```
%pip install tqdm==4.66.4  | tail -n 1
%pip install pandas==2.1.4  | tail -n 1
%pip install scikit-learn==1.5.1  | tail -n 1
```

```
Successfully installed tqdm-4.66.4
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 2.1.4 which is incompatible.
Successfully installed pandas-2.1.4
Successfully installed scikit-learn-1.5.1
```

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors
import statistics
import numpy as np
from tqdm import tqdm
```

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors
import statistics
import numpy as np
from tqdm import tqdm
```

```
### You can also use this section to supress warnings generated by our code:

def warn(*args, **kwargs): pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')

warnings.filterwarnings('ignore')
```

The dataset is taken from Kaggle. This dataset describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. Users were selected at random for inclusion. No demographic information is included. Each user is represented by an ID, and no other information is provided.

The data are contained in the files movies.csv, ratings.csv and tags.csv.

In the movies.csv file:

movieId: ID of the movie/show (unique)

title: Title of the movie/show genres: Genre of the show In the ratings.csv file:

userId: ID of the user who gave a rating

movieId: ID of the movie/show rated rating: Rating given to the show timestamp: Time when the rating was specified In the tags.csv file:

userId: ID of the user who gave a rating

movieId: ID of the movie/show rated tag: Tags given to the show timestamp: Time when the rating was specified Now, let's load these datasets into a pandas DataFrame.

```
movie_df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/BxZuF3FrO7Bdw6McwsBaBw/movies.csv')
rating_df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/R-bYYyyf7s3IUE5rsssmMw/ratings.csv')
tag_df = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/UZKHhXSl7Ft7t9mfUFZJPQ/tags.csv')
```

Let's look at some samples rows from the dataset we loaded:

```
movie_df.sample(5)
```

	movieId	title	genres
7075	69757	(500) Days of Summer (2009)	Comedy|Drama|Romance
7622	87287	American Grindhouse (2010)	Documentary
865	1140	Entertaining Angels: The Dorothy Day Story (1996)	Drama
1689	2271	Permanent Midnight (1998)	Drama
5874	33090	Mutant Aliens (2001)	Animation|Comedy|Sci-Fi

```
tag_df.sample(5)
```

	userId	movieId	tag	timestamp
1537	474	1984	halloween	1137374180
1191	474	902	Capote	1138137849
2257	474	7156	Vietnam	1138039601
3037	567	5690	downbeat	1525283801
2552	477	1274	animation	1244787931

```
rating_df.sample(5)
```

	userId	movieId	rating	timestamp
80326	506	112556	3.0	1424486905
98160	606	6564	2.0	1172012541
28440	198	1653	5.0	1034135628
62794	414	1347	3.0	962371845
85411	555	1036	4.0	978820321

```
# We will merge the three dataframes to create a single dataframe that contains all the information we need.
user_movie_df = movie_df.merge(rating_df, on = 'movieId', how = 'inner')
df = user_movie_df.merge(tag_df, on = ['movieId', 'userId'], how = 'inner')
df
```

	movieId	title	genres	userId	rating	timestamp_x	tag	timestamp_y
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	336	4.0	1122227329	pixar	1139045764
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	474	4.0	978575760	pixar	1137206825
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	567	3.5	1525286001	fun	1525286013
3	2	Jumanji (1995)	Adventure|Children|Fantasy	62	4.0	1528843890	fantasy	1528843929
4	2	Jumanji (1995)	Adventure|Children|Fantasy	62	4.0	1528843890	magic board game	1528843932
...	...	...	...	...	...	...	...	...
3471	187595	Solo: A Star Wars Story (2018)	Action|Adventure|Children|Sci-Fi	62	4.0	1528934550	star wars	1528934552
3472	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	1537098554	anime	1537098582
3473	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	1537098554	comedy	1537098587
		Gintama: The Movie						

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Here, we will drop the timestamp columns as they are not needed for our analysis.
df.drop(columns = ['timestamp_x', 'timestamp_y'], inplace = True)
df
```

	movieId	title	genres	userId	rating	tag
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	336	4.0	pixar
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	474	4.0	pixar
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	567	3.5	fun
3	2	Jumanji (1995)	Adventure|Children|Fantasy	62	4.0	fantasy
4	2	Jumanji (1995)	Adventure|Children|Fantasy	62	4.0	magic board game
...	...	...	...	...	...	...
3471	187595	Solo: A Star Wars Story (2018)	Action|Adventure|Children|Sci-Fi	62	4.0	star wars
3472	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	anime
3473	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	comedy
3474	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	gintama
3475	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	remaster

3476 rows × 6 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

Exploratory data analysis (EDA)

Before doing any preprocessing, we will be performing some simple exploratory data analysis (EDA) to know about our dataset. This includes looking at the number of unique values/number of duplicate values, the distributions, etc.

First, looking at the shape of the pd.DataFrame

```
print('Number of rows: ' , df.shape[0])
print('Number of columns: ' , df.shape[1])
```

```
Number of rows:  3476
Number of columns:  6
```

Looking at the data type of each columns:

```
df.dtypes
```

	0
movieId	int64
title	object
genres	object
userId	int64
rating	float64
tag	object

dtype: object

Next, let's see if we have any null values:

```
# Deal with null values
df.isnull().any()
```

	0
movieId	False
title	False
genres	False
userId	False
rating	False
tag	False

dtype: bool

Popularity-based recommendation

The popularity based recommendation recommends items, in this case, movies, based on what is popular accross the site. It is the most basic recommendation system. The system identifies popular items by considering metrics such as the number of views, ratings, or purchases and suggests these items to all users. For this type of recommendation system, all users get the same recommendations. The system can suggest items based on what's popular in your country.

This approach ensures that users are aware of current popular content, which can be useful for new users who have not yet developed a viewing history on the platform. However, this is also a limitation because everyone receives the same suggestions, which may not always be relevant or interesting to them. This lack of specificity can result in a less engaging user experience compared to more personalized recommendation systems.

```
df_1 = df
df_1
```

	movieId	title	genres	userId	rating	tag
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	336	4.0	pixar
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	474	4.0	pixar
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	567	3.5	fun
3	2	Jumanji (1995)	Adventure|Children|Fantasy	62	4.0	fantasy
4	2	Jumanji (1995)	Adventure|Children|Fantasy	62	4.0	magic board game
...	...	...	...	...	...	...
3471	187595	Solo: A Star Wars Story (2018)	Action|Adventure|Children|Sci-Fi	62	4.0	star wars
3472	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	anime
3473	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	comedy
3474	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	gintama
3475	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	remaster

3476 rows × 6 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

Next, we will be calculating the number of votes and the average rating for each movie.

```
num_votes = df_1.groupby('movieId').size().reset_index(name='numVotes')

# Merge the numVotes back into the original DataFrame
df_1 = pd.merge(df_1, num_votes, on='movieId')

df_1
```

	movieId	title	genres	userId	rating	tag	numVotes
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	336	4.0	pixar	3
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	474	4.0	pixar	3
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	567	3.5	fun	3
3	2	Jumanji (1995)	Adventure|Children|Fantasy	62	4.0	fantasy	4
4	2	Jumanji (1995)	Adventure|Children|Fantasy	62	4.0	magic board game	4
...	...	...	...	...	...	...	...
3471	187595	Solo: A Star Wars Story (2018)	Action|Adventure|Children|Sci-Fi	62	4.0	star wars	2
3472	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	anime	4
3473	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	comedy	4
3474	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	gintama	4
3475	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	remaster	4

3476 rows × 7 columns

Next steps: Generate code with df_1 View recommended plots New interactive sheet

```
avg_ratings = df_1.groupby('movieId')['rating'].mean().reset_index(name='avgRating')

# Merge the avgRating back into the original DataFrame
df_1 = pd.merge(df_1, avg_ratings, on='movieId')
```

```
df_1.drop_duplicates(subset = ['movieId', 'title', 'avgRating', 'numVotes'], inplace = True)
df_1
```

	movieId	title	genres	userId	rating	tag	numVotes	avgRating
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	336	4.0	pixar	3	3.833333
3	2	Jumanji (1995)	Adventure|Children|Fantasy	62	4.0	fantasy	4	3.750000
7	3	Grumpier Old Men (1995)	Comedy|Romance	289	2.5	moldy	2	2.500000
9	5	Father of the Bride Part II (1995)	Comedy	474	1.5	pregnancy	2	1.500000
11	7	Sabrina (1995)	Comedy|Romance	474	3.0	remake	1	3.000000
...	...	...	...	...	...	...	...	...
3461	183611	Game Night (2018)	Action|Comedy|Crime|Horror	62	4.0	Comedy	3	4.000000
3464	184471	Tomb Raider (2018)	Action|Adventure|Fantasy	62	3.5	adventure	3	3.500000
3467	187593	Deadpool 2 (2018)	Action|Comedy|Sci-Fi	62	4.0	Josh Brolin	3	4.000000
3470	187595	Solo: A Star Wars Story (2018)	Action|Adventure|Children|Sci-Fi	62	4.0	Emilia Clarke	2	4.000000
3472	193565	Gintama: The Movie (2010)	Action|Animation|Comedy|Sci-Fi	184	3.5	anime	4	3.500000

Next steps: Generate code with df_1 View recommended plots New interactive sheet

We will be calculating the weighted score for each type. Usually, we would think that a good score results when the rating is high and the number of votes is also high. For instance, suppose you were browsing to choose a restaurant to dine at on your trip. If restaurant A had score 8.5 with 100,000 votes and restaurant B had score 8.5 but with 10 votes, we would be more convinced that restaurant A is more enjoyable and popular. Similarly, if restaurant C had score 5.0 with 1000 votes and restaurant D had score 5.0 with 1 vote, we may not automatically think that restaurant D was not enjoyable (but we do know that it is not popular), since only one person submitted a rating, if another person gave it score 10, this would immediately bump the score of restaurant D to 7.5.

The code below creates a new column df['score'] that calculates the weighted average score for each movie.

```
import statistics

# Define the function to calculate the weighted score
def calculate_weighted_score(avgRating, num_votes, C, m):
    return (num_votes * avgRating + m * C) / (num_votes + m)
```

```
# Calculate the global average rating (C)
average_rating = statistics.mean(df_1['avgRating'])
print('The average rating across all movies is:', average_rating)

# Calculate the average number of votes (m)
avg_num_votes = statistics.mean(df_1['numVotes'])  # Use the average number of votes for threshold
print('The average number of votes is:', avg_num_votes)

# Create a new column 'score' for the weighted average rating using 'avgRating' and 'numVotes'
df_1['score'] = df_1.apply(lambda row: calculate_weighted_score(row['avgRating'], row['numVotes'], average_rating, avg_num_votes), axis=1)

# Display the DataFrame with the calculated weighted score
df_1[['movieId', 'title', 'avgRating', 'numVotes', 'score']].head()
```

```
The average rating across all movies is: 3.7323364168313313
The average number of votes is: 2.3743169398907105
```

	movieId	title	avgRating	numVotes	score
0	1	Toy Story (1995)	3.833333	3	3.788714
3	2	Jumanji (1995)	3.750000	4	3.743421
7	3	Grumpier Old Men (1995)	2.500000	2	3.168895
9	5	Father of the Bride Part II (1995)	1.500000	2	2.711680
11	7	Sabrina (1995)	3.000000	1	3.515304

```
df_1
```

	movieId	title	genres	userId	rating	tag	numVotes	avgRating	score
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	336	4.0	pixar	3	3.833333	3.788714
3	2	Jumanji (1995)	Adventure|Children|Fantasy	62	4.0	fantasy	4	3.750000	3.743421
7	3	Grumpier Old Men (1995)	Comedy|Romance	289	2.5	moldy	2	2.500000	3.168895
9	5	Father of the Bride Part II (1995)	Comedy	474	1.5	pregnancy	2	1.500000	2.711680
11	7	Sabrina (1995)	Comedy|Romance	474	3.0	remake	1	3.000000	3.515304
...	...	...	...	...	...	...	...	...	...
3461	183611	Game Night (2018)	Action|Comedy|Crime|Horror	62	4.0	Comedy	3	4.000000	3.881749
3464	184471	Tomb Raider (2018)	Action|Adventure|Fantasy	62	3.5	adventure	3	3.500000	3.602644
3467	187593	Deadpool 2 (2018)	Action|Comedy|Sci-Fi	62	4.0	Josh Brolin	3	4.000000	3.881749
3470	187595	Solo: A Star Wars Story (2018)	Action|Adventure|Children|Sci-Fi	62	4.0	Emilia Clarke	2	4.000000	3.854716

Next steps: Generate code with df_1 | View recommended plots | New interactive sheet

Exercise 1 - Get the top 5 suggestions sorting by score in descending order

```
# TODO: filtering out the top 5 suggestions
# You can use `sort_values` to sort the DataFrame by the 'score' column in descending order

# filtering out the top 5 suggestions
top_5_movies = df_1.sort_values(by = 'score', ascending = False).head(5)[['title', 'genres', 'tag', 'score']]
print('Top 5 movies:')
top_5_movies
```

```
Top 5 movies:
```

	title	genres	tag	score
199	Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller	good dialogue	4.967226
1337	Fight Club (1999)	Action|Crime|Drama|Thriller	dark comedy	4.893394
604	2001: A Space Odyssey (1968)	Adventure|Drama|Sci-Fi	Hal	4.884498
998	Big Lebowski, The (1998)	Comedy|Crime	Coen Brothers	4.868802
164	Léon: The Professional (a.k.a. The Professiona...	Action|Crime|Drama|Thriller	assassin	4.852577

Next steps: Generate code with top_5_movies | View recommended plots | New interactive sheet

Content-based recommendation

Content-based filtering focuses on the attributes of items and the user's profile. It recommends movies to users based on features that closely match the user's profile. Movie A could be recommended because it matches the user's preferred genre, cast, and keywords. However, we might get limited diversity as it may not recommend items outside the user's known preferences, potentially limiting discovery of new types of items.

We want to compute the cosine similarity based on a number of features. Next, we will be creating a column features to gather the columns that we want to recommend to users. Calculation will be based on the type, genres, origin country, language, plot, summary, and cast.

```
# We will now create a new DataFrame that contains only the columns we need for our analysis.
df_2 = df_1[['movieId', 'title', 'userId', 'avgRating', 'numVotes', 'score', 'genres', 'tag']].copy()
df_2.reset_index(drop=True, inplace=True)
df_2
```

	movieId	title	userId	avgRating	numVotes	score	genres	tag
0	1	Toy Story (1995)	336	3.833333	3	3.788714	Adventure|Animation|Children|Comedy|Fantasy	pixar
1	2	Jumanji (1995)	62	3.750000	4	3.743421	Adventure|Children|Fantasy	fantasy
2	3	Grumpier Old Men (1995)	289	2.500000	2	3.168895	Comedy|Romance	moldy
3	5	Father of the Bride Part II (1995)	474	1.500000	2	2.711680	Comedy	pregnancy
4	7	Sabrina (1995)	474	3.000000	1	3.515304	Comedy|Romance	remake
...	...	...	...	...	...	...	...	...
1459	183611	Game Night (2018)	62	4.000000	3	3.881749	Action|Comedy|Crime|Horror	Comedy
1460	184471	Tomb Raider (2018)	62	3.500000	3	3.602644	Action|Adventure|Fantasy	adventure
1461	187593	Deadpool 2 (2018)	62	4.000000	3	3.881749	Action|Comedy|Sci-Fi	Josh Brolin
1462	187595	Solo: A Star Wars Story (2018)	62	4.000000	2	3.854716	Action|Adventure|Children|Sci-Fi	Emilia Clarke
1463	193565	Gintama: The Movie (2010)	184	3.500000	4	3.586541	Action|Animation|Comedy|Sci-Fi	anime

Next steps: Generate code with df_2 | View recommended plots | New interactive sheet

```
# Replace '|' with spaces in 'genres' and combine it with 'tag' using a space
df_2['features'] = df_2['genres'].str.replace('|', ' ') + ' ' + df_2['tag'].fillna('')

df_2
```

	movieId	title	userId	avgRating	numVotes	score	genres	tag	features
0	1	Toy Story (1995)	336	3.833333	3	3.788714	Adventure|Animation|Children|Comedy|Fantasy	pixar	Adventure Animation Children Comedy Fantasy pixar
1	2	Jumanji (1995)	62	3.750000	4	3.743421	Adventure|Children|Fantasy	fantasy	Adventure Children Fantasy fantasy
2	3	Grumpier Old Men (1995)	289	2.500000	2	3.168895	Comedy|Romance	moldy	Comedy Romance moldy
3	5	Father of the Bride Part II (1995)	474	1.500000	2	2.711680	Comedy	pregnancy	Comedy pregnancy
4	7	Sabrina	474	3.000000	1	3.515304	Comedy|Romance	remake	Comedy Romance

Next steps: Generate code with df_2 View recommended plots New interactive sheet

Next, let's vectorize the features column using TF-IDF vectorizer. The Term Frequency-Inverse Document Frequency(TF-IDF) vectorizer is used to transform text into numerical representations. It evaluates the importance of a word in a document relative to a collection of documents by considering both its frequency within a specific document (TF) and its rarity across all documents (IDF).

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english')

# Fit and transform the 'features' column to create TF-IDF vectors
X = vectorizer.fit_transform(df_2['features'])
```

Finally, let's get the cosine similarity and recommend items based on users' needs.

```
from sklearn.metrics.pairwise import cosine_similarity

# Calculate Cosine Similarity
similarity = cosine_similarity(X)

# Recommendation function (including itself as first result)
def recommendation(title, df, similarity, top_n=3):
    try:
        # Get the index of the movie that matches the title
        idx = df[df['title'] == title].index[0]
    except IndexError:
        print(f"Movie '{title}' not found in the dataset.")
        return

    # Get the similarity scores for the given movie
    sim_scores = list(enumerate(similarity[idx]))

    # Sort the movies based on similarity scores in descending order
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Print the top_n most similar movies (including itself)
    print(f"Movies similar to '{title}' (First movie is itself):")
    for i, (index, score) in enumerate(sim_scores[:top_n+1]):
        movie = df.iloc[index]
        print(f"{i}. {movie['title']} (Similarity Score: {score:.3f})")
        print(f"   Genres: {movie['genres']}")
        print(f"   Tag: {movie['tag']}\n")

# Test the recommendation function
recommendation("Toy Story (1995)", df_2, similarity)
```

```
Movies similar to 'Toy Story (1995)' (First movie is itself):
0. Toy Story (1995) (Similarity Score: 1.000)
   Genres: Adventure|Animation|Children|Comedy|Fantasy
   Tag: pixar
```

```
1. Bug's Life, A (1998) (Similarity Score: 0.939)
   Genres: Adventure|Animation|Children|Comedy
   Tag: Pixar

2. Toy Story 2 (1999) (Similarity Score: 0.675)
   Genres: Adventure|Animation|Children|Comedy|Fantasy
   Tag: animation

3. Sintel (2010) (Similarity Score: 0.583)
   Genres: Animation|Fantasy
   Tag: adventure
```

Exercise 2 - Check the recommendations for the movie 'Toy Story 2 (1999)'

```
recommendation("Toy Story 2 (1999)", df_2, similarity)
```

```
Movies similar to 'Toy Story 2 (1999)' (First movie is itself):
0. Toy Story 2 (1999) (Similarity Score: 1.000)
   Genres: Adventure|Animation|Children|Comedy|Fantasy
   Tag: animation

1. Croods, The (2013) (Similarity Score: 0.856)
   Genres: Adventure|Animation|Comedy
   Tag: animation

2. Sintel (2010) (Similarity Score: 0.853)
   Genres: Animation|Fantasy
   Tag: adventure

3. Invincible Iron Man, The (2007) (Similarity Score: 0.775)
   Genres: Animation
   Tag: animation
```

Collaborative filtering

Collaborative filtering is a recommendation system technique that makes automatic predictions about a user's preferences by collecting taste or preference information from many users. The assumption behind collaborative filtering is that if users agreed on certain items in the past, they are likely to agree on similar items in the future.

1. User-based Collaborative Filtering: This method identifies users with similar preferences and recommends items that similar users have liked. In other words, a user receives recommendations based on the preferences of users who have historically rated items similarly.
2. Item-based Collaborative Filtering: In this method, items similar to those the user has liked or rated highly in the past are recommended. The system identifies items that are frequently rated similarly across a user base and suggests items that share these patterns.

```
# Pivot user-item matrix from ratings
user_rating_matrix = rating_df.pivot(index="movieId", columns="userId", values="rating")

# fill na with 0
user_rating_matrix = user_rating_matrix.fillna(0)

user_rating_matrix.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

In this section, we will be using a NearestNeighbors classifier and using it based on the cosine similarity metric.

```
from sklearn.neighbors import NearestNeighbors

rec = NearestNeighbors(metric = 'cosine')
rec.fit(user_rating_matrix)
```

```
NearestNeighbors
NearestNeighbors(metric='cosine')
```

Finally, here is our function to get 5 recommended items based on a movie previously watched.

```
# Function to get movie recommendations based on a title
def get_recommendations(title):
    # Get movie details
    movie = df_2[df_2['title'] == title]

    if movie.empty:
        print(f"Movie '{title}' not found in dataset.")
        return None

    movie_id = int(movie['movieId'])

    # Get the index of the movie in the user-item matrix
    try:
        user_index = user_rating_matrix.index.get_loc(movie_id)
    except KeyError:
        print(f"Movie ID {movie_id} not found in the user rating matrix.")
        return None

    # Get the user ratings for the movie
    user_ratings = user_rating_matrix.iloc[user_index]

    # Reshape the ratings to be a single sample (1, -1)
    reshaped_df = user_ratings.values.reshape(1, -1)

    # Find the nearest neighbors (similar movies)
    distances, indices = rec.kneighbors(reshaped_df, n_neighbors=15)

    # Get the movieIds of the nearest neighbors (excluding the first, which is the queried movie itself)
    nearest_idx = user_rating_matrix.iloc[indices[0]].index[1:]

    # Get the movie details for the nearest neighbors
    nearest_neighbors = pd.DataFrame({'movieId': nearest_idx})
    result = pd.merge(nearest_neighbors, df_2, on='movieId', how='left')

    # Return the top recommendations
    return result[['title', 'avgRating', 'genres']].head()

# Test the recommendation function
get_recommendations('Toy Story (1995)')
```

	title	avgRating	genres
0	Toy Story 2 (1999)	3.125000	Adventure|Animation|Children|Comedy|Fantasy
1	Jurassic Park (1993)	4.500000	Action|Adventure|Sci-Fi|Thriller
2	Independence Day (a.k.a. ID4) (1996)	4.000000	Action|Adventure|Sci-Fi|Thriller
3	Star Wars: Episode IV - A New Hope (1977)	4.527778	Action|Adventure|Sci-Fi
4	Forrest Gump (1994)	3.666667	Comedy|Drama|Romance|War