MOVIE RECOMMENDATION SYSTEM (CONTENT BASED)

Ankoor Das
Asansol Engineering College
10800120060

Deep Sikdar
Asansol Engineering College
10800120050

Anurag Ghosh
Asansol Engineering College
10800120034

Siddhartha Chakraborty
Asansol Engineering College
10800120047

Contents

Abstract

In this hustling world, entertainment is a necessity for each one of us to refresh our mood and energy. Entertainment regains our confidence for work and we can work more enthusiastically. For revitalizing ourselves, we can listen to our preferred music or can watch movies of our choice. For watching favorable movies online we can utilize movie recommendation systems, which are more reliable, since searching of preferred movies will require more and more time which one cannot afford to waste. In this paper, analysing the user's behaviour we developed a recommendation system, on content-based filtering and as an algorithm "cosine similarity algorithm "have been used . The methodology and comparative results have been shown which depicts that the proposed approach shows good functionality of movie recommendation system than using particular one feature in the dataset. Hybrid approach helps to get the advantages from both the approaches as well as tries to eliminate the
drawbacks of both methods.

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty (Prof. Arnab Chakraborty / Mentor / Faculty Name) for his exemplary guidance, monitoring, and constant encouragement throughout the course of this project. The blessing, help and guidance given by him/her time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

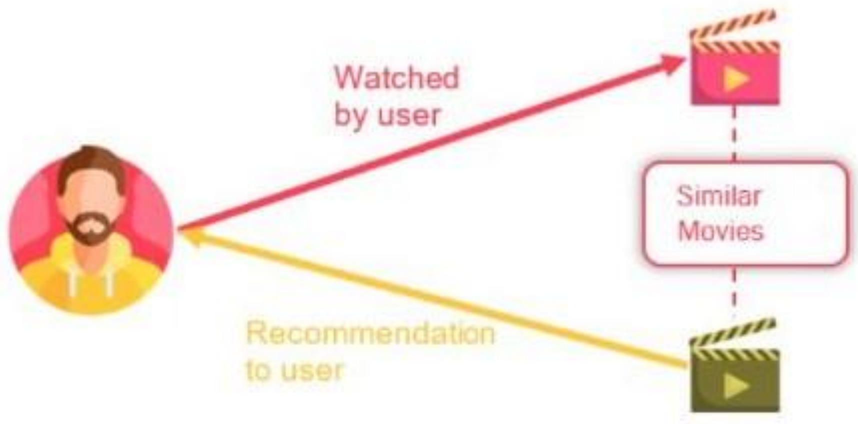
INTRODUCTION

What is a movie recommendation system:

A movie recommendation system is a fancy way to describe a process that tries to predict your preferred items based on your or people similar to you.

Types of movie recommendation system

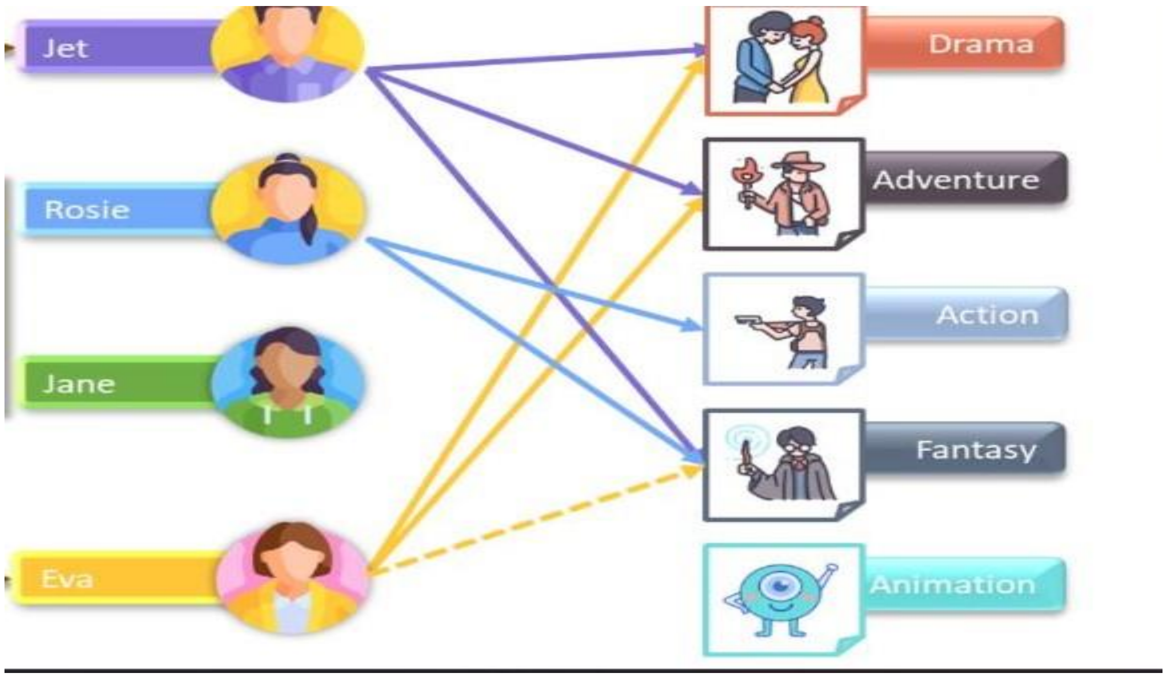
Content-Based Filtering: This type of recommendation system recommends movies based on their attributes such as genre, actors, director, plot, etc. The system uses these attributes to find other movies that are similar and recommends them to the user.



Disadvantages of Content-based Filtering

Different products do not get much exposure to the user.
Businesses cannot be expanded as the user does not try different types of products.

Collaborative Filtering: Collaborative filtering is based on the idea that people who have similar movie preferences are likely to have similar preferences for other movies. The system looks for patterns in user ratings and recommends movies that users with similar preferences have enjoyed.



Jet likes Drama, Adventure, and Fantasy-based movies.
Rosie likes Action and Fantasy-based movies.
Eva likes Drama and Adventure-based movies.
From the above data, we can say that Eva is highly correlated to Jet.
Thus, we can recommend her Fantasy movies as well.

Hybrid Recommendation System: A hybrid recommendation system combines content-based filtering and collaborative filtering to provide a more accurate and personalized recommendation. It uses both the user's past behavior and the movie's attributes to generate recommendations.

Knowledge-Based Recommendation System: Knowledge-based recommendation systems use a set of rules to recommend movies to the user based on their preferences. These systems do not require user data and rely on the user's input to generate recommendations.

Demographic-based Recommendation System: Demographic-based recommendation systems use demographic data such as age, gender, location, and language to recommend movies to the user.

Context-based Recommendation System: Context-based recommendation systems recommend movies based on the user's current context, such as time of day, day of the week, and location.

Popularity-based Recommendation System: Popularity-based recommendation systems recommend movies that are currently popular among the users or in the market.

Each of these recommendation systems has its own advantages and disadvantages, and some systems work better for certain types of users or situations than others

Project Scope

In this project we have a shortened ‘Movie Recommendation System-Content Based’ Dataset from Kaggle. The main objective of a movie recommendation system using cosine similarity algorithm is to provide personalized movie recommendations to users based on their preferences and past viewing behavior.

The system uses the cosine similarity algorithm to calculate the similarity between the user's movie preferences and those of other users in the system. This is done by analyzing the historical data of user's movie ratings and reviews.

The output of the system is a list of recommended movies that the user may enjoy, based on the similarity of their movie incorporating additional features such as genre, keywords, overview, and cast ,crew etc. It provides more relevant and diverse recommendations.

The ultimate goal of the movie recommendation system is to improve user satisfaction and engagement, by providing personalized and relevant movie recommendations that align with their interests and preferences.

Methodology for Content Based Movie Recommendation System

Methodology for content based movie recommendation system by cosine similarity-

The methodology for building a content-based movie recommendation system using cosine similarity algorithm typically involves the following steps:

Data Collection: Collect a dataset of movies along with their features such as genre, cast, director, synopsis, and other metadata.

Data Preprocessing: Clean the dataset by removing missing values, duplicates, and irrelevant features.

Feature Extraction: Extract the relevant features from the dataset using techniques such as Natural Language Processing (NLP), text processing, and feature engineering.

Vectorization: Convert the extracted features into a vector space using techniques such as TF-IDF (term frequency-inverse document frequency).

Cosine Similarity Calculation: Calculate the cosine similarity between the feature vectors of the movies using the cosine similarity algorithm. This measures the similarity between two movies based on the cosine of the angle between their feature vectors.

Recommendation Generation: Generate a list of movie recommendations for the user based on the cosine similarity between the user's profile and the movie feature vectors.

Evaluation: Evaluate the performance of the recommendation system using metrics such as precision, recall, and F1-score.

Project Objective

The objective of this project is to provide accurate content based movie recommendations to users. The goal of the project is to improve the quality of movie recommendation system, such as accuracy, quality and scalability of system than the pure approaches. This is done using Content Based approach by combining content based filtering. To eradicate the overload of the data, recommendation system is used as information filtering tool in social networking sites. Hence, there is a huge scope of exploration in this field for improving scalability, accuracy and quality of movie recommendation systems Movie Recommendation system is very powerful and important system. But, due to the problems associated with pure Content based approach, movie recommendation systems also can suffer with poor recommendation quality and scalability issues.

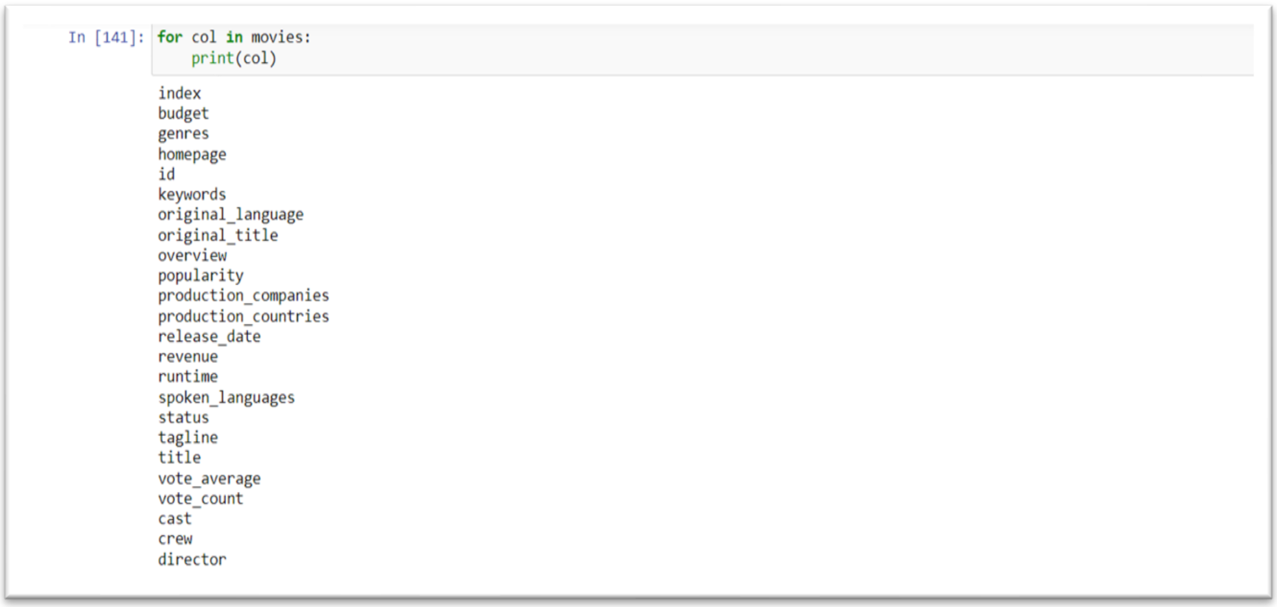
Data Description

In machine learning, data description refers to the process of analyzing and summarizing the characteristics of a dataset. The goal of data description is to gain a better understanding of the data and identify any patterns or trends that may exist within it.

Data description typically involves exploring the distribution of individual features or variables within the dataset, as well as examining the relationships between pairs or groups of features. This may involve calculating summary statistics such as mean, median, and standard deviation, as well as visualizing the data using histograms, scatterplots, or other graphical representations.

Data description is an important step in the machine learning process because it can help inform decisions about which machine learning algorithms and techniques to use, and can also help identify any potential issues or biases in the data that may need to be addressed.

Additionally, data description can be used to identify outliers or anomalies in the data that may need to be removed or treated separately in order to improve the performance of a machine learning model.



```
In [141]: for col in movies:
              print(col)

          index
          budget
          genres
          homepage
          id
          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
          crew
          director
```

<u>Description of data columns that has been used in the dataset:</u>

Index – It represents the position of a particular movie in movies dataset.

Budget- Total amount of money unvested to make that particular movie.

Genres- A **film genre** is a stylistic or thematic category for motion pictures based on similarities either in the narrative elements, aesthetic approach, or the emotional response to the film

Homepage- It is the page corresponding to a particular which depicts all information about that particular movie .

Id- Special key to discriminate one movie from the other ones

Keywords- Most relevant points of the movie are marked as keywords .

Original language- Lanuage in which movie is developed

Original title- Title of the movie is developed

Overview- Description of the movie in short

Popularity- Depicts how much that particular movie is popular among people

Production_companies- Company which is in charge to develop that movie.

Production_countries- Country in which movie is developed

Release_date- Date on which movie is going to be released

Revenue- Lifetime gross of a movie

Runtime- Total duration of the movie

Spoken_language- Language of the film

Tagline- Catch line of the movie which attract the spectators

Vote_average- Average vote the movie obtained

Vote_count- Total number of votes obtained by users who participated in poll.

Cast- All the people who act in it

Crew- A film crew is a group of people, hired by a production company, for the purpose of producing a film or motion picture.

Director- A film director is a person who controls a film's artistic and dramatic aspects and visualizes the screenplay (or script) while guiding the film crew.

Data Pre-processing

Data preprocessing is a crucial step in machine learning (ML) that involves transforming raw data into a format that can be easily understood and analyzed by machine learning algorithms. It is a vital step in ML because the quality of the data used for training the model has a significant impact on the accuracy of the final model. Data preprocessing involves several steps, including:

Data cleaning: Involves removing or correcting missing or incorrect data, identifying and removing outliers, and handling duplicates.

Data transformation: Involves scaling, normalizing, or standardizing the data to ensure that all features are on the same scale. This helps to prevent features with larger ranges from dominating the training process.

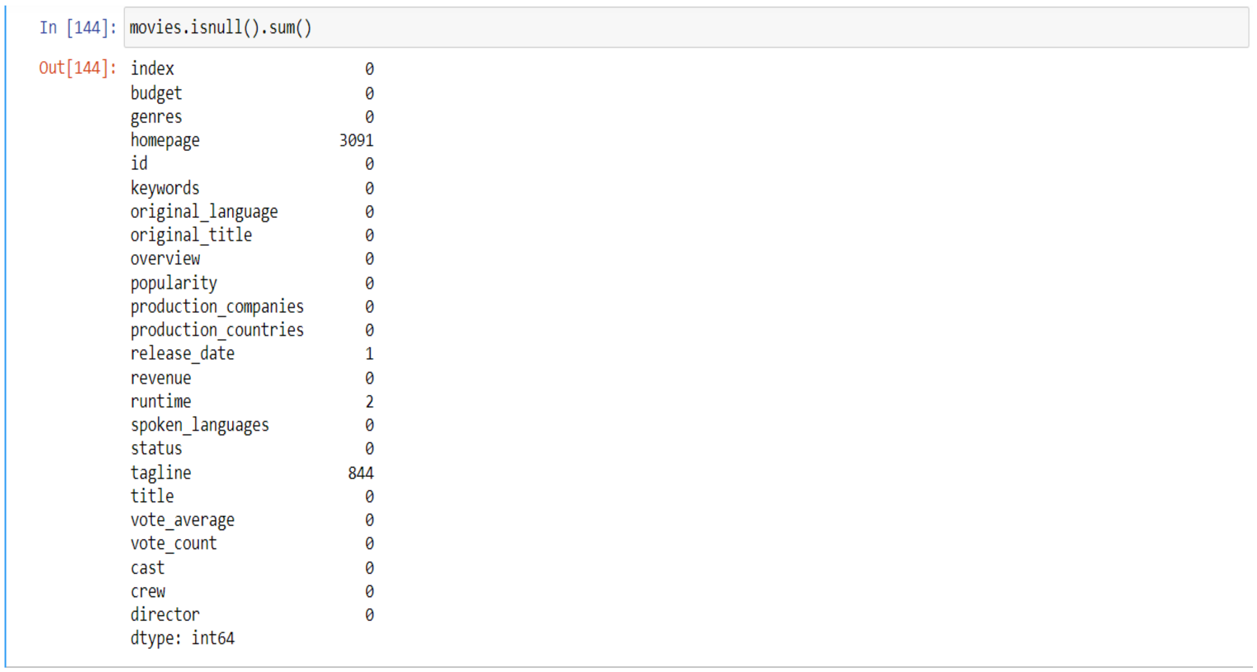
Feature selection: Involves selecting the most relevant features that can contribute to the accuracy of the final model. This reduces the dimensionality of the data and can improve the performance of the model.

Feature engineering: Involves creating new features from existing ones that can better represent the underlying patterns in the data.

Data splitting: Involves dividing the data into training, validation, and test sets to evaluate the performance of the model.

The main goal of data preprocessing is to prepare the data for modeling so that the machine learning algorithm can learn from it effectively. By using clean and normalized data, the machine learning algorithm can better identify patterns and relationships in the data, resulting in more accurate predictions.

Counting all the null values in each column-



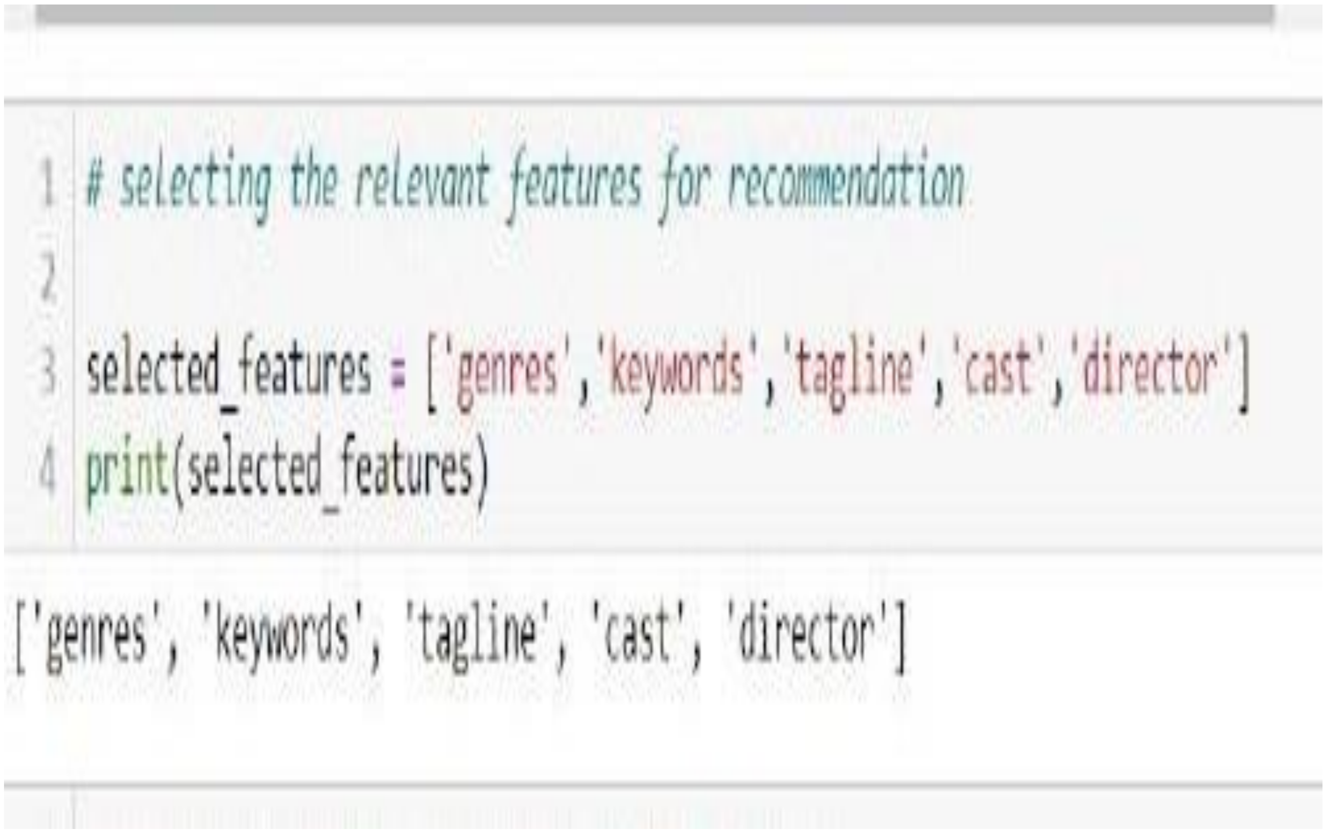
```
In [144]: movies.isnull().sum()

Out[144]: index                   0
          budget                  0
          genres                  0
          homepage             3091
          id                      0
          keywords                0
          original_language       0
          original_title          0
          overview                0
          popularity              0
          production_companies    0
          production_countries    0
          release_date            1
          revenue                 0
          runtime                 2
          spoken_languages        0
          status                  0
          tagline               844
          title                   0
          vote_average            0
          vote_count              0
          cast                    0
          crew                    0
          director                0
          dtype: int64
```

Selecting features which are essential and dropping the others:

In content-based movie recommendation systems, the system makes movie recommendations to users based on their past viewing or rating history. The system uses features of movies that are relevant to a user's interests to make recommendations. The reason why we select some features instead of all in a content-based movie recommendation system is to reduce the dimensionality of the dataset and to avoid the "curse of dimensionality."

The "curse of dimensionality" refers to the problem where the performance of machine learning algorithms deteriorates as the number of features or dimensions increases. This happens because with an increasing number of features, the amount of data required to represent the space increases exponentially, and the available data may become sparse.

Therefore, in a content-based movie recommendation system, we need to select the most relevant features for making accurate recommendations to avoid the "curse of dimensionality." We typically select the features that are most likely to have a high impact on the user's movie preferences, such as genre, actors, director, plot keywords, and ratings. By selecting only the most important features, we can reduce the number of dimensions and improve the performance of the recommendation system.

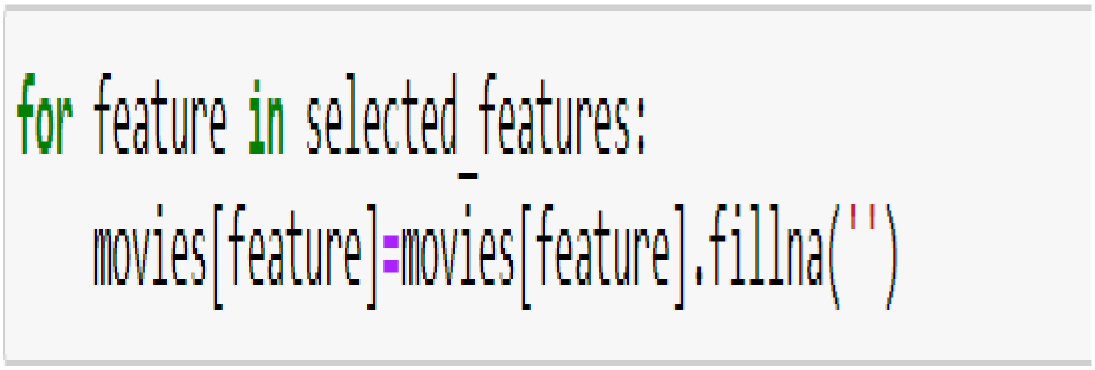


```python
# selecting the relevant features for recommendation

selected_features = ['genres','keywords','tagline','cast','director']
print(selected_features)
```

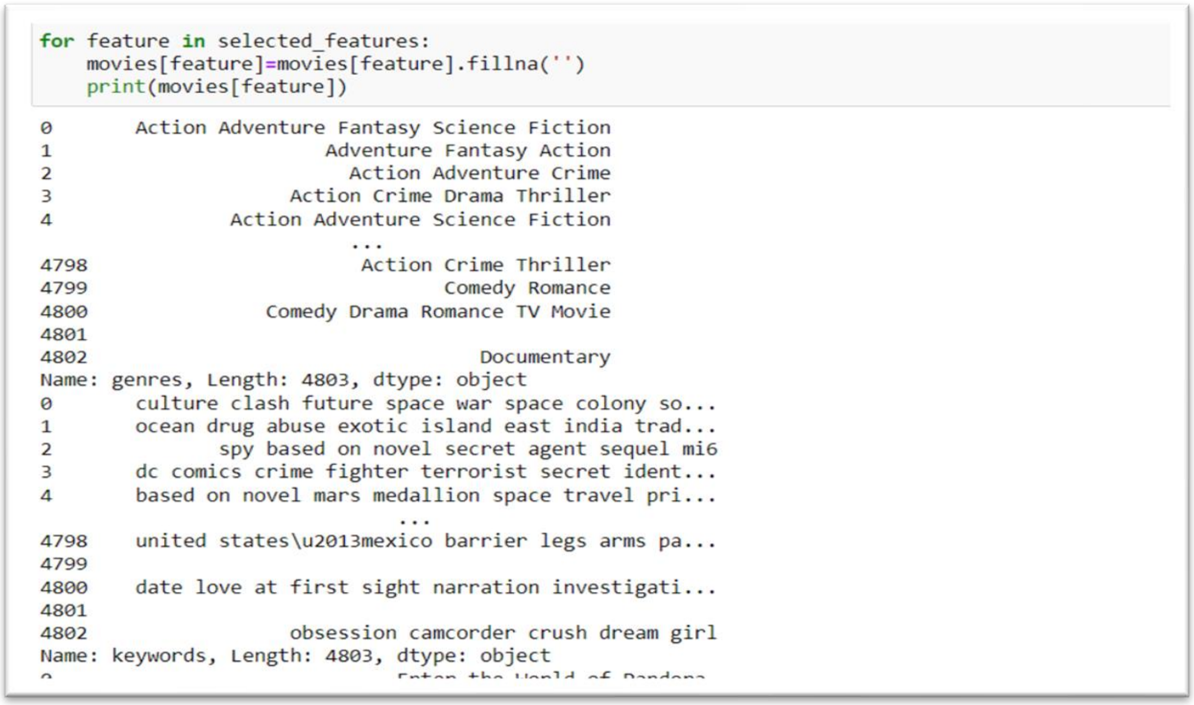
```
['genres', 'keywords', 'tagline', 'cast', 'director']
```

Replacing null values with the null string in each column of selected features:

Replacing null values with a null string is a common practice when working with data that contains missing values. In most programming languages and data analysis tools, null values are represented by a special keyword or symbol, such as "NULL" or "NaN". To replace these null values with a null string, you simply replace the null keyword with an empty string.

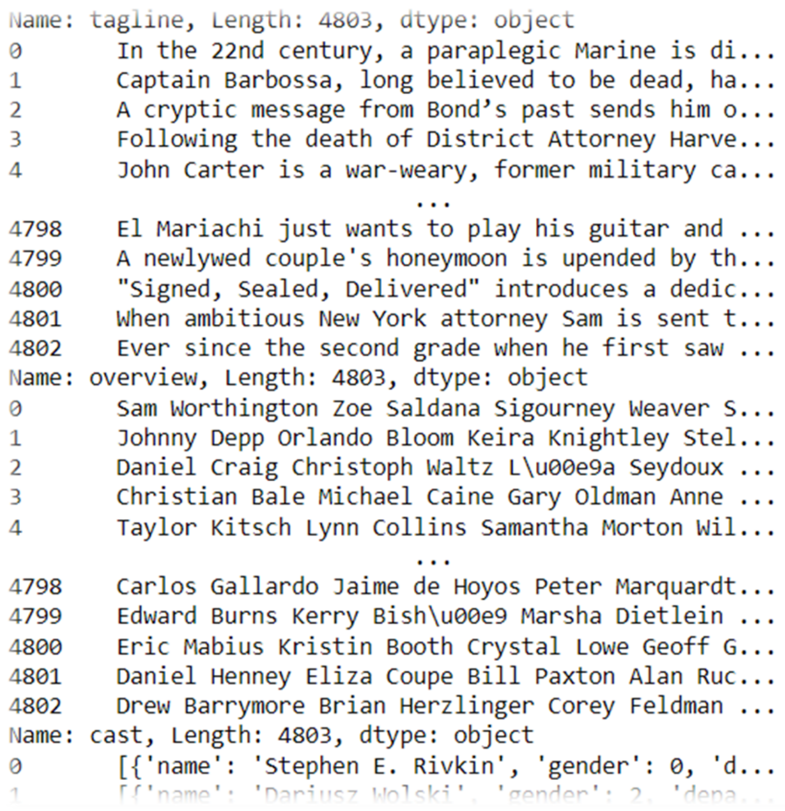


```
for feature in selected_features:
    movies[feature]=movies[feature].fillna('')
```



```
for feature in selected_features:
    movies[feature]=movies[feature].fillna('')
    print(movies[feature])
```

```
0       Action Adventure Fantasy Science Fiction
1                       Adventure Fantasy Action
2                         Action Adventure Crime
3                    Action Crime Drama Thriller
4               Action Adventure Science Fiction
                          ...
4798                       Action Crime Thriller
4799                              Comedy Romance
4800              Comedy Drama Romance TV Movie
4801
4802                                 Documentary
Name: genres, Length: 4803, dtype: object
0       culture clash future space war space colony so...
1       ocean drug abuse exotic island east india trad...
2              spy based on novel secret agent sequel mi6
3       dc comics crime fighter terrorist secret ident...
4       based on novel mars medallion space travel pri...
                              ...
4798    united states\u2013mexico barrier legs arms pa...
4799
4800    date love at first sight narration investigati...
4801
4802                obsession camcorder crush dream girl
Name: keywords, Length: 4803, dtype: object
0                            Enter the World of Pandora.
```



```
Name: tagline, Length: 4803, dtype: object
0       In the 22nd century, a paraplegic Marine is di...
1       Captain Barbossa, long believed to be dead, ha...
2       A cryptic message from Bond’s past sends him o...
3       Following the death of District Attorney Harve...
4       John Carter is a war-weary, former military ca...
                              ...
4798    El Mariachi just wants to play his guitar and ...
4799    A newlywed couple's honeymoon is upended by th...
4800    "Signed, Sealed, Delivered" introduces a dedic...
4801    When ambitious New York attorney Sam is sent t...
4802    Ever since the second grade when he first saw ...
Name: overview, Length: 4803, dtype: object
0       Sam Worthington Zoe Saldana Sigourney Weaver S...
1       Johnny Depp Orlando Bloom Keira Knightley Stel...
2       Daniel Craig Christoph Waltz L\u00e9a Seydoux ...
3       Christian Bale Michael Caine Gary Oldman Anne ...
4       Taylor Kitsch Lynn Collins Samantha Morton Wil...
                              ...
4798    Carlos Gallardo Jaime de Hoyos Peter Marquardt...
4799    Edward Burns Kerry Bish\u00e9 Marsha Dietlein ...
4800    Eric Mabius Kristin Booth Crystal Lowe Geoff G...
4801    Daniel Henney Eliza Coupe Bill Paxton Alan Ruc...
4802    Drew Barrymore Brian Herzlinger Corey Feldman ...
Name: cast, Length: 4803, dtype: object
0       [{'name': 'Stephen E. Rivkin', 'gender': 0, 'd...
1       [{'name': 'Dariusz Wolski', 'gender': 2, 'depa...
```

```
Name: cast, Length: 4803, dtype: object
0       [{'name': 'Stephen E. Rivkin', 'gender': 0, 'd...
1       [{'name': 'Dariusz Wolski', 'gender': 2, 'depa...
2       [{'name': 'Thomas Newman', 'gender': 2, 'depar...
3       [{'name': 'Hans Zimmer', 'gender': 2, 'departm...
4       [{'name': 'Andrew Stanton', 'gender': 2, 'depa...
                              ...
4798    [{'name': 'Robert Rodriguez', 'gender': 0, 'de...
4799    [{'name': 'Edward Burns', 'gender': 2, 'depart...
4800    [{'name': 'Carla Hetland', 'gender': 0, 'depar...
4801    [{'name': 'Daniel Hsia', 'gender': 2, 'departm...
4802    [{'name': 'Clark Peterson', 'gender': 2, 'depa...
Name: crew, Length: 4803, dtype: object
0           James Cameron
1          Gore Verbinski
2              Sam Mendes
3       Christopher Nolan
4          Andrew Stanton
              ...
4798     Robert Rodriguez
4799         Edward Burns
4800          Scott Smith
4801          Daniel Hsia
4802     Brian Herzlinger
Name: director, Length: 4803, dtype: object
```

Model Building

There are several machine learning models that can be used to create a content-based movie recommendation system. Here are some of them:

TF-IDF Vectorizer and Cosine Similarity: This is a commonly used approach where the content of each movie is represented as a vector using a TF-IDF vectorizer, which captures the importance of each word in the movie description. Then, the similarity between movies is calculated using cosine similarity on the vectors.

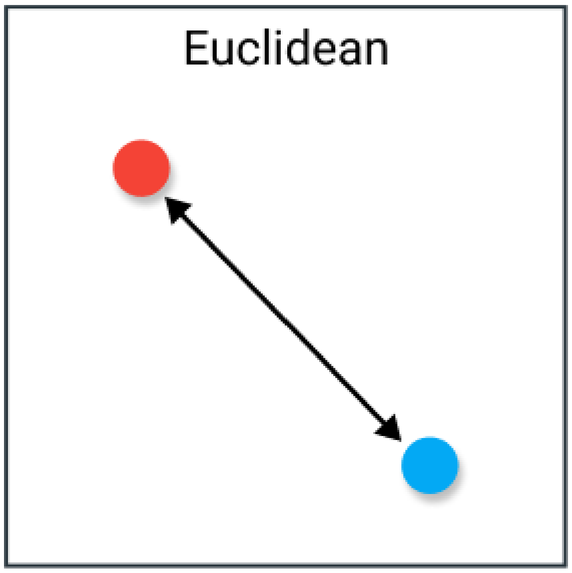
Word2Vec and Cosine Similarity: In this approach, the movie descriptions are tokenized into words and each word is represented as a dense vector using Word2Vec. Then, the movie descriptions are represented as a matrix of word vectors and similarity between movies is calculated using cosine similarity on the matrix.

Neural Networks: Another approach is to use neural networks to create a content-based recommendation system. In this approach, the movie descriptions are fed to a neural network, which learns to extract features from the text. These features can then be used to calculate similarity between movies.

Singular Value Decomposition (SVD): SVD is a matrix factorization technique that can be used to identify the latent features in movie descriptions. The movie descriptions can be represented as a matrix, which can be factorized using SVD to identify the underlying features. The similarity between movies can then be calculated based on these features.

NOTE: For most distance measures long, elaborate papers could and have been written on their use-cases, advantages, and disadvantages. I will try to cover as much as possible but may fall short! Thus, consider this article a global overview of these measures.

1.Euclidean Distance



Euclidean distance. Image by the author.

We start with the most common distance measure, namely Euclidean distance. It is a distance measure that best can be explained as the length of a segment connecting two points.

The formula is rather straightforward as the distance is calculated from the cartesian coordinates of the points using the Pythagorean theorem.

$$D(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2)}$$

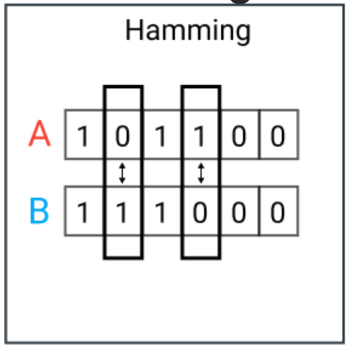
Euclidean distance

Disadvantages

Although it is a common distance measure, Euclidean distance is not scale in-variant which means that distances computed might be skewed depending on the units of the features. Typically, one needs to **normalize** the data before using this distance measure.

Moreover, as the dimensionality increases of your data, the less useful Euclidean distance becomes. This has to do with the curse of dimensionality which relates to the notion that higher-dimensional space does not act as we would, intuitively, expect from 2- or 3-dimensional space.

2. Hamming Distance



Hamming distance. Image by the author.

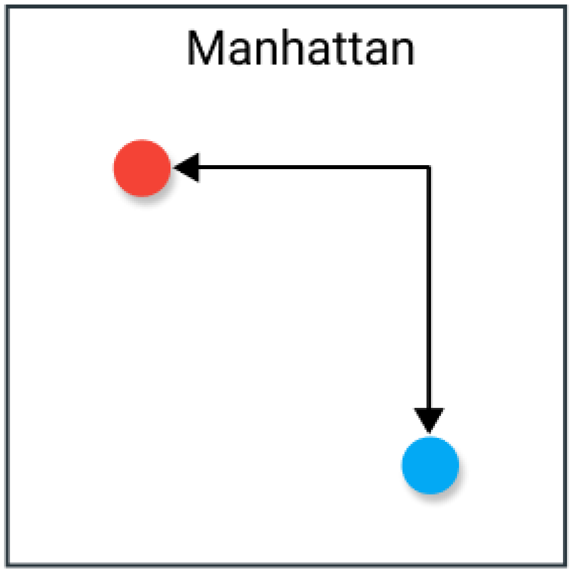
Hamming distance is the number of values that are different between two vectors. It is typically used to compare two binary strings of equal length. It can also be used for strings to compare how similar they are to each other by calculating the number of characters that are different from each other.

Disadvantages

As you might expect, hamming distance is difficult to use when two vectors are not of equal length. You would want to compare same-length vectors with each other to understand which positions do not match.

Moreover, it does not take the actual value into account as long as they are different or equal. Therefore, it is not advised to use this distance measure when the magnitude is an important measure.

3. Manhattan Distance



Manhattan distance. Image by the author.

The Manhattan distance, often called Taxicab distance or City Block distance, calculates the distance between real-valued vectors. Imagine vectors that describe objects on a uniform grid such as a chessboard. Manhattan distance then refers to the distance between two vectors if they could only move right angles. There is no diagonal movement involved in calculating the distance.

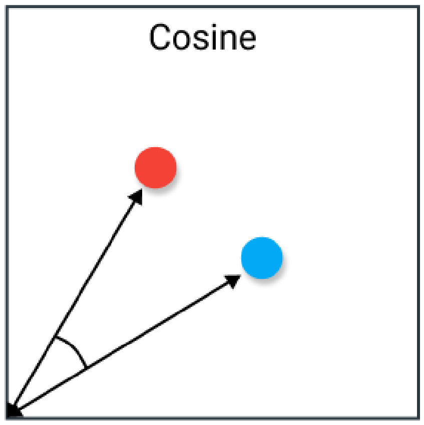
$$D(x, y) = \sum_{i=1}^{k} |x_i - y_i|$$

Disadvantages

Although Manhattan distance seems to work okay for high-dimensional data, it is a measure that is somewhat less intuitive than Euclidean distance, especially when using in high-dimensional data.

Moreover, it is more likely to give a higher distance value than Euclidean distance since it does not the shortest path possible. This does not necessarily give issues but is something you should consider

4.Cosine Similarity



Cosine distance. Image by the author.

Cosine similarity is a measure of similarity between two non-zero vectors in a multi-dimensional space. It is commonly used in various fields, including natural language processing, information retrieval, and machine learning.

The cosine similarity algorithm calculates the cosine of the angle between two vectors. The algorithm works by taking the dot product of two vectors and dividing it by the product of their magnitudes. Mathematically, it can be represented as:

$$D(x, y) = cos(\theta) = \frac{x \cdot y}{||x||\ ||y||}$$

The resulting value ranges from -1 to 1, where 1 indicates that the two vectors are identical, 0 indicates that the vectors are orthogonal (i.e., not similar), and -1 indicates that the vectors are diametrically opposite (i.e., completely dissimilar).

Cosine similarity is widely used in natural language processing, especially for document classification and retrieval. In this context, each document is represented as a vector of word frequencies, and the similarity between two documents is calculated using cosine similarity.

Some of the ***advantages*** of cosine similarity include:

- It is fast and easy to implement.
- It is a good metric for high-dimensional spaces.
- It is not affected by the magnitude of the vectors.

However, cosine similarity also has some ***limitations***, such as:

- It does not take into account the order of the terms in the vectors.
- It assumes that the vectors are linearly independent.
- It does not work well with sparse vectors.
- In practice, this means that the differences in values are not fully considered
- magnitude of vectors is not considered, merely their direction

Overall, cosine similarity is a powerful tool for measuring the similarity between vectors in various fields, including natural language processing, information retrieval, and machine learning.

Use Cases

We use cosine similarity often when we have high-dimensional data and when the magnitude of the vectors is not of importance. For text analyses, this measure is quite frequently used when the data is represented by word counts. For example, when a word occurs more frequently in one document over another this does not necessarily mean that one document is more related to that word. It could be the case that documents have uneven lengths and the magnitude of the count is of less importance. Then, we can best be using cosine similarity which disregards magnitude.

Use of cosine similarity in recommendation systems

Recommendation systems in machine learning are one such algorithm that works based on the similarity of contents. There are various ways to measure the similarity between the two contents and recommendation systems basically use the similarity matrix to recommend the similar content to the user based on his accessing characteristics.

So any recommendation data can be acquired and the required features that would be useful for recommending the contents can be taken out from the data. Once the required textual data is available the textual data has to be vectorized using the tfidfVectorizer to obtain the similarity matrix. So once the similarity matrix is obtained the cosine similarity metrics of scikit learn can be used to recommend the user.

So the cosine similarity would yield a similarity matrix for the selected textual data for recommendation and the content with higher similarity scores can be sorted using lists. Here cosine similarity would consider the frequently occurring terms in the textual data and that terms would be vectorized with higher frequencies and that content would be recommended with higher recommendation percentages. So this is how cosine similarity is used in recommendation systems.

Why to use cosine similarity algorithm over other models :

Cosine similarity is a commonly used technique in content-based recommendation systems for several reasons:

Simple and Efficient: Cosine similarity is a simple and efficient algorithm that can quickly calculate the similarity between two items based on their content features. It's computationally efficient and doesn't require much computational power to run.

Effective for High Dimensional Data: Content-based recommendation systems often deal with high-dimensional data where the number of features can be large. In such scenarios, cosine similarity is effective because it can handle large amounts of data and still provide accurate results.

Suitable for Sparse Data: Content-based recommendation systems may have sparse data where some features may be missing or not applicable. Cosine similarity is suitable for sparse data because it only considers the non-zero elements in the data, making it more robust to missing values.

Interpretability: Cosine similarity provides a measure of similarity between two items, which can be easily interpreted. This interpretability makes it easier for developers to understand how the system is making recommendations, which can be useful in debugging and improving the system.

Overall, cosine similarity is a popular choice for content-based recommendation systems because it is simple, efficient, effective, and interpretable. However, it's worth noting that there are other machine learning algorithms that can be used in recommendation systems, and the choice of algorithm depends on the specific requirements of the system and the data available.

Term Frequency (TF) and **Inverse Document Frequency (IDF) (TF-IDF) Algorithm**

In a content-based movie recommendation system, the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer is a commonly used technique for analyzing the content of movies.

The TF-IDF vectorizer is a way to represent the importance of each word in a movie description. It does this by calculating a weight for each word that reflects its frequency in the movie description, as well as its frequency across all movie

Below is the equation to calculate the TF-IDF score:

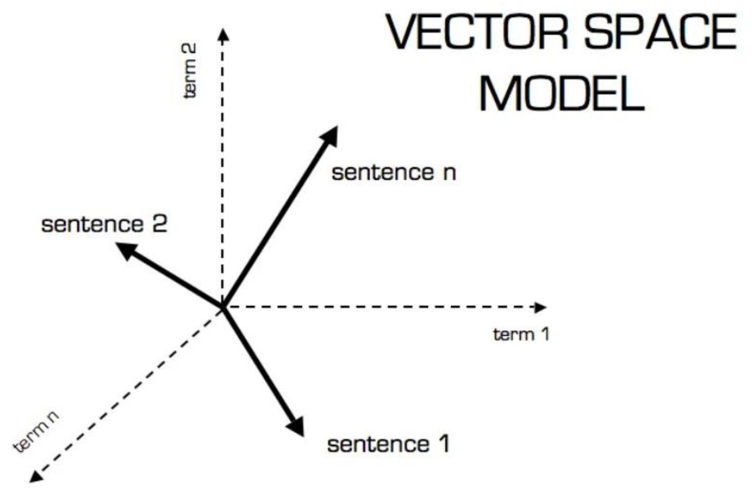
$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \times \log\left(\frac{N}{\mathrm{df}_i}\right)$$

$\mathrm{tf}_{i,j}$ = total number of occurences of i in j

df_i = total number of documents (speeches) containing i

N = total number of documents (speeches)

After calculating TF-IDF scores, we determine which items are closer to each other. This is accomplished using the **Vector Space Model** which computes the proximity based on the angle between the vectors. In this model, each item is stored as a vector of its attributes (which are also vectors) in an **n-dimensional space** and the angles between the vectors are calculated to **determine the similarity between the vectors**. Next, the user profile vectors are also created based on his actions on previous attributes of items and the similarity between an item and a user is also determined in a similar way.



Sentence 2 is more likely to be using Term 2 than using Term 1. Vice-versa for Sentence 1. The method of calculating this relative measure is calculated by taking the cosine of the angle between the sentences and the terms. The ultimate reason behind using cosine is that the **value of cosine will increase with decreasing value of the angle** between which signifies more similarity. The vectors are length normalized after which they become vectors of length 1 and then the cosine calculation is simply the sum-product of vectors.

Result of using TfidfVectorizer in content based movie Recommendation system :

The resulting TF-IDF weights create a vector for each movie, where each dimension of the vector represents a particular word in the movie description, and the value of that dimension represents the importance of that word for that movie.

The cosine similarity between any two movies can then be calculated based on the similarity between their corresponding TF-IDF vectors. This can be used to recommend movies that are similar to a given movie, based on the similarity of their descriptions.

One advantage of using TF-IDF vectorization in content-based movie recommendation systems is that it can help to identify movies that are similar in terms of their content, even if they do not share the same cast, director, or genre. However, it is important to note that the quality of the recommendations will depend on the quality of the movie descriptions and the choice of the similarity metric used to compare them.

Why to use Tfidf in content based movie recommendation system:

TF-IDF (Term Frequency-Inverse Document Frequency) is a commonly used technique in content-based movie recommendation systems for several reasons:

It can handle large amounts of text data: Movie recommendation systems often have to process a large number of text descriptions and reviews of movies. TF-IDF is designed to handle such large volumes of text data and can effectively identify the most important words in a document.

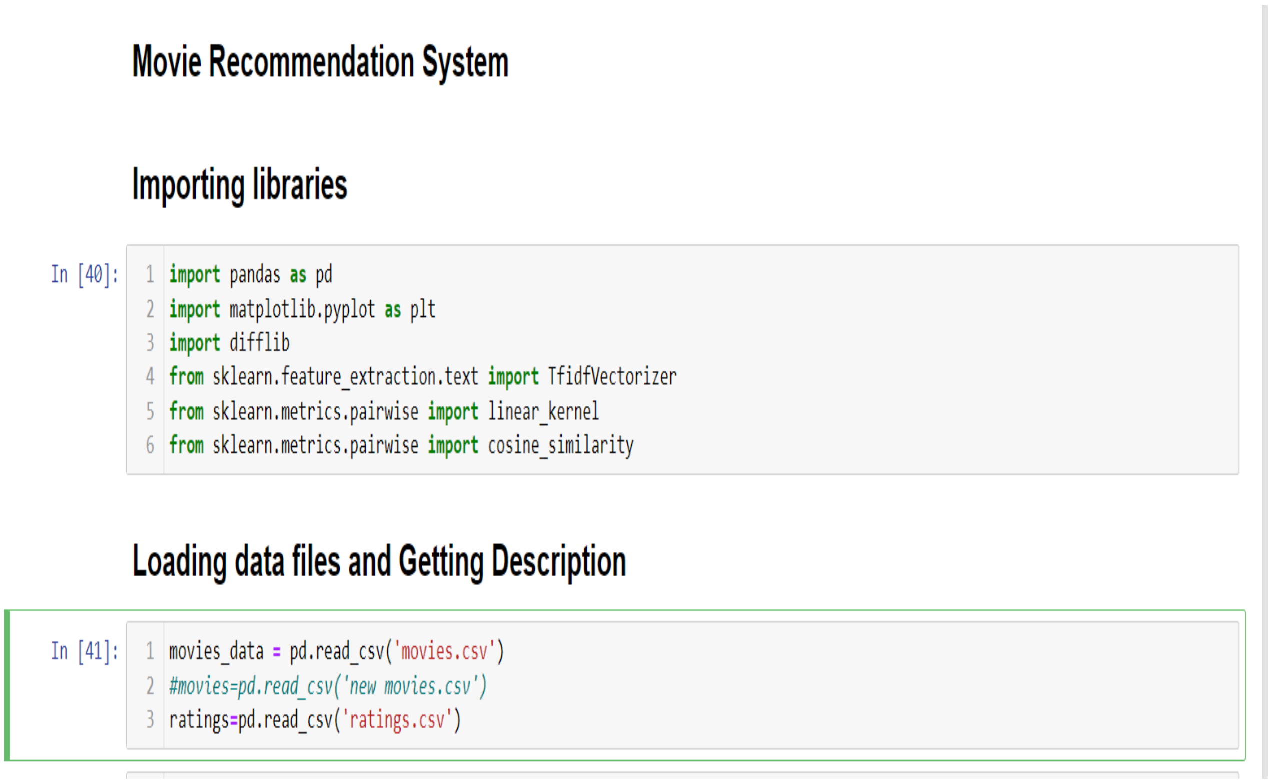
It can capture the meaning of the document: TF-IDF not only takes into account the frequency of a word in a document but also considers how important that word is in the context of the entire corpus of documents. This allows it to capture the meaning of a document more accurately and helps to identify similar documents.

It can effectively filter out noise words: Noise words are words that occur frequently in a corpus but do not carry any meaningful information. TF-IDF can effectively filter out these noise words and focus on the most meaningful words in a document.

It can scale well: TF-IDF can be easily scaled to handle large volumes of data, making it suitable for use in large-scale movie recommendation systems.

Overall, TF-IDF is a robust and effective technique for content-based movie recommendation systems and is widely used in the field.

Code



Movie Recommendation System

Importing libraries

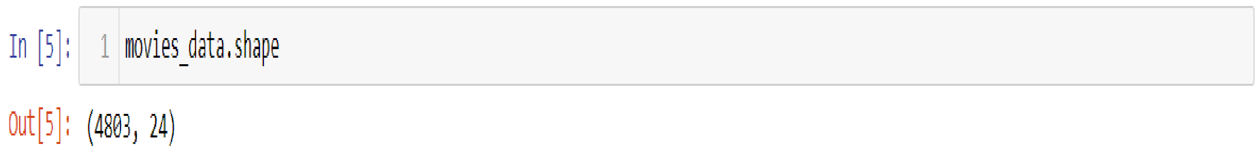
In [40]:
```
import pandas as pd
import matplotlib.pyplot as plt
import difflib
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from sklearn.metrics.pairwise import cosine_similarity
```

Loading data files and Getting Description

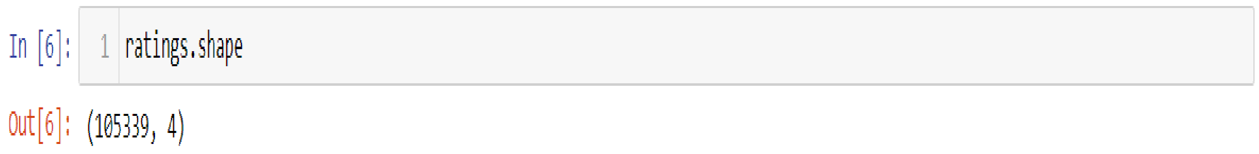
In [41]:
```
movies_data = pd.read_csv('movies.csv')
#movies=pd.read_csv('new movies.csv')
ratings=pd.read_csv('ratings.csv')
```

```
In [42]: movies_data.info()
```

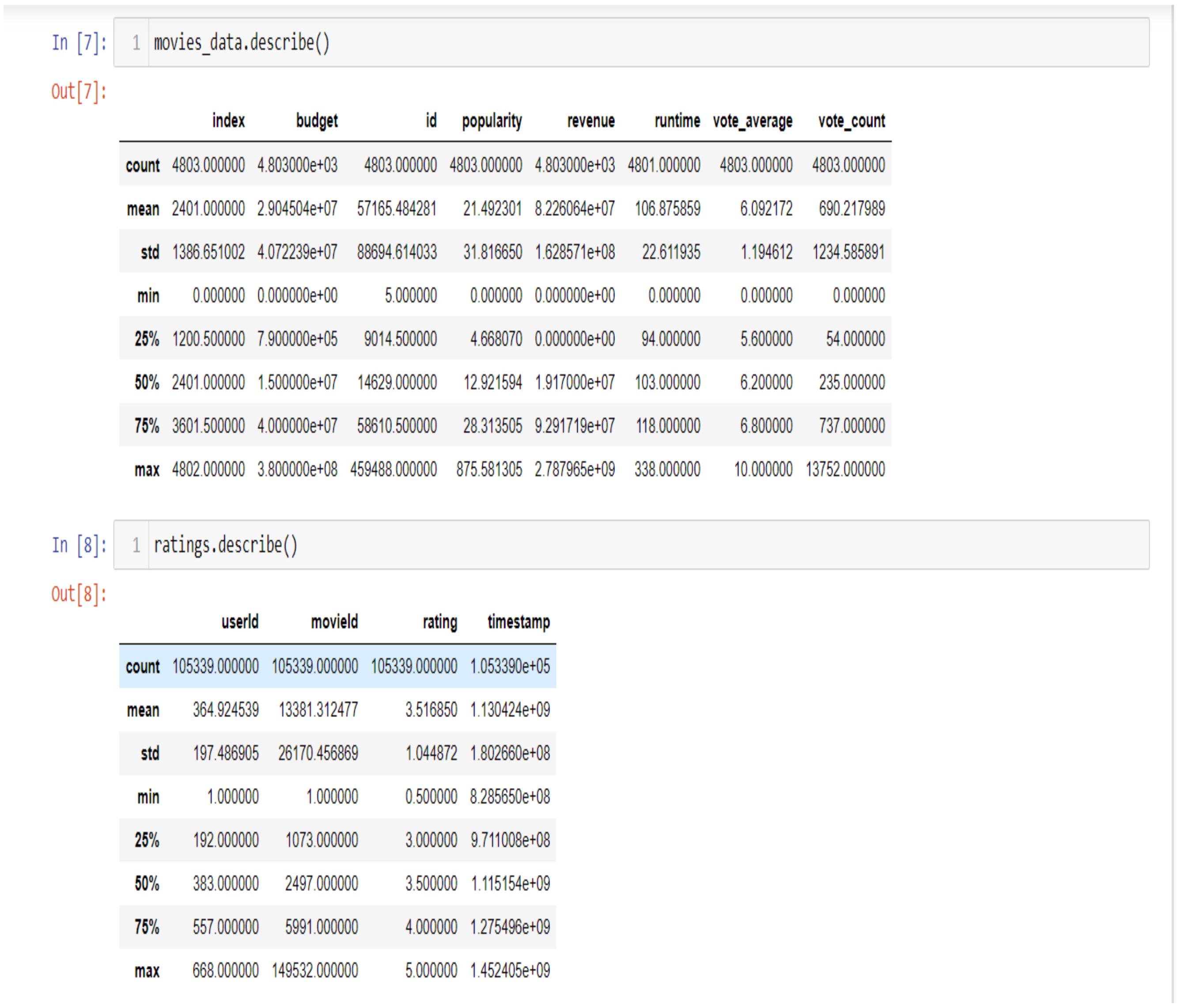
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   index                 4803 non-null   int64  
 1   budget                4803 non-null   int64  
 2   genres                4775 non-null   object 
 3   homepage              1712 non-null   object 
 4   id                    4803 non-null   int64  
 5   keywords              4391 non-null   object 
 6   original_language     4803 non-null   object 
 7   original_title        4803 non-null   object 
 8   overview              4800 non-null   object 
 9   popularity            4803 non-null   float64
 10  production_companies  4803 non-null   object 
 11  production_countries  4803 non-null   object 
 12  release_date          4802 non-null   object 
 13  revenue               4803 non-null   int64  
 14  runtime               4801 non-null   float64
 15  spoken_languages      4803 non-null   object 
 16  status                4803 non-null   object 
 17  tagline               3959 non-null   object 
 18  title                 4803 non-null   object 
 19  vote_average          4803 non-null   float64
 20  vote_count            4803 non-null   int64  
 21  cast                  4760 non-null   object 
 22  crew                  4803 non-null   object 
 23  director              4773 non-null   object 
dtypes: float64(3), int64(5), object(16)
memory usage: 900.7+ KB
```



```
In [5]: movies_data.shape

Out[5]: (4803, 24)
```



```
In [6]: ratings.shape

Out[6]: (105339, 4)
```



In [7]:
```python
movies_data.describe()
```

Out[7]:

	index	budget	id	popularity	revenue	runtime	vote_average	vote_count
count	4803.000000	4.803000e+03	4803.000000	4803.000000	4.803000e+03	4801.000000	4803.000000	4803.000000
mean	2401.000000	2.904504e+07	57165.484281	21.492301	8.226064e+07	106.875859	6.092172	690.217989
std	1386.651002	4.072239e+07	88694.614033	31.816650	1.628571e+08	22.611935	1.194612	1234.585891
min	0.000000	0.000000e+00	5.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000
25%	1200.500000	7.900000e+05	9014.500000	4.668070	0.000000e+00	94.000000	5.600000	54.000000
50%	2401.000000	1.500000e+07	14629.000000	12.921594	1.917000e+07	103.000000	6.200000	235.000000
75%	3601.500000	4.000000e+07	58610.500000	28.313505	9.291719e+07	118.000000	6.800000	737.000000
max	4802.000000	3.800000e+08	459488.000000	875.581305	2.787965e+09	338.000000	10.000000	13752.000000

In [8]:
```python
ratings.describe()
```

Out[8]:

	userId	movieId	rating	timestamp
count	105339.000000	105339.000000	105339.000000	1.053390e+05
mean	364.924539	13381.312477	3.516850	1.130424e+09
std	197.486905	26170.456869	1.044872	1.802660e+08
min	1.000000	1.000000	0.500000	8.285650e+08
25%	192.000000	1073.000000	3.000000	9.711008e+08
50%	383.000000	2497.000000	3.500000	1.115154e+09
75%	557.000000	5991.000000	4.000000	1.275496e+09
max	668.000000	149532.000000	5.000000	1.452405e+09

In [10]:
```
# from the data description we need to pre process the data before using it.
```

From the above table we can conclue that ¶

The average rating is 3.5 and minimum and maximum rating is 0.5 and 5 respectively. There are 668 user who has given their ratings for 149532 movies.

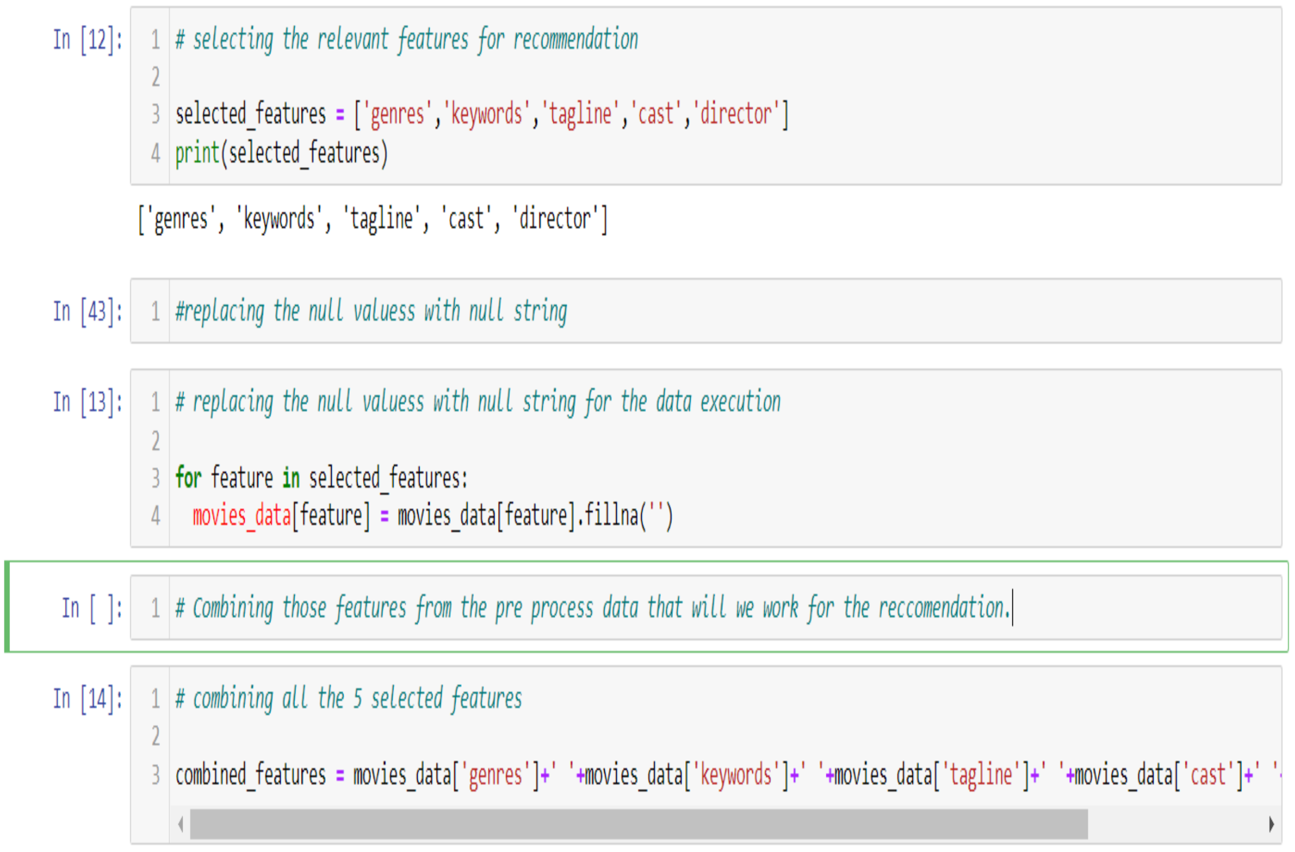
In [ ]:
```
# Pre Process the data
```

In [11]:
```
movies_data.head()
```

Out[11]:

	index	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	...	runtime
0	0	237000000	Action Adventure Fantasy Science Fiction	http://www.avatarmovie.com/	19995	culture clash future space war space colony so...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	...	162.0
1	1	300000000	Adventure Fantasy Action	http://disney.go.com/disneypictures/pirates/	285	ocean drug abuse exotic island east india trad...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	...	169.0
2	2	245000000	Action Adventure Crime	http://www.sonypictures.com/movies/spectre/	206647	spy based on novel secret agent sequel mi6	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	...	148.0
3	3	250000000	Action Crime Drama Thriller	http://www.thedarkknightrises.com/	49026	dc comics crime fighter terrorist secret ident...	en	The Dark Knight Rises	Following the death of District Attorney Harve...	112.312950	...	165.0
4	4	260000000	Action Adventure Science Fiction	http://movies.disney.com/john-carter	49529	based on novel mars medallion space travel pri...	en	John Carter	John Carter is a war-weary, former military ca...	43.926995	...	132.0

5 rows × 24 columns



In [12]:
```python
# selecting the relevant features for recommendation

selected_features = ['genres','keywords','tagline','cast','director']
print(selected_features)
```

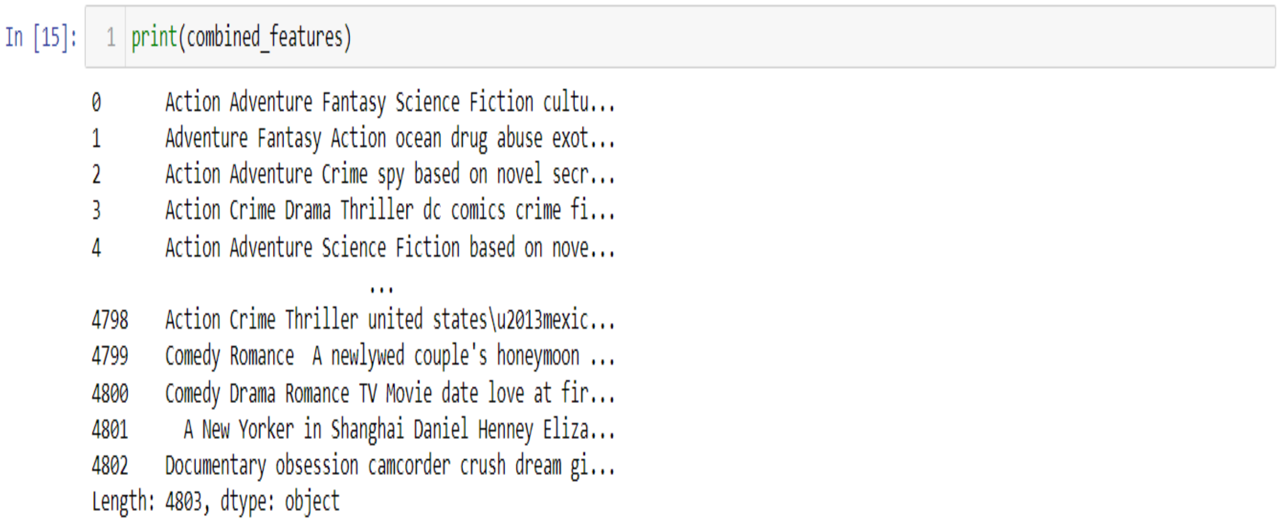
```
['genres', 'keywords', 'tagline', 'cast', 'director']
```

In [43]:
```python
#replacing the null valuess with null string
```

In [13]:
```python
# replacing the null valuess with null string for the data execution

for feature in selected_features:
  movies_data[feature] = movies_data[feature].fillna('')
```

In [ ]:
```python
# Combining those features from the pre process data that will we work for the reccomendation.
```

In [14]:
```python
# combining all the 5 selected features

combined_features = movies_data['genres']+' '+movies_data['keywords']+' '+movies_data['tagline']+' '+movies_data['cast']+' '
```



In [15]:
```python
print(combined_features)
```

```
0       Action Adventure Fantasy Science Fiction cultu...
1       Adventure Fantasy Action ocean drug abuse exot...
2       Action Adventure Crime spy based on novel secr...
3       Action Crime Drama Thriller dc comics crime fi...
4       Action Adventure Science Fiction based on nove...
                              ...
4798    Action Crime Thriller united states\u2013mexic...
4799    Comedy Romance  A newlywed couple's honeymoon ...
4800    Comedy Drama Romance TV Movie date love at fir...
4801      A New Yorker in Shanghai Daniel Henney Eliza...
4802    Documentary obsession camcorder crush dream gi...
Length: 4803, dtype: object
```

In [ ]:
```
#then we are merging the files rating amd movies description files for generating top highest rated movie
```

In [19]:
```
df=pd.merge(ratings,movies, how='left',on='movieId')
df.head()
```

Out[19]:

	userId	movieId	rating	timestamp	title	genres
0	1	16	4.0	1217897793	Casino (1995)	Crime|Drama
1	1	24	1.5	1217895807	Powder (1995)	Drama|Sci-Fi
2	1	32	4.0	1217896246	Twelve Monkeys (a.k.a. 12 Monkeys) (1995)	Mystery|Sci-Fi|Thriller
3	1	47	4.0	1217896556	Seven (a.k.a. Se7en) (1995)	Mystery|Thriller
4	1	50	4.0	1217896523	Usual Suspects, The (1995)	Crime|Mystery|Thriller

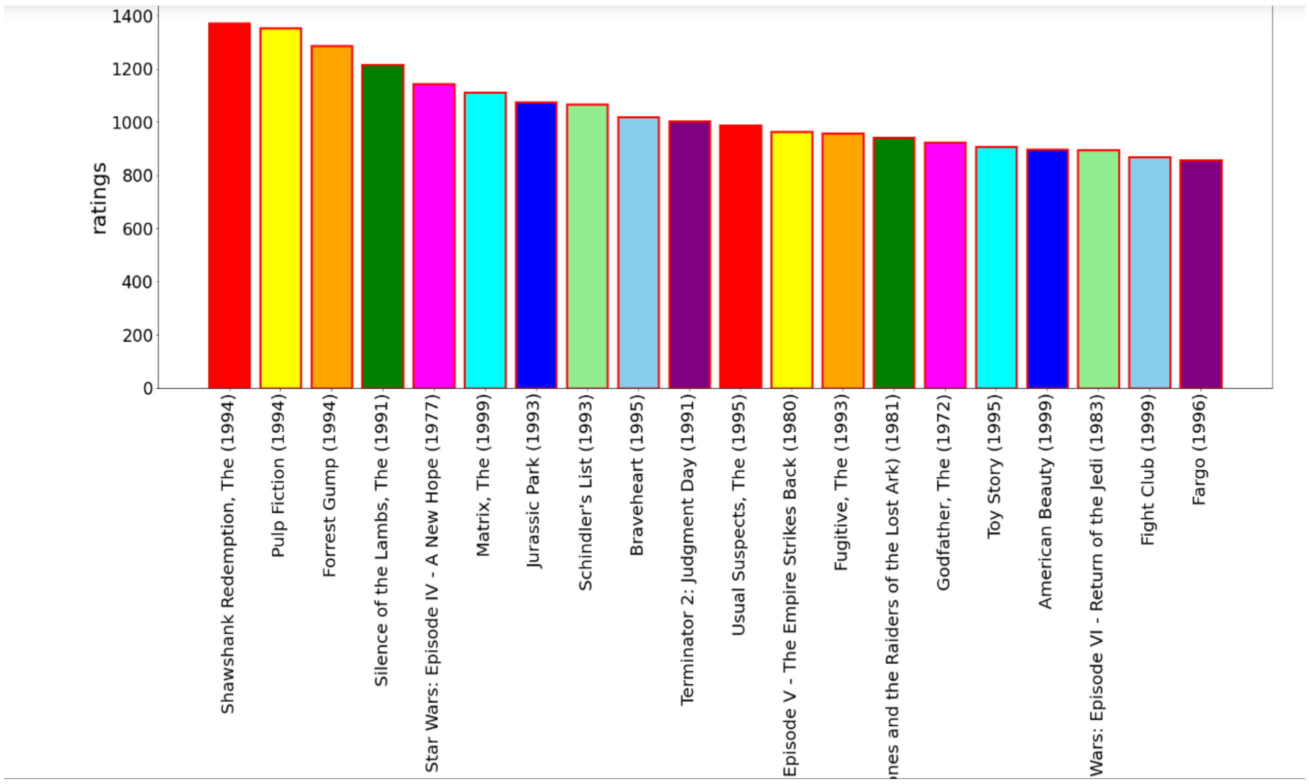
In [20]:
```
df1=df.groupby(['title'])[['rating']].sum()
high_rated=df1.nlargest(20,'rating')
high_rated.head()

```

Out[20]:

	rating
title	
Shawshank Redemption, The (1994)	1372.0
Pulp Fiction (1994)	1352.0
Forrest Gump (1994)	1287.0
Silence of the Lambs, The (1991)	1216.5
Star Wars: Episode IV - A New Hope (1977)	1143.5

In [24]:
```
plt.figure(figsize=(30,10))
plt.title('Top 20 movies with highest rating with officials',fontsize=40)
colors=['red','yellow','orange','green','magenta','cyan','blue','lightgreen','skyblue','purple']
plt.ylabel('ratings',fontsize=30)
plt.xticks(fontsize=25,rotation=90)
plt.xlabel('movies title',fontsize=30)
plt.yticks(fontsize=25)
plt.bar(high_rated.index,high_rated['rating'],linewidth=3,edgecolor='red',color=colors)
```

Out[24]:
```
<BarContainer object of 20 artists>
```



In [ ]:
```
#then we are merging the files rating amd movies description files for generating top highest
#rated movie according to the public ratings
```

In [21]:
```
genres=[]
for genre in movies.genres:

    x=genre.split('|')
    for i in x:
         if i not in genres:
            genres.append(str(i))
genres=str(genres)
movie_title=[]
for title in movies.title:
    movie_title.append(title[0:-7])
movie_title=str(movie_title)
```

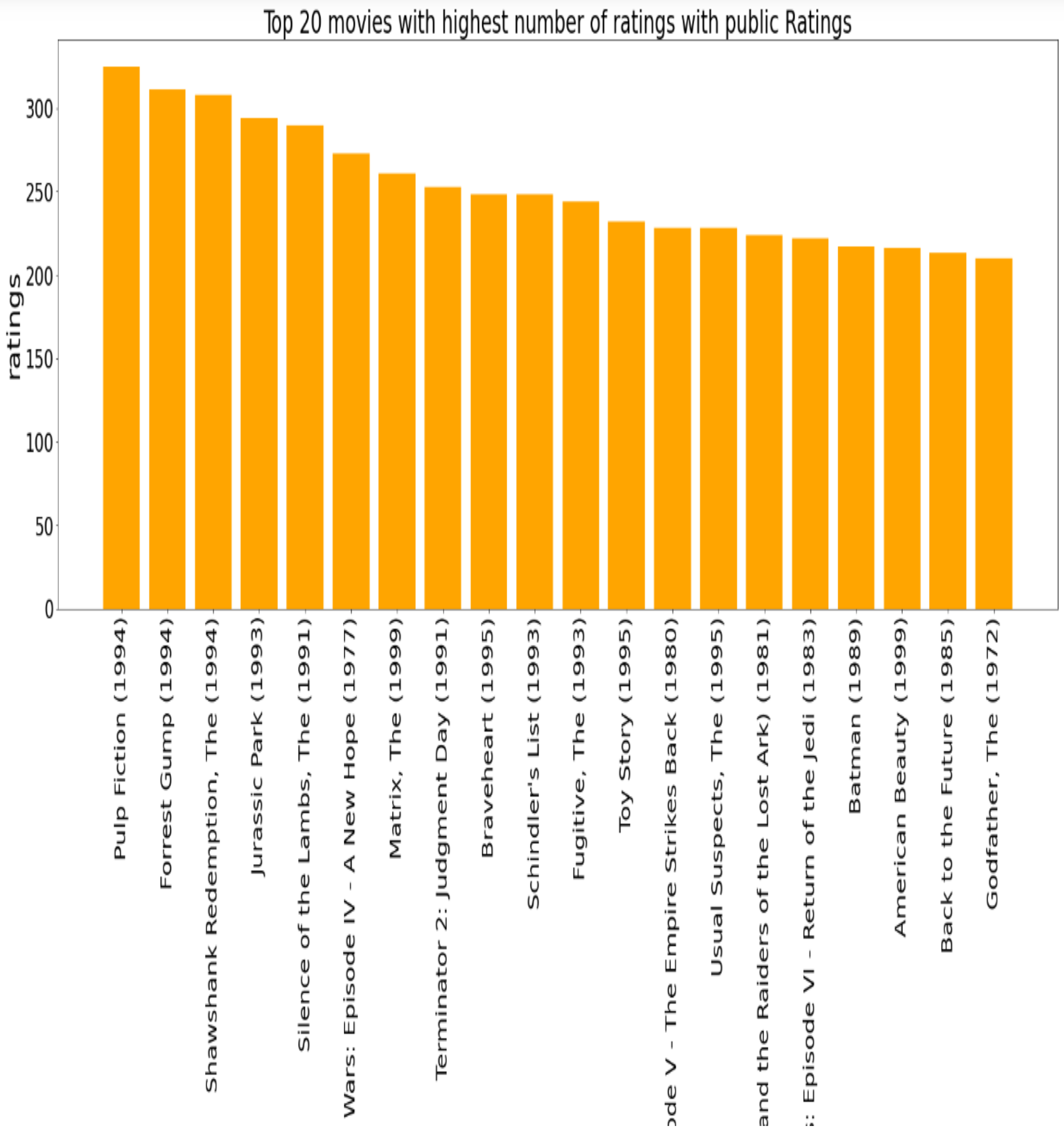
In [22]:
```
df2=df.groupby('title')[['rating']].count()
rating_count_20=df2.nlargest(20,'rating')
rating_count_20.head()
```

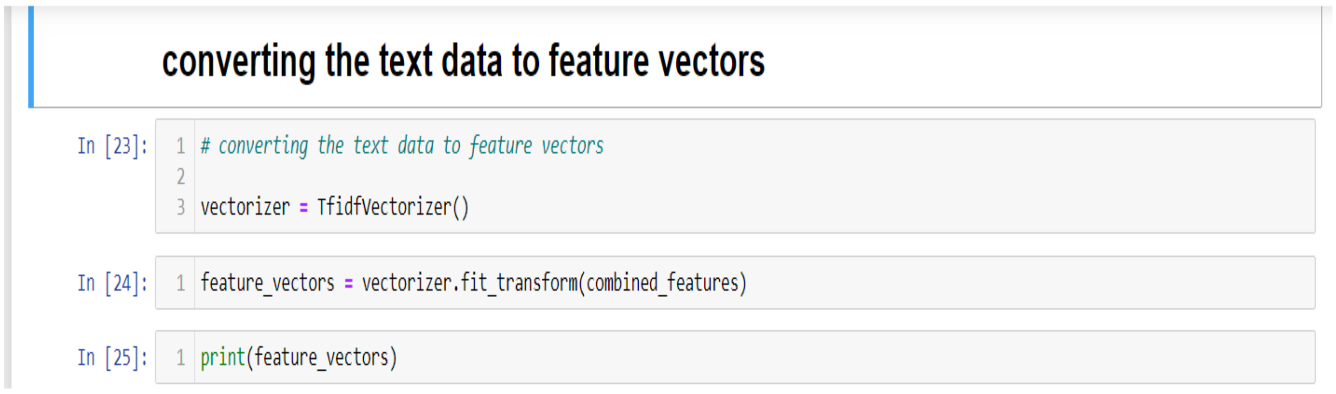
Out[22]:

	rating
title	
Pulp Fiction (1994)	325
Forrest Gump (1994)	311
Shawshank Redemption, The (1994)	308
Jurassic Park (1993)	294
Silence of the Lambs, The (1991)	290

```
In [46]:
plt.figure(figsize=(30,10))
plt.title('Top 20 movies with highest number of ratings with public Ratings',fontsize=30)
plt.xticks(fontsize=25,rotation=90)
plt.yticks(fontsize=25)
plt.xlabel('movies title',fontsize=30)
plt.ylabel('ratings',fontsize=30)

plt.bar(rating_count_20.index,rating_count_20.rating,color='orange')
```

```
Out[46]: <BarContainer object of 20 artists>
```



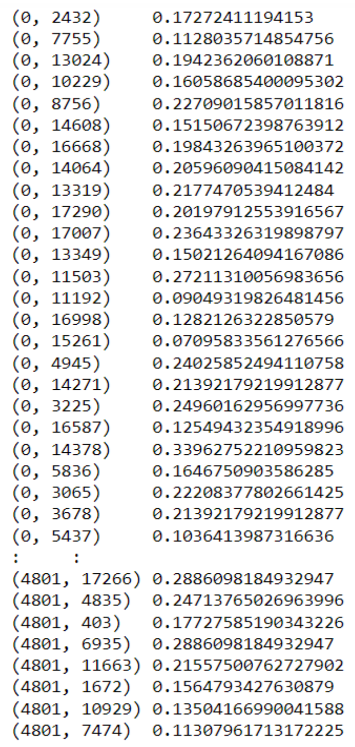


converting the text data to feature vectors

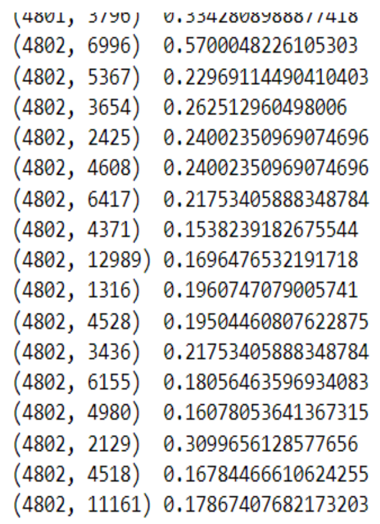
```
In [23]:
# converting the text data to feature vectors

vectorizer = TfidfVectorizer()
```

```
In [24]:
feature_vectors = vectorizer.fit_transform(combined_features)
```

```
In [25]:
print(feature_vectors)
```

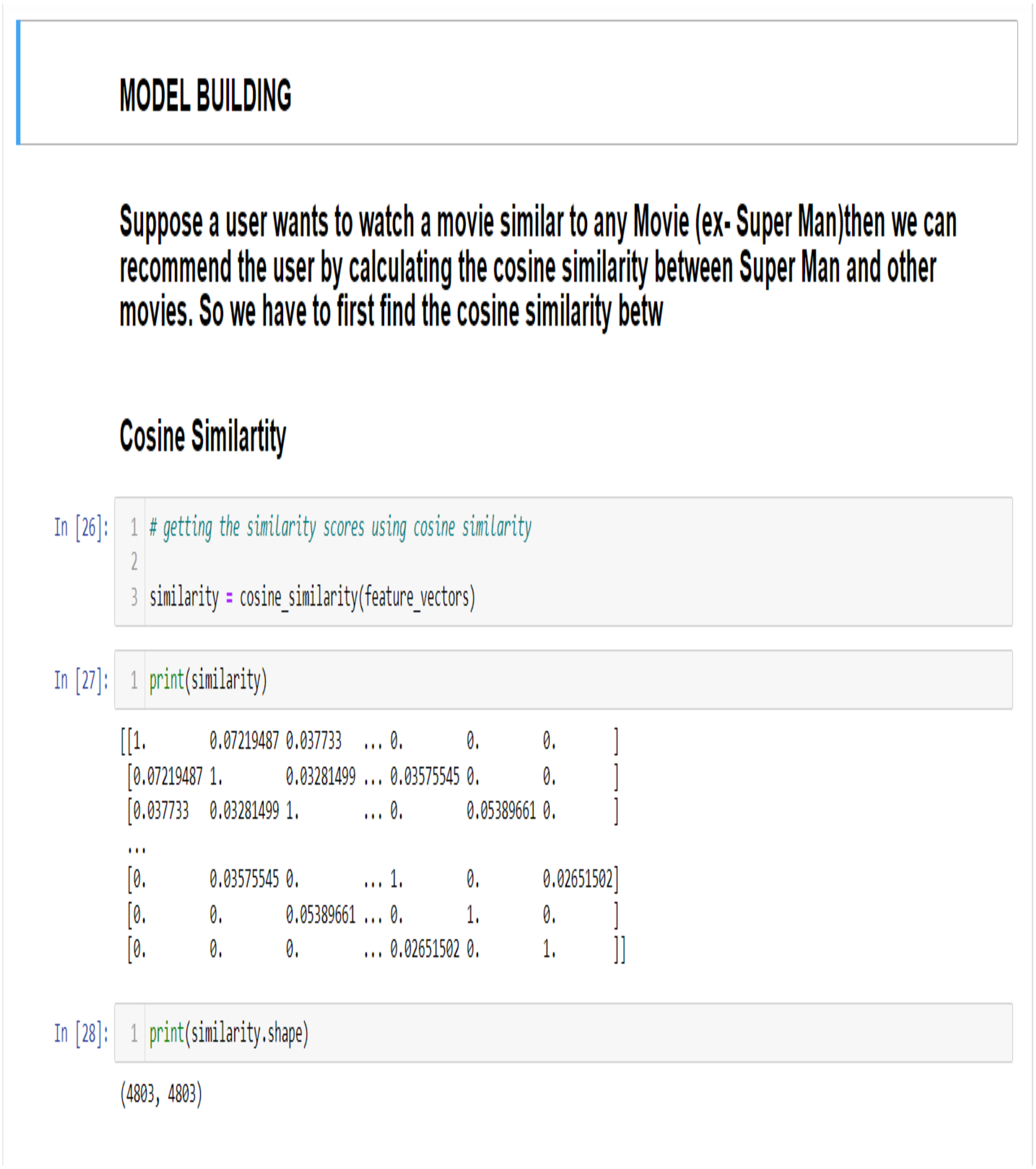


```
  (0, 2432)	0.17272411194153
  (0, 7755)	0.1128035714854756
  (0, 13024)	0.1942362060108871
  (0, 10229)	0.16058685400095302
  (0, 8756)	0.22709015857011816
  (0, 14608)	0.15150672398763912
  (0, 16668)	0.19843263965100372
  (0, 14064)	0.20596090415084142
  (0, 13319)	0.2177470539412484
  (0, 17290)	0.20197912553916567
  (0, 17007)	0.23643326319898797
  (0, 13349)	0.15021264094167086
  (0, 11503)	0.27211310056983656
  (0, 11192)	0.09049319826481456
  (0, 16998)	0.1282126322850579
  (0, 15261)	0.07095833561276566
  (0, 4945)	0.24025852494110758
  (0, 14271)	0.21392179219912877
  (0, 3225)	0.24960162956997736
  (0, 16587)	0.12549432354918996
  (0, 14378)	0.33962752210959823
  (0, 5836)	0.1646750903586285
  (0, 3065)	0.22208377802661425
  (0, 3678)	0.21392179219912877
  (0, 5437)	0.1036413987316636
  :	:
  (4801, 17266)	0.2886098184932947
  (4801, 4835)	0.24713765026963996
  (4801, 403)	0.17727585190343226
  (4801, 6935)	0.2886098184932947
  (4801, 11663)	0.21557500762727902
  (4801, 1672)	0.1564793427630879
  (4801, 10929)	0.13504166990041588
  (4801, 7474)	0.11307961713172225
```



```
  (4801, 3796)	0.33428089888777418
  (4802, 6996)	0.5700048226105303
  (4802, 5367)	0.22969114490410403
  (4802, 3654)	0.262512960498006
  (4802, 2425)	0.24002350969074696
  (4802, 4608)	0.24002350969074696
  (4802, 6417)	0.21753405888348784
  (4802, 4371)	0.1538239182675544
  (4802, 12989)	0.1696476532191718
  (4802, 1316)	0.1960747079005741
  (4802, 4528)	0.19504460807622875
  (4802, 3436)	0.21753405888348784
  (4802, 6155)	0.18056463596934083
  (4802, 4980)	0.16078053641367315
  (4802, 2129)	0.3099656128577656
  (4802, 4518)	0.16784466610624255
  (4802, 11161)	0.17867407682173203
```

Model Building Using Training Data Set



MODEL BUILDING

Suppose a user wants to watch a movie similar to any Movie (ex- Super Man)then we can recommend the user by calculating the cosine similarity between Super Man and other movies. So we have to first find the cosine similarity betw

Cosine Similartity

In [26]:
```
# getting the similarity scores using cosine similarity

similarity = cosine_similarity(feature_vectors)
```

In [27]:
```
print(similarity)
```

```
[[1.         0.07219487 0.037733   ... 0.         0.         0.        ]
 [0.07219487 1.         0.03281499 ... 0.03575545 0.         0.        ]
 [0.037733   0.03281499 1.         ... 0.         0.05389661 0.        ]
 ...
 [0.         0.03575545 0.         ... 1.         0.         0.02651502]
 [0.         0.         0.05389661 ... 0.         1.         0.        ]
 [0.         0.         0.         ... 0.02651502 0.         1.        ]]
```

In [28]:
```
print(similarity.shape)
```

```
(4803, 4803)
```

Getting Movie name from user

In [29]:
```python
# getting the movie name from the user

movie_name = input(' Enter your favourite movie name : ')
```

```
 Enter your favourite movie name : bat man
```

In [30]:
```python
# creating a list with all the movie names given in the dataset

list_of_all_titles = movies_data['title'].tolist()
print(list_of_all_titles)
```

```
['Avatar', "Pirates of the Caribbean: At World's End", 'Spectre', 'The Dark Knight Rises', 'John Carter', 'Spider-Man 3', 'Ta
ngled', 'Avengers: Age of Ultron', 'Harry Potter and the Half-Blood Prince', 'Batman v Superman: Dawn of Justice', 'Superman
Returns', 'Quantum of Solace', "Pirates of the Caribbean: Dead Man's Chest", 'The Lone Ranger', 'Man of Steel', 'The Chronicl
es of Narnia: Prince Caspian', 'The Avengers', 'Pirates of the Caribbean: On Stranger Tides', 'Men in Black 3', 'The Hobbit:
The Battle of the Five Armies', 'The Amazing Spider-Man', 'Robin Hood', 'The Hobbit: The Desolation of Smaug', 'The Golden Co
mpass', 'King Kong', 'Titanic', 'Captain America: Civil War', 'Battleship', 'Jurassic World', 'Skyfall', 'Spider-Man 2', 'Iro
n Man 3', 'Alice in Wonderland', 'X-Men: The Last Stand', 'Monsters University', 'Transformers: Revenge of the Fallen', 'Tran
sformers: Age of Extinction', 'Oz: The Great and Powerful', 'The Amazing Spider-Man 2', 'TRON: Legacy', 'Cars 2', 'Green Lant
ern', 'Toy Story 3', 'Terminator Salvation', 'Furious 7', 'World War Z', 'X-Men: Days of Future Past', 'Star Trek Into Darkne
ss', 'Jack the Giant Slayer', 'The Great Gatsby', 'Prince of Persia: The Sands of Time', 'Pacific Rim', 'Transformers: Dark o
f the Moon', 'Indiana Jones and the Kingdom of the Crystal Skull', 'The Good Dinosaur', 'Brave', 'Star Trek Beyond', 'WALL·
E', 'Rush Hour 3', '2012', 'A Christmas Carol', 'Jupiter Ascending', 'The Legend of Tarzan', 'The Chronicles of Narnia: The L
ion, the Witch and the Wardrobe', 'X-Men: Apocalypse', 'The Dark Knight', 'Up', 'Monsters vs Aliens', 'Iron Man', 'Hugo', 'Wi
ld Wild West', 'The Mummy: Tomb of the Dragon Emperor', 'Suicide Squad', 'Evan Almighty', 'Edge of Tomorrow', 'Waterworld',
'G.I. Joe: The Rise of Cobra', 'Inside Out', 'The Jungle Book', 'Iron Man 2', 'Snow White and the Huntsman', 'Maleficent', 'D
awn of the Planet of the Apes', 'The Lovers', '47 Ronin', 'Captain America: The Winter Soldier', 'Shrek Forever After', 'Tomo
rrowland', 'Big Hero 6', 'Wreck-It Ralph', 'The Polar Express', 'Independence Day: Resurgence', 'How to Train Your Dragon',
'Terminator 3: Rise of the Machines', 'Guardians of the Galaxy', 'Interstellar', 'Inception', 'Shin Godzilla', 'The Hobbit: A
n Unexpected Journey', 'The Fast and the Furious', 'The Curious Case of Benjamin Button', 'X-Men: First Class', 'The Hunger G
```

In [31]:
```python
# finding the close match for the movie name given by the user

find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)
print(find_close_match)
```

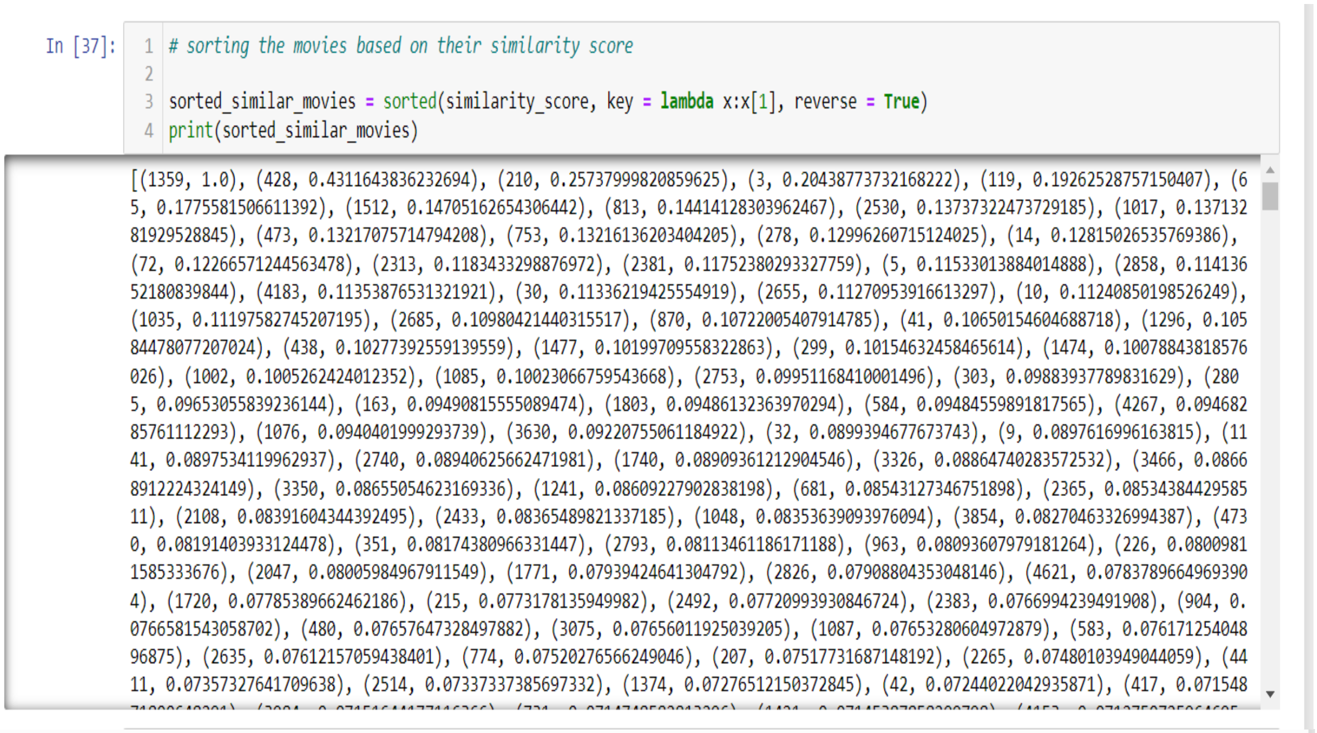
```
['Batman', 'Batman', 'Catwoman']
```

In [32]:
```python
close_match = find_close_match[0]
print(close_match)
```

```
Batman
```

In [33]:
```python
# finding the index of the movie with title

index_of_the_movie = movies_data[movies_data.title == close_match]['index'].values[0]
print(index_of_the_movie)
```

```
1359
```



In [37]:

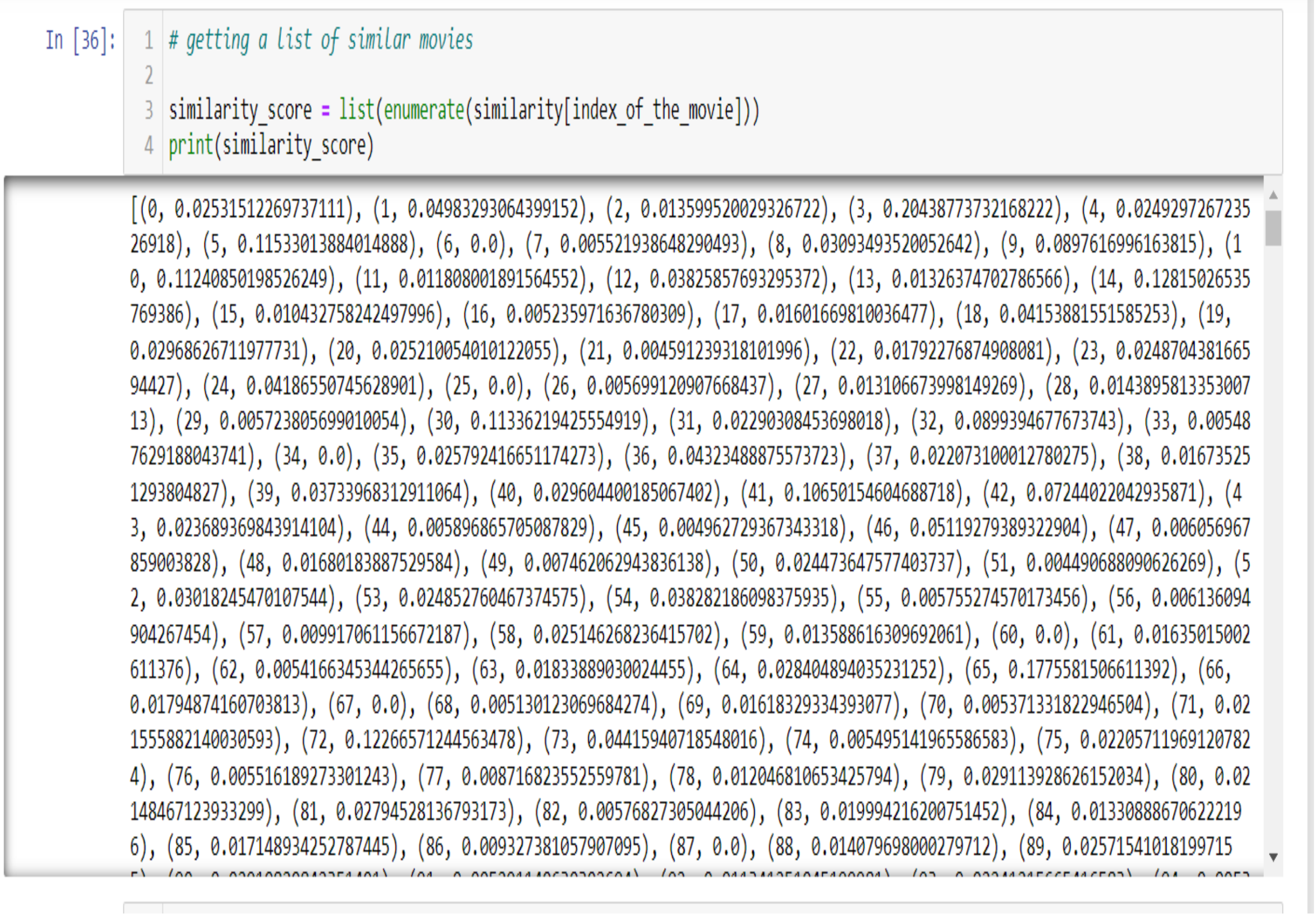
```python
# sorting the movies based on their similarity score

sorted_similar_movies = sorted(similarity_score, key = lambda x:x[1], reverse = True)
print(sorted_similar_movies)
```

```
[(1359, 1.0), (428, 0.4311643836232694), (210, 0.25737999820859625), (3, 0.20438773732168222), (119, 0.19262528757150407), (6
5, 0.177558150661392), (1512, 0.14705162654306442), (813, 0.14414128303962467), (2530, 0.13737322473729185), (1017, 0.137132
81929528845), (473, 0.13217075714794208), (753, 0.13216136203404205), (278, 0.12996260715124025), (14, 0.12815026535769386),
(72, 0.12266571244563478), (2313, 0.1183433298876972), (2381, 0.11752380293327759), (5, 0.11533013884014888), (2858, 0.114136
52180839844), (4183, 0.11353876531321921), (30, 0.11336219425554919), (2655, 0.11270953916613297), (10, 0.11240850198526249),
(1035, 0.11197582745207195), (2685, 0.10980421440315517), (870, 0.10722005407914785), (41, 0.10650154604688718), (1296, 0.105
84478077207024), (438, 0.10277392559139559), (1477, 0.10199709558322863), (299, 0.10154632458465614), (1474, 0.10078843818576
026), (1002, 0.100526242401235 2), (1085, 0.10023066759543668), (2753, 0.09951168410001496), (303, 0.09883937789831629), (280
5, 0.09653055839236144), (163, 0.09490815555089474), (1803, 0.09486132363970294), (584, 0.09484559891817565), (4267, 0.094682
85761112293), (1076, 0.0940401999293739), (3630, 0.09220755061184922), (32, 0.0899394677673743), (9, 0.0897616996163815), (11
41, 0.089753411996293 7), (2740, 0.08940625662471981), (1740, 0.08909361212904546), (3326, 0.08864740283572532), (3466, 0.0866
891222432414 9), (3350, 0.08655054623169336), (1241, 0.08609227902838198), (681, 0.0854312734675189 8), (2365, 0.0853438442958 5
11), (2108, 0.0839160434439249 5), (2433, 0.08365489821337185), (1048, 0.08353639093976094), (3854, 0.0827046332699438 7), (473
0, 0.08191403933124478), (351, 0.08174380966331447), (2793, 0.08113461186171188), (963, 0.08093607979181264), (226, 0.0800981
1585333676), (2047, 0.08005984967911549), (1771, 0.07939424641304792), (2826, 0.07908804353048146), (4621, 0.0783789664969390
4), (1720, 0.07785389662462186), (215, 0.0773178135949982), (2492, 0.07720993930846724), (2383, 0.076694239491908), (904, 0.
076658154305870 2), (480, 0.07657647328497882), (3075, 0.07656011925039205), (1087, 0.07653280604972879), (583, 0.076171254048
96875), (2635, 0.07612157059438401), (774, 0.0752027656624904 6), (207, 0.07517731687148192), (2265, 0.07480103949044059), (44
11, 0.0735732764170963 8), (2514, 0.07337337385697332), (1374, 0.07276512150372845), (42, 0.0724402204293587 1), (417, 0.071548
```

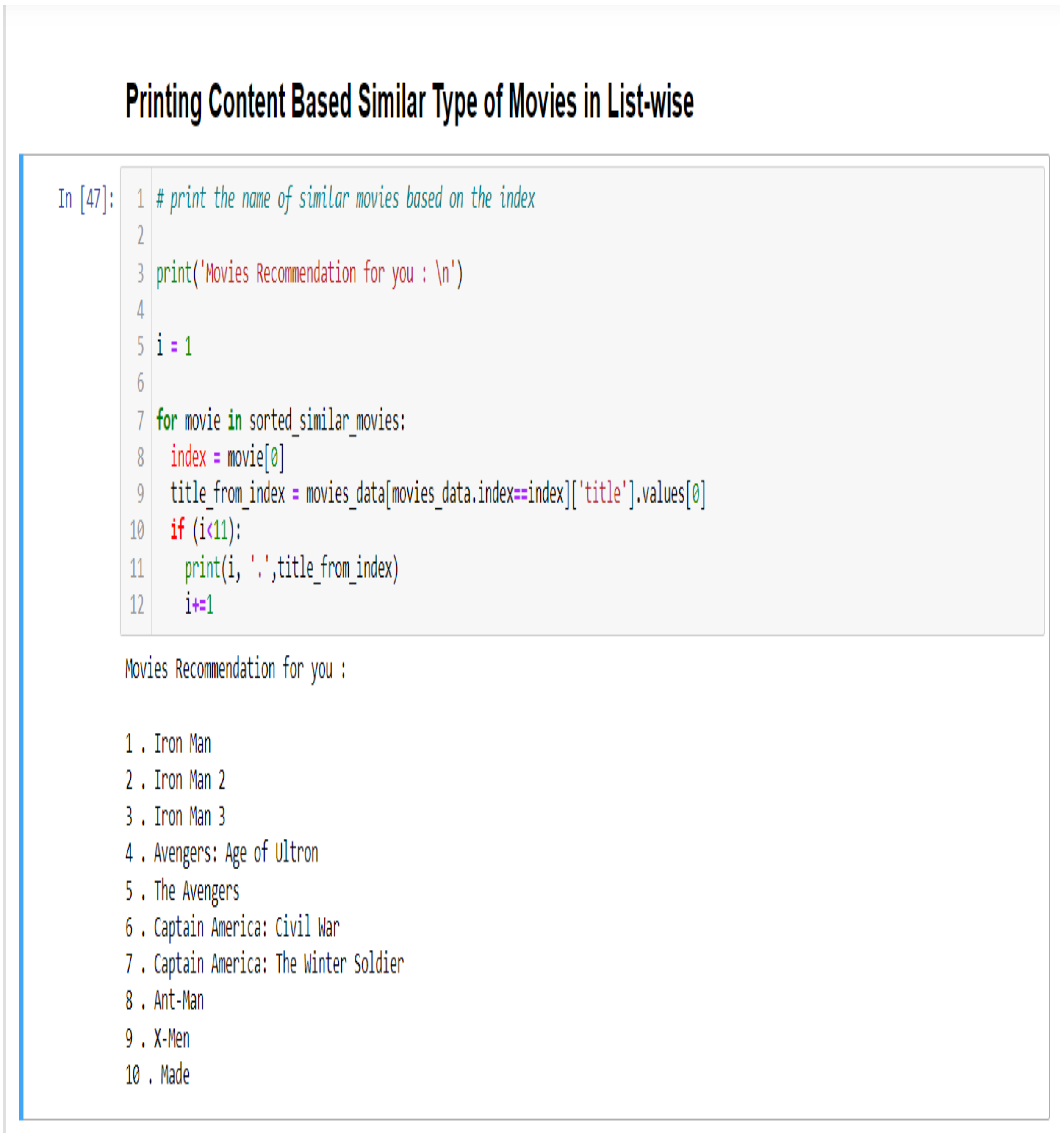
In [36]:

```python
# getting a list of similar movies

similarity_score = list(enumerate(similarity[index_of_the_movie]))
print(similarity_score)
```

```
[(0, 0.0253151226973711 1), (1, 0.04983293064399152), (2, 0.01359952002932672 2), (3, 0.20438773732168222), (4, 0.024929726723 5
26918), (5, 0.11533013884014888), (6, 0.0), (7, 0.00552193864829049 3), (8, 0.03093493520052642), (9, 0.0897616996163815), (1
0, 0.11240850198526249), (11, 0.011808001891564552), (12, 0.03825857693295372), (13, 0.01326374702786566), (14, 0.1281502653 5
769386), (15, 0.010432758242497996), (16, 0.00523597163678030 9), (17, 0.01601669810036477), (18, 0.04153881551585253), (19,
0.02968626711977731), (20, 0.02521005401012205 5), (21, 0.004591239318101996), (22, 0.01792276874908081), (23, 0.024870438166 5
94427), (24, 0.04186550745628901), (25, 0.0), (26, 0.00569912090766843 7), (27, 0.01310667399814926 9), (28, 0.014389581335300 7
13), (29, 0.005723805699010054), (30, 0.11336219425554919), (31, 0.02290308453698018), (32, 0.0899394677673743), (33, 0.00548
762918804374 1), (34, 0.0), (35, 0.02579241665117427 3), (36, 0.04323488875573723), (37, 0.02207310001278027 5), (38, 0.01673525
1293804827), (39, 0.03733968312911064), (40, 0.02960440018506740 2), (41, 0.10650154604688718), (42, 0.0724402204293587 1), (4
3, 0.02368936984391410 4), (44, 0.00589686570508782 9), (45, 0.004962729367343318), (46, 0.0511927938932290 4), (47, 0.006056967
859003828), (48, 0.0168018388752958 4), (49, 0.007462062943836138), (50, 0.02447364757740373 7), (51, 0.004490688090626269), (5
2, 0.0301824547010754 4), (53, 0.02485276046737457 5), (54, 0.03828218609837593 5), (55, 0.005755274570173456), (56, 0.006136094
904267454), (57, 0.009917061156672187), (58, 0.02514626823641570 2), (59, 0.01358861630969206 1), (60, 0.0), (61, 0.01635015002
611376), (62, 0.005416634534426565 5), (63, 0.01833889030024455), (64, 0.02840489403523125 2), (65, 0.177558150661392), (66,
0.0179487416070381 3), (67, 0.0), (68, 0.00513012306968427 4), (69, 0.01618329334393077), (70, 0.005371331822946504), (71, 0.02
155588214003059 3), (72, 0.12266571244563478), (73, 0.04415940718548016), (74, 0.005495141965586583), (75, 0.0220571196912078 2
4), (76, 0.005516189273301243), (77, 0.008716823552559781), (78, 0.01204681065342579 4), (79, 0.02911392862615203 4), (80, 0.02
14846712393329 9), (81, 0.02794528136793173), (82, 0.0057682730504420 6), (83, 0.01999421620075145 2), (84, 0.0133088867062221 9
6), (85, 0.01714893425278744 5), (86, 0.009327381057907095), (87, 0.0), (88, 0.014079698000279712), (89, 0.0257154101819971 5
```



Test :



Printing Content Based Similar Type of Movies in List-wise

In [47]:
```python
# print the name of similar movies based on the index

print('Movies Recommendation for you : \n')

i = 1

for movie in sorted_similar_movies:
  index = movie[0]
  title_from_index = movies_data[movies_data.index==index]['title'].values[0]
  if (i<11):
    print(i, '.',title_from_index)
    i+=1
```

```
Movies Recommendation for you :

1 . Iron Man
2 . Iron Man 2
3 . Iron Man 3
4 . Avengers: Age of Ultron
5 . The Avengers
6 . Captain America: Civil War
7 . Captain America: The Winter Soldier
8 . Ant-Man
9 . X-Men
10 . Made
```

Movie Recommendation System

In [39]:

```python
movie_name = input(' Enter your favourite movie name : ')

list_of_all_titles = movies_data['title'].tolist()

find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)

close_match = find_close_match[0]

index_of_the_movie = movies_data[movies_data.title == close_match]['index'].values[0]

similarity_score = list(enumerate(similarity[index_of_the_movie]))

sorted_similar_movies = sorted(similarity_score, key = lambda x:x[1], reverse = True)

print('\n Movies Recommended for you : \n')

i = 1

for movie in sorted_similar_movies:
  index = movie[0]
  title_from_index = movies_data[movies_data.index==index]['title'].values[0]
  if (i<30):
    print(i, '.',title_from_index)
    i+=1
```

```
Enter your favourite movie name : iron man


 Movies Recommended for you :

1 . Iron Man
2 . Iron Man 2
3 . Iron Man 3
4 . Avengers: Age of Ultron
5 . The Avengers
6 . Captain America: Civil War
7 . Captain America: The Winter Soldier
8 . Ant-Man
9 . X-Men
10 . Made
11 . X-Men: Apocalypse
12 . X2
13 . The Incredible Hulk
14 . The Helix... Loaded
15 . X-Men: First Class
16 . X-Men: Days of Future Past
17 . Captain America: The First Avenger
18 . Kick-Ass 2
19 . Guardians of the Galaxy
20 . Deadpool
21 . Thor: The Dark World
22 . G-Force
23 . X-Men: The Last Stand
24 . Duets
25 . Mortdecai
26 . The Last Airbender
27 . Southland Tales
28 . Zathura: A Space Adventure
29 . Sky Captain and the World of Tomorrow
```

Future Scope of Improvements

There are several potential areas for improvement and future developments for content-based movie recommendation systems. Here are a few:

Incorporating more complex features: Content-based recommendation systems typically rely on simple features such as movie genre, actor, director, and keywords. However, more complex features such as sentiment analysis of movie reviews, analysis of film techniques used, or analysis of audience reactions can be included to improve recommendation accuracy.

Hybrid recommendation systems: Hybrid recommendation systems combine content-based and collaborative filtering methods. They can leverage the strengths of both approaches to provide more accurate recommendations. For example, a hybrid system could use content-based filtering to recommend movies to new users and collaborative filtering to recommend movies to users who have already rated movies.

Personalization: Personalized recommendations can be generated by incorporating user-specific data, such as viewing history, ratings, and preferences. For example, if a user has previously watched several comedies, the system could recommend more comedies to that user.

Dynamic updates: Movie recommendation systems could be improved by updating the recommendations dynamically based on changes in user preferences, new releases, or updates in the movie metadata.

Explain-ability: Providing explanations for the recommended movies can increase user trust and satisfaction with the system. The system can provide a brief description of why a particular movie was recommended to the user based on their past interactions.

Overcoming the cold-start problem: The cold-start problem arises when there is not enough information about a new user or a new movie to make accurate recommendations. One solution is to use a knowledge graph to capture the relationships between movies and other entities such as actors, directors, and genres.

Dealing with data sparsity: Movie datasets are often sparse, with only a few ratings for each movie. Techniques such as matrix factorization, which can estimate the missing ratings based on the known ratings, can be used to overcome this issue.

In summary, the future scope of improvements for content-based movie recommendation systems includes incorporating more complex features, developing hybrid systems, personalization, dynamic updates, explain-ability, overcoming the cold-start problem, and dealing with data sparsity.

Conclusion

In conclusion, a content-based movie recommendation system is a valuable tool for recommending movies to users based on their past movie preferences. By extracting features from movies and creating movie and user profiles, the system can identify similar movies to recommend to the user. The system can be further improved by using techniques such as natural language processing, similarity calculation, and user feedback. The content-based movie recommendation system can enhance user experience, increase user engagement, and improve the revenue of movie recommendation platforms. Overall, the content-based movie recommendation system is an effective and efficient way to recommend movies to users, and its implementation can benefit both users and the movie industry.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that *Mr. ANKOOR DAS*, department of Computer Science and Engineering of Asansol Engineering College, registration number: 201080100110126 of 2020-21 has successfully completed a project on "**CONTENT BASED MOVIE RECOMMENDATION SYSTEM**" using machine learning with python under the guidance of Prof. Arnab Chakraborty.

Signature of The Project Guide
Name: Dr./ M r...............................
Designation: Assistant Professor

Recommendation & Signature of The Principal
Name: D r. P.P. Bhattacharya

Recommendation & Signature of The Head of Department
Name: D r. Monish Chatterjee

Recommendation & Signature of Internal / External Examiners

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ASANSOL ENGINEERING COLLEGE
ESTD
1998

CERTIFICATE

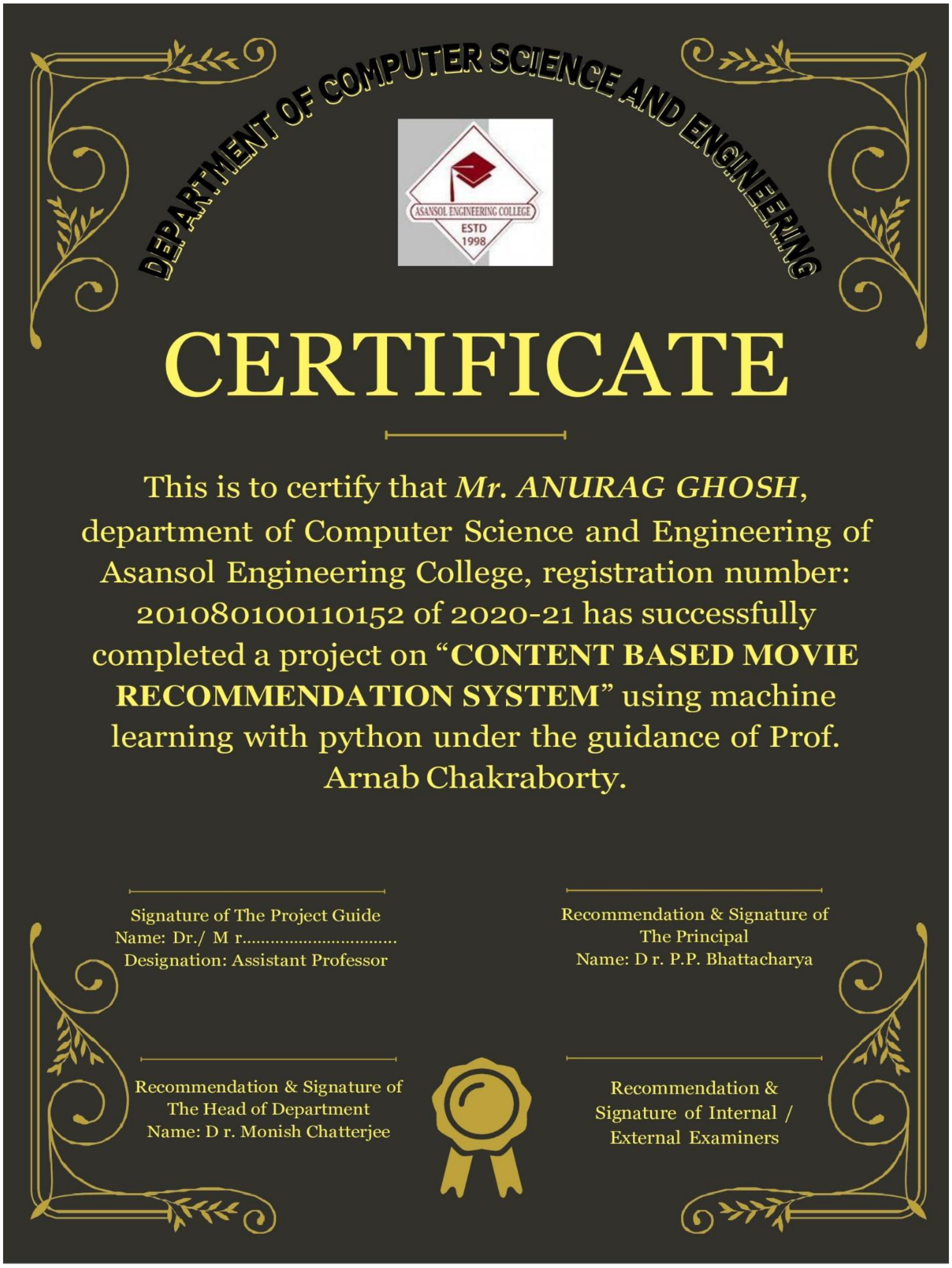
This is to certify that *Mr. DEEP SIKDAR*, department of Computer Science and Engineering of Asansol Engineering College, registration number: 201080100110136 of 2020-21 has successfully completed a project on “**CONTENT BASED MOVIE RECOMMENDATION SYSTEM**” using machine learning with python under the guidance of Prof. Arnab Chakraborty.

Signature of The Project Guide
Name: Dr./ M r................................
Designation: Assistant Professor

Recommendation & Signature of
The Principal
Name: D r. P.P. Bhattacharya

Recommendation & Signature of
The Head of Department
Name: D r. Monish Chatterjee

Recommendation &
Signature of Internal /
External Examiners



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ASANSOL ENGINEERING COLLEGE
ESTD
1998

CERTIFICATE

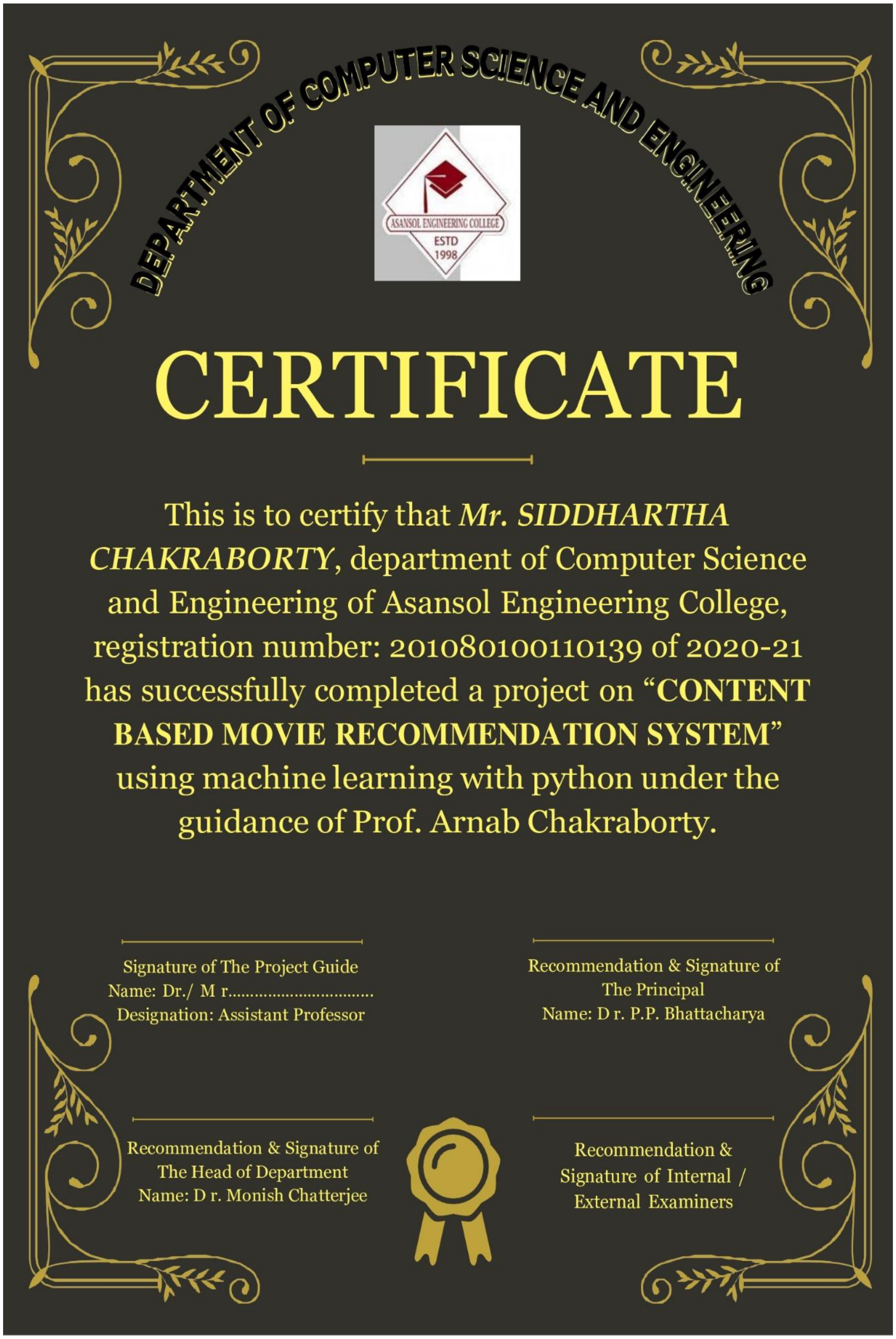
This is to certify that *Mr. ANURAG GHOSH*, department of Computer Science and Engineering of Asansol Engineering College, registration number: 201080100110152 of 2020-21 has successfully completed a project on "**CONTENT BASED MOVIE RECOMMENDATION SYSTEM**" using machine learning with python under the guidance of Prof. Arnab Chakraborty.

Signature of The Project Guide
Name: Dr./ M r................................
Designation: Assistant Professor

Recommendation & Signature of
The Principal
Name: D r. P.P. Bhattacharya

Recommendation & Signature of
The Head of Department
Name: D r. Monish Chatterjee

Recommendation &
Signature of Internal /
External Examiners



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ASANSOL ENGINEERING COLLEGE
ESTD
1998

CERTIFICATE

This is to certify that *Mr. SIDDHARTHA CHAKRABORTY*, department of Computer Science and Engineering of Asansol Engineering College, registration number: 201080100110139 of 2020-21 has successfully completed a project on "**CONTENT BASED MOVIE RECOMMENDATION SYSTEM**" using machine learning with python under the guidance of Prof. Arnab Chakraborty.

Signature of The Project Guide
Name: Dr./ M r..................................
Designation: Assistant Professor

Recommendation & Signature of
The Principal
Name: D r. P.P. Bhattacharya

Recommendation & Signature of
The Head of Department
Name: D r. Monish Chatterjee

Recommendation &
Signature of Internal /
External Examiners