

Chapter 1: INTRODUCTION

Supervised Machine Learning is when the model is getting trained on a labelled dataset. Labelled dataset is one which have both input and output parameters. In this type of learning both training and validation datasets are labelled Here, we have three components such as Training Data, Test Data and features. Training data is that where data is usually split in the ratio of 80:20 i.e. 80% as training data and rest as testing data. Testing data is that when data is good to be tested. At the time of testing, input is fed from remaining 20% data which the model has never seen before, the model will predict some value and we will compare it with actual output and calculate the accuracy

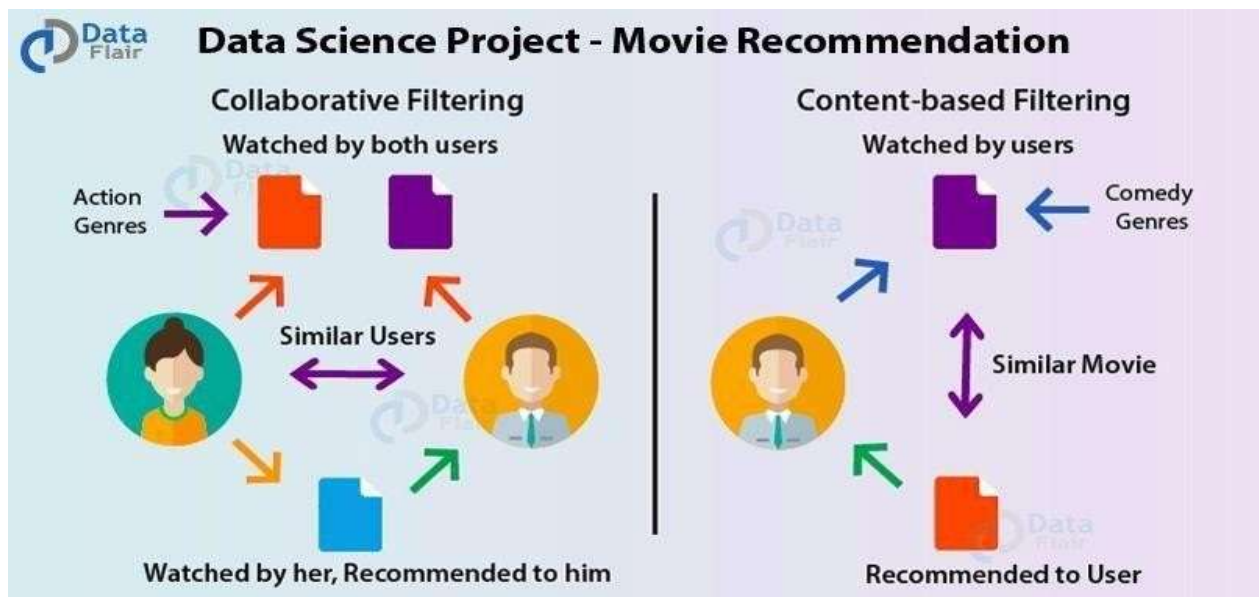


Fig 1.1 Types of Filtering

Three different types of Machine Learning methods are as follows-

1.) Data Extraction and Cleaning

This is been used to extract and clean the data by using scripting languages such as Python, Shell Scripting etc. Here we extract and filter the useful data of movies datasets according to our need.

2.) Build ML Model

Once we will extract and clean data will start building up the model with tools such as Tensor Flow, Azure ML etc. We build our movie recommendation engine here which would be in form of a Python script.

3.) Build Software Infrastructure

Here we have to build ML components such as a product for the users by using the ML algorithms in the form of a software by using JavaScript. A little knowledge of cloud infrastructure such as AWS (Amazon Web Services) and to collaborate with people a little knowledge of GitHub is also known

Users- They are the one who uses these services or acts as the consumer.

Items-Here these are the different sets of movies which are been recommended in a sort of zig-zag manner according to our previous searches.

In the field of Machine Learning, there exist three distinct methods that are utilized for various purposes. These methods play a crucial role in the process of extracting meaningful insights from raw data and developing robust models. Let's explore each of these methods in detail.

The first method is Data Extraction and Cleaning, which involves the extraction and cleansing of data using scripting languages like Python and Shell Scripting. This step is essential to ensure that the data is in a suitable format for further analysis. In the context of movie datasets, this method enables us to extract and filter relevant data based on our specific requirements. By employing techniques such as data parsing and transformation, we can obtain high-quality data that is ready for subsequent stages of the machine learning pipeline.

Moving on to the second method, we have Build ML Model. Once the data has been extracted and cleaned, the next step is to construct a machine learning model. This process involves utilizing various tools and frameworks such as Tensor Flow and Azure ML to build a robust model that can effectively analyze and predict patterns in the data. In the case of movie recommendation engines, this method enables us to develop a sophisticated model that can generate personalized recommendations based on users' preferences and previous searches. Typically, the implementation of this method involves writing a Python script that incorporates the model and its associated algorithms.

The third method is Build Software Infrastructure, which focuses on integrating machine learning components into a software system. In this step, ML algorithms and models are incorporated into a software application using programming languages such as JavaScript. Additionally, knowledge of cloud infrastructure, such as Amazon Web Services (AWS), may be necessary to deploy and scale the ML components effectively. Collaboration tools like GitHub also play a vital role in this method, facilitating teamwork and version control among developers.

In the context of these methods, it is important to consider the primary actors involved. The users of these ML services or systems are central to the entire process. They are the individuals who interact with the recommendation engines or utilize the services offered by the machine learning system. By leveraging the power of machine learning, these users can benefit from personalized and tailored recommendations, enhancing their overall experience.

Another significant element to consider is the concept of "items." In this context, items refer to different sets of movies within the dataset. These items are recommended to users in a zig-zag manner, taking into account their previous searches and preferences. By analyzing patterns and similarities among movies, the machine learning system can suggest relevant items to users, ensuring a diverse and engaging experience.

In summary, these three distinct methods of Machine Learning—Data Extraction and Cleaning, Build ML Model, and Build Software Infrastructure—play crucial roles in the development of machine learning systems, such as movie recommendation engines. Each method contributes to different stages of the overall process, from preparing the data for analysis to constructing models and integrating them into software applications. By understanding these methods and considering the actors and items involved, we can appreciate the intricate process of leveraging machine learning to provide personalized recommendations and enhance user experiences. Machine Learning encompasses a wide array of techniques and approaches that facilitate the extraction of valuable insights from data. Within this vast field, there are three distinct methods that serve as fundamental building blocks in the development of machine learning systems. These methods are integral to the process of transforming raw data into actionable knowledge and creating robust models that can make accurate predictions and recommendations. In this article, we will delve deeper into each of these methods and explore their significance in the context of movie recommendation engines.

The first method revolves around Data Extraction and Cleaning. This critical step involves the extraction and purification of data through the utilization of powerful scripting languages, such as Python and Shell Scripting. The primary objective here is to extract the relevant data from various sources and transform it into a standardized format that is amenable to further analysis. By employing sophisticated algorithms and techniques, data scientists can filter out noise, handle missing values, and ensure the data's consistency and quality.

In the context of movie datasets, the Data Extraction and Cleaning method assumes paramount importance. It enables data scientists to sift through vast repositories of information and identify the data points that are most pertinent to their specific requirements. For example, they can extract key details such as movie titles, genres, release dates, and user ratings. By carefully curating and preparing this data, they lay the foundation for subsequent steps in the machine learning pipeline.

The second method, Build ML Model, involves the creation of sophisticated machine learning models that can effectively analyze patterns and make accurate predictions. In this stage, data scientists harness the power of tools and frameworks like Tensor Flow and Azure ML to construct robust models tailored to the unique needs of movie recommendation engines. These models utilize advanced algorithms and techniques to identify underlying patterns in the data and make personalized recommendations to users.

Building an effective ML model for movie recommendations is no trivial task. Data scientists must consider various factors such as user preferences, previous searches, and historical data to develop a model that can capture the nuances of individual tastes and deliver relevant suggestions. By leveraging the power of machine learning algorithms, they can identify similarities among movies and discern hidden patterns that elude human perception. This enables the ML model to generate accurate and personalized recommendations, enhancing the overall user experience.

The third method, Build Software Infrastructure, involves integrating the machine learning components into a comprehensive software system. In this stage, data scientists and software engineers collaborate to develop a seamless user interface that can deliver the recommendations generated by the ML model. Programming languages such as JavaScript are often employed to implement the ML algorithms and create a user-friendly software application.

Furthermore, a solid understanding of cloud infrastructure, such as Amazon Web Services (AWS), becomes essential during the Build Software Infrastructure stage. Cloud platforms offer scalable and reliable solutions for hosting machine learning models and handling the computational demands of recommendation engines. By leveraging the power of cloud services, developers can ensure that the ML components operate efficiently and provide real-time recommendations to users.

Collaboration and version control play a crucial role in the Build Software Infrastructure method. Platforms like GitHub provide a centralized environment where developers can collaborate, share code, and track changes. This ensures seamless teamwork and enables the development team to work cohesively towards creating a robust software infrastructure for the machine learning system.

Now, let's delve into the key actors involved in the machine learning ecosystem. The primary users, or consumers, are central to the success of machine learning services. They are the individuals who interact with the recommendation engines and utilize the services offered by the machine learning system. These users may include movie enthusiasts seeking personalized recommendations based on their unique preferences and interests. By leveraging the power of machine learning, these users can benefit from tailored suggestions that align with their individual tastes, leading to a more engaging and satisfying experience.

Another critical aspect to consider is the concept of "items." In the context of movie recommendation engines, items....

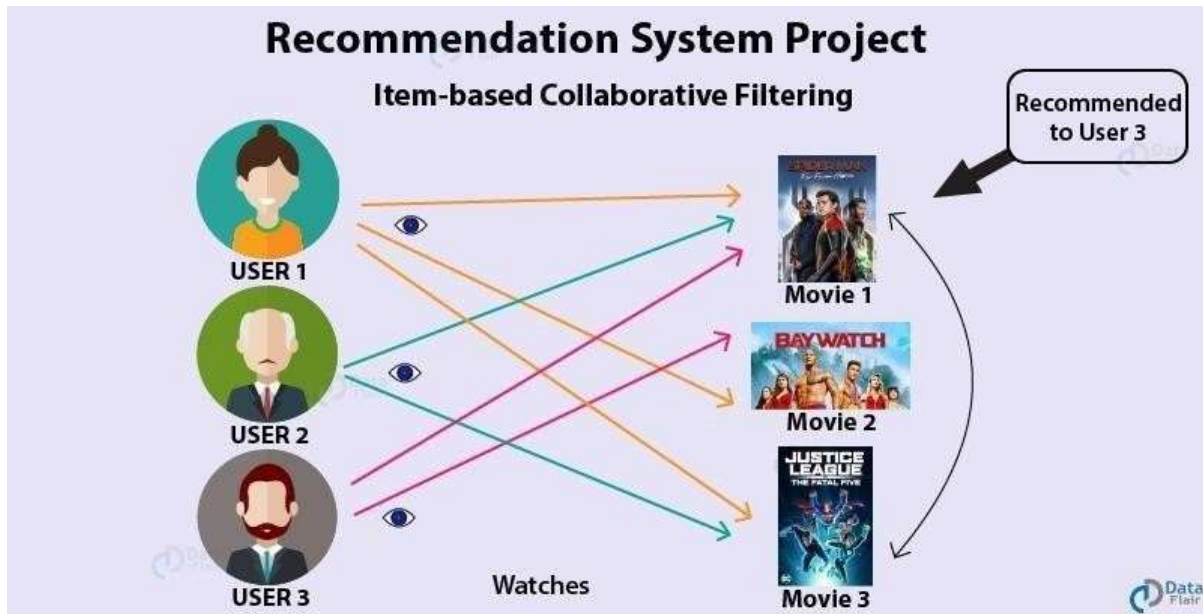


Fig 1.2 Item-Based Collaborative Filtering

Three ways of performing the filtering are as follows-

- Trending based filtering- Here the movies are been classified by the ratings and the stuff that is been liked by a majority of the population.
- Content based filtering- Here the similar articles are been recommended to the user according to his previous content search.
- Collaborative based filtering- Here the two similar user likings act as a recommend to each other. Like, we two users watch comedy movies so if a new comedy stuff appears and is watched by A user it will also be recommended to user B.

A recommender system, or a recommendation system (sometimes replacing 'system' with a synonym such as platform or engine), is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. They are primarily used in commercial applications. Recommender systems are utilized in a variety of areas and are most commonly recognized as playlist generators for video and music services like Netflix, YouTube and Spotify, product recommenders for services such as Amazon, or content recommenders for social media platforms such as Facebook and Twitter. These systems can operate using a single input, like music, or multiple inputs within and across platforms like news, books, and search queries. There are also popular recommender systems for specific topics like restaurants and online dating. Recommender systems have also been developed to explore research articles and experts, collaborators, and financial services.

These both are the recommendation engines that recommend us the movies and other related stuff based on our previous searches and watched experience.



fig 1.3 Different over-the-top Platforms

- Classification: It is a Supervised Learning task where output is having defined labels (discrete value).
- Regression: It is a Supervised Learning task where output is having continuous value.

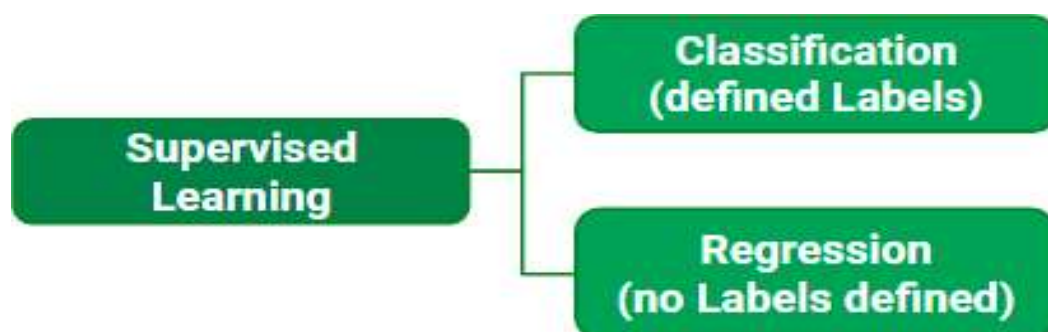


fig 1.4 Tasks of Supervised Learning

Chapter 2: EXISTING SYSTEM

- We use Digital libraries for a wide variety of digital objects (research papers, publications, journals, research projects, newspapers, magazines, and past questions).
- But some digital libraries even offer millions of digital objects. Therefore, getting or finding favourite digital objects from a large collection of available digital objects in the digital library is one of the major problems.
- The users need help in finding items that are in accordance with their interests.
- Recommender systems offer a solution to this problem as library users will get recommendations using a form of smart search.
- The problem considered here is to develop recommendation to find a particular variety of the objects to user (like publications, research paper etc.) only in a large collection of items.

Recommender systems are software applications that suggest or recommend items or products (in the case of ecommerce) to users. These systems use users' preferences or interests (supplied as inputs) and an appropriate algorithm in finding the relevant or desired items or products. Recommender systems deal with information overload problems by filtering items that potentially may match the users' preferences or interests. These systems aid users to efficiently overcome the problem by filtering irrelevant information when users search for desired information.

Content-based recommenders provide recommendations by comparing representation of contents describing an item or a product to the representation of the content describing the interest of the user (User's profile of interest). They are sometimes referred to as content-based filtering. The content-based technique is adopted or considered here for the design of the recommender system for digital libraries. Content-based technique is suitable in situations or domains where items are more than users.

Content-based recommenders analyze the characteristics or features of the items or products in order to generate recommendations. In the case of digital libraries, these systems examine the content or metadata associated with the digital objects, such as research papers, publications, journals, and research projects. By understanding the attributes of these items, the recommender system can make informed suggestions based on the user's profile of interest.

To implement a content-based recommender system for digital libraries, various steps need to be taken. First, the system needs to extract relevant information from the digital objects, including keywords, authors, publication dates, and other metadata. This data serves as the basis for comparing and matching the user's interests with the characteristics of the items.

Next, the recommender system creates a user profile or preference model based on the information provided by the user. This profile represents the user's preferences, areas of interest, and any specific criteria they may have. The system then compares the user profile with the content or metadata of the digital objects to identify the most relevant matches.

One common technique used in content-based recommenders is vectorization, where the textual content of the digital objects is transformed into numerical vectors. This process allows for efficient comparison and similarity calculation between the user's profile and the items in the digital library. Similarity measures, such as cosine similarity, can be used to determine the degree of relevance between the user's profile and each item.

Once the recommender system has identified the most suitable items based on content similarity, it presents these recommendations to the user. The user can then browse through the suggested digital objects and choose the ones that align with their interests. Additionally, the system can continuously learn from the user's feedback and update the user profile to provide more accurate recommendations over time.

By utilizing a content-based recommender system, digital libraries can enhance the user experience by assisting users in discovering relevant and personalized content. These systems alleviate the burden of searching through large collections of digital objects, allowing users to efficiently navigate and access the information they need. Whether it's academic research, literature, or any other form of digital content, content-based recommenders play a vital role in improving the discoverability and accessibility of digital libraries.

The use of collaborative-filtering technique in recommending research papers has been criticized by some authors. Authors like [15] suggest that collaborative-filtering technique is ineffective in domains where items (e.g. research papers) are more than users. [16] Said; "Users are not willing to spend time to rate items explicitly". Hence, content-based approach is adopted for the design and implementation of research paper recommender system. This approach does not depend on the ratings of other users but uses the contents describing the items and the users' taste or needs. The researchers used the following data collection procedure and methods in representing the research papers, users' profile of interest, and also in providing recommendations to the users.

The Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. The researchers used this method to determine how similar a research paper is to a user's query or paper that a user has liked in the past. The research papers are represented as vectors of weights, where each weight indicates the degree of association between the research papers and the term.

It is helpful in implementing the model with cosine similarity formula and recommend the string similar to the user choice by analysing his patterns of writing with that of the stated other given string.

When two or more movies of some different genres come it becomes hard for Sublime Text to understand.

- It gives the results of similar movies related to my applied set of movies.
- It is confined of providing the results with the help of cosine similarity formula and count vectorizer.
- It provides the output of movies with the help of datasets governed through csv files.
- Not confined to provide advance hybrid string matching elements solutions and responses.

The architecture shown in Figure 2.1 explains the overview of the recommendation system used in the project

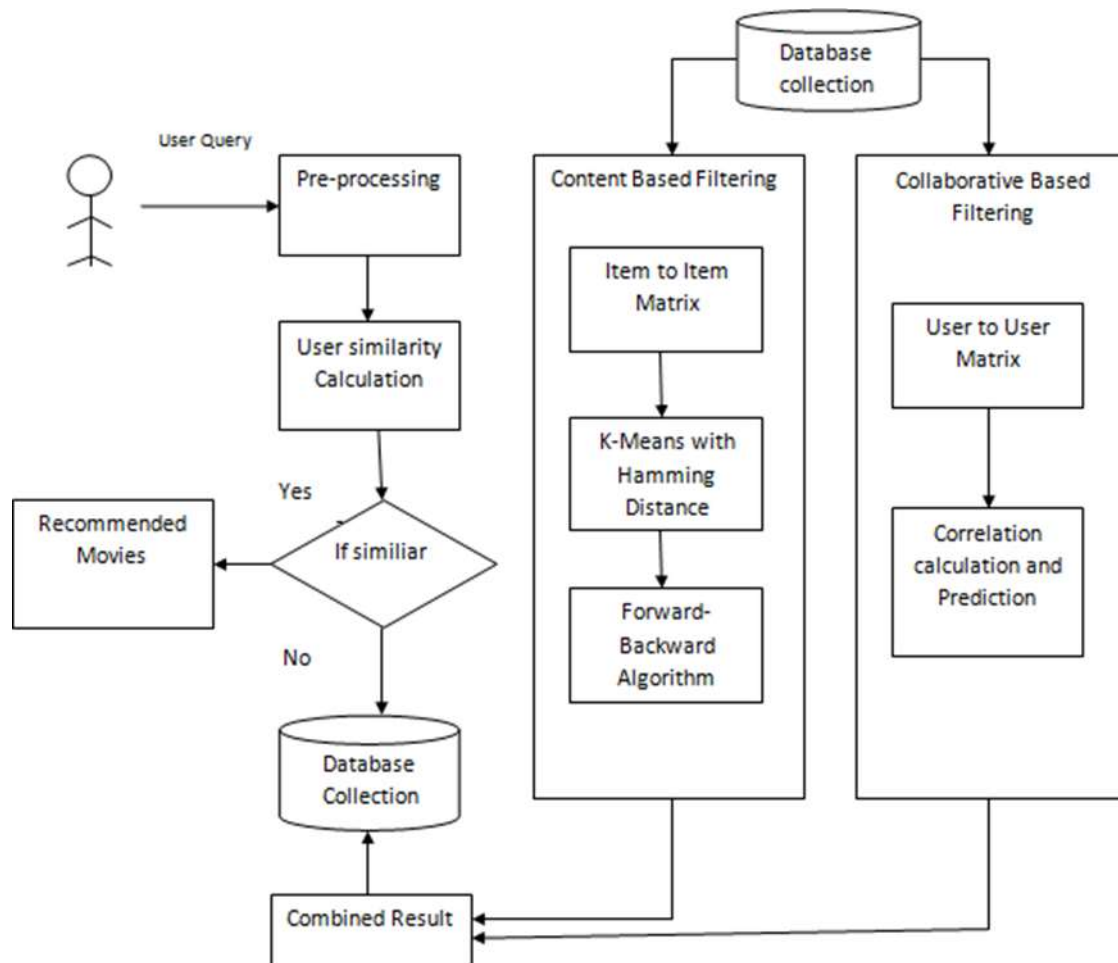


Fig 2.1 Block Diagram of Movie Recommendation System

Chapter 3: PROPOSED SYSTEM

- To find out the related content of the user in a given set of collection and to his interest feed.
- Providing the accurate and most confined results by using the distance between two vectors model and graph plotting examples.
- Using the cosine similarity formula for finding the similarity scores/matching of the two texts more accurately.
- By providing accurate results by the usage of movies datasets by IMDB in the existing project work.
- Classifying the users interest movies and recommend them to their searches fast by help of libraries and software as pip, panda etc.

Keyword-Based Vector-Space Model: The researchers used this model with basic TF-IDF weighing technique to represent a research paper as a vector of weights, where each weight indicates the degree of association between a research paper and a term or keyword.

- **Item Representation:** The items (research papers) are represented by a set of features (also called attributes or properties). These attributes are: title of the paper, abstract, keywords, research area, ID of the paper, and the authors. The abstract represents the research paper when the frequency of a term in the research paper is being determined.

Recommender systems are software applications that provide or suggest items to intended users. These systems use filtering techniques to provide recommendations. The major ones of these techniques are collaborative-based filtering technique, content-based technique, and hybrid algorithm. The motivation came as a result of the need to integrate recommendation feature in digital libraries in order to reduce information overload. Content-based technique is adopted because of its suitability in domains or situations where items are more than the users. TF-IDF (Term Frequency Inverse Document Frequency) and cosine similarity were used to determine how relevant or similar a research paper is to a user's query or profile of interest. Research papers and user's query were represented as vectors of weights using Keyword-based Vector Space model. The weights indicate the degree of association between a research paper and a user's query. This paper also presents an algorithm to provide or suggest recommendations based on users' query. The algorithm employs both TF-IDF weighing scheme and cosine similarity measure. Based on the result or output of the system, integrating recommendation feature in digital libraries will help library users to find most relevant research papers to their needs

Keyword-Based Vector-Space Model: The keyword-based vector-space model is a powerful technique used by researchers to analyze and represent research papers. By utilizing the basic TF-IDF (Term Frequency-Inverse Document Frequency) weighing technique, this model assigns weights to each term or keyword based on its association with a research paper. These weights form a vector representation of the paper, allowing for efficient analysis and comparison.

In the context of research papers, the items being represented are the papers themselves. These items are characterized by a set of features, also known as attributes or properties. These features typically include the title of the paper, abstract, keywords, research area, paper ID, and authors. Each of these attributes provides valuable information about the content and context of the research paper.

The abstract of a research paper plays a crucial role in representing the paper when determining the frequency of a term within the document. It serves as a concise summary of the paper's main objectives, methodologies, and findings. By considering the frequency of terms within the abstract, the keyword-based vector-space model captures the essence of the research paper and its relevance to specific terms or keywords.

Recommender systems have become essential tools in various domains, including digital libraries, e-commerce platforms, and streaming services. These systems aim to provide personalized recommendations to users based on their preferences and interests. In the context of research papers, recommender systems assist users in discovering relevant papers aligned with their research interests, thus reducing information overload.

There are several techniques employed by recommender systems, with collaborative-based filtering, content-based techniques, and hybrid algorithms being the most prevalent. Collaborative-based filtering relies on user behavior data, such as ratings or preferences, to generate recommendations. Content-based techniques, on the other hand, leverage the characteristics and attributes of the items themselves to make recommendations. Hybrid algorithms combine both collaborative and content-based approaches to produce more accurate and diverse recommendations.

In the case of research paper recommendation, the content-based technique is particularly suitable. This is because the number of available research papers often far exceeds the number of users, making it challenging to rely solely on user behavior data. By analyzing the content and attributes of research papers, the content-based approach can effectively match papers with users based on their specific interests and requirements.

The keyword-based vector-space model, with its TF-IDF weighing scheme and cosine similarity measure, forms the foundation of the content-based recommendation system for research papers. Research papers and users' queries are represented as vectors of weights, where each weight represents the degree of association between a paper or query and a specific term or keyword. The TF-IDF scheme takes into account the term frequency within a paper and the inverse document frequency, which considers the importance of the term across the entire collection of papers. Cosine similarity is then used to measure the similarity between the vector representations of papers and users' queries, enabling the system to provide relevant recommendations.

To implement the recommendation system, an algorithm is developed that leverages the keyword-based vector-space model, TF-IDF weighing scheme, and cosine similarity measure. This algorithm takes a user's query as input and compares it to the vector representations of research papers. By calculating the cosine similarity between the query vector and each paper vector, the algorithm ranks the papers based on their relevance to the query. The top-ranked papers are then recommended to the user, ensuring that the most pertinent and suitable papers are presented.

Integrating a recommendation feature in digital libraries holds significant benefits for users. By leveraging the algorithm and the keyword-based vector-space model, users can efficiently find research papers that align with their needs and interests. This not only saves time and effort but also improves the overall research experience by providing access to relevant and high-quality papers.

In conclusion, the keyword-based vector-space model, combined with the TF-IDF weighing technique and cosine similarity.

The integration of recommendation features in digital libraries has revolutionized the way users discover and access relevant research papers. By leveraging advanced techniques such as the keyword-based vector-space model, TF-IDF weighing scheme, and cosine similarity measure, recommender systems provide users with personalized recommendations based on their specific interests and requirements.

The keyword-based vector-space model plays a pivotal role in representing research papers as vectors of weights. Each weight corresponds to the degree of association between a paper and a particular term or keyword. This model allows for efficient analysis and comparison of papers, enabling the system to identify the most relevant ones for a user's query.

The TF-IDF weighing scheme employed in the keyword-based vector-space model takes into account the term frequency within a paper and the inverse document frequency. The term frequency reflects how frequently a term appears in a specific paper, while the inverse document frequency measures the importance of the term across the entire collection of papers. By combining these factors, the TF-IDF scheme assigns weights to terms that are both frequent within a paper and distinct across the entire collection, capturing the significance of each term in relation to a specific paper.

Cosine similarity is a key measure used in the recommendation algorithm to determine the similarity between the vector representations of research papers and users' queries. It calculates the cosine of the angle between two vectors, providing a value between 0 and 1, where 1 indicates perfect similarity. By comparing the cosine similarities between the query vector and each paper vector, the algorithm ranks the papers based on their relevance to the user's query. The top-ranked papers are then recommended to the user, ensuring that the most suitable and pertinent papers are presented.

The recommendation algorithm that incorporates the keyword-based vector-space model, TF-IDF weighing scheme, and cosine similarity measure brings significant benefits to users of digital libraries. By providing personalized recommendations, the system assists users in finding research papers that align with their needs and research interests. This not only saves time and effort but also enhances the research experience by offering access to high-quality and relevant papers.

Moreover, the recommendation system based on the keyword-based vector-space model can contribute to knowledge discovery and research collaboration. By analyzing the vector representations of papers, the system can identify patterns and similarities among different research topics and areas. This can facilitate interdisciplinary research and foster collaborations between researchers working on related subjects.

Furthermore, the integration of recommendation features in digital libraries addresses the issue of information overload. With an ever-growing volume of research papers being published, users often struggle to navigate through the vast amount of available information. The recommendation system acts as a filtering mechanism, guiding users towards papers that are most likely to be valuable and relevant to their research goals. By reducing information overload, the system enhances the efficiency and effectiveness of the research process.

In addition to aiding users in discovering research papers, the recommendation system can also benefit authors and publishers. By providing personalized recommendations, the system increases the visibility and exposure of research papers, potentially leading to higher citation rates and wider dissemination of knowledge. This, in turn, benefits researchers and institutions by enhancing their academic impact and reputation.

In conclusion, the integration of recommendation features in digital libraries through the utilization of the keyword-based vector-space model, TF-IDF weighing scheme, and cosine similarity measure has transformed the way users discover and access research papers. The system's ability to provide personalized recommendations based on users' queries enhances the research experience, saves time, and facilitates knowledge discovery.

Step 1: Install Python

If you do not have Python installed on your computer. Install the latest version of Python from Web.

Step 2: Download the pip package manager for Python Once

you have installed Python from the instructions above.

We need to install some libraries which we are going to use in our workshop. We will install Numpy, Matplotlib and Pandas to work with our datasets.

pip is a package management system used to install and manage software packages written in Python.

Right Click the following link and select Save Link As

(Save Target As): <https://bootstrap.pypa.io/get-pip.py>

Go to the folder where you saved this file. In windows explorer use Shift + Right Click and then select Open command window here to open command prompt in this directory. Then run the following command:

```
python get-pip.py
```

Step 3: Install Libraries

Open Command Prompt.

Run the following command to install necessary libraries.

```
pip install numpy matplotlib pandas scikit-learn gym opencv-python
```

If the installation completes without any errors, you are all set!

Chapter 3: IMPLEMENTATION OF CODING DIAGRAMS

- Step1: Read CSV file of dataset.
- Step2: Select features of datasets
- Step3: Create a column in DF which contains all selected features
- Step 4: Create count matrix from this new combined column
- Step5: Compute the cosine similarity based on count matrix
- Step6: Get index of this movie from title
- Step7: We will get the list of similar movies in descending order of similarity score
- Step8: Lastly, Print title of first 15 movies

- Step1: Read CSV file of dataset

```
12
13  ##Step 1: Read CSV File
14  df = pd.read_csv("movie_dataset.csv")
15  print df.head()
```

	index	...	director
0	0	...	James Cameron
1	1	...	Gore Verbinski
2	2	...	Sam Mendes
3	3	...	Christopher Nolan
4	4	...	Andrew Stanton

```
[5 rows x 24 columns]
[Finished in 3.5s]
```

- Step2: Select features of datasets

- Step3: Create a column in DF which contains all selected features

```
C:\Users\robin\OneDrive\Desktop\Movie Recommend\movie_recommender (1)\movie_recommender.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

cosine_similarity x movie_recommender.py x
18 features = ['keywords', 'cast', 'genres', 'director']
19
20 ##Step 3: Create a column in DF which combines all selected features
21 for feature in features:
22     df[feature] = df[feature].fillna('')
23     def combine_features(row):
24         try:
25             return row["keywords"]+" "+row["cast"]+" "+row["genres"]+" "+row["director"]
26         except:
27             print "Error:" , row
28
29
30 df["combined_features"] = df.apply(combine_features,axis=1)
31 print "combined features:", df["combined_features"].head()
```

- Step 4: Create count matrix from this new combined column

```
C:\Users\robin\OneDrive\Desktop\Movie Recommend\movie_recommender (1)\movie_recommender.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

cosine_similarity x movie_recommender.py x
17 ##Step 2: Select Features
18 features = ['keywords', 'cast', 'genres', 'director']
19
20 ##Step 3: Create a column in DF which combines all selected features
21 for feature in features:
22     df[feature] = df[feature].fillna('')
23     def combine_features(row):
24         try:
25             return row["keywords"]+" "+row["cast"]+" "+row["genres"]+" "+row["director"]
26         except:
27             print "Error:" , row
28
29
30 df["combined_features"] = df.apply(combine_features,axis=1)
31 print "combined features:", df["combined_features"].head()
32
33 ##Step 4: Create count matrix from this new combined column
34 cv = CountVectorizer()
35 count_matrix = cv.fit_transform(df["combined_features"])
```

- Step5: Compute the cosine similarity based on count matrix

- Step6: Get index of this movie from title

```

C:\Users\robin\OneDrive\Desktop\Movie Recommend\movie_recommender (1)\movie_recommender.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

cosine_similarity x movie_recommender.py x
34 cv = CountVectorizer()
35 count_matrix = cv.fit_transform(df["combined_features"])
36
37 ##Step 5: Compute the Cosine Similarity based on the count_matrix
38 cosine_sim = cosine_similarity(count_matrix)
39
40 movie_user_likes = "Avatar"
41
42 ## Step 6: Get index of this movie from its title
43 movie_index = get_index_from_title(movie_user_likes)
44 similar_movies = list(enumerate(cosine_sim[movie_index]))
45

```

- Step7: We will get the list of similar movies in descending order of similarity score

```

36
37 ##Step 5: Compute the Cosine Similarity based on the count_matrix
38 cosine_sim = cosine_similarity(count_matrix)
39
40 movie_user_likes = "Avatar"
41
42 ## Step 6: Get index of this movie from its title
43 movie_index = get_index_from_title(movie_user_likes)
44 similar_movies = list(enumerate(cosine_sim[movie_index]))
45
46
47 ## Step 7: Get a list of similar movies in descending order of similarity score
48 sorted_similar_movies = sorted(similar_movies, key=lambda x: x[1], reverse=True)
49

```

- Step8: Lastly, Print title of first 15 movies

```
C:\Users\robin\OneDrive\Desktop\Movie Recommend\movie_recommender (1)\movie_recommender.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

cosine_similarity x movie_recommender.py x

34 cv = CountVectorizer()
35 count_matrix = cv.fit_transform(df["combined_features"])
36
37 ##Step 5: Compute the Cosine Similarity based on the count_matrix
38 cosine_sim = cosine_similarity(count_matrix)
39
40 movie_user_likes = "Avatar"
41
42 ## Step 6: Get index of this movie from its title
43 movie_index = get_index_from_title(movie_user_likes)
44 similar_movies = list(enumerate(cosine_sim[movie_index]))
45
46
47 ## Step 7: Get a list of similar movies in descending order of similarity score
48 sorted_similar_movies = sorted(similar_movies, key=lambda x:x[1], reverse=True)
49
50 ## Step 8: Print titles of first 50 movies
51 i=0
52 for movie in sorted_similar_movies:
53     print get_title_from_index(movie[0])
54     i=i+1
55     if i>50:
56         break
```

I have taken Imdb datasets of movies here for building this project
<https://datasets.imdbws.com/>.

- Each dataset is contained in a gipped, tab-separated-values (TSV) formatted file in the UTF-8 character set. The first line in each file contains headers that describe what is in each column. A ‘\N’ is used to denote that a particular field is missing or null for that title/name. The available datasets are as follows:
- name.basics.tsv.gz
- title.akas.tsv.gz
- title.basics.tsv.gz
- title.crew.tsv.gz
- title.episode.tsv.gz
- title.principals.tsv.gz
- title.ratings.tsv.gz

Chapter 4: OUTPUT/RESULT/SCREENSHOT

```
C:\Users\robin\OneDrive\Desktop\Project Files and Datasets\movie_recommender2(sublime text).py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

movie_recommender2(sublime text).py x
1 import pandas as pd
2 import numpy as np
3 from sklearn.feature_extraction.text import CountVectorizer
4 from sklearn.metrics.pairwise import cosine_similarity
5 ##### helper functions. Use them when needed #####
6 def get_title_from_index(index):
7     return df[df.index == index]["title"].values[0]
8
9 def get_index_from_title(title):
10    return df[df.title == title]["index"].values[0]
11 #####
12
13 ##Step 1: Read CSV File
14 df = pd.read_csv("movie_dataset.csv")
15 print df.head()
16
17 ##Step 2: Select Features
18 features = ['keywords', 'cast', 'genres', 'director']
19
20 ##Step 3: Create a column in DF which combines all selected features
21 for feature in features:
22     df[feature] = df[feature].fillna('')
23     def combine_features(row):
24         try:
25             return row["keywords"]+" "+row["cast"]+" "+row["genres"]+" "+row["director"]
26         except:
27             print "Error:" , row
28
29
30     df["combined_features"] = df.apply(combine_features,axis=1)
31     print "combined features:", df["combined_features"].head()
32
33 ##Step 4: Create count matrix from this new combined column
34 cv = CountVectorizer()
35 count_matrix = cv.fit_transform(df["combined_features"])
```

```
movie_recommender2(sublime text).py x
27     print "Error:" , row
28
29
30     df["combined_features"] = df.apply(combine_features,axis=1)
31     print "combined features:", df["combined_features"].head()
32
33     ##Step 4: Create count matrix from this new combined column
34     cv = CountVectorizer()
35     count_matrix = cv.fit_transform(df["combined_features"])
36
37     ##Step 5: Compute the Cosine Similarity based on the count_matrix
38     cosine_sim = cosine_similarity(count_matrix)
39
40     movie_user_likes = "Avatar"
41
42     ## Step 6: Get index of this movie from its title
43     movie_index = get_index_from_title(movie_user_likes)
44     similar_movies = list(enumerate(cosine_sim[movie_index]))
45
46
47     ## Step 7: Get a list of similar movies in descending order of similarity score
48     sorted_similar_movies = sorted(similar_movies,key=lambda x:x[1],reverse=True)
49
50     ## Step 8: Print titles of first 50 movies
51     i=0
52     for movie in sorted_similar_movies:
53         print get_title_from_index(movie[0])
54         i=i+1
55         if i>50:
56             break
```



```

Avatar
Guardians of the Galaxy
Aliens
Star Wars: Clone Wars: Volume 1
Star Trek Into Darkness
Star Trek Beyond
Alien
Lockout
Jason X
The Helix... Loaded
Moonraker
Planet of the Apes
Galaxy Quest
Gravity
Alien³

```

C:\Users\robin\OneDrive\Desktop\Movie Recommend\movie_recommender(1)\movie_recommender.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

```

cosine_similarity x movie_recommender.py x
44 similar_movies = list(sorted(similarity_scores[similarity_index]))
45
46
47 ## Step 7: Get a list of similar movies in descending order of similarity score
48 sorted_similar_movies = sorted(similar_movies, key=lambda x:x[1], reverse=True)
49
50 ## Step 8: Print titles of first 50 movies
51 i=0
52 for movie in sorted_similar_movies:
53     print get_title_from_index(movie[0])
54     i=i+1
55     if i>50:
56         break

```

combined features: 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 m16 Dan...
3 dc comics crime fighter terrorist secret ident...
4 based on novel mars medallion space travel pri...

```

Name: combined_features, dtype: object

```

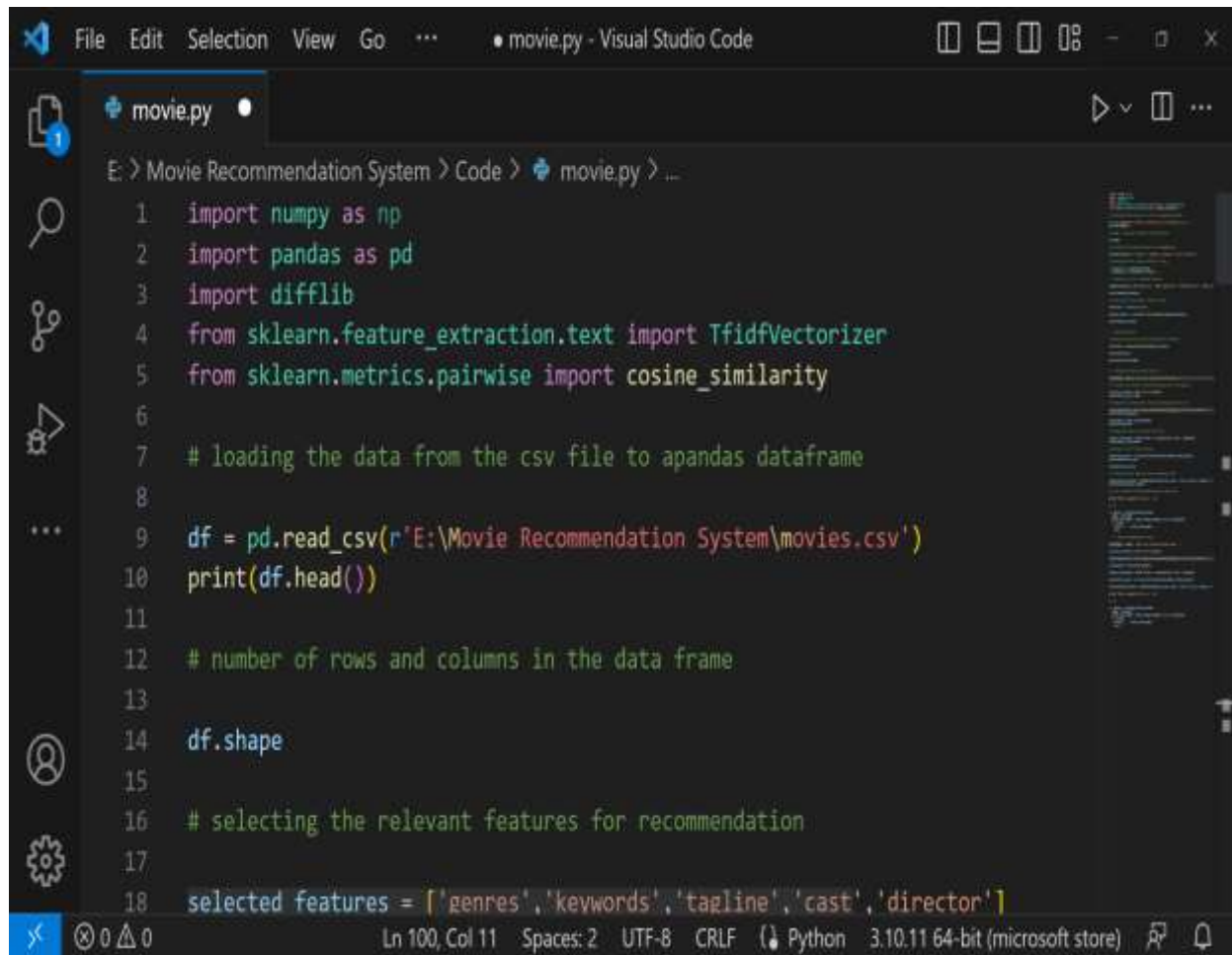
Avatar
Guardians of the Galaxy
Aliens
Star Wars: Clone Wars: Volume 1
Star Trek Into Darkness
Star Trek Beyond
Alien
Lockout
Jason X
The Helix... Loaded
Moonraker
Planet of the Apes

```

Line 48, Column 77

Type here to search

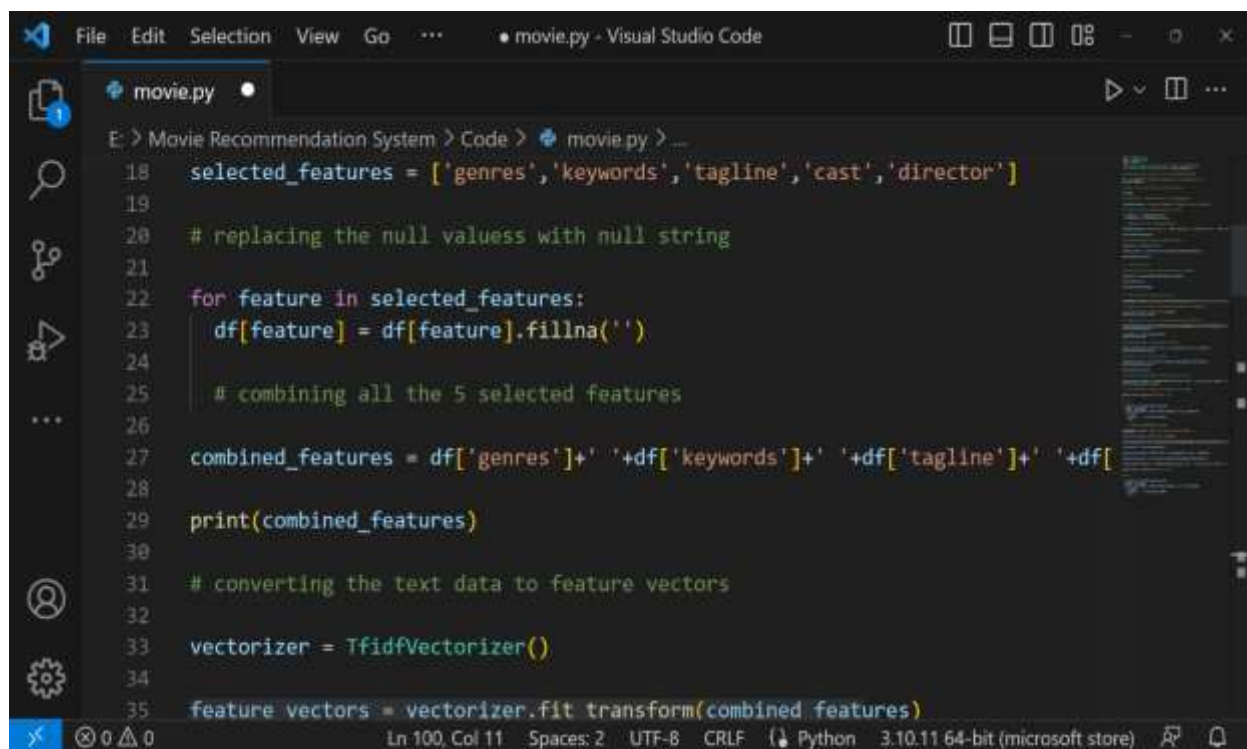
Chapter 5: Testing



```
File Edit Selection View Go ... • movie.py - Visual Studio Code

movie.py
E: > Movie Recommendation System > Code > movie.py > ...
1 import numpy as np
2 import pandas as pd
3 import difflib
4 from sklearn.feature_extraction.text import TfidfVectorizer
5 from sklearn.metrics.pairwise import cosine_similarity
6
7 # loading the data from the csv file to apandas dataframe
8
9 df = pd.read_csv(r'E:\Movie Recommendation System\movies.csv')
10 print(df.head())
11
12 # number of rows and columns in the data frame
13
14 df.shape
15
16 # selecting the relevant features for recommendation
17
18 selected_features = ['genres', 'keywords', 'tagline', 'cast', 'director']
```

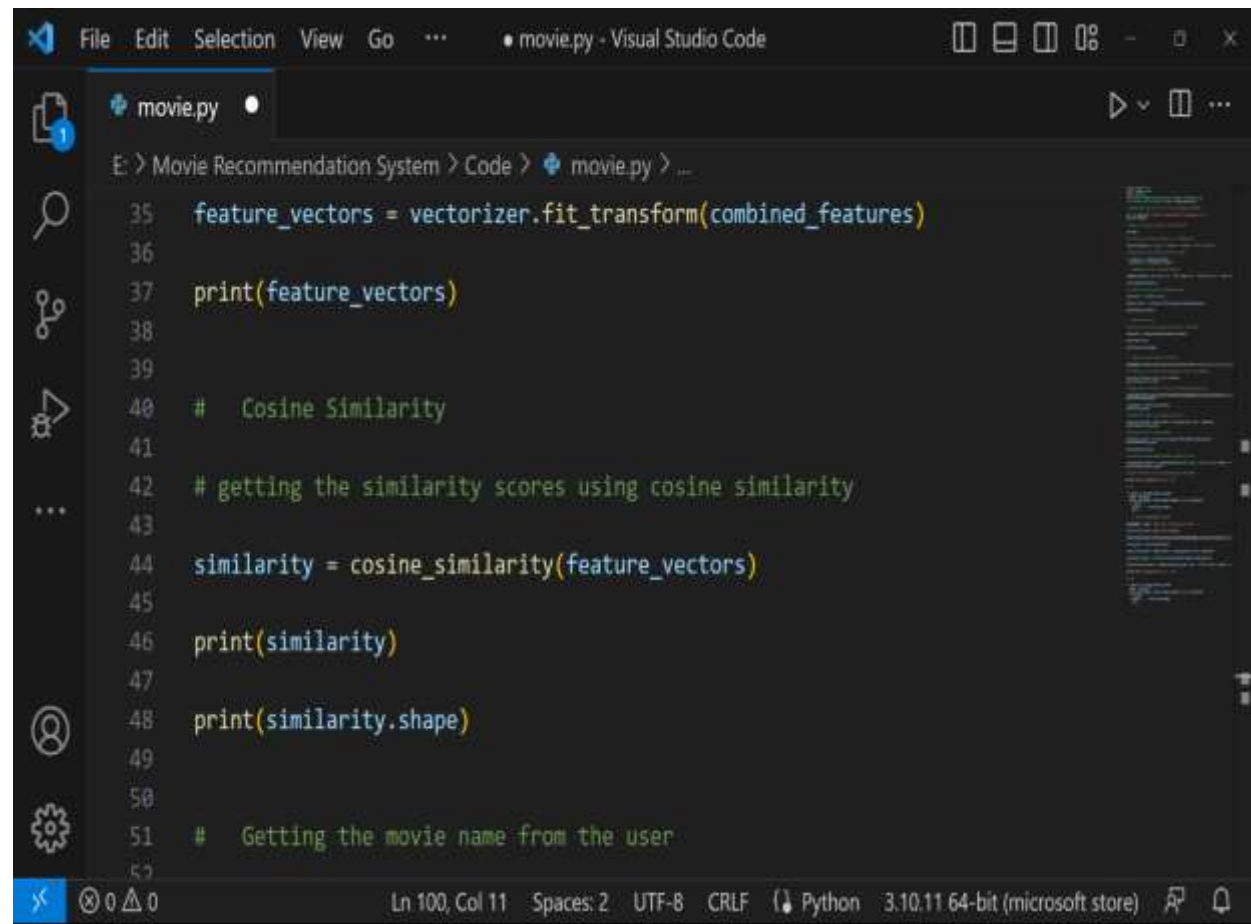
Ln 100, Col 11 Spaces: 2 UTF-8 CRLF Python 3.10.11 64-bit (microsoft store)



```
File Edit Selection View Go ... • movie.py - Visual Studio Code

movie.py
E: > Movie Recommendation System > Code > movie.py > ...
18 selected_features = ['genres', 'keywords', 'tagline', 'cast', 'director']
19
20 # replacing the null valuess with null string
21
22 for feature in selected_features:
23     df[feature] = df[feature].fillna('')
24
25     # combining all the 5 selected features
26
27 combined_features = df['genres']+' '+df['keywords']+' '+df['tagline']+' '+df[
28
29 print(combined_features)
30
31 # converting the text data to feature vectors
32
33 vectorizer = TfidfVectorizer()
34
35 feature_vectors = vectorizer.fit transform(combined_features)
```

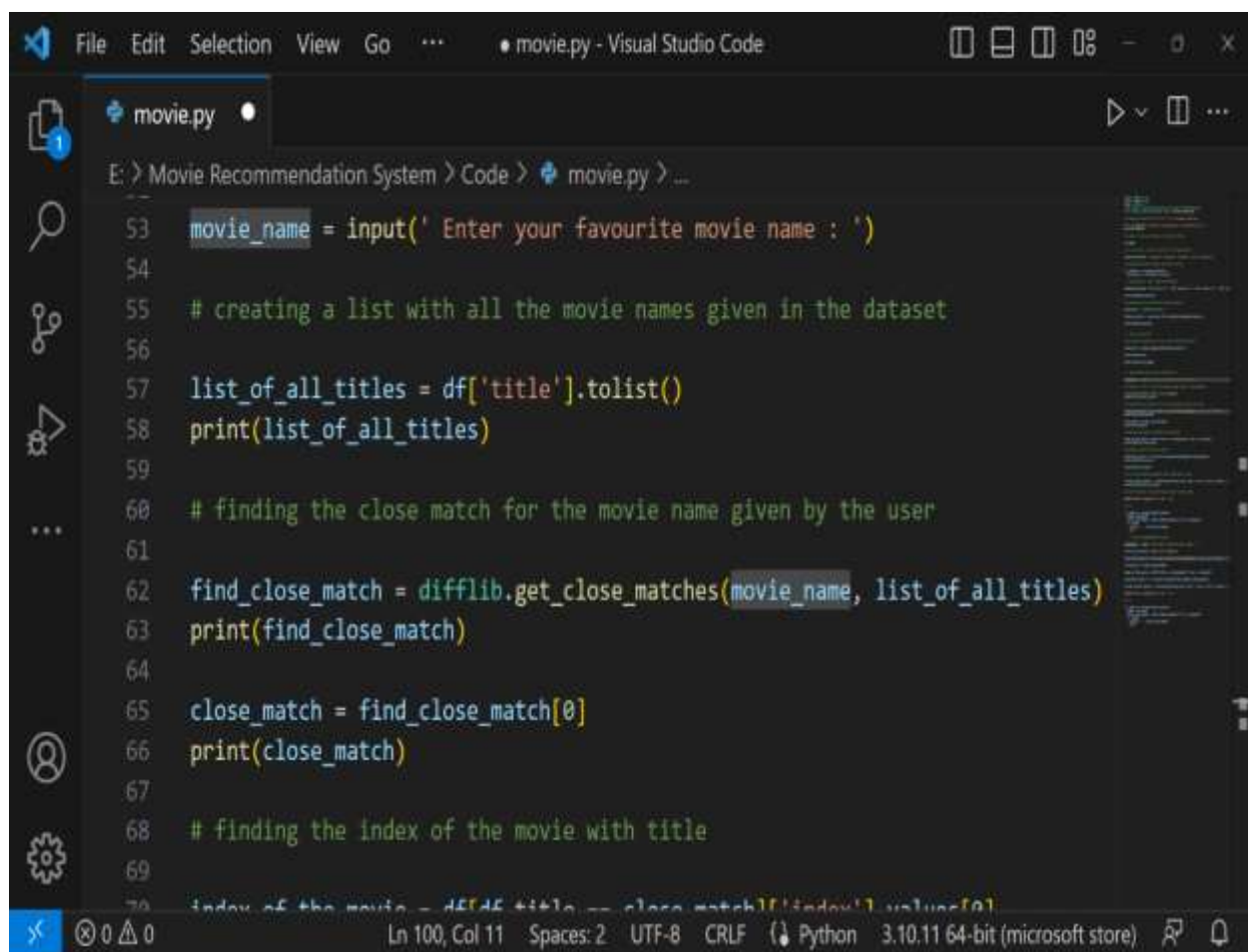
Ln 100, Col 11 Spaces: 2 UTF-8 CRLF Python 3.10.11 64-bit (microsoft store)



The screenshot shows the Visual Studio Code editor with a file named `movie.py` open. The code is in Python and is part of a movie recommendation system. The visible code lines are:

```
35 feature_vectors = vectorizer.fit_transform(combined_features)
36
37 print(feature_vectors)
38
39
40 # Cosine Similarity
41
42 # getting the similarity scores using cosine similarity
43
44 similarity = cosine_similarity(feature_vectors)
45
46 print(similarity)
47
48 print(similarity.shape)
49
50
51 # Getting the movie name from the user
```

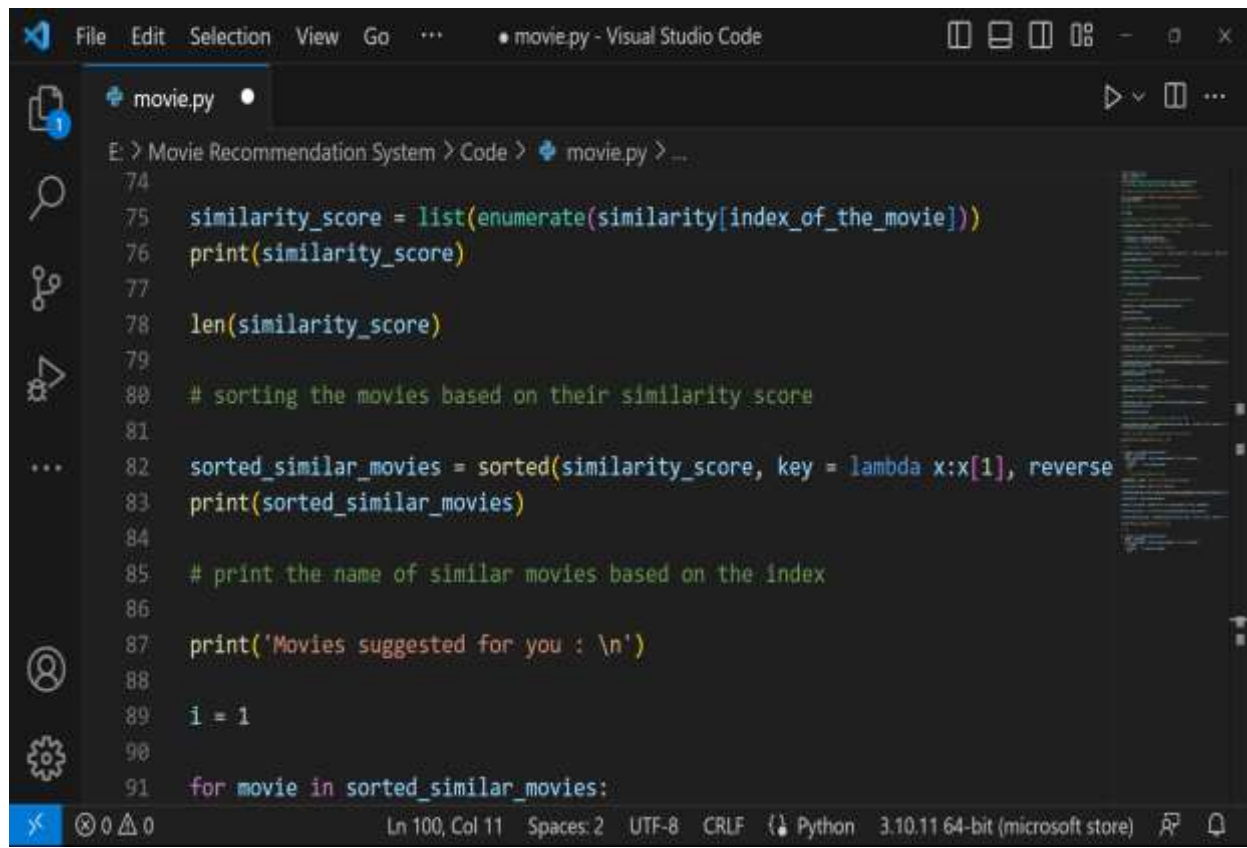
The status bar at the bottom indicates the current position is Line 100, Column 11, with 2 spaces, UTF-8 encoding, CRLF line endings, Python 3.10.11 64-bit (microsoft store).



The screenshot shows the Visual Studio Code editor with the same file `movie.py` open. The code continues from the previous snippet, showing the process of finding a close match for a user-provided movie name. The visible code lines are:

```
53 movie_name = input(' Enter your favourite movie name : ')
54
55 # creating a list with all the movie names given in the dataset
56
57 list_of_all_titles = df['title'].tolist()
58 print(list_of_all_titles)
59
60 # finding the close match for the movie name given by the user
61
62 find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)
63 print(find_close_match)
64
65 close_match = find_close_match[0]
66 print(close_match)
67
68 # finding the index of the movie with title
69
70 index_of_the_movie = df[df['title'] == close_match]['index'].values[0]
```

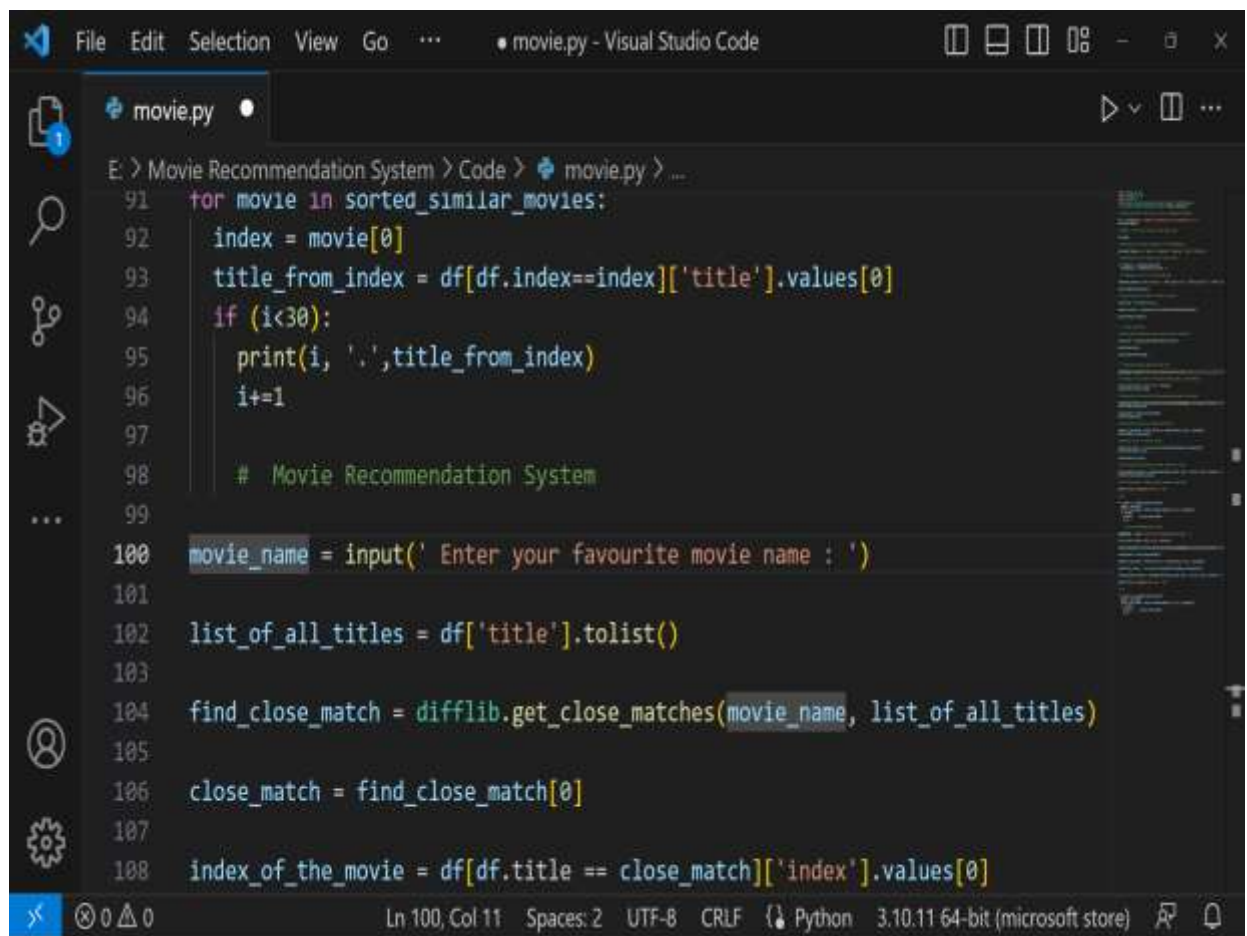
The status bar at the bottom indicates the current position is Line 100, Column 11, with 2 spaces, UTF-8 encoding, CRLF line endings, Python 3.10.11 64-bit (microsoft store).



```
File Edit Selection View Go ... • movie.py - Visual Studio Code

E: > Movie Recommendation System > Code > • movie.py > ...
74
75 similarity_score = list(enumerate(similarity[index_of_the_movie]))
76 print(similarity_score)
77
78 len(similarity_score)
79
80 # sorting the movies based on their similarity score
81
82 sorted_similar_movies = sorted(similarity_score, key = lambda x:x[1], reverse
83 print(sorted_similar_movies)
84
85 # print the name of similar movies based on the index
86
87 print('Movies suggested for you : \n')
88
89 i = 1
90
91 for movie in sorted_similar_movies:
```

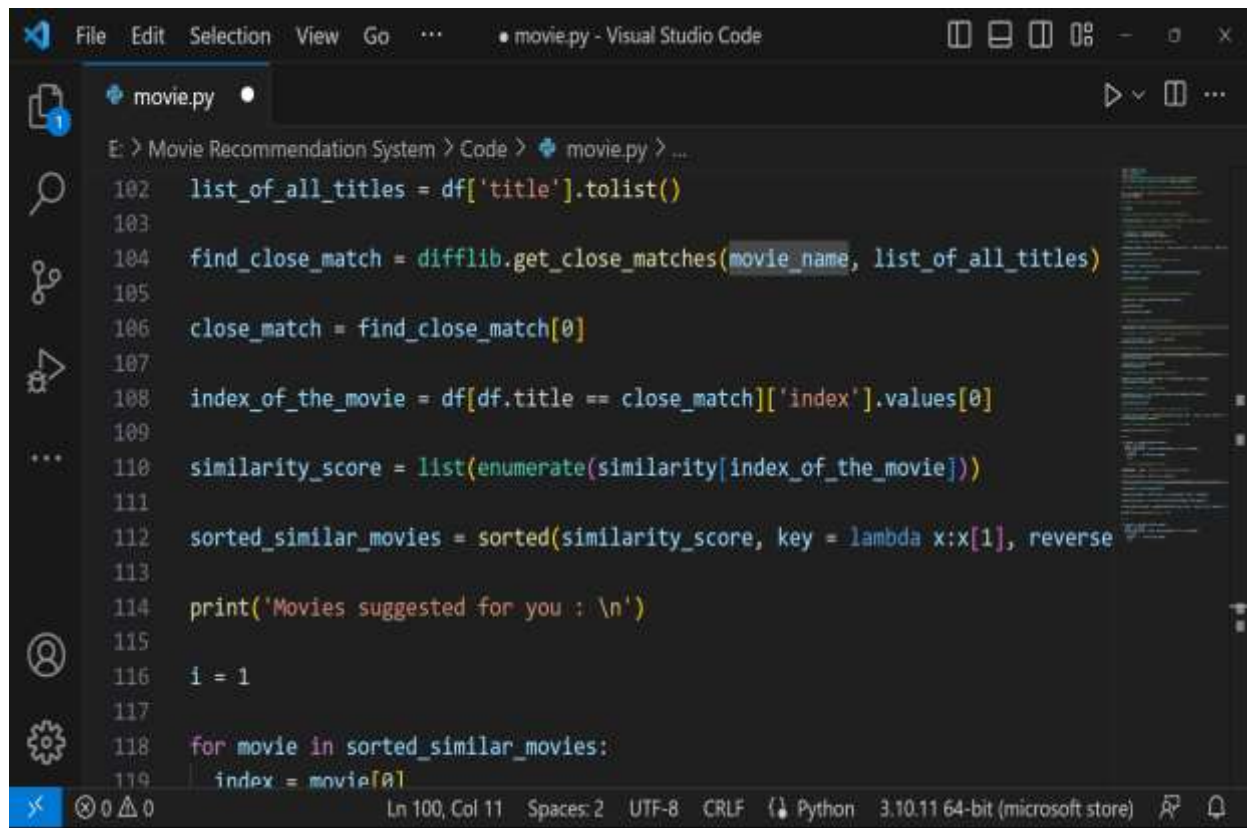
Ln 100, Col 11 Spaces: 2 UTF-8 CRLF Python 3.10.11 64-bit (microsoft store)



```
File Edit Selection View Go ... • movie.py - Visual Studio Code

E: > Movie Recommendation System > Code > • movie.py > ...
91 for movie in sorted_similar_movies:
92     index = movie[0]
93     title_from_index = df[df.index==index]['title'].values[0]
94     if (i<30):
95         print(i, '.',title_from_index)
96         i+=1
97
98     # Movie Recommendation System
99
100 movie_name = input(' Enter your favourite movie name : ')
101
102 list_of_all_titles = df['title'].tolist()
103
104 find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)
105
106 close_match = find_close_match[0]
107
108 index_of_the_movie = df[df.title == close_match]['index'].values[0]
```

Ln 100, Col 11 Spaces: 2 UTF-8 CRLF Python 3.10.11 64-bit (microsoft store)



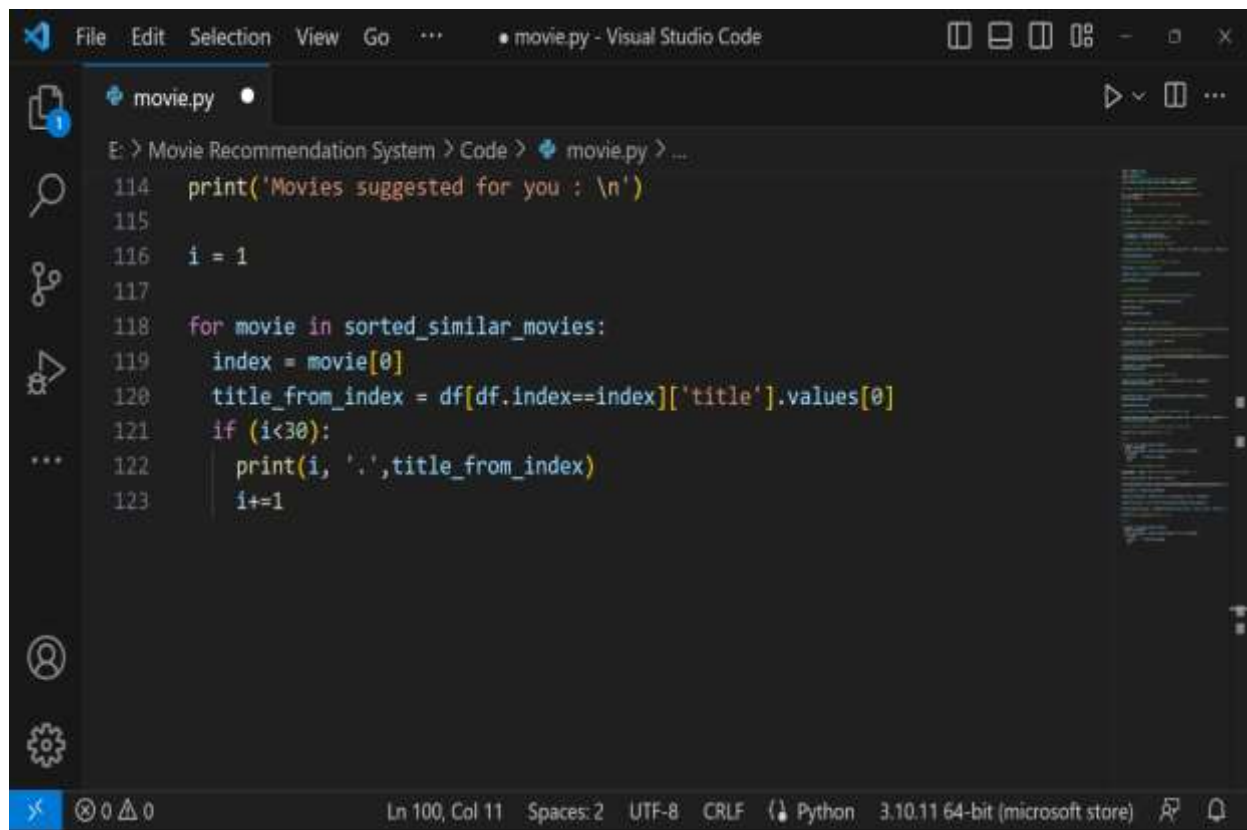
This screenshot shows the Visual Studio Code editor with a Python file named `movie.py`. The code is part of a movie recommendation system. It uses `difflib` to find close matches for a given movie name from a list of all titles. The code then finds the index of the movie in a DataFrame, calculates similarity scores, sorts them, and prints the suggested movies.

```
File Edit Selection View Go ... • movie.py - Visual Studio Code

E: > Movie Recommendation System > Code > • movie.py > ...

102 list_of_all_titles = df['title'].tolist()
103
104 find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)
105
106 close_match = find_close_match[0]
107
108 index_of_the_movie = df[df.title == close_match]['index'].values[0]
109
110 similarity_score = list(enumerate(similarity[index_of_the_movie]))
111
112 sorted_similar_movies = sorted(similarity_score, key = lambda x:x[1], reverse
113
114 print('Movies suggested for you : \n')
115
116 i = 1
117
118 for movie in sorted_similar_movies:
119     index = movie[0]
```

Ln 100, Col 11 Spaces: 2 UTF-8 CRLF Python 3.10.11 64-bit (microsoft store)



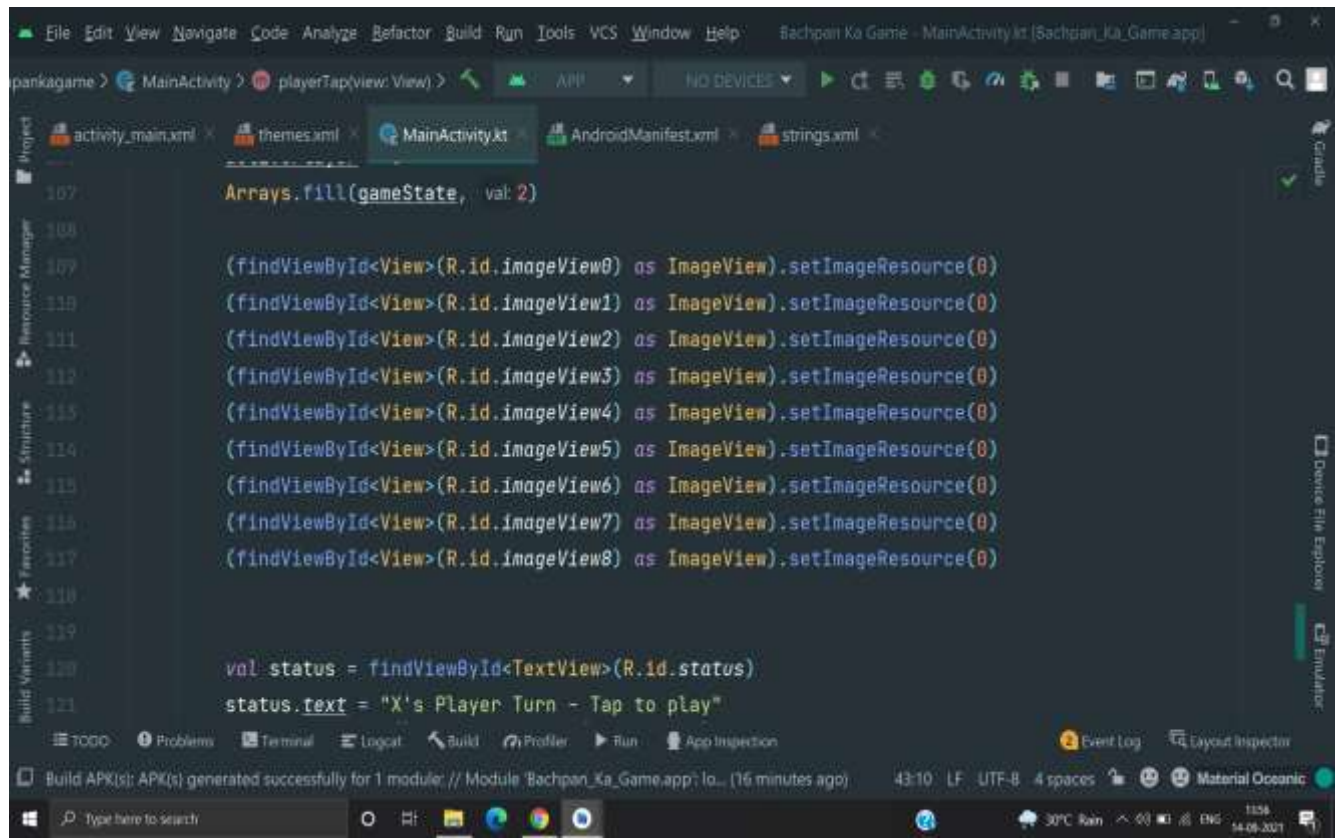
This screenshot shows the continuation of the Python code in `movie.py`. It prints the suggested movies, increments a counter `i`, and loops through the sorted similar movies. For each movie, it finds the index, retrieves the title from the DataFrame, and prints it if the counter is less than 30.

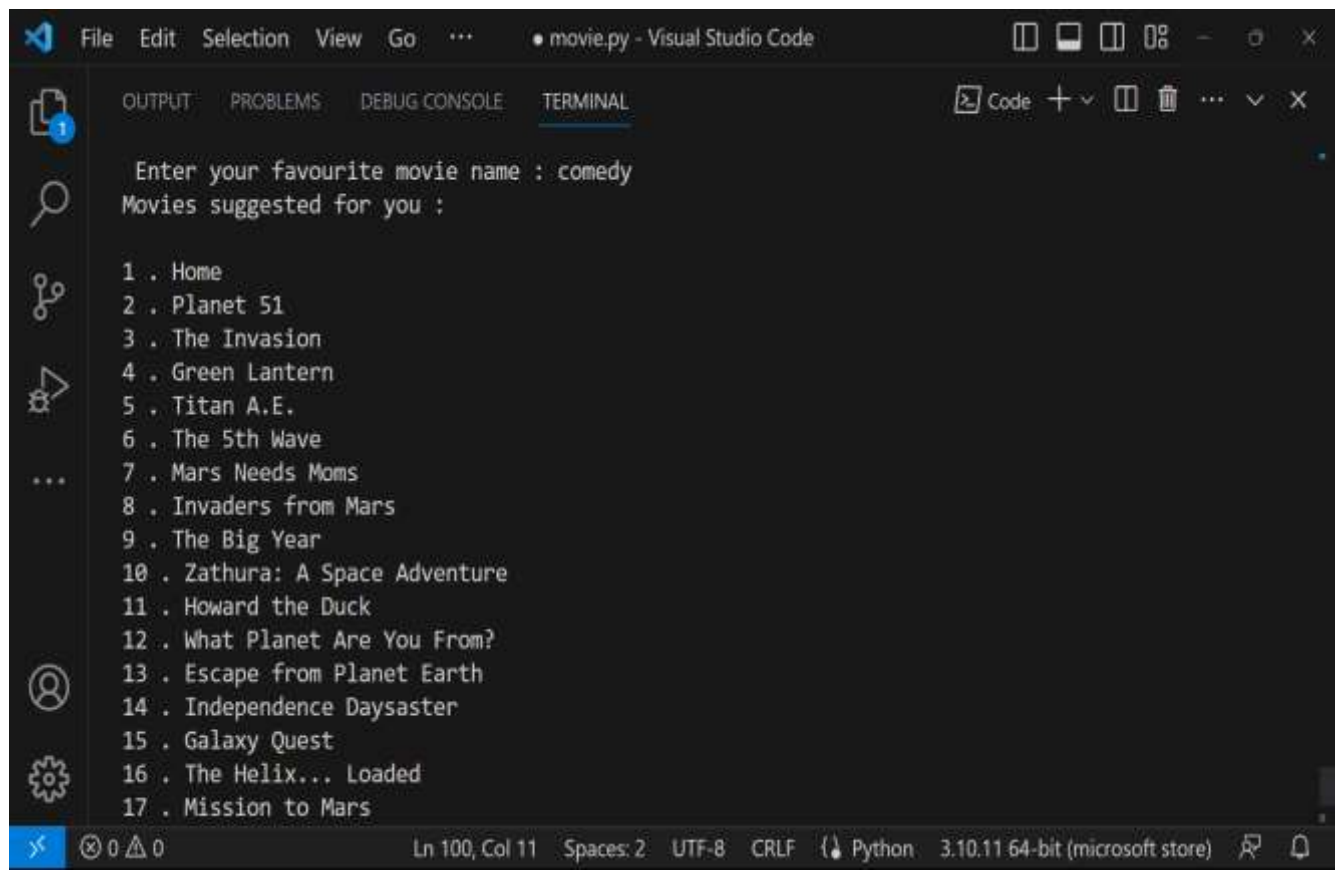
```
File Edit Selection View Go ... • movie.py - Visual Studio Code

E: > Movie Recommendation System > Code > • movie.py > ...

114 print('Movies suggested for you : \n')
115
116 i = 1
117
118 for movie in sorted_similar_movies:
119     index = movie[0]
120     title_from_index = df[df.index==index]['title'].values[0]
121     if (i<30):
122         print(i, '.',title_from_index)
123         i+=1
```

Ln 100, Col 11 Spaces: 2 UTF-8 CRLF Python 3.10.11 64-bit (microsoft store)





The image shows a Visual Studio Code window with a terminal open. The terminal title is "movie.py - Visual Studio Code". The terminal output shows a prompt "Enter your favourite movie name : comedy" followed by "Movies suggested for you :". Below this is a numbered list of 17 movie titles. The status bar at the bottom indicates the file is at line 100, column 11, with 2 spaces, UTF-8 encoding, CRLF line endings, and Python 3.10.11 64-bit (microsoft store) interpreter.

```
File Edit Selection View Go ... • movie.py - Visual Studio Code
OUTPUT PROBLEMS DEBUG CONSOLE TERMINAL Code + - [ ] [ ] [ ] 0% - [ ] [ ] [ ] [ ]
1
Enter your favourite movie name : comedy
Movies suggested for you :

1 . Home
2 . Planet 51
3 . The Invasion
4 . Green Lantern
5 . Titan A.E.
6 . The 5th Wave
7 . Mars Needs Moms
8 . Invaders from Mars
9 . The Big Year
10 . Zathura: A Space Adventure
11 . Howard the Duck
12 . What Planet Are You From?
13 . Escape from Planet Earth
14 . Independence Daysaster
15 . Galaxy Quest
16 . The Helix... Loaded
17 . Mission to Mars

Ln 100, Col 11 Spaces: 2 UTF-8 CRLF Python 3.10.11 64-bit (microsoft store)
```

output

Chapter 7: CONCLUSION/FUTURE ENHANCEMENT

- In this project we have implemented and learn the following things such as-
- Building a Movie Recommendation System
- To find the Similarity Scores and Indexes.
- Compute Distance Between Two Vectors
- Cosine Similarity
- To find Euclidian Distance and many more ML related concepts and techniques.

Research paper recommender systems help library users in finding or getting most relevant research papers over a large volume of research papers in a digital library. This paper adopted content-based filtering technique to provide recommendations to the intended users. Based on the results of the system, integrating recommendation features in digital libraries would be useful to library users.

The solution to this problem came as a result of the availability of the contents describing the items and users' profile of interest. Content-based techniques are independent of the users ratings but depend on these contents. This paper also presents an algorithm to provide or suggest recommendations based on the users' query. The algorithm employs both TF-IDF weighing and cosine similarity measure

The next step of our future work is to adopt hybrid algorithm to see how the combination of collaborative and content-based filtering techniques can gives us a better recommendation compared to the adopted technique in this paper.

The content-based technique is adopted or considered here for the design of the recommender system for digital libraries. Content-based technique is suitable in situations or domains where items are more than users.

Library users do experience difficulties in getting or finding favourite digital objects (e.g. research papers) from a large collection of digital objects in digital libraries.

Chapter 8: REFERENCES

- <https://drive.google.com/file/d/1sJ9N.>
- <https://pdfs.semanticscholar.org/c9f9/6d22422953625f1f8d9dbec221cba38e6c08.pdf>
- <https://www.google.com/search?sxsrf=ALeKk00P037U5bp3y51QqWE-wCkvkCDUKw:1588619431266&source=univ&tbm=isch&q=content+based+filterin+g+diagrams&sa=X&ved=2ahUKEwi33YjH9JrpAhWFXSsKHR56DOAQsAR6BAgKEAE&biw=1366&bih=625>
- <https://www.imdb.com/list/ls063596142/>
- <https://towardsdatascience.com/the-4-recommendation-engines-that-can-predict-your-movie-tastes-109dc4e10c52?gi=61b501c11dd4>
- <https://www.google.com/search?q=numpy+in+python&oq=numpy+in+python&aqs=chrome..69i57j0l6j69i60.4625j0j7&sourceid=chrome&ie=UTF-8>
- <https://www.google.com/search?q=pip+python&oq=pip&aqs=chrome.2.69i57j0l6j69i60.3726j0j7&sourceid=chrome&ie=UTF-8>
- <https://www.google.com/search?q=sublime+text+python&oq=sublime+&aqs=chrome.2.69i57j35i39l2j0j69i60l4.4504j0j7&sourceid=chrome&ie=UTF-8>
- <https://www.google.com/search?q=cosine+similarity+python&oq=cosine&aqs=chrome.5.69i57j0l7.6310j0j7&sourceid=chrome&ie=UTF-8>
- <https://docs.python.org/3/tutorial>
- https://en.wikipedia.org/wiki/Recommender_system
- https://www.learnpython.org/en/Pandas_Basics
- <https://www.google.com/search?q=sublime+text+python&oq=sublime+&aqs=chrome.2.69i57j35i39l2j0j69i60l4.4504j0j7&sourceid=chrome&ie=UTF-8>



Yours Sincerely,

Himanshu Kumar

(1902250100065)

Md Sibgatullah Amir

(1902250100082)

Mohd Aman

(1902250100083)