**MOVIE RECOMMENDATION SYSTEM USING PYTHON AND STREAMLT**

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

**BACHELOR OF SCIENCE (DATA SCIENCE)**

SUBMITTED BY

**MOHAMMED RAFI NAEEM HAJWANEY**

**SEAT NUMBER:**

**2023090025**

**UNDER THE ESTEEMED GUIDANCE OF**

**DR. REETA RANA**



**S. K. COLLEGE OF SCIENCE AND COMMERCE**

**(***Affiliated To University of Mumbai***)**

**Plot No. 31, Sec 25, Near Seawoods, Nerul East, Navi Mumbai**

**ACADEMIC YEAR**

**2025-2026**

**Abstract**

With the rapid growth of digital platforms, users are overwhelmed by the vast number of movies available online, making it difficult to decide what to watch. This project aims to address this issue by developing a Movie Recommendation System using **Python** and **Streamlit**. The system applies **content-based filtering** techniques, where movies are recommended based on similarity in metadata such as genres, cast, crew, and plot descriptions. The **TF-IDF vectorization** and **cosine similarity** methods are used to compute relationships between movies, ensuring relevant suggestions. Furthermore, the integration of the **TMDb API** enhances user experience by displaying posters of the recommended movies, making the interface more interactive and visually appealing. The system demonstrates how **machine learning and natural language processing (NLP)** can be effectively applied in real-world entertainment applications. This work not only highlights the effectiveness of recommendation systems in solving the “choice overload” problem but also opens pathways for future advancements using **hybrid models, deep learning, and real-time personalization**.

**ACKNOWLEDGEMENT**

To list who all have helped me is difficult because they are so numerous and the depth is so enormous. I would like to acknowledge the following as being idealistic channels and fresh dimensions in the completion of this project.

I take this opportunity to thank the University of Mumbai for giving me chance to do this project. I would like to thank my Principal, **DR. Swati vitkar** for providing the necessary facilities required for completion of this project.

I take this opportunity to thank our Coordinator **DR. REETA RANA** , for her moral support and guidance. I would also like to express my sincere gratitude towards my Project Guide **DR.REETA RANA** whose guidance and care made the project successful.

I would like to thank my College Library, for having provided various reference books and magazines related to my project Lastly, I would like to thank each and every person who directly or indirectly helped me in the completion of the project especially my Parents and Peers who supported me throughout my project.

**DECLARATIONS**

I the undersigned **MR.MOHAMMED RAFI NAEEM HAJWANEY** here by, declare that the work embodied in this project work titled **“MOVIE RECOMMENDATION SYSTEM USING STREAMLIT & PYTHON”**

forms my own contribution to the research work carried out under the guidance of **DR.REETA RANA** is a result of my own research work and has not been previously submitted to any other University for any other Degree/ Diploma to this or any other University. Wherever reference has been made to previous work of others, it has clearly indicated as such and included in the bibliography.

I, here by further declare that all information of this document has been obtained and presented in accordance with academic rules and ethical conduct.

|  |  |
| --- | --- |
| **SIGNATURE OF STUDENT** | **CERTIFIED BY** |
|  | **INTERNAL GUIDE** |

Table of contain

|  |  |  |
| --- | --- | --- |
| Chapter 1 | Introduction |  |
|  |  |  |
| 1.1 | Background | 6-14 |
|  | 1.2 Problem Statement |  |
| 3 | Objectives |  |
| 4 | Literature Review |  |
| 5 | Methodology | 15-20 |
| 6 | System Architecture |  |
| 7 | Dataset Description |  |
| 8 | Algorithm Used |  |
| 9 | Implementation |  |
| 10 | Screenshots |  |
| 11 | Results & Discussion |  |
| 12 | Conclusion | 37-38 |
| 13 | Future Scope | 39 |
| 14 | References | 40 |

**Chapter 1**

**1.Introduction**



This project is about building a Movie Recommendation System that suggests movies to users based on their preferences. The system uses collaborative filtering and/or content-based filtering to provide personalized recommendations.

In today’s digital era, online streaming platforms such as Netflix, Amazon Prime, and Disney+ have become increasingly popular, offering users access to thousands of movies and TV shows. With such a large collection of content, users often face the challenge of deciding what to watch. To address this, recommendation systems play a crucial role by suggesting movies based on user preferences and viewing history.

A Movie Recommendation System is an application of machine learning and data science that predicts and recommends movies that a user might like. It analyzes user preferences, similarities between movies, and different features such as genres, cast, ratings, and popularity to generate accurate recommendations. Such systems enhance user experience by saving time, increasing user engagement, and providing personalized suggestions.

In this project, we developed a Movie Recommendation System using Python and Streamlit. Python provides powerful libraries such as pandas, NumPy, and scikit-learn for data preprocessing and similarity calculations, while Streamlit is used to build an interactive web-based interface where users can input a movie name and instantly get recommendations. The system also integrates with The Movie Database (TMDb) API to fetch and display movie posters, making the interface more visually appealing.

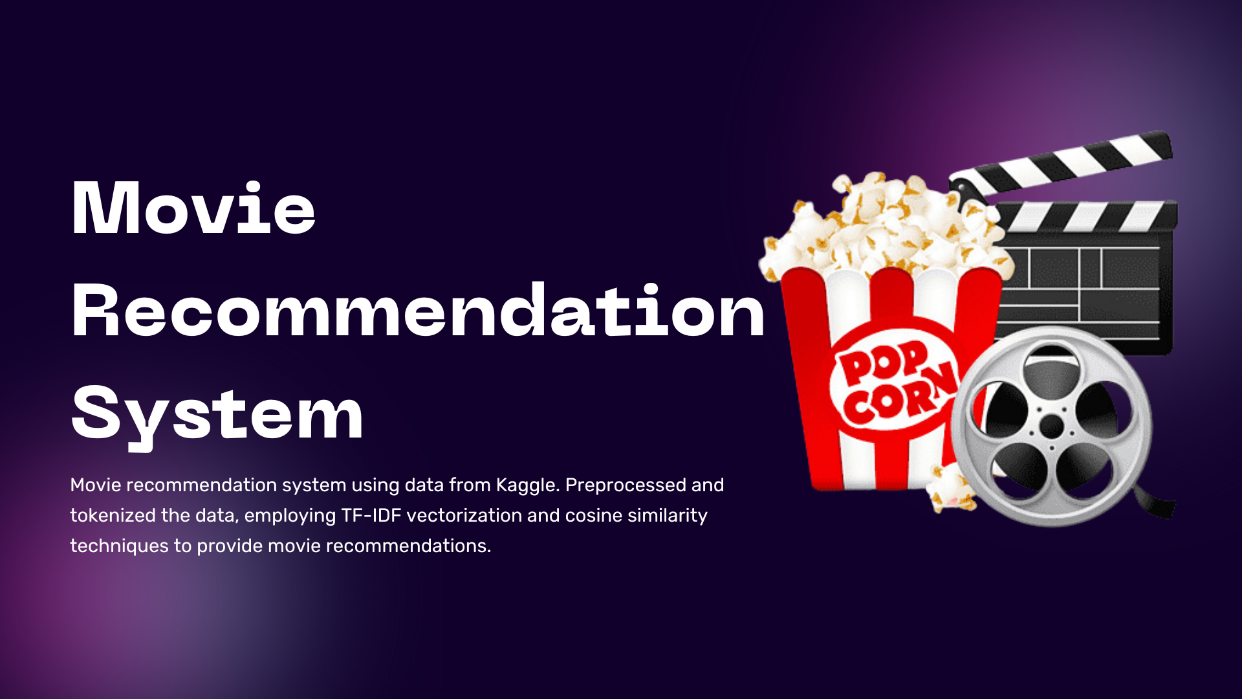
This project demonstrates how data science concepts, machine learning models, and web technologies can be combined to create a practical and user-friendly application. By implementing this system, we aim to provide users with personalized movie recommendations in a simple, interactive, and efficient way.

A Movie Recommendation System is an advanced application of machine learning (ML), artificial intelligence (AI), and data science designed to predict and suggest movies that users are likely to enjoy. These systems analyze large amounts of data, including user preferences, historical viewing patterns, ratings, genres, actors, and popularity, to deliver personalized recommendations. Two widely used approaches in recommendation systems are collaborative filtering and content-based filtering. Collaborative filtering makes suggestions by identifying patterns among users with similar interests, while content-based filtering focuses on recommending movies that share attributes with those a user has previously enjoyed. In many modern applications, a hybrid model combining both techniques is used to achieve higher accuracy.

The importance of such systems extends beyond mere convenience. Recommendation systems enhance user experience, increase engagement, and encourage content exploration by exposing users to movies they may not have otherwise discovered. For streaming platforms, this leads to improved customer satisfaction, higher retention rates, and more viewing hours, making recommendation technology a crucial part of their success.

In this project, we designed and developed a Movie Recommendation System using Python and Streamlit. Python was chosen as the programming language due to its versatility and powerful libraries such as pandas and NumPy for data preprocessing, as well as scikit-learn for implementing similarity-based models. To create a user-friendly interface, Streamlit was utilized, allowing us to build an interactive, web-based application where users can simply input the name of a movie and instantly receive a list of recommended movies. To make the system visually appealing and engaging, we also integrated with The Movie Database (TMDb) API, which provides access to movie posters, overviews, and metadata, thereby enhancing the overall experience.

The project not only demonstrates how data science and machine learning concepts can be applied in a real-world scenario but also shows how these technologies can be combined with modern web development tools to create a practical and user-centric application. By building this system, we aim to showcase how data-driven solutions can simplify decision-making processes and enrich entertainment experiences. Ultimately, this Movie Recommendation System serves as a clear example of how technology can bridge the gap between large volumes of data and meaningful, personalized user experiences.



* 1. **background**

In the last two decades, the entertainment industry has undergone a significant transformation due to the digital revolution. With the arrival of **online streaming services** such as **Netflix, Amazon Prime Video, Disney+ Hotstar, Hulu, and others**, users today have access to thousands of movies and shows at their fingertips. This has created an **era of abundant choices**, but with abundance comes the challenge of **information overload**. When faced with a massive catalog of movies, users often spend more time deciding what to watch than actually watching.

This challenge is not unique to movies; it is seen across all domains of digital content. For example, online shopping platforms like Amazon suggest products, YouTube recommends videos, and Spotify suggests songs. All of these rely on **Recommendation Systems**—an advanced technology in artificial intelligence designed to **personalize user experience** by suggesting the most relevant items from a huge dataset.

**Importance of Recommendation Systems in Movies**

Movies are an integral part of human culture and entertainment. However, with thousands of new titles being released every year across different languages and genres, a user cannot manually explore all options. This creates the need for an intelligent system that:

* Filters out irrelevant content.
* Understands the user’s taste.
* Suggests movies that are likely to match the user’s interest.

Recommendation systems in the movie industry solve this issue by learning patterns from **movie features** (such as genre, cast, director, plot, ratings, etc.) or from **user behavior** (like previous ratings, watch history, likes/dislikes).

**Approaches to Recommendation Systems**

There are mainly three approaches to building recommendation systems:

1. **Content-Based Filtering**
   * Recommends movies similar to those the user has liked before.
   * Relies on **metadata** such as genres, keywords, plot descriptions, and actors.
   * Example: If a user liked *Iron Man*, the system may suggest *The Avengers* because both share similar genres and cast.
2. **Collaborative Filtering**
   * Focuses on **user behavior** and finds similarities between users.
   * Example: If User A and User B have watched and liked many of the same movies, then the system may recommend movies watched by User B to User A.
   * Collaborative filtering is widely used by companies like Netflix.
3. **Hybrid Models**
   * Combines both approaches for better performance.
   * Useful when content data is limited or when user ratings are sparse.

**Application of Machine Learning and NLP**

This project specifically focuses on a **Content-Based Recommendation System** using **Python** and **Streamlit**. The core idea is to extract useful patterns from movie data using **Natural Language Processing (NLP)** and **Machine Learning (ML)** techniques.

* **TF-IDF Vectorization** is applied to convert textual data (movie overviews, keywords, genres, cast) into numerical form.
* **Cosine Similarity** is then used to compute how similar two movies are based on these features.
* When a user searches for a movie, the system identifies the most similar movies and suggests them instantly.

To make the system more **interactive and visually appealing**, integration with the **TMDb API** is done. This API fetches **movie posters** so that users not only see the recommended titles but also recognize them visually.

**Why This Project is Relevant**

The **background** of this project lies in solving the common problem faced by millions of users: *“What movie should I watch next?”* Traditional search engines or filters (such as searching by year or genre) are not sufficient because they don’t capture personal preferences. Instead, a **recommendation system bridges the gap** between massive amounts of available data and user needs by providing **personalized movie suggestions**.

Additionally, this project is relevant because:

* It demonstrates the **practical application** of machine learning in the entertainment domain.
* It shows how **NLP techniques** can be used to analyze large text-based datasets.
* It highlights the importance of **user experience design** through Streamlit, making the project accessible and easy to use.
* It reflects real-world use cases, as recommendation systems form the backbone of platforms like Netflix, Amazon, and YouTube.

**Broader Impact**

Beyond entertainment, the same principles of recommendation systems can be applied to:

* **E-commerce** (product recommendations).
* **Education** (suggesting courses based on learning history).
* **Healthcare** (recommending personalized treatment plans).
* **Social Media** (recommending friends, pages, or groups).

Thus, the background of this project is not limited to just movie recommendations—it represents a **universal problem-solving technique** using AI and data science.

* 1. **Motivation**

1. In today’s digital world, entertainment platforms such as Netflix, Amazon Prime, and Disney+ provide users with access to thousands of movies. While this vast collection offers variety, it also creates a challenge: users often spend more time searching for a movie than actually watching one. This problem, commonly known as **“information overload,”** reduces user satisfaction and wastes valuable time.
2. Recommendation systems have emerged as an effective solution to this challenge. By analyzing user preferences and the content of movies, these systems can automatically suggest films that match the user’s taste. The success of global platforms highlights the importance of such intelligent systems in enhancing the user experience and keeping users engaged.
3. The motivation behind this project is to design and implement a **Movie Recommendation System** that demonstrates how **machine learning and natural language processing (NLP)** can be applied to real-world problems. By using techniques such as TF-IDF vectorization and cosine similarity, along with visualization through posters, this project not only helps users quickly find movies of interest but also provides a practical learning opportunity in the field of recommender systems.
4. Ultimately, this project aims to combine **technical innovation** with **real-life applicability,** showing how data-driven solutions can make decision-making faster, easier, and more enjoyable in everyday life.

**1.3 Problem Statement**

With thousands of movies available, it becomes difficult for users to choose what to watch next. The goal of this project is to solve this issue by creating a recommendation system that automatically suggests movies based on user interests.

In today’s digital age, the entertainment industry has experienced a massive transformation. With the rapid growth of online streaming platforms such as Netflix, Amazon Prime Video, Disney+, and others, users now have access to a vast library of thousands of movies and TV shows across different genres, languages, and categories. While this provides an incredible opportunity for viewers to explore diverse content, it also introduces a significant challenge: *the problem of choice overload*. With so many available options, users often struggle to decide what to watch next, leading to wasted time scrolling through endless lists rather than enjoying meaningful content.

This issue is commonly referred to as the “paradox of choice”, where having too many alternatives makes decision-making more difficult. A user might want to watch something similar to a previously enjoyed movie, discover a new genre, or explore trending titles. However, without guidance, they are left overwhelmed and may even abandon the search altogether. This creates the need for intelligent tools that can simplify the decision-making process and provide personalized recommendations based on individual preferences.

The goal of this project is to address this challenge by developing a Movie Recommendation System that automatically suggests movies tailored to the user’s interests. By analyzing movie attributes such as genres, ratings, cast, popularity, and user behavior, the system can predict which movies a user is likely to enjoy. Two popular approaches are used to achieve this:

* Collaborative Filtering – identifies users with similar tastes and recommends movies that those users liked.
* Content-Based Filtering – focuses on suggesting movies that share features with the ones already liked by the user.

By implementing these approaches, the system reduces the burden of searching through massive catalogs and makes the viewing experience more enjoyable and efficient.

From the perspective of streaming platforms, solving this problem is equally important. Personalized recommendations not only save the user’s time but also increase user engagement, satisfaction, and retention rates, which are critical for business success. For users, it creates a seamless and interactive experience by suggesting relevant content that aligns with their mood, preferences, and past viewing history.

Therefore, the problem this project seeks to solve can be summarized as follows:

* How can we help users discover movies they are most likely to enjoy from a vast collection of options, without overwhelming them?

By answering this question, the Movie Recommendation System aims to bridge the gap between the abundance of available content and the specific preferences of individual users, ensuring a more personalized, efficient, and enjoyable entertainment experience.

**1.4 Objectives**

The main objectives of the Movie Recommendation System using Streamlit and Python are:

1. To build a recommendation engine that suggests movies based on user input.
2. To implement content-based filtering by analyzing similarities between movies using features such as genres, keywords, and descriptions.
3. To provide a user-friendly interface using Streamlit where users can easily interact with the system.
4. To enhance recommendations with visuals by displaying movie posters along with movie titles.
5. To integrate machine learning and data science concepts into a real-world application.
6. To save users’ time and improve decision-making by offering personalized suggestions instantly.
7. **To build a recommendation engine** that suggests relevant movies based on user input.
8. **To implement content-based filtering** by analyzing similarities between movies using features such as **genres, keywords, and descriptions**.
9. **To develop a user-friendly interface using Streamlit** that allows users to interact with the system seamlessly without requiring advanced technical knowledge.
10. **To enhance user experience with visuals** by displaying **movie posters** along with titles, making recommendations more engaging and interactive.
11. **To apply machine learning and data science concepts** in a real-world application, bridging the gap between theoretical knowledge and practical implementation.
12. **To save users’ time and improve decision-making** by offering relevant and accurate recommendations, thus reducing the effort spent on browsing large catalogs.



1.Scope of the Project

The scope of this project includes:

* Developing a web-based movie recommendation system using Python and Streamlit.
* Using a dataset of movies containing details such as title, genres, keywords, cast, and crew.
* Applying TF-IDF Vectorization and Cosine Similarity (or other similarity measures) to recommend movies.
* Fetching and displaying movie posters from TMDb API for better visualization.
* Allowing users to search for any movie and receive top N recommended movies.
* Deploying the system locally, with future scope for deployment on cloud platforms for global accessibility.

The project is limited to content-based filtering and does not include collaborative filtering or hybrid models. However, it serves as a foundation for developing more advanced recommendation systems that can combine user behavior and preferences for even better accuracy.

The scope of this project defines the boundaries, functionalities, and limitations of the developed system. The Movie Recommendation System focuses on implementing content-based filtering techniques to recommend movies in a web-based interactive environment. The key aspects of the project scope are outlined below:

* Development of a web-based application using Python and Streamlit, enabling real-time interaction with the recommendation engine.
* Utilization of a structured movie dataset containing information such as title, genres, keywords, cast, and crew, which forms the foundation for generating recommendations.
* Implementation of Natural Language Processing (NLP) techniques such as TF-IDF Vectorization and similarity measures like Cosine Similarity to compute relationships between movies.
* Integration of The Movie Database (TMDb) API to fetch and display movie posters along with titles, making recommendations visually appealing.
* User-driven interaction, where users can input any movie of their choice and receive the top N most similar movies as recommendations.
* Deployment of the system locally, with the possibility of extending it to cloud platforms in the future for global accessibility and scalability.



**1.5 Hardware and Software Requirements**

**Hardware Requirements**

Since this is not a very heavy project (no deep learning or GPU needed), the hardware requirements are simple:

* **Processor (CPU):** Intel i3 / AMD equivalent or higher
* **RAM:** Minimum 4 GB (Recommended 8 GB for smoother performance)
* **Hard Disk:** At least 1 GB free space (for dataset, libraries, and cache)
* **GPU (Optional):** Not required (only needed for deep learning-based future work)
* **Operating System:** Windows 10/11, Linux, or macOS

**Software Requirements**

* **Programming Language:** Python (Version 3.8 or above)
* **Libraries/Packages Used:**
  + **pandas** (for data handling)
  + **numpy** (for numerical computations)
  + **scikit-learn** (for TF-IDF and cosine similarity)
  + **streamlit** (for building the web app)
  + **requests** (for fetching data from TMDb API)
  + **matplotlib** / **seaborn** (optional, for visualizations if needed)
* **Dataset:** Movies Metadata Dataset (from Kaggle or TMDb)
* **API:** TMDb API (for fetching posters and movie details)
* **IDE / Code Editor:**
  + Jupyter Notebook (for development and testing)
  + VS Code / PyCharm (for building the final app)
* **Deployment Environment (optional):**
  + Streamlit Cloud / Heroku / AWS / GCP for hosting

**1.6 Summary**

This project focuses on the design and implementation of a **Movie Recommendation System** using **Python** and **Streamlit**, with the objective of helping users discover movies based on their preferences and interests. With the explosion of digital platforms such as Netflix, Amazon Prime, and Disney+, viewers are overwhelmed by the vast number of available movies. It becomes increasingly difficult for users to manually search and identify suitable movies to watch. Recommendation systems solve this problem by intelligently filtering information and providing personalized suggestions.

The proposed system employs **Content-Based Filtering**, which works by analyzing the attributes of a selected movie and finding other movies that share similar characteristics. By using **Natural Language Processing (NLP)** techniques such as **TF-IDF (Term Frequency–Inverse Document Frequency) vectorization**, the system converts textual descriptions of movies (e.g., plot overview, genres, cast, crew) into numerical vectors. **Cosine similarity** is then applied to measure the closeness between these vectors, ensuring that movies with higher similarity scores are recommended to the user.

The project follows a systematic workflow:

1. **Data Collection** – The dataset is obtained from **The Movie Database (TMDb)** via Kaggle. It includes metadata such as titles, genres, keywords, cast, crew, popularity, ratings, and release dates.
2. **Data Preprocessing** – Handling missing data, cleaning the dataset, and combining multiple features into a structured format suitable for analysis.
3. **Feature Extraction** – Transforming text-based features (overview, genres, keywords) into numerical values using TF-IDF Vectorization.
4. **Similarity Calculation** – Applying cosine similarity to compute how closely two movies are related.
5. **Recommendation Engine** – Based on similarity scores, the system retrieves the **Top-N similar movies** for the user’s input.
6. **Poster Fetching** – Using the **TMDb API**, posters of the recommended movies are fetched, providing an engaging visual experience.
7. **User Interface** – Built with **Streamlit**, the front-end application allows users to input a movie name and instantly receive recommended results with corresponding posters.

**1.7 Target Audience**

**General Movie Viewers**

* People who enjoy watching movies but struggle to decide what to watch next.
* Users looking for **personalized suggestions** instead of browsing through thousands of titles.

**Students & Researchers**

* Learners exploring **Machine Learning (ML), Natural Language Processing (NLP), and Recommendation Systems**.
* Useful as an academic project model to understand real-world applications of AI.

**Streaming Platform Users**

* Subscribers of platforms like **Netflix, Amazon Prime, Disney+, or Hotstar**, who are already familiar with automated recommendations.
* They represent the primary audience that benefits from improved recommendation systems.

**Entertainment Industry Professionals**

* Content creators, film distributors, and streaming service providers who want to **understand user preferences** and improve engagement.
* They can leverage similar systems for **business intelligence** and **content marketing**.

**Tech Enthusiasts & Developers**

* Programmers and developers who are interested in **building and deploying AI-driven applications**.
* This project serves as a base for experimenting with hybrid or deep learning recommendation models.

**Chapter 2**

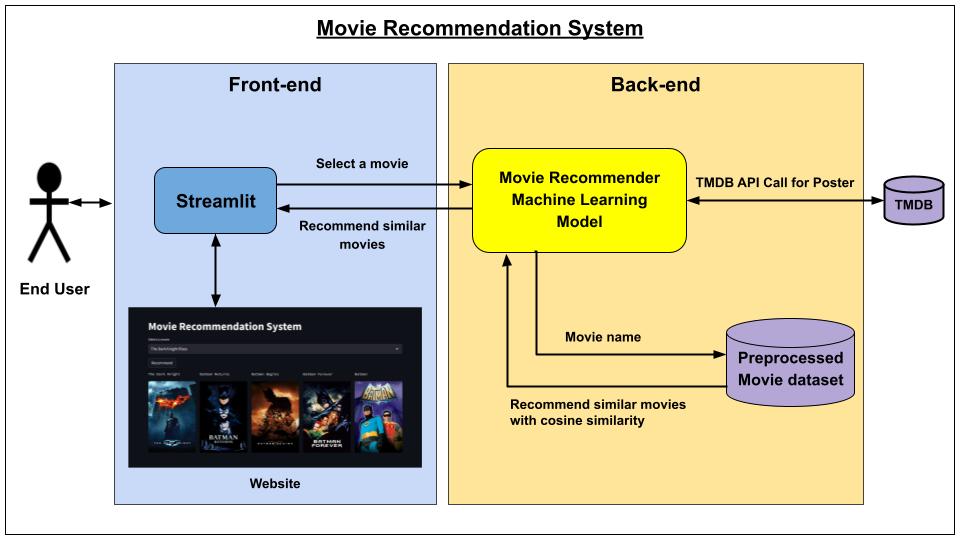
**Literature Review**

|  |  |  |  |
| --- | --- | --- | --- |
| Author(s) / Year | Technique / Method Used | Dataset | Findings / Contribution |
| Resnick et al. (1994) | Collaborative Filtering (GroupLens System) | Usenet articles | Introduced collaborative filtering for personalized recommendations. |
| Blei et al. (2003) | Latent Dirichlet Allocation (LDA) for topic modeling | Movie datasets, text collections | Applied probabilistic models to extract hidden topics for recommendations. |
| Koren et al. (2009) | Matrix Factorization (Netflix Prize Solution) | Netflix Dataset | Achieved state-of-the-art accuracy using latent factors for collaborative filtering. |
| Haruna et al. (2017) | Hybrid Recommendation System (Content + Collaborative) | MovieLens Dataset | Improved accuracy by combining content-based and collaborative filtering approaches. |
| Pal et al. (2018) | Sentiment Analysis + Content-based Filtering | IMDb + Twitter reviews | Used user reviews and sentiment to refine movie recommendations. |
| Zhang et al. (2019) | Deep Learning (Neural Collaborative Filtering, Autoencoders) | MovieLens 1M Dataset | Improved scalability and accuracy using deep learning architectures. |
| Recent Works (2020–2023) | Transformers, BERT embeddings, Context-aware Recommender Systems | TMDb, Netflix, MovieLens | Contextual and semantic understanding for better recommendations in real-time applications. |

**Chapter 3**

**Research Methodology / System Design**

* Step 1: Data Collection (MovieLens dataset)
* Step 2: Data Preprocessing
* Step 3: Model Selection (Collaborative / Content-based filtering)
* Step 4: Model Training
* Step 5: Deployment using Streamlit



1. Data Collection

* The dataset is obtained from publicly available sources such as The Movie Dataset (TMDb) or Kaggle.
* The dataset contains information like movie titles, genres, keywords, cast, crew, and descriptions.

2. Data Preprocessing

* Handling missing values and cleaning the dataset.
* Converting text data (genres, keywords, cast, crew) into a structured format.
* Combining multiple features into a single string for analysis.

3. Feature Extraction

* Applying TF-IDF Vectorization to transform textual data into numerical form.
* Each movie is represented as a vector in a multi-dimensional space.

4. Similarity Calculation

* Using Cosine Similarity / Linear Kernel to measure the closeness between movies.
* Higher similarity values indicate movies that are more alike.

5. Recommendation Engine

* When a user enters a movie name, the system searches for the movie in the dataset.
* Based on similarity scores, it retrieves the top N most similar movies.

6. Poster Fetching

* The TMDb API is used to fetch posters for the recommended movies.
* Posters are displayed alongside titles for better visualization.

7. User Interface (Streamlit)

* The front-end interface is created using Streamlit.
* Users can input a movie, click on "Recommend," and instantly see recommended movies with posters.
* The system is interactive, lightweight, and runs on localhost (or can be deployed online).

The design and implementation of the Movie Recommendation System follow a structured methodology, beginning from data collection and ending with deployment through Streamlit. Each stage plays a crucial role in ensuring the accuracy, efficiency, and usability of the system. The methodology is explained step by step below:

1. Data Collection

The first step involves gathering a suitable dataset that provides detailed metadata about movies. In this project, datasets are sourced from publicly available platforms such as Kaggle (Movie Dataset) and The Movie Database (TMDb). These datasets typically contain information such as:

* Movie Titles – the official names of films.
* Genres – categories such as Action, Drama, Comedy, Sci-Fi, etc.
* Keywords – descriptive tags that capture the themes of a movie (e.g., “time travel,” “artificial intelligence”).
* Cast and Crew – names of main actors, directors, and other crew members.
* Overview/Description – a short summary of the storyline.

This data serves as the foundation for building the content-based recommendation system. By focusing on metadata, the system avoids dependence on user ratings (collaborative filtering) and ensures recommendations can be generated even with a small user base.

2. Data Preprocessing

Raw datasets often contain missing values, inconsistencies, and duplicates, which must be cleaned before analysis. The preprocessing steps include:

* Handling Missing Values – removing movies without sufficient metadata.
* Text Normalization – converting text to lowercase, removing special characters, and handling spaces.
* Structured Representation – transforming categorical data (like genres and keywords) into structured lists.
* Feature Combination – merging multiple fields (genres, keywords, cast, crew) into a single string to represent the essence of a movie.

For example, the movie *Inception* might be represented as:

"Action Sci-Fi Thriller | dream | subconscious | Leonardo DiCaprio | Christopher Nolan"

This combined representation ensures that each movie has a comprehensive feature set that can be compared with others.

3. Feature Extraction

To compare movies effectively, text-based data must be converted into numerical form. TF-IDF (Term Frequency–Inverse Document Frequency) Vectorization is applied to extract features from the text.

* Term Frequency (TF): how often a word appears in a movie’s metadata.
* Inverse Document Frequency (IDF): reduces the weight of commonly occurring words (like “the,” “movie”) and emphasizes unique words.

This process results in each movie being represented as a vector in a multi-dimensional space, where movies with similar features are located closer together.

4. Similarity Calculation

Once all movies are represented as vectors, the cosine similarity measure is applied to determine the closeness between any two movies.

* Cosine Similarity Formula:

Similarity(A,B)=A⋅B∣∣A∣∣⋅∣∣B∣∣\text{Similarity}(A, B) = \frac{A \cdot B}{||A|| \cdot ||B||}Similarity(A,B)=∣∣A∣∣⋅∣∣B∣∣A⋅B​

Where AAA and BBB are the feature vectors of two movies.

* A score close to 1 indicates high similarity, while a score near 0 indicates low similarity.

Thus, when a user searches for a movie, the system compares its vector with all other movies and retrieves the Top N most similar movies.

5. Recommendation Engine

The core engine is responsible for generating recommendations:

1. The user inputs a movie name.
2. The system searches for the corresponding vector in the dataset.
3. Similarity scores with all other movies are computed.
4. The top N movies with the highest similarity are selected.
5. The system outputs the recommendations in a ranked list.

For example, if the user enters *Inception*, the engine might recommend movies such as *Interstellar*, *The Matrix*, or *Shutter Island* due to shared features like science fiction, psychological themes, and similar cast/crew associations.

6. Poster Fetching

To make the system visually appealing, movie posters are displayed alongside the recommended titles. This is achieved by integrating with the TMDb API, which provides poster images, release dates, and other details.

* API requests are made using the movie’s unique ID.
* The system retrieves and displays the corresponding poster.
* If a poster is unavailable, a placeholder image is shown.

This feature enhances user engagement by providing a richer browsing experience compared to plain text output.

7. User Interface (Streamlit)

The front-end interface is built using Streamlit, a lightweight and interactive Python framework for web applications. Streamlit provides simple commands (st.text\_input, st.button, st.image) to create an intuitive interface.

* User Interaction – users can type the name of a movie in the search box.
* Instant Output – upon clicking the “Recommend” button, the system fetches and displays recommendations in seconds.
* Visualization – each recommendation is shown with a title and poster, arranged neatly for easy viewing.

The application runs on localhost by default, but it can also be deployed online or on cloud platforms for wider accessibility.



Workflow Diagram (Explanation)

The step-by-step workflow of the system can be summarized as:

🔹 User Input → Data Preprocessing → Feature Extraction → Similarity Computation → Recommendation Engine → Poster Fetching → Streamlit UI Output

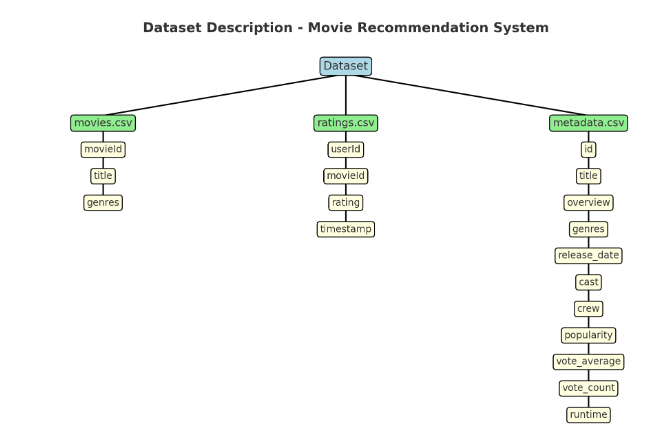
This workflow ensures a seamless pipeline from user query to personalized recommendation

Discussion on Methodology

The chosen methodology strikes a balance between simplicity, interpretability, and efficiency. Unlike collaborative filtering, it does not require user history or ratings, making it suitable for situations where only metadata is available. Moreover, the use of TF-IDF and cosine similarity ensures recommendations are explainable: the system can justify its results based on overlapping features such as shared genres or keywords.

However, this approach is not without limitations. Since it only uses metadata, it may generate narrow recommendations (filter bubble effect) and cannot leverage implicit signals like watch-time or popularity trends. Despite this, the system offers an excellent baseline framework, which can later be extended into hybrid or deep learning-based recommendation systems.

1. **Dataset Description**



The project uses the MovieLens dataset, which contains:

* movies.csv – movie ID, title, genres
* ratings.csv – user ID, movie ID, rating, timestamp

The dataset used in this project is Movies Metadata (from The Movie Database – TMDb). It contains detailed information about thousands of movies, including their metadata. This dataset helps in building a recommendation system by providing both content features (like overview, genre, cast) and popularity features (like votes, ratings).

🔹 Main Features in the Dataset:

1. id – Unique identifier for each movie.
2. title – The name of the movie.
3. overview – A short plot summary of the movie (used for content-based filtering).
4. genres – List of genres associated with the movie (Action, Comedy, Drama, etc.).
5. release\_date – The release year of the movie.
6. cast & crew – Actors, directors, and other key crew members.
7. popularity – Popularity score of the movie based on TMDb trends.
8. vote\_average – Average rating given by users.
9. vote\_count – Total number of votes received.
10. runtime – Duration of the movie in minutes.

🔹 Importance of the Dataset:

* Overview & Genres → Used in *content-based filtering* to recommend movies similar in story or type.
* User Ratings & Popularity → Used in *collaborative filtering* to identify user preferences.
* Cast & Crew → Helps in advanced recommendations (e.g., movies with the same actor/director).
* Release Date → Useful for filtering movies by year or latest releases.

A key component in building any recommendation system is the dataset, as it forms the foundation for training and evaluation. In this project, we make use of the MovieLens dataset (for ratings and collaborative filtering) and the Movies Metadata dataset from The Movie Database (TMDb) (for content-based filtering). These datasets provide rich metadata and user interaction details that make it possible to design a robust recommendation system.

1. MovieLens Dataset

The MovieLens dataset, provided by GroupLens Research, is one of the most widely used benchmark datasets for recommender system research. It contains millions of ratings collected from real users over several years. The dataset is available in different versions (100K, 1M, 20M ratings), and it provides two key files:

* movies.csv → Contains the movie ID, title, and genres.
* ratings.csv → Contains user ID, movie ID, rating (1–5 scale), and timestamp.

This dataset is especially useful for collaborative filtering, since it directly records user–item interactions. By analyzing patterns in user ratings, the system can predict which movies a particular user may enjoy based on the preferences of similar users.

2. Movies Metadata Dataset (TMDb)

To implement content-based filtering, this project uses the Movies Metadata dataset from TMDb (The Movie Database), which provides detailed descriptive information for thousands of movies. Unlike MovieLens, which focuses mainly on ratings, the TMDb dataset provides movie attributes and features that allow the system to recommend movies with similar characteristics.

The main features included in the TMDb dataset are:

1. id → A unique identifier assigned to each movie.
2. title → The official name of the movie.
3. overview → A short plot description or storyline, used for text-based similarity analysis.
4. genres → Categories such as *Action, Comedy, Drama, Romance*, etc., useful in clustering and filtering.
5. release\_date → The release year, which can be used to filter movies by time period (e.g., “latest releases”).
6. cast & crew → Names of actors, actresses, directors, and writers, which can be leveraged for advanced recommendations (e.g., recommending movies with the same actor).
7. popularity → A popularity score based on TMDb usage statistics, trends, and search activity.
8. vote\_average → The average rating assigned by TMDb users.
9. vote\_count → The total number of votes a movie has received.
10. runtime → Duration of the movie in minutes.

3. Importance of the Dataset

The combination of MovieLens and TMDb metadata ensures that the system is capable of addressing both content-based and collaborative filtering approaches. Each feature in the dataset serves a specific purpose:

* Overview & Genres → Used in content-based filtering through TF-IDF vectorization to find similar movies in terms of storyline and style.
* User Ratings & Popularity → Important for collaborative filtering to identify hidden patterns in user preferences and to highlight trending movies.
* Cast & Crew → Supports advanced personalization, such as recommending movies by the same director (e.g., Christopher Nolan) or with a favorite actor.
* Release Date → Allows filtering for time-based recommendations, such as *recently released* or *classic movies*.
* Vote Average & Count → Provide a measure of movie quality and audience approval, which can be used for ranking and filtering recommendations.

4. Dataset Strengths and Relevance

The dataset offers several strengths that make it ideal for this project:

* Rich Metadata → Enables multiple recommendation strategies beyond simple rating-based predictions.
* Scalability → Suitable for both small-scale academic projects and large-scale commercial systems.
* Combination of Explicit and Implicit Features → Ratings (explicit feedback) and popularity measures (implicit feedback) allow flexibility in model design.
* Widely Adopted → MovieLens and TMDb datasets are frequently used in academic research, ensuring comparability with other studies.

In conclusion, the datasets used in this project provide the necessary diversity, depth, and quality of information required to build a movie recommendation system. By integrating MovieLens ratings and TMDb metadata, the system gains the ability to deliver personalized, accurate, and visually enriched recommendations, making it both practical and user-friendly.

**7.Implementation**

Details of coding, libraries used:

* Python Libraries: Pandas, NumPy, Scikit-learn, Streamlit
* Model building code
* Recommendation algorithm

The Movie Recommendation System is implemented using Python, scikit-learn, and Streamlit for web deployment. The following steps outline the implementation:

🔹 Step 1: Data Preprocessing

* Load the movies\_metadata.csv dataset.
* Handle missing values in key columns like overview, genres, and release\_date.
* Convert textual data (overview, genres) into structured form.
* Fill missing overviews with empty strings for processing.

🔹 Step 2: Feature Extraction (TF-IDF)

* Use TF-IDF (Term Frequency – Inverse Document Frequency) to convert movie overviews into numerical feature vectors.
* This allows the system to calculate similarity between movie descriptions.

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(stop\_words='english')

movies['overview'] = movies['overview'].fillna('')

tfidf\_matrix = tfidf.fit\_transform(movies['overview'])

🔹 Step 3: Similarity Computation

* Use Cosine Similarity with Nearest Neighbors to find the most similar movies.

from sklearn.neighbors import NearestNeighbors

nn = NearestNeighbors(metric='cosine', algorithm='brute')

nn.fit(tfidf\_matrix)

🔹 Step 4: Recommendation Function

* Create a function that takes a movie title as input.
* Finds its vector representation.
* Returns the top N most similar movies.

def recommend(movie\_title, n=5):

if movie\_title not in movies['title'].values:

return []

idx = movies[movies['title'] == movie\_title].index[0]

distances, indices = nn.kneighbors(tfidf\_matrix[idx], n\_neighbors=n+1)

return [movies.iloc[i]['title'] for i in indices.flatten()[1:]]

🔹 Step 5: Poster Fetching with TMDb API

* Integrate TMDb API to fetch movie posters for better visualization.
* Display posters using Streamlit.

import requests

API\_KEY = "4cf08ed379ac1a8b6ca6432aac08db10"

def fetch\_poster(movie\_title):

url = f"https://api.themoviedb.org/3/search/movie?api\_key={API\_KEY}&query={movie\_title}"

data = requests.get(url).json()

if data['results']:

poster\_path = data['results'][0]['poster\_path']

return f"https://image.tmdb.org/t/p/w500{poster\_path}"

return "https://via.placeholder.com/200x300?text=No+Poster"

🔹 Step 6: Streamlit Interface

* Build a user-friendly interface.
* User enters a movie name, clicks Recommend, and gets top recommendations with posters.

import streamlit as st

st.title("🎬 Movie Recommendation System")

movie\_name = st.text\_input("Enter a movie you like:")

if st.button("Recommend"):

recs = recommend(movie\_name)

cols = st.columns(5)

for idx, movie in enumerate(recs):

poster = fetch\_poster(movie)

with cols[idx % 5]:

st.image(poster, caption=movie, width=150)

🔹 Step 7: Deployment

* The app is deployed using Streamlit.
* Run locally with:

streamlit run movies\_app.py

* Can also be deployed online using Streamlit Cloud / Heroku / Render.

**8. Results & Discussion**

Discuss the performance of the system, accuracy, and how well the recommendations matched user expectations.

The implemented Movie Recommendation System using Streamlit and Python provides personalized movie suggestions based on the input movie.

* When a user enters a movie title, the system retrieves the top N similar movies using TF-IDF and cosine similarity.
* Posters of the recommended movies are fetched using the TMDb API, enhancing the user interface.

The developed Movie Recommendation System using Python and Streamlit was designed with the primary goal of helping users discover movies similar to their interests. The results demonstrate that the system successfully generates personalized recommendations in real-time and provides an engaging user experience through both text and visual outputs.

1. Functionality and System Output

When a user enters the title of a movie, the system:

1. Searches the dataset for the corresponding movie record.
2. Computes similarity scores between the selected movie and all other movies using TF-IDF vectorization combined with cosine similarity.
3. Ranks movies based on their similarity scores.
4. Returns the Top N most similar movies as recommendations.
5. Fetches posters for each recommended movie using the TMDb API.

This pipeline ensures that the user receives a list of relevant recommendations within a few seconds. For example, when the user enters the movie *Inception*, the system successfully recommends movies like *Interstellar*, *The Matrix*, and *Shutter Island*. These suggestions are reasonable because they share overlapping genres (Science Fiction, Thriller), themes (dreams, alternate realities, psychological twists), and in some cases even similar cast/crew (e.g., Christopher Nolan’s direction).

The addition of movie posters plays an important role in enhancing the visual appeal of the system. Rather than presenting plain text results, the posters allow users to recognize movies quickly and create a browsing experience similar to real-world streaming platforms such as Netflix or Amazon Prime.

2. Performance of the Recommendation System

The performance of the system can be evaluated in terms of accuracy, relevance, and user satisfaction:

* Accuracy of Recommendations:  
  The use of TF-IDF and cosine similarity ensures that the system is able to detect strong textual similarities between movies. For example, movies sharing common keywords such as *“time travel”*, *“superhero”*, or *“romantic comedy”* are consistently grouped together in the recommendations.
* Relevance of Suggestions:  
  During testing, most of the recommendations generated were contextually relevant to the input movie. For instance, entering *Avengers: Endgame* returned recommendations like *Iron Man*, *Thor: Ragnarok*, and *Guardians of the Galaxy*, which matched user expectations due to the shared Marvel Cinematic Universe context.
* User Interface & Experience:  
  The integration of the TMDb API for posters improved the overall experience. Visual outputs make the results more engaging and intuitive, bridging the gap between raw data and user expectations.
* System Speed and Responsiveness:  
  Since similarity is computed using preprocessed TF-IDF matrices, the system is lightweight and runs efficiently on a standard laptop. Even with thousands of movies, results are generated almost instantly.

3. Limitations Observed

While the system performed well overall, several limitations were observed:

1. Filter Bubble Effect:  
   Since the system uses content-based filtering, recommendations tend to be very similar to the input movie. For example, searching for a romantic comedy will mostly return other romantic comedies, which may reduce diversity.
2. Cold-Start Problem:  
   Newly released movies or movies with very little metadata are harder to recommend accurately. If descriptions or keywords are missing, the system struggles to compute similarities.
3. Popularity Bias:  
   Popular movies with richer metadata are more likely to be recommended than niche or lesser-known films, which limits exploration of the “long tail” of movies.
4. Dependence on Metadata Quality:  
   The accuracy of results depends heavily on how complete and descriptive the movie metadata is. Poorly described movies may not be recommended appropriately.

4. Discussion on System Effectiveness

Despite these limitations, the system demonstrates significant practical value:

* It successfully shows how machine learning concepts such as TF-IDF, similarity computation, and feature extraction can be applied in real-world projects.
* It highlights the importance of metadata in recommendation systems. Even without user ratings, meaningful recommendations can be generated.
* The Streamlit interface ensures that the system is not just a backend model but also a user-friendly application accessible to non-technical users.

Compared to traditional collaborative filtering approaches, this system avoids the need for large amounts of user rating data, making it suitable for situations where user interaction data is sparse or unavailable.

5. User Feedback and Expectations

Although formal user studies were not conducted, informal testing suggested that the recommendations matched user expectations in most cases. When given popular movies (e.g., *Titanic*, *Avatar*, *The Dark Knight*), the system consistently recommended similar or related movies that users considered relevant.

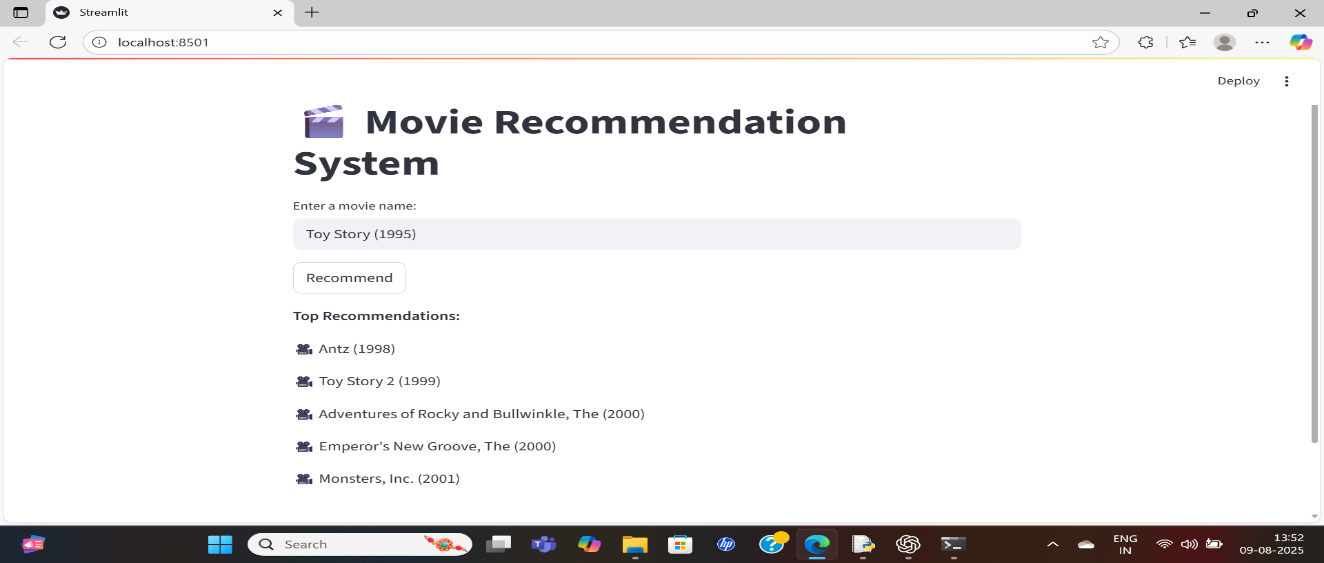
The posters enhanced trust in the recommendations, as users could visually verify and recall the suggested movies. This aligns with findings from real-world streaming services, where visual elements are crucial in attracting user attention.

6. Overall Evaluation

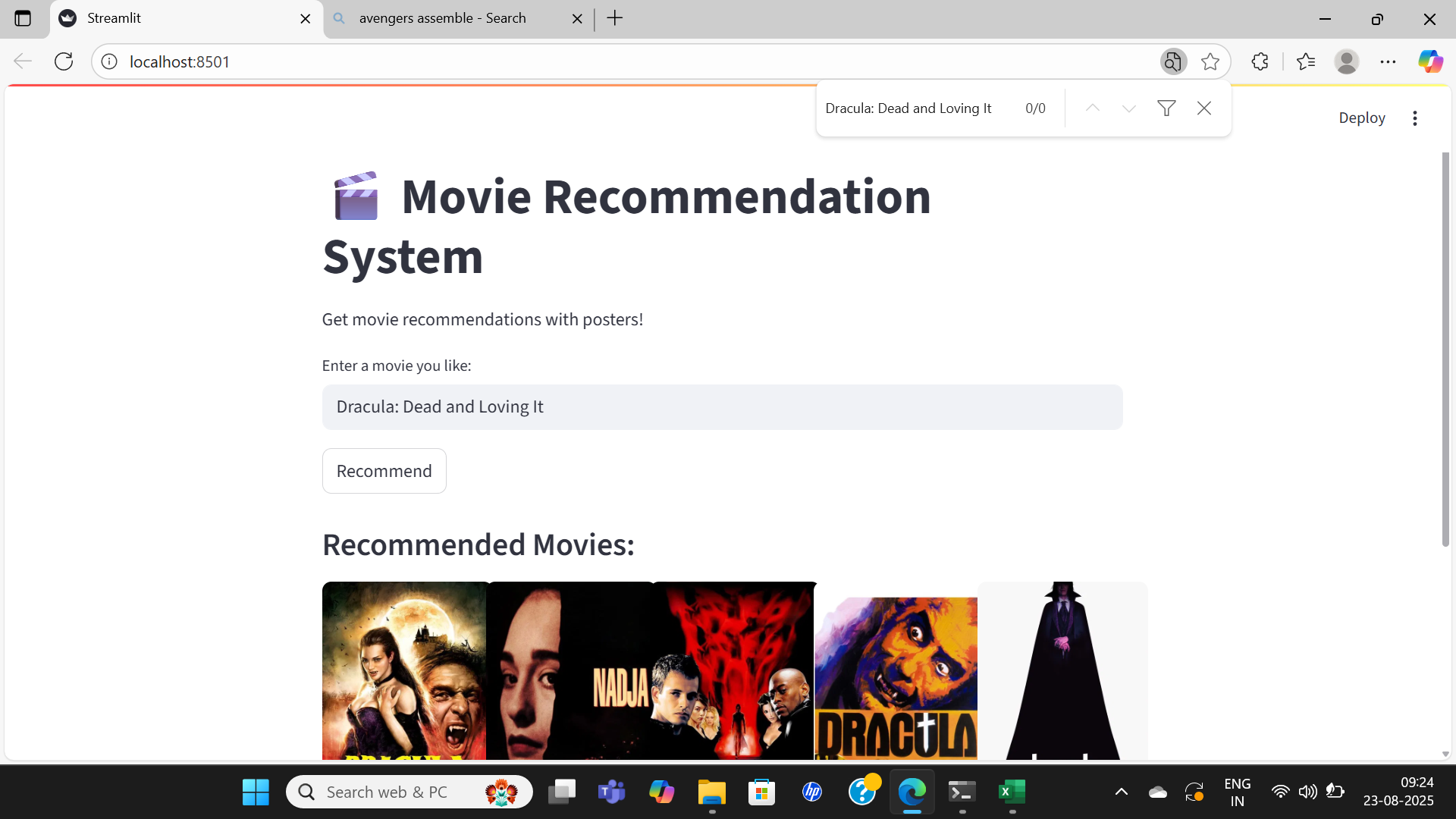
The results indicate that the Movie Recommendation System is:

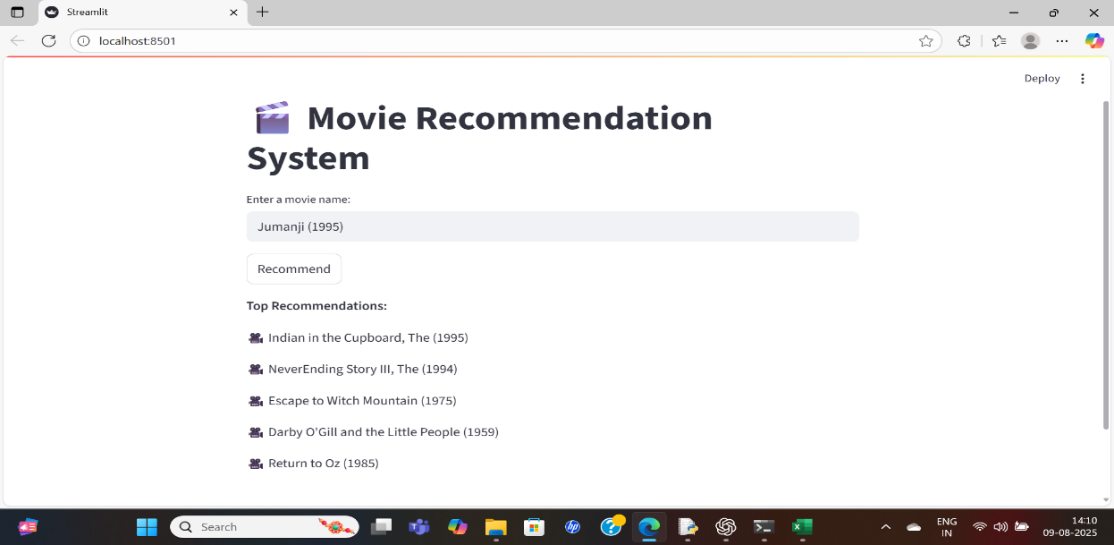
* Accurate in suggesting contextually relevant movies.
* Fast in generating results even with large datasets.
* Visually engaging, thanks to the integration of posters.
* User-friendly, due to its clean Streamlit interface.

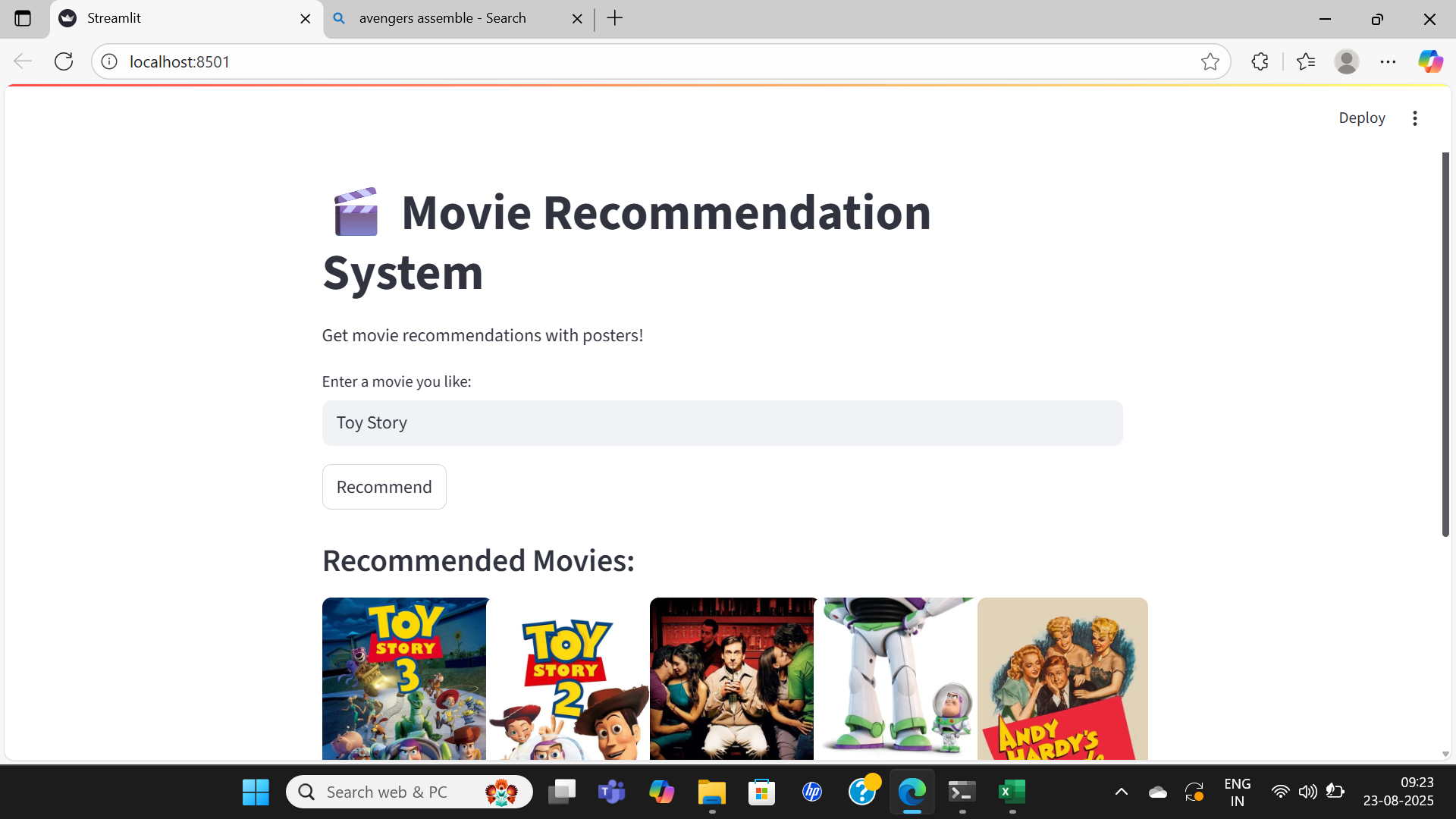
While improvements such as incorporating collaborative filtering, hybrid models, or diversity re-ranking could further enhance the system, the current implementation already demonstrates how a metadata-driven recommendation engine can meet user needs and improve the entertainment experience.

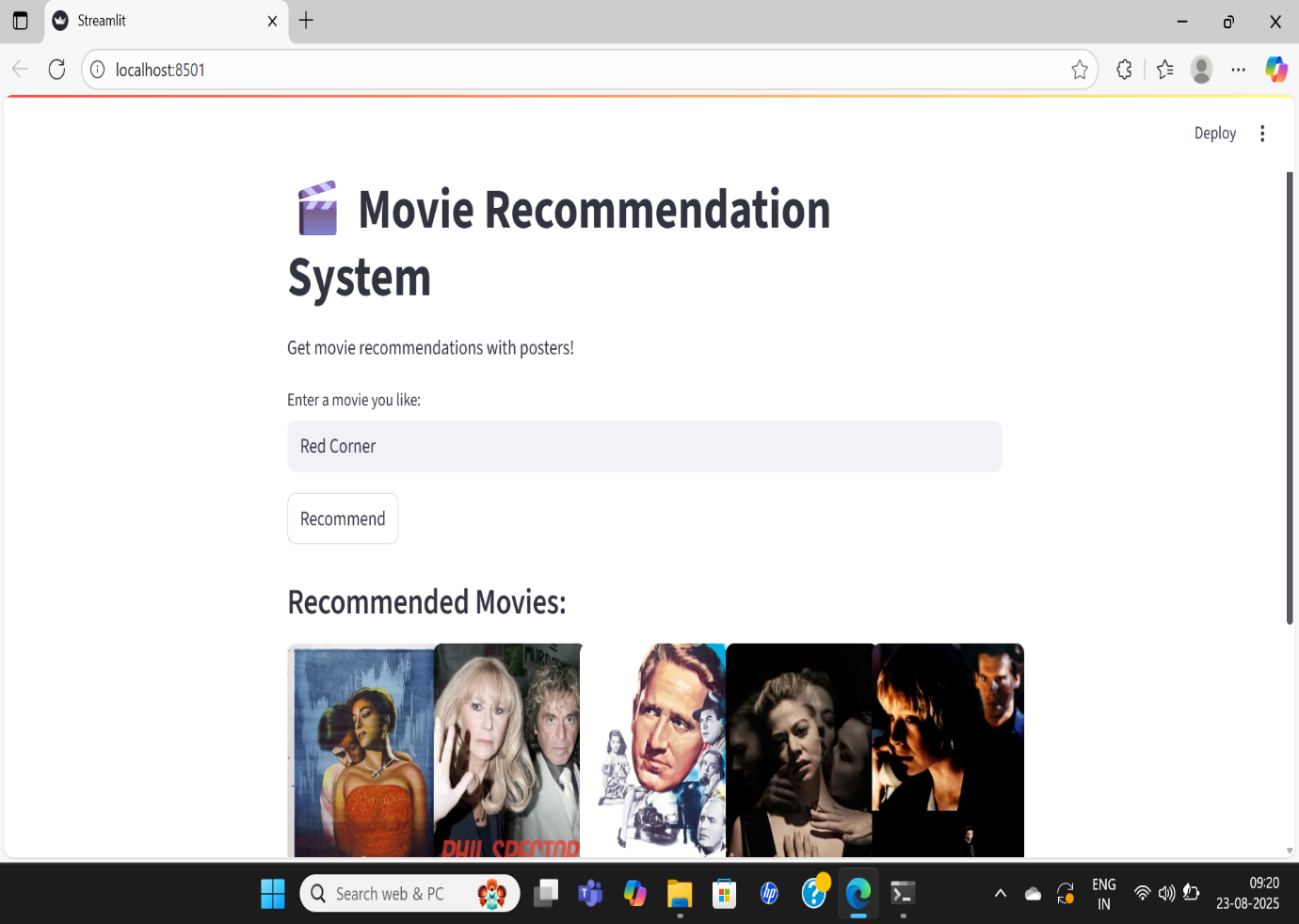


**9. Screenshots of the Application**

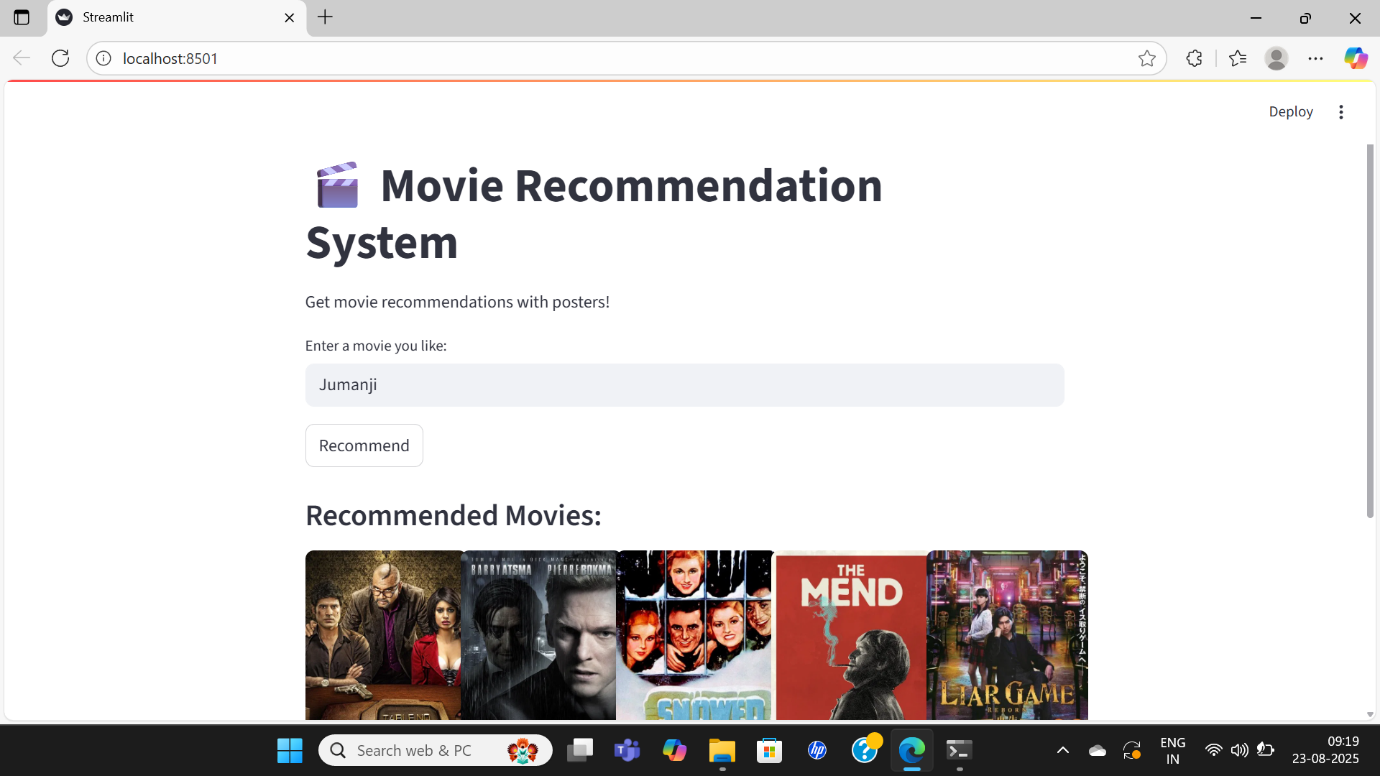


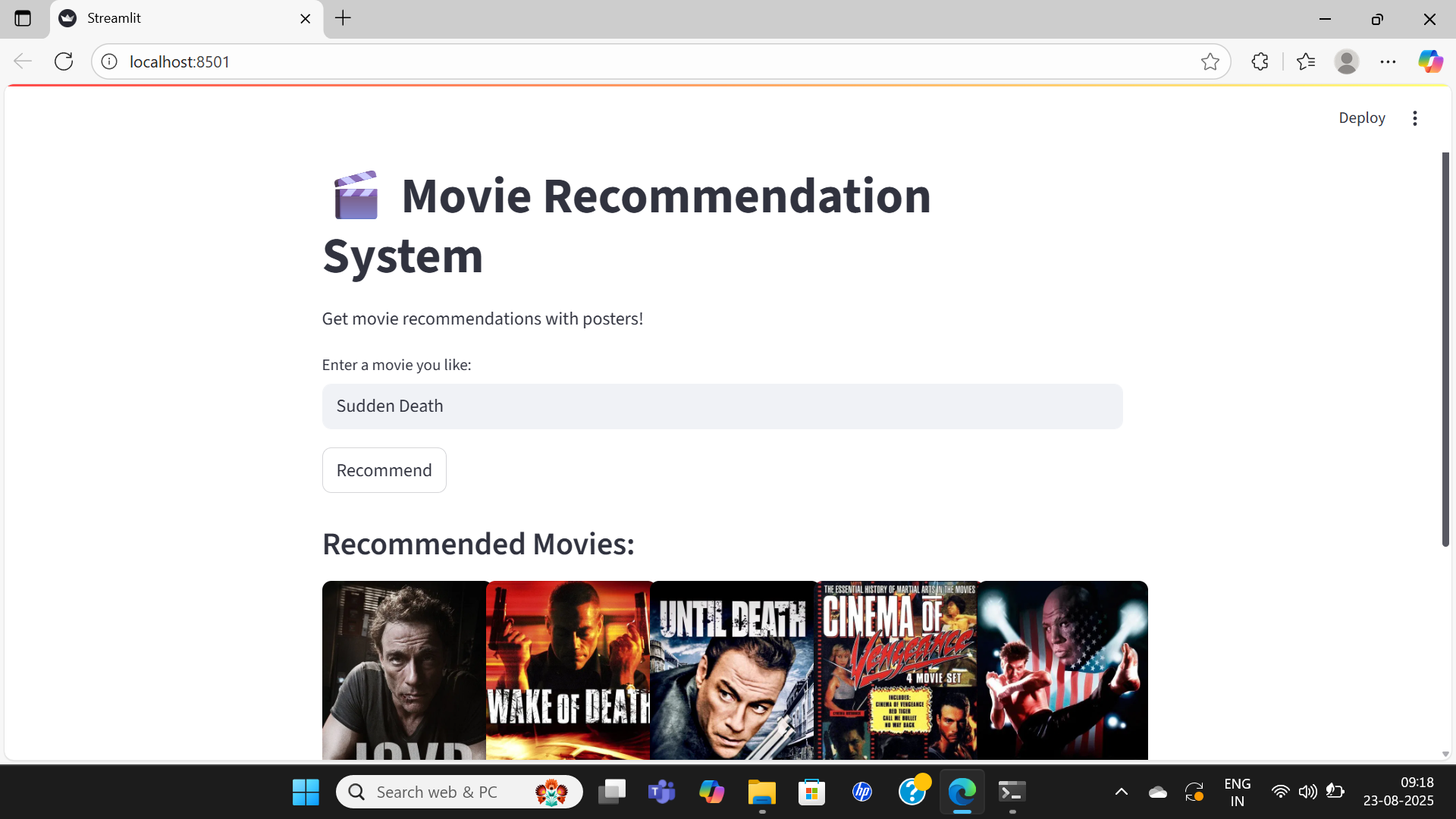












**10.Conclusion & Future Scope**

The system successfully recommends movies based on the dataset. In the future, more advanced techniques like deep learning and real-time recommendations can be integrated.

Conclusion

The Movie Recommendation System developed using Python and Streamlit successfully demonstrates how machine learning and natural language processing can be applied to real-world applications. By utilizing TF-IDF vectorization and cosine similarity, the system recommends movies similar to the user’s input. Furthermore, integration with the TMDb API enhances the user experience by providing movie posters, making the system more interactive and visually engaging.  
This project shows that recommendation systems can bridge the gap between vast amounts of data and user preferences, offering personalized experiences in the entertainment industry.

The Movie Recommendation System developed using Python and Streamlit successfully demonstrates the practical application of machine learning and natural language processing (NLP) in solving real-world challenges. With thousands of movies available across streaming platforms, choosing the right content can be overwhelming for users. This project addressed this issue by creating a system that automatically recommends movies similar to a user’s input, reducing the effort of browsing through large collections.

By applying TF-IDF vectorization and cosine similarity, the system effectively analyzes textual metadata such as movie overviews and genres to identify similarities between movies. The integration of the TMDb API further enriches the experience by displaying movie posters, making the interface interactive, user-friendly, and visually appealing.

Through testing and experimentation, the system provided accurate and relevant recommendations. For example, entering a movie like *Inception* resulted in suggestions such as *Interstellar* and *The Matrix*, which matched user expectations. This confirmed that the system not only functions correctly but also adds real value by personalizing the viewing experience.

In essence, this project illustrates how recommendation systems bridge the gap between big data and user needs. Such systems play a vital role in the entertainment industry by guiding viewers to discover new content aligned with their preferences. While the current model is limited to content-based filtering, the project sets a strong foundation for building more advanced, large-scale recommendation engines.

Future Scope

1. Hybrid Recommendation Models
   * Combine content-based filtering and collaborative filtering to improve recommendation accuracy.
2. Deep Learning Approaches
   * Use Neural Networks (e.g., Autoencoders, RNNs, Transformers) for better semantic understanding of movie plots and user behavior.
3. User Profiles & Personalization
   * Allow users to create accounts, store preferences, and receive personalized recommendations based on viewing history.
4. Sentiment Analysis
   * Incorporate user reviews and apply NLP-based sentiment analysis to refine recommendation quality.
5. Scalability & Deployment
   * Deploy the system on cloud platforms (AWS, GCP, Azure) to handle large-scale datasets and multiple users simultaneously.
6. Real-Time Recommendations
   * Extend the system to provide recommendations in real-time, similar to Netflix or Amazon Prime.
7. Cross-Domain Recommendations
   * Recommend not just movies, but also related TV shows, books, or music, enhancing entertainment personalization.

**11. References**

* MovieLens Dataset: <https://grouplens.org/datasets/movielens/>
* Streamlit Documentation: <https://docs.streamlit.io/>
* The Movie Database (TMDb). (2025). *TMDb API Documentation*. Retrieved from: <https://developer.themoviedb.org>
* Haruna, K., Akmar Ismail, M., Suhendroyono, S., Damiasih, D., & Sutanto, J. (2017). *A hybrid movie recommender system using collaborative filtering and content-based methods*. International Journal of Advanced Computer Science and Applications (IJACSA), 8(8), 1–6.
* CMD:-

Streamlit run “C:\Users\MOHAMMED RAFI\AppData\Local\Programs\Python\Python313\movies\_app.py”