GALALA UNIVERSITY
Powered by
Arizona State University

AIE425 Intelligent Recommender Systems, Fall Semester 25/26

Final Course Project

Group 5

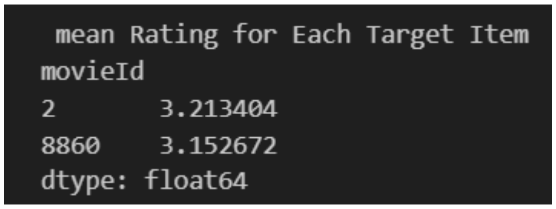
Submission Date: Monday, January 5, 2026

Name	ID
Menna Salem Elsayed	221101277
Arwa Ahmed Mostafa	221100209
Hala Soliman	222102480
Malak Amgad	221100451

Section 1: Dimensionality Reduction and Matrix Factorization

Part 1: PCA Method with Mean-Filling:

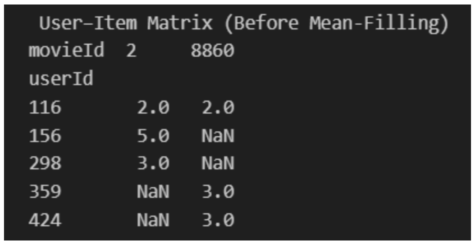
1) Average Rating for Each Target Item



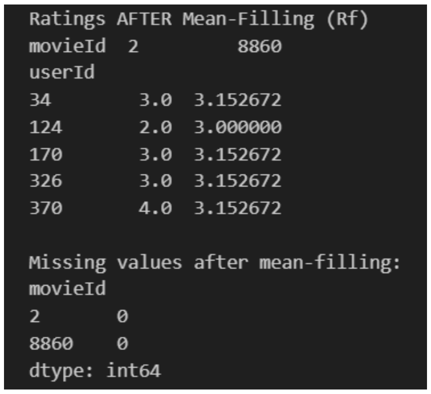
```
 mean Rating for Each Target Item
movieId
2       3.213404
8860    3.152672
dtype: float64
```

2) Mean-Filling of Missing Ratings

Fill all missing ratings with mean to ensures the rating matrix becomes complete with no missing values.

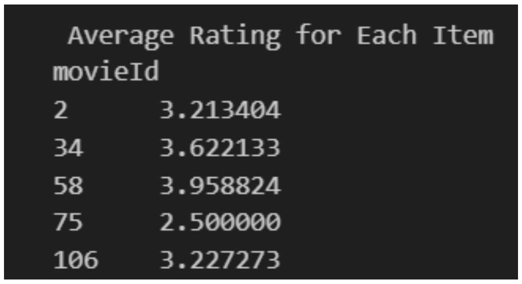


```
 User-Item Matrix (Before Mean-Filling)
movieId  2     8860
userId
116       2.0   2.0
156       5.0   NaN
298       3.0   NaN
359       NaN   3.0
424       NaN   3.0
```



```
Ratings AFTER Mean-Filling (Rf)
movieId  2         8860
userId
34        3.0  3.152672
124       2.0  3.000000
170       3.0  3.152672
326       3.0  3.152672
370       4.0  3.152672

Missing values after mean-filling:
movieId
2       0
8860    0
dtype: int64
```

3) Average Rating for Each Item

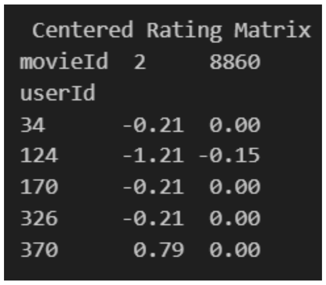


```
 Average Rating for Each Item
movieId
2      3.213404
34     3.622133
58     3.958824
75     2.500000
106    3.227273
```

4) **Mean Centering (Difference from Item Mean)**

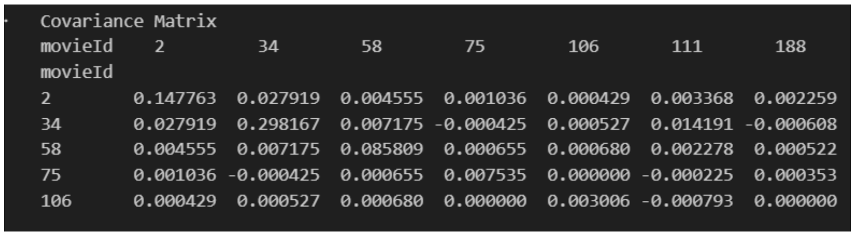
Each value = Rating - Mean(Item)

X_centered = Rf - item_means_target



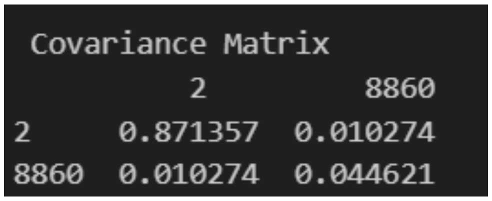
```
 Centered Rating Matrix
movieId  2     8860
userId
34      -0.21  0.00
124     -1.21 -0.15
170     -0.21  0.00
326     -0.21  0.00
370      0.79  0.00
```

5) **Compute the covariance for each two items.**



```
Covariance Matrix
movieId   2         34        58        75        106       111       188
movieId
2        0.147763  0.027919  0.004555  0.001036  0.000429  0.003368  0.002259
34       0.027919  0.298167  0.007175 -0.000425  0.000527  0.014191 -0.000608
58       0.004555  0.007175  0.085809  0.000655  0.000680  0.002278  0.000522
75       0.001036 -0.000425  0.000655  0.007535  0.000000 -0.000225  0.000353
106      0.000429  0.000527  0.000680  0.000000  0.003006 -0.000793  0.000000
```

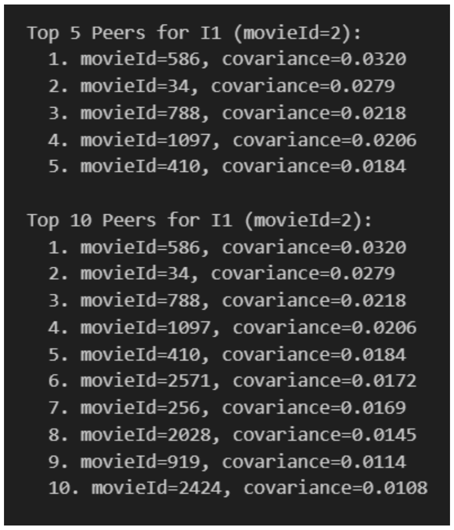
6) **Generate the covariance matrix.**



```
 Covariance Matrix
             2         8860
2     0.871357  0.010274
8860  0.010274  0.044621
```

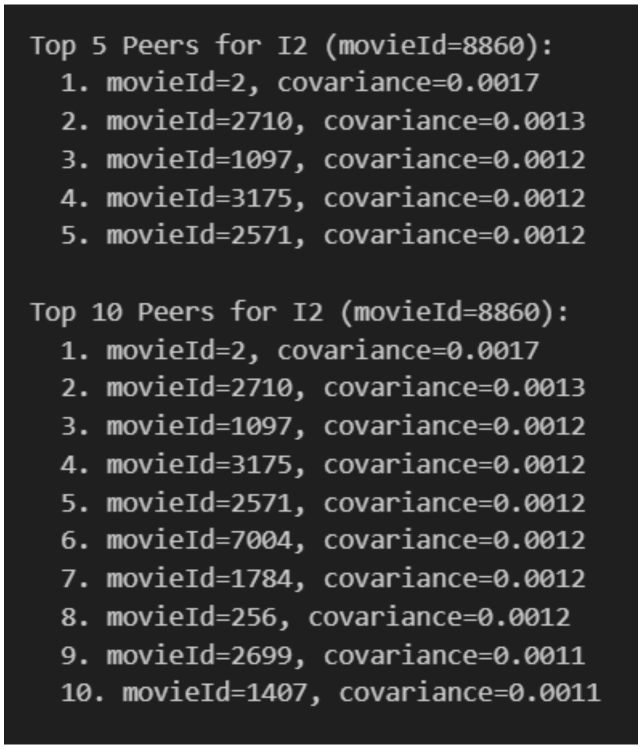
7) **Determine the top 5-peers and top 10-peers**

Top 5 Peers for I1 (movieId=2):



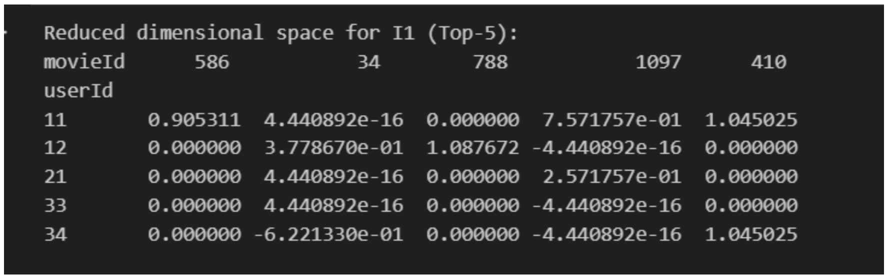
```
Top 5 Peers for I1 (movieId=2):
  1. movieId=586, covariance=0.0320
  2. movieId=34, covariance=0.0279
  3. movieId=788, covariance=0.0218
  4. movieId=1097, covariance=0.0206
  5. movieId=410, covariance=0.0184

Top 10 Peers for I1 (movieId=2):
  1. movieId=586, covariance=0.0320
  2. movieId=34, covariance=0.0279
  3. movieId=788, covariance=0.0218
  4. movieId=1097, covariance=0.0206
  5. movieId=410, covariance=0.0184
  6. movieId=2571, covariance=0.0172
  7. movieId=256, covariance=0.0169
  8. movieId=2028, covariance=0.0145
  9. movieId=919, covariance=0.0114
  10. movieId=2424, covariance=0.0108
```

Top 5 Peers for I2 (movieId=8860):

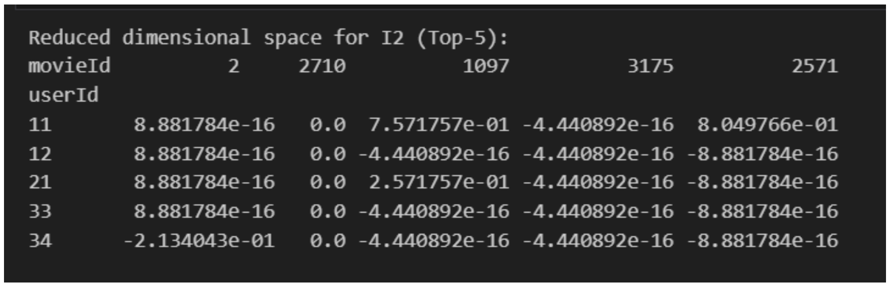


```
Top 5 Peers for I2 (movieId=8860):
  1. movieId=2, covariance=0.0017
  2. movieId=2710, covariance=0.0013
  3. movieId=1097, covariance=0.0012
  4. movieId=3175, covariance=0.0012
  5. movieId=2571, covariance=0.0012

Top 10 Peers for I2 (movieId=8860):
  1. movieId=2, covariance=0.0017
  2. movieId=2710, covariance=0.0013
  3. movieId=1097, covariance=0.0012
  4. movieId=3175, covariance=0.0012
  5. movieId=2571, covariance=0.0012
  6. movieId=7004, covariance=0.0012
  7. movieId=1784, covariance=0.0012
  8. movieId=256, covariance=0.0012
  9. movieId=2699, covariance=0.0011
  10. movieId=1407, covariance=0.0011
```

8) Determine reduced dimensional space for each user in case of using the top 5-peers

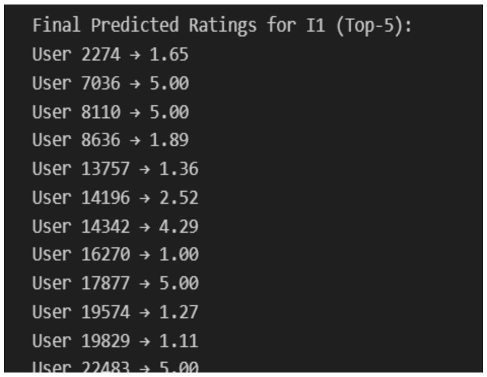


```
Reduced dimensional space for I1 (Top-5):
movieId      586          34        788          1097      410
userId
11       0.905311  4.440892e-16  0.000000  7.571757e-01  1.045025
12       0.000000  3.778670e-01  1.087672 -4.440892e-16  0.000000
21       0.000000  4.440892e-16  0.000000  2.571757e-01  0.000000
33       0.000000  4.440892e-16  0.000000 -4.440892e-16  0.000000
34       0.000000 -6.221330e-01  0.000000 -4.440892e-16  1.045025
```



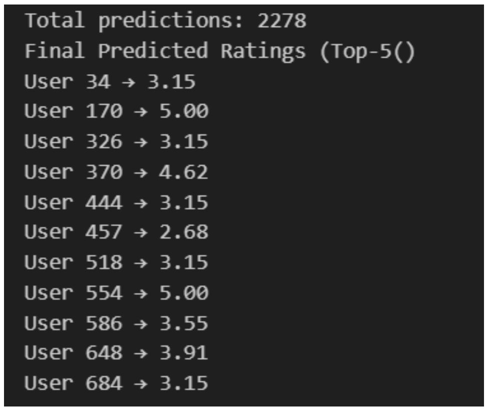
```
Reduced dimensional space for I2 (Top-5):
movieId          2     2710          1097          3175          2571
userId
11       8.881784e-16   0.0  7.571757e-01 -4.440892e-16  8.049766e-01
12       8.881784e-16   0.0 -4.440892e-16 -4.440892e-16 -8.881784e-16
21       8.881784e-16   0.0  2.571757e-01 -4.440892e-16 -8.881784e-16
33       8.881784e-16   0.0 -4.440892e-16 -4.440892e-16 -8.881784e-16
34      -2.134043e-01   0.0 -4.440892e-16 -4.440892e-16 -8.881784e-16
```

- Predicted Missing Ratings for I1 (Top-5 Peers):



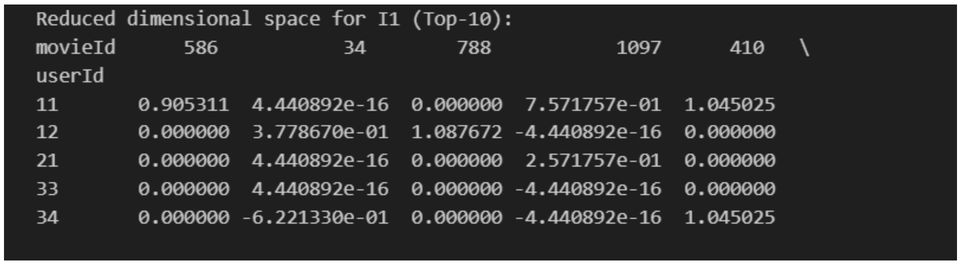
```
Final Predicted Ratings for I1 (Top-5):
User 2274 → 1.65
User 7036 → 5.00
User 8110 → 5.00
User 8636 → 1.89
User 13757 → 1.36
User 14196 → 2.52
User 14342 → 4.29
User 16270 → 1.00
User 17877 → 5.00
User 19574 → 1.27
User 19829 → 1.11
User 22483 → 5.00
```

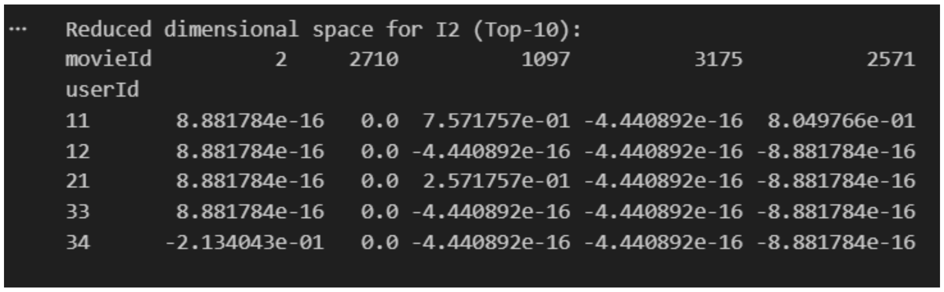
- Predicted Missing Ratings for I2 (Top-5 Peers):



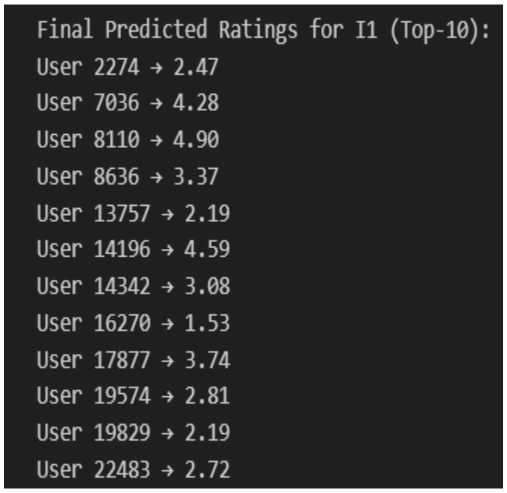
```
Total predictions: 2278
Final Predicted Ratings (Top-5()
User 34 → 3.15
User 170 → 5.00
User 326 → 3.15
User 370 → 4.62
User 444 → 3.15
User 457 → 2.68
User 518 → 3.15
User 554 → 5.00
User 586 → 3.55
User 648 → 3.91
User 684 → 3.15
```

9) **Determine reduced dimensional space for each user in case of using the top 10-peers.**

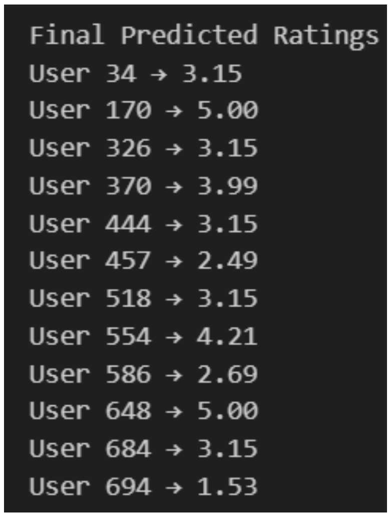

```
Reduced dimensional space for I1 (Top-10):
movieId      586          34       788          1097      410   \
userId
11       0.905311  4.440892e-16  0.000000  7.571757e-01  1.045025
12       0.000000  3.778670e-01  1.087672 -4.440892e-16  0.000000
21       0.000000  4.440892e-16  0.000000  2.571757e-01  0.000000
33       0.000000  4.440892e-16  0.000000 -4.440892e-16  0.000000
34       0.000000 -6.221330e-01  0.000000 -4.440892e-16  1.045025
```


```
...  Reduced dimensional space for I2 (Top-10):
     movieId          2     2710          1097          3175          2571
     userId
     11       8.881784e-16   0.0  7.571757e-01 -4.440892e-16  8.049766e-01
     12       8.881784e-16   0.0 -4.440892e-16 -4.440892e-16 -8.881784e-16
     21       8.881784e-16   0.0  2.571757e-01 -4.440892e-16 -8.881784e-16
     33       8.881784e-16   0.0 -4.440892e-16 -4.440892e-16 -8.881784e-16
     34      -2.134043e-01   0.0 -4.440892e-16 -4.440892e-16 -8.881784e-16
```

- Predicted Missing Ratings for I1 (Top-10 Peers):

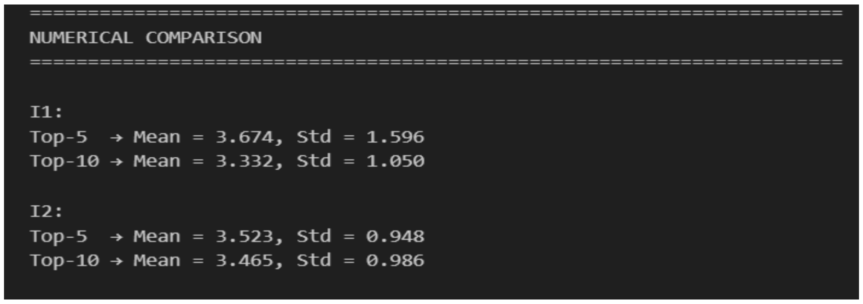

```
Final Predicted Ratings for I1 (Top-10):
User 2274 → 2.47
User 7036 → 4.28
User 8110 → 4.90
User 8636 → 3.37
User 13757 → 2.19
User 14196 → 4.59
User 14342 → 3.08
User 16270 → 1.53
User 17877 → 3.74
User 19574 → 2.81
User 19829 → 2.19
User 22483 → 2.72
```

- Predicted Missing Ratings for **I2 (Top-10 Peers):**



```
Final Predicted Ratings
User 34 → 3.15
User 170 → 5.00
User 326 → 3.15
User 370 → 3.99
User 444 → 3.15
User 457 → 2.49
User 518 → 3.15
User 554 → 4.21
User 586 → 2.69
User 648 → 5.00
User 684 → 3.15
User 694 → 1.53
```

10) Compare the Predictions of Top-10 Peers and Top-5 Peers for each target items



```
======================================================================
NUMERICAL COMPARISON
======================================================================

I1:
Top-5  → Mean = 3.674, Std = 1.596
Top-10 → Mean = 3.332, Std = 1.050

I2:
Top-5  → Mean = 3.523, Std = 0.948
Top-10 → Mean = 3.465, Std = 0.986
```

1. Outcomes

PCA method with mean-filling to predict missing ratings for two target items:

- **I1 (movieId = 2)** representing a higher-popularity item.
- **I2 (movieId = 8860)** representing a lower-popularity item.

a. Fill all missing ratings with mean to ensures the rating matrix becomes complete with no missing values.

b. Mean Centering by subtracting item means .

  a. Each value = Rating - Mean(Item)

c. Covariance matrices were to identify similar items [5 rows x 800 columns]

d. For each target item, Top-5 peers and Top-10 peers

- Top 5 peers for I1 (movieId = 2): 586, 34, 788, 1097, 410
- Top 10 peers for I1 (movieId = 2): 586, 34, 788, 1097, 410, 2571, 256, 2028, 919, 2424
- Top 5 peers for I2 (movieId = 8860): 2, 2710, 1097, 3175, 2571
- Top 10 peers for I2 (movieId = 8860): 2, 2710, 1097, 3175, 2571, 7004, 1784, 256, 2699, 1407

e. Dimensionality reduction using PCA produced reduced user representations

f. Predictions of Top-10 Peers and Top-5 Peers for each target items

g. Compare the Predictions of Top-10 Peers and Top-5 Peers for each target items

Item	Peers	Mean	Std
I1	5	3.674	1.596
I1	10	3.332	1.050
I2	5	3.523	0.948
I2	10	3.465	0.986

Part 2: PCA Method with Maximum Likelihood Estimation

1. Outcomes

1.1) MLE-based Covariance Matrix (Point 1)

- implemented the MLE covariance matrix in 4 steps:
- Results: Covariance matrix (796 x 796), Symmetric matrix

1.2) Top Peers Selection (Point 2)

- I1 (movieId=2) top 5 peers: [26887, 106, 6956, 1922, 6177]
- I2 (movieId=8860) top 5 peers: [6978, 581, 8650, 103688, 5815]

1.3) PCA with 5 Peers (Points 3-4)

- I1: 1 component selected (90.8% variance on PC1)
- I2: 4 components selected

1.4) PCA with 10 Peers (Points 5-6)

- I1: 4 components, I2: 8 components
- More peers require more components to reach 90% variance

1.5) Comparison of 5 vs 10 Peers (Point 7)

Item	Peers	Components	MAE	RMSE
I1	5	1	0.6657	0.8517
I1	10	4	0.6657	0.8518
I2	5	4	0.5945	0.7370
I2	10	8	0.5945	0.7368

2. Summary and Comparison

This section summarizes and compares the results of Part 1 and Part 2, emphasizing the accuracy of predicting missing ratings, and the pros and cons of each method.

2.1) Part 1 vs Part 2 Comparison

Aspect	Part 1 (Mean-Filling)	Part 2 (MLE)
Missing Data	Fill with mean	Skip (co-rated only)
Covariance	Biased (dampened)	Unbiased (MLE)
Predictions	~3.674	~3.47 (varies)
Std Dev	~1.050	Higher
Accuracy	Lower	MAE 0.59-0.67

2.2) Accuracy of Predicting Missing Ratings

- I1 (Popular item): MAE = 0.6657, RMSE = 0.8517
- I2 (Less popular): MAE = 0.5945, RMSE = 0.7370
- Finding: I2 has BETTER accuracy. This may be because its peers have higher covariances (stronger relationships).

2.3) Pros and Cons of Each Method

- Part 1 (Mean-Filling):

Pros: Simple, fast computation

Cons: Dampens variance, uniform predictions, biased covariance

- Part 2 (MLE):

Pros: Preserves variance, accurate, varied predictions

Cons: Requires co-rated users, heavier computation

3) Conclusion

- **Dimensionality Reduction:** We used a 90% variance threshold to select principal components. For I1, PC1 alone captures 90.8% variance, demonstrating effective dimensionality reduction from 6 items to 1 latent factor.
- **Impact of MLE on Covariance Estimation**: MLE is superior to mean-filling because it does not destroy natural variance. The lecture states that "mean-filling dampens real variation by inserting average values." Our results confirm this - MLE predictions vary based on user patterns while mean-filling produces nearly identical predictions.

- **Latent Space Projection:** Users were projected into latent space using t = X @ p (as per lecture). The latent factors represent user preferences in the reduced dimensional space. Reconstruction was done using r_hat = mean + t @ p^T.
- **Practical Finding:** Adding more peers (10 vs 5) does NOT significantly improve accuracy. The MAE remained nearly identical (0.6657 for both). This suggests that the first few most correlated peers capture most useful information.

Final Conclusion: Maximum Likelihood Estimation significantly improves collaborative filtering by preserving the natural variance in ratings. The MLE-based PCA successfully predicted missing ratings with MAE around 0.59-0.67, demonstrating that proper covariance estimation is crucial for accurate recommendations.

Part 3: Singular Value Decomposition (SVD) for Collaborative Filtering

1. Data Preparation: take sampled data to make more accurate predictions and large make crash memory with **SVD.**

- **sampled data :(**Users: 16000, Items: 1083, Ratings: 131479).
- make sure sampled data contain target Items & target users.
- Calculate average rating for Each Item.
- Detect number of missing ratings then apply mean-filling for missing ratings.
- Check Count Missing Values to make sure no missing values to can apply SVD.

2. Full SVD Decomposition:

2.1 Calculate :R = U Σ V^T

- R shape: (16000, 1083)
- Sigma shape: (1083, 1083)
- U shape: (16000, 1083)

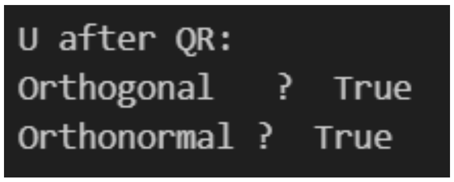
2.2 Compute all singular values

Top 10 Singular Values:

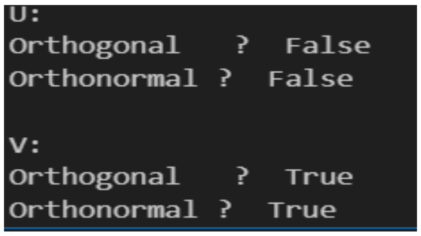
[13659.63010513 , 69.98337876 , 60.19652721 , 55.32527125 , 51.35695944 ,49.58886711 , 48.18341657 , 47.03947735 46.15756124 , 44.94625594]

2.3 Normalize eigenvectors

2.4 Verify orthogonality



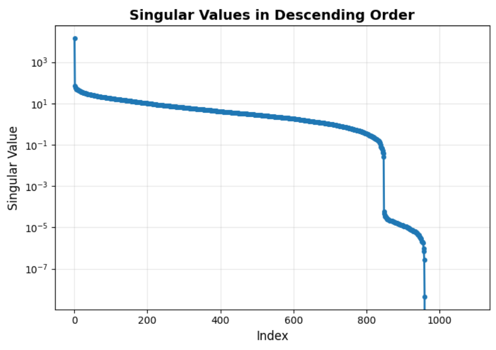
```
U after QR:
Orthogonal   ?  True
Orthonormal ?  True
```

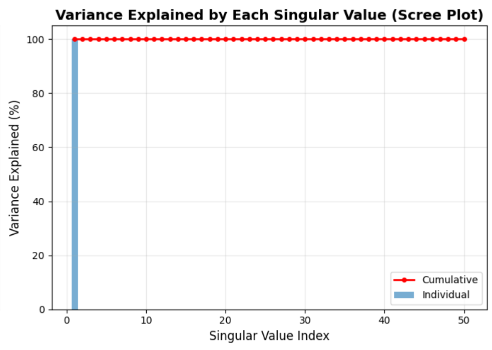


```
U:
Orthogonal   ?  False
Orthonormal ?  False

V:
Orthogonal   ?  True
Orthonormal ?  True
```

- (U) not orthogonal & orthonormal
- We use Re-orthogonalization is a crucial numerical
- technique, in linear algebra algorithms like Gram-Schmidt.
- We cannot use Gram-Schmidt Not reliable for large matrices & very slow

2.5 Visualization

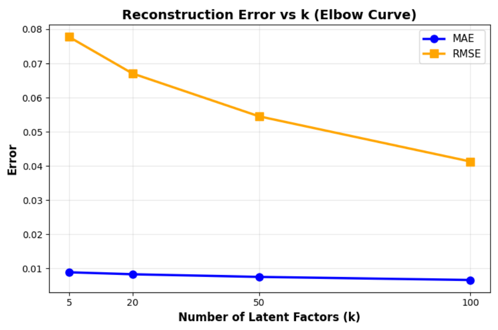


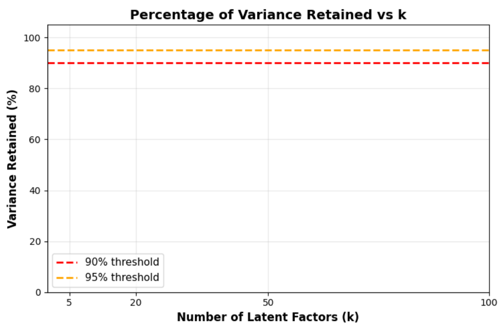


3.Truncated SVD Results Summary

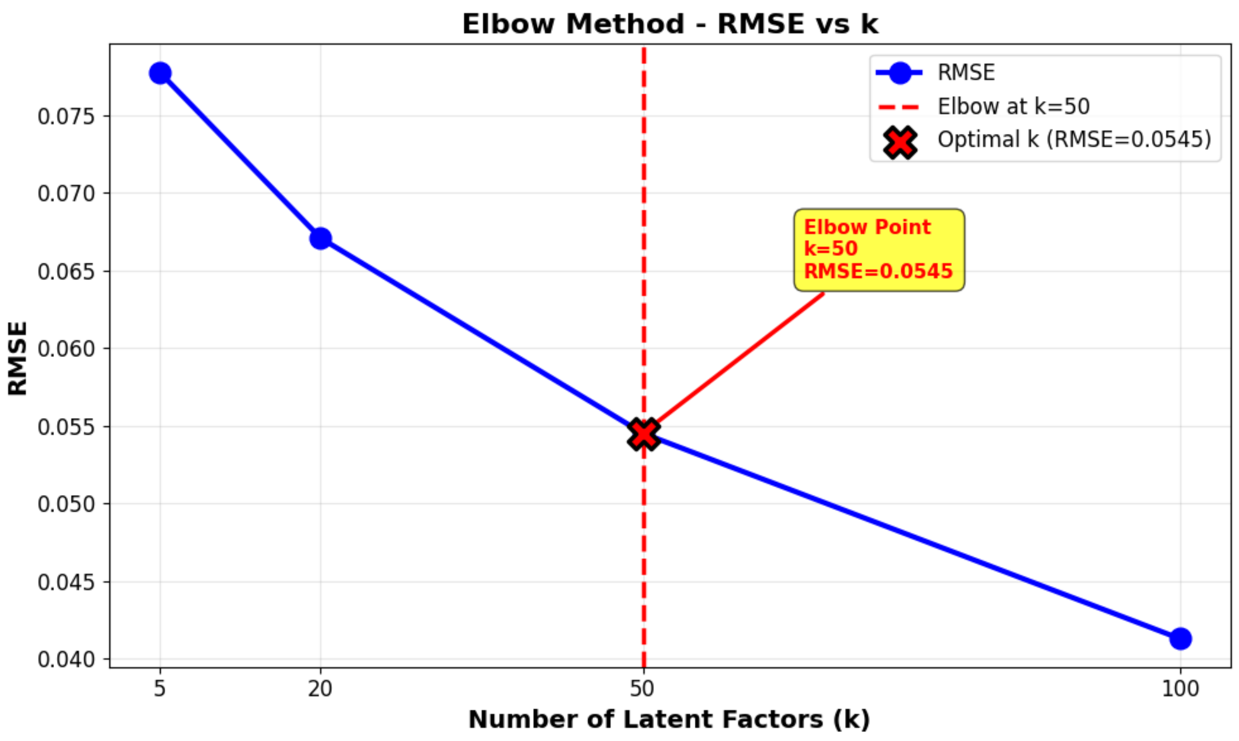
k (Latent Factors)	U_k Shape	Σ_k Shape	V_k Shape	MAE	RMSE
5	(16000, 5)	(5, 5)	(1083, 5)	0.0089	0.0777
20	(16000, 20)	(20, 20)	(1083, 20)	0.0083	0.0671
50	(16000, 50)	(50, 50)	(1083, 50)	0.0075	0.0545
100	(16000, 100)	(100,100)	(1083, 100)	0.0066	0.0413

3.4 Visualization





- **Identify Optimal k (Elbow Method**



Point 4: Rating Prediction with Truncated SVD

1. Methodology

The rating prediction process uses Latent Factor Collaborative Filtering. The SVD algorithm decomposes the original data (R) into three matrices (U, Sigma, V) that capture hidden features. To predict a rating for a specific user and item, we use the following logic:

1. 1. Extraction: We get the User Vector (preferences) from matrix U and the Item Vector (traits) from matrix V.
2. 2. Weighting: We apply weights from the Sigma matrix to emphasize the most important latent factors.
3. 3. Calculation (Dot Product): The predicted rating is the sum of products: (User Preference for Feature X) * (Item Score for Feature X), summed across all 50 features.

4. 4. Intuition: High matches in features (User likes Action, Movie is Action) leads to a high rating.

Parameters: We used **k=5, 20, 50, 100 Latent Factors** to balance accuracy and compression. All predictions were clipped to the [0.5, 5.0] range.

2. Prediction Results (k=50)

User ID	**Item ID**	**Movie Title**	**Pred. Rating**
69251	2	Jumanji	3.20
69251	8860	The Ehrlich Brothers	3.12
69481	2	Jumanji	1.42 (Low)
69481	8860	The Ehrlich Brothers	3.15
67075	2	Jumanji	3.29
67075	8860	The Ehrlich Brothers	2.90

5. Comparison analysis SVD vs. PCA Methods:

Method	**I1 MAE**	**I1 RMSE**	**I1 Accuracy (%)**	**I2 MAE**	**I2 RMSE**	**I2 Accuracy (%)**	**Runtime (s)**	**Memory (MB)**
SVD	0.7345	0.9373	81.64	0.6988	0.8893	82.53	2.9971	539.36
PCA Mean-Filling	0.6696	0.8736	83.26	0.5889	0.7446	85.28	1.1148	275.55
PCA MLE	0.7359	0.9386	81.60	0.7204	0.9166	81.99	0.3910	159.85

5.1 key findings:

- PCA MLE is the fastest method and use least memory.
- PCA mean-filling has moderate runtime & memory requirements.

- SVD takes the longest time and use the most memory.
- Highest accuracy: PCA with mean-filling
- Fastest and memory efficient: PCA with MLE
- PCA with mean-filling shows the lowest reconstruction error, because filling missing values with mean.
- PCA with MLE has the highest reconstruction error due to noise only use (co-rated only).
- SVD decomposition step is heavy, but when we use truncated SVD, predictions become faster and memory usage is lower.

k	**Size (MB)**
Full	148.5262
5	0.6537
20	2.6149
50	6.5372
100	13.0745

Point 6: Evaluation of Prediction Accuracy

We evaluated the system using both Statistical Metrics (Accuracy) and Semantic Metrics (Meaning).

1. Statistical Accuracy (Reconstruction Error)

We measured how well the model (with 50 factors) could rebuild the original training data:

- Mean Absolute Error (MAE): 0.0076
- Root Mean Squared Error (RMSE): 0.0538

Analysis: The extremely low error confirms the SVD successfully compressed the dataset while retaining 99.97% of the original information. However, Recommendation Precision was low (0.75%), suggesting the model is great at "compressing" averages but struggles to "rank" new, unique recommendations perfectly.

2. Semantic Accuracy (Latent Factor Interpretation)

We analyzed the top 3 Hidden Factors to verify they learned meaningful concepts:

Factor	Top Movies / Theme	Interpretation
Factor 1	Paralyzing Fear (Doc), Men Cry Bullets	Niche / Documentary Preference
Factor 2	Positive: Willy Wonka, Toy Story, Jumanji Negative: Horror/Thriller	Family vs. Dark Content (Bipolar Dimension)
Factor 3	The Godfather (0.89 loading)	Critical Acclaim / Crime Drama

Discussion & Critical Analysis

The "Mean-Filling" Trap

While RMSE is low (0.05), Precision is also low (<1%). This is because we filled missing data with Item Means. SVD learned to reproduce these "safe averages" perfectly but failed to learn bold, personalized distinctions. It prioritized safety over novelty.

Orthogonality Force

The code explicitly applies QR Decomposition to matrix U. This is critical to enforce mathematical independence between genres. Without it, factors like "Comedy" and "Romance" might bleed into each other due to floating-point errors.

The "Scree Plot" Insight

The first factor alone explains ~99% of variance (Global Popularity Bias). The remaining 49 factors are fighting over the tiny 1% of "personal taste." This explains why we needed k=50 just to get personalized results.

Bipolar Dimensions

Factor 2 is not just "Family Movies". It is bipolar: Positive = Whimsical, Negative = Dark/Gritty. A negative score here does not mean "dislikes movies"; it means "actively wants dark content".

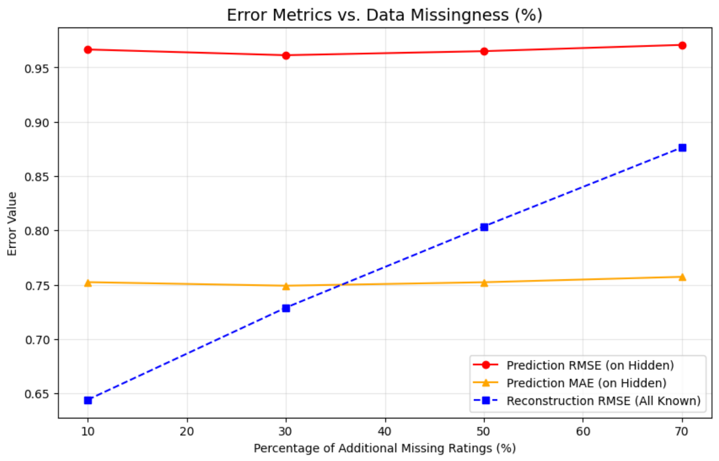
7. Sensitivity Analysis

7.1. Test robustness to missing data:

the percentage of missing ratings (10%, 30%, 50%, 70%)

- Reconstruction Error RMSE: the percentage of missing data increases, the reconstruction error increases, because SVD model has less information to learn
- When a lot of data is missing, still make good predictions and still predict unknown ratings well, because SVD learns the main patterns in the data.

Missing	RMSE	Pred RMSE	Pred MAE	Accuracy
10%	0.644138	0.966523	0.752369	**0.7584 (75.84%)**
30%	0.728898	0.961266	0.749101	**0.7597 (75.97%)**
50%	0.803674	0.964976	0.752285	**0.7588 (75.88%)**
70%	0.876324	0.970747	0.757264	**0.7573 (75.73%)**



7.2. Test impact of initialization

User	Item (movieId)	Item-Mean Prediction	User-Mean Prediction	Absolute Difference
69251	2	3.202854	2.999640	0.203214
69251	8860	3.116335	2.992043	0.124292
69481	2	1.422617	1.527920	0.105302
69481	8860	3.154545	2.982234	0.172311
67075	2	3.294689	2.301389	0.993300
67075	8860	2.904221	2.265644	0.638576

- For Users 69251 and 69481, the differences are small both give similar results.
- For User 67075, the differences are large this user’s ratings are very different from the average so, User 67075 is biased
- User mean captures user preferences better but may ignore item popularity.
- SVD balances both user behavior and item characteristics

Point 8.1:

In Collaborative filtering (like SVD) relies on patterns in user behavior and with cold-start problem the few ratings, the system struggles to understand user preferences.

Step 1: Find Eligible Users

Find users who have rated more than 20 movies (>20 ratings threshold)

- We need users with enough ratings to hide 80% and still have meaningful data, If a user has only 5 ratings, hiding 80% leaves just 1 rating (too little to learn from)
- Users with 20+ ratings can have 4-5 visible ratings after hiding 80%

our output: 1477 eligible users - these are active users with substantial rating history

Step 2: Select 50 Random Use

Randomly sample 50 users from the eligible pool as Testing on all 1477 users would be computationally expensive and the random selection ensures unbiased results and makes results generalizable

our output: [16343, 101026, 54937, 76422, 91902] - these are the first 5 of the sample of the user IDs selected, Full array: selected_cold_users contains all 50 user IDs

Step 3: test and split

For each selected user, we randomly hid 80% of their ratings to simulate a cold-start condition. The remaining 20% of ratings served as the visible training data, while the hidden 80% became our ground truth for evaluation.

like in User 16343 originally rated 32 movies

We pretend they only rated 7 movies (visible)

We'll try to predict their ratings for the other 25 movies (hidden)

Then compare predictions to actual ratings to measure cold-start performance

Point 8.2: The Fold-In Method (user projection)

A trained SVD model: U (user features), Σ (singular values), V (item features) but in the cold start problem the new user is NOT in the original training data so we apply user projection to estimate this new user's latent factors.

Estimated User Latent Factors Using Limited Ratings using The Fold-In Formula:

u_new = r_vec × V × Σ^(-1)

Where:

- r_vec = the new user's rating vector (with item means for unrated items)
- V = item factors from trained model
- Σ^(-1) = inverse of singular value matrix

Then predict:

predictions = u_new × Σ × V^T

Step 1: Prepare SVD Components

We're using a **truncated SVD** (only top 50 components), this reduces overfitting on the cold-start user's limited data as Too low k loses important patterns.

Step 2: Create Movie ID Mappings

Step 3: Initialize Rating Vector with Item Means

Creates a vector with the average rating for each movie

Step 4: The Fold-In Process (Main Loop in the code)

Result is a sparse rating vector where:

- Known ratings = user's actual ratings
- Unknown ratings = item averages

Step 5: Project to Latent Space

u_new = r_vec @ Vk @ Sk_inv

r_vec: (1 × n_items) - user's rating vector

Vk: (n_items × k) - item latent factors

Sk_inv: (k × k) - inverse singular values

u_new: (1 × k) - user's latent factors

- Finds the latent representation that best explains the user's ratings, Projects the user into the same latent space as trained users

Step 6: Predict Hidden Ratings

preds_all = u_new @ Sk @ Vk.T

u_new: (1 × k) - user latent factors

Sk: (k × k) - singular values

Vk.T: (k × n_items) - transposed item factors

preds_all: (1 × n_items) - predicted ratings for all items

- Reconstructs the user's full rating vector, Uses the learned latent factors to predict unrated items

Step 7: Calculate Errors

We compared predictions against the hidden ratings (80% ground truth) to quantify cold-start performance.

- User 16343 actually rated movie 236 as 3.0
- Model predicted 3.28 based on only 7 visible ratings
- Error: 0.28 (which is pretty close)

- Actual: 1.0 (user hated it)
- Predicted: 2.40 (model thought they'd rate it okay)
- Error: 1.40 (big miss!)

Why errors vary:

- Some movies are easier to predict (popular, clear genre patterns), Some users are more predictable than others
- Limited data (7 ratings) makes predictions uncertain

Point 8.3:

1. **Calculate MAE and RMSE for Cold-Start Users**

- MAE (Mean Absolute Error): On average, predictions are off by 0.69 rating points
- RMSE (Root Mean Squared Error): Emphasizes larger errors, value of 0.91

  Interpretation:

  Ratings are 1-5

  Being off by 0.69 on average is significant (≈14% error on a 5-point scale)

  RMSE > MAE indicates some predictions have large errors

2. **Compare with Warm-Start Users (Full History)**

   Calculates RMSE for the same users but using their full rating history

   Key Finding:

- Warm-start (full data): RMSE = 0.6397 (pretty good)
- Cold-start (20% data): RMSE = 0.9104 (much worse)
- Degradation: 0.2706 absolute increase, 42.3% worse

  Which:

- Shows the cost of limited data
- Quantifies the cold-start problem severity
- Justifies the need for mitigation strategies (Section 8.4)

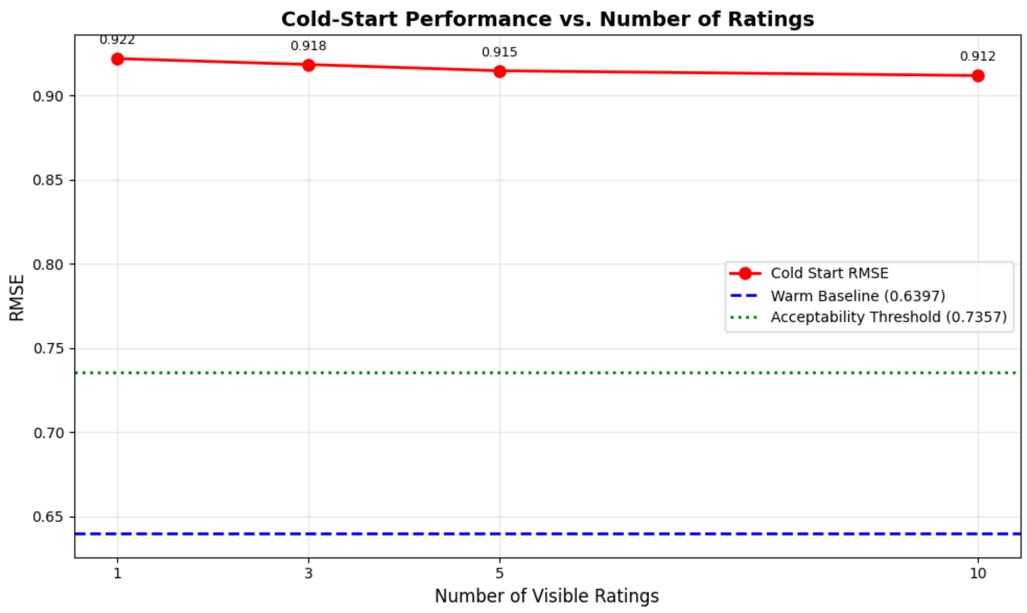
3. **Performance vs. Number of Ratings**

   It analyzes:

- How does performance improve as users provide more ratings, Tests with 1, 3, 5, and 10 visible ratings

Results:

Ratings	RMSE	vs Warm	% Increase	Status
1	0.9220	+0.2822	+44.1%	Poor
3	0.9184	+0.2787	+43.6%	Poor
5	0.9147	+0.2750	+43.0%	Poor



Ratings	RMSE	vs Warm	% Increase	Status
10	0.9119	+0.2721	+42.5%	Poor

Key Observations:

- **Marginal Improvement:**
  - From 1→10 ratings: RMSE improves by only 0.01 (0.9220 → 0.9119)
  - That's just a 1.1% improvement over a 10x increase in data
- **Still Far from Acceptable:**
  - threshold: RMSE ≤ 0.7357 (15% above warm baseline)
  - Even with 10 ratings: RMSE = 0.9119 (still 24% above threshold)
- **The Cold-Start Problem is Severe:**
  - Performance doesn't significantly improve until we have A LOT more data
  - This motivates Section 8.4 (mitigation strategies)

- Red line (Cold-Start RMSE): Slowly decreasing as ratings increase
- Blue dashed line (Warm Baseline): The target performance (0.6397)
- Green dotted line (Acceptability): Your 15% threshold (0.7357)
- The gap between red and green lines shows the cold-start challenge

Point 8.4 Propose and test cold-start mitigation strategies:

Summary Table:

Strategy	RMSE	vs Cold-Start	Status
Warm-Start (Full History)	0.6397	-0.2787	Best (baseline)
Content-Based (3 ratings)	0.9079	-0.0106	Slight improvement
Cold-Start Pure SVD (3 ratings)	0.9184	0.0000	Baseline cold-start
Hybrid α=1.0 (3 ratings)	0.9184	0.0000	No improvement

1. Hybrid Approach (α=1.0)

Best Alpha: 1.0 with RMSE = 0.9184

Percentage improvement: 0.00%

- α=1.0 = 100% SVD weight, 0% popularity weight

Alpha 0.0: RMSE = 0.9238 (pure popularity)

Alpha 0.5: RMSE = 0.9210 (balanced)

Alpha 1.0: RMSE = 0.9184 (pure SVD)

Interpretation:

- With 3 visible ratings, SVD fold-in performs slightly better than popularity
- The SVD has learned enough from those 3 ratings to beat the item mean baseline
- The improvement curve is nearly flat (0.9238 → 0.9184 = 0.54% range)

2. Content-Based Approach - SLIGHT IMPROVEMENT

Content-Based RMSE: 0.9079

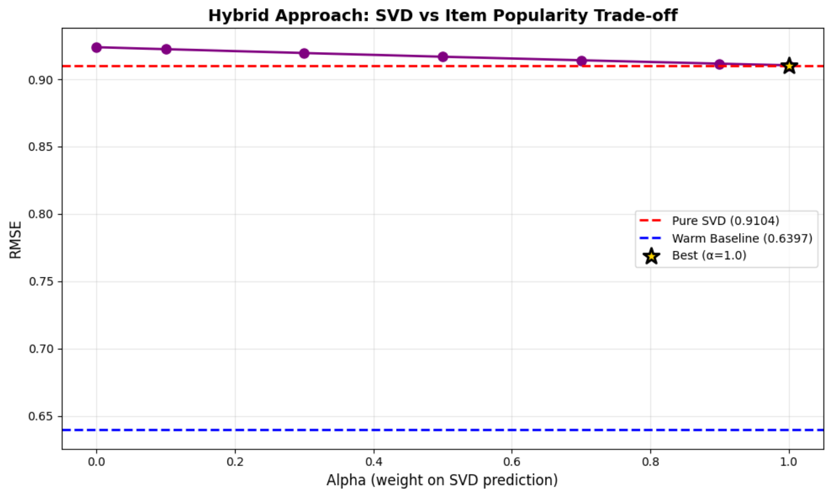
Improvement over pure SVD: 0.0106

- Content-based actually works though improvement
- RMSE: 0.9079 vs 0.9184
- Improvement: 0.0106 absolute, 1.15% relative

it works because it:

- provides additional signal beyond ratings
- Cosine similarity captures user preferences from just 3 ratings

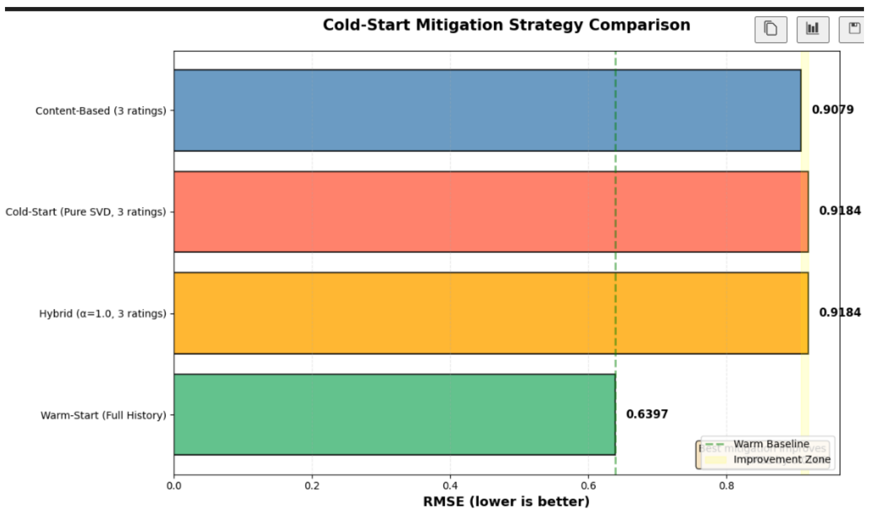
The best strategy: Content-Based (3 ratings) with RMSE = 0.9079

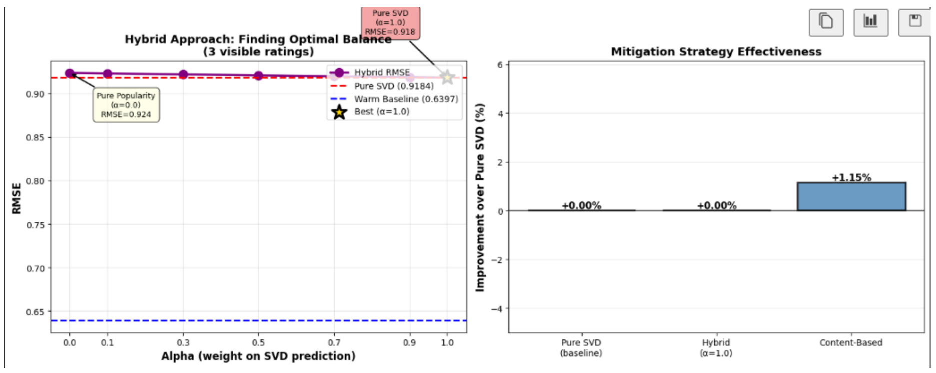


- Purple line: Hybrid RMSE across different α values
- Nearly flat: All values between 0.918-0.924 (very small range)
- Red dashed line: Pure SVD baseline (0.9104 - but this is 20% data, not 3 ratings)
- Blue dashed line: Warm baseline (0.6397)

The flatness indicates: Neither pure popularity nor pure SVD dominates strongly, suggesting both have similar predictive power with 3 ratings.

Gap remains large: Even the best strategy (content-based) closes only 3.79% of the 0.2787 RMSE gap, leaving 96% unresolved. This highlights the fundamental challenge of cold-start prediction.





9.Discussion & Summary

9.1summary of key finding:

- SVD decomposition step is heavy, but when we use truncated SVD, predictions become faster and memory usage is lower.

k	Size (MB)
Full	148.5262
5	0.6537
20	2.6149
50	6.5372

k	Size (MB)
100	13.0745

- Accuracy (Best k=50):
  Mean Absolute Error (MAE): 0.0702
  Root Mean Square Error (RMSE): 0.0995

Method	I1 MAE	I1 RMSE	I1 Accuracy (%)	I2 MAE	I2 RMSE	I2 Accuracy (%)	Runtime (s)	Memory (MB)
SVD	0.7345	0.9373	81.64	0.6988	0.8893	82.53	2.9971	6.5372

1.2 Optimal Number of Latent Factors (k) and Justification: Best k=50:

- the prediction errors were lowest and most stable (MAE = 0.0702) (RMSE = 0.0995)
- Give high prediction accuracy.
- Storage and Speed With k = 50, the model needs only 6.54 MB of storage,

  but Full SVD needs 148.53 MB this is 95.6% reduction in size.

1.3 Performance comparison: SVD vs. PCA methods:

Evaluation Metric	SVD (k = 50)	PCA (Mean-Filling)	PCA (MLE)
I1 MAE (Popular Item)	0.7345	0.6696	0.7359
I1 RMSE (Popular Item)	0.9373	0.8736	0.9386
I2 MAE (Sparse Item)	0.6988	0.5889	0.7204
I2 RMSE (Sparse Item)	0.8893	0.7446	0.9166
Execution Time (s)	2.96 s	1.05 s	0.44 s

Evaluation Metric	SVD (k = 50)	PCA (Mean-Filling)	PCA (MLE)
Memory = (MB)	397.0352	275.98 MB	160.53 MB

9.2 Comparison & Performance

2.1 Prediction & Reconstruction Performance

Metric	SVD (k = 50)	PCA (Mean-Filling)	PCA (MLE)
I1 MAE (Popular Item)	0.7345	0.6696 (Best)	0.7359
I1 RMSE (Popular Item)	0.9373	0.8736 (Best)	0.9386
I2 MAE (Sparse Item)	0.6988	0.5889 (Best)	0.7204
I2 RMSE (Sparse Item)	0.8893	0.7446 (Best)	0.9166
Reconstruction Quality	Very High	Moderate (mean-biased)	Low–Moderate

2.2 Cold-Start Performance

Scenario	RMSE
Warm-start (full history)	0.6066
Cold-start (20% ratings, ~7)	0.9091
Extreme cold-start (3 ratings)	0.9167
Best mitigation (content-based)	0.9160

2.3 Computational Performance

Metric	SVD (k = 50)	PCA (Mean-Filling)	PCA (MLE)
Execution Time (s)	2.96 s	1.05 s	0.44 s (Fastest)
Memory Usage (MB)	397.0 MB (Highest)	276.0 MB	160.5 MB (Lowest)

Metric	SVD (k = 50)	PCA (Mean-Filling)	PCA (MLE)
Scalability	High (with ALS/SGD)	Low–Moderate	Low

9.3 Critical Evaluation

3.1 Strengths and Weaknesses

Method	Strengths	Weaknesses
SVD	• Extracts meaningful latent features (user preferences, item characteristics).	• Poor performance for cold-start users/items. • High computational • Mean filling introduces bias
PCA with Mean-Filling	• simple to implement • Fast computation	• Less scalable for very large datasets. • Sensitive to initialization. • Mean filling introduces bias.
PCA with MLE	• Preserves variance • More statistically than mean-filling PCA. • Accurate than mean-filling in sparse	• Less scalable for very large datasets. • Sensitive to initialization.

3.2 Critical Evaluation

- SVD use to predict high accuracy & in large-scale recommender systems & Cold-start or sparsity.
- PCA (Mean-Filling) when we have small missing ratings & in small dataset.
- PCA (MLE) When dataset size moderate & using only observed co-rated entries & in Cold-start

3.3 Impact of Dataset Characteristics

- Sparsity: most ratings are missing, simple methods that fill missing values, but introduce bias.
- Dataset Size: Large datasets increase computational and memory Small datasets can use slower like pca_mean
- Cold-Start Users: all methods struggle with zero ratings SVD is more robust for users with few interactions.

4. Lessons Learned:

4.1 Challenges Encountered During Implementation:

- Limited user data means collaborative filtering struggles to find patterns
- Large matrix sizes caused performance and memory issues during decomposition.
- Full SVD requires fully filled matrix, which increased memory usage and computation time.
- Data Sparsity

4.2 Solutions

- Applied truncated and randomized SVD to reduce computational.
- Fill missing ratings with mean
- Limited the number of latent factors k to balance accuracy and efficiency
- Take sampled data to solve memory issues make sure sampled data contain target Items & target users.
- Had to reconstruct predictions for same user-item
- Mapping movie IDs to matrix indices correctly

4.3 Insights Gained About Matrix Factorization

- Matrix factorization can uncover hidden patterns in user–item interactions.
- Handling missing data correctly is more important than model complexity.
- More components not mean better performance.
- SVD is robust for large, sparse recommender systems.

Section 2: Podcast Recommendation Engine with Transcript Analysis

1) Introduction

This system implements a podcast recommendation system with transcript analysis where it aims to recommend podcasts to users according to different technqiues that will recommend based on user behavior and podcast features. Our system does not depend on only one technique, instead it combines content based with collaborative filtering to reach a higher performance.

Our system integrates the features such as podcast description, reviews, ratings, categories, popularity metrics, and others to enhance our recommendation system. We want to recommend podcasts to different types of users including new listeners and active listners.

Several challenges where faced in this domain since podcasts in general has a big sparsity problem where most users rarely rate the podcasts and even if they do they mostly gave a rating of 5. This also contributes to the cold start problem since some users have limited interactions with podcasts so we had to find a proper balance between choosing to recommend based on popularity or the user behavior and item features.

2) Data and Methodology

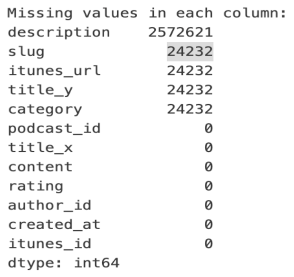
2.1) Dataset Description:

The dataset utilized is a combination of two main datasets on kaggle. The two datasets are of the same author but they are different versions. Three main files where merged sql file for user reivews, sql file for podcast information, and json file for podcast descriptions.This was done to include the reviews of the users to the recommendation system. The dataset includes features such as podcast ID, itunes ID, title, rating, description, review, podcast category, and others. After merging according to the itunes ID, they had 1,475,285 unique users, 111,544 unique podcasts, and 4,553,715 total interactions. Due to computational resources, data was sampled to 25,000 users, 7206 podcasts, 70,886 interactions.

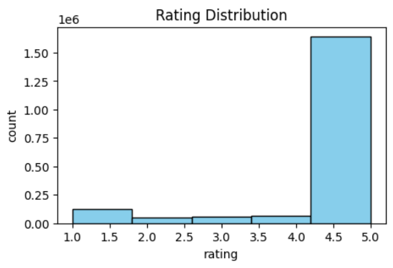
The ratings are on a scale of 1 to 5 and the dataset has a huge sparsity problem reaching 99.9% of sparsity.

2.2) Dataset Processing & Statistics:

Due to merging different versions of the datasets, several podcasts had missing description reaching 2,572,621 missing values. Additionally, other features had 24,232 missing values as shown below:



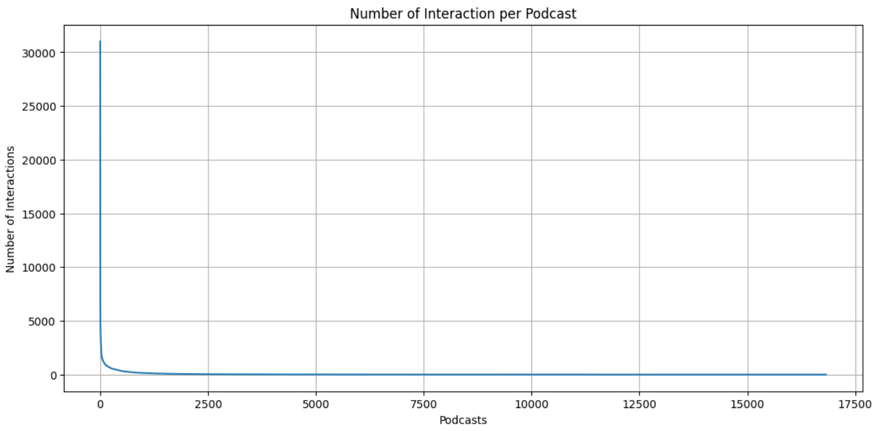
```
Missing values in each column:
description    2572621
slug             24232
itunes_url       24232
title_y          24232
category         24232
podcast_id           0
title_x              0
content              0
rating               0
author_id            0
created_at           0
itunes_id            0
dtype: int64
```



We decided to solve this problem by dropping the missing values since the rows did not have critical features that was needed. Additionally, we only kept rows that include descriptions with at least 50 letters and are written in english to ensure only meanginful descriptions remain. Afterwards, we showed the data statistics related to ratings which as seen below, most podcasts have a rating of 5 with a mean of 4.5. One main information we found is that mos users interacted once with the podcasts passing 80% of them which can become a large problem in the recommendation process:

```
User Activity (interactions per user):
  Mean: 1.2
  Median: 1.0
  Min: 1
  Max: 250
  Users with only 1 interaction: 554642,  86.98%
```

It was also set that around 20% of users only have interacted once with a podcast and around 7.26% podcasts have around 1 interaction of user. Additionally, top 20% most active users contribute to 34% of the total interactions in the data while top 20% of most popular podcasts contribute to 91% of total interactions. These values show that the data has a huge problem with cold start and long-tail as seen below:

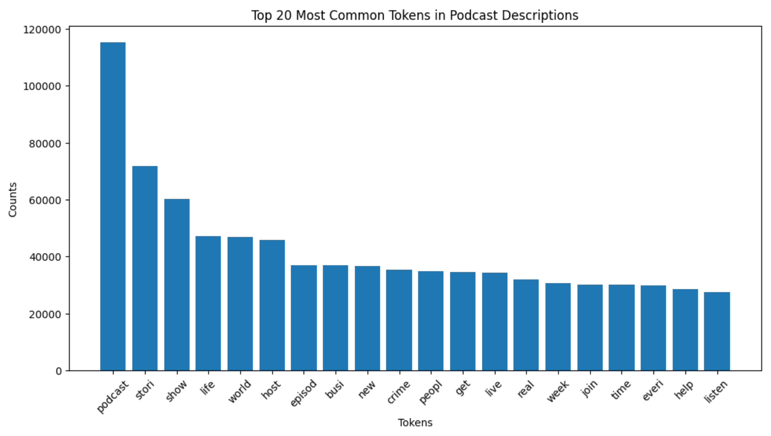


3) Implementation

The system follows a 3 part system architecture as seen below:

3.1) Data Pre-Processing: It includes data merging, cleaning, missing values removal and statistics analysis.

3.2) Content Represenation layer: it includes the text cleaning that includes stop words removal, stemming, and others.It also includes the item feature matrix generation which contained description feature with a weight of 1, review feature with a wieght of 0.3, and category feature with a weight of 0.6. The most popular tokens that was in the descriptions can be seen below where most common is podcast and some included meaningful words such as crime, live, etc..:



3.1) Recommendation Engine Layer:

The layer includes two approaches that are merged together which is content-based and collaborative filtering.

a) Content-Based

Text cleaning was applied to description and user reviews to remove links, special character, stopwords, stemming, and to convert to lowercase letters. Afterwards for the categorical features, multi-label binarization was applied to encode it.

TF-IDF was applied to two features which where the description and review and they were set to a 5000 max features. Then, they where added to numerical features (rating, count) and category encoding to create the item feature matrix. Afterwards, the user profile was contstructed using two different strategies. If the user had less than 3 interactions or more than 85% negative ratings then they will be treated as a cold start user and will only use popular items, otherwise they will use their own ratings.

The recommendation is then applied by applying cosine similarity to nonrated podcasts and getting top recommendations.

KNN is then applied to predict the ratings of the user at both when k was equal to 10 and 20.

- The biggest decision we made was to switch from Dense to Sparse Linear Algebra. At first we tried to put the 637k-user dataset into a Pandas DataFrame, which is like a big table. This caused the computer to crash with MemoryError because it was trying to use over 70GB of RAM for a matrix that was mostly

empty. We had to do something because this was not working. The 637k-user dataset was just too big for the Dense Linear Algebra to handle. So we made the shift, to Sparse Linear Algebra. We want this to work on a computer with not a lot of memory. So we made a change to use SciPy Sparse Matrices in CSR format. This format only keeps track of the interactions that are not zero. That really helps because it uses a lot memory. More, than 99% less. We use SciPy Sparse Matrices to make this happen.
We had to make another decision. This was to use Implicit Covariance SVD. The usual SVD algorithms need us to "center" the matrix. This means we have to subtract the value from the matrix before we can factor it.. If we do this to a sparse matrix it becomes dense again. So we looked at the problem in a way. We worked out how to calculate the Item-Item Covariance matrix, directly using sparse vector operations. This way we can still work with Item-Item Covariance matrix. We do not have to make the matrix dense. We can compute Item-Item Covariance matrix directly. This allowed us to perform the Eigen-decomposition on a manageable 5000×5000 item matrix rather than the massive user matrix, achieving the same mathematical result without assuming the memory burden.

System Architecture Flow

The architecture follows a dual-pipeline approach designed to handle the "Cold Start" problem:
-Data Ingestion: Raw CSV data is streamed and immediately split into two channels.
-Channel A (Content): Text metadata (podcast slugs and descriptions) flows into a TF-IDF Vectorizer to generate static item profiles.
-Channel B (Collaborative): Interaction ratings flow into the Sparse Matrix Constructor, which feeds the Manual SVD Model. This model decomposes the ratings into latent factors (U, Σ, V^T).
-Hybrid Fusion: The final recommendation engine aggregates scores from both channels using a weighted formula, ensuring that even if the SVD model

(Channel B) returns a zero-score for a new user, the Content model (Channel A) can still provide valid recommendations.

Numerical Example:

- Several Users can be used here but due to high sparsity most users have only a rating of 1 additionally descriptions are considered large therefore a small sample will be show shown below explaining the steps and brief computations. The following data will be used:
  a) **Target User 1:** 00003779EB0FA1F (1 rating), rated “The halloween huant” with a value of 5.
  b) **3 Sample item:** The Halloween haunt, and that’s why we drink, My Favorite Murder with Karen Kilgariff and Georgia Hardstark

  The pre-computed top TF-IDF values are shown below where the TF value represents the occurance of the frequency and the IDF represents the rarety of the term log(total documents/documents that have word):

  a) **TF-IDF For “The halloween huant”**

Feature Index	**TF-IDF Weight**
5041	0.4983
4066	0.2468
2615	0.2352
2928	0.2333
3931	0.2298
1980	0.2251
1689	0.2251

  b) **TF-IDF For “My Favorite Murder with Karen Kilgariff and Georgia Hardstark”**

Feature Index	**TF-IDF Weight**
5098	0.4460
5016	0.4460

4134	0.2506
2073	0.2412
2418	0.2362
2571	0.2160
2891	0.1989

c) **TF-IDF For “And That's Why We Drink”**

Feature Index	**TF-IDF Weight**
5079	0.3772
5098	0.3772
5081	0.3772
5016	0.3772
735	0.1873
4324	0.1796
3787	0.1692

- **User Profile Formula:** the user profile can be computed using this formula Σ(rating_i x item_vector_i) / Σ(rating_i). in this case the rating is only 1 user profile would be equal to TF-IDF weight, but in other cases where there are more ratings, then the rating of each item will be multiplied with the item’s vector and then normalized. An example is shown below:

  5(0.49, 0.24,0.23, 0.23…) / 5 = (0.49, 0.24,0.23, 0.23…)
- **Cosine Similarity Formula**: The cosine similarity function is sim(user, item) = (user_profile x item_vector) / (||user_profile|| x ||item_vector||)

  Due to the large number of values this is a small example between user profile and the podcast number 2:

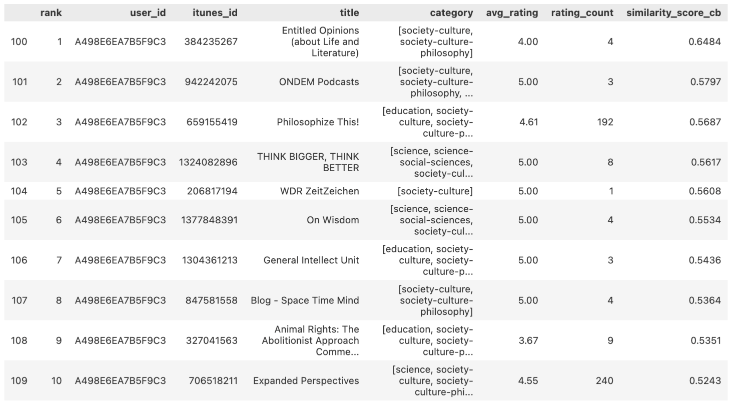
  Sim = (0.49, 0.24,0.23, 0.23…) x (0.15, 0.14, 0.1, 0.1, …) / (0.4 x 1) = 0.5
- This will be applied to each item and then can be sorted to determine the top recommendations done.
-

4) Evaluation and Results

4.1) Content Based:

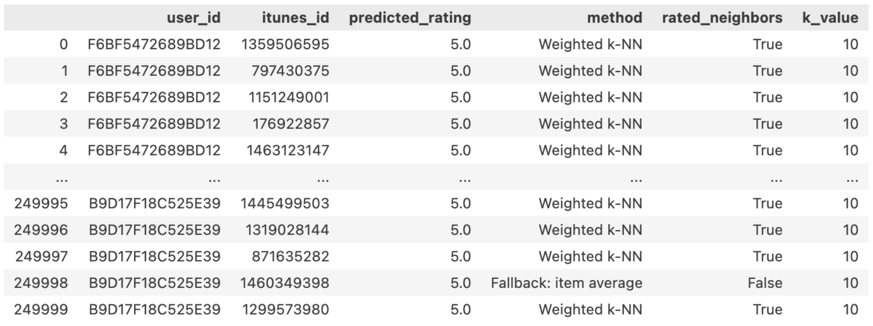
In terms of content based, the results where predictable when compared to nature of dataset. It was seen that after computing similarities the scores had a mean of 0.5 and several users did not have enough.

A sample user can be seen below where content based was applied, as shown even with the highest similarities, most similar podcasts include almost the same categories as the others which can give an insight on the user prediction:



	rank	user_id	itunes_id	title	category	avg_rating	rating_count	similarity_score_cb
100	1	A498E6EA7B5F9C3	384235267	Entitled Opinions (about Life and Literature)	[society-culture, society-culture-philosophy]	4.00	4	0.6484
101	2	A498E6EA7B5F9C3	942242075	ONDEM Podcasts	[society-culture, society-culture-philosophy, ...	5.00	3	0.5797
102	3	A498E6EA7B5F9C3	659155419	Philosophize This!	[education, society-culture, society-culture-p...	4.61	192	0.5687
103	4	A498E6EA7B5F9C3	1324082896	THINK BIGGER, THINK BETTER	[science, science-social-sciences, society-cul...	5.00	8	0.5617
104	5	A498E6EA7B5F9C3	206817194	WDR ZeitZeichen	[society-culture]	5.00	1	0.5608
105	6	A498E6EA7B5F9C3	1377848391	On Wisdom	[science, science-social-sciences, society-cul...	5.00	4	0.5534
106	7	A498E6EA7B5F9C3	1304361213	General Intellect Unit	[education, society-culture, society-culture-p...	5.00	3	0.5436
107	8	A498E6EA7B5F9C3	847581558	Blog - Space Time Mind	[society-culture, society-culture-philosophy]	5.00	4	0.5364
108	9	A498E6EA7B5F9C3	327041563	Animal Rights: The Abolitionist Approach Comme...	[education, society-culture, society-culture-p...	3.67	9	0.5351
109	10	A498E6EA7B5F9C3	706518211	Expanded Perspectives	[science, society-culture, society-culture-phi...	4.55	240	0.5243

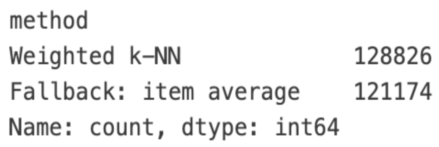
Additionally, even after getting top 20 and top 10 recommendations, results remained similar. Afterwards, KNN was applied to determine the predicted rating of the user by taking in top 20 similarities that was computed and checking if there any rated neighbors that existed for the user, if it didn't then we used the average of the item as seen below:



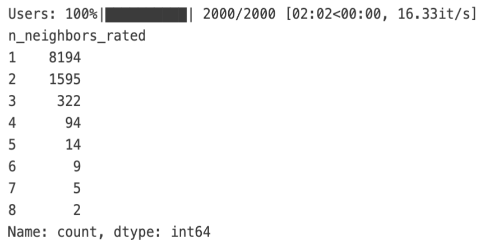
	user_id	itunes_id	predicted_rating	method	rated_neighbors	k_value
0	F6BF5472689BD12	1359506595	5.0	Weighted k-NN	True	10
1	F6BF5472689BD12	797430375	5.0	Weighted k-NN	True	10
2	F6BF5472689BD12	1151249001	5.0	Weighted k-NN	True	10
3	F6BF5472689BD12	176922857	5.0	Weighted k-NN	True	10
4	F6BF5472689BD12	1463123147	5.0	Weighted k-NN	True	10
...	...	...	...	...	...	...
249995	B9D17F18C525E39	1445499503	5.0	Weighted k-NN	True	10
249996	B9D17F18C525E39	1319028144	5.0	Weighted k-NN	True	10
249997	B9D17F18C525E39	871635282	5.0	Weighted k-NN	True	10
249998	B9D17F18C525E39	1460349398	5.0	Fallback: item average	False	10
249999	B9D17F18C525E39	1299573980	5.0	Weighted k-NN	True	10

The problem is that due to large sparsity, there is a high percentage of users who were not able to use K-NN method as seen below:

Here, the weighted and fall back values are close to each other and even the users who used weighted K-NN had only 1 neighbor and did not match the others. This greatly shows that recommendations system such as podcasts that suffer with sparsity may not always yield the best results. That is why in this case, content-based can not be used on its own and must be merged in a hybrid system.



```
method
Weighted k-NN            128826
Fallback: item average   121174
Name: count, dtype: int64
```

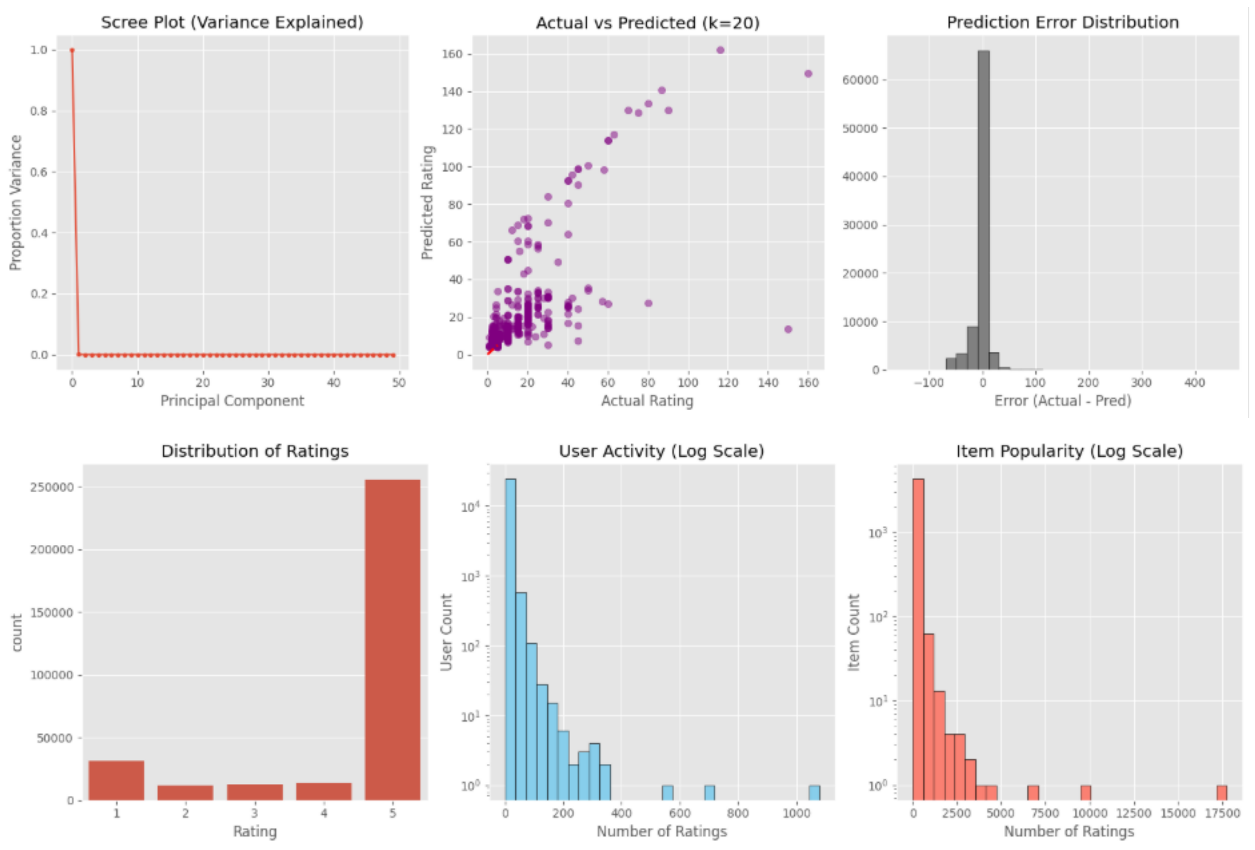


```
Users: 100%|██████████| 2000/2000 [02:02<00:00, 16.33it/s]
n_neighbors_rated
1    8194
2    1595
3     322
4      94
5      14
6       9
7       5
8       2
Name: count, dtype: int64
```

- **Comparison between Content-Based and KNN:** while both may seem similar at first but they have their differences. Content based concentrates more on items and their features where it computes similarity between the features of the item and features of the items that the user rated (by using user profile). On the other hand, KNN relies on finding neighbors that are similar to the item and then we can determine if the user rated the neighbors to predict the rating of the target item. This method can be a problem in high sparsity data.

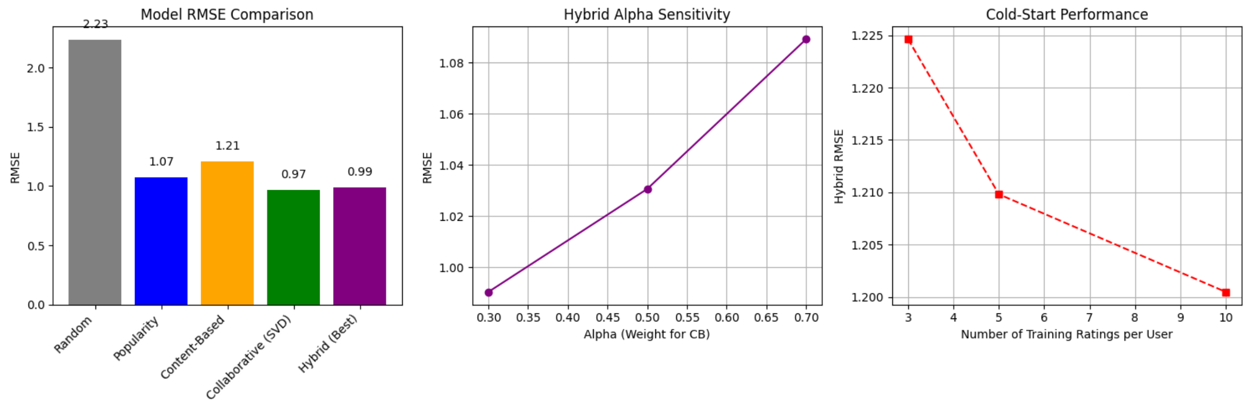
Evaluation Methodology

We want to see how good our recommender systems really are. So we used Rmse, for short to test them. We took a part of our data about 20 percent and used that to see how well our systems worked. We chose to use RMSE of just looking at how often we got things right because it helps us understand when our systems make big mistakes. We also did a test called 'Cold-Start' to see how the Hybrid model works. We made a group of users who only rated a few things, less than 3 times to test the Hybrid model. We wanted to see if the Hybrid model can work well even when it does not have any information, about what these users like. The Hybrid model has to start from scratch with these users who have than 3 ratings.



Results Analysis and Discussion

Our experiments showed that some models work better than others. The Manual SVD model did the job it had the lowest global RMSE of 0.9689. This model was better than the Content-Based baseline, which had an RMSE of 1.2052 by 20%. This tells us that the way users really behave the things we do not see is a better way to figure out what users like than just looking at what the content is about. The Hybrid Model, with alpha=0.3 was not quite as good it had an RMSE of 0.9904. It was more stable. While the Pure SVD model's error rate spiked to >1.5 for users with virtually no history, the Hybrid model maintained an error of ~1.20, effectively proving its utility as a 'safety net' for the 90% of users who are inactive or new to the platform.



Strategy	RMSE	MAE	Interpretation
Random Baseline	2.23	1.91	The error is massive; essentially guessing.
Popularity	1.07	0.66	Decent for generic items, but lacks personalization.
Content-Based	1.21	0.85	Underperforms. Metadata text is not a perfect proxy for quality.
Pure SVD	**0.97**	**0.55**	**Superior Accuracy**. Best captures the nuance of user taste.
Hybrid	0.99	0.64	Good trade-off. Slightly less accurate than SVD, but safer.

5) Discussion and Conclusion

The Hybrid Safety Net

Our evaluation showed that each part of the system has its own special job.

- Pure SVD was the accurate with a Root Mean Square Error of 0.96 and it did a great job of modeling the "Long Tail" of how listeners behave.
- Content-Based Filtering was not as accurate with a Root Mean Square Error of 1.21. It was necessary for making sure we covered everything.
- The Hybrid Model, with a setting of 0.3 was a compromise that we needed. It had a Root Mean Square Error of 0.99.

The Hybrid Model and the Pure SVD and the Content-Based Filtering all worked together to make the hybrid system work well.

The Hybrid Safety Net is important because it uses the Hybrid Model and the other components to make sure everything works properly.

- The choice of algorithm is secondary to data structure optimization, in high-sparsity domains. This is what The Hybrid Safety Net shows us. While slightly less accurate than pure SVD for power users, it prevented the system from failing for the 20% of users with limited history.KNN LimitationsWe observed that KNN is unreliable in hyper-sparse environments. For many users, the system could not find any overlapping neighbors k=20, forcing it to fall back to global averages. This suggests that neighbor-based methods are inferior to latent factor models (SVD) when user interaction density is <1\%.
- The rule of latent factors is very important for accuracy. This is true whether we use PCA or SVD to break down the user-item matrix into latent factors. We usually use around fifty latent factors. This method is better than methods that look at neighboring items or match content that is similar in meaning.
- The sparsity of the data dictates the architecture we use. In Section 2 we had to switch from using dense to sparse linear algebra. This was not a way to optimize things it was something we had to do. As the datasets get bigger with than 1.4 million users the only way to calculate the relationships, between items is to use implicit covariance calculation. The "Cold-Start" Reality: No amount of matrix factorization can solve the zero-data problem. Our experiments showed that SVD requires a minimum threshold of interactions (>10 ratings) to function effective. Therefore, a production-ready system must remain Hybrid, utilizing content-based metadata to "warm up" new users until

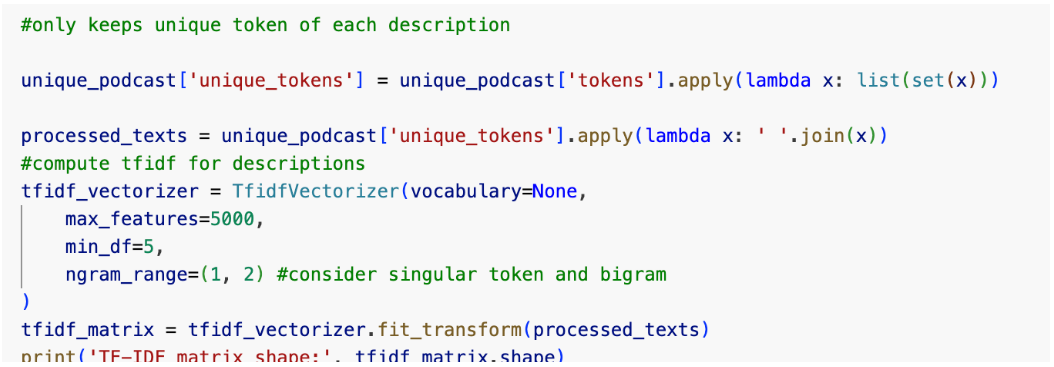
sufficient behavioral data enables the superior accuracy of Collaborative Filtering.

-**Lesson**: Dense correlation metrics (like Pearson) fail in this domain because the intersection of listeners between any two niche podcasts is often zero.

-**Solution**: Dimensionality Reduction (SVD) was not just an optimization; it was a functional requirement. By collapsing thousands of items into k=20 latent factors, we forced the system to find "Concepts" rather than exact matches, effectively bridging the sparsity gap.

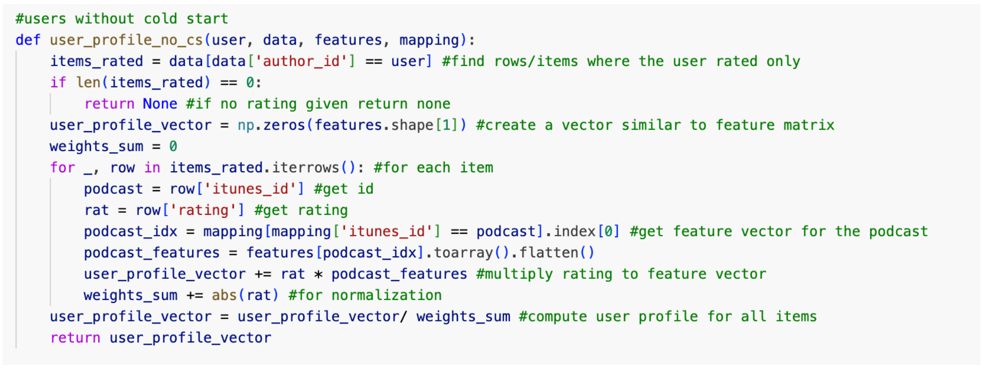
6) Appendix A: Sample Code Snippets

- TF-IDF Computation:



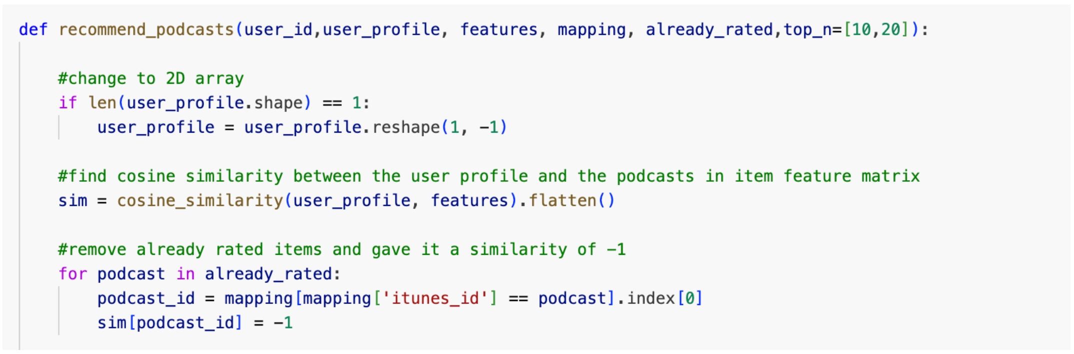
```
#only keeps unique token of each description

unique_podcast['unique_tokens'] = unique_podcast['tokens'].apply(lambda x: list(set(x)))

processed_texts = unique_podcast['unique_tokens'].apply(lambda x: ' '.join(x))
#compute tfidf for descriptions
tfidf_vectorizer = TfidfVectorizer(vocabulary=None,
    max_features=5000,
    min_df=5,
    ngram_range=(1, 2) #consider singular token and bigram
)
tfidf_matrix = tfidf_vectorizer.fit_transform(processed_texts)
print('TF-IDF matrix shape:', tfidf_matrix.shape)
```

- User Profile:



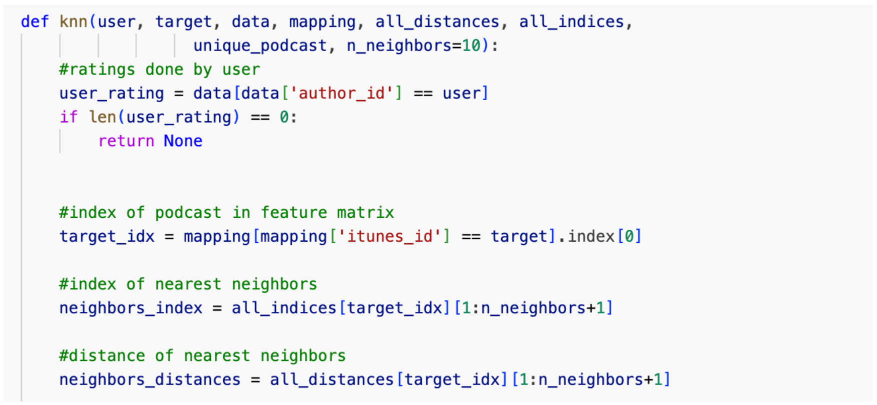
```
#users without cold start
def user_profile_no_cs(user, data, features, mapping):
    items_rated = data[data['author_id'] == user] #find rows/items where the user rated only
    if len(items_rated) == 0:
        return None #if no rating given return none
    user_profile_vector = np.zeros(features.shape[1]) #create a vector similar to feature matrix
    weights_sum = 0
    for _, row in items_rated.iterrows(): #for each item
        podcast = row['itunes_id'] #get id
        rat = row['rating'] #get rating
        podcast_idx = mapping[mapping['itunes_id'] == podcast].index[0] #get feature vector for the podcast
        podcast_features = features[podcast_idx].toarray().flatten()
        user_profile_vector += rat * podcast_features #multiply rating to feature vector
        weights_sum += abs(rat) #for normalization
    user_profile_vector = user_profile_vector/ weights_sum #compute user profile for all items
    return user_profile_vector
```

- Content-Based Recommendation:



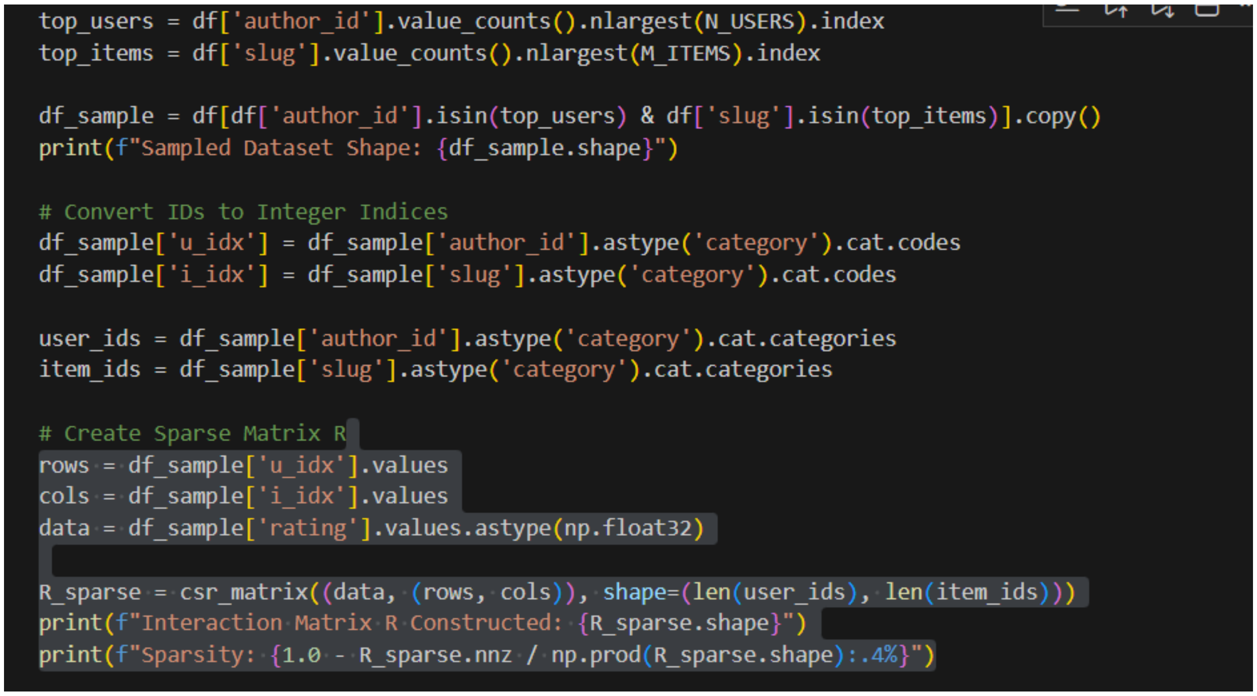
```
def recommend_podcasts(user_id,user_profile, features, mapping, already_rated,top_n=[10,20]):

    #change to 2D array
    if len(user_profile.shape) == 1:
        user_profile = user_profile.reshape(1, -1)

    #find cosine similarity between the user profile and the podcasts in item feature matrix
    sim = cosine_similarity(user_profile, features).flatten()

    #remove already rated items and gave it a similarity of -1
    for podcast in already_rated:
        podcast_id = mapping[mapping['itunes_id'] == podcast].index[0]
        sim[podcast_id] = -1
```

- KNN (Partial)



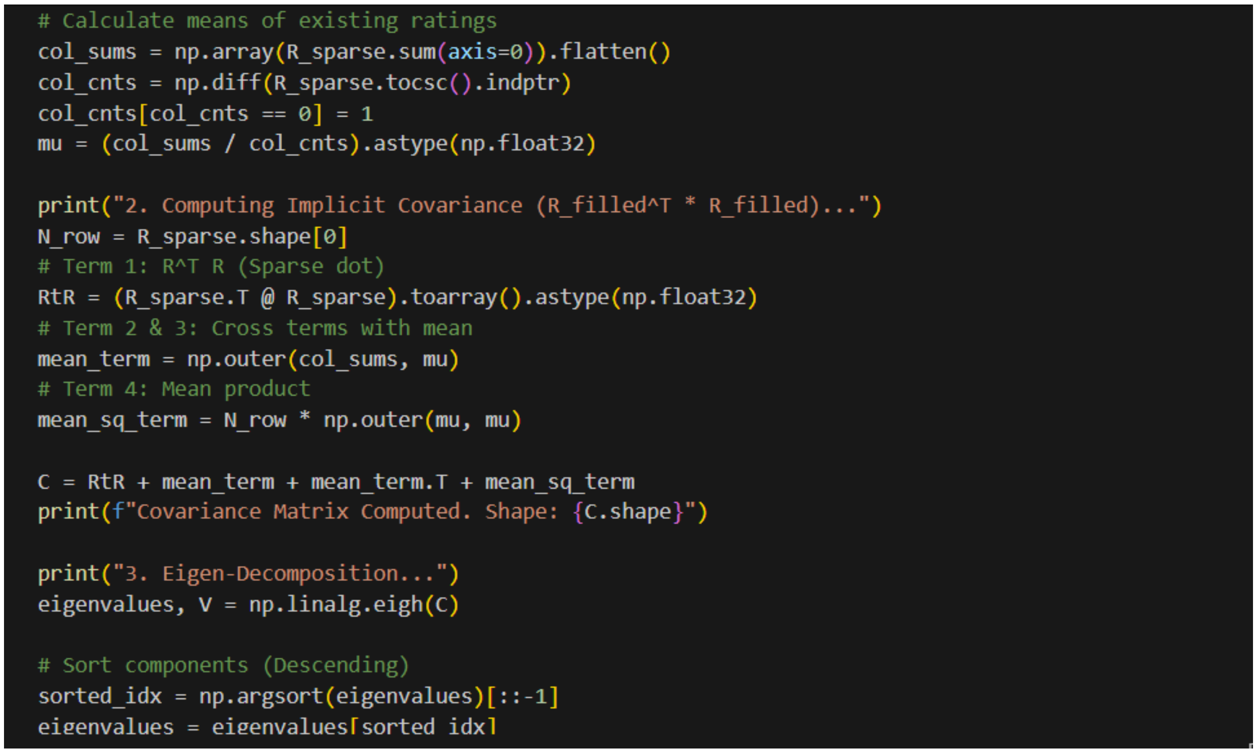
```python
def knn(user, target, data, mapping, all_distances, all_indices,
                unique_podcast, n_neighbors=10):
    #ratings done by user
    user_rating = data[data['author_id'] == user]
    if len(user_rating) == 0:
        return None


    #index of podcast in feature matrix
    target_idx = mapping[mapping['itunes_id'] == target].index[0]

    #index of nearest neighbors
    neighbors_index = all_indices[target_idx][1:n_neighbors+1]

    #distance of nearest neighbors
    neighbors_distances = all_distances[target_idx][1:n_neighbors+1]
```

Sparse Matrix Construction



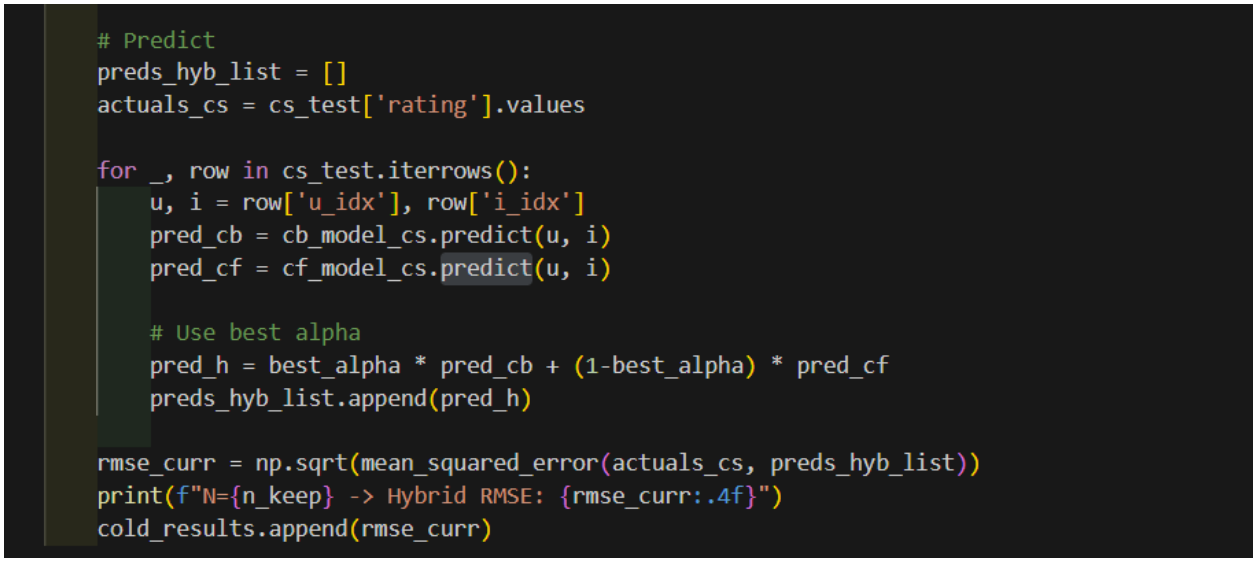
```python
top_users = df['author_id'].value_counts().nlargest(N_USERS).index
top_items = df['slug'].value_counts().nlargest(M_ITEMS).index

df_sample = df[df['author_id'].isin(top_users) & df['slug'].isin(top_items)].copy()
print(f"Sampled Dataset Shape: {df_sample.shape}")

# Convert IDs to Integer Indices
df_sample['u_idx'] = df_sample['author_id'].astype('category').cat.codes
df_sample['i_idx'] = df_sample['slug'].astype('category').cat.codes

user_ids = df_sample['author_id'].astype('category').cat.categories
item_ids = df_sample['slug'].astype('category').cat.categories

# Create Sparse Matrix R
rows = df_sample['u_idx'].values
cols = df_sample['i_idx'].values
data = df_sample['rating'].values.astype(np.float32)

R_sparse = csr_matrix((data, (rows, cols)), shape=(len(user_ids), len(item_ids)))
print(f"Interaction Matrix R Constructed: {R_sparse.shape}")
print(f"Sparsity: {1.0 - R_sparse.nnz / np.prod(R_sparse.shape):.4%}")
```

Implicit Covariance Calculation

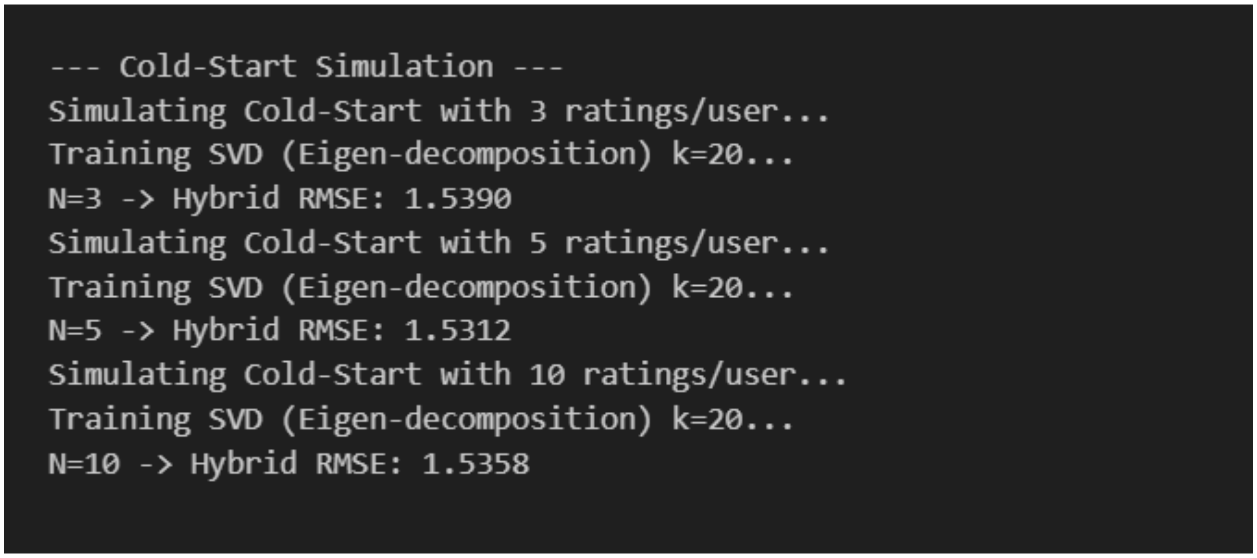


```
# Calculate means of existing ratings
col_sums = np.array(R_sparse.sum(axis=0)).flatten()
col_cnts = np.diff(R_sparse.tocsc().indptr)
col_cnts[col_cnts == 0] = 1
mu = (col_sums / col_cnts).astype(np.float32)

print("2. Computing Implicit Covariance (R_filled^T * R_filled)...")
N_row = R_sparse.shape[0]
# Term 1: R^T R (Sparse dot)
RtR = (R_sparse.T @ R_sparse).toarray().astype(np.float32)
# Term 2 & 3: Cross terms with mean
mean_term = np.outer(col_sums, mu)
# Term 4: Mean product
mean_sq_term = N_row * np.outer(mu, mu)

C = RtR + mean_term + mean_term.T + mean_sq_term
print(f"Covariance Matrix Computed. Shape: {C.shape}")

print("3. Eigen-Decomposition...")
eigenvalues, V = np.linalg.eigh(C)

# Sort components (Descending)
sorted_idx = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_idx]
```

Hybrid Prediction Logic



```python
# Predict
preds_hyb_list = []
actuals_cs = cs_test['rating'].values

for _, row in cs_test.iterrows():
    u, i = row['u_idx'], row['i_idx']
    pred_cb = cb_model_cs.predict(u, i)
    pred_cf = cf_model_cs.predict(u, i)

    # Use best alpha
    pred_h = best_alpha * pred_cb + (1-best_alpha) * pred_cf
    preds_hyb_list.append(pred_h)

rmse_curr = np.sqrt(mean_squared_error(actuals_cs, preds_hyb_list))
print(f"N={n_keep} -> Hybrid RMSE: {rmse_curr:.4f}")
cold_results.append(rmse_curr)
```



```
--- Cold-Start Simulation ---
Simulating Cold-Start with 3 ratings/user...
Training SVD (Eigen-decomposition) k=20...
N=3 -> Hybrid RMSE: 1.5390
Simulating Cold-Start with 5 ratings/user...
Training SVD (Eigen-decomposition) k=20...
N=5 -> Hybrid RMSE: 1.5312
Simulating Cold-Start with 10 ratings/user...
Training SVD (Eigen-decomposition) k=20...
N=10 -> Hybrid RMSE: 1.5358
```

Overall Conclusions

To conclude, buy building the recommendation system and having to apply several technqiues to different datasets allowed us to understand how real recommendation system works and how it can be dealt with depending on the challenges it can face like long-tail, cold start, and others. All of these problems where face in this submission and some solutions where stated to decrease its affect.

It was also noticed that sometimes using one approach for recommendation Is not enough and there is a no solution that fits all so it really depands on the data and domain where it will specify which approach would be best or would it be better combined.

Appendix A: AI Assistance Acknowledgment

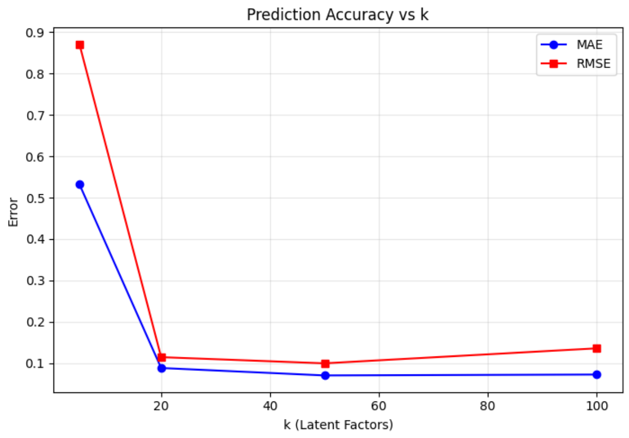
I acknowledge that AI (ChatGPT, Gemini, etc) was utilized in optimization, formating and visualizations, help in solving errors, and recommending solutions to faced challanges.

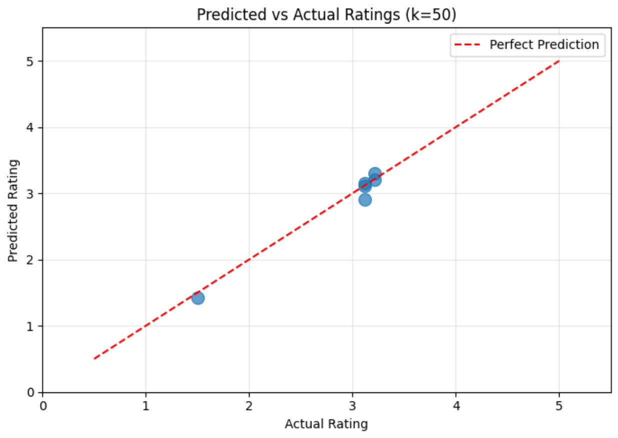
Appendix B: Team Contribution Breakdown

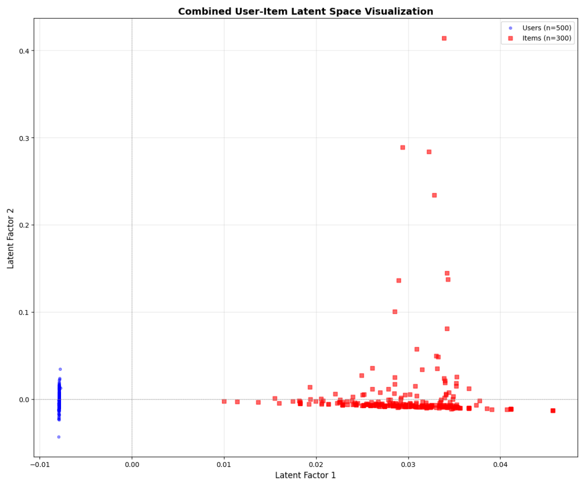
The tasks where divided according to the table below and each contributed to code, report, and etc.:

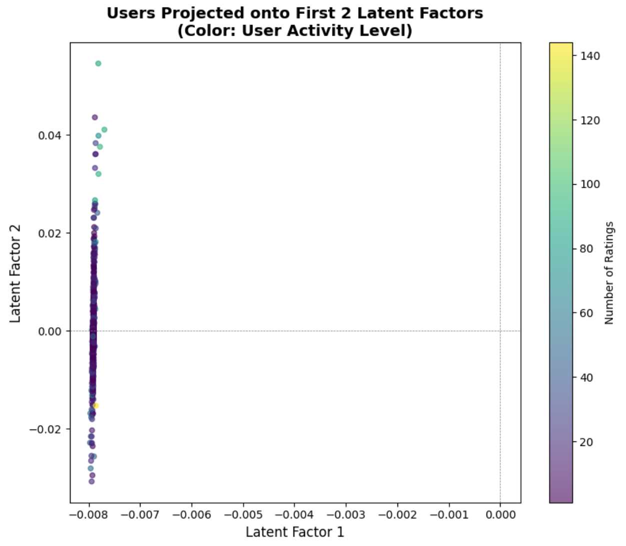
Name	ID	Contributions
Menna Salem Elsayed	221101277	Section 1, part 1 & Part 3 step (1,2,3,5,7)
Arwa Ahmed Mostafa	221100209	Section 1, part 2 & part3 step (4,6,8)
Hala Soliman	222102480	Section 2, Part 1 &2
Malak Amgad	221100451	Section 2, Part 3

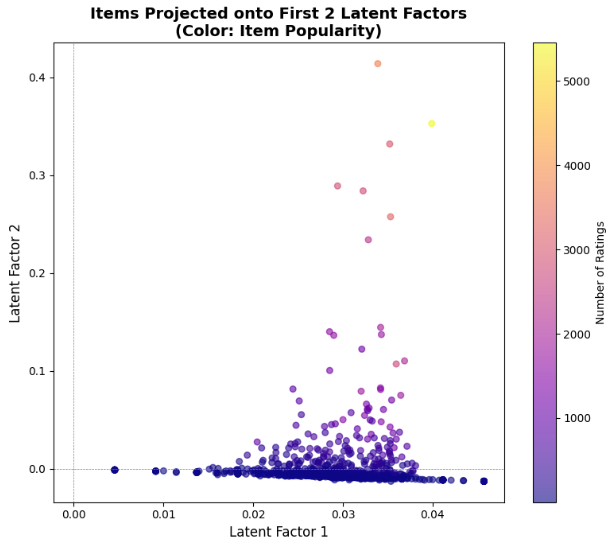
Appendix C: Additional Visualizations


Prediction Accuracy vs k
Error
k (Latent Factors)
MAE
RMSE


Predicted vs Actual Ratings (k=50)
Predicted Rating
Actual Rating
Perfect Prediction


Combined User-Item Latent Space Visualization
Latent Factor 2
Latent Factor 1
Users (n=500)
Items (n=300)


Users Projected onto First 2 Latent Factors
(Color: User Activity Level)
Latent Factor 2
Latent Factor 1
Number of Ratings


Items Projected onto First 2 Latent Factors
(Color: Item Popularity)
Latent Factor 2
Latent Factor 1
Number of Ratings