**GitHublink :** [**https://github.com/mani-1903/movie-recommendations**](https://github.com/mani-1903/movie-recommendations)

**Delivering personalized movie recommendations with an AI-driven match making system**

**PHASE-3**

**Student Name: MANIKANDAN.P**

**Register Number: 732423104013**

**Institution: SASURIE COLLEGE OF ENGINEERING**

**Department: BE.ELECTRONICS AND COMMUNICATION ENGINEERING**

**Date of Submission: 14.05.2025**

# 1. Problem Statement

* In the digital age, streaming platforms offer vast libraries of movies, making it increasingly difficult for users to find content that matches their preferences.
* Existing recommendation systems often rely on basic algorithms that lack personalization, leading to user dissatisfaction and decreased engagement.
* There is a need for an AI-driven matchmaking system that can accurately analyze user preferences and deliver tailored movie recommendations, enhancing user experience and satisfaction.

# 2. Abstract

In an era of overwhelming content availability, delivering personalized movie recommendations has become essential for enhancing user engagement and satisfaction. This paper presents an AI-driven matchmaking system designed to analyze user preferences, behavioral patterns, and contextual factors to generate tailored movie suggestions. By integrating machine learning techniques such as collaborative filtering, content-based filtering, and deep learning models, the system dynamically adapts to individual user profiles and evolves with their changing tastes. The proposed architecture leverages user interaction data, genre affinity, sentiment analysis, and metadata tagging to establish high-accuracy matches between users and films. Experimental results demonstrate the system’s effectiveness in improving recommendation precision and user retention compared to traditional methods. This approach not only personalizes the viewing experience but also addresses cold-start

challenges and scalability, making it suitable for modern streaming platforms and digital entertainment services

# 3. System Requirements

**Hardware Requirements:**

**RAM:** Minimum 8 GB (16 GB recommended for large datasets or deep learning) **Processor:** Intel i5 or AMD equivalent (GPU recommended for deep learning)

**Software Requirements:**

**Python Version:** 3.8+

**Libraries:**

* pandas, numpy
* scikit-learn
* surprise (for collaborative filtering)
* LightFM / TensorFlow / PyTorch (for deep models)
* matplotlib, seaborn (for visualization)
* streamlit / gradio (for deployment)

**IDE:** Google Colab, Jupyter Notebook, or VS Code

# 4. Objectives

1. Enhanced User Experience: Create a seamless and intuitive interface where users receive personalized movie suggestions without manual filtering.

2. Dynamic Adaptation: Continuously update recommendations based on real-time user interactions, preferences, and trending content.

3. Diverse Data Utilization: Integrate data from multiple sources, including user ratings, watch history, genre preferences, and social media trends, for a comprehensive recommendation model.

4. Scalability and Efficiency: Develop the system to handle large datasets efficiently while maintaining high performance and responsiveness.

5. Explainable Recommendations: Incorporate features that allow users to understand why specific movies are recommended, building trust and transparency.

6. Cross-Platform Integration: Ensure compatibility across various streaming platforms and devices to deliver consistent recommendations.

7. User Privacy and Data Security: Implement robust data protection measures to ensure user information remains confidential and secure.

# 5. Flowchart of Project Workflow



**data**

**collection**



**Data**

**preprocessing**



**Model**

**Building**



**Model**

**Evaluation**



**Exploratory**

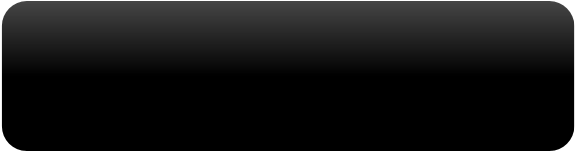
**Data**

**Analysis**



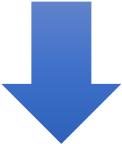
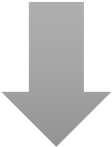
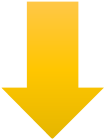
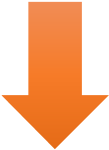
**Feature**

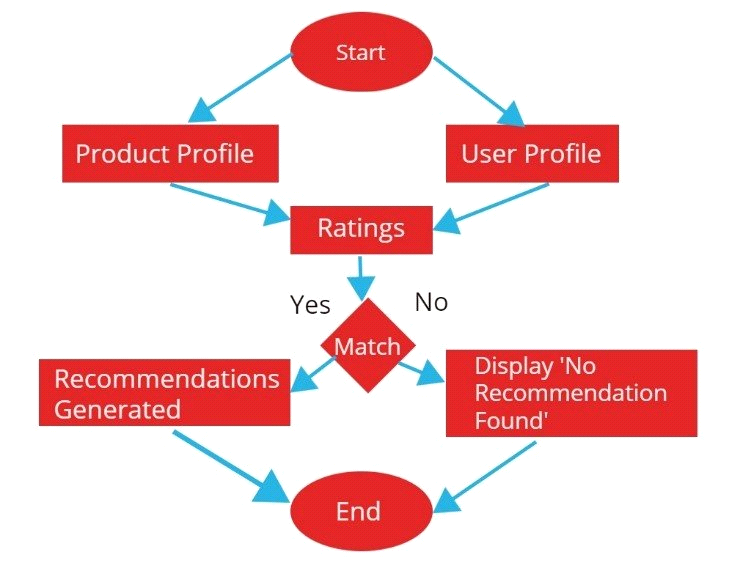
**Engineering**



**Feature**

**Engineering**





# 6. Dataset Description

**Source:** Kaggle (e.g., MovieLens Dataset) **Type:** Public **Size:**

* ~20 million ratings
* ~138,000 users
* ~27,000 movies
* Multiple CSV files (ratings.csv, movies.csv, tags.csv)



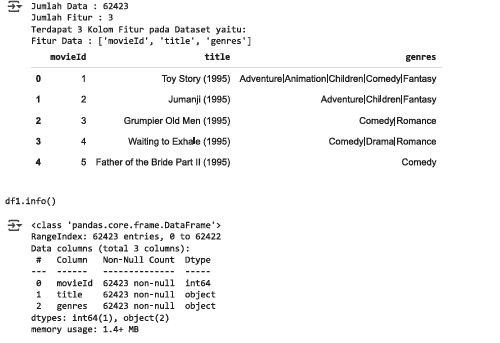
**Sample**

**:**

**(**

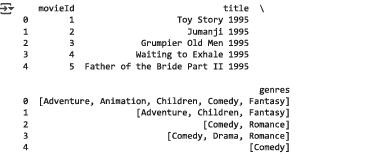
df.head()

**)**



# 7. Data Preprocessing

* **Missing Values:** Checked and dropped/inferred based on movie metadata.
* **Duplicates:** Removed duplicate ratings.
* **Encoding:** Mapped genres to binary vector features (multi-hot encoding).
* **Scaling:** Not typically required for IDs, but used for metadata features (e.g., popularity scores).



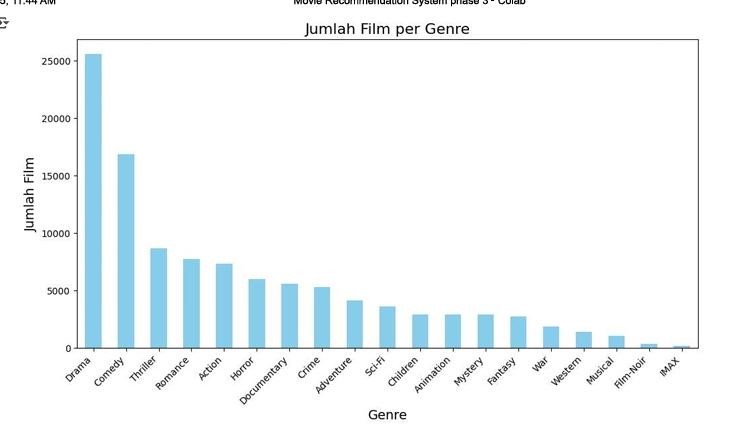
# 8. Exploratory Data Analysis (EDA)

**Visualizations:**

* Ratings distribution histogram
* Heatmap of user-movie interactions
* Most rated and highest rated movies Correlation between genres and average ratings

**Insights:**

* Users rate fewer movies over time → cold start problem
* Certain genres (e.g., Drama, Action) dominate user interest



# 9. Feature Engineering

**New Features:**

* Genre vector encoding
* Average user rating
* Movie popularity index
* **Selection:** Filtered active users (>20 ratings) and popular movies
* **Transformations:** Normalized rating scale, vector embeddings for hybrid models



# 10. Model Building

**Models Tried:**

**Content-Based Filtering:** Cosine similarity on genre vectors **Collaborative Filtering:**

* Matrix Factorization (SVD, NMF)
* k-NN based user/movie similarity

**Hybrid Models:** Combining metadata + collaborative signals **Justification:**

Content-based: good for cold-start Collaborative: scalable with usage

Hybrid: balances both weaknesses

# 11. Model Evaluation

**Metrics:**

* **RMSE** (for rating prediction)
* **Precision@k, Recall@k** (for top-k recommendations) **Coverage and Diversity**

**Visuals:**

* Precision/Recall plot
* Comparison bar chart (SVD vs. kNN vs. Hybrid)

# 12. Source Code

|  |
| --- |
| # IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES, # THEN FEEL FREE TO DELETE THIS CELL.  # NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON # ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR # NOTEBOOK.  import kagglehub parasharmanas\_movie\_recommendation\_system\_path = kagglehub.dataset\_download('parasharmanas/movie-recommendation-system')  print('Data source import complete.') import pandas as pd import numpy as np df1 = pd.read\_csv("/kaggle/input/movie-recommendation-system/movies.csv") print('Jumlah Data :', len(df1.iloc[:,1])) print('Jumlah Fitur :', len(df1.iloc[1,:])) print(f'Terdapat {len(df1.iloc[1,:])} Kolom Fitur pada Dataset yaitu:') print('Fitur Data :', df1.columns.tolist()[:]) pd.options.display.max\_columns = None df1.head() df1.info() import re  def clean\_title(title):  return re.sub("[^a-zA-Z0-9 ]", "", title) # Pisahkan genre menggunakan pemisah '|' df1['genres'] = df1['genres'].str.split('|')  # Bersihkan judul film df1['title'] = df1['title'].apply(clean\_title) |

# Perbarui movies\_data movies\_data = df1[['movieId', 'title', 'genres']]

# Mendapatkan genre unik dari semua film unique\_genres = pd.Series([genre for genres\_list in movies\_data['genres'] for genre in genres\_list]).unique()

|  |
| --- |
| # Output hasil  print(movies\_data.head()) print(f"Terdapat {movies\_data['title'].nunique()} Judul Film") print(f"Terdapat {len(unique\_genres)} Genre Film.") print("Genre Film:", unique\_genres)  # Periksa jumlah baris dengan '(no genres listed)'  no\_genres\_count = movies\_data[movies\_data['genres'].apply(lambda x: '(no genres listed)' in x)].shape[0]  print(f"Terdapat {no\_genres\_count} film tanpa genre.")  # Hapus baris dengan '(no genres listed)' movies\_data = movies\_data[~movies\_data['genres'].apply(lambda x: '(no genres listed)' in x)] |

# Perbarui daftar genre unik unique\_genres = pd.Series([genre for genres\_list in movies\_data['genres'] for genre in genres\_list]).unique()

# Tampilkan hasil setelah penghapusan print(f"Setelah penghapusan, terdapat {movies\_data['title'].nunique()} Judul Film.") print(f"Terdapat {len(unique\_genres)} Genre Film setelah pembaruan.")

|  |
| --- |
| print("Genre Film:", unique\_genres) import pandas as pd import matplotlib.pyplot as plt  # Menghitung jumlah film per genre genre\_counts = pd.Series([genre for genres\_list in movies\_data['genres'] for genre in genres\_list]).value\_counts() |
| plt.figure(figsize=(12, 6)) genre\_counts.plot(kind='bar', color='skyblue') plt.title('Jumlah Film per Genre', fontsize=16) plt.xlabel('Genre', fontsize=14) plt.ylabel('Jumlah Film', fontsize=14) plt.xticks(rotation=45, ha='right') plt.show() df2 = pd.read\_csv("/kaggle/input/movie-recommendation-system/ratings.csv") print('Jumlah Data :', len(df2.iloc[:,1])) print('Jumlah Fitur :', len(df2.iloc[1,:])) print(f'Terdapat {len(df2.iloc[1,:])} Kolom Fitur pada Dataset yaitu:') print('Fitur Data :', df2.columns.tolist()[:]) pd.options.display.max\_columns = None df2.head()  # Drop timestamp column  ratings\_data = df2.drop(['timestamp'], axis=1) print(ratings\_data.head()) # Melihat Missing Values  print("Jumlah Missing Values per Kolom:") print(df2.isnull().sum()) print("\nJumlah Data Duplicates:") print(df2.duplicated().sum()) ratings\_data.info() print("Distribusi Rating:") print(df2['rating'].value\_counts()) print("\nRating Rata-Rata per Film:") print(df2.groupby('movieId')['rating'].mean().head()) print("\nRating Rata-Rata per Pengguna:") print(df2.groupby('userId')['rating'].mean().head()) plt.show()import seaborn as sns |
| plt.figure(figsize=(8, 6))  sns.histplot(df2['rating'], bins=5, kde=False, color='skyblue') plt.title('Distribusi Rating', fontsize=16) plt.xlabel('Rating', fontsize=14) plt.ylabel('Frekuensi', fontsize=14) combined\_data = ratings\_data.merge(movies\_data, on='movieId') print(combined\_data.head()) # Rating Rata-Rata per Film avg\_ratings\_per\_movie =  combined\_data.groupby('title')['rating'].mean().sort\_values(ascending=False) print("Top 10 Film dengan Rating Rata-Rata Tertinggi:") print(avg\_ratings\_per\_movie.head(10)) movie\_rating\_counts = combined\_data.groupby('title')['rating'].count().sort\_values(ascending=False) print("Top 10 Film dengan Jumlah Rating Terbanyak:") print(movie\_rating\_counts.head(10))  # Memisahkan Genre exploded\_data = combined\_data.explode('genres') |

# Menghitung Jumlah Pengguna yang Memberi Rating pada Tiap Genre users\_per\_genre = exploded\_data.groupby('genres')['userId'].nunique()

# Rata Rata Rating di Tiap Genre avg\_rating\_per\_genre = exploded\_data.groupby('genres')['rating'].mean()

|  |
| --- |
| # Cari 3 film terbaik berdasarkan rating rata-rata di tiap genre top\_movies\_per\_genre = ( exploded\_data.groupby(['genres', 'title'])['rating']  .mean()  .reset\_index()  .sort\_values(['genres', 'rating'], ascending=[True, False])  .groupby('genres')  .head(3)  ) |

print("Jumlah Pengguna yang Memberi Rating pada Tiap Genre:") print(users\_per\_genre) print("\nRata-Rata Rating per Genre:") print(avg\_rating\_per\_genre) print("\n3 Film Terbaik di Tiap Genre:") print(top\_movies\_per\_genre) from sklearn.feature\_extraction.text import TfidfVectorizer from sklearn.metrics.pairwise import cosine\_similarity vectorizer\_title = TfidfVectorizer(ngram\_range=(1,2))

tfidf\_title = vectorizer\_title.fit\_transform(movies\_data['title'])

|  |
| --- |
| def search\_by\_title(title):  title = clean\_title(title) query\_vec = vectorizer\_title.transform([title]) similarity = cosine\_similarity(query\_vec, tfidf\_title).flatten() indices = np.argpartition(similarity, -5)[-5:] results = movies\_data.iloc[indices][::-1] return results |
| movie\_results = search\_by\_title("Interstellar") print(movie\_results) def search\_by\_title(title):  title = clean\_title(title) query\_vec = vectorizer\_title.transform([title])  similarity = cosine\_similarity(query\_vec, tfidf\_title).flatten() indices = np.argpartition(similarity, -5)[-5:] results = movies\_data.iloc[indices][::-1] return results  movie\_results = search\_by\_title("Fast and Furious") print(movie\_results)  vectorizer\_genres = TfidfVectorizer(ngram\_range=(1,2))  # Gabungkan genre list menjadi string movies\_data['genres\_text'] = movies\_data['genres'].apply(lambda x: ' '.join(x)) |

tfidf\_genres = vectorizer\_genres.fit\_transform(movies\_data['genres\_text'])

|  |
| --- |
| def search\_similar\_genres(genres):  query\_vec = vectorizer\_genres.transform([genres]) similarity = cosine\_similarity(query\_vec, tfidf\_genres).flatten() indices = np.argpartition(similarity, -10)[-10:] |

results = movies\_data.iloc[indices][::-1] return results

|  |
| --- |
| gen = 'Adventure Action' print(search\_similar\_genres(gen)) def scores\_calculator(movie\_id): # Filter data untuk pengguna serupa similar\_users = combined\_data.loc[  (combined\_data['movieId'] == movie\_id) & (combined\_data['rating'] >= 4), 'userId'  ].unique()  # Dapatkan rekomendasi berdasarkan pengguna serupa similar\_user\_recs = combined\_data.loc[  (combined\_data['userId'].isin(similar\_users)) & (combined\_data['rating'] >= 4),  'movieId'  ].value\_counts(normalize=True) |

|  |
| --- |
| # Dapatkan rekomendasi berdasarkan semua pengguna all\_user\_recs = combined\_data.loc[ combined\_data['movieId'].isin(similar\_user\_recs.index) & (combined\_data['rating'] >= 4  ] all\_user\_recs = all\_user\_recs['movieId'].value\_counts(normalize=True) |

)

|  |
| --- |
| # Filter genre dari film yang dipilih selected\_genres = combined\_data.loc[combined\_data['movieId'] == movie\_id, 'genres'].iloc[0 if isinstance(selected\_genres, list):  selected\_genres = " ".join(selected\_genres) |

]

# Cari film dengan genre serupa movies\_with\_similar\_genres = search\_similar\_genres(selected\_genres) similar\_genre\_ids = movies\_with\_similar\_genres['movieId']

# Kalikan skor berdasarkan genre serupa similar\_user\_recs.loc[similar\_user\_recs.index.isin(similar\_genre\_ids)] \*= 1.5 all\_user\_recs.loc[all\_user\_recs.index.isin(similar\_genre\_ids)] \*= 0.9

|  |
| --- |
| # Gabungkan skor dan hitung peringkat scores = pd.DataFrame({  'similar': similar\_user\_recs,  'all': all\_user\_recs  }).fillna(0) |

# Hindari pembagian nol

scores['score'] = np.where(scores['all'] > 0, scores['similar'] / scores['all'], 0)

# Urutkan berdasarkan skor tertinggi return scores.sort\_values('score', ascending=False)

|  |
| --- |
| scores\_calculator(3114) def recommendation\_results(user\_input, title=0): # user\_input = clean\_title(user\_input) title\_candidates = search\_by\_title(user\_input) movie\_id = title\_candidates.iloc[title]['movieId'] scores = scores\_calculator(movie\_id) results = scores.head(10).merge(movies\_data, left\_index=True, right\_on='movieId')[['title'  'score', 'genres']] resutls = results.rename(columns={'title': 'title', 'genres': 'genres'}, inplace=True) return results user\_input = "Interstellar" |

,

|  |
| --- |
| print("Here a similar movies: ") for i in range(5):  print(i, ": ", search\_by\_title(user\_input)['title'].iloc[i]) |

title = 0 print("Recommendation\_results: ") print(recommendation\_results(user\_input))

# 14. Future Scope

* Integrate real-time feedback to update user profiles dynamically.
* Use NLP (BERT) on movie plots/tags for deeper content-based filtering.
* Develop a mobile app version with offline capabilities.
* Add support for trailers, ratings explanation (explainable AI).

# 15. Team Members and Roles

|  |  |
| --- | --- |
| **NAME** | **ROLES AND RESPONSIBILITIES** |
| **AJAYKUMAR.S** | ***Project Manager & Data Preprocessing Specialist*** |
| **MANIKANDAN.P** | ***Data collector and developer*** |
| **SHANMUGAPRIYAN.N** | ***Backward Developer and training specialist*** |