**Movie Recommendation System**

Internship Project Report Elevate Labs

AI ML Intern: Charoo K C

**Abstract**

With the rapid growth of digital entertainment platforms, users are faced with an overwhelming number of movie choices, making content discovery a significant challenge. Traditional search-based methods require users to know in advance what they want to watch, often leading to missed opportunities for discovering movies that align with their interests. To address this issue, this project presents a Movie Recommendation System that employs content-based filtering techniques to provide personalized movie suggestions based on a user-selected title.

The system leverages movie metadata, including genres, cast, directors, and plot summaries, to calculate similarity scores between movies. By applying TF-IDF vectorization, the textual descriptions of movies are converted into numerical feature vectors, capturing the importance of each term relative to the dataset. Cosine similarity is then used to measure the closeness between movies, enabling the identification of the most relevant recommendations.

In addition to the backend machine learning model, a Streamlit-based web interface has been developed to offer a real-time, interactive experience. Users can select a movie from a dropdown menu and instantly receive a list of recommended movies, along with their corresponding posters, making the system accessible even to users without technical knowledge.

This project demonstrates a complete end-to-end solution, combining data preprocessing, feature engineering, machine learning, natural language processing, and frontend web development. The system not only enhances the user experience by simplifying movie discovery but also provides a practical example of applying AI and ML techniques to real-world problems in the entertainment industry. Future improvements, such as incorporating collaborative filtering, user ratings, and advanced NLP embeddings, can further increase recommendation accuracy and personalization.

**Table of Contents**

1. Chapter 1 – Introduction
2. Chapter 2 – Problem Statement
3. Chapter 3 – System Requirement Specifications
4. Chapter 4 – System Design and Tools/Technologies Used
5. Chapter 5 – Methodology
   * 5.1 Dataset Description
   * 5.2 Data Preprocessing
   * 5.3 Feature Engineering
   * 5.4 Similarity Computation
   * 5.5 Backend Implementation
   * 5.6 Frontend Implementation (Streamlit)
6. Chapter 6 – Results and Discussion
7. Chapter 7 – Limitations
8. Chapter 8 – Future Enhancements
9. References
10. Appendix

**Chapter 1 – Introduction**

In today’s digital era, the consumption of movies and video content has increased exponentially. Online streaming platforms like Netflix, Amazon Prime, Disney+, and others host thousands of movies and TV shows, making it difficult for users to choose content that aligns with their preferences. This vast availability of options can often lead to decision fatigue, where users struggle to select what to watch, resulting in an unsatisfactory user experience.

Recommendation systems have emerged as a solution to this problem. These systems analyze user preferences and item characteristics to suggest content that a user is likely to enjoy. Broadly, recommendation systems are categorized into two types:

1. Content-Based Filtering:
   * Recommends items similar to those a user has already liked or selected.
   * Analyzes item attributes such as genres, cast, director, and textual descriptions to determine similarity.
2. Collaborative Filtering:
   * Recommends items based on the preferences and behaviors of similar users.
   * Relies on user ratings, interactions, or reviews to generate recommendations.

This project focuses on content-based filtering to build a Movie Recommendation System. By analyzing the features of movies, the system generates recommendations that are relevant to the user’s selection. Key movie metadata, including genres, cast, director, and plot summaries, are processed using natural language processing (NLP) techniques, converting textual information into numerical feature vectors via TF-IDF vectorization. The cosine similarity measure is then used to calculate the closeness between movies, allowing the system to recommend the most similar titles.

A key aspect of this project is the Streamlit-based web interface, which allows users to interact with the recommendation system in real time. The interface is user-friendly, requiring no technical knowledge, and displays recommendations along with movie posters to enhance user experience.

This project not only provides a practical solution for content discovery but also demonstrates the application of machine learning, NLP, and web development in building an end-to-end AI-powered recommendation system. It showcases how technology can improve user engagement and satisfaction on digital platforms.

**Chapter 2 – Problem Statement**

With the explosion of digital content on streaming platforms, users are confronted with an overwhelming number of movie choices. Platforms like Netflix, Amazon Prime, Disney+, and Hulu host thousands of movies across multiple genres and languages, making it increasingly difficult for users to identify content that matches their personal interests. This vast availability of choices often leads to decision fatigue, where users spend excessive time browsing or may end up choosing content at random, resulting in a suboptimal viewing experience.

Challenges faced by users:

1. Information Overload:

The sheer volume of movies and TV shows available can overwhelm users, making it difficult to find content of interest.

1. Limited Search Capabilities:

Traditional search functions rely on keywords or titles, which require users to know in advance what they want to watch. Users often miss out on movies they might enjoy because they cannot effectively search the entire library.

1. Lack of Personalization:

Without a recommendation system, users are offered a generic content list that does not consider their preferences, leading to dissatisfaction and lower engagement.

Challenges faced by developers:

1. Feature Extraction from Metadata:

Extracting meaningful features such as genres, cast, directors, and plot summaries from large datasets can be complex, especially when handling missing or inconsistent data.

1. Similarity Computation:

Calculating similarity between movies requires an efficient method that can handle high-dimensional data while providing accurate recommendations.

1. User-Friendly Interface:

Even a technically accurate recommendation engine is insufficient if the system is not accessible to users. Designing an interactive interface that integrates the recommendation engine and displays results effectively is critical.

Problem Definition:

The primary objective of this project is to design and implement a Movie Recommendation System that:

* Automatically identifies and suggests movies similar to a user-selected title.
* Processes movie metadata efficiently to extract meaningful features for recommendation.
* Provides an interactive and user-friendly interface that displays recommendations in real time.
* Enhances the user experience by offering personalized movie suggestions based on content similarity rather than relying solely on user ratings or prior interactions.

By addressing these challenges, the system aims to reduce decision fatigue, improve content discovery, and provide a practical example of how machine learning and natural language processing can be applied to real-world entertainment platforms.

Scope of the Problem:

* This project focuses on content-based recommendations, analyzing movie attributes to determine similarity.
* Collaborative filtering or hybrid recommendation methods are not the primary focus but can be incorporated as future enhancements.
* The system is designed for users with minimal technical knowledge, ensuring accessibility through a Streamlit web application.

**Chapter 3 – System Requirement Specifications**

The Movie Recommendation System is a combination of machine learning, natural language processing, and web-based technologies. To develop and run this system effectively, both hardware and software requirements must be clearly defined. This chapter outlines the system requirements, including hardware, software, datasets, programming environment, and libraries.

**3.1 Hardware Requirements**

Efficient execution of the system requires the following minimum hardware specifications:

| **Component** | **Minimum Requirement** | **Recommended Requirement** |
| --- | --- | --- |
| Processor (CPU) | Intel i3 / AMD Ryzen 3 | Intel i5 / AMD Ryzen 5 or higher |
| RAM | 4 GB | 8 GB or higher |
| Storage | 20 GB free space | 50 GB free space |
| Graphics (Optional) | Integrated graphics | Dedicated GPU (optional for future enhancements) |
| Monitor | Standard 13-inch | 15-inch or higher |

**Explanation:**

* The CPU is used for data preprocessing, vectorization, and similarity calculations.
* RAM is essential to handle large datasets and matrix computations during TF-IDF vectorization and similarity calculation.
* Storage space is required to store datasets, project files, and dependencies.
* A dedicated GPU is not necessary for this project but can improve performance if larger datasets or deep learning methods are implemented in the future.

**3.2 Software Requirements**

The software environment provides the platform to implement the backend machine learning algorithms, frontend interface, and data processing modules.

| **Software Component** | **Version / Specification** |
| --- | --- |
| Operating System (OS) | Windows 10/11, Linux (Ubuntu 20.04+), MacOS |
| Programming Language | Python 3.8 or higher |
| IDE / Code Editor | Jupyter Notebook, VS Code |
| Web Interface Framework | Streamlit (latest stable release) |
| Version Control | Git (latest) and GitHub repository |
| Web Browser | Google Chrome / Firefox for testing Streamlit app |

**Explanation:**

* **Python** is chosen due to its extensive libraries for data analysis, machine learning, and web development.
* **Jupyter Notebook** is used for prototyping and experimentation with machine learning models.
* **VS Code** is recommended for writing production-ready scripts like app.py.
* **Streamlit** is used for building an interactive web-based interface to showcase recommendations in real time.
* **Git/GitHub** ensures version control, collaboration, and backup of code and project files.

**3.3 Dataset Requirements**

The system uses **movie metadata datasets** to generate recommendations. Essential dataset attributes include:

| **Attribute** | **Description** |
| --- | --- |
| Title | Name of the movie |
| Genres | Movie genre(s) |
| Cast | Main actors |
| Director | Director of the movie |
| Overview | Plot summary / description of the movie |
| Other Metadata | Year of release, ratings, etc. (optional) |

**Dataset Source:**

* Kaggle: TMDb Movie Metadata Dataset

**Explanation:**

* The dataset provides structured information required to compute similarity between movies.
* Textual attributes like “overview” are used for TF-IDF vectorization, while categorical attributes like genres, cast, and director are combined into a unified feature representation.

**3.4 Library Requirements**

The project leverages several Python libraries for data processing, machine learning, and web development:

| **Library** | **Purpose** |
| --- | --- |
| Pandas | Data loading, cleaning, manipulation |
| NumPy | Numerical computations |
| scikit-learn | TF-IDF vectorization, cosine similarity |
| NLTK | Text preprocessing (tokenization, lemmatization) |
| Streamlit | Frontend web application |
| Requests (Optional) | Fetching external resources like posters |
| Matplotlib / Seaborn (Optional) | Data visualization |

**Explanation:**

* **Pandas & NumPy** are core libraries for handling datasets and numerical operations efficiently.
* **scikit-learn** provides robust implementations of TF-IDF vectorizer and cosine similarity functions.
* **NLTK** ensures proper natural language preprocessing to improve recommendation accuracy.
* **Streamlit** allows users to interact with the recommendation engine without technical knowledge.
* **Visualization libraries** can be used to analyze distributions and similarity matrices during experimentation.

**3.5 System Environment Setup**

To run the project:

1. Install Python 3.8+ from the official website.
2. Create a virtual environment for dependency management:
3. Install required libraries:
4. Download the dataset from Kaggle and place it in the project directory.
5. Run the backend scripts (recommendation\_engine.py) to precompute similarity matrices.
6. Launch the Streamlit app for interactive recommendations

**3.6 Explanation of Requirements**

* The combination of hardware, software, and libraries ensures smooth execution of machine learning tasks and real-time recommendations.
* The setup allows scalability, meaning larger datasets or additional features can be integrated in the future.
* Using a virtual environment prevents dependency conflicts and keeps the project organized.

**Chapter 4 – System Design and Tools/Technologies Used**

The system design of the Movie Recommendation System provides a structured approach to building the application, covering data processing, recommendation logic, and user interaction through the web interface. This chapter explains the design methodology, architecture, flow of data, and the tools and technologies used.

**4.1 System Architecture**

The Movie Recommendation System is divided into **three major modules**:

1. **Data Module:** Handles loading, cleaning, and preprocessing of movie datasets.
2. **Recommendation Engine Module:** Implements machine learning and NLP techniques to compute similarity scores and generate recommendations.
3. **User Interface Module:** A Streamlit-based frontend that allows users to interact with the system and view recommendations in real time.

**4.2 Data Flow**

The data flow within the system can be explained step by step:

1. **Dataset Loading:**  
   The system reads a CSV file containing movie metadata, including title, genres, cast, director, and overview.
2. **Data Preprocessing:**  
   Missing values in key columns are filled, and text columns are cleaned by lowercasing, removing punctuation, and tokenizing sentences.
3. **Feature Engineering:**  
   Relevant features such as genres, cast, director, and overview are combined into a single text feature. This enables the TF-IDF .
4. **TF-IDF Vectorization:**  
   Converts the combined text feature into numerical vectors. TF-IDF (Term Frequency–Inverse Document Frequency) captures the words describing each movie .
5. **Similarity Computation:**  
   Cosine similarity is computed between all movie vectors to determine how closely related each movie is to others in the dataset.
6. **Recommendation Engine:**  
   The top N movies with the highest similarity scores to the selected movie are retrieved and returned as recommendations.
7. **Streamlit User Interface:**  
   Recommendations are displayed interactively in the frontend, including movie titles and posters. Users can select a movie from a dropdown and instantly view the results.

**4.3 Tools and Technologies Used**

The project uses a combination of programming languages, libraries, and frameworks. Each tool serves a specific purpose:

| **Tool / Technology** | **Purpose** |
| --- | --- |
| Python | Primary programming language for data processing and backend logic |
| Pandas & NumPy | Data handling, cleaning, and numerical operations |
| scikit-learn | TF-IDF vectorization and cosine similarity computation |
| NLTK | Natural language preprocessing (tokenization, lemmatization) |
| Streamlit | Web-based interactive frontend for recommendations |
| Matplotlib / Seaborn | Optional: Data visualization for dataset exploration |
| Git / GitHub | Version control and collaboration |
| Jupyter Notebook | Prototyping and experimentation with machine learning models |
| VS Code | Writing production scripts such as app.py |

**Explanation of Key Tools:**

* **Python:** Chosen for its versatility and extensive libraries for machine learning and NLP.
* **Pandas & NumPy:** Enable efficient manipulation of datasets and matrix operations, which are crucial for similarity calculations.
* **scikit-learn:** Provides robust implementations for TF-IDF vectorizer and cosine similarity functions.
* **NLTK:** Ensures proper text preprocessing, which improves the accuracy of recommendations.
* **Streamlit:** Allows users to interact with the recommendation engine in real time without requiring programming knowledge.
* **Matplotlib/Seaborn:** Useful for visualizing data distributions, such as genre frequency and similarity matrices during development.

**4.4 Design Considerations**

* **Modularity:** Each module (data, engine, frontend) is designed independently, allowing easy updates or replacement.
* **Scalability:** The system can handle larger datasets or additional features without significant redesign.
* **User Experience:** The Streamlit UI ensures recommendations are easy to access, with clear and intuitive navigation.
* **Performance:** Cosine similarity calculations and vectorizations are optimized to handle medium-sized datasets efficiently.

**4.5 Advantages of the Design**

1. Clear separation of backend and frontend reduces complexity.
2. Streamlit interface allows real-time user interaction without requiring technical expertise.
3. Reusable recommendation engine functions can be integrated into larger platforms.
4. Scalable and extendable design allows future integration of hybrid or collaborative filtering techniques.

**Chapter 5 – Methodology**

The Movie Recommendation System was implemented using a structured approach that combines data processing, machine learning, natural language processing (NLP), and a web-based interface. This chapter describes the methodology and implementation steps in detail.

**5.1 Dataset Preparation**

The system uses a movie metadata dataset containing key attributes such as movie title, genres, cast, director, and plot summary. Initially, the dataset was explored to identify missing or inconsistent data. Only relevant columns were retained for the recommendation process. Proper dataset preparation ensures that the recommendation engine has accurate and meaningful information to analyze.

**5.2 Data Preprocessing**

To make the dataset usable for analysis, data preprocessing was performed. This included:

1. **Handling Missing Values:**

Any missing information in important columns such as genre or plot summary was replaced with placeholder text.

1. **Text Normalization:**

Text data, such as movie descriptions and cast names, were standardized by converting to lowercase and removing unnecessary characters.

1. **Feature Combination:**

Key attributes (genres, cast, director, and overview) were combined into a single feature. This allows the system to consider all aspects of a movie when determining similarity.

Data preprocessing ensures that the dataset is clean, consistent, and ready for feature extraction.

**5.3 Feature Engineering**

The system uses feature engineering to represent movies as numerical vectors that can be compared. By analyzing the combined movie features, the system identifies which movies share similar attributes. This process involves assigning weights to different terms in the movie descriptions and metadata to highlight the most important features, such as genre-specific keywords or unique cast members. This structured representation of movies is crucial for generating accurate recommendations.

**5.4 Similarity Computation**

The core of the recommendation engine is similarity computation. Each movie is compared with every other movie in the dataset to determine how similar they are. The similarity is calculated based on the combined features of the movies, allowing the system to identify the top matches. Movies with high similarity scores are considered closely related, while movies with low scores are less related. This approach ensures that recommendations are contextually relevant.

**5.5 Backend Implementation**

The backend of the system consists of the recommendation engine, which processes the dataset and provides recommendations:

* When a user selects a movie, the system locates its position in the dataset.
* It retrieves similarity scores for all other movies relative to the selected movie.
* The system identifies the top recommended movies based on these scores.

This backend is designed to be modular, allowing the recommendation engine to function independently and provide results in real time.

**5.6 Frontend Implementation – Streamlit Interface**

To make the system accessible, a web-based interface was developed using Streamlit:

* Users can select a movie from a dropdown menu.
* The interface displays the top recommended movies.
* Movie posters or other visuals can be included to enhance user experience.

The frontend is designed to be intuitive and interactive, enabling users to receive recommendations instantly without any technical knowledge.

**5.7 Integration of Backend and Frontend**

The backend recommendation engine and the frontend interface are seamlessly integrated:

1. The user selects a movie in the frontend.
2. The backend computes the top recommendations using similarity analysis.
3. The frontend displays the recommended movies immediately.

This end-to-end integration demonstrates a complete pipeline from data processing to user interaction.

**Chapter 6 – Results and Discussion**

The Movie Recommendation System was tested using the TMDb movie dataset to evaluate its ability to provide accurate and meaningful movie suggestions. This chapter discusses the outcomes, user interaction, and observations from the system.

**6.1 System Testing**

The system was tested with a variety of movie titles across multiple genres, including action, drama, comedy, and science fiction. The goal was to determine whether the recommendation engine could consistently provide relevant suggestions. Testing involved:

1. Selecting a movie from the dropdown menu in the Streamlit interface.
2. Observing the top recommended movies generated by the system.
3. Evaluating whether the recommended movies shared similar themes, genres, or cast members with the selected movie.

**6.2 Sample Recommendations**

**Example 1:**

* **Selected Movie:** *The Dark Knight*
* **Recommended Movies:** *Inception*, *Interstellar*, *The Prestige*, *Batman Begins*, *Memento*, *Insomnia*, *Tenet*, *Dunkirk*, *The Dark Knight Rises*, *Shutter Island*

**Observations:**

* The recommendations included movies from similar genres (action, thriller) and shared the same director or thematic style.
* Users are provided with movies they may enjoy even if they are not aware of them beforehand.

**Example 2:**

* **Selected Movie:** *Forrest Gump*
* **Recommended Movies:** *The Green Mile*, *Cast Away*, *The Pursuit of Happyness*, *Big Fish*, *Saving Private Ryan*

**Observations:**

* Recommendations focus on movies with similar emotional or narrative content.
* The system successfully identifies contextually similar movies beyond just genre matching.

**6.3 User Interaction and Interface**

The **Streamlit web interface** was designed to provide a simple and interactive experience:

* Users can select movies from a searchable dropdown menu.
* The top 10 recommendations are displayed immediately below the selection.
* Optional movie posters or brief summaries can enhance the interface further.

**User Feedback:**

* The interface is easy to use and does not require any technical knowledge.
* Users appreciate the instant display of recommendations, improving their decision-making process for choosing movies.

**6.4 Analysis of Results**

1. **Accuracy of Recommendations:**
   * The system provides recommendations that are contextually relevant to the selected movie.
   * Similarity is calculated based on combined features, which ensures multi-dimensional relevance (genre, cast, director, plot).
2. **Strengths:**
   * Real-time recommendations through a user-friendly interface.
   * Works effectively across different genres.
   * Modular design allows future enhancements.
3. **Limitations:**
   * Recommendations are limited to movies available in the dataset.
   * Movies with incomplete metadata may receive less accurate suggestions.
   * Cold-start problem: Newly released movies without metadata cannot be recommended effectively.

**Chapter 7 – Limitations**

While the Movie Recommendation System provides a functional and interactive platform for movie suggestions, several limitations were observed during development and testing. Understanding these limitations is important for future improvements and realistic expectations of system performance.

**7.1 Dataset Limitations**

1. **Incomplete Metadata:**
   * Some movies in the dataset had missing information for key attributes such as genres, cast, director, or overview.
   * Missing data reduces the accuracy of similarity computations, resulting in less relevant recommendations for those movies.
2. **Static Dataset:**
   * The system relies on a pre-existing dataset, which is updated only manually.
   * Newly released movies or lesser-known titles may not be present, leading to a **cold-start problem** where the system cannot recommend these movies effectively.
3. **Limited Diversity:**
   * Certain movie genres or regional films may be underrepresented in the dataset.
   * This can cause biased recommendations, favoring popular or frequently occurring movies.

**7.2 System Design Limitations**

1. **Content-Based Filtering Only:**
   * The current system is purely content-based, which limits its ability to capture user preferences beyond the attributes of the movie itself.
   * Unlike collaborative filtering, it does not consider user ratings, watch history, or community trends.
2. **Scalability Constraints:**
   * For small to medium datasets, the system performs efficiently.
   * However, as the dataset size grows significantly, the **cosine similarity computation** can become resource-intensive, impacting performance and response time.
3. **No Personalization:**
   * Recommendations are based solely on movie content similarity, not on individual user profiles or preferences.
   * Users with unique tastes may receive generic recommendations if their interests are not reflected in the dataset features.

**7.3 Technical Limitations**

1. **Dependence on Feature Quality:**
   * The accuracy of recommendations heavily depends on the quality and richness of the metadata.
   * Poorly described movies or insufficient feature information can reduce recommendation relevance.
2. **Limited Advanced NLP Techniques:**
   * The current system uses TF-IDF vectorization for text features.
   * It does not incorporate advanced NLP embeddings like **BERT** or **Sentence Transformers**, which could capture deeper semantic relationships between movies.
3. **Frontend Limitations:**
   * While the Streamlit interface is interactive, it is relatively simple.
   * It lacks advanced filtering options, sorting by ratings, or user feedback integration.

Despite these limitations, the system provides a functional, easy-to-use, and accurate movie recommendation experience for users, serving as a strong foundation for future enhancements and improvements.

**Chapter 8 – Future Enhancements**

While the Movie Recommendation System provides a robust foundation for content-based movie recommendations, there are multiple opportunities for improvement. This chapter outlines potential future enhancements aimed at increasing accuracy, personalization, scalability, and overall user experience.

**8.1 Incorporating Collaborative Filtering**

* **Objective:** Combine user behavior with content-based filtering to create a **hybrid recommendation system**.
* **Approach:**
  + Utilize user ratings, watch history, or interactions to identify patterns in user preferences.
  + Combine these patterns with content-based recommendations to provide more personalized results.
* **Benefit:** This approach addresses the limitation of purely content-based systems, offering tailored recommendations for users with unique tastes.

**8.2 Using Advanced NLP Techniques**

* **Objective:** Improve semantic understanding of movie descriptions for more accurate recommendations.
* **Approach:**
  + Implement **pre-trained language models** such as BERT, RoBERTa, or Sentence Transformers to generate embeddings for movie plot summaries and other text features.
  + These embeddings capture deeper contextual meaning compared to TF-IDF, allowing the system to detect subtle similarities between movies.
* **Benefit:** This enhancement will increase the quality of recommendations, especially for movies with complex narratives or unique plotlines.

**8.3 Expanding the Dataset**

* **Objective:** Increase coverage and diversity of movies to improve recommendations.
* **Approach:**
  + Integrate multiple datasets from sources such as IMDb, TMDb API, or other public movie databases.
  + Include regional, independent, and newly released films to reduce bias and ensure broader representation.
* **Benefit:** Users will receive recommendations that cover a wider range of genres, languages, and categories.

**8.4 Improving the Frontend Interface**

* **Objective:** Enhance user interaction and experience in the Streamlit app.
* **Approach:**
  + Add features such as movie poster display, ratings, release year, and brief plot summaries.
  + Include advanced filtering options by genre, director, or actor.
  + Implement a search bar and recommendation history for returning users.
* **Benefit:** A more engaging interface encourages frequent use and improves satisfaction.

**8.5 Real-Time Updates and Scalability**

* **Objective:** Make the system capable of handling large datasets and real-time updates.
* **Approach:**
  + Optimize similarity computations using techniques like approximate nearest neighbor (ANN) algorithms.
  + Set up automated scripts to periodically update the dataset with new movies and metadata.
* **Benefit:** Users can access the latest movies and recommendations without delays, making the system more scalable and responsive.

**8.6 Personalization Features**

* **Objective:** Tailor recommendations to individual users.
* **Approach:**
  + Create user profiles based on past selections, ratings, and preferences.
  + Use machine learning algorithms to adapt recommendations based on user feedback.
* **Benefit:** Enhances user engagement and ensures that recommendations are more relevant to each individual.

**8.7 Integration with External Services**

* **Objective:** Enrich recommendations by integrating external data sources.
* **Approach:**
  + Use APIs from IMDb or TMDb to fetch movie ratings, reviews, and posters.
  + Integrate with social media or streaming platforms to recommend trending or popular movies.
* **Benefit:** Users receive more informative and visually appealing recommendations

**References**

* Kaggle Movie Dataset: https://www.kaggle.com/tmdb/tmdb-movie-metadata
* Scikit-learn Documentation: https://scikit-learn.org
* NLTK Documentation: https://www.nltk.org
* Streamlit Documentation: https://docs.streamlit.io

**Appendix**

* GitHub Repository: https://github.com/charookc5/Movie\_Recommendation\_System