A

**Project Report**

On

**“Movie Matcher”**

Submitted by

**Harsh Srivastava** (19223)

**Anand Verman** (19214)

**S**ubmitted to

**Prof. A. K. Malviya**

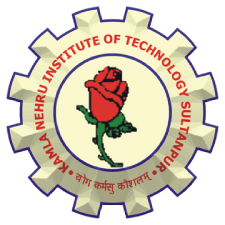
**Prof. Vinay Kumar**

For Partial Fulfilment of Award of Degree

In

**Bachelor of Technology**

at



Department of Computer Science & Engineering

**Kamla Nehru Institute of Technology, Sultanpur**

Affiliated to Dr. A.P.J. Abdul Kalam Technical University, Lucknow

April, 2023

# Declaration

We hereby declare that this submission is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person or material which to a substantial extent has been accepted for the award of any other degree or diploma of the University or other institute of higher education, except where due acknowledgement has been made in the text.

Harsh Srivastava (19223)

Anand Verman (19214)

# Certificate

Certified that **Harsh Srivastava** (Roll No. 19223) and **Anand Verman** (Roll No. 19214) have carried out the project work presented in this project report entitled **“Movie Matcher”** for the award of Bachelor of Technology in Computer Science and Engineering from Department of Computer Science and Engineering, Kamla Nehru Institute of Technology, Sultanpur under our guidance. The project report embodies results of original work, and studies are carried out by the students themselves and the contents of the project report do not form the basis for the award of any other degree to the candidates or to anybody else from this or any other University/Institution.

Supervisor Signature                                               Supervisor Signature

(Prof. A. K. Malviya)                                                 (Prof. Vinay Kumar)

# Acknowledgement

After bowing our heads in front of almighty god, we take this opportunity to thank all the people who have continuously worked for the betterment and successful completion of this project. We sincerely would like to pay our deepest gratitude to Prof. A.K. Malviya and Prof. Vinay Kumar Department of Computer Science and Engineering, Kamla Nehru Institute of Technology, Sultanpur our project guide for their constant motivation to carry our works with utmost dedication, without their constant support this project was a distant dream. We would like to pay sincere thanks to Dr. Awadhesh Kumar our Head of Department, Computer Science and Engineering. We also extend our thanks to the director of Kamla Nehru Institute of Technology for providing us with a conducive academic environment and for enabling us to pursue our academic aspirations. Additionally, we would like to express our gratitude to the entire staff of the department for their continuous support, guidance, and suggestions. Lastly, we would like to thank our parents, whose unwavering love and support have been a constant source of strength for us. We are forever grateful for their encouragement and guidance.

# Abstract

The abstract presents a content-based movie recommendation system that utilizes movie attributes to generate personalized suggestions. By considering genre, cast, director, and keywords, the system identifies similarities between movies to recommend relevant content. The system operates without relying on user data, instead focusing on intrinsic movie characteristics. Through feature extraction and similarity measures, it provides accurate and customized recommendations based solely on the content of movies. The aim of this content-based approach is to enrich the movie-watching experience by introducing users to films that align with their preferences, fostering exploration, and engagement within the realm of cinema.

# Table of Contents

[Introduction 9](#_Toc134753322)

[Problem Statement 10](#_Toc134753323)

[Issues 11](#_Toc134753324)

[Objectives 11](#_Toc134753325)

[Literature Survey 13](#_Toc134753326)

[Implementation 16](#_Toc134753336)

[Software Requirements and Analysis: 16](#_Toc134753337)

[Hardware and Software Requirements 17](#_Toc134753338)

[Dataset 19](#_Toc134753339)

[Vectorization and Cosine Similarity 21](#_Toc134753340)

[Dataset pre-processing 22](#_Toc134753341)

[Cleaning the Data: 23](#_Toc134753342)

[Making data ready for feature extraction 27](#_Toc134753343)

[Stemming the tags: 27](#_Toc134753344)

[Feature Extraction and Implementation of Recommendation System 28](#_Toc134753345)

[Making the Vectors 28](#_Toc134753346)

[Applying Cosine Similarity to Vectors 29](#_Toc134753347)

[Exporting the final dataset and similarity matrix by pickling 30](#_Toc134753348)

[Making WebApp using Streamlit 31](#_Toc134753349)

[Main components of the WebApp: 31](#_Toc134753350)

[Result 35](#_Toc134753351)

[Conclusion 39](#_Toc134753352)

[Future Scope: 39](#_Toc134753353)

[References 41](#_Toc134753354)

# List Of Figures

[Figure 3.1 Modules implemented in Movie Matcher 32](#_Toc134118753)

[Figure 3.2 Data Flow Diagram](#_Toc134118755) 33

[Figure 3.3 Use Case Diagram 33](#_Toc134118756)

[Figure 3.4 Working Flow of Movie Matcher](#_Toc134118757) 34

**Chapter 1**

# Introduction

In the vast world of movies, with countless options available across various genres, finding the perfect film to watch can be a daunting task. Movie recommendation systems have emerged as valuable tools to assist users in discovering films that align with their preferences. Among the different types of recommendation systems, content-based approaches have gained significant attention. A content-based movie recommendation system leverages the inherent attributes and characteristics of movies to generate personalized suggestions without relying on explicit user data.

The goal of a content-based movie recommendation system is to analyze the features and content of movies, such as genre, cast, director, and keywords, and use this information to recommend similar films to users. By focusing on the intrinsic characteristics of movies, content-based systems provide recommendations based on the content itself, rather than relying on collaborative filtering or user behavior patterns.

One of the key advantages of a content-based approach is that it does not require explicit user feedback or historical data. This makes it particularly useful for new users or in situations where user data is sparse or unavailable. By considering the attributes of movies, the system can generate relevant suggestions that align with the user's preferences, regardless of their previous interactions with the system.

To create a content-based movie recommendation system, several steps are involved. The system starts by acquiring a comprehensive dataset of movies, which includes information such as titles, genres, cast members, directors, and keywords. This dataset serves as the foundation for the recommendation engine. Feature extraction techniques are then applied to capture the important characteristics of movies from the dataset.

Once the features are extracted, similarity measures are utilized to identify the similarity between movie cosine similarity or Jaccard similarity, are commonly employed to calculate these. Various algorithms, such as e resemblance between movies based on their feature vectors. The system then selects the most similar movies as recommendations for a given movie or user query.

The content-based movie recommendation system can also be enhanced by incorporating additional techniques such as natural language processing (NLP) for analyzing movie summaries or sentiment analysis to understand user preferences based on text reviews. These techniques can further refine the recommendation process and provide more accurate and personalized suggestions.

In conclusion, a content-based movie recommendation system offers a promising approach to alleviate the information overload problem in the world of movies. By analyzing movie attributes and content, it enables users to explore a tailored selection of films that align with their preferences and interests. With the ability to generate recommendations without explicit user data, content-based systems provide a valuable solution for both new users and situations where user feedback is limited.

## Problem Statement

A movie recommender system serves as a valuable solution to the challenges faced by users in navigating the extensive realm of movies. With an overwhelming number of films available, it can be daunting for individuals to find movies that match their preferences. The recommender system tackles this issue by offering personalized recommendations based on user data and preferences. By analyzing factors like viewing history, ratings, and genre preferences, it provides tailored suggestions, saving users time and effort in their search for enjoyable movies. Furthermore, it promotes movie diversity by introducing users to films they may not have otherwise discovered, expanding their cinematic horizons. This personalized approach enhances user satisfaction, engagement, and the overall movie-watching experience. By delivering accurate and relevant recommendations, the recommender system addresses information overload, lack of personalization, limited movie awareness, and the challenge of discovering new and diverse films. Ultimately, a well-designed movie recommender system streamlines movie selection, helps users find films aligned with their preferences, and enriches their movie-watching journey.

## Issues

Content-based recommender systems have their own set of challenges and limitations. Here are some of the key issues related to content-based recommender systems:

1. **Limited Serendipity**: Content-based recommenders primarily rely on analyzing item features or attributes to make recommendations. While this approach is effective in suggesting similar items based on shared characteristics, it may result in a lack of serendipity or the element of surprise. These systems tend to recommend items that are too similar to what the user has already consumed, potentially limiting the discovery of new and diverse content.
2. **Overemphasis on Item Features**: Content-based systems heavily rely on item features, such as genres, actors, or keywords, to generate recommendations. However, they may not capture more nuanced aspects of user preferences, such as the emotional tone, narrative style, or thematic elements of a movie. This limitation may result in recommendations that fail to capture the user's holistic preferences and overlook important subjective factors.
3. **Limited Diversity**: Since content-based recommender systems primarily focus on item attributes, they may overlook the aspect of diversity in recommendations. They may tend to suggest items with similar characteristics, leading to a lack of variety in the recommendations. This can limit the potential for users to explore different genres, styles, or cultural perspectives.

## Objectives

The objectives of Movie Matcher, can be outlined as follows:

1. **Content Similarity**: The primary objective of a Movie Matcher is to identify and recommend movies that are similar in terms of their content characteristics. This involves analyzing movie attributes such as genres, actors, directors, keywords, and plot summaries to determine similarities and recommend movies with comparable content.
2. **Personalization based on Item Features**: Movie Matcher aims to provide personalized movie recommendations by considering the individual's preferences in terms of movie attributes. The system takes into account the user's expressed or implied preferences for specific content features (e.g., favorite genres, preferred actors) to generate recommendations that align with their stated preferences.
3. **Enhancing Movie Discoverability**: Movie Matcher to improve the discoverability of movies by suggesting relevant and lesser-known films to users. By considering the content characteristics of movies and identifying patterns or associations, the system can recommend movies that users may not have encountered before, expanding their movie repertoire.
4. **Addressing the Cold Start Problem**: The cold start problem refers to the challenge of making recommendations for new movies that lack sufficient user interaction data. Movie Matcher aims to overcome this problem by leveraging the intrinsic attributes of new movies, such as genre, plot, and keywords, to make initial recommendations before gathering user feedback.

By pursuing these objectives, Movie Matcher can provide personalized, diverse, and contextually relevant recommendations to users without explicitly relying on user data, thereby enhancing their movie-watching experience and facilitating movie discovery.

**Chapter 2**

# Literature Survey

Netflix: Netflix, one of the world's leading streaming platforms, has gained popularity not only for its extensive content library but also for its movie recommendation system. While the algorithm has its advantages, it also comes with certain drawbacks. Here are the pros and cons of Netflix's movie recommender algorithm.

### Pros:

1. **Personalization**: Netflix's algorithm leverages user data, viewing history, and ratings to provide personalized recommendations, enhancing the user experience by suggesting movies tailored to individual preferences.
2. **Discovery of New Content**: The algorithm introduces users to a wide range of movies they may not have otherwise discovered, expanding their movie-watching horizons and facilitating the exploration of different genres, languages, and styles.
3. **Continuous Improvement**: Netflix regularly updates and refines its recommendation algorithm, employing machine learning and data analysis to improve the accuracy and relevance of its movie suggestions over time.

### Cons:

1. **Lack of Serendipity**: The algorithm tends to prioritize similar content, resulting in a potential lack of serendipity or surprise factor in recommendations. Users may miss out on discovering unexpected or diverse movies that fall outside their usual preferences.
2. **Limited User Control**: While the algorithm provides recommendations, some users may feel a lack of control over the recommendations they receive. There may be instances where users prefer a different type of content but struggle to customize the suggestions accordingly.
3. **Cold Start Problem**: For new users or when there is limited viewing history, the algorithm may struggle to provide accurate recommendations, leading to initial inaccuracies or generic suggestions until more data is available.

IMDb: IMDb (Internet Movie Database) is a widely-used platform that offers movie information, ratings, and a recommendation system. While IMDb's movie recommender algorithm has its strengths, it also presents some challenges. Let's explore the pros and cons of IMDb's movie recommender algorithm.

### Pros:

1. **Extensive Movie Database**: IMDb boasts a vast movie database, allowing its algorithm to draw from a diverse range of films and provide recommendations across various genres and languages.
2. **User Ratings and Reviews**: IMDb's algorithm considers user ratings and reviews, providing recommendations based on collective opinions and enabling users to discover popular or critically acclaimed movies.
3. **Rich Movie Information**: IMDb's algorithm benefits from comprehensive movie information, including cast, crew, plot summaries, and genres, ensuring that recommendations align with specific preferences and interests.

### Cons:

1. **Lack of Personalization**: IMDb's algorithm primarily relies on movie attributes and user ratings, potentially resulting in recommendations that lack personalization and fail to cater to individual tastes or viewing history.
2. **Limited Diversity**: The algorithm may favor popular or mainstream movies, overlooking lesser-known or independent films, which can restrict the diversity of recommendations and limit exposure to niche or underappreciated content.
3. **Limited Contextual Understanding**: IMDb's algorithm may struggle to capture nuanced aspects of movie preferences, such as emotional tone, narrative style, or thematic elements, leading to recommendations that fail to account for subjective tastes and preferences.

Amazon Prime: Amazon Prime, a popular streaming platform, utilizes a sophisticated movie recommender algorithm to enhance the user experience. While the algorithm has its advantages, it also presents certain drawbacks. Let's delve into the pros and cons of Amazon Prime's movie recommender algorithm.

### Pros:

1. **Tailored Recommendations**: Amazon Prime's algorithm analyzes user viewing history, ratings, and preferences to deliver personalized movie recommendations. This level of customization ensures that users are presented with movies that align with their individual tastes.
2. **Cross-Category Suggestions**: The algorithm extends beyond movies and offers cross-category recommendations, suggesting related content such as TV shows, documentaries, or original series, enabling users to explore a wider range of entertainment options.
3. **User Feedback Integration**: Amazon Prime's algorithm takes into account user feedback, including ratings and reviews, to fine-tune recommendations. This dynamic feedback loop improves the accuracy and relevance of subsequent movie suggestions.

### Cons:

1. **Overemphasis on Popularity**: The algorithm tends to prioritize popular and mainstream movies, potentially overshadowing lesser-known or independent films. This can limit the exposure to diverse and niche content, leading to a lack of variety in recommendations.
2. **Limited Discovery of New Content**: While the algorithm excels in personalized suggestions, it may occasionally fall short in introducing users to novel and unexplored movies. Users may miss out on discovering hidden gems or lesser-known titles outside their established preferences.

**Chapter 3**

# Implementation

## Software Requirements and Analysis:

Requirements of a Content-Based Movie Recommendation System:

1. Movie Data Management:
   * Maintain a comprehensive database of movies with relevant attributes such as genres, actors, directors, and keywords.
   * Regularly update the movie database to include new releases and remove outdated information.
2. Content Analysis:
   * Implement algorithms to analyze movie attributes and determine content similarities.
   * Consider attributes like genres, actors, directors, and keywords to identify patterns and associations.
3. Personalized Recommendations:
   * Generate personalized movie recommendations based on the user's preferences and content analysis.
   * Prioritize recommendations based on relevance and user preferences.
4. Recommendation Display:
   * Present movie recommendations to users in a user-friendly interface.
5. Performance:
   * Ensure the system generates recommendations within an acceptable response time, even with a large dataset.
6. Usability:
   * Design an intuitive and user-friendly interface for easy navigation and interaction.
7. Scalability:
   * Develop the system to handle increasing numbers of users and movies without compromising performance.
   * Design the architecture to accommodate future growth and scalability requirements.
8. Integration:

* Integrate with external APIs or data sources to access movie information and additional features.
* Ensure seamless integration with other platforms or services, if required.

1. Maintenance and Updates:
   * Plan for regular maintenance and updates of the system to address bug fixes, improve performance, and introduce new features.

## Hardware and Software Requirements

**Hardware:**

**Client:** A modern computer/mobile device with access to high speed internet connection.

**Developer:**

* Processor: A modern processor with multiple cores, such as Intel Core i5/AMD Ryzen 5 or higher, is required.
* Memory (RAM): A minimum of 8 GB of RAM is recommended. For larger datasets or more resource-intensive algorithms, 16 GB or more may be required.
* Storage: Sufficient storage space is needed to store the dataset, algorithms, and any auxiliary files. The storage capacity required depends on the size of the dataset. A minimum of 500 MB of storage space is required for the project excluding the libraries and development tools.

**Software:**

**Client:** A modern web browser such as Microsoft Edge, Firefox or Chrome should be present on the device for accessing the webapp.

**Developer:**

* Operating System: Windows 10/11, MacOS or Linux.
* Python3: Install the latest version of Python, which is compatible with your operating system. Python provides a rich ecosystem of libraries and frameworks for developing recommendation systems.
* Integrated Development Environment (IDE): Choose an IDE that suits your preferences and facilitates Python development. Popular options include PyCharm, Visual Studio Code, and Jupyter Notebook.
* Libraries and Frameworks:
  + NumPy: A fundamental library for numerical computations in Python.
  + Pandas: Used for data manipulation and analysis, especially for handling structured data.
  + Scikit-learn: Provides various machine learning algorithms and tools for model training and evaluation.
  + . Python pickle is a module that allows you to serialize and deserialize Python objects, enabling easy storage and retrieval of complex data structures.
  + NLTK (Natural Language Toolkit): NLTK is a powerful Python library used for natural language processing (NLP) tasks, providing a wide range of tools and resources for tasks such as tokenization, stemming, part-of-speech tagging, and more.
  + Streamlit: Streamlit is an open-source Python library used to create interactive web applications for data science and machine learning tasks. It simplifies the process of building and deploying data-driven applications by seamlessly converting Python scripts into web interfaces.
* A modern web browser such as Microsoft Edge, Firefox or Chrome with internet connection for running Jupyter Notebook and Streamlit app and API requests.

## Dataset

The TMDB (The Movie Database) dataset is a comprehensive collection of movie-related information that offers a wealth of data points, providing an in-depth glimpse into the world of cinema.

**Understanding the Dataset**: The TMDB dataset comprises a diverse range of attributes that capture essential movie details. It includes information such as budget, genres, homepage, keywords, original language, original title, overview, popularity, production companies, production countries, release date, revenue, runtime, spoken languages, status, tagline, title, vote average, vote count, cast, and crew. This extensive set of attributes provides a comprehensive overview of each movie's key aspects.

The following attributes are present in the dataset:

**budget**: This attribute provides information about the total budget in production of the movie.

**genres**: This attribute tells us about the different genre of the movie.

**homepage**: This attribute contains the link to the homepage of the movie, if any.

**keywords**: This attribute contains the list of keywords which are related to the movie. These words are used to describe what the movie is about in brief.

**original** **language**: This attribute tells us about the original production language of the movie.

**original** **title**: This attribute gives us the original title, that is what the movie was called in original production language.

**overview**: This attribute gives us a quick and brief description about the plot of the movie, that is what the movie is about and brief descriptions of the main characters.

**popularity**: This attribute measures the popularity of the movie and gives it a popularity score.

**production** **companies**: This attribute provides information about all the production companies involved in making the movie.

**production** **countries**: This attribute provides information about all the countries in which the movie has been made.

**release** **date**: This attribute tells us about the release date of the movie.

**revenue**: This attribute tells us about the total revenue generated by the movie.

**runtime**: This attribute tells us about the total running time of the movie.

**spoken** **languages**: This attribute gives the list of languages spoken in the movie.

status: This attribute provides information about the status of the movie, that is, whether it is still in production or it has been released.

**tagline**: This attribute gives the tagline of the movie, if any.

**title**: This attribute gives the title of the movie in English.

**vote** **average**: This attribute gives the average of all the votes given by the users in rating the movie.

**vote** **count**: This attribute gives the total amount of votes given to the movie.

**cast**: This attribute gives us information of all the cast members of the movie.

**crew**: This attribute gives us information of all the crew members involved in the production of the movie.

## Vectorization and Cosine Similarity

In content-based movie recommendation systems, vectorization and cosine similarity are essential techniques used for measuring the similarity between movies based on their content features. Here's an explanation of how these techniques are applied:

**Vectorization:**

Vectorization is the process of converting textual or categorical data into a numerical representation that machine learning algorithms can process. In the context of content-based movie recommendation systems, vectorization is used to transform movie features, such as genres, cast, crew, or keywords, into numerical vectors.

Techniques like CountVectorizer or TF-IDFVectorizer are commonly used for vectorization. CountVectorizer converts text data into a matrix where each row represents a movie, and each column represents a unique word or feature. The values in the matrix denote the frequency or occurrence of each word or feature in a movie. TF-IDFVectorizer assigns weights to words based on their frequency in a movie and their rarity across all movies.

**Cosine Similarity:**

Cosine similarity is a metric used to measure the similarity between two vectors. In the context of content-based movie recommendation systems, cosine similarity is applied to compare the vectorized representations of movies and determine their similarity based on content features.

Cosine similarity calculates the cosine of the angle between two vectors, where a value of 1 indicates the vectors are identical, 0 indicates no similarity, and -1 indicates they are dissimilar. By computing the cosine similarity between the vectorized representations of movies, it is possible to identify movies with similar content features.

The cosine similarity is calculated using the formula:

cosine\_similarity = dot\_product(A, B) / (norm(A) \* norm(B))

where dot\_product(A, B) is the dot product of vectors A and B, and norm(A) and norm(B) are the Euclidean norms of vectors A and B, respectively.

By employing vectorization and cosine similarity, content-based movie recommendation systems can compare movies based on their content features, identify similar movies, and make recommendations to users who have shown interest in certain movies or content characteristics.

## Dataset pre-processing

Dataset pre-processing is a crucial step in building a content-based movie recommendation system. This process involves transforming and organizing the raw movie dataset to improve its quality, structure, and suitability for recommendation algorithms. The following are key aspects of dataset pre-processing in a content-based movie recommendation system:

1. Data Cleaning: Raw movie data often contains missing values, inconsistent formats, and outliers. Data cleaning techniques are applied to handle these issues, ensuring data integrity and eliminating potential biases in the recommendation process.
2. Feature Extraction: In content-based recommendation systems, relevant features need to be extracted from the dataset. This involves selecting attributes such as genres, keywords, and movie descriptions that are essential for determining similarity and relevance between movies.
3. Text Pre-processing: Textual attributes like movie titles and overviews require pre-processing. Techniques such as tokenization, removing stop words, stemming, and lemmatization are applied to standardize and normalize text data, making it suitable for analysis and comparison.
4. Feature Encoding: Categorical features, such as genres or production countries, need to be encoded into numerical representations. This allows the recommendation algorithm to process and compare these features effectively.
5. Feature Scaling: Numeric features, such as movie popularity or runtime, often have different scales. Normalizing or scaling these features ensures that they contribute equally to the recommendation process, preventing any single feature from dominating the similarity calculations.

We have used attributes movie\_id, title, overview, genres, keywords, cast, crew for feature extraction and getting recommendations for the movie as we consider them to be most relevant for finding the similarity between different movies in the dataset.

## Cleaning the Data:

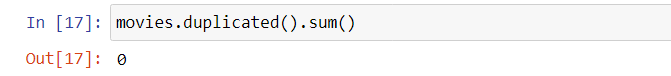
Cleaning datasets eliminates errors such as missing values, duplicates, and inconsistent formats, ensuring the accuracy and reliability of the data. We have removed the fields from the dataset where the values of our decided important parameters are omitted.

For example: if there is no information about the overview of the movie in the dataset, we remove that field of the movie entirely from the dataset.

Graphical user interface, text, application

Description automatically generated

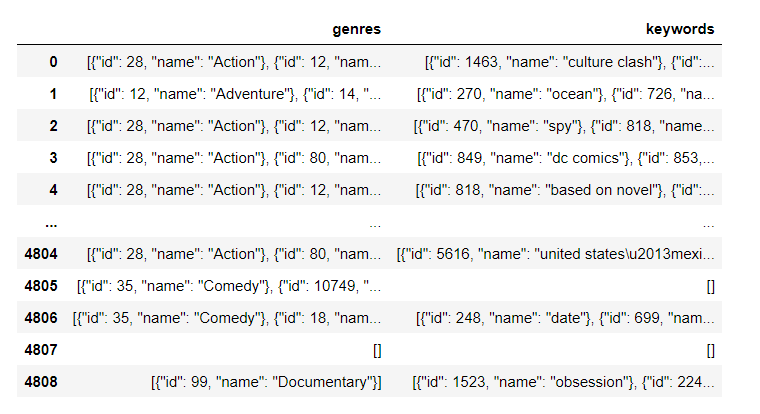
We have also checked for any duplicate entry of movies in the dataset.

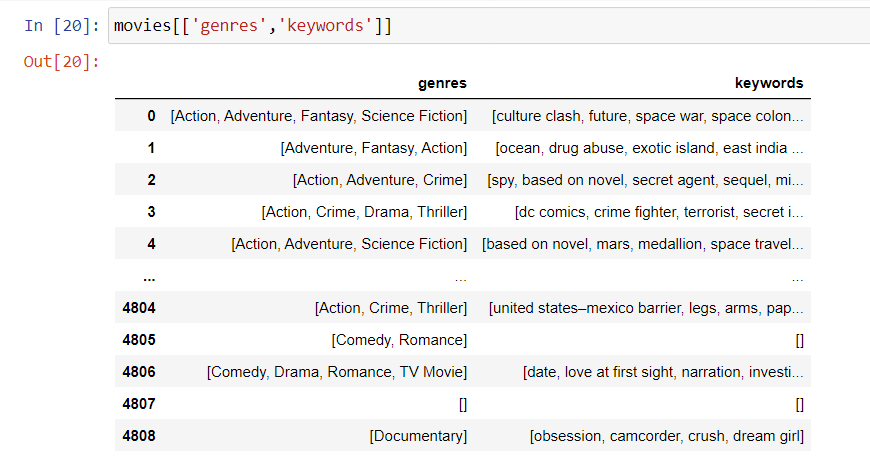


There are multiple genre and keywords for a particular movie. They are important fields that are required for feature extraction and are currently present in complex dictionary format and need to be converted from the dictionary format to a list format easy implementation later. This can be achieved using the following function given below.

**convert():** The function convert() assumes that the input parameter obj represents a list of dictionaries, where each dictionary has a key 'name'. It extracts the values associated with the 'name' key from each dictionary and returns them as a list of names.

This function is applied to keywords and genres field to extract required values from the list of dictionaries.

Before running the above function on genres and keywords:

After running the above function on genres and keywords:

We have also extracted top three actors from the cast field and director from the crew field, which were earlier in a complex dictionary of casts and crew as given below. We selected only top three members of the cast and only the director from the crew field because including more actors or crew members will harm the efficiency of feature extraction and in general the whole recommendation system.

Text

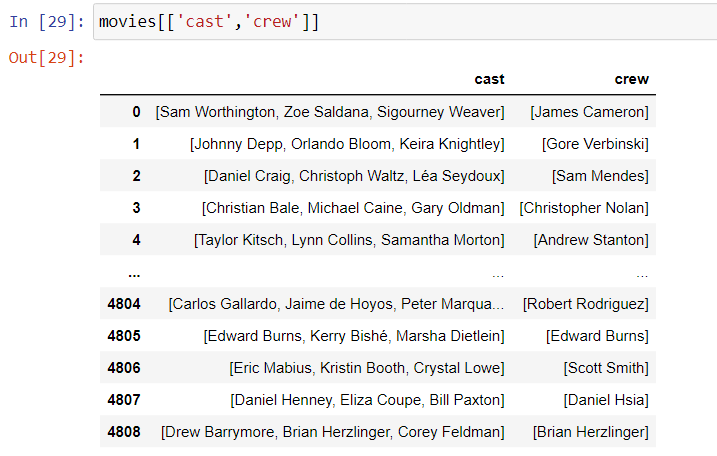
Description automatically generatedBefore cleaning, the cast and crew fields:

The following convercast() function is applied to the cast field and fetch\_director() function is applied to the crew field.

**convertcast():** The function convertcast() is a utility to extract the names from the first three dictionaries in the provided string representation of a list of dictionaries. It provides flexibility by allowing the extraction of specific values from a structured data format, such as extracting cast member names from a movie dataset.

**fetch\_director():** Thefetch\_director() function is designed to extract the name of the director from a list of dictionaries. It iterates through the list, finds the dictionary with the 'job' key set to 'Director', and retrieves the corresponding 'name' value. It then returns a list containing the director's name (or an empty list if no director is found).

The cast and crew fields after applying the above functions:



We have also converted each text in the 'overview' field in the 'movies' DataFrame into a list of words. This is useful for further text processing and analysis, such as performing sentiment analysis, word frequency analysis, or text mining on the movie overviews. Splitting the text into words allows for more granular manipulation and understanding of the textual data.

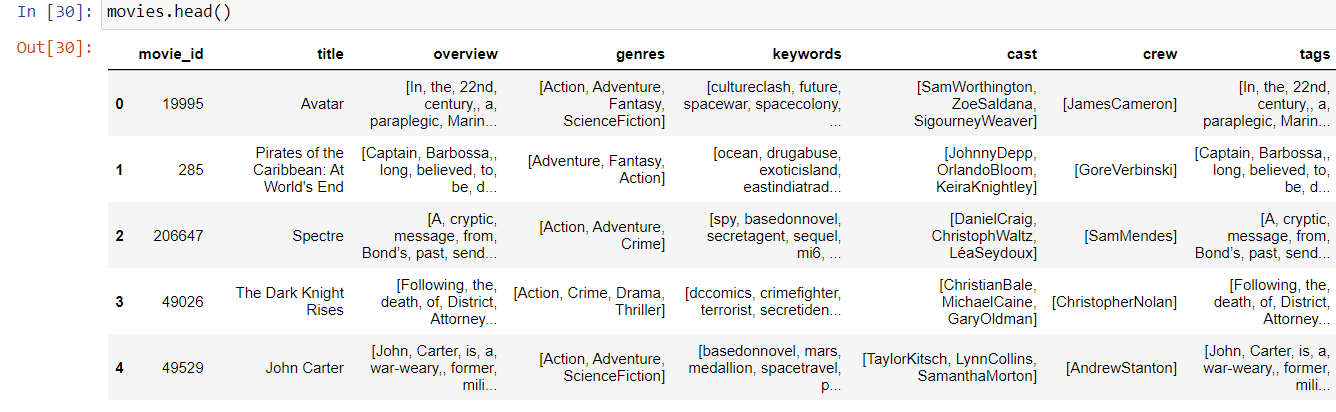
We also need apply transformations to the 'genres', 'keywords', 'cast', and 'crew' columns using lambda functions and list comprehensions.

For the 'genres' column, the code removes spaces from each genre by replacing them with an empty string. The same process is applied to the 'keywords', 'cast', and 'crew' columns, where spaces in keywords, cast member names, and crew member names are also replaced with empty strings.

By performing these pre-processing steps, the code ensures consistency in the data and removes unwanted spaces. This can be beneficial for subsequent analysis or tasks that rely on the cleaned data. Removing spaces helps in maintaining data integrity and facilitating operations such as text matching or analysis on the modified columns.

Now we can combine the overview, genres, keywords, cast, and crew columns to form a new column called tags. This can be used for feature extraction.

Now the dataset looks like this:



## Making data ready for feature extraction

The columns we need for feature extraction are only movie\_id, title, and tags. So, we have separated them to make a new DataFrame called 'new\_movies' which will be further processed and simplified to make it easier to work with for the prediction algorithm.

Firstly, we apply a transformation, that code converts a list of words in tags field for each movie into a single string representation where the words are separated by spaces. This can be useful for certain tasks or analyses that require the tags to be in a string format, such as text processing or feature extraction.

Then we apply the transformation, that makes all the words in the 'tags' column in lowercase format. This will be helpful for tasks such as text matching, grouping, or analysis, where case sensitivity is not desired and having consistent lowercase tags can simplify the processing or comparison of the data.

### Stemming the tags:

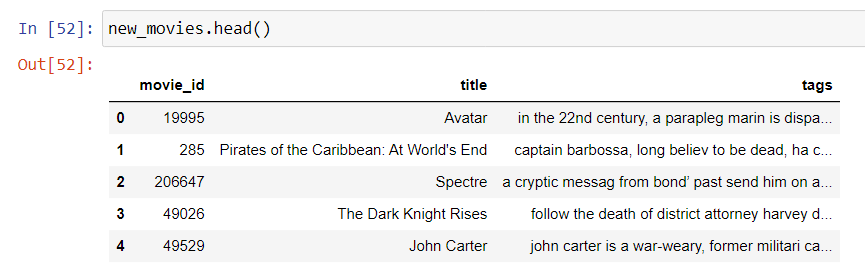
We now use the NLTK library and utilize the PorterStemmer class from nltk.stem.porter module, and also defines a function named 'stem()' that applies stemming to a given text.

The 'stem()' function takes a text as input and splits it into individual words. For each word, it applies stemming using the PorterStemmer algorithm, which reduces words to their base or root form. The stemmed words are then stored in a list. Finally, the function joins the stemmed words back into a single string using a space as the separator and returns them back as text.

We apply the 'stem' function to the 'tags' column of the DataFrame 'new\_movies'. It uses the apply() method to apply the 'stem' function to each value in the 'tags' column, replacing the original values with the stemmed versions.

By performing stemming, the code reduces words in the 'tags' column to their base or root form, which can help in standardizing and simplifying the text data. Stemming can be beneficial in text-based tasks like text mining, information retrieval, or natural language processing, where reducing words to their root form can improve text analysis, search, or classification.Top of Form

The 'new\_movies' DataFrame after applying stemming to the 'tags' column.



## Feature Extraction and Implementation of Recommendation System

### Making the Vectors

We now utilize the CountVectorizer class from the sklearn.feature\_extraction.text module. And initialize an instance of CountVectorizer with the following parameters:

- `max\_features=5000`: Specifies the maximum number of features (words) to be included in the vectorized representation. Only the top 5000 most frequent words will be considered.

- `stop\_words='english'`: Specifies that common English words (e.g., 'a', 'the', 'is') should be excluded from the analysis as stop words.

We then apply the CountVectorizer's `fit\_transform()` method to the 'tags' column of the 'new\_movies' DataFrame. It converts the text data in the 'tags' column into a matrix of token counts. The resulting matrix is then transformed into an array representation using the `toarray()` method.

In summary, we use CountVectorizer to convert the textual 'tags' data into a numerical matrix representation. It limits the number of features to the top 5000 most frequent words and excludes common English stop words. The resulting matrix, stored in the 'vectors' variable, will be further used for analysis, modeling, or similarity calculations in the content-based movie recommendation system.

### Applying Cosine Similarity to Vectors

After creation of vectors in 'vectors' array we import the `cosine\_similarity()` function from the `sklearn.metrics.pairwise` module. And apply the `cosine\_similarity()` function to the 'vectors' matrix.

The `cosine\_similarity()` function calculates the cosine similarity between vectors in the 'vectors' matrix. Cosine similarity is a measure of similarity between two vectors that takes into account the angle between them, rather than the magnitude. It is commonly used in content-based recommendation systems to measure the similarity between items based on their feature vectors.

By calculating the cosine similarity between the feature vectors, the code enables the content-based movie recommendation system to measure the similarity between movies based on their 'tags' data. This similarity information can then be used to make recommendations by identifying movies with high similarity scores.

The resulting similarity matrix is stored in the 'similarity' variable. Each entry in the similarity matrix represents the cosine similarity between the corresponding pair of vectors in the 'vectors' matrix.

Graphical user interface, application

Description automatically generatedText

Description automatically generated with medium confidenceThe features involved in making the 'similarity' matrix and 'similarity' matrix are as follows:

### Exporting the final dataset and similarity matrix by pickling

We now utilizes the pickle module to perform pickling, a process of serializing Python objects for storage or transmission.

First we convert 'new\_movies' DataFrame into dictionary format and pickle it using pickle.dump() function in binary format. The pickled object, which is the dictionary representation of 'new\_movies', and is stored in the 'movie.pkl' file.

Then, the pickle.dump() function is used to pickle the 'similarity' matrix in binary matrix. The pickled object, which is the 'similarity' matrix, is then stored in the 'similarity.pkl' file.

By pickling these objects, the code allows for saving the 'new\_movies' DataFrame and the similarity matrix as binary files. This enables easy storage and later retrieval of the objects, preserving their state and structure. The pickled files can be used in future sessions or shared with others for further analysis or integration into a content-based movie recommendation system.

## Making WebApp using Streamlit

Streamlit is an open-source Python library used for building interactive and customizable web applications for data science and machine learning projects. With Streamlit, developers can quickly create user-friendly and visually appealing applications without the need for extensive web development knowledge.

Streamlit simplifies the process of turning data scripts into shareable and interactive applications. It offers an intuitive and declarative API that allows developers to effortlessly add elements such as sliders, buttons, plots, and text inputs to create dynamic user interfaces.

One of the key advantages of Streamlit is its ability to automatically update the application in real-time whenever the underlying code is modified. This enables fast iteration and easy exploration of different visualizations or model outputs.

Streamlit also provides seamless integration with popular Python libraries such as Pandas, Matplotlib, and Scikit-learn, making it convenient to incorporate data analysis and machine learning functionalities into the application.

Overall, Streamlit is a powerful tool that empowers data scientists and developers to create interactive and engaging web applications with minimal effort, facilitating effective data exploration, sharing of insights, and showcasing machine learning models to a wider audience.

### Main components of the WebApp:

similarity DataFrame: It is created by imported into the webapp using the 'similarity.pkl' file. It is used in the recommender() function to make recommendations on user inputs.

movies DataFrame: It is created by importing into the webapp the 'movie.pkl' file. It is used by the recommender() function, fetch\_poster() function and other UI elements of the webapp that need to access the DataFrame.

fetch\_poster() function: The fetch\_poster() function takes a movie ID as input and makes a GET request to The Movie Database (TMDB) API to retrieve information about the movie. It appends the movie's poster path to the base URL of TMDB's image repository and returns the URL of the movie poster. This function is used to fetch the poster image for a given movie which will be displayed by the webapp.

recommender() function: The recommender() function takes a movie title as input. It first retrieves the index of the movie in the 'movies' DataFrame. It then calculates the similarity scores between the specified movie and all other movies using the 'similarity' matrix. Based on these similarity scores, it selects the top 15 similar movies. For each selected movie, it retrieves the movie ID from the 'movies' DataFrame and adds the movie title and corresponding poster URL to separate lists. Finally, it returns two lists: recommended\_movies (containing the titles of recommended movies) and recommended\_movies\_poster (containing the URLs of their poster images).

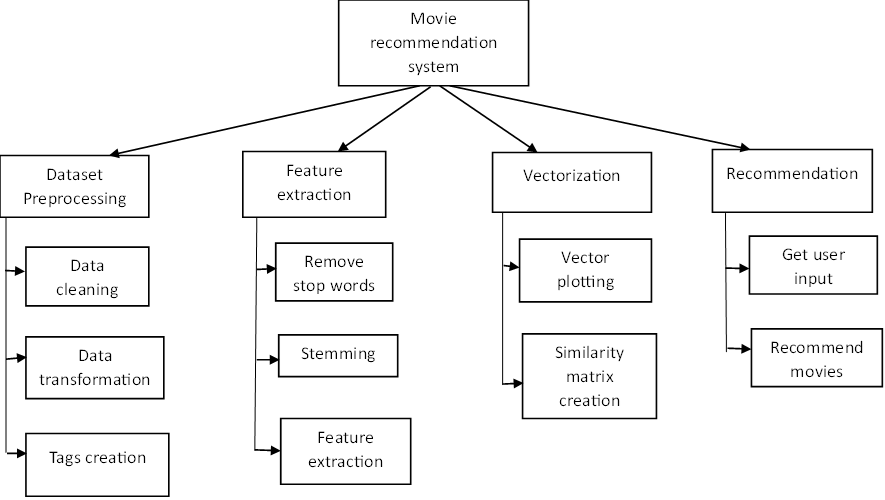
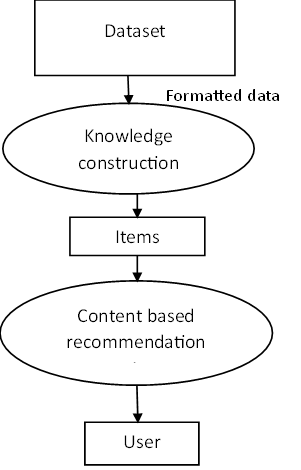


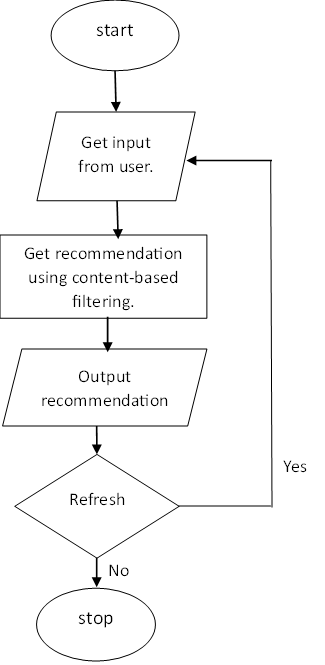
Figure 3.1 Modules implemented in Movie Matcher



Diagram

Description automatically generatedFigure 3.2 Dataflow Diagram

Figure 3.3 Use Case Diagram

Figure 3.4 Working Flow of Movie Matcher

**Chapter 4**

# Result

The Movie Matcher movie recommendation web application is the result of a project aimed at providing users with personalized movie recommendations based on their input. The app utilizes machine learning techniques and data from The Movie Database (TMDB) to offer movie suggestions like the ones users have watched and enjoyed.

The application allows users to enter the names of movies they want to find matches for using a multiselect interface. Upon clicking the "Match" button, the app retrieves the movie data from a pre-processed dataset and calculates the similarity between the selected movies and other movies in the dataset. The similarity scores are then used to generate a list of recommended movies.

The recommendations are displayed in an organized and visually appealing manner. The app showcases the recommended movie titles and their corresponding poster images in a grid-like structure. The number of columns per row is determined based on the available space, ensuring optimal use of the app's layout.

The app leverages the Streamlit framework for its user interface and interactivity. Additionally, it utilizes the pickle library to store and retrieve pre-processed movie data and similarity matrices, enabling efficient and fast recommendations.

Overall, the content-based movie recommendation web application provides users with a convenient and user-friendly way to discover new movies that align with their preferences and interests. It harnesses the power of machine learning and data processing to offer tailored movie recommendations, enhancing the movie-watching experience for users.

**User Interface:**

Here the user can input the list of movies they have watched and get recommendations for them.

Graphical user interface, application, website

Description automatically generated

**Testing:**

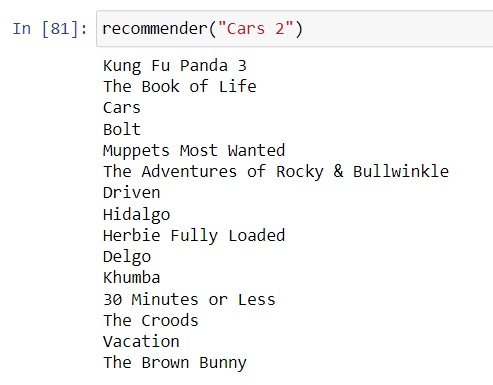
The webapp takes movies as input and gives out recommendations as output.

Input 1: The Avengers

Text

Description automatically generatedOutput 1:

Input 2: Cars 2

Output 2:

Input 3: Interstellar

Text

Description automatically generatedOutput 3:

Graphical user interface, website

Description automatically generatedOutput in WebApp: Image 1of 3

Image 2 of 3

Graphical user interface, website

Description automatically generated

Image 3 of 3

Graphical user interface, website

Description automatically generated

**Chapter 5**

# Conclusion

In conclusion, this project successfully developed a content-based movie recommendation system using machine learning techniques. The system takes user input of movie names and generates personalized movie recommendations based on similarity scores. The recommendations are displayed in an intuitive and visually appealing manner.

By leveraging the TMDB dataset and utilizing data preprocessing techniques, the system was able to extract meaningful features from movies such as genres, keywords, and cast. These features were used to calculate similarity scores and identify movies that closely align with the user's preferences.

The web application, built with Streamlit, provides a seamless user experience with its interactive interface and responsive design. Users can easily explore and discover new movies that match their interests.

Overall, the project successfully achieved its goal of creating a content-based movie recommendation system that enhances movie discovery and promotes personalized movie recommendations tailored to the user's taste and preferences.

Future Scope:In the future, several improvements can be made to enhance this movie recommendation project:

1. Incorporating user feedback: Implementing a feedback system where users can rate or provide feedback on recommended movies will allow the system to learn and improve its recommendations over time.
2. Considering more features: Expanding the feature set used for similarity calculation, such as including director information, release year, or movie ratings, can provide a more comprehensive understanding of movies and lead to more accurate recommendations.
3. Implementing a collaborative filtering approach: Incorporating collaborative filtering techniques alongside content-based methods can offer a hybrid recommendation system, leveraging both user preferences and movie attributes to generate more diverse and personalized recommendations.
4. Improving scalability: If the dataset grows significantly, optimizing the recommendation algorithm and leveraging distributed computing techniques can ensure efficient processing and scalability.

By implementing these improvements, the movie recommendation system can become more accurate, diverse, and user-centric, offering users a highly personalized and enjoyable movie discovery experience**.**

# References

* https://www.geeksforgeeks.org/
* Python Machine Learning book by Sebastian Kaschka
* https://developers.google.com/machine-learning/recommendation/content-based/basics
* <https://labelyourdata.com>
* [www.online.visual-paradigm.com](http://www.online.visual-paradigm.com)
* www.kaggle.com