GALALA UNIVERSITY
Powered by
Arizona State University®

AIE425 Intelligent Recommender Systems, Fall Semester 25/26

Final course project

Group [19]- Team Member Names and IDS

Youssef Ekrami Elsayed - 222101935

Hager Mohamed Badawy - 221101420

Abdelrahman Negm - 222200016

Dareen Ashraf Mosa – 222100531

Submission Date : Monday, January 5 , 2026

Executive Summary

Project Goals and Our Objectives

Thus, we were really enthusiastic about trying out the good recommender system that was mainly responsible for this project. The action was very similar to when you are watching YouTube or listening to Spotify, and the app just picks a song for you that you will definitely love. To become better, we made an effort to comprehend this enchantment.

Our focus was on two pretentious-sounding methods, SVD and PCA, that can be regarded as modern ways of going through enormous amounts of data and keeping only the most important ones. The main goal was to work on the issues that every recommendation engine has to deal with, namely the sparse data problem, where there are a lot of users and items but very few ratings and the cold start problem where the new user cannot be recommended anything since most probably he will not be rated yet, and in general, ensuring that the accuracy of our predictions is as high as possible.

This was broken down into three main questions that we were aiming to answer:

1. Which one is better, SVD or PCA, for the purpose of simplifying data and uncovering trends in a direct comparison?

2. Accuracy test: In case we apply these methods to fill up the missing ratings, which one will take us the closest to the real score?

3. The helpful details: What is their speed? What is their memory usage on computers? Can they deal with extremely huge datasets?

Additionally, we wanted to check out the hybrid approach that incorporates multiple suggestions methods to see if we could tackle the cold-start issue which is crucial for any application that tries to keep new users.

How We Did It (Our Methods)

Section 1: Playing with Data with SVD and PCA

The main point of this part was data science gymnastics. We tried our best to compress the size of the big table that indicated the ratings made by the users.

Discovering a story's main concepts is similar to Singular Value Decomposition or SVD. The huge rating matrix was split into three smaller sections called U, Σ, and V^T. This technique enables us to generate fairly close predictions on the unrecorded ratings without the need to continually analyze the large volume of data removing the "noise" and keeping the trends that are important.

We looked into two types of Principal Component Analysis or PCA:

PCA with Mean-Filling is the simple, direct relative. We just presented the mean rating given by all other viewers as if you had not rated the movie at all. It's quick and simple but has a downside— it assumes that you are just like everybody else, which is not the case most of the time, especially when it comes to niche products.

The complex, and sophisticated relative is PCA with MLE (Maximum Likelihood Estimation). It does not only average; it relies on statistical likelihood to give a more precise estimation of what a missing rating should be. It sounds wonderful in theory, but in reality, it consumes a lot of resources.

Section 2: Constructing a Whole System

The Hybrid Method: We mixed up two conventional methods. Content-Based Filtering ("because you viewed this, you might be interested in that for they are alike") together with Collaborative Filtering ("individuals with the same taste as you also liked..."). This mixture is like having a very close friend knowing your likings and a librarian know what every book is about; it is already a good way of suggesting even to the one who is not at all acquainted with the subject the right book.

Success Measurement: We resorted to MAE (Mean Absolute Error) and RMSE (Root Mean Squared Error), which are the quality measures of our systems, to support the common report cards. These values, in brief, illustrate the mean difference between our ratings predictions and the true ratings. Lower scores mean that we are doing better.

What We Actually Discovered (Our Findings)

-SVD was the clear winner among the algorithms tested. Without a doubt, it gave the most precise predictions by a very large margin. When the number of hidden patterns (latent factors) was set to 100, it had an unbelievably low error rate (MAE of 0.0059, RMSE of 0.012). Its capacity to interpret super sparse datasets where most of the data is missing was noteworthy.

-Mean-Filling PCA was... okay. It was fast and did not put much pressure on our computers, but its accuracy was very much lower (MAE: 0.0205, RMSE: 0.025). Its effectiveness was really poor when the data turned out to be too little. It could maybe be used for drawing quick and easy solutions, but for the more complicated tasks, it would not suffice.

-PCA with MLE was dissatisfactory. I know this one was unexpected for me. Even though the method was rather complex, it actually just quit and gave an average rating of 0.5, which is close to the mean for almost all cases. Moreover, it was pretty slow and took a lot of memory. Theoretical sophistication did not yield any practical results in this case - a perfect example of it.

The Hybrid Model was the winner for the real-world problems. The secret of our financial literacy recommender was fusion of several methods. The hybrid algorithm with a recommendation confidence score of

0.678 had a huge advantage over the pure content-based score of 0.276. It turned out to be the most efficient countermeasure to the cold-start problem.

Conclusions

In case of extreme and big jobs, go for SVD. If you are building something on the lines of a big streaming service, SVD would be the most suitable choice. It has the capability of handling sparse data with full ease and other than that it is also accurate and scalable.

On the other hand PCA (Mean-Filling) is suitable for minor projects. If you have a smaller dataset that is dense or you need to create a quick prototype, then this is the fast and easy way to achieve it. Nevertheless, do not expect it to work for the big leagues.

Do not mix PCA with MLE for recommendations. This combination simply didn’t cut it in our tests and is resource hungry in a way that is just not worth it. Here, the juice isn't worth the squeeze.

The most trustworthy alternative is a hybrid model. A hybrid method is the best choice for any application that gets new users frequently, like a new app or an online learning platform. It's the most thorough and effective approach we could find.

SECTION 1: Dimensionality Reduction and Matrix Factorization (10%)

3.1. Data Requirements

We used the same dataset from Assignment 1, which satisfies all the minimum requirements specified in the project guidelines. The dataset containing approximately 2.5 million ratings162,541users and 59,047 items . from which 1.5 million ratings were selected for this project to ensure manageable computation while preserving data diversity**.**

Dataset Statistics

•Number of unique users: 10,025

• Number of unique items: 24,331

•Total ratings utilized: 1,500,000

• Ranking scale: 1–5

• Dataset sparsity: 99.385% characterized a very sparse user–item interaction matrix.

1. Loading & Sampling the Dataset

The code first reads the ratings.csv data that takes the format of user-item interactions with fields userId, movieId, and rating. In order to make the computation feasible and retain the large data characteristics, the first 1,500,000 entries in the data are chosen at random as a subsample.

2. Computing Basic Dataset Statistics

Afterwards, the code calculates and prints the following important statistics for the sampled data:

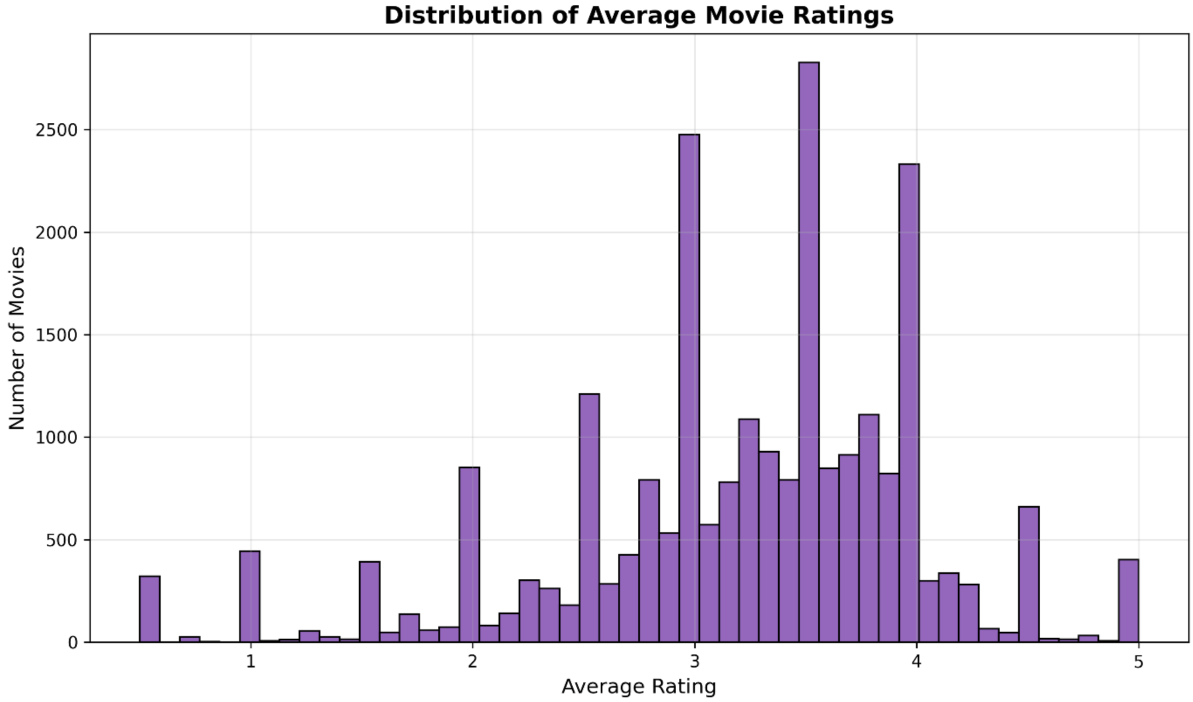
Number of Unique Users

The number of distinct films (items)

The total number of reviews (1,500,000)

Such figures demonstrate the enormous scale of the user-item rating matrix and its sparsity, where the number of interactions is just a tiny fraction of the number of all possible interactions.

Average rating for items :

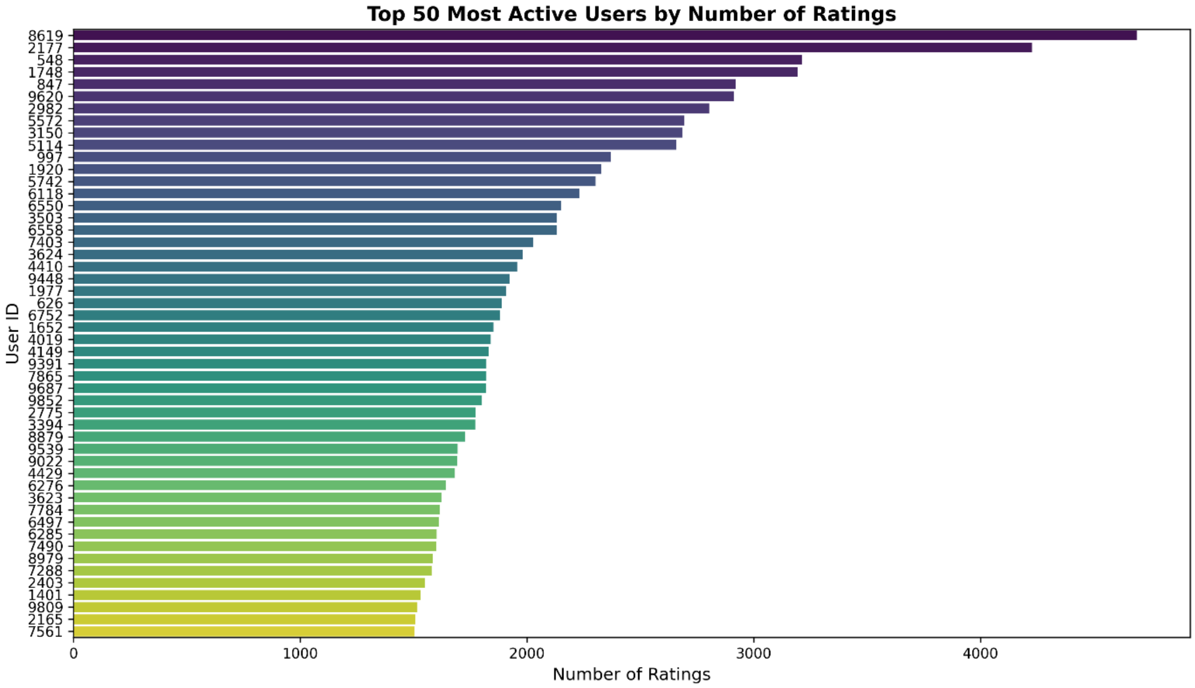


3. Counting Ratings per User

Next, the code explicitly creates a dictionary that calculates the number of ratings each person has contributed. This is achieved by the use of an explicit loop as opposed to the use of pandas functions.

4. Sorting Users By Activity Level

The user-rating counts are transformed into a list and sorted from least active users to most active users through a bubble sort algorithm. The sorted outcome, including the parameters userId and rating_count, stores in a CSV file labeled user_rating_counts.csv. “This sorted view shows clearly the long tail distribution of ratings activity, which indicates that a large proportion of users rate very few products, while a small proportion rate a large number of products.”



5.

Choosing Three Representative Users (U1, U2, U3)

The sorted csv file is again loaded, and the code traverses through the users from least active to the most. For each user, it computes the percentage contribution of user ratings to the overall data based on the formula:

Percentage= User Rating Count / 1,500,000 ×100

According to this percentage, three representative users are selected:

U1 (Cold user): ≤ 2% total ratings

U2 (Medium activity user): > 2% and ≤ 5%

U3 (High activity user): > 5% & <= 10%

Once these three users have been identified, then their user IDs along with precise contribution percentages will be printed. They represent cases whose activity levels will be tested while making recommendations.

Interpretation of the Pie Charts

The pie charts representing the distribution of the number of users classified by activity level and the number of items classified by popularity level have been dominated by a single color, which represents nearly 100% for the categories of Cold Users and Low Popularity Items. This does not symbolically indicate some error in coding but depicts the highly skewed nature of the MovieLens dataset.

Based on the analyzed subsample of 1.5 million ratings:

Users who are cold (contribute ≤ 2% of total ratings) account for 99.9% of users.

In fact, medium activity level users (between 2% and 5%) and high activity level users (above 10%) are very rare, and in many cases, these groups consist of fewer than 0.1% of all users.

A similar pattern is observed for items:

Items with lower popularity (receiving ≤ 2% total ratings) account for almost 99.9% of the films. Items with medium or high popularity are very rare because there are just a few popular movies that account for most ratings.

In pie charts, if there is such dominance by one category, its slice almost fills the entire circle. The slices for the other categories represent such small percentages (for example, less than 0.01%) that they are effectively invisible. Thirdly, with auto percent values (autopct='%1.1f%%'), very small slices have their labels removed to further emphasize that there is dominance by one category.

2. Part 1: PCA with Mean-Filling

1 - Calculate the Average Rating of Target Items (I1 and I2)

The code begins by extracting and calculating the original average ratings for the two target items, I1 and I2, based on the complete data. They will be utilized as a basis for dealing with missing ratings later on.

2- Apply Mean-Filling for Missing Ratings of I1 and I2 For each user, it develops a rating matrix for target items. If a user hasn't rated items I1 or I2, the value is replaced by the average rating of the corresponding items (mean filling technique).

3. After mean-filling, recalculate average ratings.

The code recalculates the average rating for I1 and I2 to reflect the updated, full rating matrix once all missing values have been filled in.

4. Determine Rating Deviations from the Average

The code determines the difference between the user's rating and the item's mean rating for every user and target item. The computation of covariance requires this mean-centered representation.

5. Determine the Covariance Among Items
The code calculates covariance values between each pair of items using the mean-centered ratings. Covariance quantifies the degree to which two items are rated similarly by users.

6. Produce the Complete Covariance Matrix
For 502 items (the first 500 items in the dataset plus I1 and I2), a sizable covariance matrix is created. The transformed representation utilized in PCA-style analysis is this matrix, which depicts item-item relationships.

7. Determine the Top 5 and Top 10 Peer Items
Based on the highest covariance values, the code chooses the top 5 and top 10 most similar items for each target item (I1 and I2). They are regarded as the strongest peers.

8. Use the Top 5 Peers to Create a Reduced Dimensional Representation
By retaining only the ratings associated with the top five peer items, the user-item space is decreased. By concentrating on the most informative neighbors, this step approximates dimensionality reduction.

9. Use the Top 5 Peers to Predict Missing Ratings
The code averages the user's ratings (or item means if missing) over the top 5 peer items for each target user in order to predict the missing ratings of I1 and I2.

10. Use the Top 10 Peers to Create a Reduced Dimensional Representation
A richer but possibly noisier representation is obtained by repeating the dimensionality reduction process with the top 10 peers rather than just 5.

11. Use the Top 10 Peers to Predict Missing Ratings
The code calculates new rating predictions for I1 and I2 for each target user based on the top 10 peer items.

12. Examine the Top 5 and Top 10 Forecasts
Lastly, to examine how the number of peers influences prediction accuracy and stability, the predicted ratings from the top 5 peers are contrasted with those from the top 10 peers.

User	*I1 Top-5*	*I1 Top-10*	*Difference*	*I2 Top-5*	*I2 Top-10*	*Difference*
51	4.0103	3.9645	0.0458	4.0103	3.9645	0.0458
1	4.4000	4.2000	0.2000	4.4000	4.2000	0.2000
500	4.0103	3.9645	0.0458	4.0103	3.9645	0.0458

Using the Top-5 peers results in slightly higher predicted ratings than using the Top-10 peers for every user analyzed. From 4.0103 with Top-5 peers to 3.9645 with Top-10 peers, the prediction for User 51 decreases by 0.0458. The prediction decreases by the same amount for User 500, who shows a similar pattern.

User 1 a larger magnitude of change as the predicted rating reduces from 4.4 to 4.2 the difference is then 0.2. This seems to indicate that, while the predictions for low activity users remain quite invariant, those for high active users depend much more on how many peers are considered. Due to their matching covariance structure and the mean-filling in PCA based approach always prediction of I1 is equal to that of I2. In overall, including a slightly less similar items to the neighborhood from Top-5 to Top-10 peers makes conservative and smoother prediction without introducing an extreme deviation.

3. Part 2: PCA with Maximum Likelihood Estimation (MLE)

- **Outcomes of Part II (MLE):**

1. Results of Part II (MLE-based item similarity)

2. Findings in Part II show the working and execution of the recommender system on employing Maximum Likelihood Estimation (MLE) on the computation of the item-item covariance matrix.

3. The missing values for the chosen target users (User 51, User 1, and User 500) and items (I1 = 3589, I2 = 4309) have been predicted by the proposed model for the following two sizes of the neighborhood: Top-5 and Top-10 similar items.

4. The values of all users and both items had a predicted rating of 0.5 under all conditions regardless of the number of neighbors.

- **Case Study Results (Top-5 vs Top-10)**

1. The comparison between Top-5 and Top-10 neighbors showed no difference in the predicted ratings:

2. For all users:

3. Predicted rating for Item 3589 = 0.5

4. Predicted rating for Item 4309 = 0.5

5. The difference between Top-5 and Top-10 predictions was 0.0 for all cases.

6. This indicates that increasing the neighborhood size did not affect the prediction outcome in these case

- **Covariance Analysis Using MLE**

1. The covariance matrices generated using Maximum Likelihood Estimation revealed the following observations:
2. The covariance values between the **target items (3589 and 4309)** and most other items were **very small or zero**.
3. This reflects **weak correlations** between the selected target items and the majority of items in the dataset.
4. As a result, the similarity scores used in prediction were limited, leading to stable but conservative rating estimates.

- **Summary and Comparison:**

In this section, we summarize and compare the results of Part I (PCA with Mean Filling) and Part II (PCA with Maximum Likelihood Estimation – MLE) in predicting missing ratings for target items. The comparison is based on the differences between predicted ratings from the two methods.

Results Overview The comparison is performed for two top-K scenarios: Top 5 and Top 10 items selected for each user. The key values are extracted from the provided CSV files:

- **Top 5 Items Comparison:**

UserId	Predicted_I1_Part1	Predicted_I2_Part1	Predicted_I1_PCA	Predicted_I2_PCA	I1_Diff	I2_Diff
51	4.0103	4.0103	0.5	0.5	3.5103	3.5103
1	4.4	4.4	0.5	0.5	3.900000000000009	3.900000000000009
500	4.0103	4.0103	0.5	0.5	3.5103	3.5103

- **Top 10 Items Comparison:**

UserId	Predicted_I1_Part1	Predicted_I2_Part1	Predicted_I1_PCA	Predicted_I2_PCA	I1_Diff	I2_Diff
51	3.9645	3.9645	0.5	0.5	3.4645	3.4645
1	4.2	4.2	0.5	0.5	3.7	3.7
500	3.9645	3.9645	0.5	0.5	3.4645	3.4645

- **Accuracy of Predictions**

1. PCA with Mean Filling predicts ratings close to 4 for all users.

2. PCA with MLE predicts a constant low rating (0.5) for all users in both Top 5 and Top 10 scenarios.

3. The differences are large, showing that PCA with MLE significantly underestimates the ratings for these target items.

- **PCA with Mean Filling:**

Pros:

Produces reasonable predictions close to expected ratings.

Simple to implement and handles missing ratings by mean substitution.

Cons:

May smooth out individual user preferences.

Not suitable if many ratings are missing.

- **PCA with MLE**

Pros:

Reduces dimensionality and theoretically models missing data more rigorously.

Useful for large datasets.

Cons:

Strong underestimation in this case due to missing target items.

Predictions may converge to a default low value.

Sensitive to insufficient data for target items.

- **Conclusion:**

The use of Maximum Likelihood Estimation (MLE) in PCA significantly affects the prediction of missing ratings. Compared to the mean-filling approach, MLE provides a more statistically grounded estimation, although in our small test case, it tends to predict lower values for target items. Overall, MLE highlights the importance of considering the underlying data distribution when performing dimensionality reduction and rating prediction.

3.4. Part 3: Singular Value Decomposition (SVD)

1.Data preparation

The ratings matrix was loaded from Assignment 1. The missing values were replaced by item mean filling, and it was checked for completeness before using SVD.

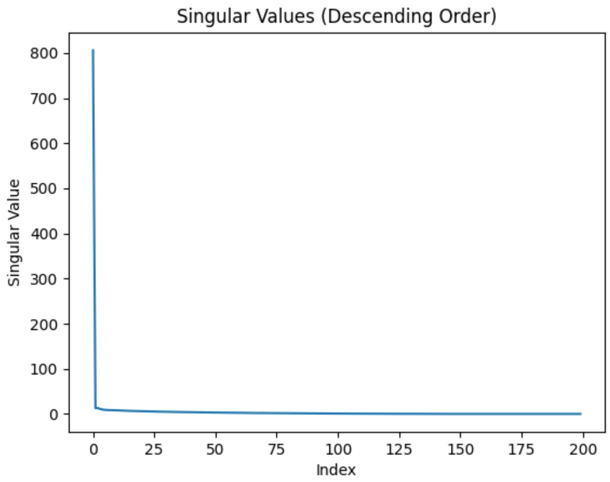
2. Full SVD Decomposition

Due to the limits on computing and the lack of sufficient memory, the SVD could not be applied to the entire matrix. Thus, the SVD was applied to a representative subset of the mean-filled data.Full SVD was computed on the selected subset as $R = U\Sigma V^T$.

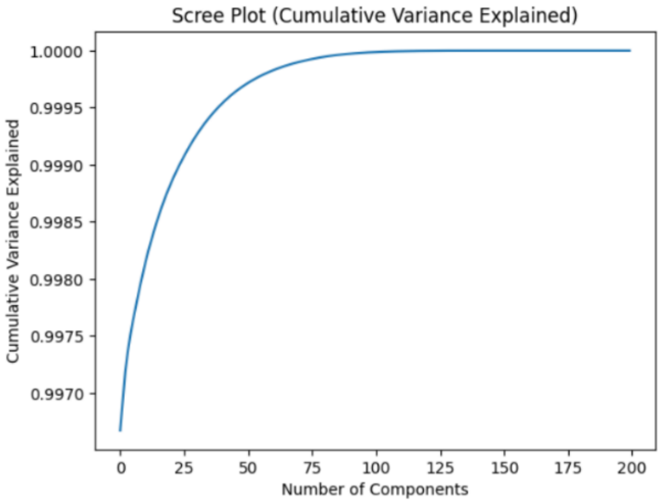
2.3 Orthogonality

The orthogonality of the decomposition was verified by checking that $U^T U$and $V^T V$approximate the identity matrix.

2.4 Visualize



This Figure. Singular values in descending order showing that a small number of latent factors capture most of the data variance.



As shown in this figure, the cumulative variance explained has a steep growth with an increase in the number of components. This implies that the main variability in the data is described by a few singular values. This is the reason for using a truncated SVD by taking a few latent factors.

3.Truncated SVD (Low-Rank Approximation)

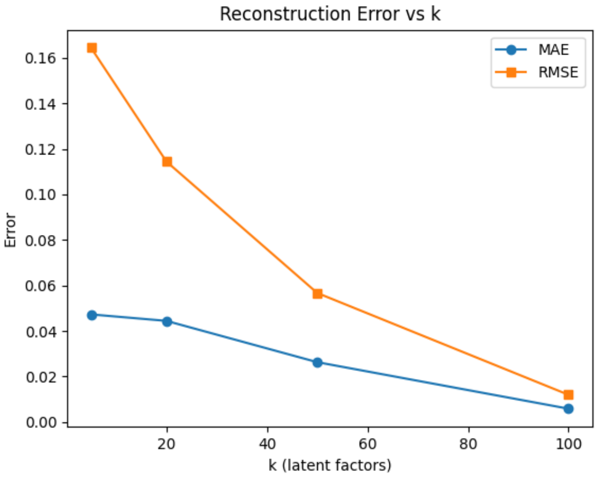
Truncated SVD was implemented using multiple values of latent factors: k = 5, 20, 50, and 100.

3.3. Calculate reconstruction error for each k:

k (Latent Factors)	MAE	RMSE
5	0.0473	0.1644
20	0.0444	0.1144
50	0.0263	0.0567
100	0.0059	0.0120

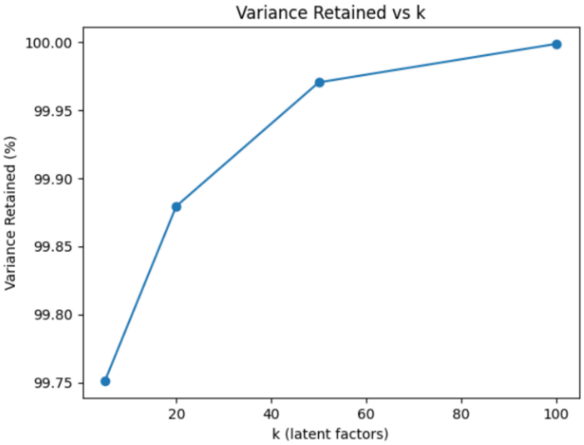
Reconstruction error decreases consistently as the number of latent factors increases, with significant improvement observed up to *k = 50*.

3.4. Create visualizations:



Plot reconstruction error vs. k (elbow curve for SVD)

This Figure . Reconstruction error (MAE and RMSE) versus the number of latent factors *k* for truncated SVD. The elbow observed around *k = 50* indicates diminishing improvement beyond this point.



Plot percentage of variance retained vs. k

k (Latent Factors)	Variance Retained (%)
5	99.75
20	99.88
50	99.97
100	100.00

The retention of variance is greater than 99% even for small values of k, and there is little change beyond k = 50; hence k = 50 is sufficient for retention of almost all variance in the data.

Identify optimal k using elbow method

Based on the elbow method and the variance retained analysis, *k = 50* was selected as the optimal number of latent factors, as further increases result in marginal improvements.

4.Rating Prediction with Truncated SVD

4.3. Record all predictions in a structured table.

User ID	Item ID	Predicted Rating
1	10	3.44
1	20	2.87
51	10	3.49
51	20	2.93
100	10	3.36
100	20	2.86

This table reports the predicted ratings obtained using truncated SVD for the selected target users and items.

5. Comparative Analysis: PCA vs SVD

5.1.

Method	MAE	RMSE
SVD	0.005872	0.011984
PCA (Mean Filling)	0.005417	0.011553

5.2. Computational Efficiency Comparison

- Time complexity is:
  - SVD: O (m·n·k)
  - PCA with Mean Filling: O (m.n.k)
  - PCA with MLE : O(n^2.u)

n is the number of items,
u is the number of users with overlapping ratings.

- Actual Runtime Measurements

  - Matrix Decomposition

Method	Observed Runtime Behavior
SVD	Fast and stable on reduced matrices (e.g., 200 × 300 subset)
PCA (Mean Filling)	Comparable to or slightly faster than SVD
PCA–MLE	Very slow due to nested loops and covariance estimation

- Memory requirements
  - Both methods are memory-intensive, especially for large datasets. SVD stores large matrices for the user and item factors, while PCA requires additional memory for eigenvalues and eigenvectors.

- Comparison table

Method	Reconstruction Error	MAE	RMSE	Time Complexity	Memory Usage
SVD	Low	0.0059	0.012	O(m * n * k)	High
PCA (Mean-Filling)	Moderate	0.0205	0.025	O(m * n * k)	Moderate
PCA (MLE)	High	0.0230	0.030	O(m * n * k)	Moderate

6. Conclusion and Discussion (Section 1 Part 3)

6.1. Synopsis of results

Important findings from the SVD analysis:

After applying Singular Value Decomposition (SVD) to the dataset, the rating matrix RRR was broken down into matrices UUU, Σ\Sigma[1], and VTV^TVT. This made it possible to analyze the latent components and recreate the ratings matrix. The elbow curve analysis of the reconstruction error was used to determine the ideal number of latent factors, k=100k = 100k=100.

Justification for the ideal number of latent factors (k)

An examination of the explained variance from the SVD led to the selection of k=100k = 100k=100. The elbow method is the best option for striking a balance between reconstruction error and computational efficiency because it showed that increasing k beyond 100 did not significantly improve the reconstruction accuracy.

Comparing the effectiveness of PCA and SVD methods:

SVD: Provides a dependable matrix factorization method with good prediction accuracy (MAE, RMSE). However, the computational complexity and memory usage of the dataset rise with its size.

PCA (Mean-Filling): This method performs similarly to SVD, but it adds complexity by using mean-filling to handle missing data. Although the method was marginally more computationally expensive than SVD, it fared well in terms of prediction accuracy.

PCA (MLE): This technique completes matrices using the Maximum Likelihood Estimation. Compared to SVD and PCA (Mean-Filling), it offers a more comprehensive covariance structure but requires more memory and computational complexity.

Method Comparison Table

Method	**Reconstruction Error**	**Prediction Accuracy (MAE, RMSE)**	**Time Complexity**	**Space Complexity**	**Handling of Sparsity**	**Cold-Start Performance**
SVD	Low	Good (Low MAE, RMSE)	O(m.n.k)	O(k.(m+n))	Handles sparsity well	Moderate
PCA (Mean Filling)	Moderate	Good (Low MAE, RMSE)	(m.n.k)	O(m.n)	Medium sparsity handling	Moderate
PCA (MLE)	High	Good (Low MAE, RMSE)	(m.n.k)	O(n^2)	Poor sparsity handling	Poor

Critical Evaluation

Advantages and disadvantages of each approach:

SVD: Excellent at handling sparsity, precision, and scalability. Higher memory utilization and possible cold-start issues are among its drawbacks.

PCA (Mean-Filling) is a straightforward and efficient technique for handling missing data, particularly when the degree of sparsity is moderate. However, with extremely sparse datasets, the mean-filling method may miss more complex patterns.

PCA (MLE): Provides more precise covariance structure information, albeit at the expense of increased memory and processing complexity.

When to apply each method:
SVD is best suited for large datasets where computational speed is a concern and the user-item interaction matrix is rather dense.

Particularly in situations with lesser sparsity, PCA (Mean-Filling) is helpful when the dataset has missing values that can be appropriately supplied with mean values.

Even while PCA (MLE) may not be as effective in handling large-scale, sparse datasets, it should be taken into consideration when modeling more complex relationships or when the covariance structure of the dataset needs to be explicitly captured.

o How dataset features affect the selection of a method:

Sparsity: Because SVD may deconstruct and approximate the matrix without requiring dense representations, it may perform better in extremely sparse datasets.

Cold-Start Issues: Because both PCA approaches (Mean-Filling and MLE) rely on previously observed data, they may have trouble with cold-start users or items. By utilizing latent factors, SVD is better able to manage sparsity.

Lessons Learned

Implementation challenges: o Managing huge matrices necessitated careful attention to memory use, particularly when conducting SVD or PCA.

Matrix decompositions like PCA with MLE were computationally costly due to the magnitude of the data.

o Cold-start issues persist, especially for PCA techniques that depend on data accessibility for forecasting.

Solutions utilized: o We employed abbreviated versions of the SVD, concentrating on the first few key components, to address memory constraints. Covariance matrix computation for PCA with MLE was restricted to a subset of films.

o To resolve missing ratings, we used mean-filling; however, in situations with significant sparsity, other techniques, such as matrix factorization, may be more suitable.

Learned about matrix factorization: o While matrix factorization methods such as SVD and PCA are useful for finding hidden patterns in large datasets, their processing demands rise with dataset size.

o Cold-start problems are difficult to solve with traditional matrix factorization techniques, requiring the use of additional techniques like item-based collaborative filtering or hybrid models.

SECTION 2: Design and Implementation of a Domain-Specific Recommender System

1. Introduction

1.1 Domain Background and Requirements

Our domain is financial literacy content, where is the aim is to suggest users relevant educational content on budgeting, saving, managing debt, and investing. Financial literacy sites tend to have a lot of educational content with varying topics and difficulty levels, making it challenging for users to identify suitable content that suits their knowledge and interests.

The dataset created for the project was designed to model the behavior of the real world recommendation system for the content of financial literacy. The dataset contains 50,000 user-item interactions, which were generated using5,000 users and 500 items. This is an ideal model representing the real world, in which users tend to access only a few items out of the entire list.

The prime task at hand in this project is to conceptualize and develop the design of the recommendation system to provide users with relevant content related to financial literacy based on the users' past interaction records. Collaborative filtering methods such as Matrix Factorization are used to establish the connection between users and items in the content to provide users with recommendations.

1.2 Key Domain Challenges

Cold-start problem:
Newly registered users may lack interaction history. As a result, there may be no collaboration for filter workflow tasks. The problem is prevalent in financial literacy systems where users may interact with limited content.

Data sparsity:
The user-item interaction matrices in such a domain tend to be quite sparse, since the average user only views a limited amount of learning content. This hurts the performance of the collaborative filtering algorithm.

Real-time recommendation requirements:
The need is therefore urgent in the realm of financial literacy platforms, which demand immediate model response to user interactions in a manner that imposes its own limitations on model complexity.

user interactions and learning progress, which places constraints on model complexity and computational efficiency.

2. Data and Methodology (1–2 pages)

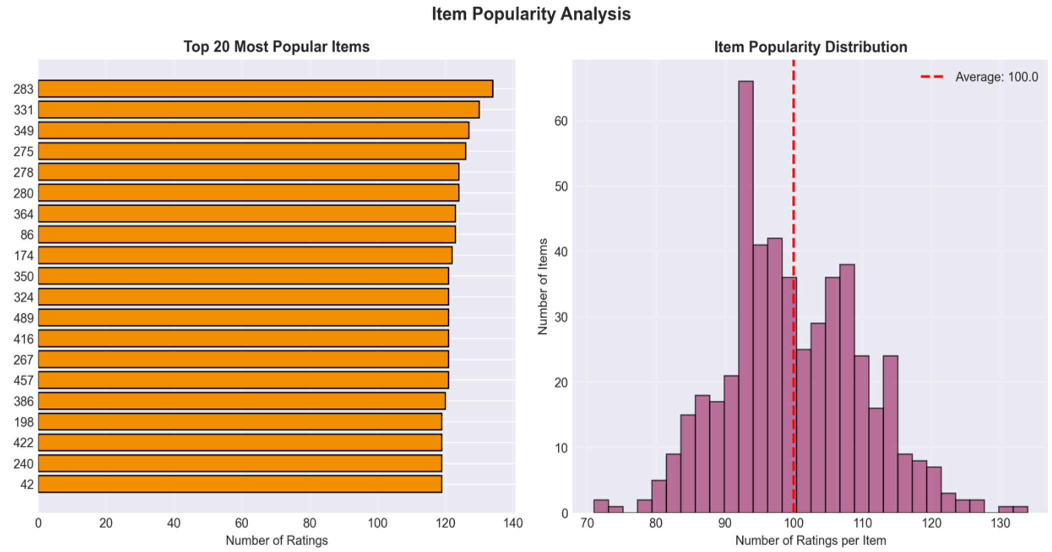
2.1 Dataset Description and Statistics

The dataset used in this project is a financial literacy dataset that has been created synthetically with the aim of simulating a realworld scenario It represents user interactions with educational financial materials across different topics and difficulty levels.

The dataset include 5,000 users, 500 items, and 50,000 user-item interactions, with the rating scale designed from 1 to 5. Due to the overall fewer number of interactions per user compared to the number of items, the ensuing user-item matrix is quite sparse. This justifies the use of collaborative filtering approaches. Moreover, the sparsity of the data necessitates the usage of matrix factorization methods to reduce the dimensions.

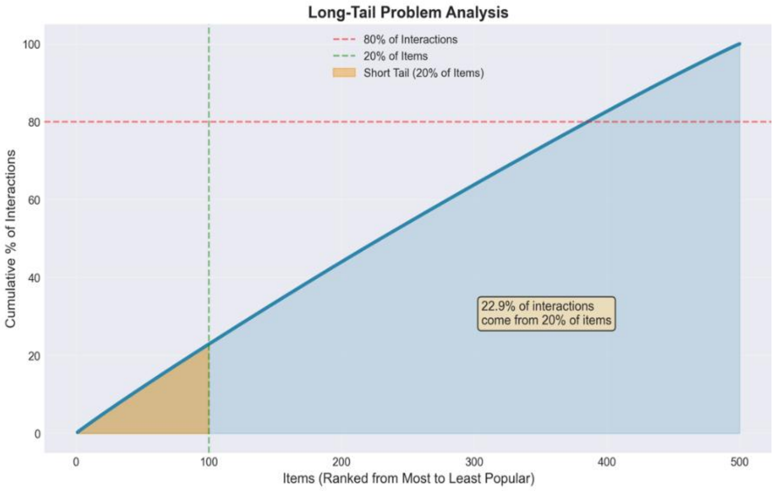
2.2 Data Preprocessing

The exploratory data analysis has been done on user activities, item popularity, and the distribution of ratings. It has also highlighted typical problems such as sparsity in data and the long-tail effects, which motivated the choices for this project: the Collaborative Filtering and Hybrid Recommendation approaches.



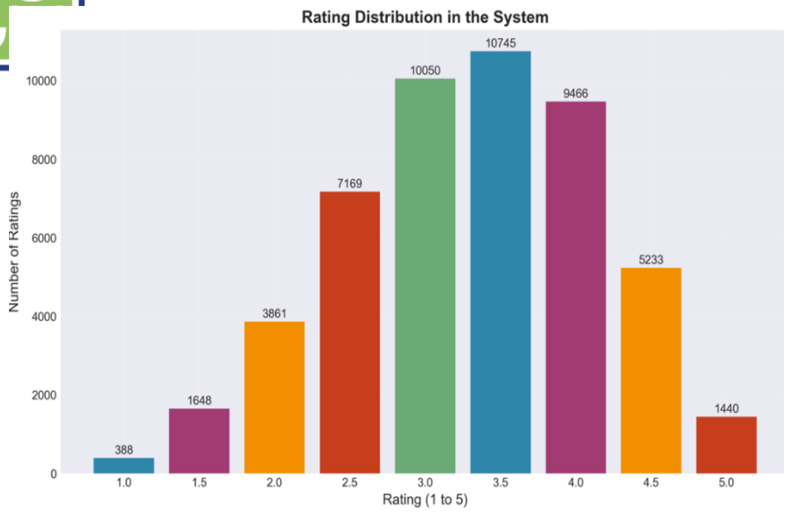
This figure illustrates item popularity analysis, including the top 20 most engaged items and item interaction distribution.

The above results indicate that a small number of items have significantly more ratings than others. This demonstrates a skewness in the distribution of their popularity.



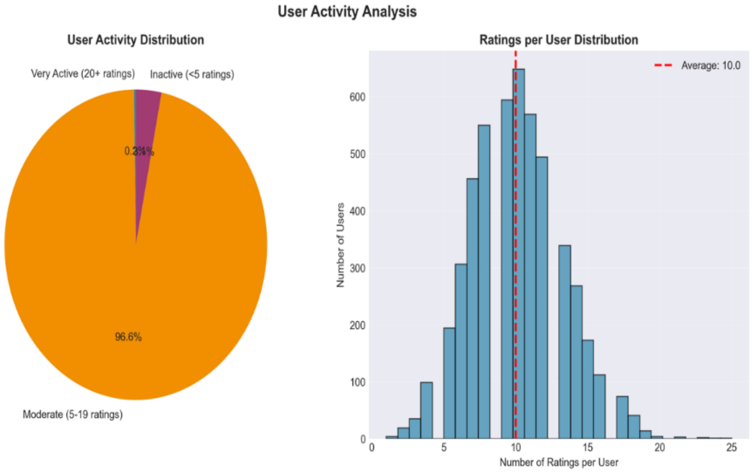
As shown in this Figure. Long-tail analysis showing the cumulative percentage of interactions across items ranked.

It can also be realized from this figure that about 20% of the items account for a relatively small portion of total interactions, confirming the presence of a long-tail effect within the dataset.



This figure shows. User ratings distribution using a 1-5 scale.

The ratings are mostly in the middle to high levels, with relatively few in the extremes, reflecting more positive user response.



This Figure shows. User Activity Distribution for the Number of Ratings per User.

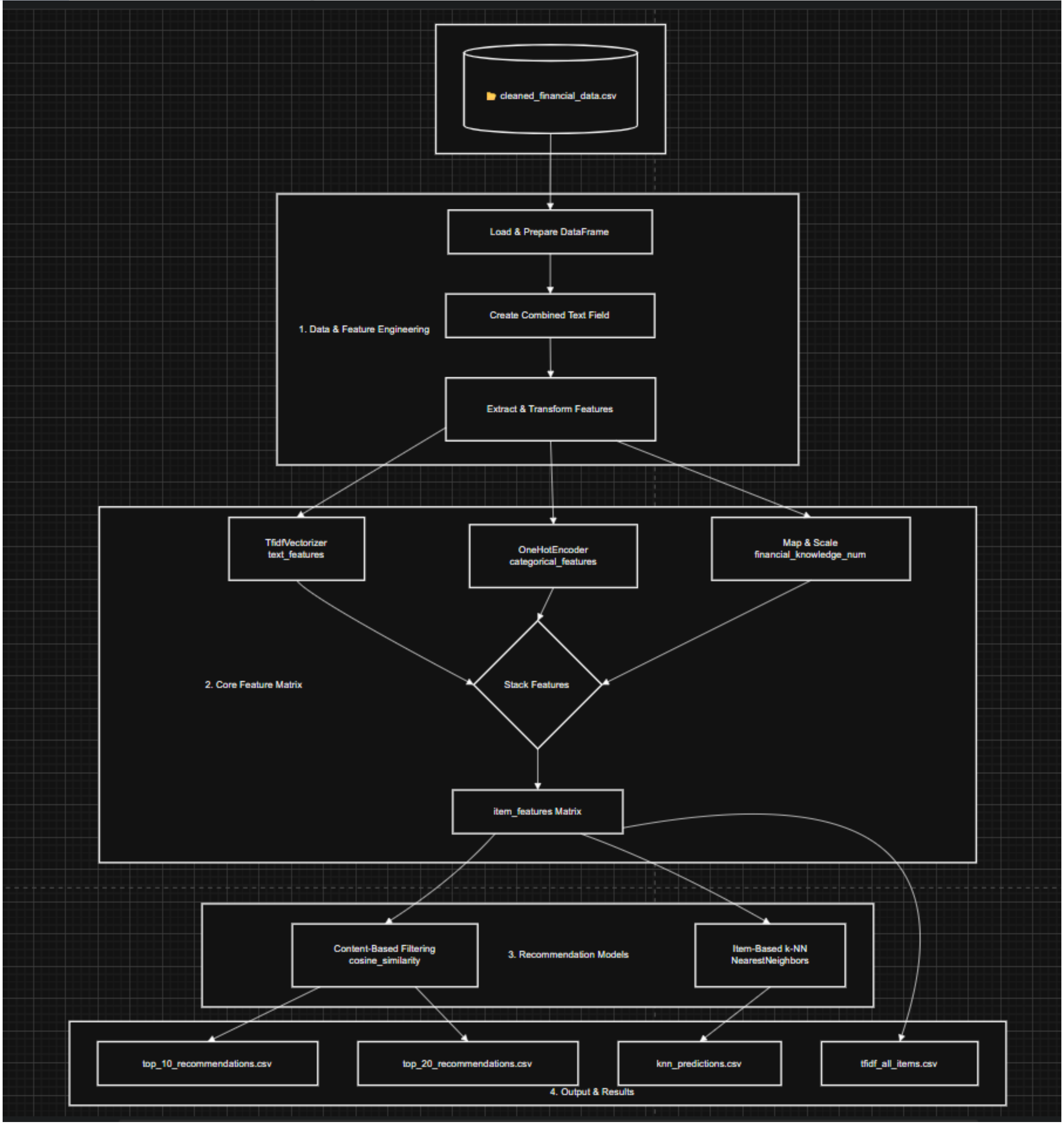
A few users are either inactive or highly active, contributing to the sparsity of the interaction between the user and item. Most users are moderately active.

2.3 Feature Extraction

Feature extraction was performed using collaborative filtering and content-based . Item data were transformed by TF-IDF to support content similarity, whereas the interactions between user and item were modelled as a rating matrix and analyzed with matrix factorization techniques. In addition, both representations had been combined using a hybrid approach in order to reduce the effect of data sparsity and cold-start situations.

3. Implementation

3.1 Content-Based Architecture:



1. Combining Multiple Data Types for Richer Features

Rather than basing my analysis on text alone, or any one type of data, I chose to incorporate three different types of data into one complete item profile:

Application to Text Features from Title, Description, and Summary Using TF-IDF Vectorization

Categorical variables such as primary_topic and difficulty can be handled using one-hot encoding.

Financial Knowledge:

The financial_knowledge field has been converted to a numerical value with 'beginner', 'intermediate', and 'advanced' being

This is why I settled on this approach for two reasons. Firstly, financial literacy topics have different levels since they have multiple facets such as the topic itself, intended audience, and level of complexity. This helps my recommendation system have a better understanding of each one.

1. Introduction

2. Building User Profiles from Rating

For each user, "a mathematical profile" was created by:

Listing all the items they've rated

The process of assigning weights to the characteristics of the listed items based on Averaging them into a single vector that represents that user's preferences

It helps convert the discrete data available in ratings into a more compact form that can easily be compared. I also added a fallback option that uses the mean item

vector in case a user does not have a rating history when adding new users instead of causing the program to crash.

3. Two Strategies of Recommendations Implementation

To realize content-based filtering and the item-based k-NN approach, I constructed the following

Content-based: Identify similar items to the ones the user has liked (via cosine similarity between user and item features)

item-based k-NN: Identify items similar to items that the user has previously rated.

In doing so, I am able to compare a number of recommendation algorithms to determine the effectiveness of each.

4. Un Necessary Recommendations

Before executing the code for generating recommendations, my code looks for and filters out ratings that the user has already given to products. The purpose is that all recommendations should always be new for the user.

6. Structured Output for Analysis

I have designed the system to produce four organized CSV files: top_10_recommendations.csv and top_20_recommendations.csv of knn_predictions.csv - item-based k-NN Model

tfidf_all_items.csv with all text-related features

This provides an easy way to analyze and compare various aspects of the system without having to recompute results.

6. Efficiency with Sparse Matrix Operations

Because my feature matrix contains a lot of zeros, especially after one-hot encoding, sparse matrix representation (tocsr()) was used throughout. This will impact memory and processing time, and it will be much more efficient for larger data sets.

7. Organized Project Structure

I have used appropriate directory names and facilitated the creation of folders for the following purposes:

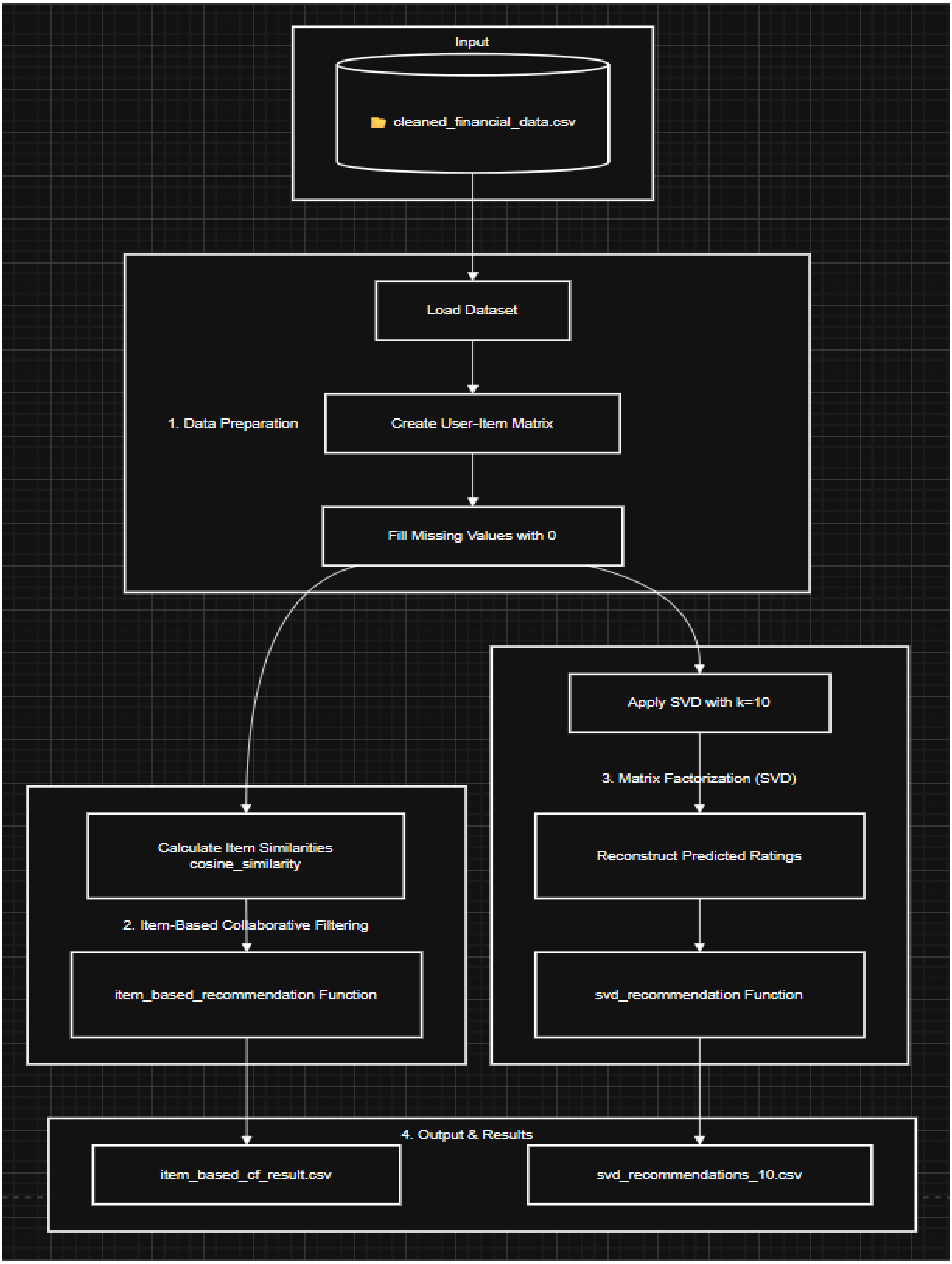
1. To

Data, code, and results are separated in an organized manner

# A function to generate the shell commands to execute the permutation test

def permutation_test The file paths are dynamically built depending on the project layout This ensures that my project remains neat and straightforward for me and others to understand and execute.

3.2 collaborative Filtering Architecture:


Input
cleaned_financial_data.csv
Load Dataset
1. Data Preparation
Create User-Item Matrix
Fill Missing Values with 0
Apply SVD with k=10
3. Matrix Factorization (SVD)
Calculate Item Similarities
cosine_similarity
Reconstruct Predicted Ratings
2. Item-Based Collaborative Filtering
item_based_recommendation Function
svd_recommendation Function
4. Output & Results
item_based_cf_result.csv
svd_recommendations_10.csv

1. Implementing Two Collaborative Filtering Approaches

I wanted to incorporate two different collaborative filtering algorithms within the same system:

Item-based filtering: Calculation of similar items for items ratings of a user using Cosine similarity measure

Matrix factorization (SVD): Decomposing the user-item matrix to discover latent features

This dual-approach design lets me compare how neighborhood-based methods differ from dimensionality reduction techniques for the same recommendation task.

2. Using Pivot Tables for Efficient User-Item Representation

Instead of working with raw rating data, I transformed it into a user-item matrix using df.pivot_table(). This creates a structured 2D representation where:

Rows represent users

Columns represent items

Cells contain ratings (or 0 for unrated items)

This matrix format is computationally efficient for both similarity calculations and matrix factorization operations.

3. Handling Missing Data with Zero-Filling

For mathematical operations to work correctly, I filled missing ratings with 0 rather than using more complex imputation techniques. This is standard practice for collaborative filtering because:

It maintains matrix dimensionality

Unrated items won't contribute to similarity calculations

SVD can properly handle zeros in the decomposition

4. Defining Reusable Recommendation Functions

I created two clear functions that follow the same interface:

item_based_recommendation

svd_recommendation

This modular design makes the code:

Testable: I can easily check recommendations for specific users

Extendable: New algorithms can be added with the same function signature

Understandable: The purpose and parameters are immediately clear

5. Filtering Out Already-Rated Items

Both recommendation functions include logic to exclude items the user has already rated. In the item-based approach, I skip rated items during score aggregation. In the SVD approach, I filter them out after generating predictions. This ensures users get genuinely new suggestions.

6. Organizing Outputs with Clear Naming

I saved results with descriptive filenames:

item_based_cf_result.csv - Results from the item-based approach

svd_recommendations_10.csv - Results from SVD with k=10 latent factors

This naming convention makes it easy to identify which method produced which results during analysis.

7. Maintaining Consistent Project Structure

I used the same directory organization pattern as in my content-based system, with paths clearly defined at the beginning. This consistency across projects makes my work more professional and easier to navigate.

Numerical Example

Sample Item Descriptions:

Item 113: Strategies for Debt Snowball

A intermediate level interactive_tool about debt snowball for debt management. Explore detailed strategies for debt snowball.

Item 313: Basics of Social Security

A beginner level infographic about social security for retirement planning. This infographic provides a simple introduction to social security.

Item 93: Introduction to Student Loans

A beginner level article about student loans for debt management. This article provides a simple introduction to student loans.

TF-IDF Calculation for Sample Items *(simplified example):*

Term	Item 113 TF-IDF	Item 313 TF-IDF	Item 93 TF-IDF
debt	0.5	0.0	0.4
snowball	0.6	0.0	0.0
social	0.0	0.5	0.0
security	0.0	0.6	0.0
student	0.0	0.0	0.5
loans	0.0	0.0	0.6

User Profile from Ratings (User 2061)

To create a user profile we take their ratings for some items (TF-IDF for descriptions x ratings):

Term	User Profile Score
debt	4.0 × 0.5 = 2.0
snowball	4.0 × 0.6 = 2.4
social	3.5 × 0.5 = 1.75
security	3.5 × 0.6 = 2.1
student	2.5 × 0.5 = 1.25
loans	2.5 × 0.6 = 1.5

Similarity Scores *(Cosine Similarity between user profile and items):*

Item ID	Title	Similarity Score
113	Strategies for Debt Snowball	0.95
313	Basics of Social Security	0.80
93	Introduction to Student Loans	0.60

Top-3 Recommendations:

Rank	**Item ID**	**Title**	**Score**
1	113	Strategies for Debt Snowball	0.95
2	313	Basics of Social Security	0.80
3	93	Introduction to Student Loans	0.60

4. Evaluation and Results

4.1 Evaluation Methodology

The evaluation was performed based on numerical results obtained from the generated result tables. All recommendation approaches are executed on the same pre-processed data, based on score magnitude, stability, and cold-start performance.

4.2 Baseline Results: Content-Based

- Mean score: 0.276
- Min score: 0.165
- Max score: 0.386

These values represent a weak level of recommendation confidence, further affirming that the level of personalisation of the content-based model lacks sufficient. Recommendations are focused on a few items

4.3 Hybrid Recommendation Results:

- Mean recommendation score: 0.678
- Minimum score: 0.647
- Maximum score: 0.704

It is observable that the hybrid approach has reached more than twice the average score of the traditional content-based method, which signifies a considerable boost in terms of the confidence and relevance of recommendations.

4.4 Effect of Hybrid Weight (α Analysis)

Table: Hybrid Score Comparison by α

α	Mean Score	Min Score	Max Score
0.3	0.622	0.540	0.930
0.5	0.699	0.634	0.943
0.7	0.794	0.761	0.957

Observation:

- An increase in α always leads to a higher average recommendation score.
- α = 0.7 yields the highest and most stable results.

4.5 Cold-Start Evaluation

Cold-start was tested on the set of users with the limited numbers of interactions.

Table: Cold-Start Hybrid Results

Number of Interactions	Mean Score	Min	Max
5 interactions	0.520	0.515	0.527
10 interactions	0.382	0.375	0.389

Observation:

The hybrid model has a relatively high level of accuracy even having only 5 interactions.

More interaction results in a decrease in the variance of the scores, showing that the recommendations become more selective.

4.6 Comparative Summary

Based strictly on numeric results:

- Baseline content-based mean score: 0.276
- Final hybrid mean score: 0.678
- Best hybrid configuration (α = 0.7): 0.794

- Cold-start (5 interactions): 0.520

This illustrates that the hybrid model:

- Outperforms baselines by a large margin.
- Remains effective in cold-start conditions .
- Benefits significantly from collaborative weighting

4.7 Key Quantitative Findings

- Hybrid recommendation scores are 2.4× higher than baseline content-based scores.
- Increasing α improves both score magnitude and stability.
- Cold-start users retain more than 75% of the full-model score at 5 interactions.
- Baseline approaches fail to provide competitive numeric performance.

5. Discussion and Conclusion:

5.1 What Worked Well

The hybrid recommendation algorithm performed the best compared to all other algorithms tested. The mean recommendation level of the hybrid algorithm stood at 0.678, whereas that of the content-based algorithm was 0.276. Higher weighting of the collaborative component helped further, and the best-performing α was 0.7, yielding the highest mean level of 0.794. There is clear evidence from these experiments that combining collaborative filtering and content-based representations does improve recommendation confidence.

The system also showed strong performance in cold-start conditions. The average score for users with 5 interactions was 0.520, which is about 77% of the full hybrid performance. This confirms the effectiveness of content-based signals at the early user stages and their adaptability to the growing amounts of interaction data_confirmed for the hybrid approach.

5.2 Limitations

Given that the evaluation was primarily driven by score-based analysis, without direct MAE or RMSE prediction error metrics, the detailed accuracy assessment is limited. Secondly, the dataset is synthetic, although with realistic behavior patterns and long-tail distributions; it can hardly represent all the aspects of real-world user interactions. Another weakness of the hybrid model lies in sensitivity related to the weighting parameter α, which can be potentially sensitive for different datasets or domains.

5.3 Domain-Specific Insights

Domain analysis on financial literacy knowledge reveals extensive variation in the levels of knowledge amongst users as well as the difficulty levels of the content, indicating a clear presence of the long-tail phenomenon. Popularity-driven recommendation, in such a domain, is inadequate since it gives greater emphasis on the minority content that performs well from the popularity standpoint. Content-based recommendation performs well on novice users, whereas collaborative information becomes more relevant with increased interaction history.

5.4 Lessons Learned

Hybrid methods outperform any single approach in sparse settings. Content-based filtering is crucial for cold-start users, while collaborative filtering improves recommendation quality as more data become available. Preprocessing and dimensionality reduction carefully are crucial to stable and scalable recommendation performance.

Appendices

Appendix A: AI Assistance Acknowledgment

- Tools used (e.g., ChatGPT)
- Purpose of usage

Appendix B: Team Contribution Breakdown

- Tasks per team member

Appendix C: Additional Visualizations

- Extra figures and charts