The Movie Recommender

Yuanxin Tuo 251369584, Jiaming Ma 251299592, Haochen Shen 251317254, Zhenghao Wang 251299639

I. Introduction

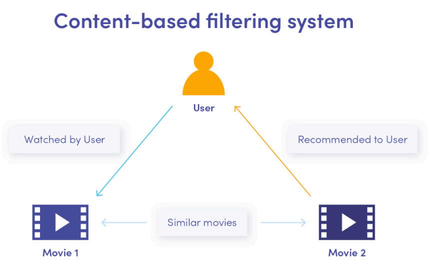
In today's digital age, recommendation systems have become the backbone of user experience, profoundly impacting how we discover new favorites across e-commerce, video streaming, and social networks. From the personalized shopping suggestions on Amazon to the movies we binge on Netflix, these systems help navigate the vast ocean of options, tailoring choices to individual preferences. In this project we will show you how I build a recommendation system, using Content-Based Filtering (CBF), Collaborative Filtering (CF) and combined them as a Hybrid Recommender.

A. *Different recommendataion system algorithms*

	Content-Based Filtering	Collaborative Filtering	Hybrid Recommendation
Concept	Recommends items using detailed analysis of item features.	Suggests items by analyzing user behavior across the user base.	Integrates product characteristics and user preferences to provide recommendations.
Principle	People with similar opinions on one item will likely agree on others.	Users will prefer items similar to those they liked before, as inferred from user preference patterns.	Combines user's historical data and item attributes to predict preferences, aiming to provide more accurate suggestions.
Strengths	- Independently functional without user data. - Identifies niche items effectively.	- Domain knowledge not required. - Facilitates discovery of new user preferences.	- Mitigates the cold-start problem. - Offers more personalized recommendations by leveraging both content and user data.
Limitations	- Requires deep understanding of item attributes. - Does not promote exploration of new interests.	- Struggles with new items or users due to the cold-start issue. - Feature expansion for recommendati ons is complex.	- Can be more complex to implement. - Requires significant data processing and storage.

B. *Content-Based Filtering*

Content-Based Filtering (CBF) is a recommendation system approach that makes use of item features to recommend additional items similar to what a user has liked in the past. It operates by creating item profiles (which describe the items' properties) and user profiles (which describe the user's preferences), and then it matches these profiles to suggest new items to the user.



In this project, we will test a variety of different machine learning algorithms to establish which one is most beneficial for content-based recommendations on this dataset.

Linear Regression: it could be used to predict a user's rating for an item based on the features of the item. The algorithm learns to weigh each feature by how much it influences the user's preferences.

Least Absolute Shrinkage and Selection Operator (Lasso): It is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces.

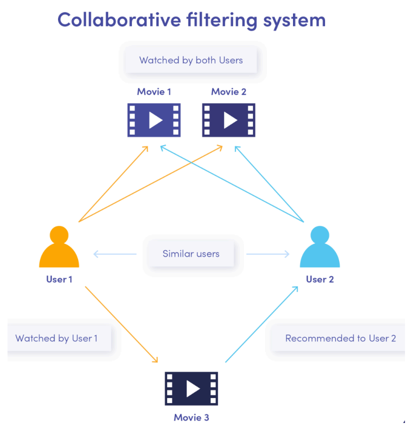
K-Nearest Neighbors (KNN): it can be used to find items that are most similar to the items the user has liked before. It does this by calculating a distance metric between the features of items.

Random Forest Regressor (RFR): is an ensemble learning method that uses multiple decision trees to predict the user's rating of an item. It can handle the interaction between different item features well.

Support Vector Regression (SVR): it can be applied to predict ratings similarly to LR but is more flexible due to the use of kernels, which allow the method to model non-linear relationships.

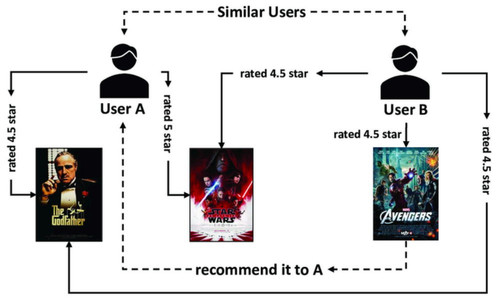
C. *Collaborative Filtering*

Collaborative Filtering (CF) is a technique used by recommender systems to predict the preferences of one user based on the preferences of many other users. It operates on the premise that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a random person.



In this Project we will test user-based collaborative filtering with the KNN algorithm and model it in the different number of neighbors.

In user-based CF, KNN finds the k nearest neighbors (most similar users) to a particular user. The system then averages the ratings of these neighbors to predict the ratings for items not yet rated by that user.



An example of Collaborative Filtering with KNN

D. *Dataset*

All data was sourced from the MovieLens 100-k Dataset, The data was collected through the MovieLens web site (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. This data has been cleaned up users who had less than 20 ratings or did not have complete demographic information were removed from this data set.

In our project only u.data, u.item, and u.user are needed.
u.data -- The full u data set, 100000 ratings by 943 users on 1682 items.

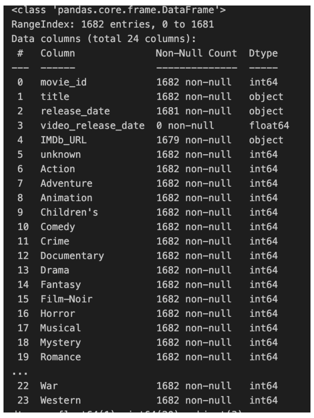
This is a tab separated list of :
user id | item id | rating | timestamp.

u.item -- Information about the items (movies); this is a tab separated list of:
movie id | movie title | release date | video release date | IMDb URL | unknown | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western |
u.user -- Demographic information about the users; this is a tab
separated list of
user id | age | gender | occupation | zip code

II. Data Cleaning

A. *Handle Unusual Values*

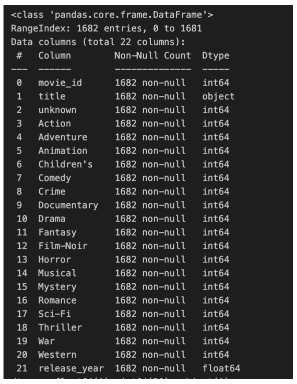
In these 3 files, only u.item has missing values and un usable value.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1682 entries, 0 to 1681
Data columns (total 24 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   movie_id            1682 non-null   int64
 1   title               1682 non-null   object
 2   release_date        1681 non-null   object
 3   video_release_date  0 non-null      float64
 4   IMDb_URL            1679 non-null   object
 5   unknown             1682 non-null   int64
 6   Action              1682 non-null   int64
 7   Adventure           1682 non-null   int64
 8   Animation           1682 non-null   int64
 9   Children's          1682 non-null   int64
 10  Comedy              1682 non-null   int64
 11  Crime               1682 non-null   int64
 12  Documentary         1682 non-null   int64
 13  Drama               1682 non-null   int64
 14  Fantasy             1682 non-null   int64
 15  Film-Noir           1682 non-null   int64
 16  Horror              1682 non-null   int64
 17  Musical             1682 non-null   int64
 18  Mystery             1682 non-null   int64
 19  Romance             1682 non-null   int64
...
 22  War                 1682 non-null   int64
 23  Western             1682 non-null   int64
```

We decided to fill the missing value, an average, in release_data column rather than deleted the rows, because it would decrease the size of data.

Then I changed the release_date to just the year and saves it in a new column called release_year, simplifies the data related to movie release year, which could make the analysis and model much more easier. Finally, I Removed the release_date, video_release_date, and IMDb_URL columns, because they are unnecessary for this project.

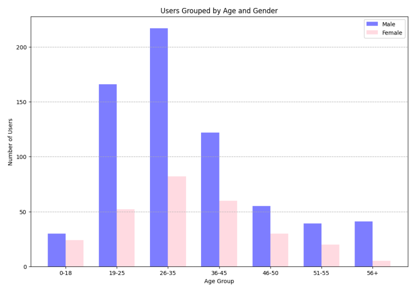


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1682 entries, 0 to 1681
Data columns (total 22 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   movie_id      1682 non-null   int64
 1   title         1682 non-null   object
 2   unknown       1682 non-null   int64
 3   Action        1682 non-null   int64
 4   Adventure     1682 non-null   int64
 5   Animation     1682 non-null   int64
 6   Children's    1682 non-null   int64
 7   Comedy        1682 non-null   int64
 8   Crime         1682 non-null   int64
 9   Documentary   1682 non-null   int64
 10  Drama         1682 non-null   int64
 11  Fantasy       1682 non-null   int64
 12  Film-Noir     1682 non-null   int64
 13  Horror        1682 non-null   int64
 14  Musical       1682 non-null   int64
 15  Mystery       1682 non-null   int64
 16  Romance       1682 non-null   int64
 17  Sci-Fi        1682 non-null   int64
 18  Thriller      1682 non-null   int64
 19  War           1682 non-null   int64
 20  Western       1682 non-null   int64
 21  release_year  1682 non-null   float64
```

III. Data Exploration

This section explores data analysis and visualization methods to thoroughly examine the processed datasets.

A. *User Dataset Exploration*



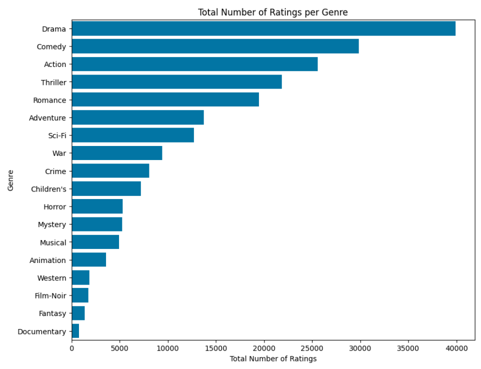
- The age group with the highest number of users is 26-35, followed by 19-25, suggesting that the user base is predominantly young adults.

- In every age group, there are more male users than female users.

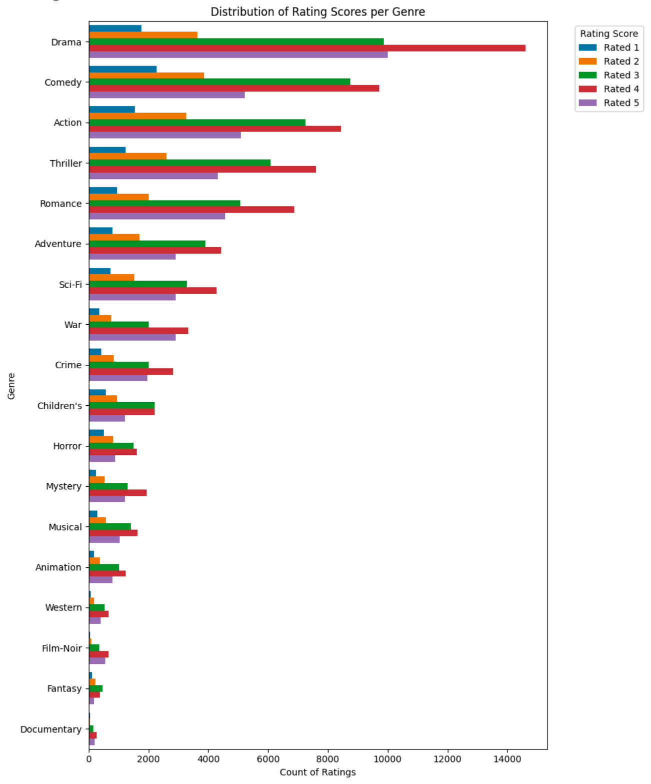
- The difference in the number of users between genders is greatest in the 26-35 age group.

- The number of users tends to decrease as age increases, with the 56+ age group having the fewest users.

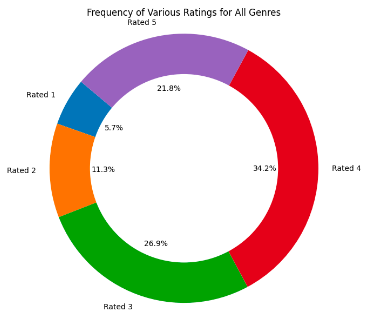
B. *The Movies and Ratings Datasets Explorations*



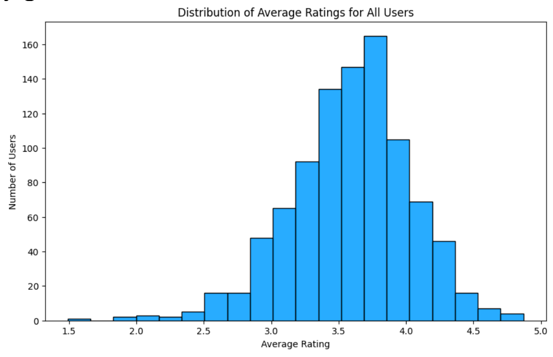
The chart demonstrates clear preferences in movie genres among users, with Drama being the most popular for ratings, indicating either a higher number of Drama movies in the dataset or a greater interest in this genre among the users rating the movies.



Drama, Comedy, and Action are the most frequently rated genres, with Drama at the top. Most ratings across all genres are 3 or above, indicating that users tend to rate movies positively.



Most movie ratings are 4 or 5, indicating people often rate movies highly. Fewer ratings are 1 or 2, showing that people rarely give the lowest scores.



Most users have an average rating between 3 and 4, with the highest number of users around 3.5 to 4. There are very few users with an average rating below 2.5 or above 4.5. This suggests that users generally give moderate to high ratings, and very high or very low average ratings are uncommon.

IV. Data Modeling

A. *Splitting the Datasets*

We've opted for a distribution of 70% training, 15% validation, and 15% testing, prioritizing a more substantial training dataset over larger validation and test sets. Our rationale is that the simplicity of the machine learning models being employed, with their limited hyperparameters, should mitigate any potential downsides of smaller validation or test datasets in terms of the models' ability to generalize.

B. *Baseline Model*

To set a benchmark for assessing the performance of sophisticated recommender systems, we will initiate the process by establishing an elementary baseline model. Employing a strategy centered on popularity, we intend to ascertain the mean rating each film receives and project that all viewers will assign ratings to movies that align with these average ratings. Despite the absence of customization in this rudimentary model, it is commonly observed that the majority of viewers tend to rate films in proximity to their average ratings. Consequently, I anticipate that the root mean square error (RMSE) for this simplistic model may hard to surpass.

```
The RMSE of the baseline model is: 1.000070019527579
```

C. *Content-Based Filtering*

In this study, we're using content-based filtering (CBF) to recommend movies. This method uses a user's past ratings and favorite genres to suggest new movies they haven't seen yet. We'll test different machine learning algorithms to see which works best for our data, although we think linear models might do a good job.

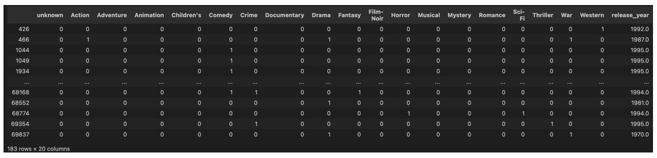
CBF is good because it makes personal recommendations based on what each user likes, like their favorite genres or directors. But our data, from the MovieLens dataset, doesn't have a lot of details about the movies, which could make it hard to make very personalized suggestions.

We suspect that just using genres and user ratings, our CBF might not beat a simple average rating model because of these data issues. But we'll try different approaches to see how close we can get to or even improve on the basic model. The best-performing model will then be used in a more complex, hybrid recommendation system.

1) Data Processing for CBF

- Preparing Training Data

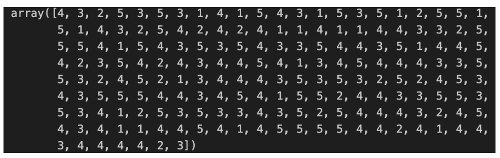
x_train



	unknown	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western	release_year
426	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1992.0
466	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1987.0
1044	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1995.0
1049	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1995.0
1934	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1995.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
68168	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	1994.0
68552	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1981.0
68774	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1994.0
69354	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1995.0
69837	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1970.0

183 rows × 20 columns

y_train: ratings by user_id

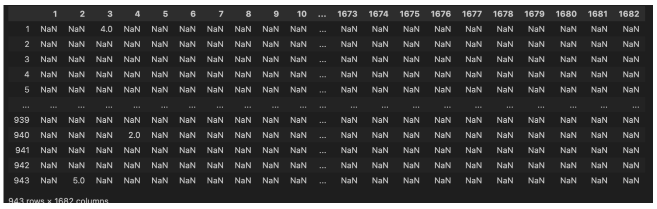


```
array([4, 3, 2, 5, 3, 5, 3, 1, 4, 1, 5, 4, 3, 1, 5, 3, 5, 1, 2, 5, 5, 1,
       5, 1, 4, 3, 2, 5, 4, 2, 4, 2, 4, 1, 1, 4, 1, 1, 4, 4, 3, 3, 2, 5,
       5, 5, 4, 1, 5, 4, 3, 5, 3, 5, 4, 3, 3, 5, 4, 4, 3, 5, 1, 4, 4, 5,
       4, 2, 3, 5, 4, 2, 4, 3, 4, 4, 5, 4, 1, 3, 4, 5, 4, 4, 4, 3, 3, 5,
       5, 3, 2, 4, 5, 2, 1, 3, 4, 4, 4, 3, 5, 3, 5, 3, 2, 5, 2, 4, 5, 3,
       4, 3, 5, 5, 5, 4, 4, 3, 4, 5, 4, 1, 5, 5, 2, 4, 4, 3, 3, 5, 5, 3,
       5, 3, 4, 1, 2, 5, 3, 5, 3, 3, 4, 3, 5, 2, 5, 4, 4, 4, 3, 2, 4, 5,
       4, 3, 4, 1, 1, 4, 4, 5, 4, 1, 4, 5, 5, 5, 5, 4, 4, 2, 4, 1, 4, 4,
       3, 4, 4, 4, 4, 2, 3])
```

Ratings from the first user

- Preparing Validation Data

A two dimensional Matrix: Axis 1: User IDs, Axis 2: Movie IDs.



	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676	1677	1678	1679	1680	1681	1682
1	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
939	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
940	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
941	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
942	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
943	NaN	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

943 rows × 1682 columns

2) Creating the Different Content-Based Filtering Models

Here are models and parameters:

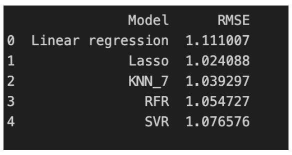
Linear Regression: A basic regression algorithm without any regularization.

Lasso: It's set with a regularization strength alpha of 1.0 and a maximum number of iterations set to 10,000.

KNN_7: K-Nearest Neighbors Regression with the number of neighbors set to 7.

RFR: Random Forest Regressor with 1,000 estimators (trees), utilizing a maximum of 3 parallel jobs, and considering the square root of the number of features when looking for the best split, set with a random state of 0 for reproducibility.

SVR: Support Vector Regression with a penalty parameter C of 1.0.



```
               Model      RMSE
0  Linear regression  1.111007
1              Lasso  1.024088
2              KNN_7  1.039297
3                RFR  1.054727
4                SVR  1.076576
```

Lasso demonstrated a significantly lower RMSE compared to other models, outshining both linear models and the KNN classifier, which did not surpass the baseline model. Consequently, Lasso will be employed for the hybrid recommender.

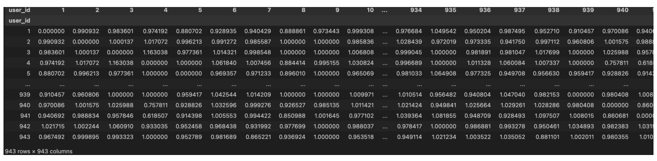
D. Collaborative Filtering

In this part, we're going to work on collaborative filtering (CF) models, using a method called K-Nearest Neighbours (KNN) to recommend movies. This method looks at what movies people with similar tastes have enjoyed and suggests those movies to others. It doesn't take into account any personal information about the users, just how they've rated movies.

We make sure to even out everyone's ratings by taking away their average rating from all their movie ratings. This helps because people might give different average scores based on their own habits. We measure how similar users are by using something called the Pearson correlation coefficient. For more accurate recommendations, especially when we use a larger number of nearest neighbors (a higher K value), we're also tweaking our model. We're making sure that opinions from users who have more in common count for more.
Like we did with the content-based filtering, we'll test different CF models. The goal is to find the one that works best so we can add it to our hybrid system, which uses both CF and content-based methods. This way, we hope to make even better movie suggestions.

1) Data Processing for CF

Calculate the personal distance between all users in training data

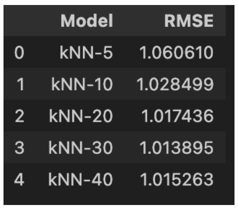


user_id	1	2	3	4	5	6	7	8	9	10	...	934	935	936	937	938	939	940	9
user_id																			
1	0.000000	0.990932	0.983601	0.974192	0.880702	0.928935	0.940429	0.888861	0.973443	0.999308	...	0.976684	1.049542	0.950204	0.987495	0.952710	0.910457	0.970086	0.940
2	0.990932	0.000000	1.000137	1.017072	0.996213	0.991272	0.985587	1.000000	1.000000	0.985836	...	1.028439	0.972019	0.973335	0.941750	0.997112	0.960806	1.001575	0.988
3	0.983601	1.000137	0.000000	1.163038	0.977361	1.014321	0.998548	1.000000	1.000000	1.006808	...	0.999045	1.000000	0.981891	0.981047	1.017699	1.000000	1.025988	0.957
4	0.974192	1.017072	1.163038	0.000000	1.000000	1.061840	1.007456	0.884414	0.995155	1.030824	...	0.996689	1.000000	1.011328	1.060084	1.007337	1.000000	0.757811	0.618
5	0.880702	0.996213	0.977361	1.000000	0.000000	0.969357	0.971233	0.896010	1.000000	0.965069	...	0.981033	1.064908	0.977325	0.949708	0.956630	0.959417	0.928826	0.914
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
939	0.910457	0.960806	1.000000	1.000000	0.959417	1.042544	1.014209	1.000000	1.000000	1.009971	...	1.010514	0.956482	0.940804	1.047040	0.982153	0.000000	0.980408	1.008
940	0.970086	1.001575	1.025988	0.757811	0.928826	1.032596	0.999276	0.926527	0.985135	1.011421	...	1.021424	0.949841	1.025664	1.029261	1.028286	0.980408	0.000000	0.860
941	0.940692	0.988834	0.957846	0.618507	0.914398	1.005553	0.994422	0.850988	1.001645	0.977102	...	1.039364	1.081855	0.948709	0.928493	1.097507	1.008015	0.860681	0.000
942	1.021715	1.002244	1.060910	0.933035	0.952458	0.968438	0.931992	0.977699	1.000000	0.988037	...	0.978417	1.000000	0.986881	0.993278	0.950461	1.034893	0.982383	1.031
943	0.967492	0.999895	0.993323	1.000000	0.952789	0.981689	0.865221	0.936924	1.000000	0.953518	...	0.949114	1.021234	1.003522	1.035052	0.881101	1.002011	0.980355	1.010

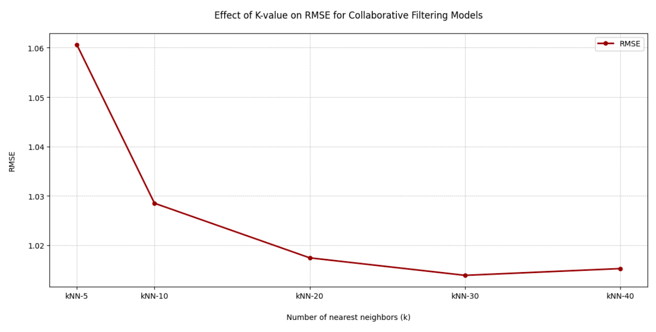
943 rows × 943 columns

2) Creating Collaborative Filtering Models

Here are different configurations of the K-Nearest Neighbors algorithm for collaborative filtering. Each key in the dictionary corresponds to a version of the KNN algorithm with different K values: 5, 10, 20, 30, and 40.



	Model	RMSE
0	kNN-5	1.060610
1	kNN-10	1.028499
2	kNN-20	1.017436
3	kNN-30	1.013895
4	kNN-40	1.015263



The graph illustrates the impact of the K-value on the performance of collaborative filtering models. The root mean square error (RMSE) decreases as the number of nearest neighbors increases from 5 to 40, indicating improved accuracy. With a K-value of 30, the model achieves its lowest RMSE, suggesting this is the optimal number of neighbors for the best predictive performance among the values tested.

Therefore, I'll use the K-value of 30 for the final collaborative filtering model in our hybrid recommender system.

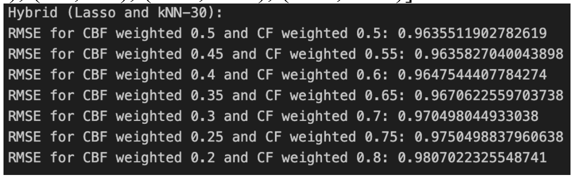
E. Hybrid Recommendation System

In this section, we're going to mix the best parts of content-based filtering (CBF) and collaborative filtering (CF) to make a hybrid recommender system. By doing this, we hope to create a system that's better than either CBF or CF on their own.

For our project, We decided to choose Lasso and KNN-30 as parts of hybrid system, which have best performance in the CBF and CF separately.

The different weights is applied in our project to find a better performance

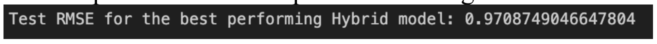
weighted_avgs = [(0.5, 0.5), (0.45, 0.55), (0.4, 0.6), (0.35, 0.65), (0.3, 0.7), (0.25, 0.75), (0.20, 0.80)]



```
Hybrid (Lasso and kNN-30):
RMSE for CBF weighted 0.5 and CF weighted 0.5: 0.9635511902782619
RMSE for CBF weighted 0.45 and CF weighted 0.55: 0.9635827040043898
RMSE for CBF weighted 0.4 and CF weighted 0.6: 0.9647544407784274
RMSE for CBF weighted 0.35 and CF weighted 0.65: 0.9670622559703738
RMSE for CBF weighted 0.3 and CF weighted 0.7: 0.970498044933038
RMSE for CBF weighted 0.25 and CF weighted 0.75: 0.9750498837960638
RMSE for CBF weighted 0.2 and CF weighted 0.8: 0.9807022325548741
```

Test and Evaluation

In the end, we test the performance of the hybrid system by receiving the prediction from the test data. We decide the weight of CBF and CF is 0.5. Because this combination has the best performance from previous training.



```
Test RMSE for the best performing Hybrid model: 0.9708749046647804
```

The RMSE of Hybrid Recommendation System is 0.971 based on the test data, and the RMSE of the baseline model is 1.000. So there has a substantial improvement for Hybrid Recommendation System, in terms of RMSE, after combining CF and CBF in a proper rate.

There are still lots of points we can explore to improve the performance of the model. Here are few thoughts:

1) Feature Engineering: we can add the average age of movie fan to the movies dataset. Movies are made for specific audiences. After analyzing the data, it seems that audiences vary in age, but not as widely as the overall population. There's a chance that a movie's ratings could be linked to the average age of its typical viewers, especially for genres aimed at kids or teens. Yet, even movies targeted at a specific group may attract a broader range of viewers who also give ratings. we want to check if knowing the average age of users who give high ratings to movies can help make better recommendations.
2) Model Complexity: While the models chosen are appropriate, exploring more complex or recent recommendation system approaches, such as matrix factorization techniques or deep learning models, could offer interesting comparisons.
3) Evaluation Metrics: Beyond RMSE, consider using additional metrics like precision, recall, or F1-score, which can provide a more nuanced view of the model's recommendation quality, especially in terms of user satisfaction.

In the future, we aim to embed the recommendation system into a website, adjusting the model based on the actual behavior of real users, rather than just relying on the numerical results of metrics.