eBuss Sentiment-Based Product Recommendation System Project Report

1. Problem Statement

eBuss is an e-commerce platform offering a wide range of consumer goods—from household essentials and personal care items to electronics and books. To stand out amid fierce competition (Amazon, Flipkart, etc.), eBuss sought a richer, more user-centric recommendation experience that not only leverages collaborative filtering but also accounts for the **sentiment** in product reviews.

Traditional CF may recommend items a user is likely to purchase, but some of those items may have predominantly negative feedback in reviews. By integrating a sentiment analysis model trained on user reviews, we can filter out poorly reviewed items—boosting customer satisfaction and conversion.

Project Goal

Build and deploy an end-to-end product recommendation system that:

- Generates personalized top-20 product suggestions via collaborative filtering.
- Re-ranks those suggestions by positive review sentiment, surfacing a final top-5 list per user.
- CF + Sentiment: CF captures purchase patterns; sentiment adds qualitative feedback.
- Hybrid Pipeline: Precompute CF candidates for sub-second lookup; sentiment scores aggregated offline.
- Modular Design: Clear separation of data prep, model training, and deployment.
- Exposes a lightweight Flask web interface where any existing eBuss username returns its 5 best products.

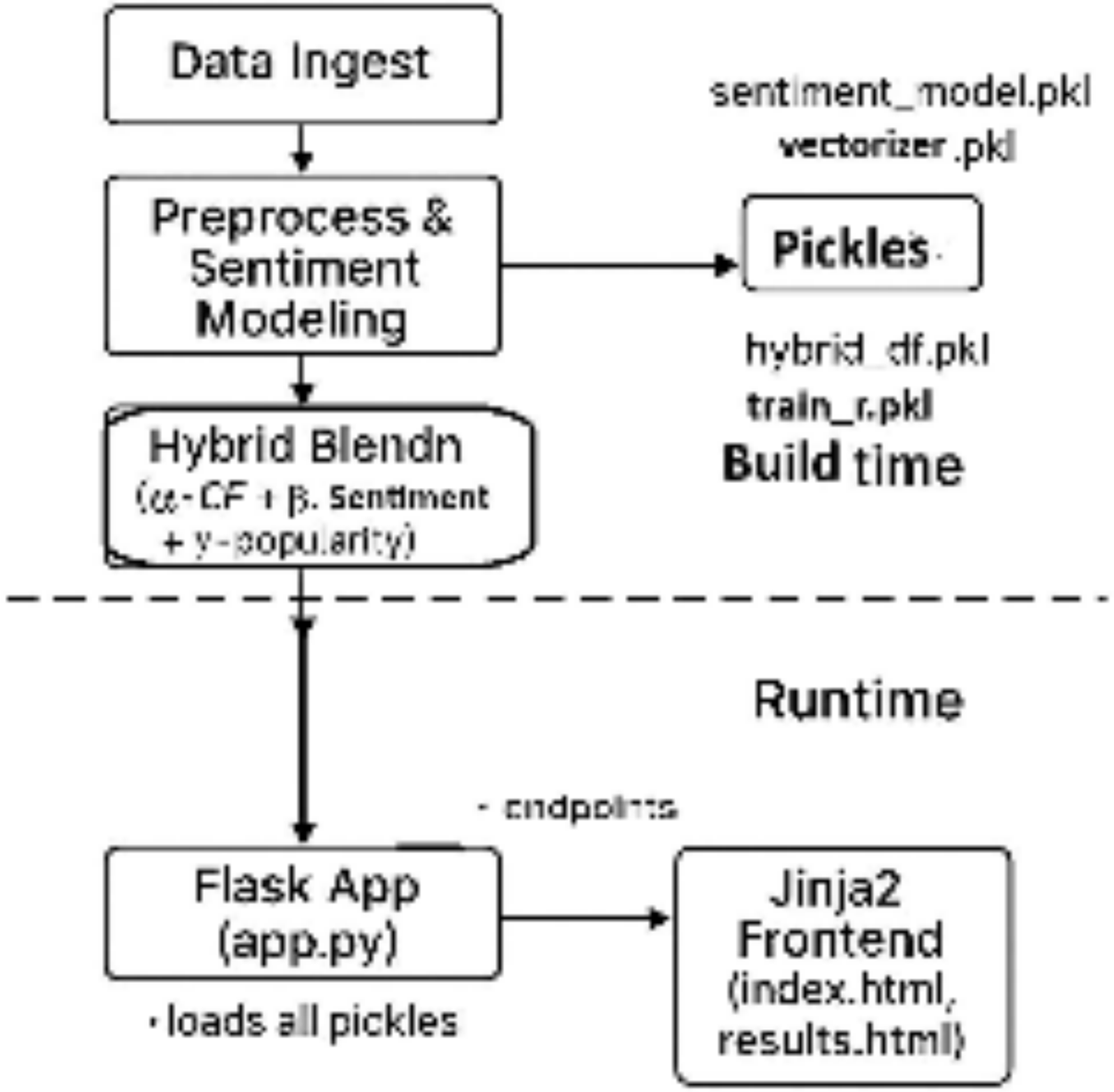
Data Sources

• Sample30.csv: 30 000 Amazon-style reviews spanning 200+ products and 20 000+ users.
• Reviews fields:
    o reviews_username (user ID)
    o name (product)
    o reviews_rating (1–5 stars)
    o reviews_text (free-text review)
    o sentiment (Positive/Negative)

- review_username, name (product), reviews_rating are used for Collaborative Filtering (CF)

- reviews_text and sentiment are used to build sentiment classification model

System Architecture



Component Breakdown

1. Data Prep & Sentiment

o Text cleaning: lowercase, non-alphabetic removal, stopword removal, lemmatization.
o Label ratings ≥ 4 as positive, < 4 negative; augment negatives via synonym replacement.
o TF-IDF vectorization (5 000 features) → train four models (LR, RF, XGB, NB) under GridSearchCV → select best by F1.

2. Collaborative Filtering

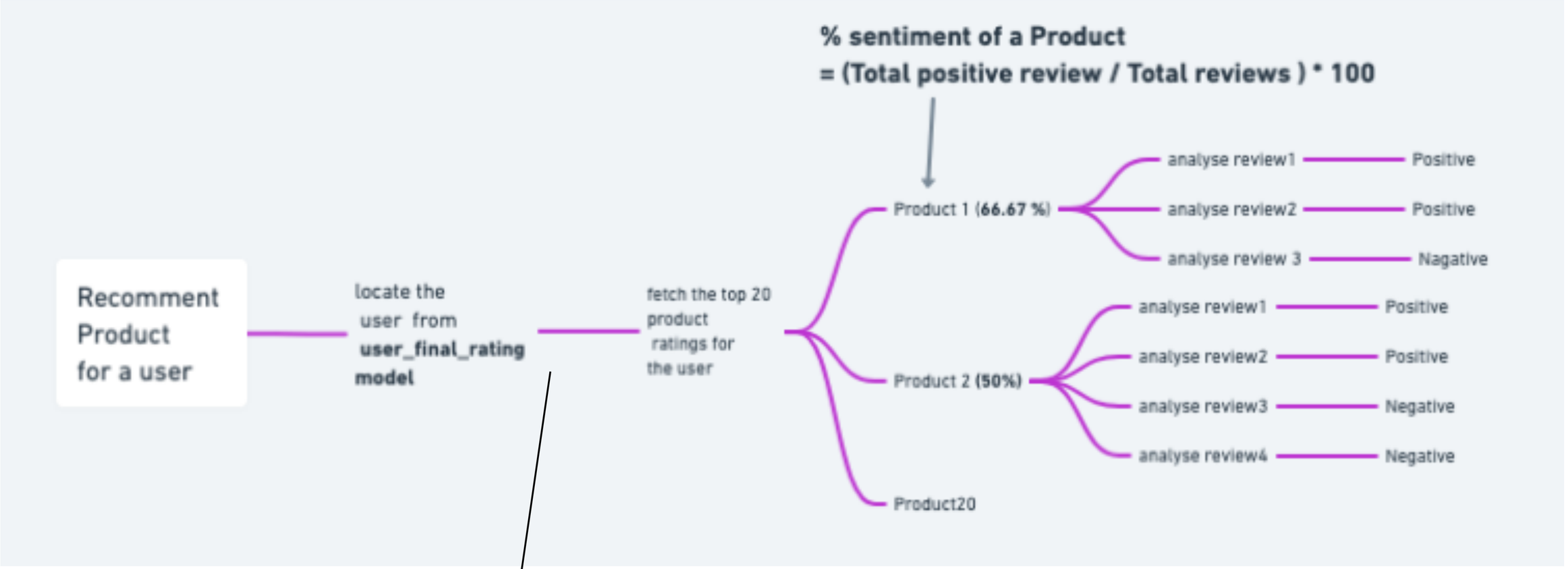
o Pivot into user–item rating matrix.
o Leave-one-out split to compute UBCF & IBCF via adjusted-cosine similarity.
o Select the better by RMSE; blend UBCF/IBCF as final CF score.

3. Hybrid & Sentiment Re-Ranking

o Compute per-item sentiment score = mean positive prediction across all reviews.
o Compute popularity score = normalized review frequency.
o Hybrid score = $\alpha \cdot CF + \beta \cdot sentiment + \gamma \cdot popularity$; mask already-rated items.
o For each user, pick top-20 CF candidates, then top-5 by sentiment

4. Deployment

Combining both CF and Sentiment Model




```
user_final_rating.loc[user_input].sort_values(ascending=False)[0:20]
```

Detailed Development Workflow

1. **Data Preprocessing** - Remove missing rows, clean text data, stop words removal, lemmatisation, correcting sentiment based on review rating (ratings ≥ 4 as positive, < 4 negative), Augmenting minor class (negative) review data using synonym replacement, Handling class imbalance using random oversampling technique.
2. **Exploratory Data Analysis** - Analyse N-grams in text data using CountVectorizer, Topic modelling using LSA (Latent Semantic Analysis) and NMF (Non-negative Matrix Factorisation), Word Cloud.
3. **Data Processing** - Feature extraction using TF-IDF vectorisation with n-grams, Dimensionality reduction using LSA.
4. **Sentiment Model Building - T**rain-Test data split, fit the processed train data into various classifiers like logistic regression, Naive Bayes, Random Forest, XG-Boost with Hyper parameter turning, Also build deep learning models like LSTM and BERT.
5. **Sentiment Model Evaluation** - Evaluate best performing classification model using test data and measure each of their performance using F1 score, accuracy, precision and recall. And choose the best performing model
6 **Collaborative Filtering -** Compute User based CF (UBCF) matrix and Item based CF (IBCF) matrix. Evaluate the Prediction using k-Nearest Neighbors (k-NN) and RMSE. Form a hybrid model with wieghted sum of both UBCF and IBCF to take advantage of both model. Fit Alternating Least Squares (ALS) the CF matrix to reduce the dimension and improve the accuracy. Blend these models & Evaluate the performance,
7. **Artifact Serialization** - serialize best sentiment model, td-idf vestorization model, and best performing cf model to pickle files
8. **Flask App & Templates** - Integrate all these models to perform sentiment based product recommendation.
9. **Continuous Deployment** on Heroku

Detailed Development Workflow

1. Data Preprocessing
- Remove rows with missing values.
- Clean text data: remove stop words, apply lemmatization.
- Adjust sentiment labels based on review ratings (ratings ≥ 4 as positive, < 4 as negative).
- Augment minority class (negative reviews) using synonym replacement.
- Address class imbalance using random oversampling.

2. Exploratory Data Analysis (EDA)
- Analyze N-grams using CountVectorizer.
- Perform topic modeling using Latent Semantic Analysis (LSA) and Non-negative Matrix Factorization (NMF).
- Generate word clouds for visual insights.

3. Feature Engineering
- Extract features using TF-IDF vectorization with N-gram support.
- Apply dimensionality reduction using LSA.

4. Sentiment Model Building
- Split data into training and test sets.
- Train multiple classifiers: Logistic Regression, Naive Bayes, Random Forest, XGBoost (with hyperparameter tuning).
- Develop deep learning models: LSTM and BERT.

5. Sentiment Model Evaluation

- Evaluate models on test data using F1 score, accuracy, precision, and recall.
- Select the best-performing model based on overall metrics.

6. Collaborative Filtering

- Compute User-Based Collaborative Filtering (UBCF) and Item-Based Collaborative Filtering (IBCF) matrices.
- Evaluate predictions using k-Nearest Neighbors (k-NN) and Root Mean Square Error (RMSE).
- Build a hybrid model using a weighted sum of UBCF and IBCF to leverage both approaches.
- Apply Alternating Least Squares (ALS) for matrix factorization to reduce dimensionality and enhance accuracy.
- Blend models and evaluate overall performance.

7. Model Serialization

- Serialize the best sentiment model, TF-IDF vectorizer, and top-performing collaborative filtering model using pickle.

8. Flask Application & Templates

- Integrate all models into a Flask web application to deliver sentiment-based product recommendations

9. Continuous Deployment

- Deploy the application on Heroku for continuous integration and delivery.

Technologies Used

- Language & Framework: Python 3.9, Flask, Gunicorn
- Data & ML: pandas, NumPy, scikit-learn, XGBoost, imbalanced-learn,NLTK
- Recommender: custom CF implementation (no external library)
- Deployment: heroku.com, Python build scripts

Design Choices

- Offline Precomputation: All heavy lifts (model training, similarity matrices) happen at build time.
- Hybrid Blend: Balances CF signals with sentiment & popularity for a well-rounded score.
- Modularity: Easy to swap in new

Challenges Faced

- Imbalanced Sentiment Labels: Over-sampled negatives via synonym augmentation to avoid bias.
- Sparse Rating Matrix: Ensured robust similarity by de-meaning and filling NaNs.
- Deployment Without LFS Pickles: Pivoted to runtime artifact generation to simplify CI/CD.

Lessons Learned

• Balancing quantitative (ratings) and qualitative (text) signals yields more user-friendly recommendations.
• Precomputation at build time enables sub-second API latencies.
• Clear config management (explicit column names) prevents accidental index/column flips.

Future Work

• ALS / Matrix Factorization: test implicit-feedback methods via the implicit library.
• Online Updates: incremental retraining as new reviews arrive.
• A/B Testing: compare hybrid vs. pure CF in production for conversion lift.• Mobile App: lightweight React front-end for on-the-go recommendations.

Report Prepared by: Davis Varkey
Project URL: https://github.com/davisvarkey/sentiment_based_product_recommendation
Live Demo: https://retail-product-recommend-app-f7d438c6b0fe.herokuapp.com

Understanding User-Based Collaborative Filtering

what a user is more likely to prefer is highly correlated to what the other users similar to him/her have liked in the past. Used in e-commerce, product and services

Predict rating of Nikhil for each movie he has not seen

User	Fast 7 Furious	Star Wars	Toy Story	Pulp Fiction
Vivek	5	5	1	4
Nikhil	-	4	2	4.5
Abhirami	1	2	-	-
Reetesh	-	-	5	-

User vector

pred(Nikhil, Fast & Furious)
= (5(0.91) +1(0.70))/(0.91 + 0.7)
= 3.26

To summarise the algorithm of user-based filters:

1. Find users similar to the user u (called the peer users) for whom predictions are to be made using any similarity measure like the **correlation coefficient**.
2. For each movie m that the user has not seen, calculate the **weighted average** of the ratings given to m by the peer users.
3. Recommend the top n movies to the user u.

Pearson correlation coefficient

Ai - ratings of Nikhil
Bi - ratings of Vivek

$$\text{similarity}(A,B) = \frac{\sum (A_i - \bar{A})(B_i - B)}{\sqrt{\sum (A_i - \bar{A})^2} \cdot \sqrt{\sum (B_i - \bar{B})^2}}$$

→ Nikhil ↔ Vivek -> 0.91
Nikhil ↔ Abhirami -> 0.7

Understanding Item-Based Collaborative Filtering

Collaborative based filtering depends on rating given by users to product movies

Predict rating Nikhil will give for each movie he has not seen

User	Fast 7 Furious	Star Wars	Toy Story	Pulp Fiction
Vivek	5	5	1	4
Nikhil	-	4	2	4.5
Abhirami	1	2	-	-
Reetesh	-	-	5	

Item Vector

pred(Nikhil, Fast & Furious) =

$$\frac{4 * 0.98 + 4.5 * 0.99}{0.98 + 0.99} = 4.25$$

To summarise the algorithm of Item-based filters:

1. Find items similar to the movie m (often called peer group of items) using a similarity measure like cosine.
2. Calculate the rating that the user will give to the movie m using the weighted average of the ratings given to the nearest movies by the user.
3. Recommend the top-n movies to the user.

Consine Similarities

Ai - ratings of Nikhil
Bi - ratings of Vivek

$$\text{similarity}(A, B) = \frac{\sum A_i B_i}{\sqrt{\sum A_i^2} \cdot \sqrt{\sum B_i^2}}$$

Fast and Furious ↔ Star Wars -> 0.99 Fast and Furious ↔ Pulp Fiction -> 0.98

Compute Collaborative Recommendation System

Product dataset (train)

id
name
reviews_rating
reviews_username

```
df_pivot =  pd.pivot_table(train,index= reviews_username, columns = id, values = reviews_rating)
```

df_pivot(70x6)

id	AV14LG0R-jtxr-f38QfS	AV16khLE-jtxr-f38VFn	AV1d76w7vKc47QAVhCqn	AV1h6Gu0glJLPUi8IjA_	AV1h6gSl-jtxr-f31p40	AV1l8zRZvKc47QAVhnAv
reviews_username						
5742870423	0.0	0.0	0.0	0.0	0.0	3.0
5alarm	0.0	0.0	5.0	0.0	0.0	0.0
allycat	0.0	0.0	0.0	0.0	0.0	3.0
alnscoob97	0.0	0.0	0.0	0.0	0.0	1.0
amanda	0.0	5.0	0.0	0.0	0.0	0.0
anonymous8589	0.0	0.0	0.0	5.0	0.0	0.0
ashley a	0.0	5.0	0.0	0.0	0.0	0.0
beccagrl532	0.0	1.0	0.0	0.0	0.0	0.0
bre234	0.0	1.0	0.0	0.0	0.0	0.0
browns fan	0.0	3.0	0.0	0.0	0.0	0.0

```
dummy_train = train.copy()

# The products not rated by user is marked as 1 for prediction.
dummy_train[value_column] = dummy_train[value_column].apply(lambda x: 0 if x>=1 else 1)

dummy_train = pd.pivot_table(dummy_train,index=user_column, columns = product_column, values = value_column)
```

dummy_train(70x6)

id	AV14LG0R-jtxr-f38QfS	AV16khLE-jtxr-f38VFn	AV1d76w7vKc47QAVhCqn	AV1h6Gu0glJLPUi8IjA_	AV1h6gSl-jtxr-f31p40	AV1l8zRZvKc47QAVhnAv
reviews_username						
5742870423	1.0	1.0	1.0	1.0	1.0	0.0
5alarm	1.0	1.0	0.0	1.0	1.0	1.0
allycat	1.0	1.0	1.0	1.0	1.0	0.0
alnscoob97	1.0	1.0	1.0	1.0	1.0	0.0
amanda	1.0	0.0	1.0	1.0	1.0	1.0
anonymous8589	1.0	1.0	1.0	0.0	1.0	1.0
ashley a	1.0	0.0	1.0	1.0	1.0	1.0
beccagrl532	1.0	0.0	1.0	1.0	1.0	1.0
bre234	1.0	0.0	1.0	1.0	1.0	1.0
browns fan	1.0	0.0	1.0	1.0	1.0	1.0

Predict User Rating

user_correlation (75x75) = consine_similarity(df_pivot(75x6))

```
[[1. 0. 1. ... 0. 0. 1.]
 [0. 1. 0. ... 0. 0. 0.]
 [1. 0. 1. ... 0. 0. 1.]
 ...
 [0. 0. 0. ... 1. 1. 0.]
 [0. 0. 0. ... 1. 1. 0.]
 [1. 0. 1. ... 0. 0. 1.]]
(75, 75)
```

user_predicted_ratings(75x6) = np.dot(user_correlation, df_pivot)

#since we are interested in products that are not rated by the user, we multiply with dummy train to make it zero

user_final_rating = np.multiply(user_predicted_ratings,dummy_train)

id	AV13O1A8GV-KLJ3akUyj	AV14LG0R-jtxr-f38QfS	AV16khLE-jtxr-f38VFn	AV1YGDqsGV-KLJ3adc-O	AV1YIch7GV-KLJ3addeG	AV1YlENIglJLPUi8IHsX	AV1YmBrdGV-KLJ3adewb
reviews_username							
00sab00	0.0	0.0	1.21	13.58	0.0	0.0	0.0
01impala	0.0	0.0	3.12	15.58	0.0	0.0	0.0
02dakota	0.0	0.0	3.12	15.58	0.0	0.0	0.0
02deuce	0.0	0.0	3.12	15.58	0.0	0.0	0.0
0325home	0.0	0.0	0.00	11.34	0.0	0.0	0.0

user_final_rating

Understand Content based Filtering(CF)

Item Vector (I)

	Action	Animation	Children
Mission Impossibl	1	0	0
Star Wars	1	1	0
Toy Story	0	1	1

Movie, books, etc

Eg User Review

	Review	Rating
Mission Impossible	1	Good
Star Wars	1	Good
Toy Story	1	Bad

User Vector (U)

	Action	Animation	Children
Nikhil	9	2	-6
Baby Tom	-6	2	9

Top Recommendations

Nikhil - Star Wars
Baby Tom - Toy Story

I x U

	Nikhil	Baby Tom
Mision Impuso	9	-6
Star Wars	11	-4
Toy Story	-6	9