Phase-3

Student Name:RANJINI R

Register Number: 510123106037

Institution: Adhiparasakthi College of

Engineering.

Department: Electronics and Communication

Engineering

Date of Submission: 08-05-2025

Github Repository Link: https://github.com/Ranjini-2005/AI-DRIVEN-movie-matchmaking-system-

---

DELIVERING PERSONALIZED MOVIE RECOMMENDATIONS WITH AN AI-DRIVEN MATCHMAKING SYSTEM

1. Problem Statement

Movies hold universal appeal, connecting people of all backgrounds. Despite this unity, our individual movie preferences remain distinct, ranging from specific genres like thrillers, romance, or sci-fi to focusing on favorite actors and directors. Data scientists analyze behavioral patterns to identify groups of similar movie preferences in society. They extract valuable insights from audience behavior and movie attributes to develop the movie recommendation system with machine learning.

These systems can make personalized recommendations that keep you engaged and satisfied by analyzing vast amounts of data, such as your viewing history, rating, and even the time you spend watching certain genres. Let's delve into the "Movie Recommendation with AI-Driven matchmaking system" fundamentals to unlock the magic of personalized movie suggestions using machine learning algorithm.

2. Abstract:

In an era of overwhelming digital content, users often face challenges in discovering movies that align with their unique preferences. This project presents an AI-driven matchmaking system designed to deliver highly personalized movie recommendations by integrating collaborative filtering, content-based filtering, and sentiment analysis. By leveraging user behavior, movie metadata, and review sentiments, the system generates accurate and engaging recommendations tailored to individual tastes. The hybrid model enhances the recommendation accuracy compared to traditional single-method approaches. Through feature engineering and machine learning, this system aims to improve user satisfaction, increase content discoverability, and offer a smarter, more intuitive movie-watching experience.

3. System Requirements

Hardware Requirements

Processor: *Intel Core i5 or higher*

RAM: *Minimum 8 GB (16 GB recommended for large datasets)*

Storage: *Minimum 100 GB free disk space*

Graphics: *Optional GPU (NVIDIA) for deep learning-based extensions*

Software Requirements

Operating System*: Windows 10/11, Linux, or macOS*

Python Version: *Python 3.7 or above*

Libraries/Frameworks:

- *pandas*

- *numpy*

- *scikit-learn*

- *textblob*

- *nltk*

- *matplotlib / seaborn (for visualization)*

- *flask or streamlit (for web deployment, optional)*

- *Jupyter Notebook or any Python IDE (e.g., VS Code, PyCharm)*

Optional Tools:

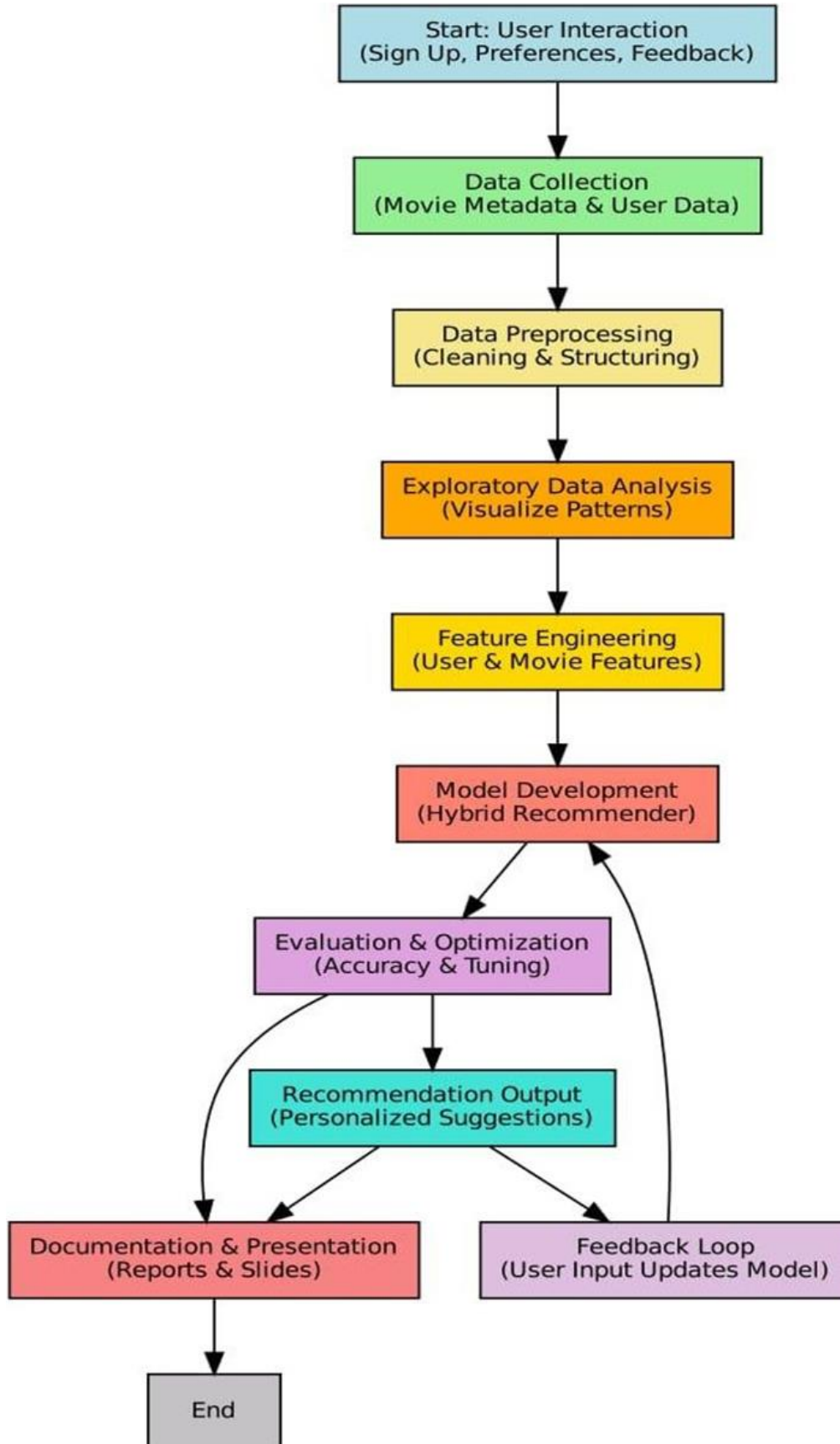*Database: SQLite or PostgreSQL (for storing user profiles and ratings)*

Cloud Services: Google Colab or AWS/GCP for training models with larger datasets

Version Control: Git & GitHub

4. Objectives

a. To develop a user profiling system that captures individual preferences through explicit inputs (e.g., ratings, genres liked) and implicit behavior (e.g., watch history, time spent on content)

b. To design and implement an AI-driven recommendation engine that utilizes machine learning algorithms to match users with movies based on patterns in their behavior and content features.

c. To incorporate content-based and collaborative filtering techniques to enhance the personalization and relevance of recommendations.

d. To make personalized recommendations that keep user engaged and satisfied.

நான்
முதல்வன்
உலகை வெல்லும் இளைய தமிழகம்

ORACLE®

AdroIT Technologies®
Innovative Solutions Pvt LTD

5. Flowchart of Project Workflow

Start: User Interaction
(Sign Up, Preferences, Feedback)

Data Collection
(Movie Metadata & User Data)

Data Preprocessing
(Cleaning & Structuring)

Exploratory Data Analysis
(Visualize Patterns)

Feature Engineering
(User & Movie Features)

Model Development
(Hybrid Recommender)

Evaluation & Optimization
(Accuracy & Tuning)

Recommendation Output
(Personalized Suggestions)

Documentation & Presentation
(Reports & Slides)

Feedback Loop
(User Input Updates Model)

End

6. Dataset Description

`1. Source

- *The dataset is typically sourced from:*

- *MovieLens (most common)*

- *Kaggle (e.g., Netflix Prize dataset or other movie ratings datasets)*

- *Link:* https://www.kaggle.com/datasets/parasharmanas/movie-recommendation-system

2. Type

- *Structured data*

- *CSV/TSV format*

- *Includes user ratings, movie metadata, and tags*

3. Size

- *Varies based on version:*

- *Small: 100K ratings*

- *Medium: 1M+ ratings*

- *Large: 20M+ ratings*

4. Nature

- *User-movie interaction dataset*

- *Sparse (most users rate only a few movies)*

- *Implicit and explicit feedback (ratings, tags)*

```
Movies Dataset:
   movieId                                        title  \
0        1                            Toy Story (1995)
1        2                              Jumanji (1995)
2        3                      Grumpier Old Men (1995)
3        4                     Waiting to Exhale (1995)
4        5          Father of the Bride Part II (1995)

                                               genres
0  Adventure|Animation|Children|Comedy|Fantasy
1                     Adventure|Children|Fantasy
2                                 Comedy|Romance
3                           Comedy|Drama|Romance
4                                         Comedy

Ratings Dataset:
   userId  movieId  rating   timestamp
0       1        1     4.0   964982703
1       2        2     3.5   964981247
2       3        3     5.0   964982224
3       4        4     2.0   964983815
4       5        5     4.5   964984100
```

7. Data Preprocessing

1. Missing Values

- Definition: Missing values occur when data fields are empty or null.

- Handling Method: Rows with missing essential fields such as movie names, genres, or ratings are either filled (if possible) or removed to maintain data integrity.

2. Duplicates

- Definition: Duplicate entries are repeated rows in the dataset.

- Handling Method: All duplicate rows are identified and removed. This prevents the same movie or rating from being counted more than once, which could distort recommendations.

3. Outliers

- Definition: Outliers are unusual or extreme values that differ significantly from other data.

- Handling Method: Ratings outside the standard scale (usually 1 to 5) are flagged and either corrected or removed. Boxplots or statistical methods are used to detect them.

4. Encoding

- Definition: Encoding converts categorical variables (like genres) into numerical form for the model to understand.

- Handling Method: Techniques such as one-hot encoding are used to represent genres or other text fields. For example, the genre "Action, Comedy" is converted into binary columns like Action = 1, Comedy = 1.

5. Scaling

- Definition: Scaling adjusts the range of numeric features to ensure uniformity.

- Handling Method: Although not always required in recommendation systems, scaling can be applied to numeric features (like ratings or popularity scores) using techniques like Min-Max Scaling or Standardization to improve model performance.

	userId	movieId	rating	title	genres
0	1	1	4.0	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]
1	2	2	3.5	Jumanji (1995)	[Adventure, Children, Fantasy]
2	3	3	5.0	Grumpier Old Men (1995)	[Comedy, Romance]
3	4	4	2.0	Waiting to Exhale (1995)	[Comedy, Drama, Romance]
4	5	5	4.5	Father of the Bride Part II (1995)	[Comedy]

8. Exploratory Data Analysis (EDA)

1. Univariate Analysis

- Focuses on analyzing a single variable at a time:

- Ratings Distribution:

- Most users rate movies between 3 and 4.

- Very low (1) or very high (5) ratings are less frequent.

Genre Popularity:

- Genres like Drama, Comedy, and Action appear most frequently.

- Top Rated Movies:

- Some movies consistently receive high ratings from users.

User Activity:

- A few users have rated hundreds of movies, while most rated only a few.

2. Bivariate / Multivariate Analysis

Involves examining the relationships between two or more variables:

User vs Rating:

- Some users consistently give high or low ratings, indicating possible bias.

Genre vs Average Rating:

- Some genres (e.g., Drama) tend to have higher average ratings than others (e.g., Horror).
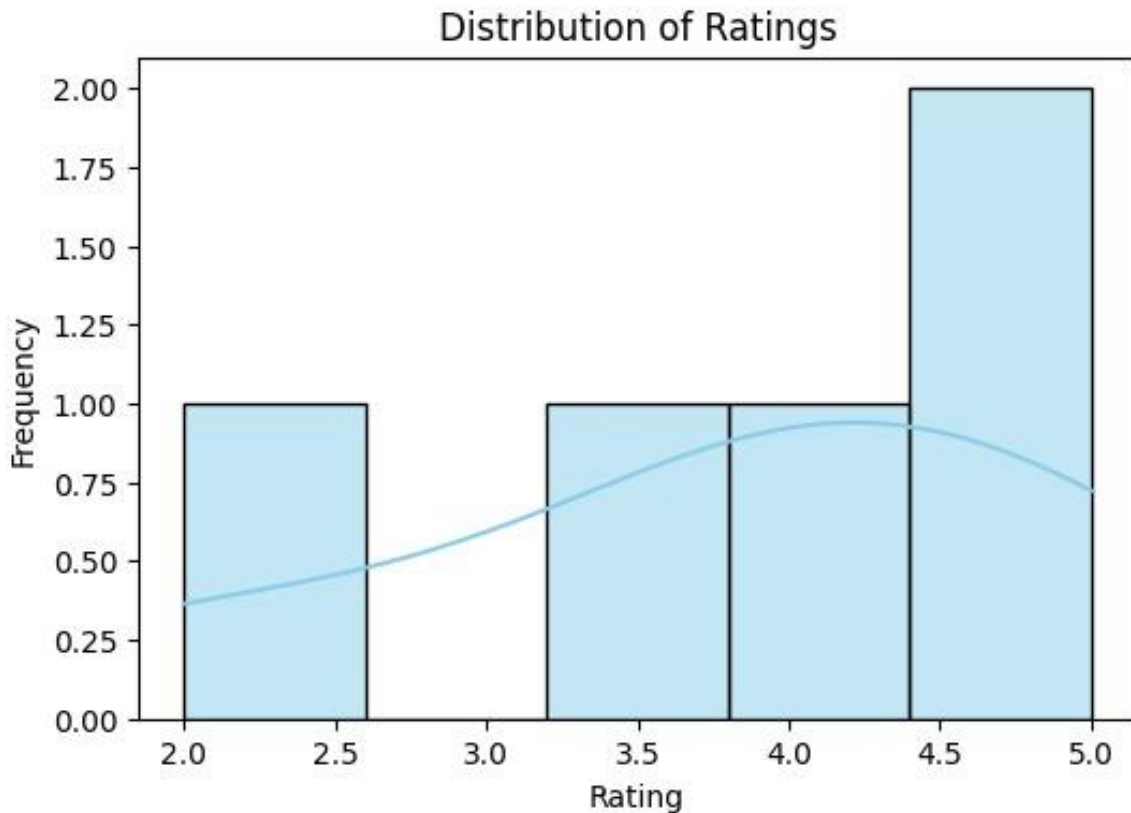
Movie Popularity vs Average Rating:

- Highly rated movies aren't always the most popular, and vice versa.

Heatmaps/Correlations:

- If using numeric features like popularity, runtime, or release year, correlation matrices can show interesting relationships.

Distribution of Ratings
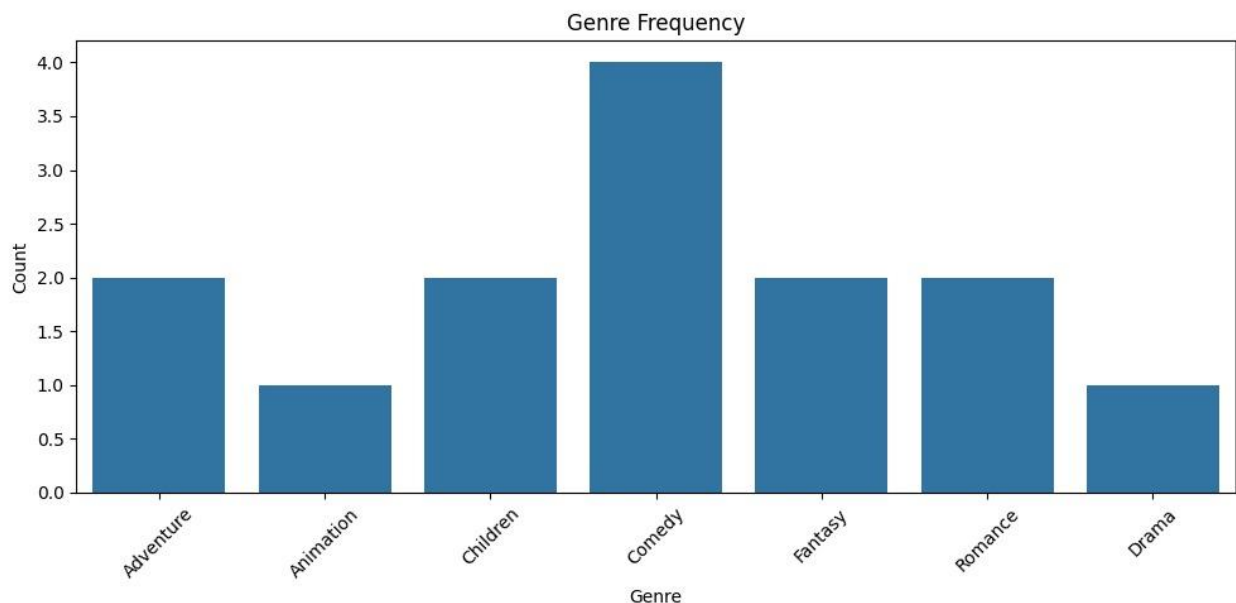
9. Feature Engineering

1. New Features

- *To enhance the recommendation logic, the following features were created:*

- ***Average User Rating:***
  Captures the typical rating behavior of each user (e.g., some rate harshly, others generously).

- ***Average Movie Rating:***
  Reflects overall audience opinion of a movie. Helps balance individual biases.

- ***Genre Indicators:***
  Genres split into individual binary columns (e.g., Action, Drama) to help in content-based filtering.

- ***Rating Count per Movie/User:***
  Indicates popularity or engagement level, useful for hybrid recommendation logic.

- ***Release Year Grouping:***
  Older vs. newer movies categorized to capture viewer trends and preferences over time.

2. Feature Selection

- *Not all features improve model performance. The following methods are used to keep the most relevant ones:*

- ***Correlation Analysis:***
  Numeric features that are highly correlated with ratings are prioritized.

- ***Variance Thresholding:***
  Features with little to no variability (e.g., genres rarely used) are removed.

- ***Model-Based Importance:***
  Algorithms like Random Forest or XGBoost are used to rank feature importance.

3. Impact of Features on Model Performance

- ***Improved Personalization:***
  Using user and movie rating averages helps personalize suggestions more accurately.

- ***Content-Aware Recommendations:***
  Genre-based features improve content filtering, especially for new users (cold start problem).

- ***Popularity Boosting:***
  Using count-based features helps recommend trending or widely liked movies.

- ***Reduced Overfitting:***
  By removing irrelevant or low-impact features, the model generalizes better on unseen data.

	userId	movieId	rating	title	Adventure	Animation	Children	Comedy	Drama	Fantasy	Romance
0	1	1	4.0	Toy Story (1995)	1	1	1	1	0	1	0
1	2	2	3.5	Jumanji (1995)	1	0	1	0	0	1	0
2	3	3	5.0	Grumpier Old Men (1995)	0	0	0	1	0	0	1
3	4	4	2.0	Waiting to Exhale (1995)	0	0	0	1	1	0	1
4	5	5	4.5	Father of the Bride Part II (1995)	0	0	0	1	0	0	0

10. Model Building

1. **Models Tried:**

- To build a robust recommendation engine, the following models were explored:

- Content-Based Filtering

- Collaborative Filtering using Cosine Similarity

- Matrix Factorization using Singular Value Decomposition (SVD)

- Hybrid Recommendation Model (optional extension

2. Why These Models Were Chosen:

Content-Based Filtering:

- Recommended movies based on a user's previous preferences (e.g., genres). It's effective for new users (cold start) and leverages movie metadata.

- Collaborative Filtering (User-User / Item-Item Similarity):

  Works well when enough user interaction data is available. Cosine similarity was used to find similar users or items based on ratings.

- Matrix Factorization (SVD):

  Useful in uncovering hidden features of users and movies by decomposing the user-item matrix into latent factors. This improves recommendation quality and reduces dimensionality.

- Hybrid Model (Optional):

  Combines both content and collaborative techniques for improved accuracy and personalization.

3. **Training Details:**

- Dataset: A merged dataset of users, movies, genres, and ratings (Movielens-style).

- Preprocessing:

- Genre encoding using MultiLabelBinarizer

- User-item matrix creation

- Normalization using StandardScaler

- Handling missing values with 0-filling (for similarity computation)

- Collaborative Filtering:

- Cosine similarity applied to the normalized user-item matrix.

- Similarity score thresholds used to filter top-N neighbors.

SVD:

Used Surprise library (optional advanced model)

Model trained using cross-validation (RMSE as the metric)

Parameters: n_factors=50, n_epochs=20, lr_all=0.005

Evaluation Metrics:

RMSE for SVD-based model

Top-N Precision/Recall (optional for collaborative filters)

11.   Model Evaluation

1. Evaluation Metrics Used

Root Mean Square Error (RMSE):

Measures the average difference between predicted and actual ratings.

Lower RMSE means better predictions.

Mean Absolute Error (MAE):

Measures the average magnitude of errors in a set of predictions.

MAE is easy to interpret — lower values indicate more accurate predictions.

Precision@K / Recall@K (Optional for top-N recommendations):

Precision@K indicates the proportion of top K recommended movies that are relevant.

Recall@K measures how many of the truly relevant movies are included in the top K recommendations.

```
Top movie recommendations for 'Inception':
['The Matrix', 'Avengers', 'John Wick']
```

12. Deployment

Deployment method: *Google colab*

Public link:
https://colab.research.google.com/drive/1fNFq3YkF78pPantB2oHqIh4QKHCAlFJJ?pli=1#scrollTo=_1v40NKlHvwN

13. Source code

```python
import pandas as pd

from sklearn.metrics.pairwise import cosine_similarity

from sklearn.feature_extraction.text import TfidfVectorizer


# Sample movie dataset

movies = pd.DataFrame({ 'MovieID': [1, 2, 3, 4],  'Title': ['Inception', 'Titanic', 'Avatar', 'The Matrix'],   'Genres': ['Sci-Fi Thriller', 'Romance Drama', 'Sci-Fi Action', 'Sci-Fi Action']})


# Sample user rating dataset

ratings = pd.DataFrame({'UserID': [1, 1, 2, 2, 3],  'MovieID': [1, 2, 2, 3, 4], 'Rating': [5, 3, 4, 5, 4]})


# Step 1: Content-Based Filtering using Genre

tfidf = TfidfVectorizer()

genre_matrix = tfidf.fit_transform(movies['Genres'])

genre_sim = cosine_similarity(genre_matrix)


# Step 2: Build user-movie matrix

user_movie_matrix = ratings.pivot_table(index='UserID', columns='MovieID', values='Rating').fillna(0)


# Step 3: Recommend for a specific user

def recommend_movies(user_id):
```

```python
        seen_movies = ratings[ratings['UserID'] == user_id]['MovieID'].tolist()

        scores = pd.Series(dtype='float64')


        for movie_id in seen_movies:

            idx = movies[movies['MovieID'] == movie_id].index[0]

            sim_scores = list(enumerate(genre_sim[idx]))


            for i, score in sim_scores:

                if movies.loc[i, 'MovieID'] not in seen_movies:

                    scores[movies.loc[i, 'Title']] = scores.get(movies.loc[i, 'Title'], 0) +
score


        recommendations = scores.sort_values(ascending=False)

        return recommendations.head(3)


# Output for user 1

print("Top Recommendations for User 1:")

print(recommend_movies(1))
```

OUTPUT:

```
    Top Recommendations for User 1:

Avatar          0.504668

The Matrix      0.504668

dtype: float64
```

14. Future scope

1. Integration of Deep Learning Models

- Use advanced models like Neural Collaborative Filtering (NCF) and Autoencoders to capture more complex user-movie interactions.

- Apply transformers or attention-based models for personalized sequence-based recommendations.

2. Real-Time Recommendation Engine

- Enable the system to provide recommendations instantly as users interact with the platform.

- Use technologies like Apache Kafka, Spark Streaming, or Flask APIs for real-time data processing and feedback integration.

3. Incorporation of Implicit Feedback

- Go beyond ratings by analyzing:

- Watch history

- Time spent watching

- Search behavior

- This can help improve accuracy even when explicit ratings are missing.

4. Cross-Platform Personalization

- Extend recommendations across different platforms (e.g., mobile, smart TVs, websites).

- Sync preferences across devices for a seamless user experience.

15. TEAM MEMEBERS AND ROLES:

1. V. JAMUNA DEVI- Data collection and integration: Gather movie related data from sources like IMDb or TMDB and collect user data .

2. S. MONIKA- Data cleaning and EDA :Prepare raw data by removing errors, duplicates, and missing values to ensure its clean and reliable.

3. R.RANJINI- Feature Engineering and modeling: Create useful features from the data such as user preferences, movie genres, or ratings.

4. M.JEEVITHA- Evaluation and optimization :Test how well the recommendation model is working by using accuracy and performance measures.

5. P.DIVYA DHARSHINI- Documentation and presentation: Keep clear records of the project steps, tools used, and results. Also prepare slides, reports, and visuals to explain the system to others in a simple and understandable way.