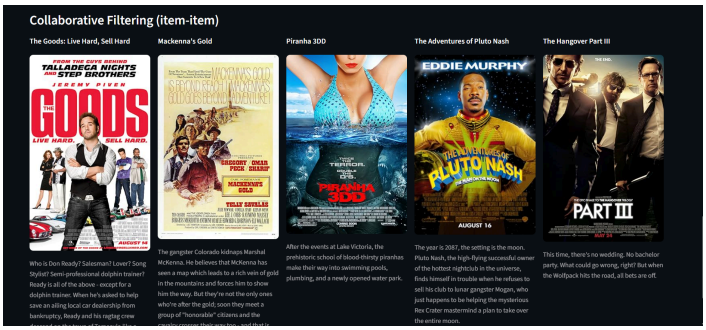





## Deployment on Hugging Face Spaces and Github Repository

Hugging Face

Github