Capstone II

Movie Recommendation Systems

Nikhil Valse

Department of Business, Long Island University

Gurpreet Singh

20th December 2022

Introduction:

With the advancements in information technologies, people are extensively utilizing the online digital platforms to consume all the entertainment content. This in turn creates huge amount of data related to attributes of the content. During this project, a movie dataset that contains metadata for all 45,000 movies listed in the Full MovieLens Dataset for the movies released on or before July 2017.

Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages. We will be analysing it visually, then we will be implementing the recommendation system based on it. You can check the Dataset on Kaggle from here.

Objective:

Recommender system is an algorithm which aims to provide most relevant information to a user by discovering the patterns or similarities in the records. Algorithm rates the items and shows the user items that they would rate highly. We are helping an online content distribution platform improve user experience and view time on the platform.

Data Understanding:

The data we are provided for this task has multiple files that we have to merge in order to access them at once. Also, they are of different types and nature so we are needed to transform some data such as ratings, overview into the form that the algorithm will be able to understand. There will be multiple files, one will be the metadata of the movies and another will be of the ratings of the users with timestamp and one more for the cast and crew involved. We mutate the variables accordingly to merge them to get useful insights from the data.

Dataset Variables:

Metadata file contains the following variables,

adult: Indicates if the movie is X-Rated or Adult.

belongs_to_collection: A stringified dictionary that gives information on the movie series the particular film belongs to.

budget: The budget of the movie in dollars.

genres: A stringified list of dictionaries that list out all the genres associated with the movie.

homepage: The Official Homepage of the move.

id: The ID of the move.

imdb_id: The IMDB ID of the movie.

original_language: The language in which the movie was originally shot in.

original_title: The original title of the movie.

overview: A brief blurb of the movie.

popularity: The Popularity Score assigned by TMDB. Learn more about how it's calculated here.

poster_path: The URL of the poster image.

production_companies: A stringified list of production companies involved with the making of the movie.

production_countries: A stringified list of countries where the movie was shot/produced in.

release_date: Theatrical Release Date of the movie.

revenue: The total revenue of the movie in dollars.

runtime: The runtime of the movie in minutes.

spoken_languages: A stringified list of spoken languages in the film.

status: The status of the movie (Released, To Be Released, Announced, etc.)

tagline: The tagline of the movie.

title: The Official Title of the movie.

video: Indicates if there is a video present of the movie with TMDB.

vote_average: The average rating of the movie.

vote_count: The number of votes by users, as counted by TMDB.

cast: A stringified list of dictionaries consisting of cast names

crew: A stringified list of dictionaries consisting of crew names and the function they performed.

The crew (credits) data file has just three columns which are cast crew and id, first two are the json objects.

The ratings dataset contains four columns,

userId: the ID of the user who rated the movie.

movieid: the ID of the movie

ratings: ratings given by each user (from 0 to 5)

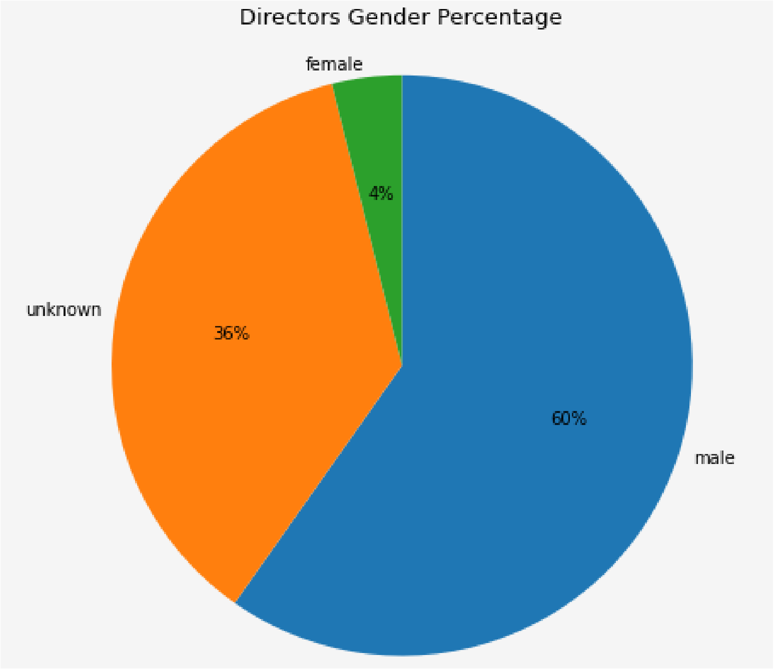
timstamp: The time the movie was rated.

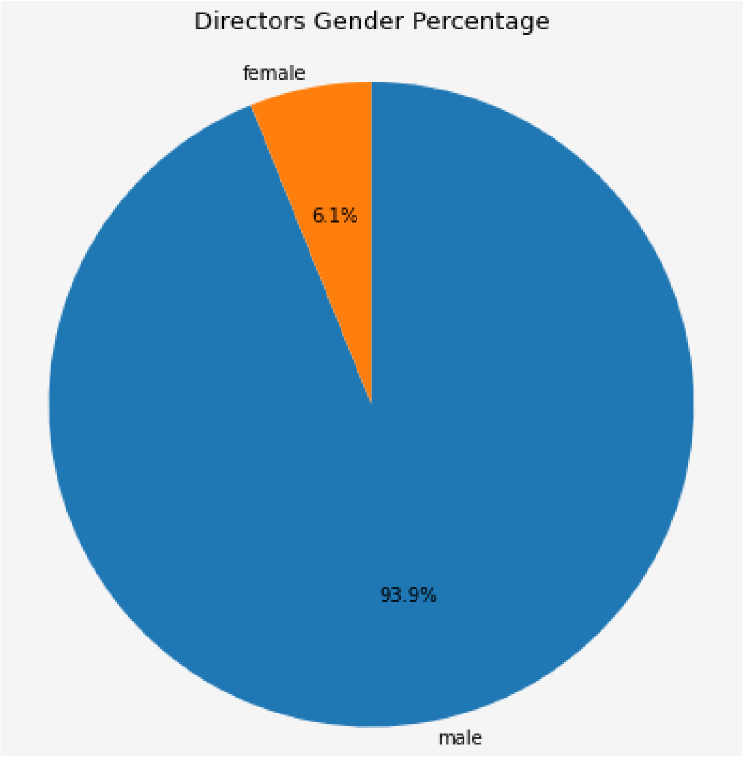
Data Exploration and Transformation:

As we move through data, we can see that there are lots of duplicate values which we delt with the bult-in functions in Pandas. Also, we dropped many columns which are not necessary for our visualizations, such as 'imdb_id', 'tagline', 'video', 'homepage', 'adult', 'revenue', 'budget' and ‘franchise’. Further more we had to drop records with unreleased status and the ones with runtime as zero.

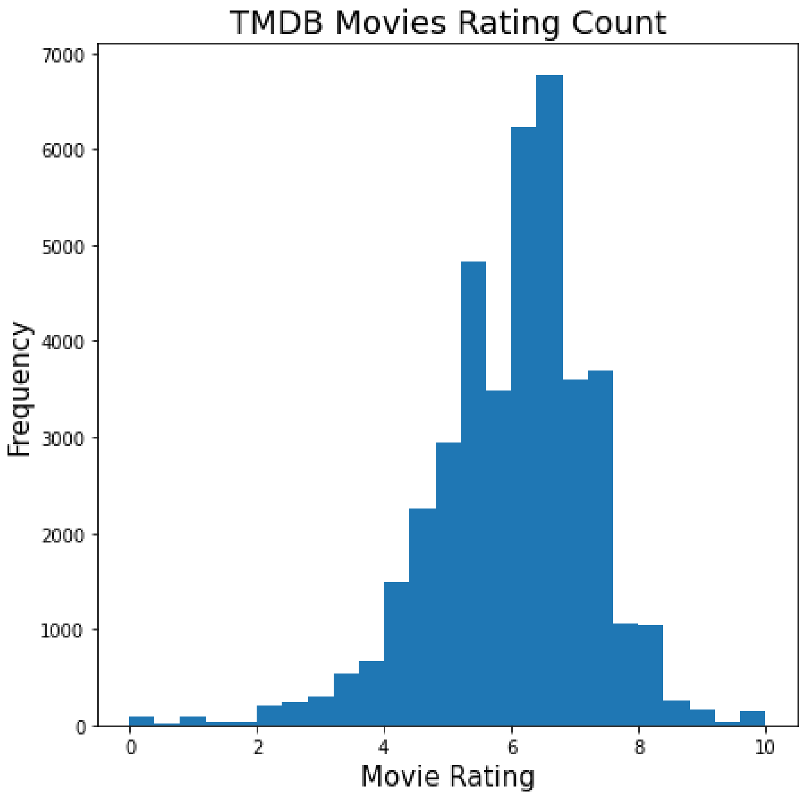
We changed popularity metric to numeric and extracted the year from the release date column with pandas to_datetime function. After that in order to get information about the directors from the crew’s data that we merged into the metadata, we used Abstract Syntax Tree to traverse json data and retrieve the name and gender.

We can see that after removing unknown gender, we see that only 6.1 percent directors are female and 93 percent are males.



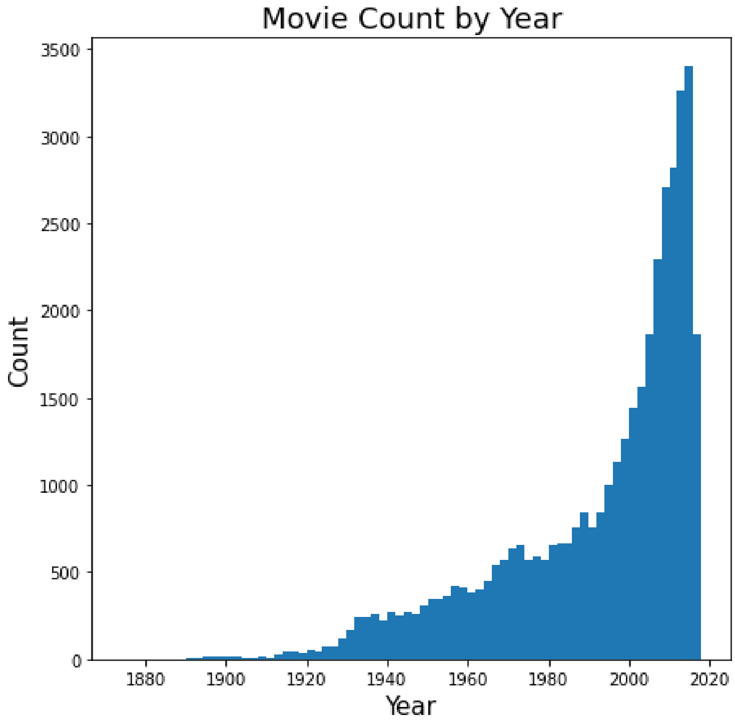


Moving onto the next feature which is vote average.



Movie ratings are little bit skewed to left with more typical values between 6 and 8.

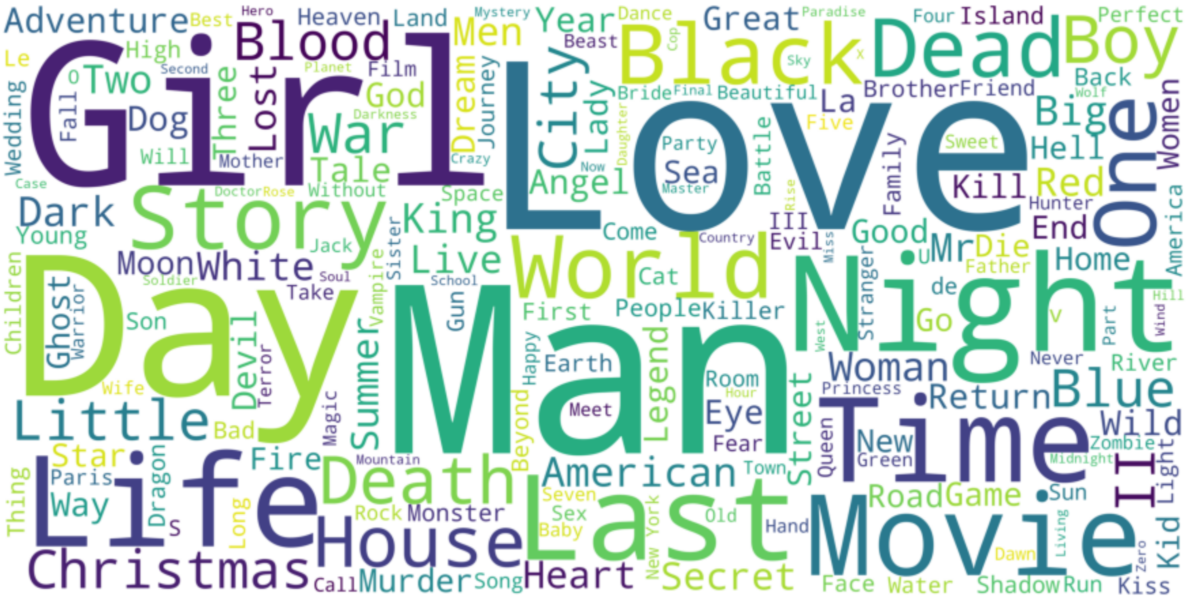
Coming up next, we explore the movie counts by year,



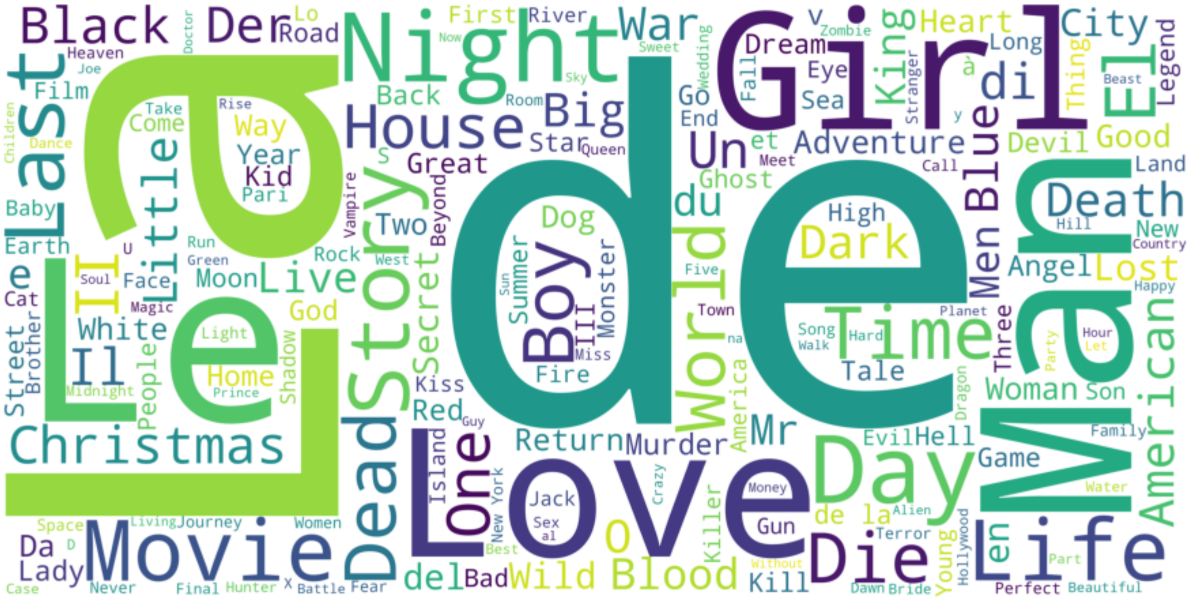
We can see that there is clear negative skewness in this histogram chart. Which means that movies count concentrate in recent years. There is rapid increase in 1990s and 2010s are the richest in cinema production.

Let’s create a cool chart displaying the frequent words in movie titles.

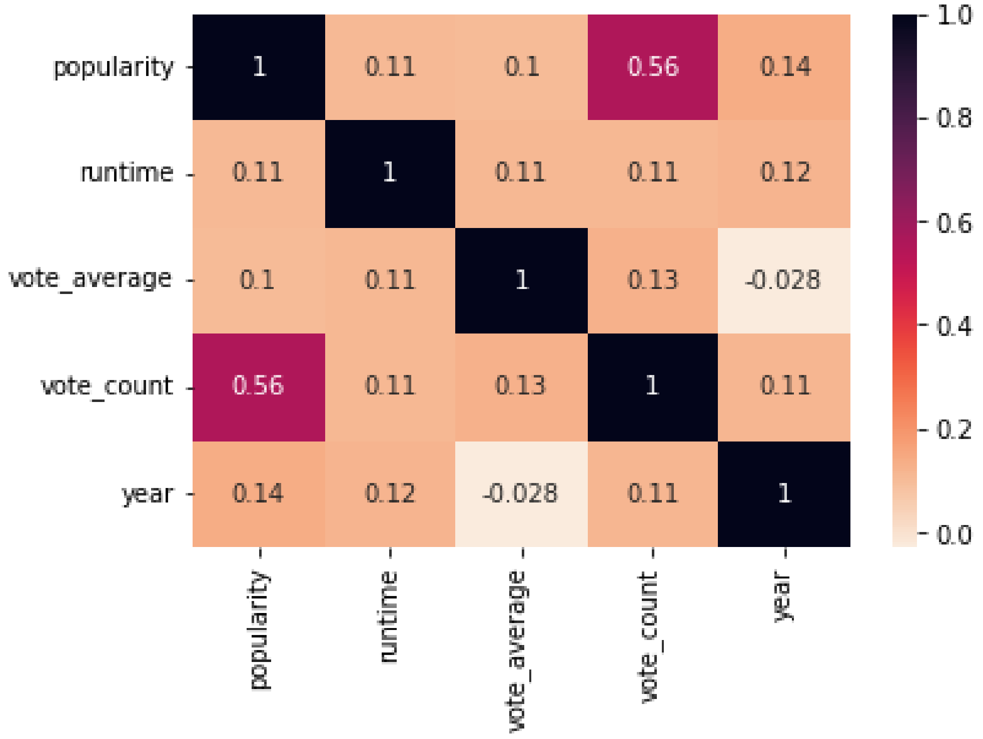
First we will create for titles in English,



This one is based on original languages,

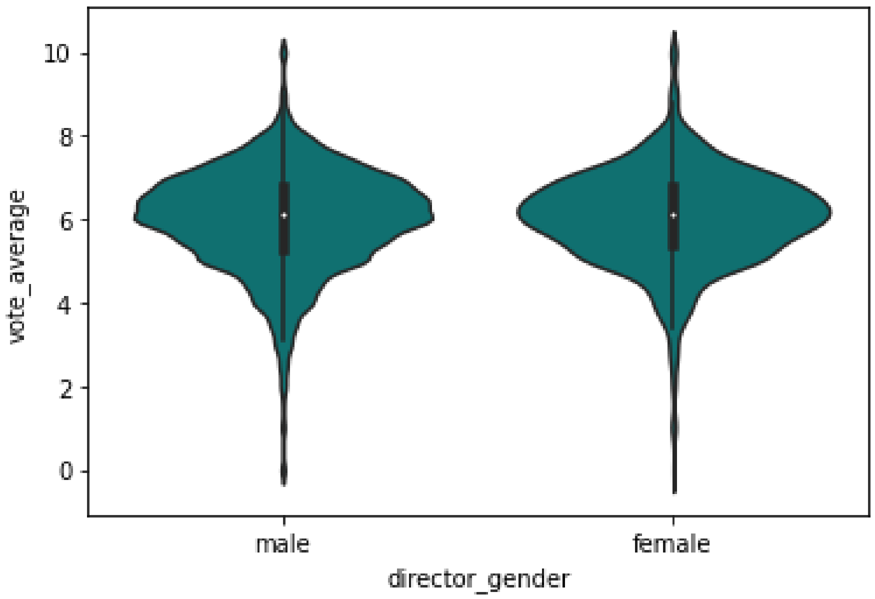


We now try to see correlation between different numerical variables in our data.



We can see moderate positive linear relationship between popularity and vote count (0.56) which suggests that the more votes or ratings a movie has, the more popular it is.

We now see the average votes vs gender,



We can observe that there is little to none difference between genders for average vote count.

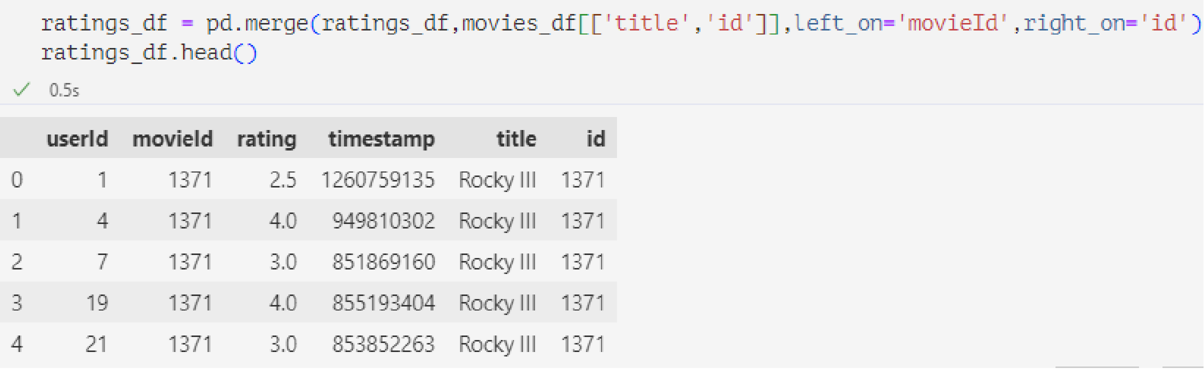
Conclusion on Data Visualizations:

Gender gap is prominent between women and men filmmakers contributing in the industry. We can’t state the clear reason which gives the space for another elaboration and usage of other techniques such as regression analysis.

Film’s ratings for men and women have almost the same average, implying that the lack of women in film industry isn't due to their lack of skill or talent.

Preprocessing for model:

In order to get our data ready for the Apriori algorithm, we need ratings data, which we are going to import and merge with the movies metadata. We are going to retain only the title from the movies data. Which looks like this,

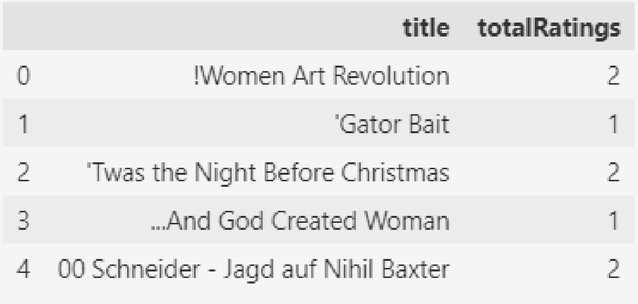


```
ratings_df = pd.merge(ratings_df,movies_df[['title','id']],left_on='movieId',right_on='id')
ratings_df.head()
```

✓ 0.5s

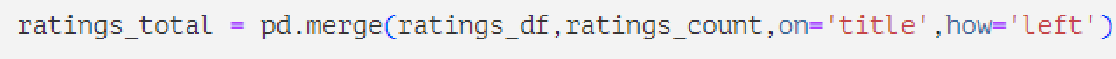
	userId	movieId	rating	timestamp	title	id
0	1	1371	2.5	1260759135	Rocky III	1371
1	4	1371	4.0	949810302	Rocky III	1371
2	7	1371	3.0	851869160	Rocky III	1371
3	19	1371	4.0	855193404	Rocky III	1371
4	21	1371	3.0	853852263	Rocky III	1371

We don’t need timestamp in this data frame so we drop it and then we count the rating count of each movie title, by using group by on title and count of rating.

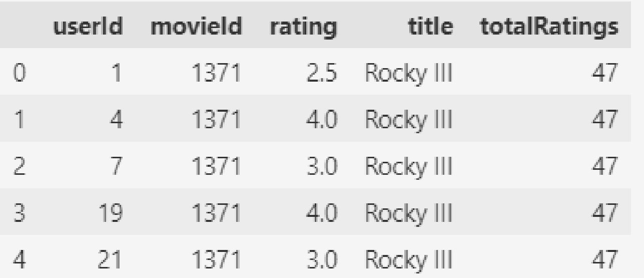


	title	totalRatings
0	!Women Art Revolution	2
1	'Gator Bait	1
2	'Twas the Night Before Christmas	2
3	...And God Created Woman	1
4	00 Schneider - Jagd auf Nihil Baxter	2

Again, we merge this to our main data frame,

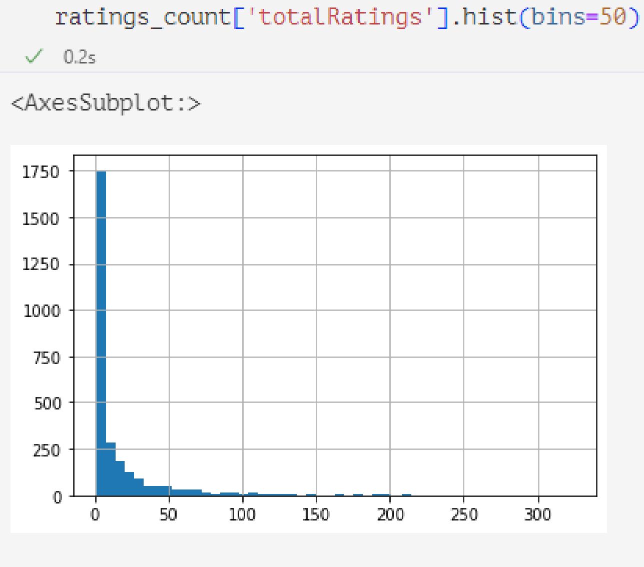


```
ratings_total = pd.merge(ratings_df,ratings_count,on='title',how='left')
```



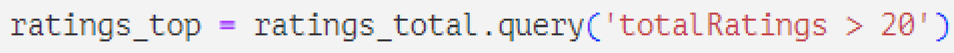
	userId	movieId	rating	title	totalRatings
0	1	1371	2.5	Rocky III	47
1	4	1371	4.0	Rocky III	47
2	7	1371	3.0	Rocky III	47
3	19	1371	4.0	Rocky III	47
4	21	1371	3.0	Rocky III	47

But our ratings count has too much difference in minimum number of ratings and maximum number of ratings. We can see it by this histogram,



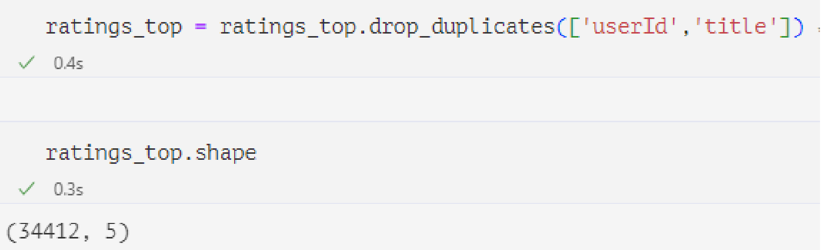
```
ratings_count['totalRatings'].hist(bins=50)
✓ 0.2s
<AxesSubplot:>
```

So, we should remove the movies except the ones which have at least 20 votes. So that they will be effective in algorithm prediction.



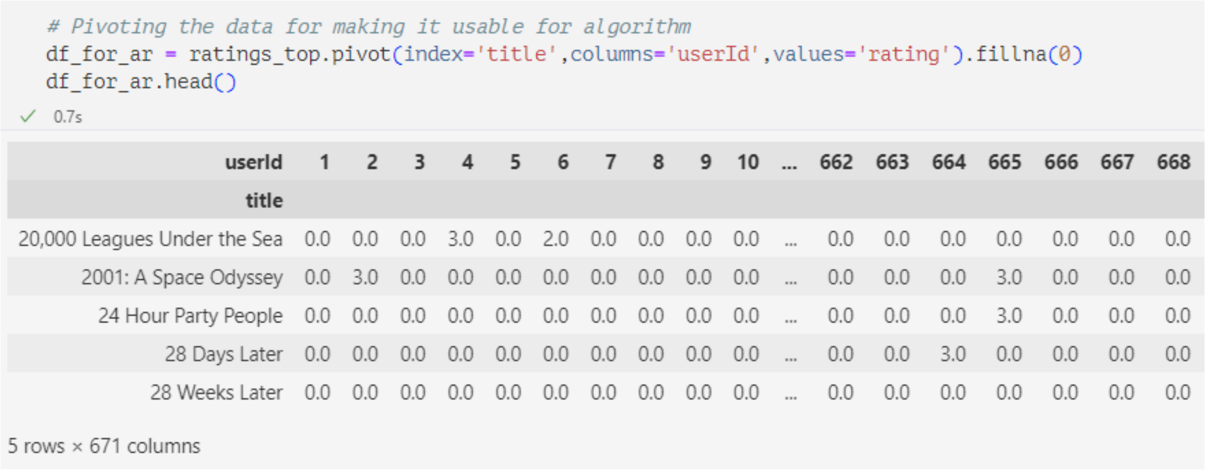
```
ratings_top = ratings_total.query('totalRatings > 20')
```

We'll now drop duplicates from it.



```
ratings_top = ratings_top.drop_duplicates(['userId','title'])
✓ 0.4s

ratings_top.shape
✓ 0.3s
(34412, 5)
```

Now the main part is to pivot the data frame to calculate the frequent item sets.



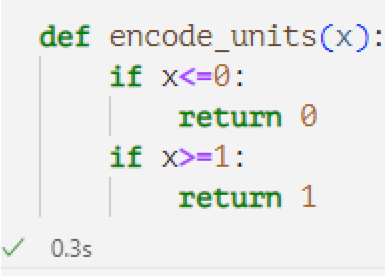
```
# Pivoting the data for making it usable for algorithm
df_for_ar = ratings_top.pivot(index='title',columns='userId',values='rating').fillna(0)
df_for_ar.head()
```

✓ 0.7s

userId	1	2	3	4	5	6	7	8	9	10	...	662	663	664	665	666	667	668
title																		
20,000 Leagues Under the Sea	0.0	0.0	0.0	3.0	0.0	2.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2001: A Space Odyssey	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	3.0	0.0	0.0	0.0
24 Hour Party People	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	3.0	0.0	0.0	0.0
28 Days Later	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	3.0	0.0	0.0	0.0	0.0
28 Weeks Later	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

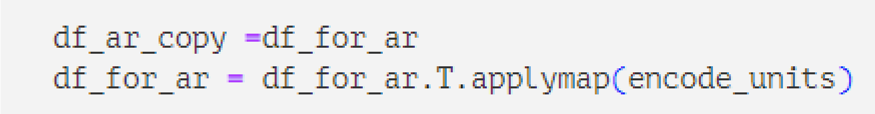
5 rows × 671 columns

Again, we need to put wherever the rating is given so as to feed it to the frequent item sets function.

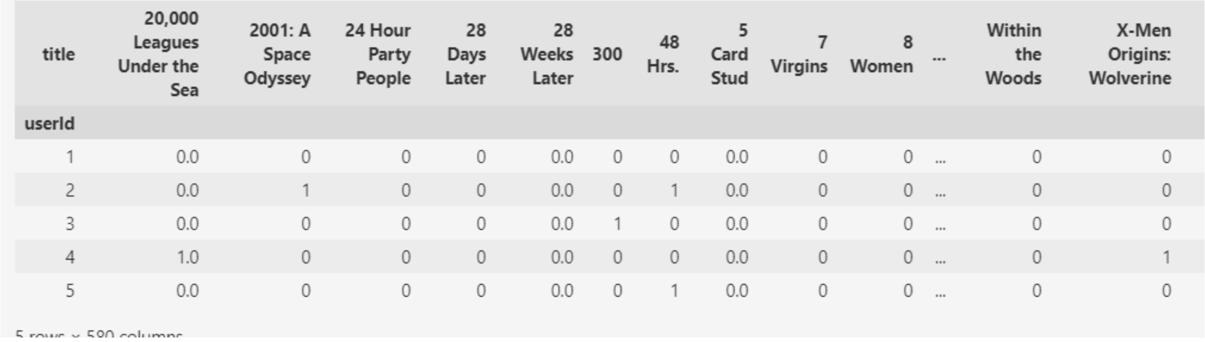


```
def encode_units(x):
    if x<=0:
        return 0
    if x>=1:
        return 1
```

✓ 0.3s



```
df_ar_copy =df_for_ar
df_for_ar = df_for_ar.T.applymap(encode_units)
```



title	20,000 Leagues Under the Sea	2001: A Space Odyssey	24 Hour Party People	28 Days Later	28 Weeks Later	300	48 Hrs.	5 Card Stud	7 Virgins	8 Women	...	Within the Woods	X-Men Origins: Wolverine
userId													
1	0.0	0	0	0	0.0	0	0	0.0	0	0	...	0	0
2	0.0	1	0	0	0.0	0	1	0.0	0	0	...	0	0
3	0.0	0	0	0	0.0	1	0	0.0	0	0	...	0	0
4	1.0	0	0	0	0.0	0	0	0.0	0	0	...	0	1
5	0.0	0	0	0	0.0	0	1	0.0	0	0	...	0	0

5 rows × 580 columns

So, this is our final pivot ready for finding the association rules.

Apriori Algorithm background:

Let’s now see what an association rule exactly looks like. It consists of an antecedent and a consequent, both of which are a list of items. Note that implication here is co-occurrence and not causality. For a given rule, itemset is the list of all the items in the antecedent and the consequent.

Various metrics are in place to help us understand the strength of association between these two. Let us go through them all.

1) Support

$$Support(\{X\} \to \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Total\ number\ of\ transactions}$$

Value of support helps us identify the rules worth considering for further analysis. For example, one might want to consider only the itemsets which occur at least 50 times out of a total of 10,000 transactions i.e. support = 0.005. If an itemset happens to have a very low support, we do not have enough information on the relationship between its items and hence no conclusions can be drawn from such a rule.

2) Confidence

This measure defines the likeliness of occurrence of consequent on the cart given that the cart already has the antecedents. That is to answer the question — of all the transactions containing say, {Captain Crunch}, how many also had {Milk} on them? We can say by common knowledge that {Captain Crunch} → {Milk} should be a high confidence rule. Technically, confidence is the conditional probability of occurrence of consequent given the antecedent.

$$Confidence(\{X\} \to \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Transactions\ containing\ X}$$

3) Lift

Lift controls for the *support* (frequency) of consequent while calculating the conditional probability of occurrence of {Y} given {X}. *Lift* is a very literal term given to this measure. Think of it as the *lift* that {X} provides to our confidence for having {Y} on the cart. To rephrase, *lift* is the rise in probability of having {Y} on the cart with the knowledge of {X} being present over the probability of having {Y} on the cart without any knowledge about presence of {X}. Mathematically,

$$Lift(\{X\} \rightarrow \{Y\}) = \frac{(Transactions\ containing\ both\ X\ and\ Y)/(Transactions\ containing\ X)}{Fraction\ of\ transactions\ containing\ Y}$$

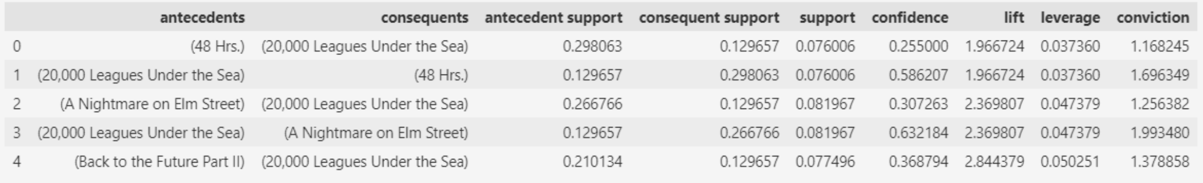
References:

https://towardsdatascience.com/association-rules-2-aa9a77241654

Recommendations:

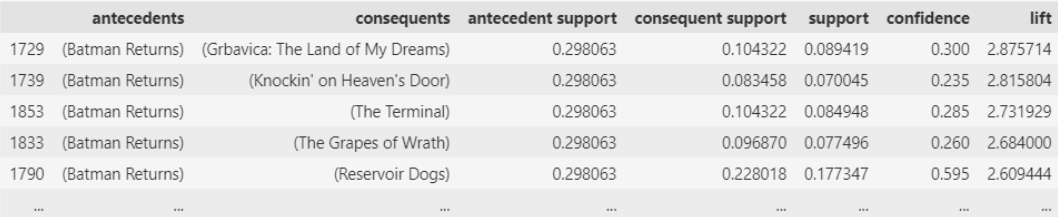
Now we can create the rules set off the frequent item sets with metric as a lift and its threshold as a 1 we can change this threshold to incorporate more items in the consequent list.

Our rules set looks like this, we now try to find the relevant results related to Batman Returns.



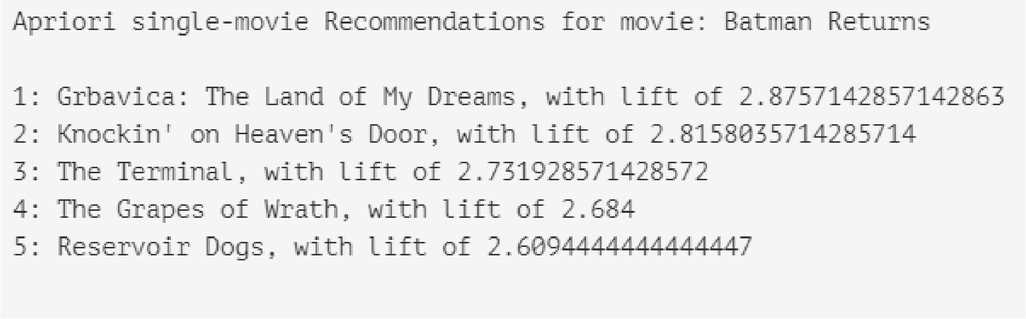
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(48 Hrs.)	(20,000 Leagues Under the Sea)	0.298063	0.129657	0.076006	0.255000	1.966724	0.037360	1.168245
1	(20,000 Leagues Under the Sea)	(48 Hrs.)	0.129657	0.298063	0.076006	0.586207	1.966724	0.037360	1.696349
2	(A Nightmare on Elm Street)	(20,000 Leagues Under the Sea)	0.266766	0.129657	0.081967	0.307263	2.369807	0.047379	1.256382
3	(20,000 Leagues Under the Sea)	(A Nightmare on Elm Street)	0.129657	0.266766	0.081967	0.632184	2.369807	0.047379	1.993480
4	(Back to the Future Part II)	(20,000 Leagues Under the Sea)	0.210134	0.129657	0.077496	0.368794	2.844379	0.050251	1.378858

We can now search for the top consequents which have Batman Returns as an antecedent with descending order of lift.



	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
1729	(Batman Returns)	(Grbavica: The Land of My Dreams)	0.298063	0.104322	0.089419	0.300	2.875714
1739	(Batman Returns)	(Knockin' on Heaven's Door)	0.298063	0.083458	0.070045	0.235	2.815804
1853	(Batman Returns)	(The Terminal)	0.298063	0.104322	0.084948	0.285	2.731929
1833	(Batman Returns)	(The Grapes of Wrath)	0.298063	0.096870	0.077496	0.260	2.684000
1790	(Batman Returns)	(Reservoir Dogs)	0.298063	0.228018	0.177347	0.595	2.609444
...	...	...	...	...	...	...	...

We get recommendations for ‘Batman Returns’ as,



```
Apriori single-movie Recommendations for movie: Batman Returns

1: Grbavica: The Land of My Dreams, with lift of 2.8757142857142863
2: Knockin' on Heaven's Door, with lift of 2.8158035714285714
3: The Terminal, with lift of 2.731928571428572
4: The Grapes of Wrath, with lift of 2.684
5: Reservoir Dogs, with lift of 2.6094444444444447
```

Recommendation Algorithm - Cosine Similarity:

Cosine similarity computes the L2-normalized dot product of vectors. That is, if and are row vectors, their cosine similarity

is defined as:

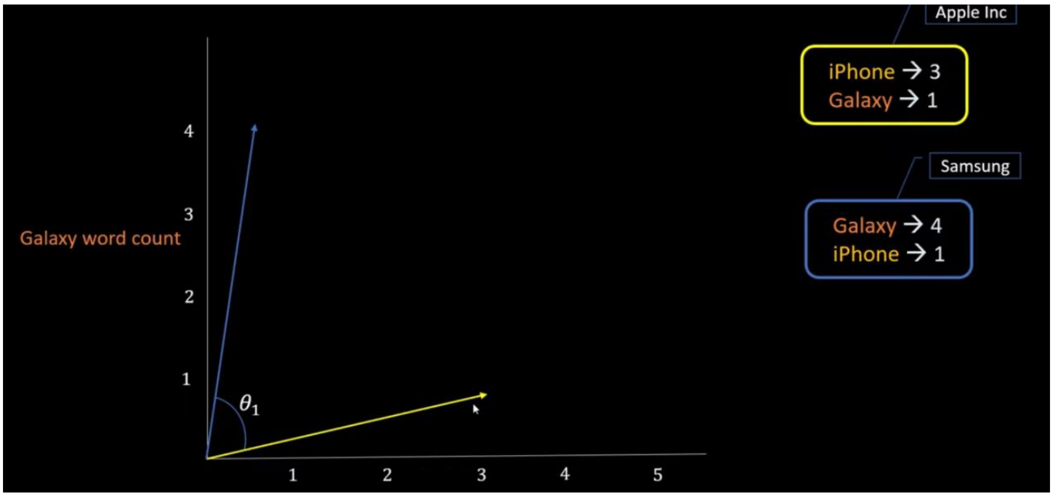
$$k(x, y) = \frac{x y^\top}{\|x\| \|y\|}$$

This is called cosine similarity, because Euclidean (L2) normalization projects the vectors onto the unit sphere, and their dot product is then the cosine of the angle between the points denoted by the vectors.

This kernel is a popular choice for computing the similarity of documents represented as tf-idf vectors. Cosine similarity accepts scipy.sparse matrices. (Note that the tf-idf functionality in sklearn.feature extraction.text can produce normalized vectors, in which case cosine similarity is equivalent to linear kernel, only slower.)

References:

- C.D. Manning, P. Raghavan and H. Schütze (2008). Introduction to Information Retrieval. Cambridge University Press. https://nlp.stanford.edu/IR-book/html/htmledition/the-vector-space-model-for-scoring-1.html
- https://scikit-learn.org/stable/modules/metrics.html



For example, if one document contains iPhone 3 times and galaxy 1 time and second document contains iPhone one time and Galaxy 4 times then we can pin point the co ordinates of both the documents as (3,1) and (1,4). We then draw the line from origin (0,0) to both the points and calculate the Cosine of the angle between the line. This is basically the cosine similarity coefficient which is similar to correlation coefficient but it ranges from 0 to 1.

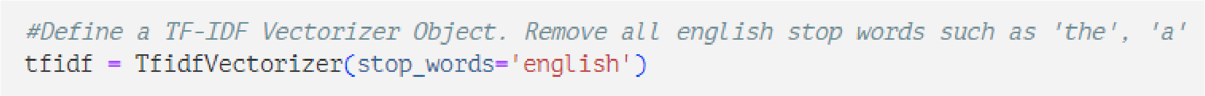
Reference:

https://www.youtube.com/watch?v=m_CooIRM3UI&t=208s

Preprocessing:

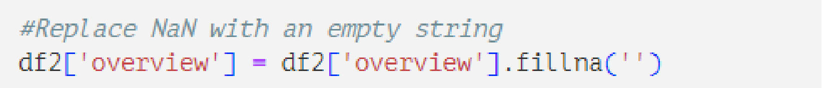
The main advantage of recommendations using Cosine Similarities is it doesn't require multiple iterations over the data. Here we are recommending the movies based on the movie description given in the overview column of the movies metadata file.

First, we are removing the stop words such as 'the', 'a' from the list of strings obtained.

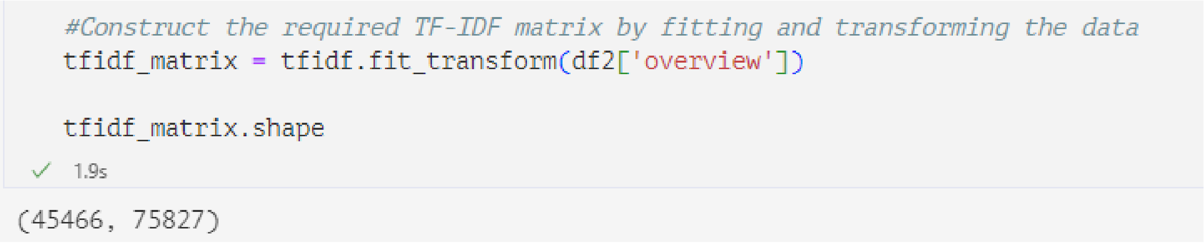


```
#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')
```

We are replacing the NaN's with empty string and then we create final transformed matrix from the data frame



```
#Replace NaN with an empty string
df2['overview'] = df2['overview'].fillna('')
```



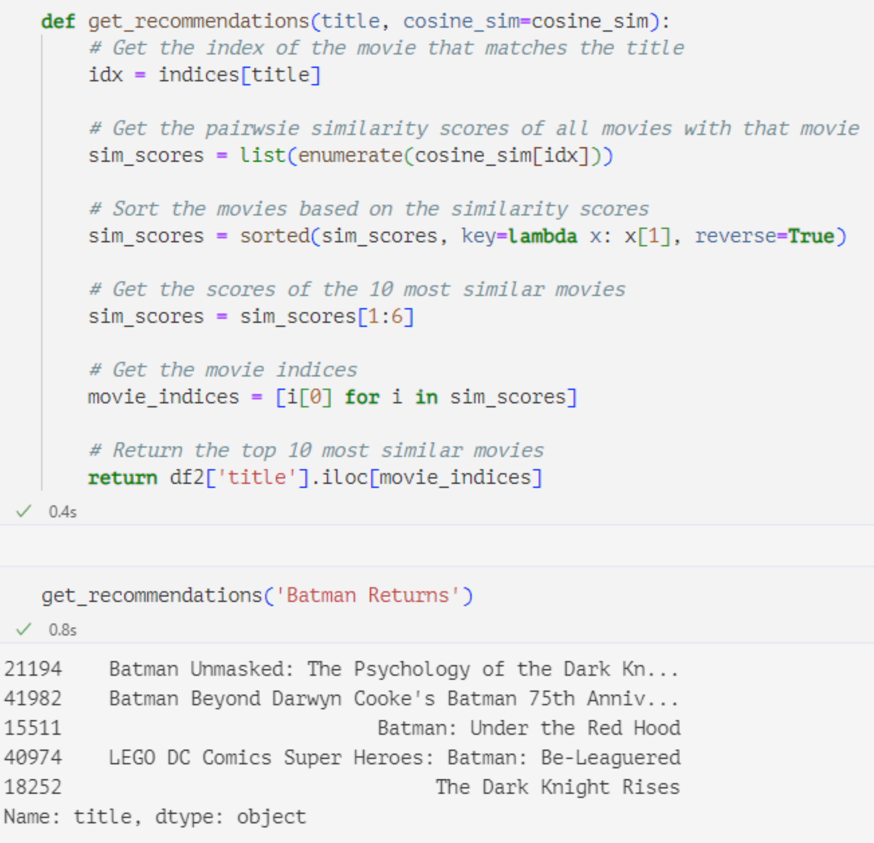
```
#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(df2['overview'])

tfidf_matrix.shape
```

✓ 1.9s

```
(45466, 75827)
```

For this algorithm we are computing the frequency of each word occurring in the overview of each movie and then of frequencies of **mutual** words are similar in two movie records overviews, the cosine similarity is higher between two.

After sorting the similarity scores for ‘Batman Returns’ in descending order, we get recommendations as



```
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

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

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:6]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return df2['title'].iloc[movie_indices]
```

✓ 0.4s

```
get_recommendations('Batman Returns')
```

✓ 0.8s

```
21194    Batman Unmasked: The Psychology of the Dark Kn...
41982    Batman Beyond Darwyn Cooke's Batman 75th Anniv...
15511                          Batman: Under the Red Hood
40974    LEGO DC Comics Super Heroes: Batman: Be-Leagured
18252                               The Dark Knight Rises
Name: title, dtype: object
```

Which look more relevant than the Apriori algorithm which was based on the ratings data. The Cosine similarity algorithm is more reliable in this regard.

Conclusion for Algorithms:

As we explored various techniques for finding the association between the two records. One of which was Apriori algorithm in which we tried to simulate user ratings as a transaction done on a grocery store. Which in my opinion is not the best analogy, but in that case, we get results based on the public opinions. That means if the people are interested in the similar sets of movies, you will get those recommendations regardless of the other similarity metrics.

In the other method which is Cosine similarity, we took feature as overview (description) of the movie, which doesn't care about that if people would watch the relevant movies as stated in the description of the movie, it just matches depending on the frequency match of the terms included in the description of the movie.

Both algorithms are good and bad at one or another scenario, Cosine is efficient in terms of runtime, while Apriori is effective in deep traversing the associations in depth.

Github Link:

https://github.com/nvalse81/Capstone-2-620.git

Thank You!