“Movie Recommendation System” – By Mohamed Shabeeb

Department of Software Engineering

Faculty of Computing

Sabaragamuwa University of Sri Lanka

February 2025

1 DECLARATION

I hereby declare that the report entitled “Movie Recommendation System” was submitted to Venturit Inc. The report submitted herewith of the results of my own effort in totality and every aspect of the project works. All information that has been obtained from other sources had been fully acknowledged.

27-02-2025	Shabeeb
Date	A.I.M. Sabeeb
	19APSE4289

2 ACKNOWLEDGMENT

I would like to extend my sincere gratitude to **Venturit Inc.** for providing me with the opportunity to work on the development of the movie recommendation system. Their support and trust in my abilities allowed me to explore and apply various machine learning techniques, enhancing my skills in data processing and model implementation.

A special thanks to the team at **Venturit Inc.** for their guidance and resources, which were instrumental in the completion of this project. I truly appreciate their commitment to fostering a learning environment and providing hands-on experiences that contribute to my growth as a data science professional.

Thank you once again for this invaluable opportunity

3 ABSTRACT

This report presents the development and implementation of a Movie Recommendation System using Content-Based Filtering and Item-Based Collaborative Filtering. The system leverages the TMDB Movies Dataset, which includes key attributes such as movie title, genre, overview, and user ratings. The primary objective of the system is to recommend similar movies to users based on both the content of movies they have already shown interest in and patterns in user ratings.

The system first preprocesses the textual data, combining the overview and genre columns into a unified feature called "tags". These tags are then cleaned through tokenization, stopword removal, and lemmatization. The preprocessed data is vectorized using CountVectorizer, transforming the textual data into numerical form suitable for similarity computation.

Cosine similarity is used to measure the similarity between movies based on their content. The system ranks the movies according to their similarity and generates a list of the top 5 most similar movies for each user input. To enhance the utility of the system, the movie data and similarity matrix are serialized using Pickle for efficient model deployment and retrieval.

The content-based approach ensures that recommendations are personalized and based on the intrinsic features of the movies themselves, offering users a tailored viewing experience. This report also outlines the process of building, testing, and deploying the recommendation model using Python and machine learning techniques.

Unlike content-based filtering, which analyzes movie descriptions, item-based collaborative filtering recommends movies based on user rating patterns. This method assumes that if two movies receive similar ratings from multiple users, they are likely to be similar. The system constructs a user-item matrix, where rows represent movies, columns represent users and values indicate ratings given by users to movies.

Keywords: Movie Recommendation System, Content Filtering, Collaborative Filtering, TMDB Movie Dataset, Cosine Similarity, Pickle, Streamlit.

4 TABLE OF CONTENTS

5 LIST OF FIGURES

6 LIST OF TABLES

7 LIST OF ABBREVIATION

TMDb	The Movie Database
ID	Identifier
ML	Machine Learning
API	Application Programming Interface
NLTK	Natural Language Toolkit

9 CHAPTER 01: MOVIE RECOMMENDATION SYSTEM

A movie recommendation system is a type of system that suggests movies to users based on certain criteria. The goal is to predict what movies a user might like based on their preferences, viewing history, and behavior. These systems typically use machine learning algorithms and data analysis techniques to generate personalized movie recommendations. There are several types of recommendation systems:

1. **Collaborative Filtering**: This is one of the most common techniques. It uses the past behavior of users to recommend movies. There are two main types:

   - **User-based collaborative filtering**: Recommends movies that similar users have liked.

   - **Item-based collaborative filtering**: Recommends movies that are similar to ones the user has already liked.

2. **Content-based Filtering**: This method recommends movies based on the content the user has previously enjoyed. For example, if a user liked a science fiction movie, the system might recommend other science fiction films.

3. **Hybrid Systems**: These systems combine both collaborative filtering and content-based filtering to improve recommendations.

4. **Knowledge-based Systems**: This type of system recommends movies based on user preferences (like genre, director, or actors) rather than historical behavior.

Movie recommendation systems are widely used on platforms like Netflix, Amazon Prime, and YouTube to enhance user experience by providing personalized content suggestions.

10 CHAPTER 02: DATASET SELECTION

For building the movie recommendation system, I selected the TMDB Movies Dataset from Kaggle. This dataset contains detailed information about movies, including movie titles, genres, popularity, vote averages, and more. This rich set of attributes makes it ideal for building a personalized recommendation system. The dataset provides valuable insights into user preferences, movie characteristics, and trends, enabling the system to recommend movies based on various factors.

Dataset 01 Overview – 'dataset.csv':

- **ID**: Unique movie identifier on the TMDB website, helping to distinguish between different movies.
- **Title**: The name of the movie.
- **Genre**: Categorized genre of the movie, such as crime, adventure, action, etc. This information can help in filtering movies by user preferences in specific genres.
- **Original Language**: The language in which the movie was originally released. This helps in recommending movies based on user's language preference.
- **Overview**: A short summary or description of the movie, which can be used to provide more context about the movie to the user.
- **Popularity**: The movie's popularity score, which can be a factor in determining trending or widely accepted movies.
- **Release Date**: The release date of the movie, which is useful for time-based recommendations (e.g., recent releases).
- **Vote Average**: Average rating given by users, which helps determine the quality or user reception of a movie.
- **Vote Count**: Total number of votes given by users. This gives insight into the movie's exposure and how popular it is among users.

Dataset 02 Overview – ‘movies.csv’:

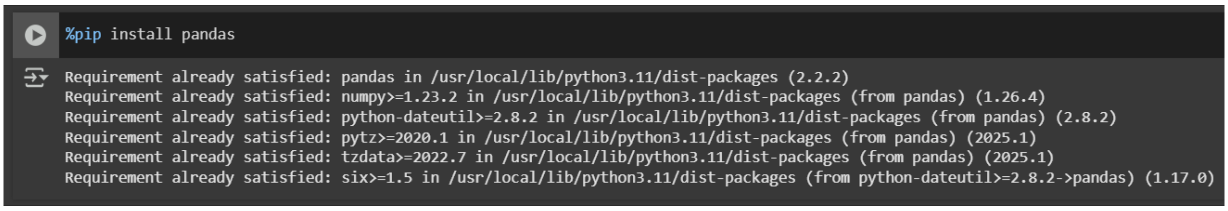
- **movieId**: Unique movie identifier helping to distinguish between different movies.
- **title**: The name of the movie.

Dataset 03 Overview – ‘ratings.csv’:

- **userId**: Unique user identifier helping to distinguish between different users.
- **movieId**: Unique movie identifier helping to distinguish between different movies.
- **rating**: The ratings given for different movies by users.

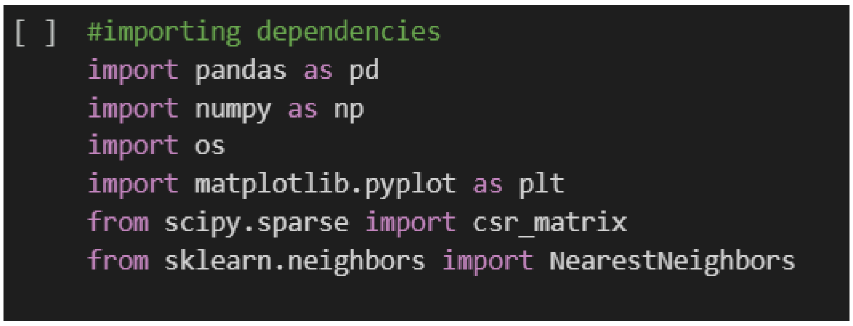
11 CHAPTER 03: DATA PREPROCESSING

For the data preprocessing phase of the movie recommendation system, I utilized Google Colab as the primary platform. Google Colab provides a powerful and convenient environment for data analysis and machine learning tasks with its seamless integration of Python libraries such as Pandas, NumPy, and Matplotlib. In this project, I began by loading the dataset from Kaggle, followed by inspecting the data structure using methods like info() and describe() to understand the data types and detect any missing values. I performed several preprocessing steps, including selecting relevant columns, handling missing values by either dropping or filling null entries, and creating new features, such as concatenating the overview and genre columns to form a tags feature. Google Colab's cloud-based infrastructure allowed me to efficiently process the data and run the necessary transformations, ensuring that the dataset was clean and ready for building the recommendation system.



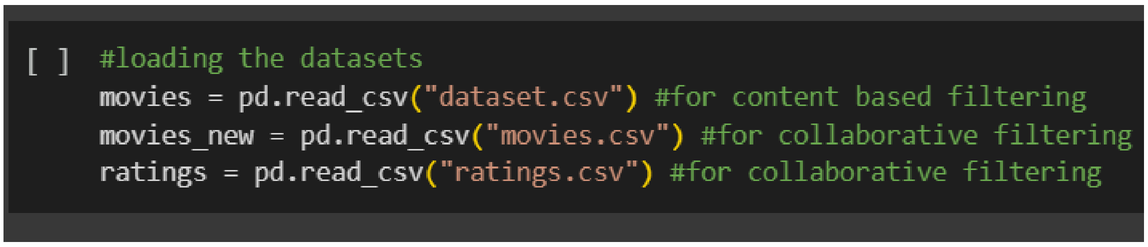
```
%pip install pandas

Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

Figure 1 - Installing Pandas



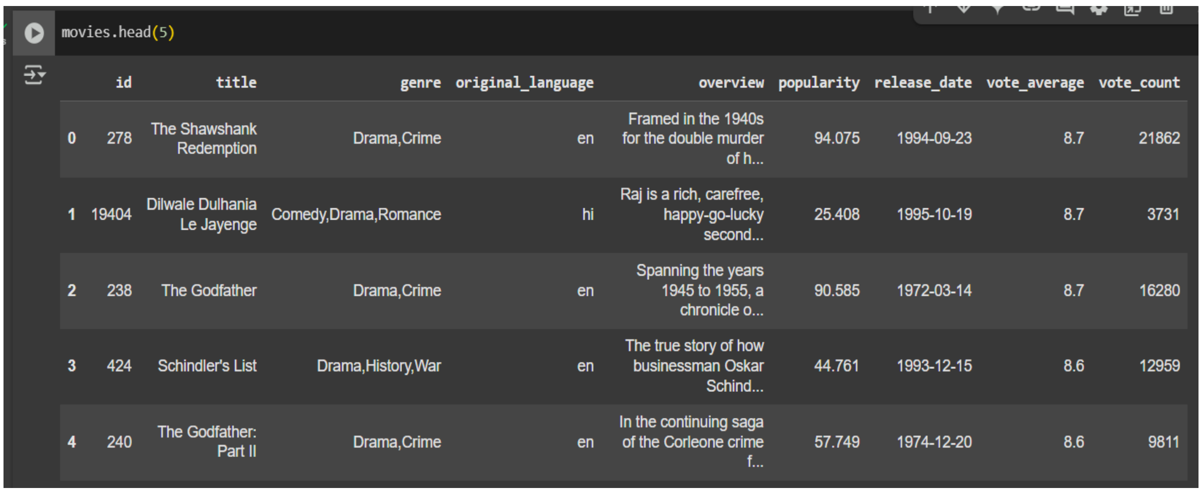
```
[ ]  #importing dependencies
     import pandas as pd
     import numpy as np
     import os
     import matplotlib.pyplot as plt
     from scipy.sparse import csr_matrix
     from sklearn.neighbors import NearestNeighbors
```

Figure 2 - Importing Dependencies



```
[ ]  #loading the datasets
     movies = pd.read_csv("dataset.csv") #for content based filtering
     movies_new = pd.read_csv("movies.csv") #for collaborative filtering
     ratings = pd.read_csv("ratings.csv") #for collaborative filtering
```

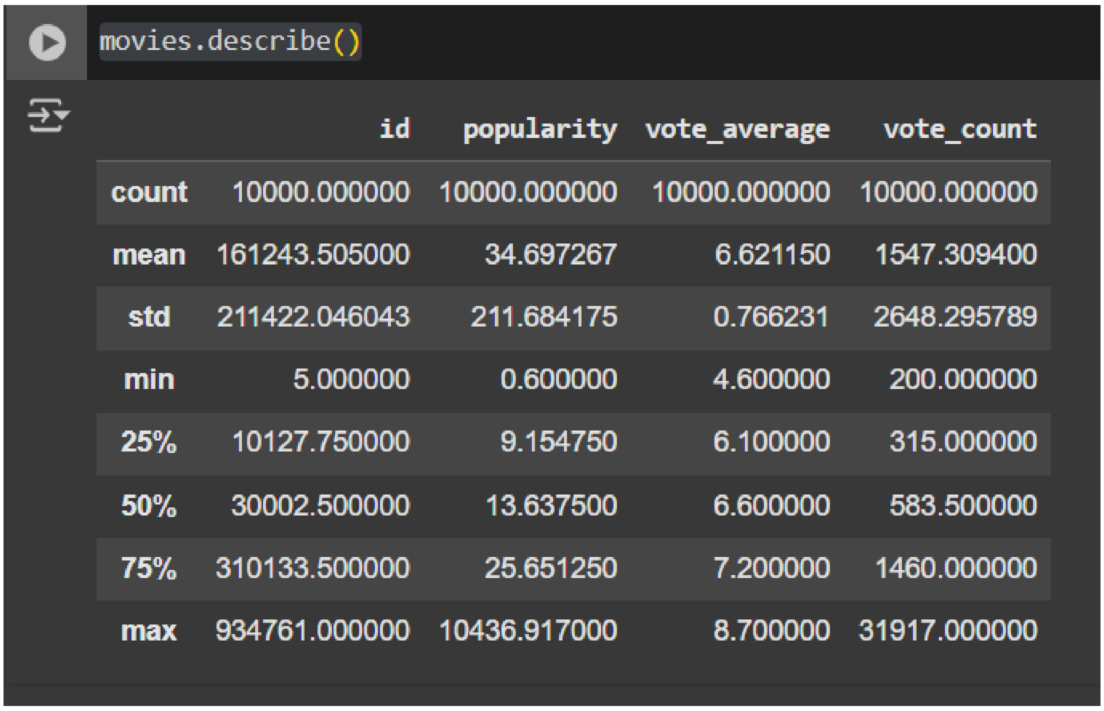
Figure 3 - Loading the datasets



```
movies.head(5)
```

	id	title	genre	original_language	overview	popularity	release_date	vote_average	vote_count
0	278	The Shawshank Redemption	Drama,Crime	en	Framed in the 1940s for the double murder of h...	94.075	1994-09-23	8.7	21862
1	19404	Dilwale Dulhania Le Jayenge	Comedy,Drama,Romance	hi	Raj is a rich, carefree, happy-go-lucky second...	25.408	1995-10-19	8.7	3731
2	238	The Godfather	Drama,Crime	en	Spanning the years 1945 to 1955, a chronicle o...	90.585	1972-03-14	8.7	16280
3	424	Schindler's List	Drama,History,War	en	The true story of how businessman Oskar Schind...	44.761	1993-12-15	8.6	12959
4	240	The Godfather: Part II	Drama,Crime	en	In the continuing saga of the Corleone crime f...	57.749	1974-12-20	8.6	9811

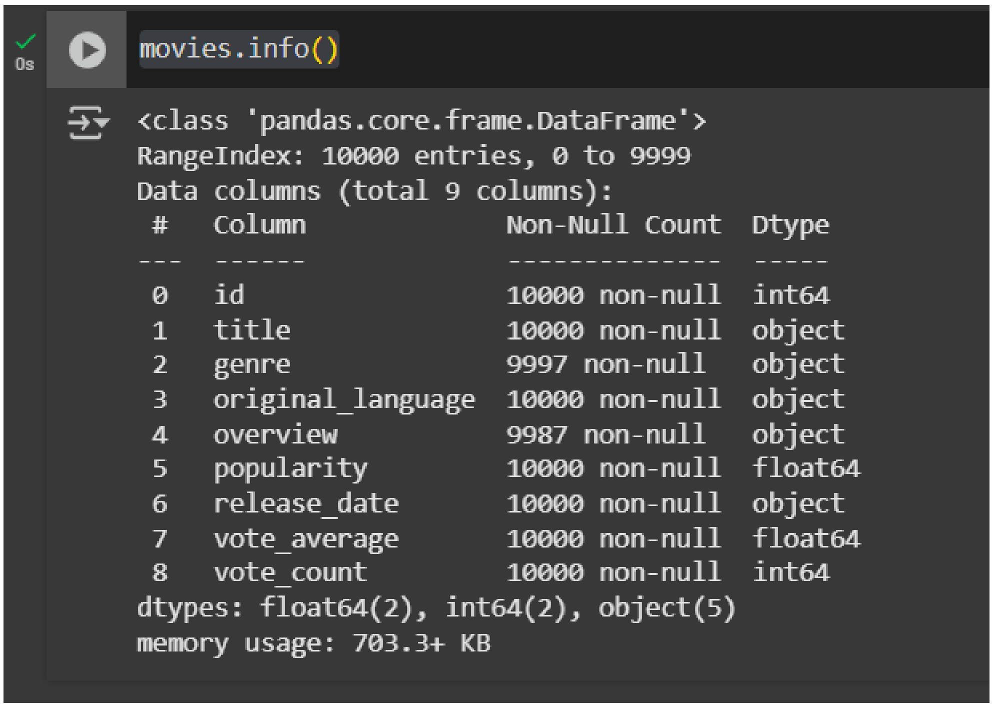
Figure 4 - Retrieving the First 05 Rows of Data



```
movies.describe()
```

	id	popularity	vote_average	vote_count
count	10000.000000	10000.000000	10000.000000	10000.000000
mean	161243.505000	34.697267	6.621150	1547.309400
std	211422.046043	211.684175	0.766231	2648.295789
min	5.000000	0.600000	4.600000	200.000000
25%	10127.750000	9.154750	6.100000	315.000000
50%	30002.500000	13.637500	6.600000	583.500000
75%	310133.500000	25.651250	7.200000	1460.000000
max	934761.000000	10436.917000	8.700000	31917.000000

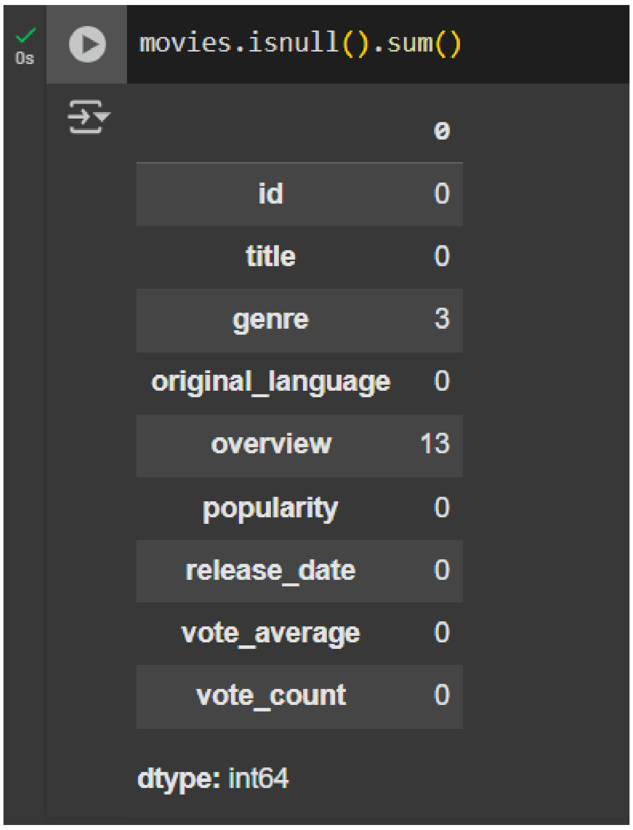
Figure 5 - Generating a Descriptive Statistics of the Dataset



```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 10000 non-null  int64
 1   title              10000 non-null  object
 2   genre              9997 non-null   object
 3   original_language  10000 non-null  object
 4   overview           9987 non-null   object
 5   popularity         10000 non-null  float64
 6   release_date       10000 non-null  object
 7   vote_average       10000 non-null  float64
 8   vote_count         10000 non-null  int64
dtypes: float64(2), int64(2), object(5)
memory usage: 703.3+ KB
```

Figure 6 - Generating a Concise Summary of the DataFrame

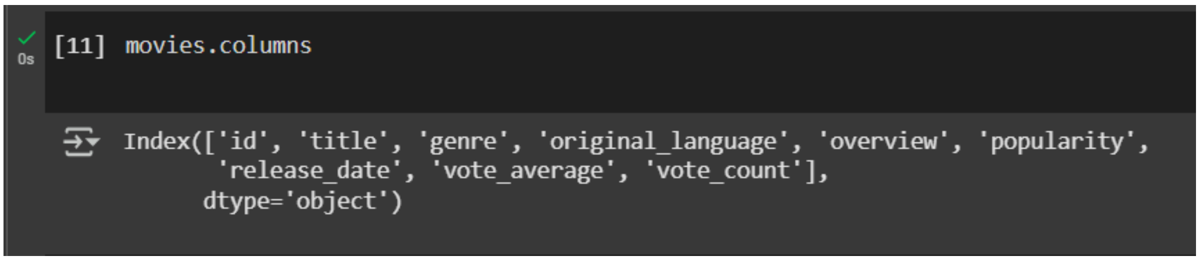


```
movies.isnull().sum()
```

	0
id	0
title	0
genre	3
original_language	0
overview	13
popularity	0
release_date	0
vote_average	0
vote_count	0

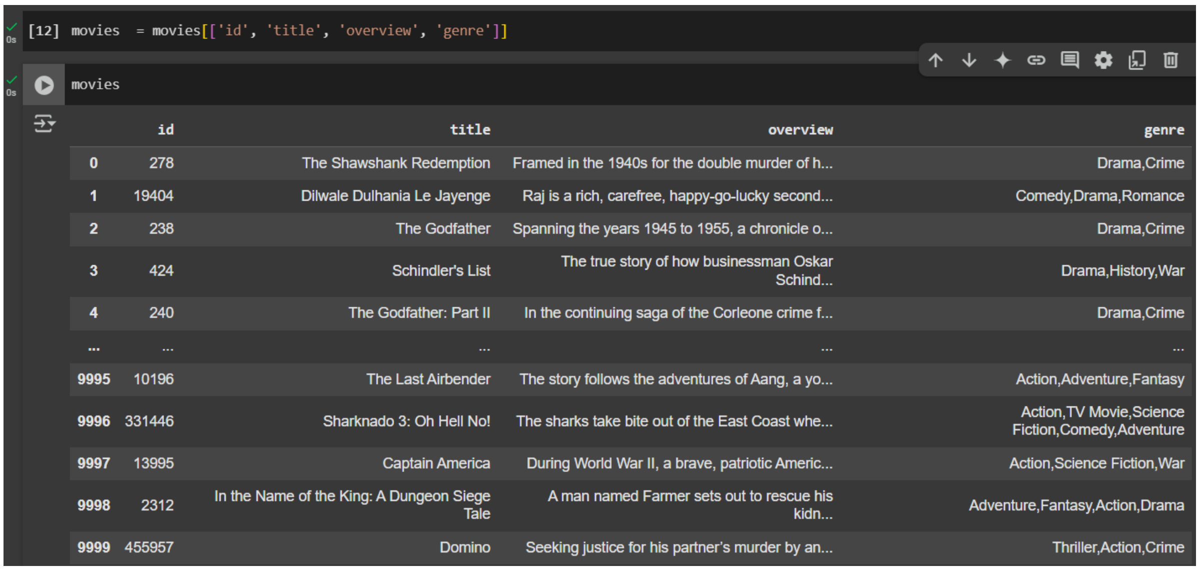
dtype: int64

Figure 7 - Checking for Null Values



```
[11] movies.columns

Index(['id', 'title', 'genre', 'original_language', 'overview', 'popularity',
       'release_date', 'vote_average', 'vote_count'],
      dtype='object')
```

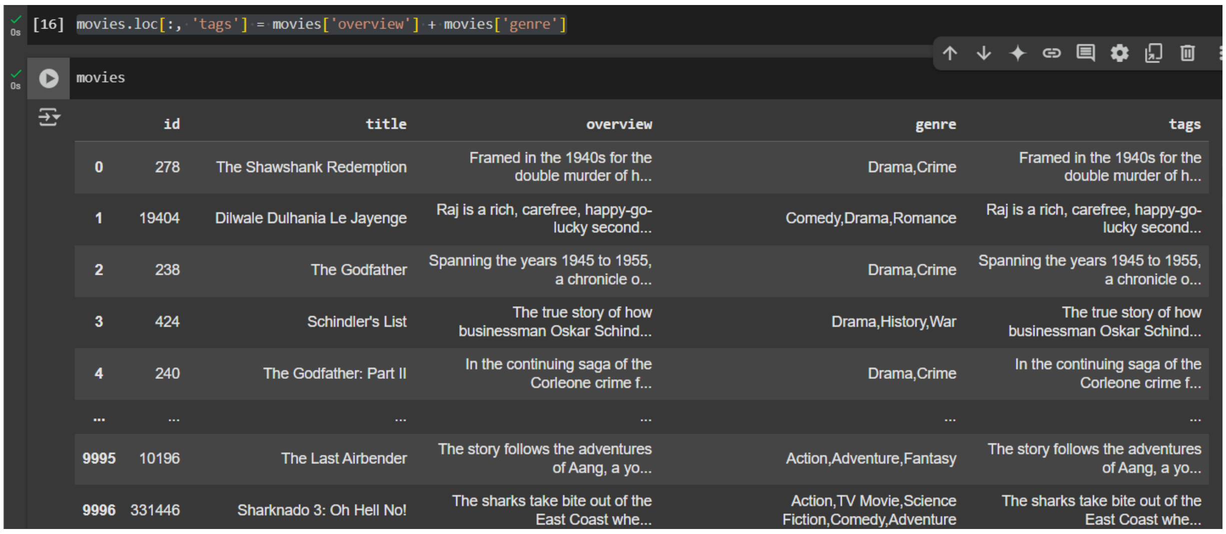
Figure 8 - Retrieving the Column Names



```
[12] movies  = movies[['id', 'title', 'overview', 'genre']]

movies
```

	id	title	overview	genre
0	278	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	Drama,Crime
1	19404	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...	Comedy,Drama,Romance
2	238	The Godfather	Spanning the years 1945 to 1955, a chronicle o...	Drama,Crime
3	424	Schindler's List	The true story of how businessman Oskar Schind...	Drama,History,War
4	240	The Godfather: Part II	In the continuing saga of the Corleone crime f...	Drama,Crime
...	...	...	...	...
9995	10196	The Last Airbender	The story follows the adventures of Aang, a yo...	Action,Adventure,Fantasy
9996	331446	Sharknado 3: Oh Hell No!	The sharks take bite out of the East Coast whe...	Action,TV Movie,Science Fiction,Comedy,Adventure
9997	13995	Captain America	During World War II, a brave, patriotic Americ...	Action,Science Fiction,War
9998	2312	In the Name of the King: A Dungeon Siege Tale	A man named Farmer sets out to rescue his kidn...	Adventure,Fantasy,Action,Drama
9999	455957	Domino	Seeking justice for his partner's murder by an...	Thriller,Action,Crime

Figure 9 - Selecting Specific Columns from the DataFrame

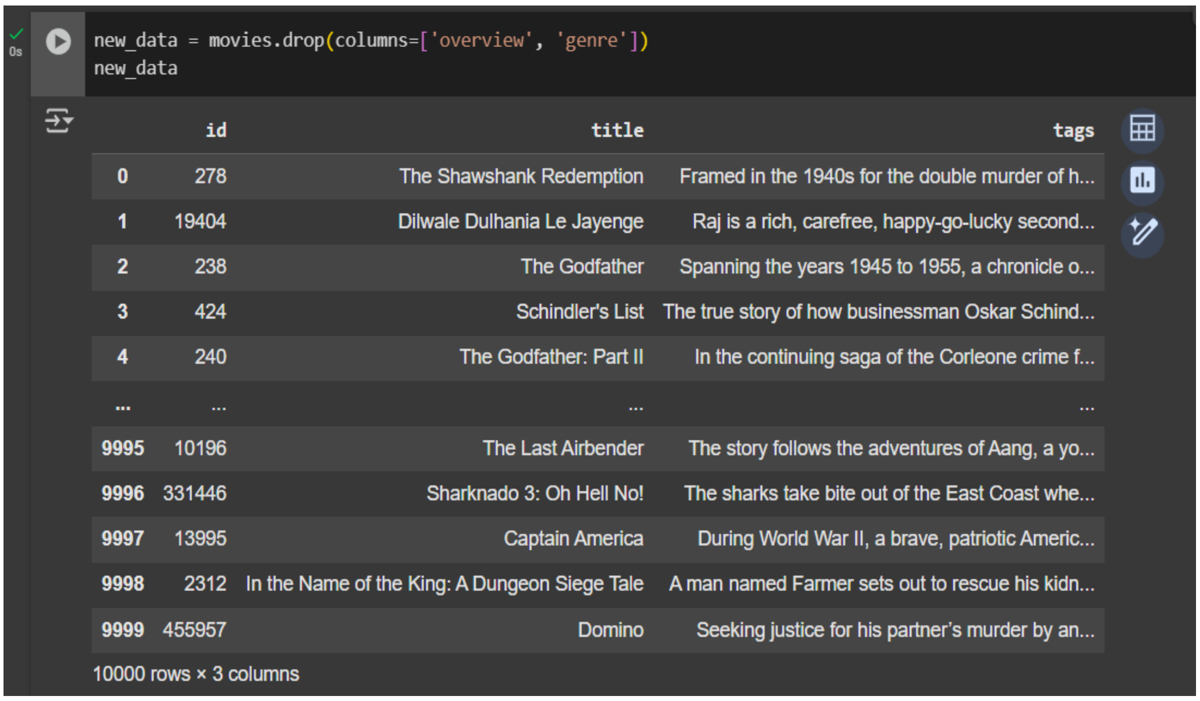


```
[16] movies.loc[:, 'tags'] = movies['overview'] + movies['genre']
```

```
movies
```

```
        id                        title                         overview                          genre                             tags
0      278     The Shawshank Redemption       Framed in the 1940s for the                    Drama,Crime       Framed in the 1940s for the
                                                   double murder of h...                                          double murder of h...
1    19404    Dilwale Dulhania Le Jayenge     Raj is a rich, carefree, happy-go-        Comedy,Drama,Romance   Raj is a rich, carefree, happy-go-
                                                         lucky second...                                                lucky second...
2      238                  The Godfather    Spanning the years 1945 to 1955,                   Drama,Crime     Spanning the years 1945 to 1955,
                                                         a chronicle o...                                              a chronicle o...
3      424               Schindler's List            The true story of how              Drama,History,War            The true story of how
                                             businessman Oskar Schind...                                   businessman Oskar Schind...
4      240          The Godfather: Part II     In the continuing saga of the                   Drama,Crime      In the continuing saga of the
                                                        Corleone crime f...                                           Corleone crime f...
...    ...                            ...                              ...                            ...                              ...
9995 10196              The Last Airbender  The story follows the adventures     Action,Adventure,Fantasy   The story follows the adventures
                                                           of Aang, a yo...                                              of Aang, a yo...
9996 331446       Sharknado 3: Oh Hell No!    The sharks take bite out of the       Action,TV Movie,Science    The sharks take bite out of the
                                                         East Coast whe...     Fiction,Comedy,Adventure               East Coast whe...
```

Figure 10 - Concatenating the Values of the Overview and Genre Columns



```
new_data = movies.drop(columns=['overview', 'genre'])
new_data
```

```
        id                                   title                                            tags
0      278                The Shawshank Redemption     Framed in the 1940s for the double murder of h...
1    19404             Dilwale Dulhania Le Jayenge         Raj is a rich, carefree, happy-go-lucky second...
2      238                           The Godfather     Spanning the years 1945 to 1955, a chronicle o...
3      424                        Schindler's List  The true story of how businessman Oskar Schind...
4      240                   The Godfather: Part II     In the continuing saga of the Corleone crime f...
...    ...                                     ...                                               ...
9995 10196                      The Last Airbender     The story follows the adventures of Aang, a yo...
9996 331446               Sharknado 3: Oh Hell No!     The sharks take bite out of the East Coast whe...
9997 13995                         Captain America     During World War II, a brave, patriotic Americ...
9998  2312  In the Name of the King: A Dungeon Siege Tale  A man named Farmer sets out to rescue his kidn...
9999 455957                                  Domino     Seeking justice for his partner's murder by an...

10000 rows × 3 columns
```

Figure 11 - Dropping the Overview and Genre Columns

11 CHAPTER 04: MODEL SELECTION, TRAINING AND EVALUATION

Content Based Filtering:

In the development of the movie recommendation system, Content-Based Filtering was utilized to recommend movies based on their content, particularly focusing on attributes such as genre and overview. The system employs cosine similarity between movie descriptions to identify and suggest similar movies to the user. Listed below are some key steps that were followed to create the recommendation system.

- **Data Preprocessing:** Textual data from the overview and genre columns were combined into a single 'tags' column for content-based filtering. Text data was preprocessed by converting it to lowercase, removing punctuation, stopwords, and performing lemmatization to standardize the words.

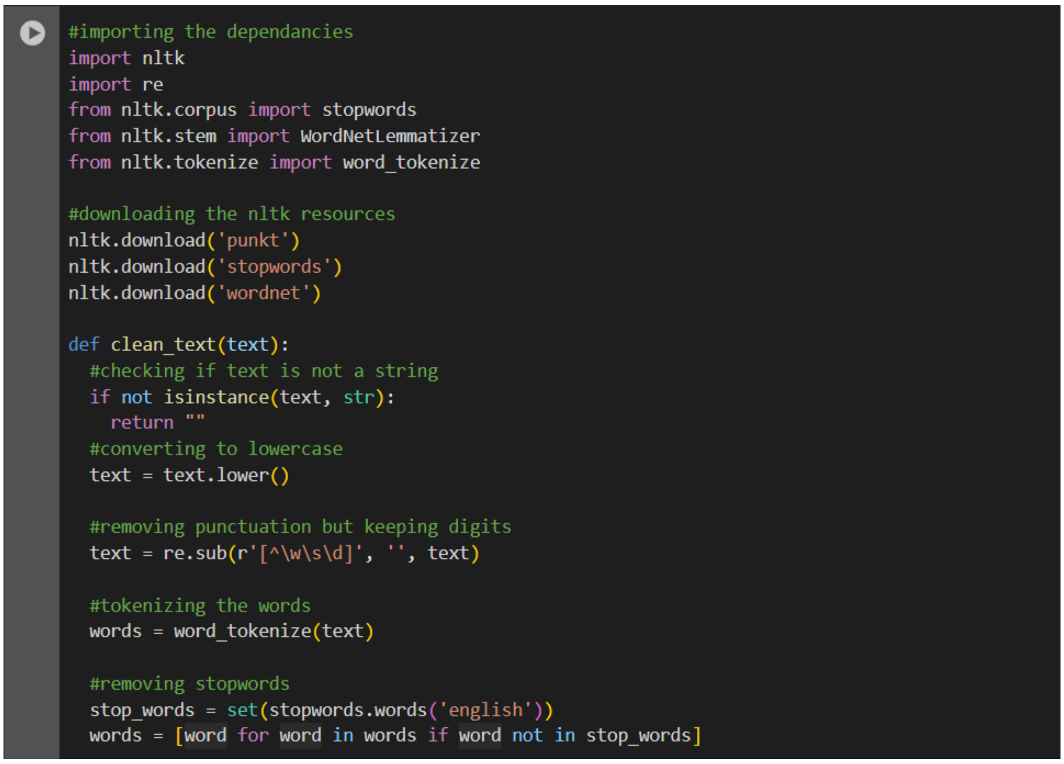
- **Vectorization:** The CountVectorizer was applied to the cleaned tags_clean column to convert the textual data into a matrix of token counts. This transformation enabled the system to represent each movie as a vector of word occurrences.

- **Cosine Similarity Calculation:** To measure the similarity between movies, the cosine similarity technique was used. It calculates the cosine of the angle between two vectors in the multidimensional space, where smaller angles which means larger cosine values indicate higher similarity.

- **Recommendation Function:** A recommendation function was defined to suggest movies similar to a given movie. The function works by finding the index of the movie based on its title, computing its cosine similarity with all other movies, sorting the movies based on similarity, and then returning the top 05 most similar movies.

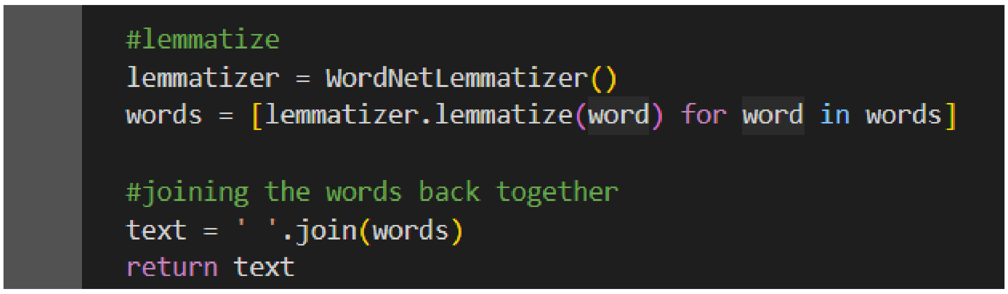
- **Model Serialization:** After training the model, the new_data DataFrame and the similarity matrix were serialized using pickle to save the model and similarity data for later use.

- **Model Deployment:** movies_list.pkl file containing movie data and the similarity.pkl file containing the similarity matrix were loaded using pickle to make real-time recommendations.



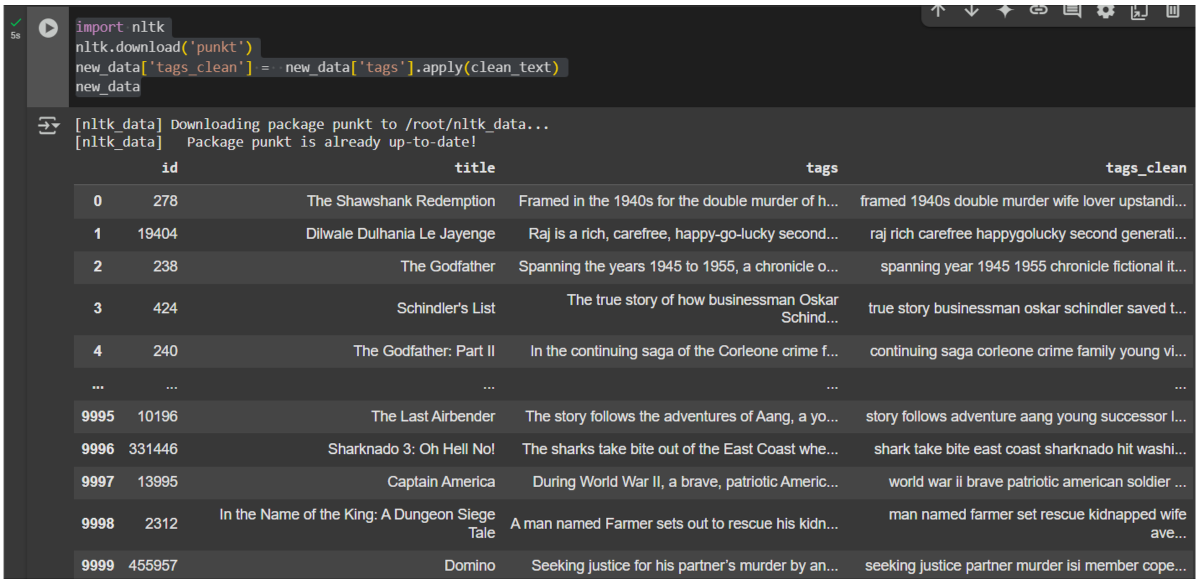
```python
#importing the dependancies
import nltk
import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

#downloading the nltk resources
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

def clean_text(text):
  #checking if text is not a string
  if not isinstance(text, str):
    return ""
  #converting to lowercase
  text = text.lower()

  #removing punctuation but keeping digits
  text = re.sub(r'[^\w\s\d]', '', text)

  #tokenizing the words
  words = word_tokenize(text)

  #removing stopwords
  stop_words = set(stopwords.words('english'))
  words = [word for word in words if word not in stop_words]
```

Figure 12 - Cleaning Text by Lowercasing, Removing Punctuation and Stopwords,



```python
  #lemmatize
  lemmatizer = WordNetLemmatizer()
  words = [lemmatizer.lemmatize(word) for word in words]

  #joining the words back together
  text = ' '.join(words)
  return text
```

Figure 13 - Tokenizing, Lemmatizing, and Rejoining words for Analysis.

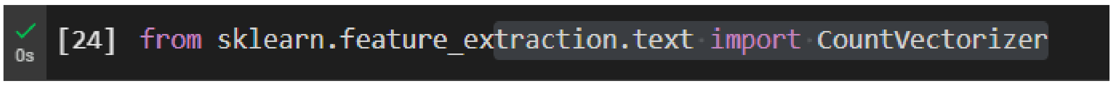


```python
import nltk
nltk.download('punkt')
new_data['tags_clean'] = new_data['tags'].apply(clean_text)
new_data
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

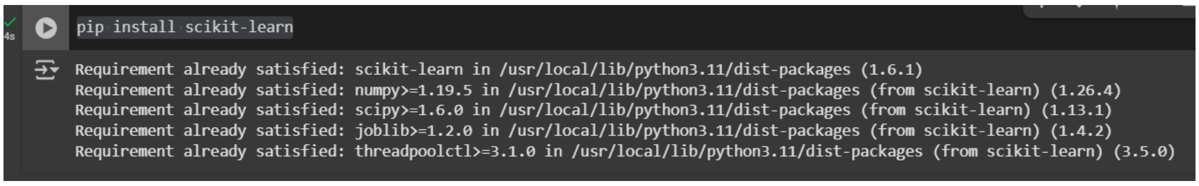
	id	title	tags	tags_clean
0	278	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	framed 1940s double murder wife lover upstandi...
1	19404	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...	raj rich carefree happygolucky second generati...
2	238	The Godfather	Spanning the years 1945 to 1955, a chronicle o...	spanning year 1945 1955 chronicle fictional it...
3	424	Schindler's List	The true story of how businessman Oskar Schind...	true story businessman oskar schindler saved t...
4	240	The Godfather: Part II	In the continuing saga of the Corleone crime f...	continuing saga corleone crime family young vi...
...	...	...	...	...
9995	10196	The Last Airbender	The story follows the adventures of Aang, a yo...	story follows adventure aang young successor l...
9996	331446	Sharknado 3: Oh Hell No!	The sharks take bite out of the East Coast whe...	shark take bite east coast sharknado hit washi...
9997	13995	Captain America	During World War II, a brave, patriotic Americ...	world war ii brave patriotic american soldier ...
9998	2312	In the Name of the King: A Dungeon Siege Tale	A man named Farmer sets out to rescue his kidn...	man named farmer set rescue kidnapped wife ave...
9999	455957	Domino	Seeking justice for his partner's murder by an...	seeking justice partner murder isi member cope...

Figure 14 - Tokenizing, Cleaning, and Updating the 'tags' Column in new_data.



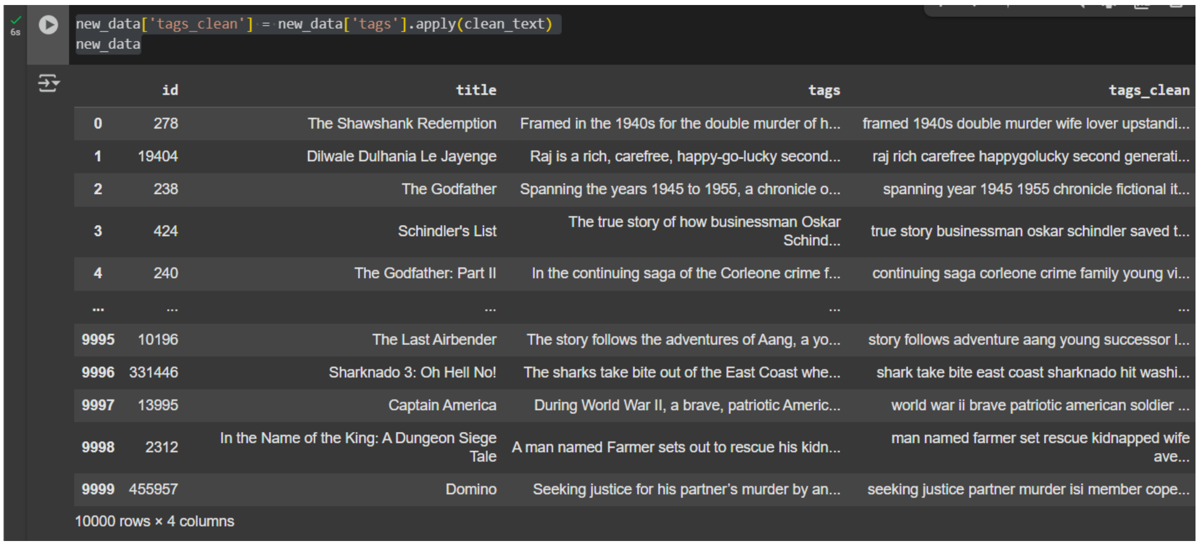
```
[24] from sklearn.feature_extraction.text import CountVectorizer
```

Figure 15 - Importing CountVectorizer from sklearn.feature_extraction.text to Convert Text Data into a Bag-of-Words (BoW) Representation.



```
pip install scikit-learn

Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.5.0)
```

Figure 16 - Installing scikit-learn for Text Feature Extraction

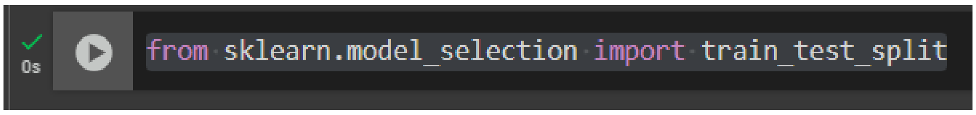


```
new_data['tags_clean'] = new_data['tags'].apply(clean_text)
new_data
```

	id	title	tags	tags_clean
0	278	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	framed 1940s double murder wife lover upstandi...
1	19404	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...	raj rich carefree happygolucky second generati...
2	238	The Godfather	Spanning the years 1945 to 1955, a chronicle o...	spanning year 1945 1955 chronicle fictional it...
3	424	Schindler's List	The true story of how businessman Oskar Schind...	true story businessman oskar schindler saved t...
4	240	The Godfather: Part II	In the continuing saga of the Corleone crime f...	continuing saga corleone crime family young vi...
...	...	...	...	...
9995	10196	The Last Airbender	The story follows the adventures of Aang, a yo...	story follows adventure aang young successor l...
9996	331446	Sharknado 3: Oh Hell No!	The sharks take bite out of the East Coast whe...	shark take bite east coast sharknado hit washi...
9997	13995	Captain America	During World War II, a brave, patriotic Americ...	world war ii brave patriotic american soldier ...
9998	2312	In the Name of the King: A Dungeon Siege Tale	A man named Farmer sets out to rescue his kidn...	man named farmer set rescue kidnapped wife ave...
9999	455957	Domino	Seeking justice for his partner's murder by an...	seeking justice partner murder isi member cope...

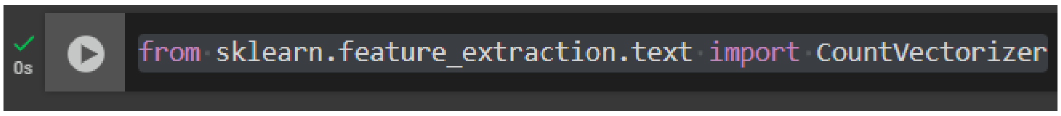
10000 rows × 4 columns

Figure 17 - Cleaning the text in the 'tags' column using clean_text() and storing the result in the 'tags_clean' column of new_data.



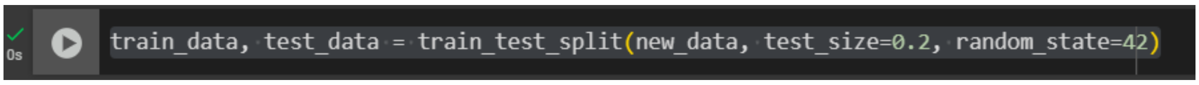
```
from sklearn.model_selection import train_test_split
```

Figure 18 - Importing train_test_split from sklearn.model_selection to split data intotraining and testing sets for model evaluation.



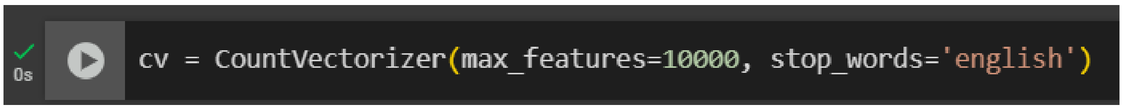
```
from sklearn.feature_extraction.text import CountVectorizer
```

Figure 19 - Importing CountVectorizer to convert text into a bag-of-words matrix.



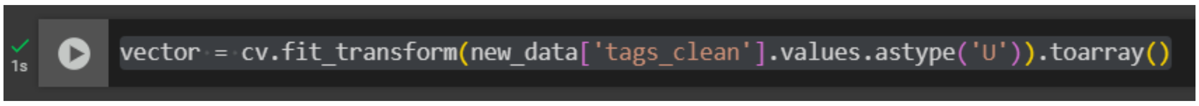
```
train_data, test_data = train_test_split(new_data, test_size=0.2, random_state=42)
```

Figure 20 - Spliting new_data into training and testing sets, with 80% for training and 20% for testing, using a random seed of 42.



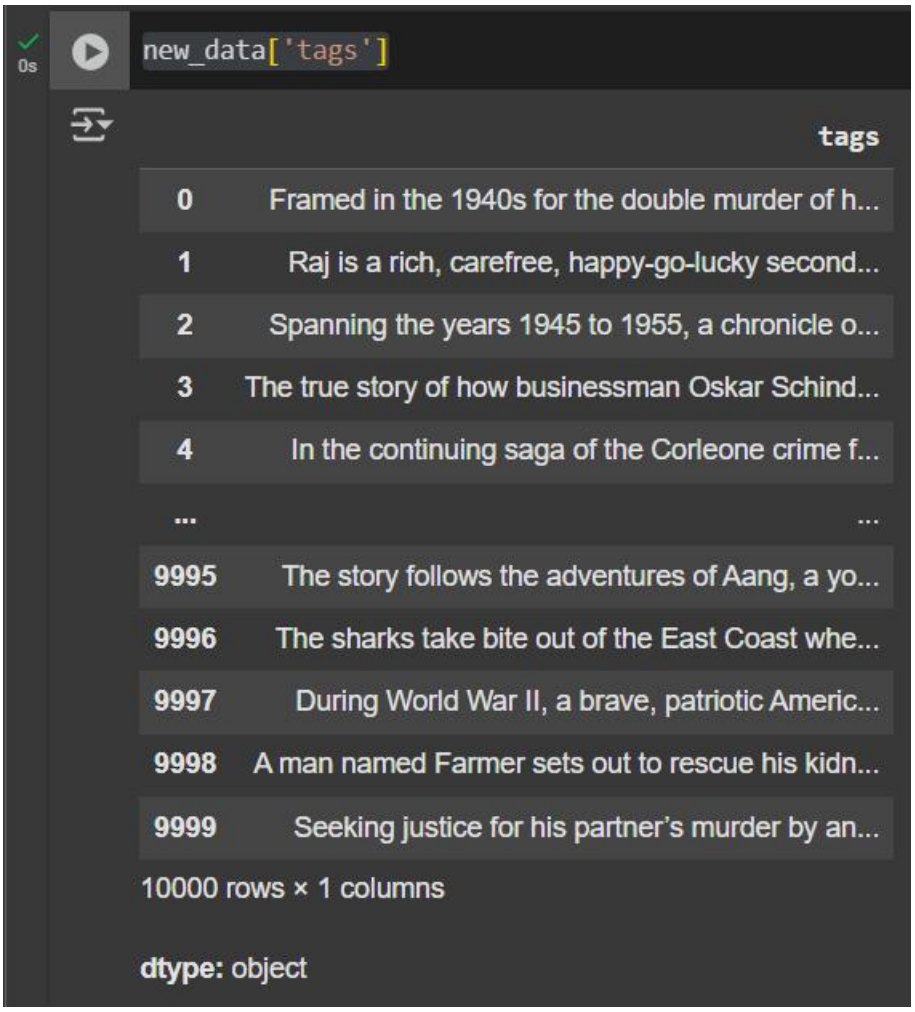
```
cv = CountVectorizer(max_features=10000, stop_words='english')
```

Figure 21 - Initializing CountVectorizer with a maximum of 10,000 features and removes English stopwords during the text vectorization process.



```
vector = cv.fit_transform(new_data['tags_clean'].values.astype('U')).toarray()
```

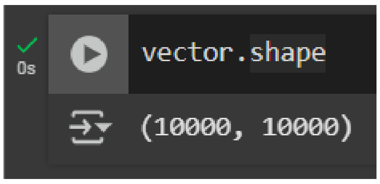
Figure 22 - Vectorizing the 'tags_clean' column into a numerical array of token counts.



```
new_data['tags']
```

```
                                                   tags
0       Framed in the 1940s for the double murder of h...
1         Raj is a rich, carefree, happy-go-lucky second...
2       Spanning the years 1945 to 1955, a chronicle o...
3     The true story of how businessman Oskar Schind...
4          In the continuing saga of the Corleone crime f...
...                                                     ...
9995     The story follows the adventures of Aang, a yo...
9996    The sharks take bite out of the East Coast whe...
9997       During World War II, a brave, patriotic Americ...
9998  A man named Farmer sets out to rescue his kidn...
9999      Seeking justice for his partner's murder by an...
10000 rows × 1 columns

dtype: object
```

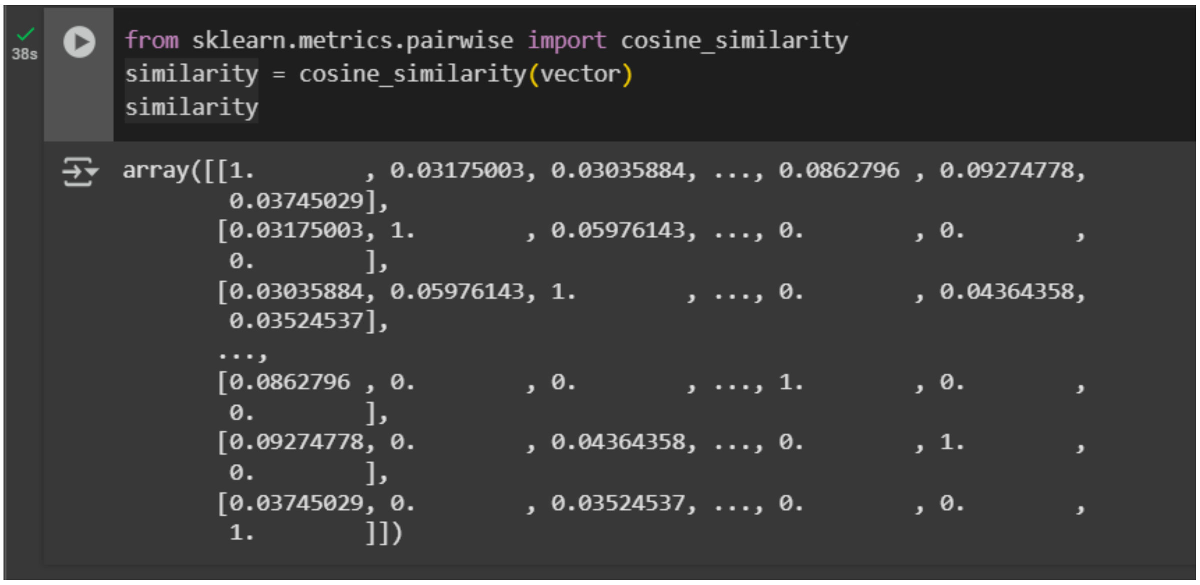
Figure 23 - Referencing the 'tags' column in the new_data dataframe, which contains the original text data before any cleaning or transformation.



```
vector.shape
```

```
(10000, 10000)
```

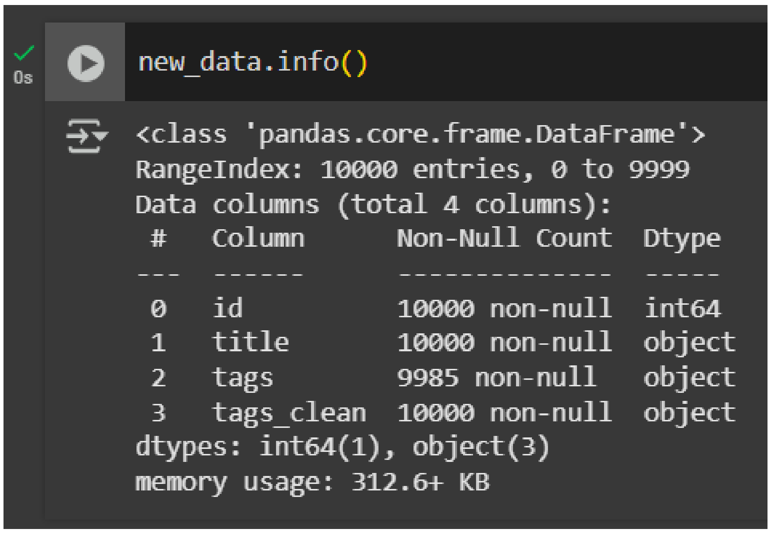
Figure 24 - Showing the number of rows (documents) and columns (features/tokens) in the tokenized representation.



```
from sklearn.metrics.pairwise import cosine_similarity
similarity = cosine_similarity(vector)
similarity
```

```
array([[1.        , 0.03175003, 0.03035884, ..., 0.0862796 , 0.09274778,
        0.03745029],
       [0.03175003, 1.        , 0.05976143, ..., 0.        , 0.        ,
        0.        ],
       [0.03035884, 0.05976143, 1.        , ..., 0.        , 0.04364358,
        0.03524537],
       ...,
       [0.0862796 , 0.        , 0.        , ..., 1.        , 0.        ,
        0.        ],
       [0.09274778, 0.        , 0.04364358, ..., 0.        , 1.        ,
        0.        ],
       [0.03745029, 0.        , 0.03524537, ..., 0.        , 0.        ,
        1.        ]])
```

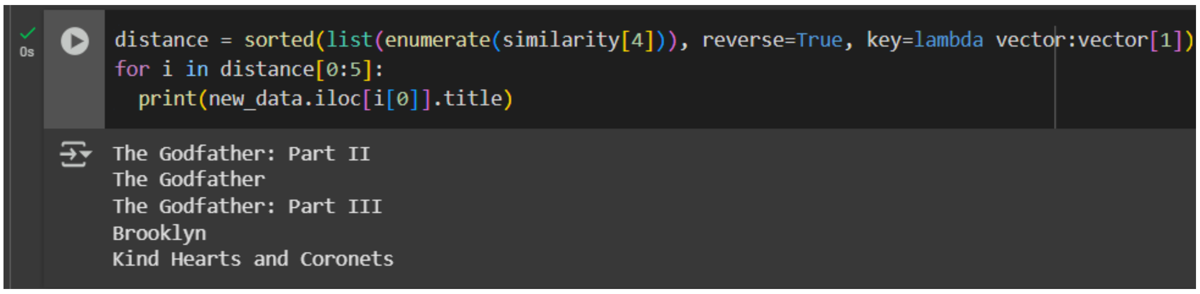
Figure 25 - Calculating the cosine similarity between all vectors in the dataset.



```
new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   id          10000 non-null  int64 
 1   title       10000 non-null  object
 2   tags        9985 non-null   object
 3   tags_clean  10000 non-null  object
dtypes: int64(1), object(3)
memory usage: 312.6+ KB
```

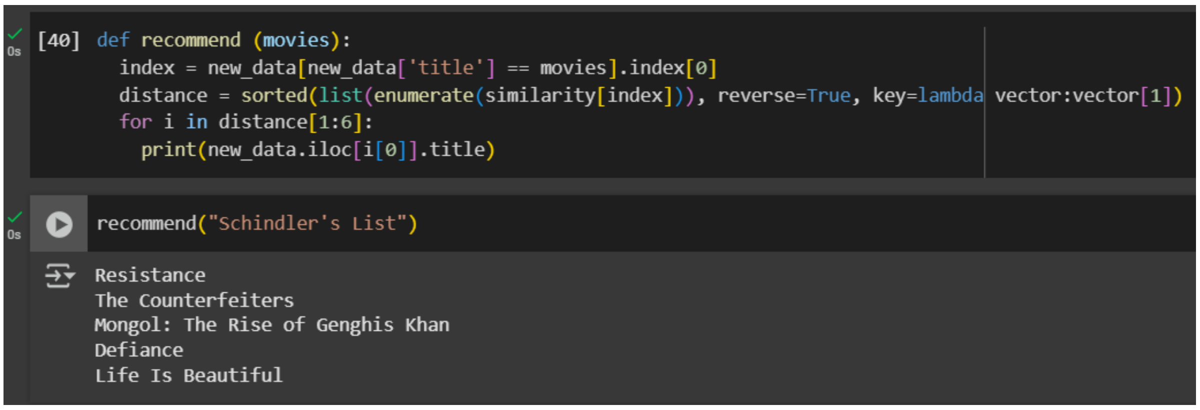
Figure 26 - Displaying the summary information of the new_data dataframe.



```
distance = sorted(list(enumerate(similarity[4])), reverse=True, key=lambda vector:vector[1])
for i in distance[0:5]:
  print(new_data.iloc[i[0]].title)
```

```
The Godfather: Part II
The Godfather
The Godfather: Part III
Brooklyn
Kind Hearts and Coronets
```

Figure 27 - Finding and printing the titles of the top 05 most similar movies to the one at index 04.

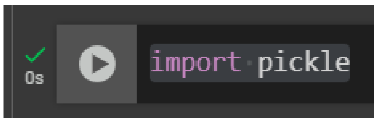


```
[40] def recommend (movies):
       index = new_data[new_data['title'] == movies].index[0]
       distance = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda vector:vector[1])
       for i in distance[1:6]:
         print(new_data.iloc[i[0]].title)
```

```
recommend("Schindler's List")
```

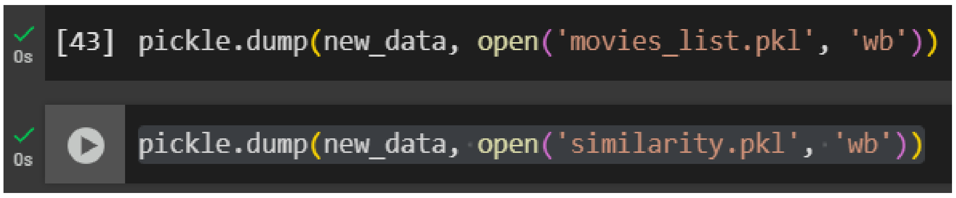
```
Resistance
The Counterfeiters
Mongol: The Rise of Genghis Khan
Defiance
Life Is Beautiful
```

Figure 28 - Defining a recommend function to find and print the top 05 similar movies to a given title.



```
import pickle
```

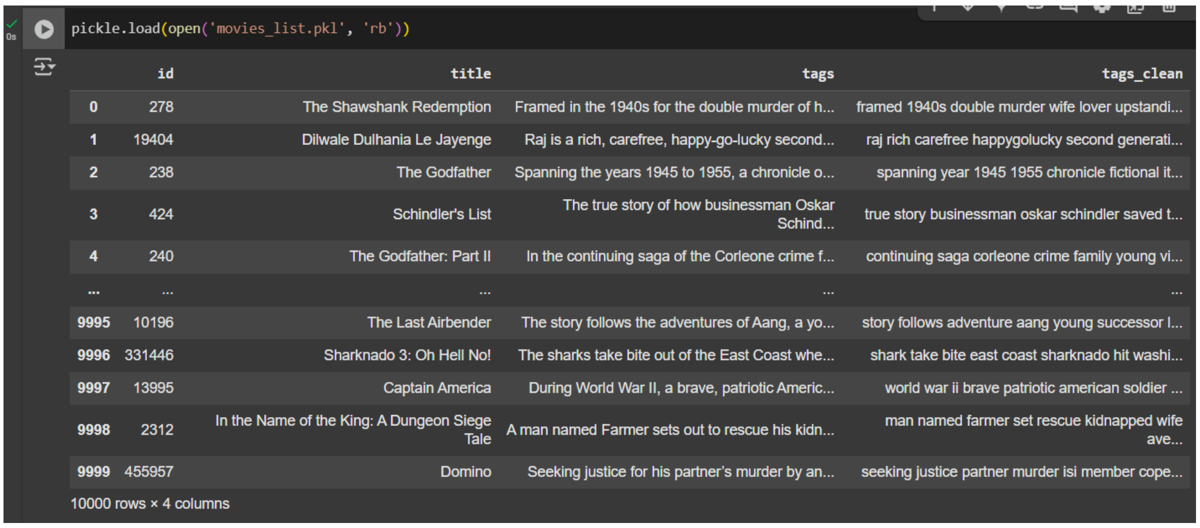
Figure 29 - Importing pickle for object serialization and deserialization.



```
[43] pickle.dump(new_data, open('movies_list.pkl', 'wb'))
```

```
pickle.dump(new_data, open('similarity.pkl', 'wb'))
```

Figure 30 - Serializing and saving the new_data dataframe and the similarity matrix as movies_list.pkl and similarity.pkl files, respectively.



```
pickle.load(open('movies_list.pkl', 'rb'))
```

	id	title	tags	tags_clean
0	278	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	framed 1940s double murder wife lover upstandi...
1	19404	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...	raj rich carefree happygolucky second generati...
2	238	The Godfather	Spanning the years 1945 to 1955, a chronicle o...	spanning year 1945 1955 chronicle fictional it...
3	424	Schindler's List	The true story of how businessman Oskar Schind...	true story businessman oskar schindler saved t...
4	240	The Godfather: Part II	In the continuing saga of the Corleone crime f...	continuing saga corleone crime family young vi...
...	...	...	...	...
9995	10196	The Last Airbender	The story follows the adventures of Aang, a yo...	story follows adventure aang young successor l...
9996	331446	Sharknado 3: Oh Hell No!	The sharks take bite out of the East Coast whe...	shark take bite east coast sharknado hit washi...
9997	13995	Captain America	During World War II, a brave, patriotic Americ...	world war ii brave patriotic american soldier ...
9998	2312	In the Name of the King: A Dungeon Siege Tale	A man named Farmer sets out to rescue his kidn...	man named farmer set rescue kidnapped wife ave...
9999	455957	Domino	Seeking justice for his partner's murder by an...	seeking justice partner murder isi member cope...

10000 rows × 4 columns

Figure 31 - Loading the serialized movies_list.pkl file and deserializing it back into the new_data dataframe.

Item-Based Collaborative Filtering:

In the development of the movie recommendation system, Collaborative Filtering was utilized to recommend movies considering the user's behavior and compares it with other users in the database. It uses the history of all users to influence the recommendation algorithm. Unlike a content-based recommender system, a collaborative filtering recommender relies on multiple users' interactions with items to generate recommendations.

In this implementation, when the user searches for a movie, we recommend the top 10 similar movies. I utilized an item-based collaborative filtering algorithm for our purpose.

Item-based Collaborative Filtering focuses on finding similar movies instead of similar users to recommend based on their past preferences. It identifies pairs of movies rated by the same users, measures their similarity across all users who rated both, and then recommends similar films based on the similarity scores.

Listed below are some key steps that were followed to create the collaborative filtering recommendation system.

1. **Preparing the User-Item Interaction Matrix:** Collaborative filtering relies on past user interactions rather than movie content. I used a dataset containing user ratings and transformed it into a user-item matrix, where:

   - Rows represent movies.
   - Columns represent UserId.
   - Values represent ratings given by users to movies.

   Since users rate only a subset of movies, the dataset is highly sparse, such that many missing values exist. To handle this, I filled missing values with zeros, ensuring a structured matrix.

2. **Optimizing the Data for Faster Computation:** To reduce memory usage and improve performance, we convert the user-item matrix into a Compressed Sparse Row (CSR) matrix using the scipy library. This format efficiently stores only nonzero values, making similarity computations

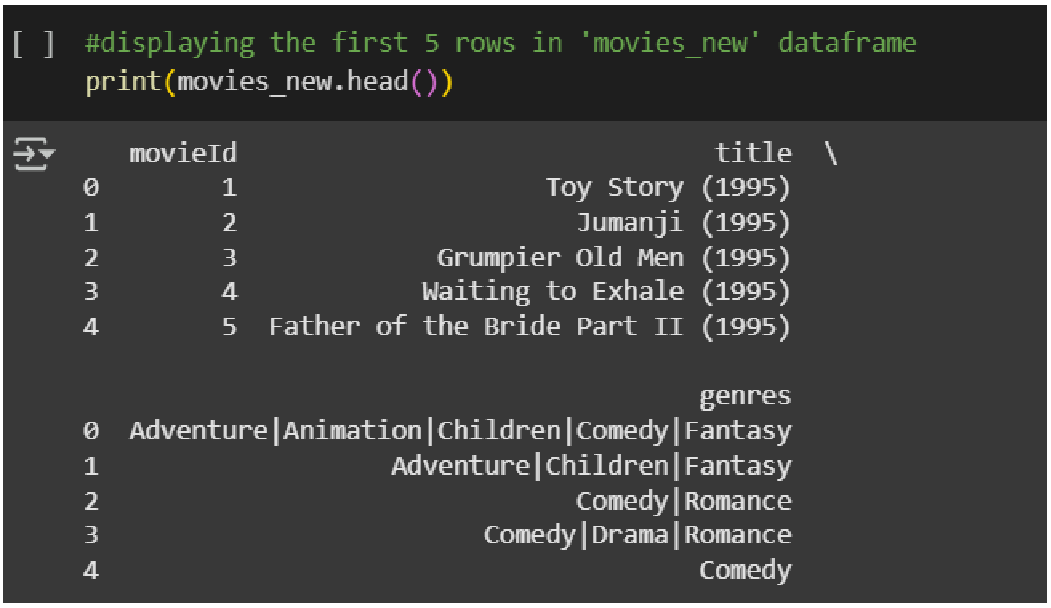
faster.

3. Finding Similar Movies Using Nearest Neighbors (KNN): I applied the K-Nearest Neighbors (KNN) algorithm with cosine similarity as the distance metric. KNN clusters movies based on user rating patterns, identifying movies that are frequently rated together.

4. Making Movie Recommendations To generate recommendations, I:

- Located the selected movie in the similarity matrix.
- Retrieved movies with the highest similarity scores.
- Ranked the similar movies and return the top recommendations.

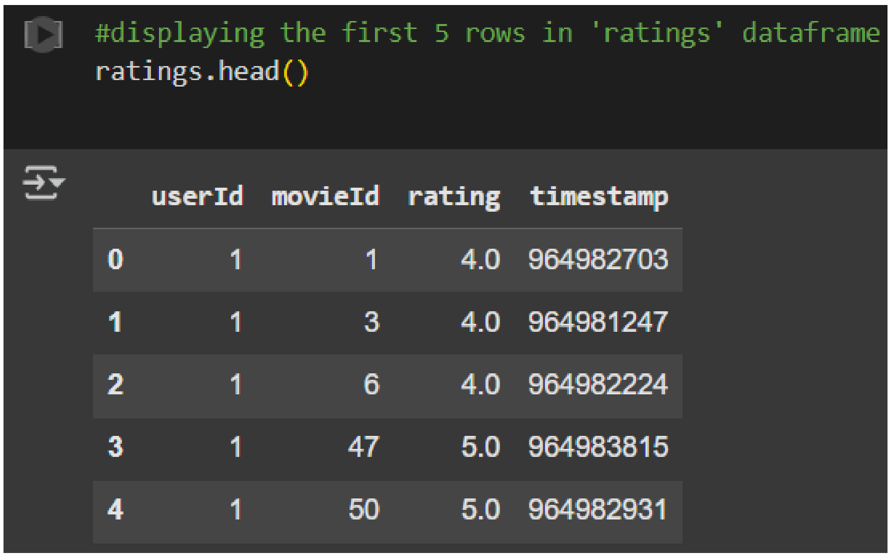
Finally, we test the system by querying different movie titles and validating whether the suggested movies align with expected genres and themes.



```
#displaying the first 5 rows in 'movies_new' dataframe
print(movies_new.head())
```

```
   movieId                               title  \
0        1                    Toy Story (1995)
1        2                      Jumanji (1995)
2        3             Grumpier Old Men (1995)
3        4            Waiting to Exhale (1995)
4        5  Father of the Bride Part II (1995)

                                        genres
0  Adventure|Animation|Children|Comedy|Fantasy
1                   Adventure|Children|Fantasy
2                               Comedy|Romance
3                         Comedy|Drama|Romance
4                                       Comedy
```

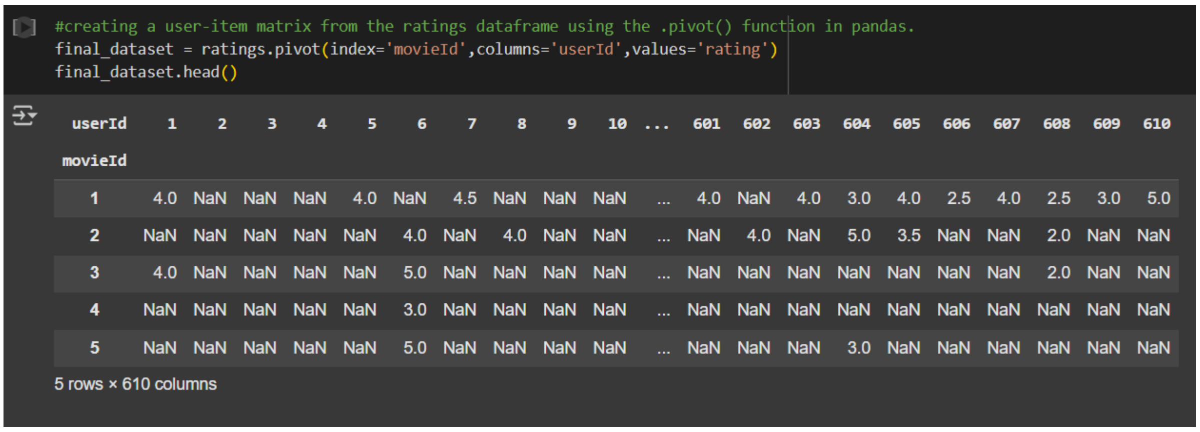
Figure 32 - Displaying the first 05 rows in movies_new dataframe



```
#displaying the first 5 rows in 'ratings' dataframe
ratings.head()
```

```
   userId  movieId  rating  timestamp
0       1        1     4.0  964982703
1       1        3     4.0  964981247
2       1        6     4.0  964982224
3       1       47     5.0  964983815
4       1       50     5.0  964982931
```

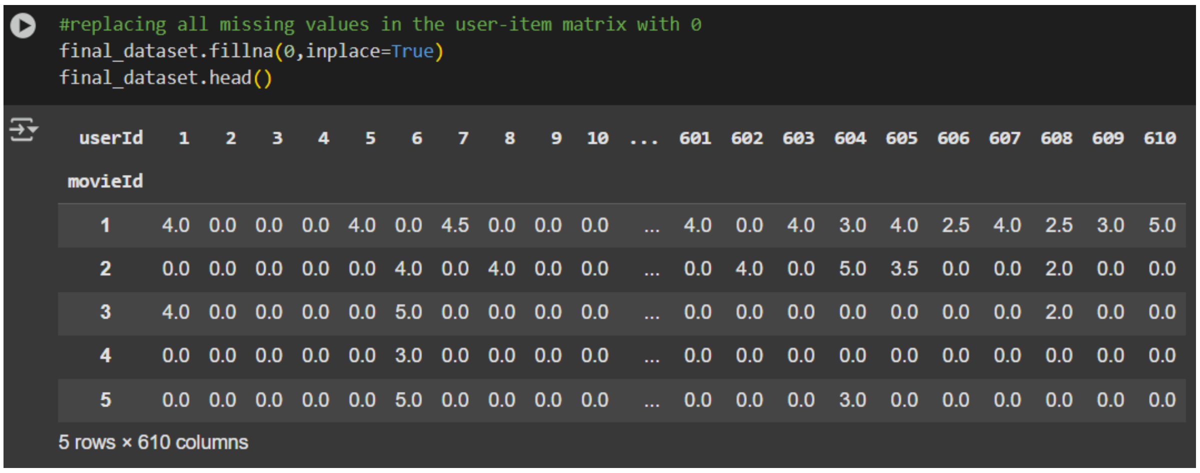
Figure 33 - Displaying the first 05 rows in ratings dataframe



```
#creating a user-item matrix from the ratings dataframe using the .pivot() function in pandas.
final_dataset = ratings.pivot(index='movieId',columns='userId',values='rating')
final_dataset.head()
```

```
userId    1    2    3    4    5    6    7    8    9   10  ...  601  602  603  604  605  606  607  608  609  610
movieId
      1  4.0  NaN  NaN  NaN  4.0  NaN  4.5  NaN  NaN  NaN  ...  4.0  NaN  4.0  3.0  4.0  2.5  4.0  2.5  3.0  5.0
      2  NaN  NaN  NaN  NaN  NaN  4.0  NaN  4.0  NaN  NaN  ...  NaN  4.0  NaN  5.0  3.5  NaN  NaN  2.0  NaN  NaN
      3  4.0  NaN  NaN  NaN  NaN  5.0  NaN  NaN  NaN  NaN  ...  NaN  NaN  NaN  NaN  NaN  NaN  NaN  2.0  NaN  NaN
      4  NaN  NaN  NaN  NaN  NaN  3.0  NaN  NaN  NaN  NaN  ...  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
      5  NaN  NaN  NaN  NaN  NaN  5.0  NaN  NaN  NaN  NaN  ...  NaN  NaN  NaN  3.0  NaN  NaN  NaN  NaN  NaN  NaN
5 rows × 610 columns
```

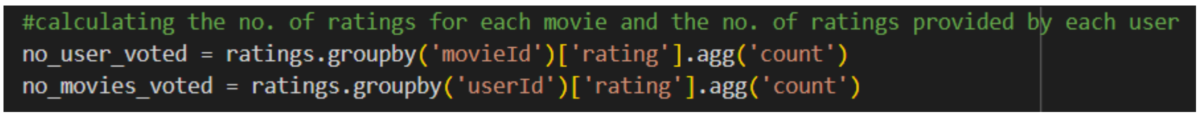
Figure 34 - Creating a user-item matrix from the ratings dataframe using the .pivot() function in pandas.



```
#replacing all missing values in the user-item matrix with 0
final_dataset.fillna(0,inplace=True)
final_dataset.head()
```

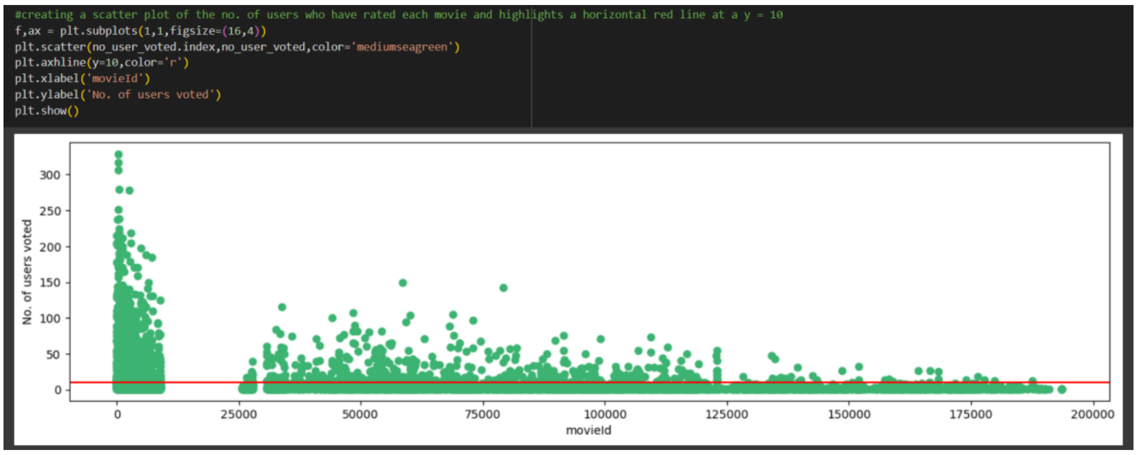
```
userId    1    2    3    4    5    6    7    8    9   10  ...  601  602  603  604  605  606  607  608  609  610
movieId
      1  4.0  0.0  0.0  0.0  4.0  0.0  4.5  0.0  0.0  0.0  ...  4.0  0.0  4.0  3.0  4.0  2.5  4.0  2.5  3.0  5.0
      2  0.0  0.0  0.0  0.0  0.0  4.0  0.0  4.0  0.0  0.0  ...  0.0  4.0  0.0  5.0  3.5  0.0  0.0  2.0  0.0  0.0
      3  4.0  0.0  0.0  0.0  0.0  5.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  2.0  0.0  0.0
      4  0.0  0.0  0.0  0.0  0.0  3.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
      5  0.0  0.0  0.0  0.0  0.0  5.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0  3.0  0.0  0.0  0.0  0.0  0.0  0.0
5 rows × 610 columns
```

Figure 35 - Replacing the missing values in the user-item matrix with 0



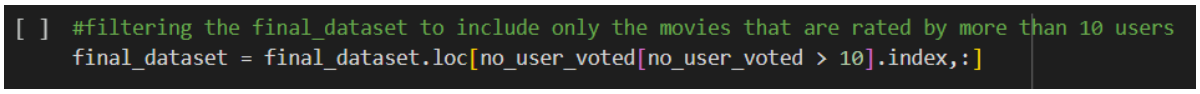
```
#calculating the no. of ratings for each movie and the no. of ratings provided by each user
no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
```

Figure 36 - Calculating the no. of ratings for each movie and the no. of ratings provided by each user.



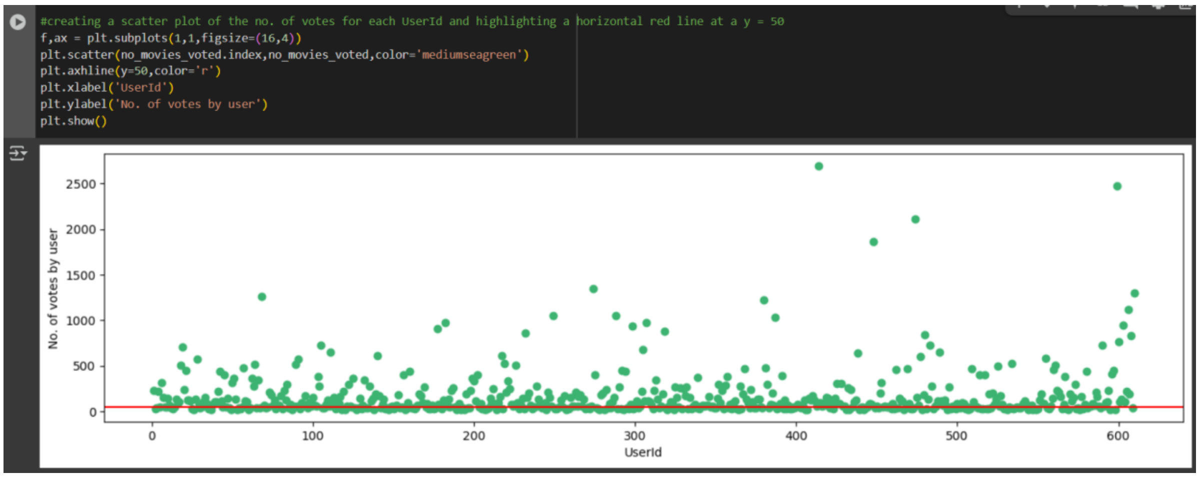
```
#creating a scatter plot of the no. of users who have rated each movie and highlights a horizontal red line at a y = 10
f,ax = plt.subplots(1,1,figsize=(16,4))
plt.scatter(no_user_voted.index,no_user_voted,color='mediumseagreen')
plt.axhline(y=10,color='r')
plt.xlabel('movieId')
plt.ylabel('No. of users voted')
plt.show()
```

Figure 37 - Creating a scatter plot with the no. of users who have rated for each movie



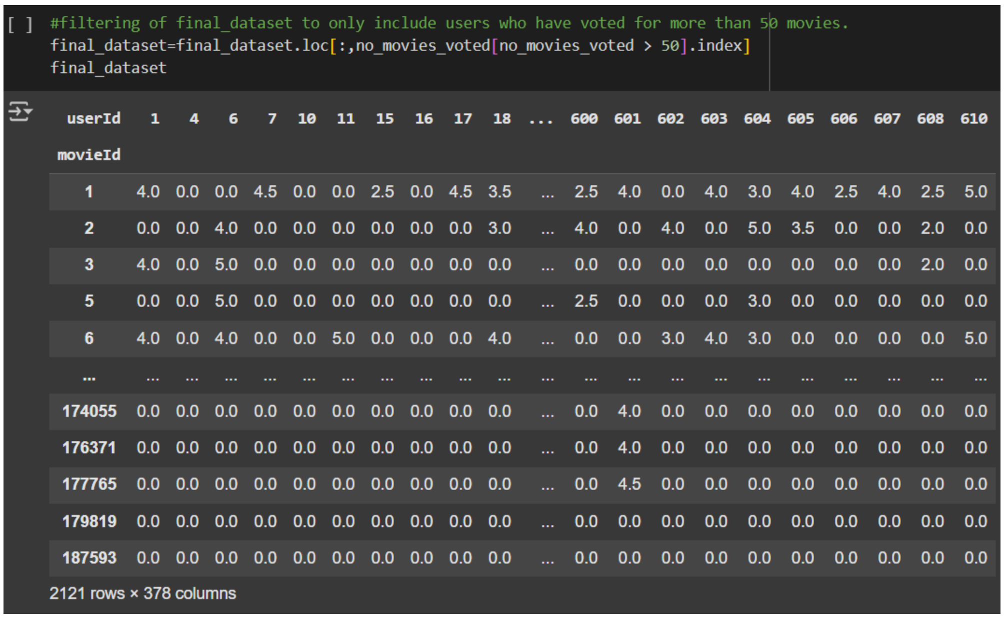
```
[ ]  #filtering the final_dataset to include only the movies that are rated by more than 10 users
     final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index,:]
```

Figure 38 - Filtering the final dataset to include only the movies rated by more than 10 users



```
#creating a scatter plot of the no. of votes for each UserId and highlighting a horizontal red line at a y = 50
f,ax = plt.subplots(1,1,figsize=(16,4))
plt.scatter(no_movies_voted.index,no_movies_voted,color='mediumseagreen')
plt.axhline(y=50,color='r')
plt.xlabel('UserId')
plt.ylabel('No. of votes by user')
plt.show()
```

Figure 39 - Creating a scatter plot of the no. of votes for each UserId and highlighting a horizontal red line at a y = 50

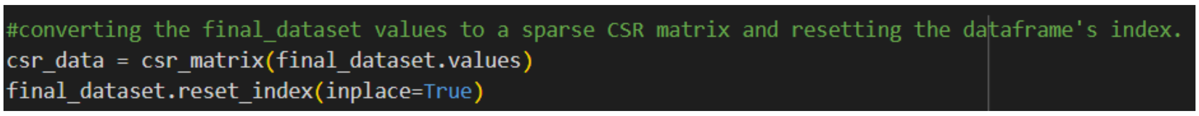


```
[ ]  #filtering of final_dataset to only include users who have voted for more than 50 movies.
     final_dataset=final_dataset.loc[:,no_movies_voted[no_movies_voted > 50].index]
     final_dataset
```

userId	1	4	6	7	10	11	15	16	17	18	...	600	601	602	603	604	605	606	607	608	610
movieId																					
1	4.0	0.0	0.0	4.5	0.0	0.0	2.5	0.0	4.5	3.5	...	2.5	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	5.0
2	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	...	4.0	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0
3	4.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0
5	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2.5	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0
6	4.0	0.0	4.0	0.0	0.0	5.0	0.0	0.0	0.0	4.0	...	0.0	0.0	3.0	4.0	3.0	0.0	0.0	0.0	0.0	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
174055	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
176371	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
177765	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
179819	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
187593	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

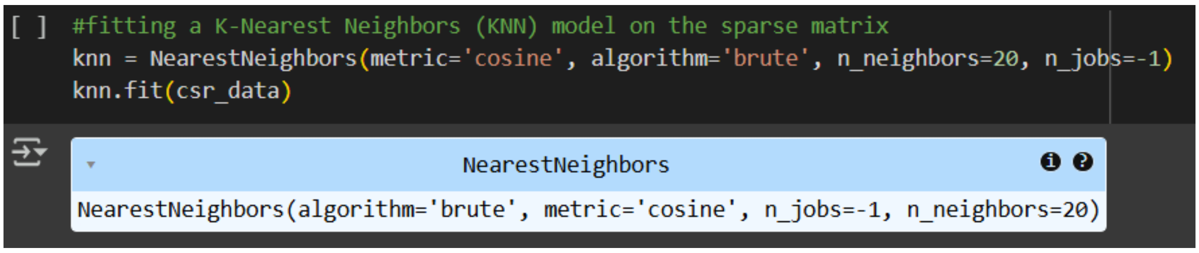
2121 rows × 378 columns

Figure 40 - Filtering the final_dataset to only include users who have voted for more than 50 movies.



```
#converting the final_dataset values to a sparse CSR matrix and resetting the dataframe's index.
csr_data = csr_matrix(final_dataset.values)
final_dataset.reset_index(inplace=True)
```

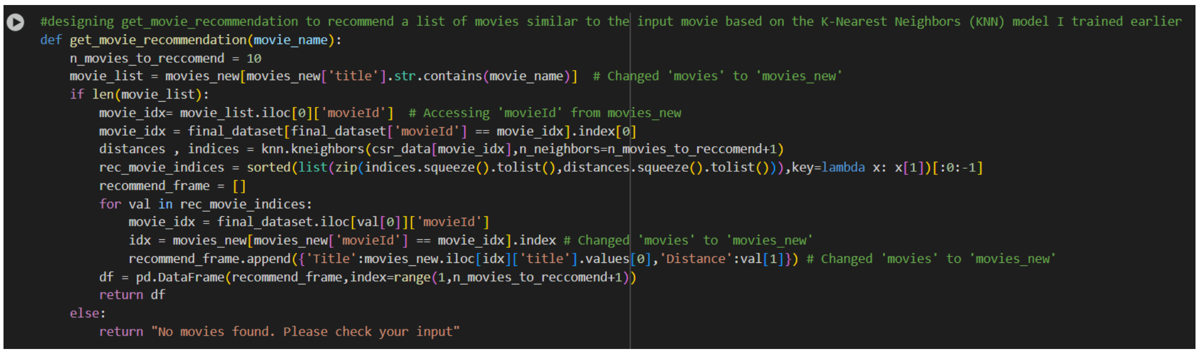
Figure 41 - Converting the final_dataset values to a sparse CSR matrix and resetting the dataframe's index.



```
[ ]  #fitting a K-Nearest Neighbors (KNN) model on the sparse matrix
     knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
     knn.fit(csr_data)
```

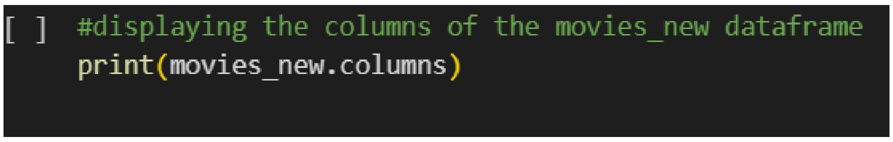
```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

Figure 42 - Fitting a K-Nearest Neighbors (KNN) model on the sparse matrix



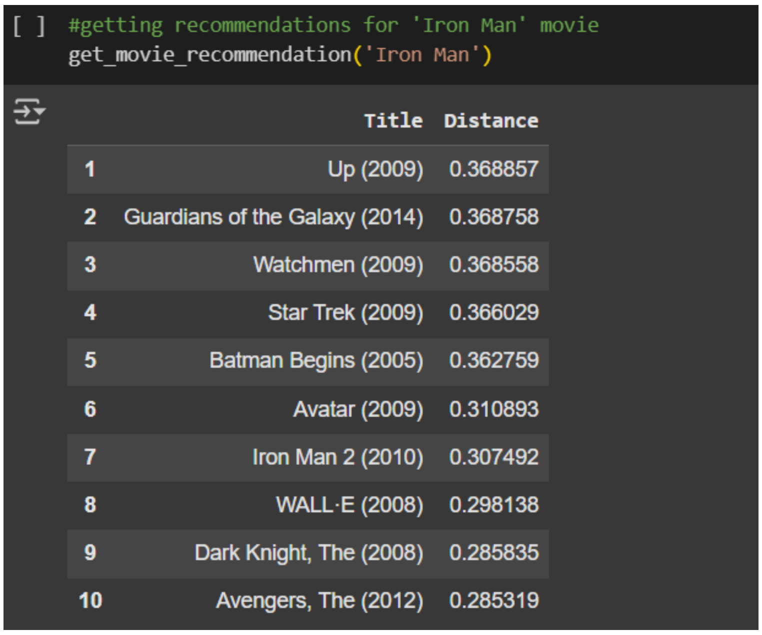
```
#designing get_movie_recommendation to recommend a list of movies similar to the input movie based on the K-Nearest Neighbors (KNN) model I trained earlier
def get_movie_recommendation(movie_name):
    n_movies_to_reccomend = 10
    movie_list = movies_new[movies_new['title'].str.contains(movie_name)]  # Changed 'movies' to 'movies_new'
    if len(movie_list):
        movie_idx= movie_list.iloc[0]['movieId']  # Accessing 'movieId' from movies_new
        movie_idx = final_dataset[final_dataset['movieId'] == movie_idx].index[0]
        distances , indices = knn.kneighbors(csr_data[movie_idx],n_neighbors=n_movies_to_reccomend+1)
        rec_movie_indices = sorted(list(zip(indices.squeeze().tolist(),distances.squeeze().tolist())),key=lambda x: x[1])[:0:-1]
        recommend_frame = []
        for val in rec_movie_indices:
            movie_idx = final_dataset.iloc[val[0]]['movieId']
            idx = movies_new[movies_new['movieId'] == movie_idx].index # Changed 'movies' to 'movies_new'
            recommend_frame.append({'Title':movies_new.iloc[idx]['title'].values[0],'Distance':val[1]}) # Changed 'movies' to 'movies_new'
        df = pd.DataFrame(recommend_frame,index=range(1,n_movies_to_reccomend+1))
        return df
    else:
        return "No movies found. Please check your input"
```

Figure 43 - Creating get_movie_recommendation to recommend a list of movies similar to the input movie based on the K-Nearest Neighbors (KNN) model trained earlier



```
[ ]  #displaying the columns of the movies_new dataframe
     print(movies_new.columns)
```

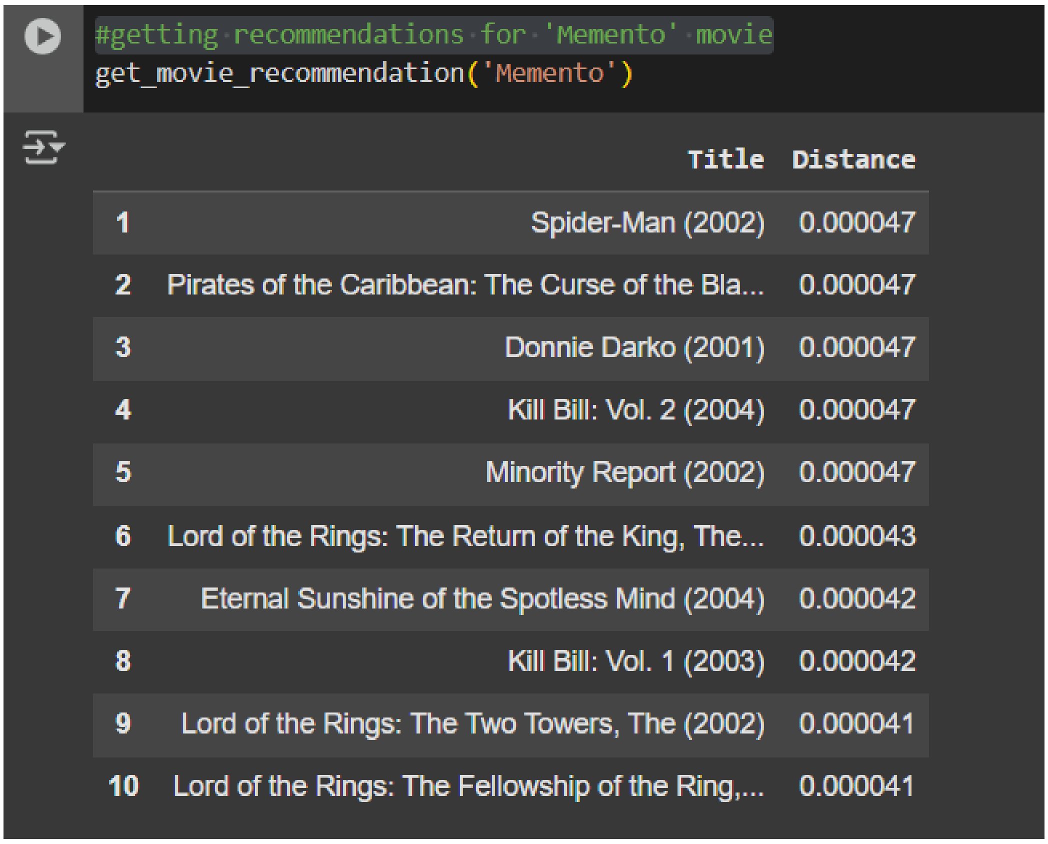
Figure 44 - Displaying the columns of the movies_new dataframe



```
[ ]  #getting recommendations for 'Iron Man' movie
     get_movie_recommendation('Iron Man')
```

	Title	Distance
1	Up (2009)	0.368857
2	Guardians of the Galaxy (2014)	0.368758
3	Watchmen (2009)	0.368558
4	Star Trek (2009)	0.366029
5	Batman Begins (2005)	0.362759
6	Avatar (2009)	0.310893
7	Iron Man 2 (2010)	0.307492
8	WALL·E (2008)	0.298138
9	Dark Knight, The (2008)	0.285835
10	Avengers, The (2012)	0.285319

Figure 45 -Getting the recommendations for the "Iron Man" movie



```
#getting recommendations for 'Memento' movie
get_movie_recommendation('Memento')
```

	Title	Distance
1	Spider-Man (2002)	0.000047
2	Pirates of the Caribbean: The Curse of the Bla...	0.000047
3	Donnie Darko (2001)	0.000047
4	Kill Bill: Vol. 2 (2004)	0.000047
5	Minority Report (2002)	0.000047
6	Lord of the Rings: The Return of the King, The...	0.000043
7	Eternal Sunshine of the Spotless Mind (2004)	0.000042
8	Kill Bill: Vol. 1 (2003)	0.000042
9	Lord of the Rings: The Two Towers, The (2002)	0.000041
10	Lord of the Rings: The Fellowship of the Ring,...	0.000041

Figure 46 - Getting the recommendations for the "Memento" movie

12 CHAPTER 05: MODEL DEPLOYMENT

To provide an interactive and user-friendly experience, Streamlit was used to develop the UI for the movie recommendation system. Streamlit is a powerful Python framework that enables the creation of web applications with minimal effort. The application was deployed locally using localhost, allowing users to input a movie title and receive personalized recommendations in real time. The user interface was designed to be simple yet effective, displaying the recommended movies dynamically based on content-based filtering and item-based collaborative filtering. The backend, integrated with the preprocessed movie data and similarity matrix, ensures seamless interaction between user input and the recommendation engine. This approach allows for quick testing, iteration, and deployment while maintaining an intuitive interface for users.

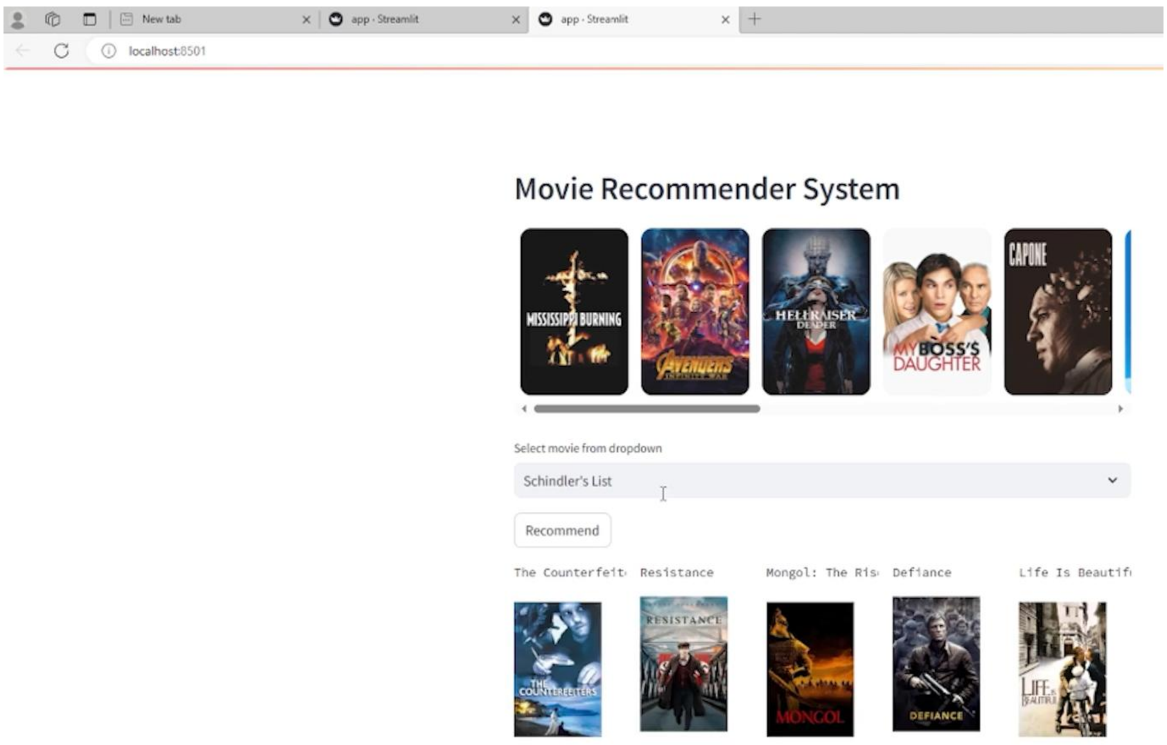


Figure 47 - Local Host UI

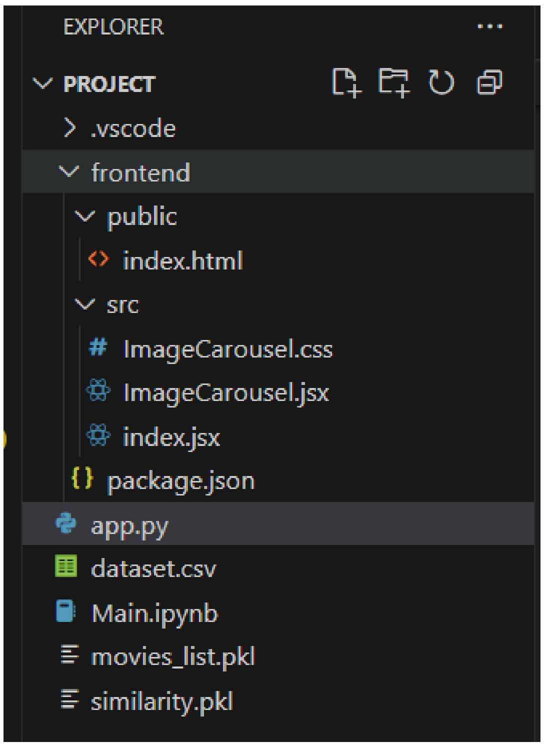
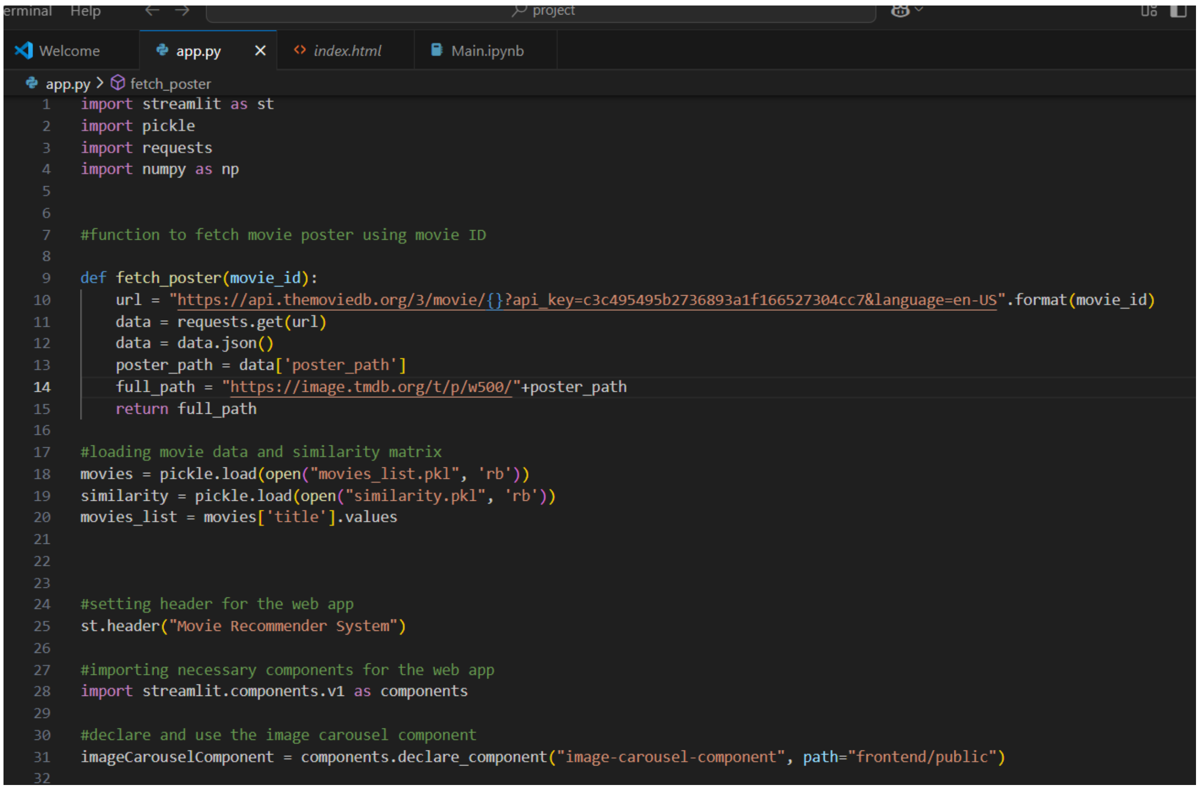
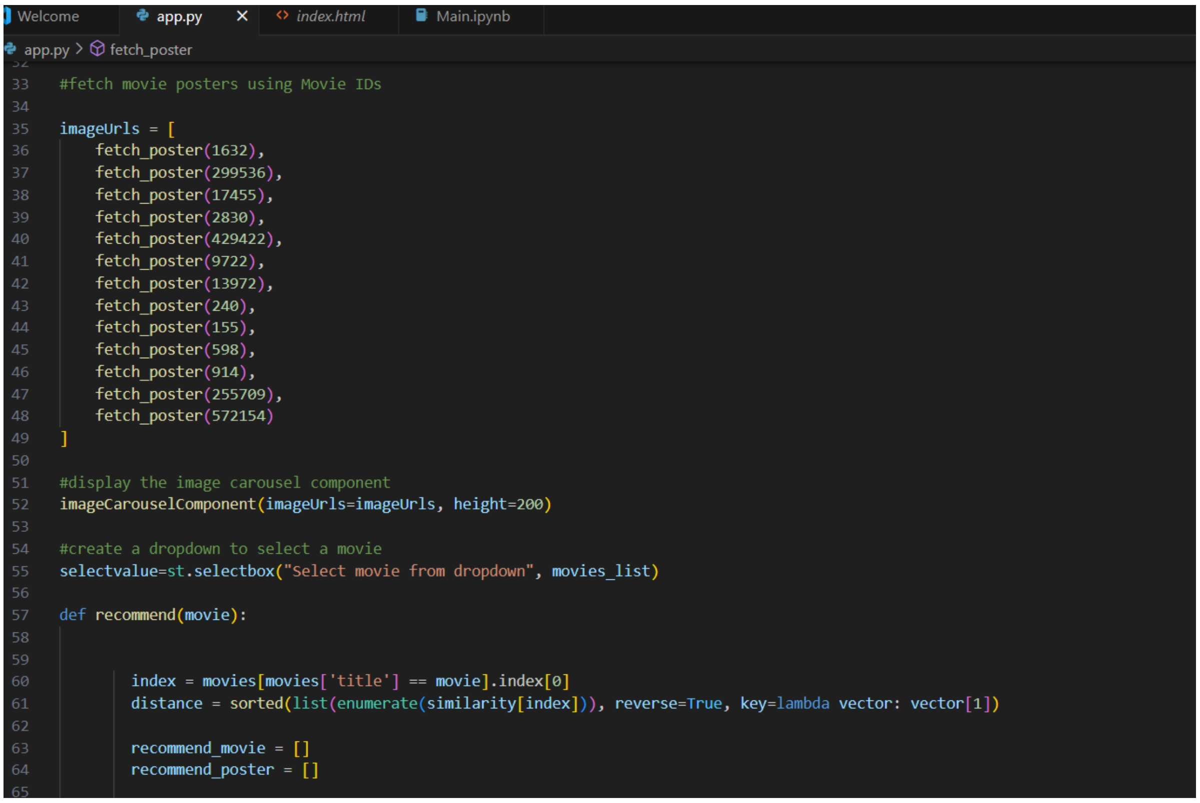


Figure 48 - Folder Hierarchy



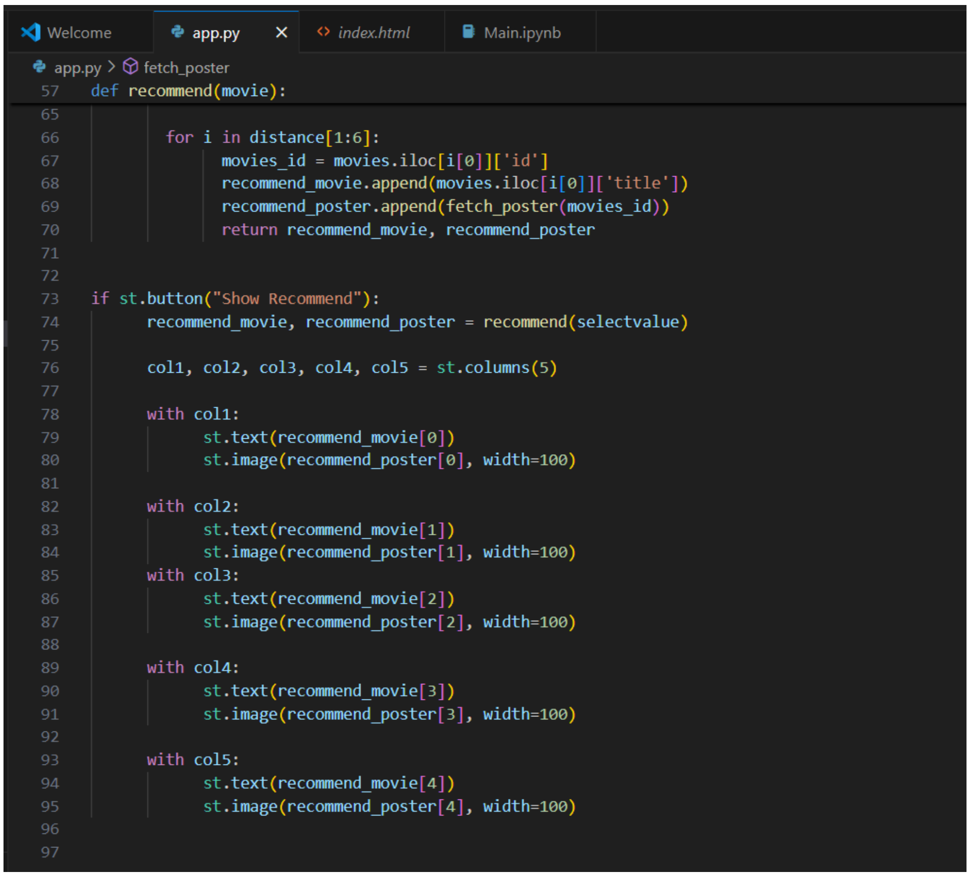
```python
import streamlit as st
import pickle
import requests
import numpy as np


#function to fetch movie poster using movie ID

def fetch_poster(movie_id):
    url = "https://api.themoviedb.org/3/movie/{}?api_key=c3c495495b2736893a1f166527304cc7&language=en-US".format(movie_id)
    data = requests.get(url)
    data = data.json()
    poster_path = data['poster_path']
    full_path = "https://image.tmdb.org/t/p/w500/"+poster_path
    return full_path

#loading movie data and similarity matrix
movies = pickle.load(open("movies_list.pkl", 'rb'))
similarity = pickle.load(open("similarity.pkl", 'rb'))
movies_list = movies['title'].values



#setting header for the web app
st.header("Movie Recommender System")

#importing necessary components for the web app
import streamlit.components.v1 as components

#declare and use the image carousel component
imageCarouselComponent = components.declare_component("image-carousel-component", path="frontend/public")

```

Figure 49 - app.py [1]



```python
#fetch movie posters using Movie IDs

imageUrls = [
    fetch_poster(1632),
    fetch_poster(299536),
    fetch_poster(17455),
    fetch_poster(2830),
    fetch_poster(429422),
    fetch_poster(9722),
    fetch_poster(13972),
    fetch_poster(240),
    fetch_poster(155),
    fetch_poster(598),
    fetch_poster(914),
    fetch_poster(255709),
    fetch_poster(572154)
]

#display the image carousel component
imageCarouselComponent(imageUrls=imageUrls, height=200)

#create a dropdown to select a movie
selectvalue=st.selectbox("Select movie from dropdown", movies_list)

def recommend(movie):


        index = movies[movies['title'] == movie].index[0]
        distance = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda vector: vector[1])

        recommend_movie = []
        recommend_poster = []

```

Figure 50 - app.py [2]



```python
def recommend(movie):

        for i in distance[1:6]:
            movies_id = movies.iloc[i[0]]['id']
            recommend_movie.append(movies.iloc[i[0]]['title'])
            recommend_poster.append(fetch_poster(movies_id))
            return recommend_movie, recommend_poster


if st.button("Show Recommend"):
    recommend_movie, recommend_poster = recommend(selectvalue)

    col1, col2, col3, col4, col5 = st.columns(5)

    with col1:
        st.text(recommend_movie[0])
        st.image(recommend_poster[0], width=100)

    with col2:
        st.text(recommend_movie[1])
        st.image(recommend_poster[1], width=100)
    with col3:
        st.text(recommend_movie[2])
        st.image(recommend_poster[2], width=100)

    with col4:
        st.text(recommend_movie[3])
        st.image(recommend_poster[3], width=100)

    with col5:
        st.text(recommend_movie[4])
        st.image(recommend_poster[4], width=100)
```

Figure 51 - app.py [3]

13 CHAPTER 06: OVERVIEW OF TECH STACK

Project Phase	**Tools & Technologies Used**
Models	Content-Based Filtering & Item-Based Collaborative Filtering
Tools	VS Code, Google Collab
Dataset	Kaggle.com
Frontend Implementation	Streamlit
Language	Python 3.13.1
Python Libraries & Modules	Pandas, Numpy, NLTK, Scikit-Learn, Pickle, SciPy, Csr_Matrix
Hosting	Localhost
API	TMDb (The Movie Database) API

Table 1- Overview of Tech Stack

14 CHAPTER 07: FUTURE WORK

For future improvements and additions to the Movie Recommendation System, the following recommendations can be considered:

1. **Incorporating More Features for Recommendation:**
   - Currently, the system relies on genre and overview for recommendations. Other attributes, such as director, cast, release year, and user ratings, can be integrated to enhance the recommendation quality.
   - Integrating user preferences or feedback through ratings can refine suggestions by personalizing the recommendations.
2. **Visualizing Recommendations:**
   - Enhance the user interface in Streamlit by displaying additional information about recommended movies, such as movie posters, rating statistics, synopsis, and even trailer previews.
   - Implement interactive elements like allowing users to filter recommendations by genre, release date, or rating.

3. **Real-Time Personalization:**
   - Incorporating user preferences into the recommendation system through user profiles and user-item interactions (e.g., click-through rate, watch history) could make the system more personalized.
   - Allow users to rate or "like" movies, which could be incorporated to refine the recommendations.

15 REFERENCES

- Dataset Links:
  a. For Content Based Filtering:
     https://www.kaggle.com/datasets/ahsanaseer/top-rated-tmdb-movies-10k
  b. For Item-Based Collaborative Filtering:
     https://www.kaggle.com/datasets/shubhammehta21/movie-lens-small-latest-dataset
- Github Repository Link:
  https://github.com/Shabeeb321/Movie-Recommendation-System
- Google Collab Link:
  https://colab.research.google.com/drive/1WizATYmu9ivDBCXDKM4ZyNVnf_8Tg7o#scrollTo=Hh5TdC6mFADJ