A REPORT ON

MOVIE-RECOMMENDER-SYSTEM

By

Shreyaan Subhankar (2020B5A72170H)

Kunal Bansal (2020B2A72345H)

Vanshika Jain (2020B4A12268H)

Under Supervision of
Dr. Aneesh Sreevallabh Chivukula

Course Code: CS F407

Course Title: Artificial Intelligence

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI HYDERABAD CAMPUS

(21th June 2023)

1. Stage One - Business Understanding

1.1 Assess the Current Situation

The movie industry is highly competitive and vast, with a vast number of movies being released regularly. With such a vast selection, users often struggle to discover movies that suit their tastes. Therefore, a movie recommender system can be a valuable tool to assist users in finding relevant and enjoyable content.

1.1.1. Requirements, assumptions and constraints -

- Requirements of the proper dataset
- Only limited number of movies in any given dataset.

1.1.2. Objective

- The objective of the movie recommender system is to provide personalized movie recommendations to users based on movie metadata.
- By analysing various aspects of movies such as genres, keywords, cast, crew, and overview, the system aims to offer suggestions that align with users' preferences and interests.

1.1.3. Goal

- The primary goal of this system is to enhance user satisfaction and engagement by providing tailored movie recommendations.
- By leveraging movie metadata and analysing similarities between movies, the system can identify patterns and connections that may not be apparent to users.
- This can lead to the discovery of hidden gems and the exploration of movies outside users' usual preferences.

1.2 What Questions Are We Trying To Answer?

- Moreover, the movie recommender system can gather valuable data on user preferences and viewing habits. This data can be leveraged for further business intelligence, such as targeted marketing, content acquisition decisions, and improving the overall user experience.
- Overall, the movie recommender system aims to meet the growing demand for personalized movie recommendations, enhance user satisfaction and engagement, drive business growth, and enable data-driven decision-making in the movie industry.

2. Stage Two - Data Understanding

Understanding the data is crucial for extracting meaningful insights and building an effective recommender system.

2.1 Initial Data Report

In [ ]:

We obtained the dataset from TMDB (https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata?select=tmdb_5000_movies.csv).

There are two files which we used:

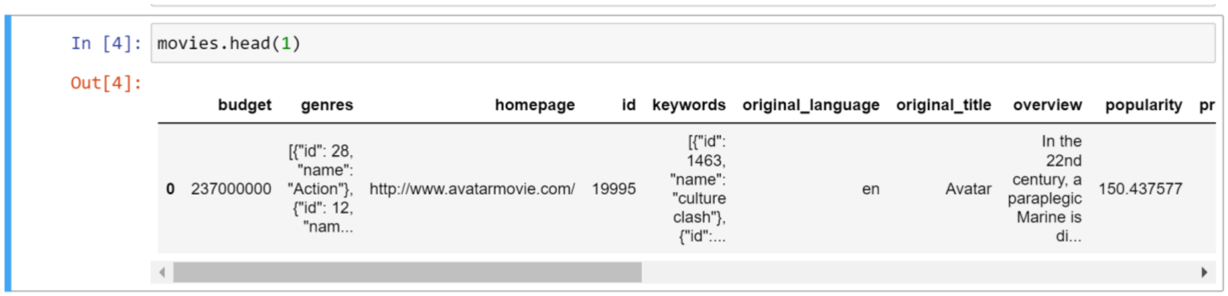
1. **tmdb_5000_movies.csv:** It includes 20 columns and it contains information about movies, including features such as movie ID, title, genres, keywords, overview, and other metadata..

2. **tmdb_5000_credtis.csv:** It includes 4 columns and the dataset complements the movie dataset by providing details about the movie credits, including cast and crew information - movie_id, title, cast, and crew.

2.2 Describing Data

In [ ]:

tmdb_5000_movies.columns:

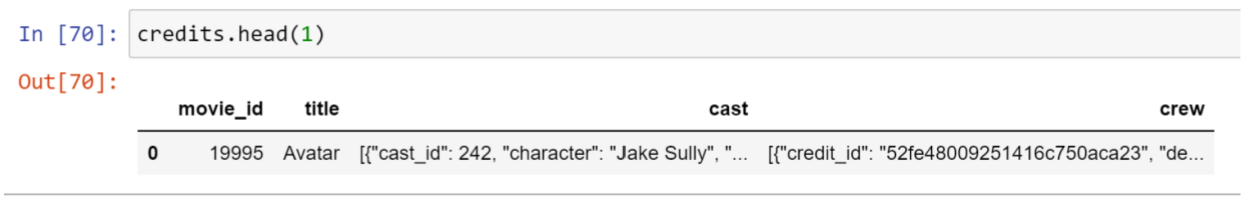


```
In [4]: movies.head(1)
Out[4]:
```

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	pr
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	

In [ ]:

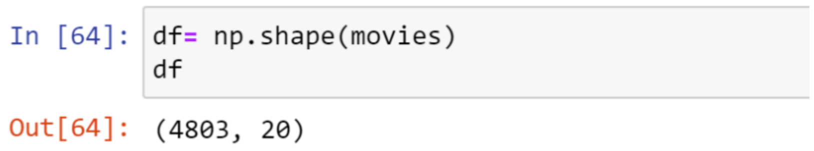
tmdb_5000_credits.columns:



```
In [70]: credits.head(1)
Out[70]:
```

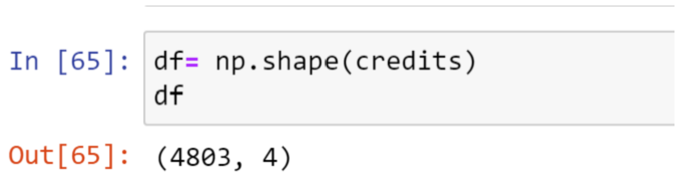
	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...

tmdb_5000_movies.shape



```
In [64]: df= np.shape(movies)
         df

Out[64]: (4803, 20)
```

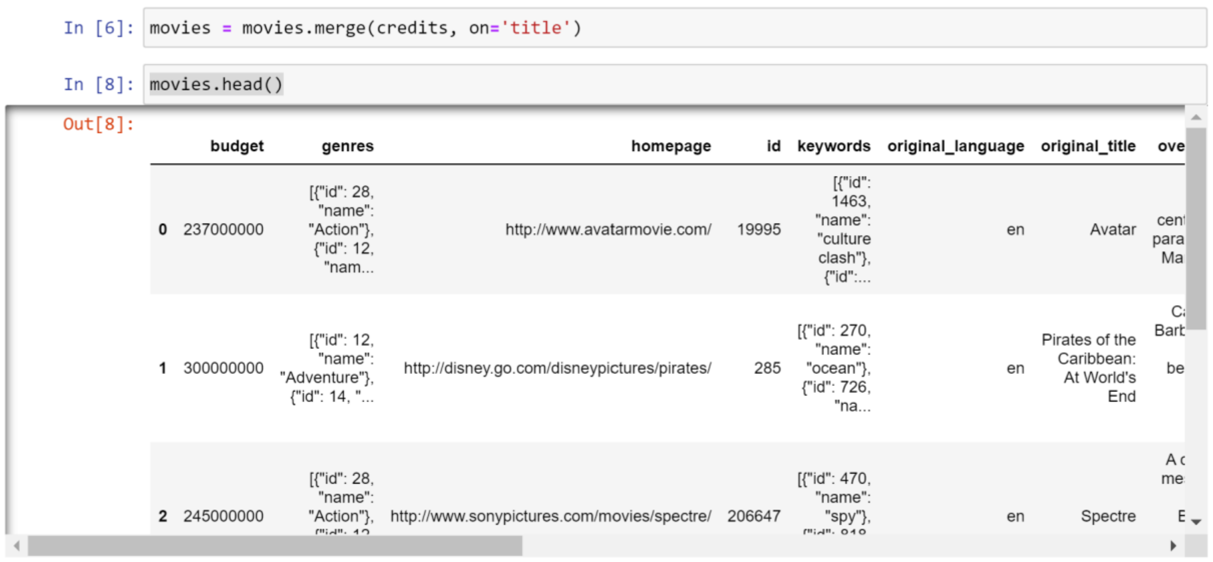
tmdb_5000_credits.shape



```
In [65]: df= np.shape(credits)
         df

Out[65]: (4803, 4)
```

2.3 Merging Datasets

Since both the datasets complement each other, we merged both of them on the common attribute title:



```
In [6]: movies = movies.merge(credits, on='title')

In [8]: movies.head()

Out[8]:
```

	budget	genres	homepage	id	keywords	original_language	original_title	ove
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	cent para Ma
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	C Barb be
2	245000000	[{"id": 28, "name": "Action"},	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"},	en	Spectre	A c me E

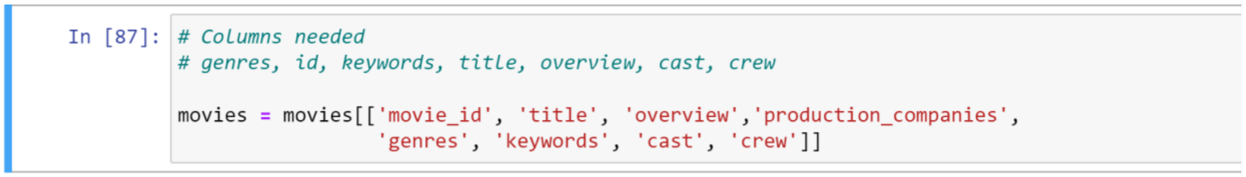
2.4 Initial Data Exploration

During the initial data exploration, the 'tmdb_5000_movies.csv' and 'tmdb_5000_credits.csv' datasets are examined to understand the structure and content. The dimensions of the datasets are checked, and a sample of records is inspected to gain insights into the available features. This process helps to identify the relevant variables for the movie recommender system, such as movie ID, title, genres, keywords, cast, crew, and overview as we did before. Moreover, an assessment is conducted to identify any missing values or duplicated records, ensuring the data's quality and integrity. This initial exploration sets the stage for further analysis and prepares the data for subsequent steps in building an effective movie recommender system.

2.4.1 Selecting Attributes

By business logic and common understanding, we need to eradicate the unnecessary columns which would not help in the recommendation.

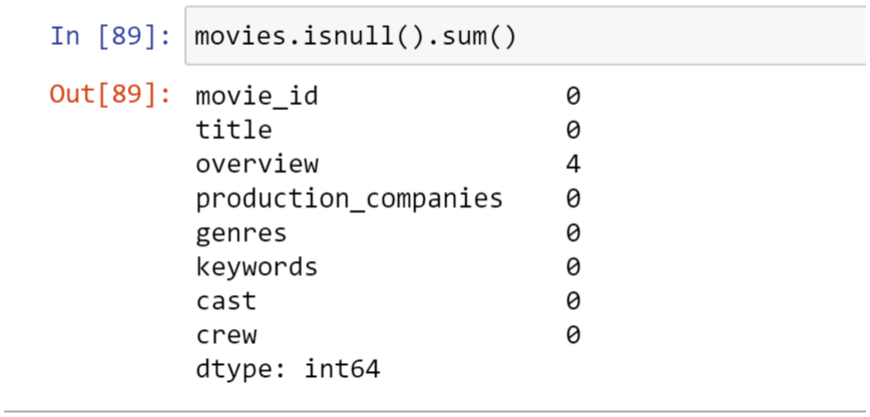
So we came up with 8 attributes that will give us the most personalized movies.



```
In [87]: # Columns needed
         # genres, id, keywords, title, overview, cast, crew

         movies = movies[['movie_id', 'title', 'overview','production_companies',
                          'genres', 'keywords', 'cast', 'crew']]
```

2.4.2. Missing Data

In addition to incorrect datatypes, another common problem when dealing with dataset is missing values. These can arise for many reasons and must be either filled in or removed before we implement our algorithm. First, let’s get a sense of how many missing values are in each column.



```
In [89]: movies.isnull().sum()

Out[89]: movie_id                0
         title                   0
         overview                4
         production_companies    0
         genres                  0
         keywords                0
         cast                    0
         crew                    0
         dtype: int64
```

We can see there are only 4 null values in the whole dataset, so since it doesn’t contribute as a major percentage, so we can directly eradicate the rows from the dataset.

3. Data Preparation

In this stage, we have to refine the columns of the movie dataset that we prepared. We need to convert the data into unified format for better data set.

3.1 Data Pre-Processing

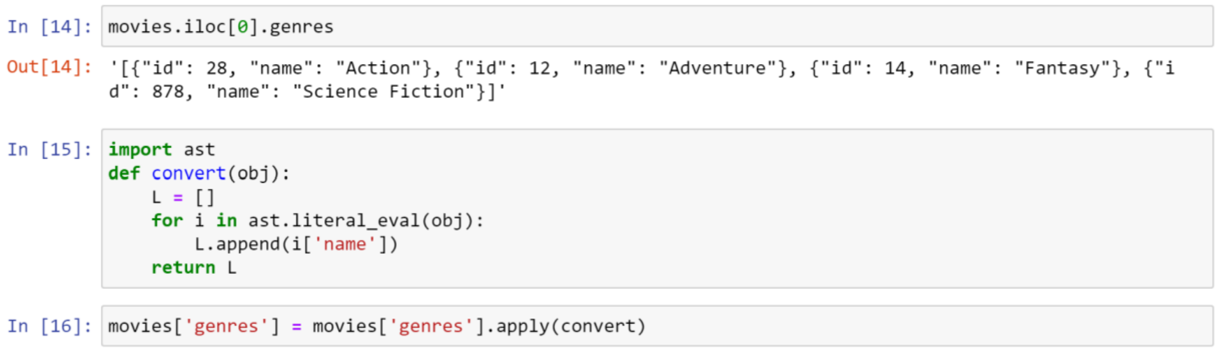
We will pre process dataset on some specific columns like genres, keywords, cast, crew, overview.

3.1.1 Genres

To convert a list of dictionaries to a list of names, we first need to remove the string format and convert it into an actual list. We can achieve this by using the `ast` module's `literal_eval()` function. This function allows us to safely evaluate and parse the string representation of a list into its corresponding Python object.

By applying `literal_eval()` to the string of list of dictionaries, we obtain the desired list of dictionaries. We can then iterate over each dictionary in the list and extract the value associated with the 'name' key. These values represent the names of the genres.

Finally, we collect all the genre names into a new list. This process effectively converts the list of dictionaries to a list of genre names.

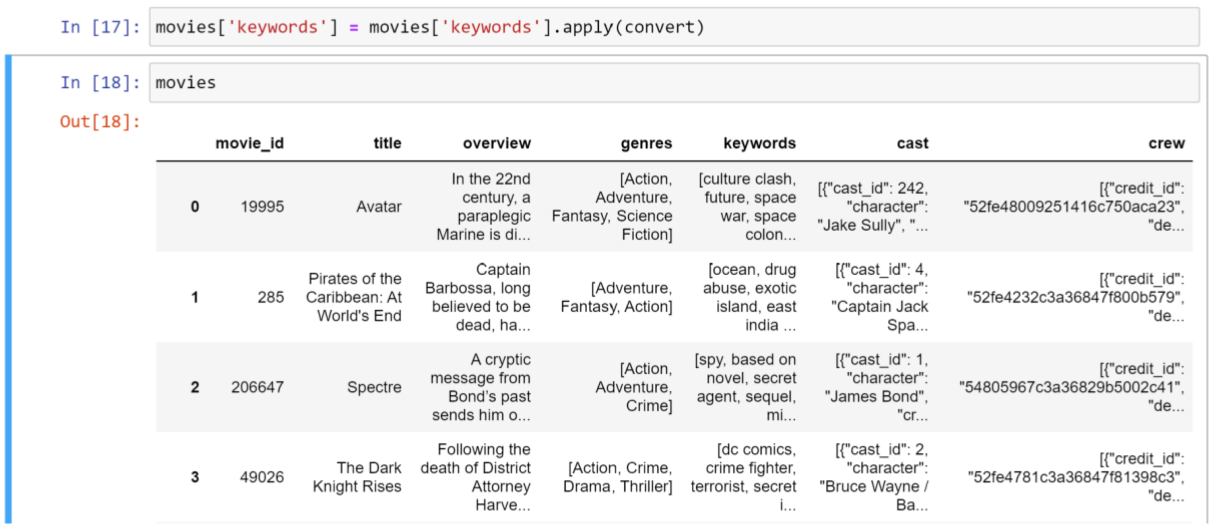


```
In [14]: movies.iloc[0].genres

Out[14]: '[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"i
         d": 878, "name": "Science Fiction"}]'

In [15]: import ast
         def convert(obj):
             L = []
             for i in ast.literal_eval(obj):
                 L.append(i['name'])
             return L

In [16]: movies['genres'] = movies['genres'].apply(convert)
```

.

3.1.2 Keywords

The same function is applied on keywords also.

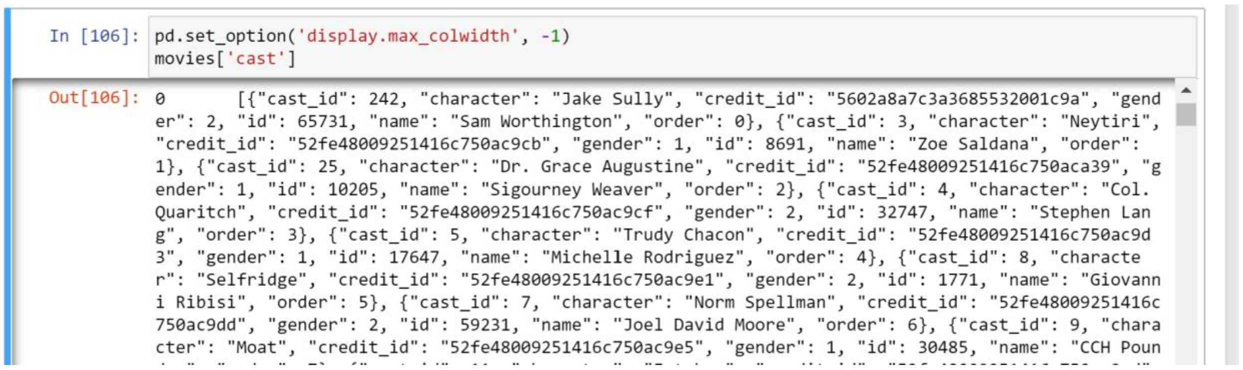


```
In [17]: movies['keywords'] = movies['keywords'].apply(convert)

In [18]: movies

Out[18]:
```

	movie_id	title	overview	genres	keywords	cast	crew
0	19995	Avatar	In the 22nd century, a paraplegic Marine is di...	[Action, Adventure, Fantasy, Science Fiction]	[culture clash, future, space war, space colon...	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	[Adventure, Fantasy, Action]	[ocean, drug abuse, exotic island, east india ...	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	A cryptic message from Bond's past sends him o...	[Action, Adventure, Crime]	[spy, based on novel, secret agent, sequel, mi...	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	Following the death of District Attorney Harve...	[Action, Crime, Drama, Thriller]	[dc comics, crime fighter, terrorist, secret i...	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...

3.1.3 Cast

Our data looks like:

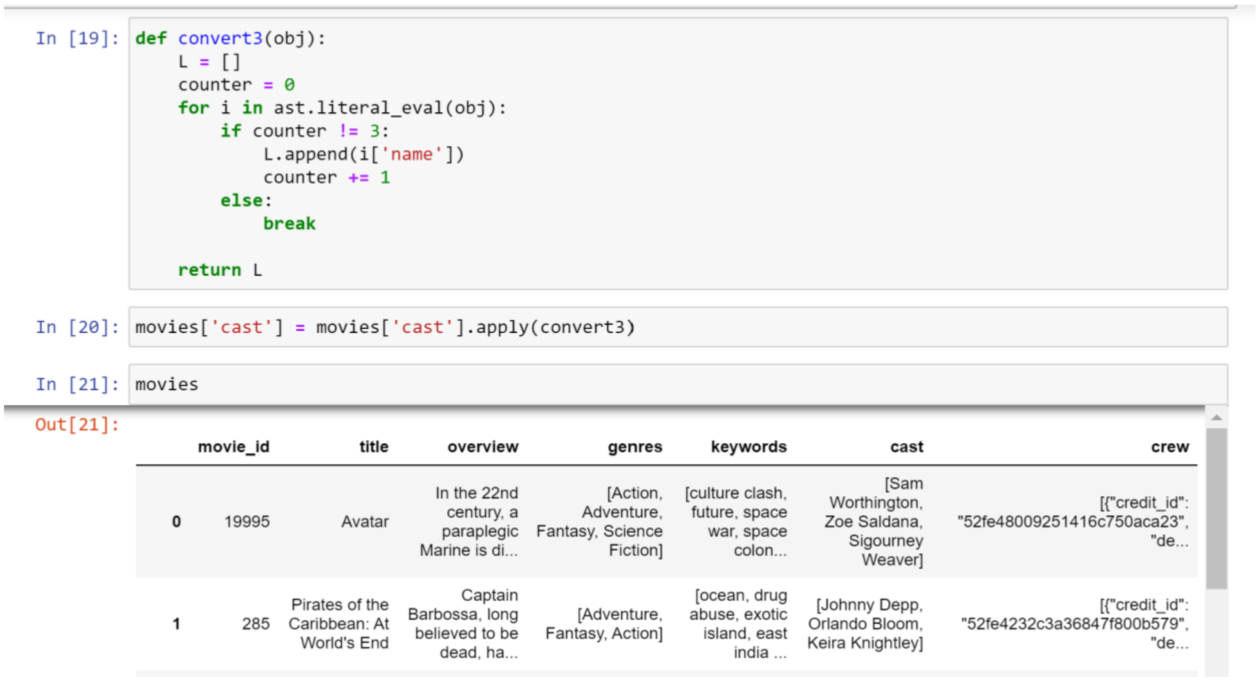


```
In [106]: pd.set_option('display.max_colwidth', -1)
          movies['cast']
```

```
Out[106]: 0       [{"cast_id": 242, "character": "Jake Sully", "credit_id": "5602a8a7c3a3685532001c9a", "gend
er": 2, "id": 65731, "name": "Sam Worthington", "order": 0}, {"cast_id": 3, "character": "Neytiri",
"credit_id": "52fe48009251416c750ac9cb", "gender": 1, "id": 8691, "name": "Zoe Saldana", "order":
1}, {"cast_id": 25, "character": "Dr. Grace Augustine", "credit_id": "52fe48009251416c750aca39", "g
ender": 1, "id": 10205, "name": "Sigourney Weaver", "order": 2}, {"cast_id": 4, "character": "Col.
Quaritch", "credit_id": "52fe48009251416c750ac9cf", "gender": 2, "id": 32747, "name": "Stephen Lan
g", "order": 3}, {"cast_id": 5, "character": "Trudy Chacon", "credit_id": "52fe48009251416c750ac9d
3", "gender": 1, "id": 17647, "name": "Michelle Rodriguez", "order": 4}, {"cast_id": 8, "characte
r": "Selfridge", "credit_id": "52fe48009251416c750ac9e1", "gender": 2, "id": 1771, "name": "Giovann
i Ribisi", "order": 5}, {"cast_id": 7, "character": "Norm Spellman", "credit_id": "52fe48009251416c
750ac9dd", "gender": 2, "id": 59231, "name": "Joel David Moore", "order": 6}, {"cast_id": 9, "chara
cter": "Moat", "credit_id": "52fe48009251416c750ac9e5", "gender": 1, "id": 30485, "name": "CCH Poun
```

To extract the names of the top actors from the 'crew' data, we have created a function called 'convert3'. In this function, we specifically focus on retrieving the names of the top 3 actors in the crew.

The 'convert3' function iterates over the list of dictionaries in the 'crew' data and checks the 'job' key to identify crew members with the job of an actor. It retrieves the 'name' value of each actor and appends it to a list, limited to a maximum of 3 actors.By utilizing the 'convert3' function, we can extract the names of the top actors from the 'crew' data, which can be further utilized for analysis or any relevant processing within the movie recommender system.



```
In [19]: def convert3(obj):
             L = []
             counter = 0
             for i in ast.literal_eval(obj):
                 if counter != 3:
                     L.append(i['name'])
                     counter += 1
                 else:
                     break

             return L
```

```
In [20]: movies['cast'] = movies['cast'].apply(convert3)
```

```
In [21]: movies
```

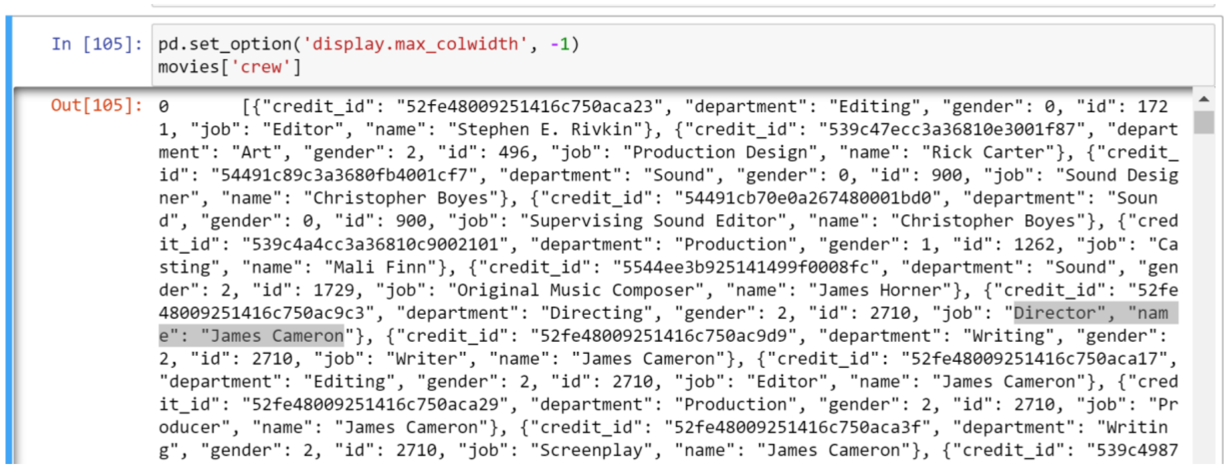
Out[21]:

	movie_id	title	overview	genres	keywords	cast	crew
0	19995	Avatar	In the 22nd century, a paraplegic Marine is di...	[Action, Adventure, Fantasy, Science Fiction]	[culture clash, future, space war, space colon...	[Sam Worthington, Zoe Saldana, Sigourney Weaver]	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	[Adventure, Fantasy, Action]	[ocean, drug abuse, exotic island, east india ...	[Johnny Depp, Orlando Bloom, Keira Knightley]	[{"credit_id": "52fe4232c3a36847f800b579", "de...

So the crew is processed now and we get the names of the top 3 actors.

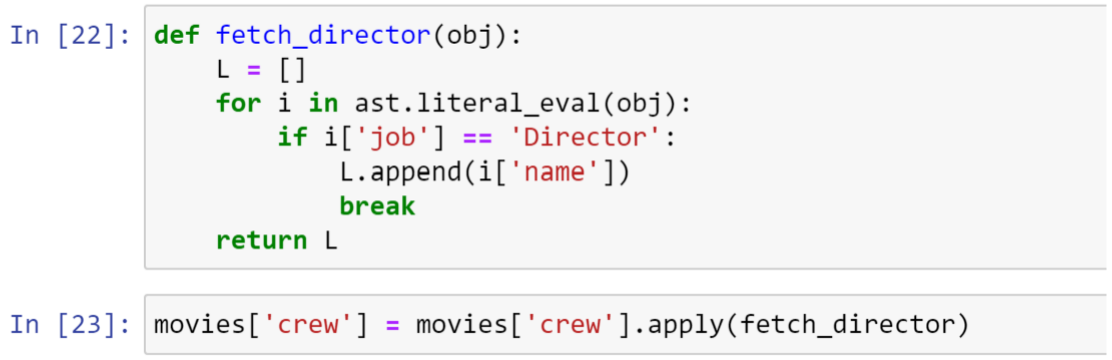
3.1.4 Crew

Our data looks like:


```
In [105]: pd.set_option('display.max_colwidth', -1)
          movies['crew']

Out[105]: 0       [{"credit_id": "52fe48009251416c750aca23", "department": "Editing", "gender": 0, "id": 172
          1, "job": "Editor", "name": "Stephen E. Rivkin"}, {"credit_id": "539c47ecc3a36810e3001f87", "depart
          ment": "Art", "gender": 2, "id": 496, "job": "Production Design", "name": "Rick Carter"}, {"credit_
          id": "54491c89c3a3680fb4001cf7", "department": "Sound", "gender": 0, "id": 900, "job": "Sound Desig
          ner", "name": "Christopher Boyes"}, {"credit_id": "54491cb70e0a267480001bd0", "department": "Soun
          d", "gender": 0, "id": 900, "job": "Supervising Sound Editor", "name": "Christopher Boyes"}, {"cred
          it_id": "539c4a4cc3a36810c9002101", "department": "Production", "gender": 1, "id": 1262, "job": "Ca
          sting", "name": "Mali Finn"}, {"credit_id": "5544ee3b925141499f0008fc", "department": "Sound", "gen
          der": 2, "id": 1729, "job": "Original Music Composer", "name": "James Horner"}, {"credit_id": "52fe
          48009251416c750ac9c3", "department": "Directing", "gender": 2, "id": 2710, "job": "Director", "nam
          e": "James Cameron"}, {"credit_id": "52fe48009251416c750ac9d9", "department": "Writing", "gender":
          2, "id": 2710, "job": "Writer", "name": "James Cameron"}, {"credit_id": "52fe48009251416c750aca17",
          "department": "Editing", "gender": 2, "id": 2710, "job": "Editor", "name": "James Cameron"}, {"cred
          it_id": "52fe48009251416c750aca29", "department": "Production", "gender": 2, "id": 2710, "job": "Pr
          oducer", "name": "James Cameron"}, {"credit_id": "52fe48009251416c750aca3f", "department": "Writin
          g", "gender": 2, "id": 2710, "job": "Screenplay", "name": "James Cameron"}, {"credit_id": "539c4987
```

To extract the 'director' name from the 'crew' data, we need to process the list of dictionaries that contains details about the crew members' jobs. Specifically, we are interested in extracting the name of the crew member who holds the job of 'Director'.

To achieve this, we can iterate over the list of dictionaries and check the 'job' key for each dictionary. If the 'job' value matches 'Director', we can retrieve the corresponding 'name' value, which represents the director's name.By extracting the 'director' name from the 'crew' data, we can obtain the specific information we need for our analysis or any further processing related to the movie recommender system.

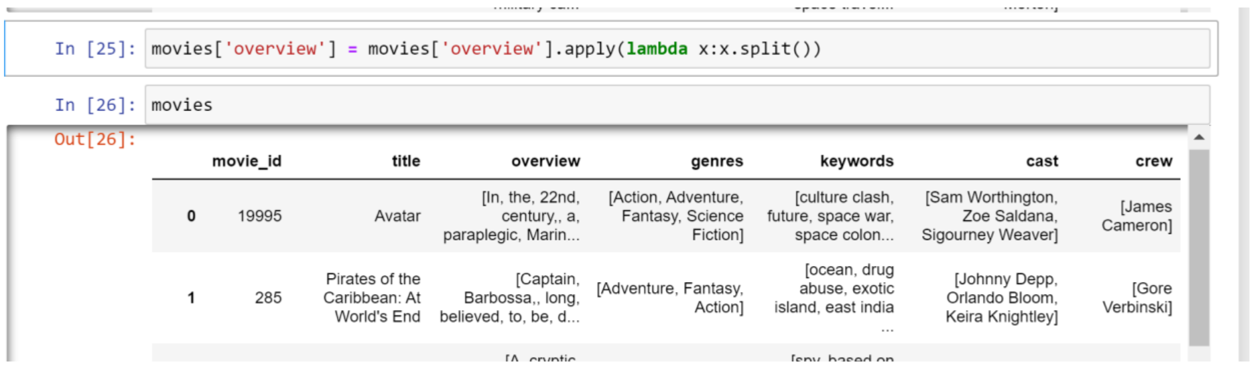

```
In [22]: def fetch_director(obj):
             L = []
             for i in ast.literal_eval(obj):
                 if i['job'] == 'Director':
                     L.append(i['name'])
                     break
             return L

In [23]: movies['crew'] = movies['crew'].apply(fetch_director)
```

3.1.5 Overview

'overview' is actually a string and all other columns are lists. Hence, we will convert the string to the list.

To pre-process the 'overview' column in the 'movies' dataset, a lambda function is applied using the 'apply' method. The lambda function splits each overview text into individual words by using the 'split()' function. By applying the lambda function to the 'overview' column, the text within each cell is split into a list of words. This allows for further processing and analysis on a word-level basis, such as extracting important keywords or performing natural language processing tasks.

The result of this pre-processing step is that the 'overview' column in the 'movies' dataset is transformed into a list of words for each movie. This can facilitate subsequent text-based analysis or feature extraction processes within the movie recommender system.



```
In [25]: movies['overview'] = movies['overview'].apply(lambda x:x.split())
```

```
In [26]: movies
Out[26]:
```

	movie_id	title	overview	genres	keywords	cast	crew
0	19995	Avatar	[In, the, 22nd, century,, a, paraplegic, Marin...	[Action, Adventure, Fantasy, Science Fiction]	[culture clash, future, space war, space colon...	[Sam Worthington, Zoe Saldana, Sigourney Weaver]	[James Cameron]
1	285	Pirates of the Caribbean: At World's End	[Captain, Barbossa,, long, believed, to, be, d...	[Adventure, Fantasy, Action]	[ocean, drug abuse, exotic island, east india ...	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Gore Verbinski]
			[A, cryptic		[spy, based on		

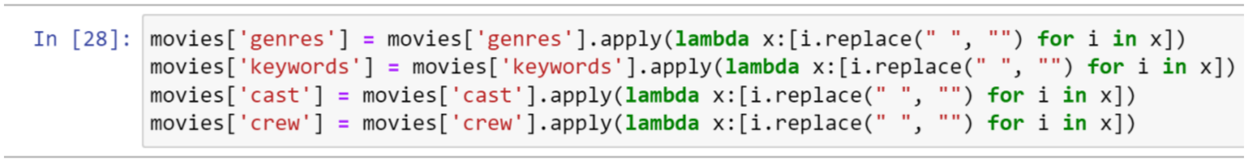
3.2 Clean The Data

To finalize the preprocessing steps, the next task is to concatenate the preprocessed columns ('overview', 'genres', 'keywords', 'cast', 'crew') into a single column called 'tags'. However, there is a concern regarding the presence of white spaces within the 'keywords', 'cast', and 'crew' columns, which could potentially lead to inefficiencies or confusion in the recommendation process.

For instance, consider the example of two individuals with the same first name, such as 'Sam Worthington' and 'Sam Mendes'. If the model treats these names as separate entities due to the white spaces, it may mistakenly recommend unrelated movies to users.

To address this issue, it is important to remove the white spaces within the 'keywords', 'cast', and 'crew' columns before concatenation. By removing the spaces, the model can correctly identify and associate related keywords, cast members, and crew members, resulting in more accurate recommendations.

This preprocessing step ensures that the 'tags' column contains a unified and cohesive representation of the movie features, enabling the recommender system to generate more reliable and relevant suggestions.



```
In [28]: movies['genres'] = movies['genres'].apply(lambda x:[i.replace(" ", "") for i in x])
         movies['keywords'] = movies['keywords'].apply(lambda x:[i.replace(" ", "") for i in x])
         movies['cast'] = movies['cast'].apply(lambda x:[i.replace(" ", "") for i in x])
         movies['crew'] = movies['crew'].apply(lambda x:[i.replace(" ", "") for i in x])
```

To remove the spaces in the 'genres', keywords, 'cast', 'crew' column in the 'movies' dataset, it is being processed using a lambda function. This lambda function applies a list comprehension to each value in the 'genres' column.

Within the list comprehension, each individual element in the list is processed by replacing any spaces (" ") with an empty string (""). This operation removes any spaces within the genre names.

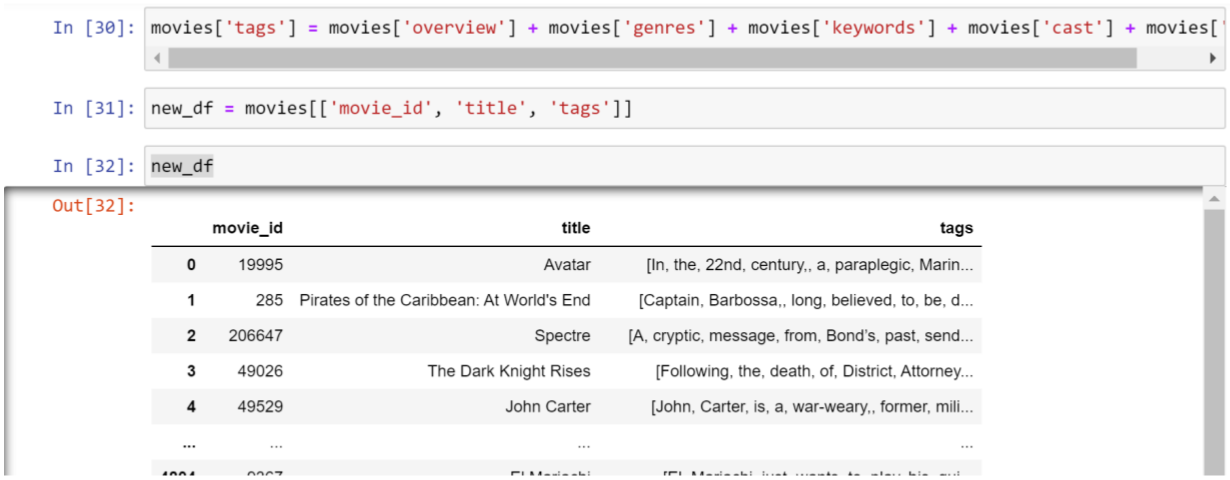
By applying this lambda function to the above column, each attribute value is modified to remove spaces, resulting in a clean and standardized representation of the genres. This can be beneficial for consistency in further analysis or when working with the genre data within the movie recommender system

3.2.1 Creating Tags

The 'tags' column is created in the code to serve as a consolidated representation of important movie features, including the 'overview', 'genres', 'keywords', 'cast', and 'crew'. By combining these elements into a single 'tags' column, the code aims to capture a comprehensive summary of each movie's characteristics.

This allows the recommender system to consider a broader range of information when generating recommendations. The 'tags' column provides a holistic view of a movie's content, genre, keywords, and key personnel, enabling more accurate and nuanced similarity calculations for effective movie recommendations.

Now our new data frame will only contain 3 columns: ‘id’, ‘title’, ‘tags’.

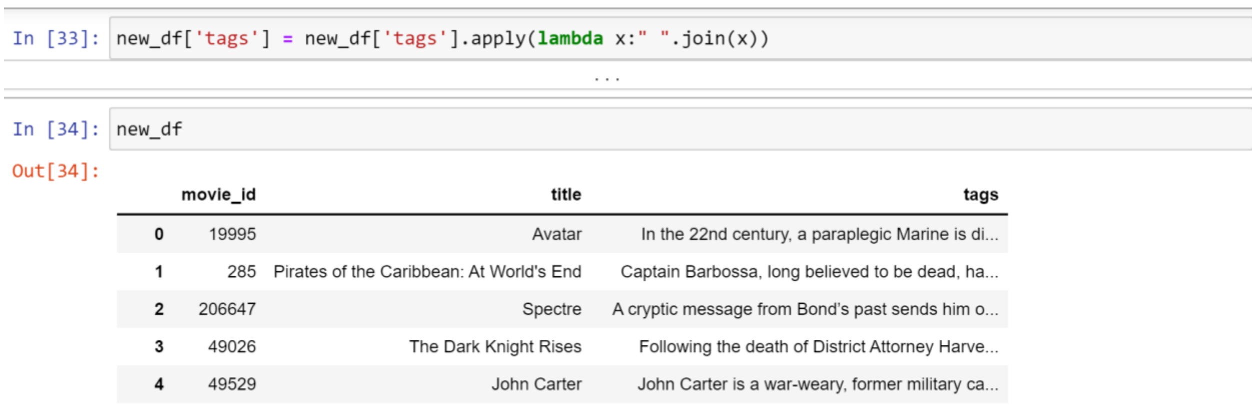


```
In [30]: movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] + movies['cast'] + movies['

In [31]: new_df = movies[['movie_id', 'title', 'tags']]

In [32]: new_df
Out[32]:
```

	movie_id	title	tags
0	19995	Avatar	[In, the, 22nd, century,, a, paraplegic, Marin...
1	285	Pirates of the Caribbean: At World's End	[Captain, Barbossa,, long, believed, to, be, d...
2	206647	Spectre	[A, cryptic, message, from, Bond's, past, send...
3	49026	The Dark Knight Rises	[Following, the, death, of, District, Attorney...
4	49529	John Carter	[John, Carter, is, a, war-weary,, former, mili...
...	...	...	...
4804	9367	El Mariachi	[El, Mariachi, just, wants, to, play, his, gui...

3.2.2 Making tags into paragraph

To ensure compatibility with the model, the 'tags' column needs to be converted into a single paragraph, represented as a string. This means concatenating the individual elements of the 'tags' column to form a cohesive and understandable text structure. By converting the 'tags' into a paragraph-like string, it becomes a suitable input format for the model, enabling it to process and interpret the combined movie features effectively.



```
In [33]: new_df['tags'] = new_df['tags'].apply(lambda x:" ".join(x))
```

```
In [34]: new_df
```

Out[34]:

	movie_id	title	tags
0	19995	Avatar	In the 22nd century, a paraplegic Marine is di...
1	285	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...
2	206647	Spectre	A cryptic message from Bond's past sends him o...
3	49026	The Dark Knight Rises	Following the death of District Attorney Harve...
4	49529	John Carter	John Carter is a war-weary, former military ca...

4. Modelling

To determine the similarity between movies and recommend similar ones, we need an efficient method to compare the 'tags' representing each movie. Manual comparison of words between tags is impractical and inefficient. Therefore, we employ a technique called vectorization, where each 'tags' text is transformed into a numerical vector representation.

4.1. Modelling technique

By vectorizing the 'tags', we convert the textual information into a format that can be compared using text similarity measures. This enables us to quantify the similarity between the vectors representing different movies' tags. To find similar movies, we calculate the similarity scores between the vectors and identify the five nearest vectors as the recommendations for each movie.

Vectorization and similarity measures provide an automated and scalable approach to assess the similarity between movies based on their textual information. This allows the recommender system to provide accurate and relevant recommendations to users, enhancing their movie-watching experience.

What is Vectorization Technique?

Vectorization is a fundamental concept in machine learning and natural language processing (NLP). It involves converting raw input data, such as text, into numerical vectors that can be processed and understood by machine learning models. This process enables models to work with textual data by extracting distinct features and representing them as numerical values. Various techniques can be used for vectorization, ranging from basic binary term occurrence features to more advanced context-aware representations. The choice of vectorization method depends on the specific use case and the requirements of the model. Ultimately, vectorization plays a crucial role in transforming text data into a format that can be effectively utilized for training and inference in machine learning applications.

4.2. Explaining Model

We use the technique called ‘**Bag of words’** in vector tokenization.

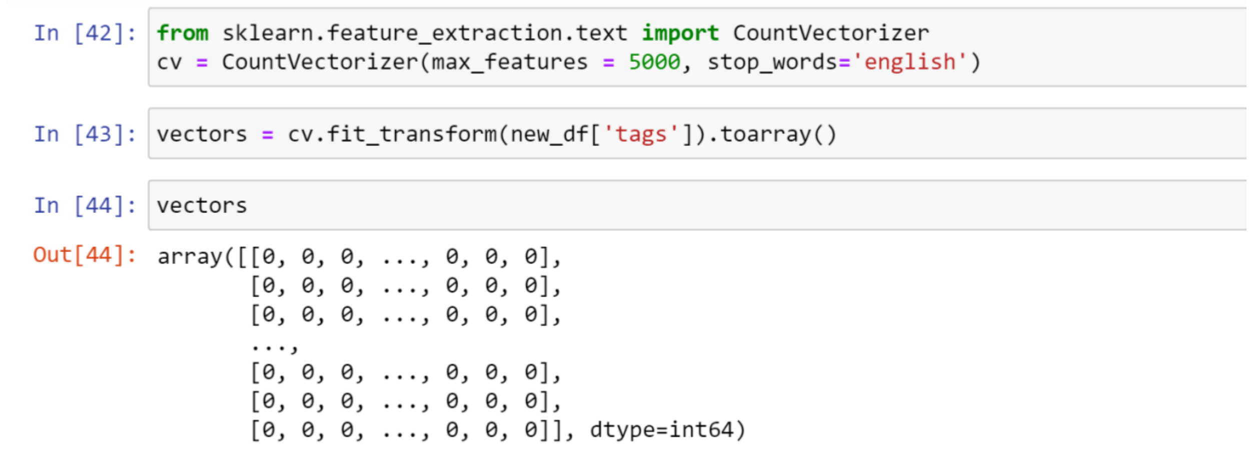
In the context of the movie recommender system, the 'Bag of Words' approach is used to represent the 'tags' data. The process involves combining all the 'tags' to form a large text corpus. From this corpus, the 5000 most frequent words are identified.

For each word, its frequency is calculated across each row of the 'tags' data. These frequency counts create vectors in a 5000x5000 dimensional space, where 5000 represents the number of words and 5000 represents the number of movies.

By representing each movie as a vector in this space, similarity calculations can be performed. The vectors are plotted against each other, using words as axes. The most similar vectors indicate movies with similar content or characteristics.

To ensure accurate vectorization, it is important to remove stop words. Stop words are common words, such as 'and', 'or', 'to', 'is', etc., that do not contribute significant meaning to the overall context of the 'tags' data. By removing stop words, the vectorization process focuses on more meaningful and informative words.

The 'Bag of Words' approach, combined with stop word removal and vectorization, allows for efficient representation and similarity calculations in the movie recommender system, leading to effective movie recommendations.



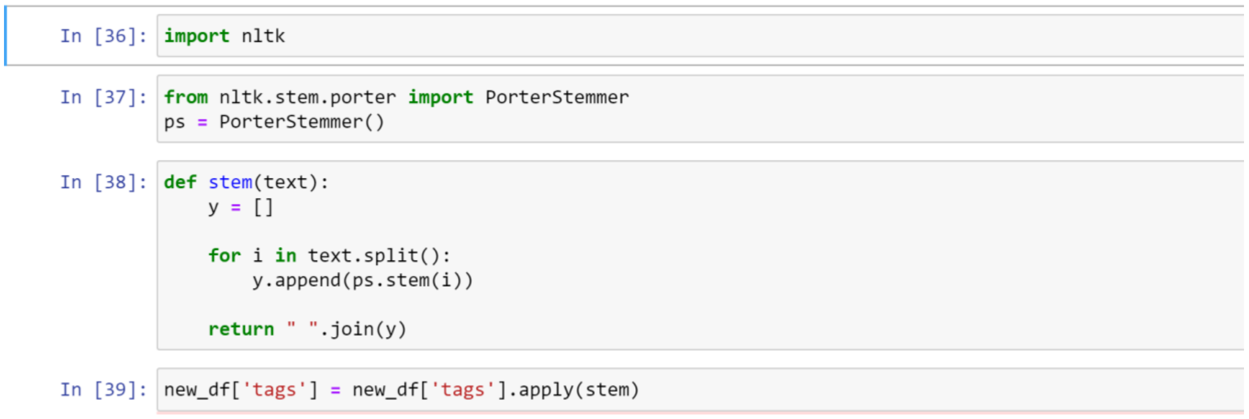
```
In [42]: from sklearn.feature_extraction.text import CountVectorizer
         cv = CountVectorizer(max_features = 5000, stop_words='english')

In [43]: vectors = cv.fit_transform(new_df['tags']).toarray()

In [44]: vectors

Out[44]: array([[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]], dtype=int64)
```

4.3. Build Model

Model settings - To address the issue of similar words appearing in different forms, such as "accident," "accidentally," and "accidental," stemming from the NLTK library is employed. Stemming is a technique used to reduce words to their base or root form.

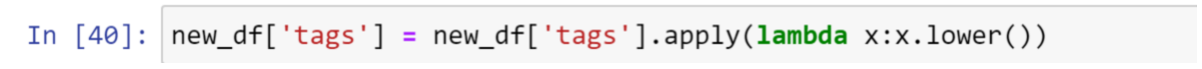
By applying stemming to the feature data, variations of words are transformed into their common root form. This allows for better normalization and consolidation of similar words. For example, all variations mentioned earlier would be reduced to the common root "accident" through stemming.

The use of stemming ensures that words with similar meanings or semantic connections are treated as the same entity during the vectorization process. This helps to improve the accuracy and effectiveness of the movie recommender system by capturing the underlying meaning of words while reducing redundancy and noise in the feature data.



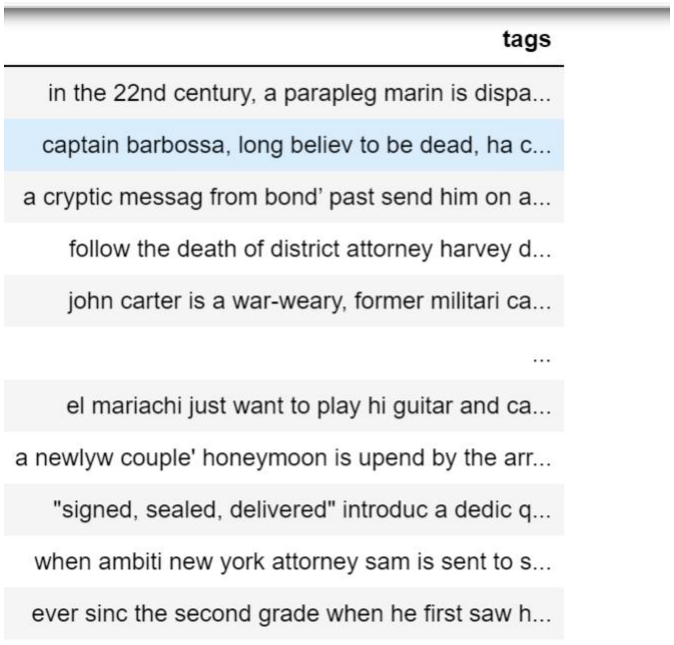
```
In [36]: import nltk

In [37]: from nltk.stem.porter import PorterStemmer
         ps = PorterStemmer()

In [38]: def stem(text):
             y = []

             for i in text.split():
                 y.append(ps.stem(i))

             return " ".join(y)

In [39]: new_df['tags'] = new_df['tags'].apply(stem)
```

Small casing the words:



```
In [40]: new_df['tags'] = new_df['tags'].apply(lambda x:x.lower())
```

Tags look like this now:



```
                                              tags
in the 22nd century, a parapleg marin is dispa...
captain barbossa, long believ to be dead, ha c...
a cryptic messag from bond' past send him on a...
follow the death of district attorney harvey d...
john carter is a war-weary, former militari ca...
                                               ...
el mariachi just want to play hi guitar and ca...
a newlyw couple' honeymoon is upend by the arr...
"signed, sealed, delivered" introduc a dedic q...
when ambiti new york attorney sam is sent to s...
ever sinc the second grade when he first saw h...
```

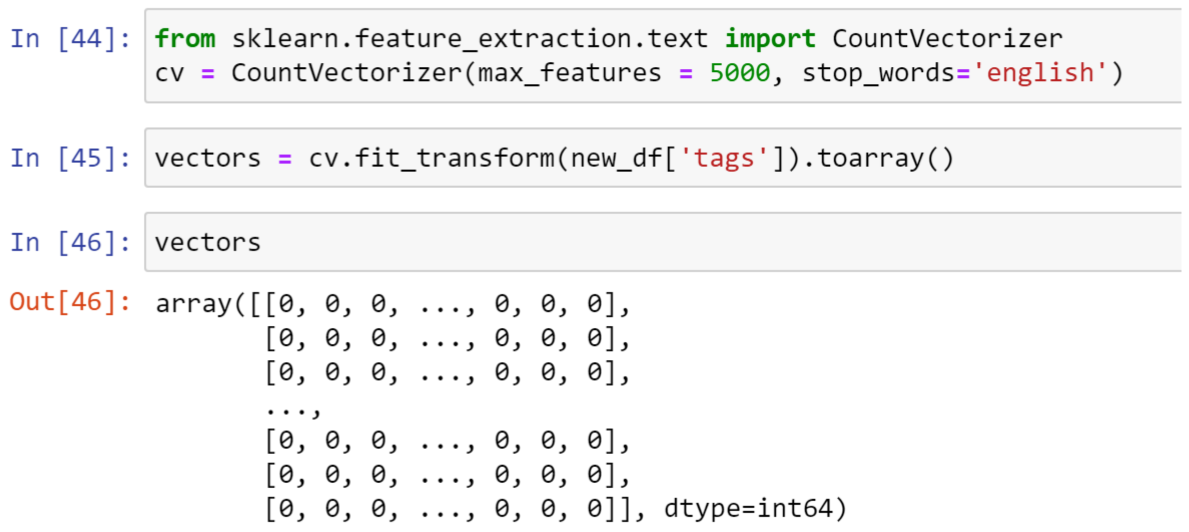
5. Model descriptions

5.1 Count Vectorization

In the context of vectorization, we will leverage scikit-learn, a powerful module. Within this module, we have the CountVectorizer class, which facilitates the process of vectorization. It takes two essential parameters: "max_features," representing the number of words or features to consider, and "stop_words" that comprises common English words to be excluded.

Now, utilizing the 'cv' object, we can transform our data into a vectorized form. By invoking the "fit_transform()" method, we obtain a sparse matrix in scikit-learn. To further process this data, it becomes necessary to convert it into a more manageable and useful format, such as a NumPy array.

Remarkably, this process allows us to retrieve the 5000 most frequent words or feature names within the given dataset. These words may encompass a range of information, including numbers or even years that hold particular significance within the context of the dataset.



```
In [44]: from sklearn.feature_extraction.text import CountVectorizer
         cv = CountVectorizer(max_features = 5000, stop_words='english')

In [45]: vectors = cv.fit_transform(new_df['tags']).toarray()

In [46]: vectors

Out[46]: array([[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]], dtype=int64)
```

5.2 Algorithm- Cosine Similarity

In the subsequent phase, we possess a collection of many movies, with each movie represented as a vector. Our objective now is to compute the distance or similarity between each pair of movies. However, it's important to note that we won't be using the traditional Euclidean distance. Instead, we will employ the concept of cosine angle between two movie vectors.

By measuring the cosine similarity, we can gauge how similar or related two movies are. A smaller distance indicates a higher degree of similarity between them. Luckily, scikit-learn provides a built-in function that allows us to easily calculate this cosine similarity. Baically we are applying clustering method.

Observe how the similarity between each movie and all other movies is presented. Notably, a movie exhibits a perfect similarity of 1.00 with itself, which is expected, forming a diagonal line of ones.

To arrange the vectors, we encounter an issue during the sorting process. If we sort the similarity scores directly, they will become disorganized, and that's not desirable since these scores play a crucial role in determining the movie indices. Therefore, we adopt a different approach: we utilize the "enumerate()" function to assign indices to the similarity scores. Then, we can sort them based on the values of similarity[movie_index], which enables us to maintain the association between movies and their corresponding similarity scores intact.

```
In [47]: from sklearn.metrics.pairwise import cosine_similarity
```

```
In [48]: similarity = cosine_similarity(vectors)
```

```
In [49]: sorted(list(enumerate(similarity[0])), reverse=True, key=lambda x:x[1])[1:6]

Out[49]: [(1216, 0.28676966733820225),
          (2409, 0.26901379342448517),
          (3730, 0.2605130246476754),
          (507, 0.255608593705383),
          (539, 0.2503866978335957)]
```

5.2 Main Function

Exactly, since we are interested in recommending only the top 5 movies for each movie, we employ slicing to extract the necessary information. However, we must omit the first score in the sorted similarity list, as it will always represent the similarity of the movie to itself, which is not relevant for recommendation purposes.

By slicing the sorted similarity scores from index 1 to index 6 (inclusive), we can obtain the top 5 most similar movies to the current movie, disregarding its own similarity score. This way, we ensure that the movie recommendations are accurate and meaningful.

```
In [50]: def recommend(movie):
             movie_index = new_df[new_df['title']==movie].index[0]
             distances = similarity[movie_index]
             movies_list = sorted(list(enumerate(distances)), reverse=True, key=lambda x:x[1])[1:6]

             for i in movies_list:
                 print(new_df.iloc[i[0]].title)
```

```
In [51]: recommend('Gandhi')

         Gandhi, My Father
         Guiana 1838
         The Wind That Shakes the Barley
         Mr. Turner
         A Passage to India
```