ABSTRACT

The project aims to develop a machine learning project for recommending a movie to a user. A movie recommendation system is used to predict the **rating** or **preference** that a user would give to an item. Almost every major tech company has applied them in some form or the other: Amazon uses it to suggest products to customers, YouTube uses it to decide which video to play next on autoplay, and Facebook uses it to recommend pages to like and people to follow. Moreover, companies like Netflix and Spotify depend highly on the effectiveness of their recommendation engines for their business and success.

INTRODUCTION

The rapid growth of data collection has led to a new era of information. Data is being used to create more efficient systems and this is where Recommendation Systems come into play. Recommendation Systems are a type of information filtering systems as they improve the quality of search results and provide items that are more relevant to the search item or are related to the search history of the user.

Types of recommendation systems:

1.) **Demographic Filtering:** They offer generalized recommendations to every user, based on movie popularity and/or genre. The System recommends the same movies to users with similar demographic features. Since each user is different , this approach is considered to be too simple. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience.

2.) **Content Based Filtering**: They suggest similar items based on a particular item. This system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations. The general idea behind these recommender systems is that if a person likes a particular item, he or she will also like an item that is similar to it.

3.) **Collaborative Filtering**: This system matches persons with similar interests and provides recommendations based on this matching. Collaborative filters do not require item metadata like its content-based counterparts.

The model that we have made focuses on Demographic Filtering and Content Based Filtering

Working and Functionality

Demographic Filtering

1.) **import pandas as pd**
```
import numpy as np
f1=pd.read_csv('tmdb_5000_credits.csv')
f2=pd.read_csv('tmdb_5000_movies.csv')
```

The first dataset contains the following features:-

- movie_id - A unique identifier for each movie.
- cast - The name of lead and supporting actors.
- crew - The name of Director, Editor, Composer, Writer etc.

The second dataset has the following features:-

- budget - The budget in which the movie was made.
- genre - The genre of the movie, Action, Comedy ,Thriller etc.
- homepage - A link to the homepage of the movie.
- id - This is in fact the movie_id as in the first dataset.
- keywords - The keywords or tags related to the movie.
- original_language - The language in which the movie was made.
- original_title - The title of the movie before translation or adaptation.
- overview - A brief description of the movie.
- popularity - A numeric quantity specifying the movie popularity.
- production_companies - The production house of the movie.
- production_countries - The country in which it was produced.
- release_date - The date on which it was released.
- revenue - The worldwide revenue generated by the movie.
- runtime - The running time of the movie in minutes.
- status - "Released" or "Rumored".
- tagline - Movie's tagline.
- title - Title of the movie.
- vote_average - average ratings the movie received.
- vote_count - the count of votes received.

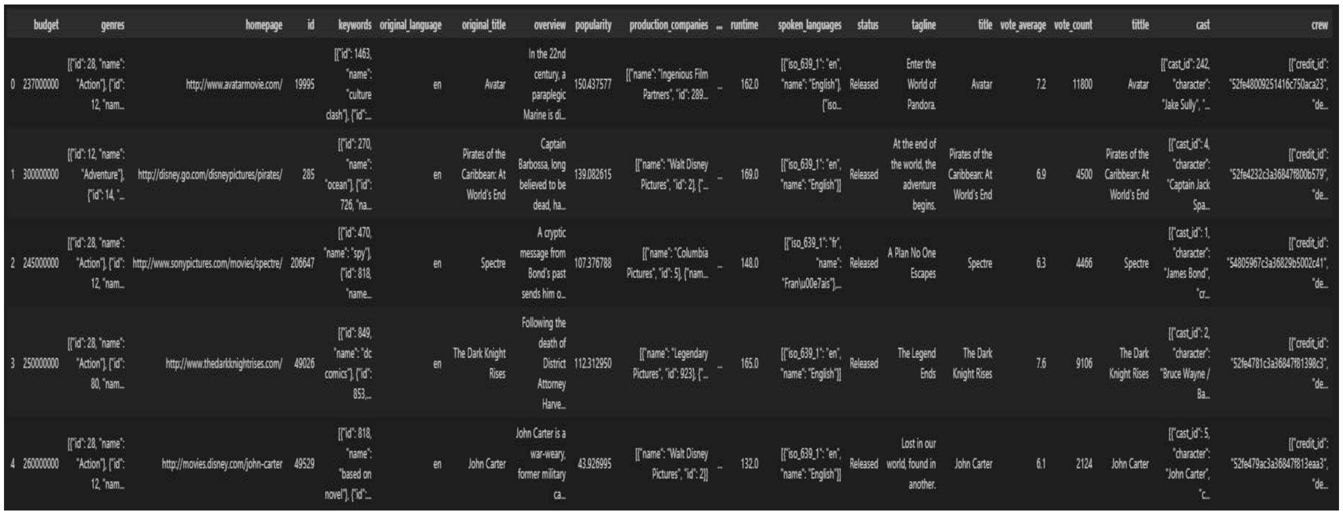
Let's join the two dataset on the 'id' column

2.) f1.columns = ['id','tittle','cast','crew']

f2= f2.merge(f1,on='id')

Just a peak at our data.

3.) f2.head(5)



	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_companies	...	runtime	spoken_languages	status	tagline	title	vote_average	vote_count	tittle	cast	crew
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	[{"name": "Ingenious Film Partners", "id": 289...	...	162.0	[{"iso_639_1": "en", "name": "English"}, {"iso...	Released	Enter the World of Pandora.	Avatar	7.2	11800	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
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	Captain Barbossa, long believed to be dead, ha...	139.082615	[{"name": "Walt Disney Pictures", "id": 2}, {"...	...	169.0	[{"iso_639_1": "en", "name": "English"}]	Released	At the end of the world, the adventure begins.	Pirates of the Caribbean: At World's End	6.9	4500	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	[{"name": "Columbia Pictures", "id": 5}, {"nam...	...	148.0	[{"iso_639_1": "fr", "name": "Fran\u00e7ais"},...	Released	A Plan No One Escapes	Spectre	6.3	4466	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	en	The Dark Knight Rises	Following the death of District Attorney Harve...	112.312950	[{"name": "Legendary Pictures", "id": 923}, {"...	...	165.0	[{"iso_639_1": "en", "name": "English"}]	Released	The Legend Ends	The Dark Knight Rises	7.6	9106	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	en	John Carter	John Carter is a war-weary, former military ca...	43.926995	[{"name": "Walt Disney Pictures", "id": 2}]	...	132.0	[{"iso_639_1": "en", "name": "English"}]	Released	Lost in our world, found in another.	John Carter	6.1	2124	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

Demographic Filtering -

Before getting started with this -

- we need a metric to score or rate a movie
- Calculate the score for every movie
- Sort the scores and recommend the best rated movie to the users.

We can use the average ratings of the movie as the score but using this won't be fair enough since a movie with 8.9 average rating and only 3 votes cannot be considered better than the movie with 7.8 as as average rating but 40 votes. So, I'll be using IMDB's weighted rating (wr) which is given as :-

$$\text{Weighted Rating (WR)} = \left(\frac{v}{v+m} . R\right) + \left(\frac{m}{v+m} . C\right)$$

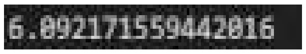
where,

- **v is the number of votes for the movie;**
- **m is the minimum votes required to be listed in the chart;**
- **R is the average rating of the movie; And**
- **C is the mean vote across the whole report**

We already have v(vote_count) and R (vote_average) and C can be calculated as

4.) a= f2['vote_average'].mean()

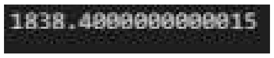
    a



```
6.092171559442016
```

So, the mean rating for all the movies is approx 6 on a scale of 10.The next step is to determine an appropriate value for m, the minimum votes required to be listed in the chart. We will use 90th percentile as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 90% of the movies in the list.

5.) b= f2['vote_count'].quantile(0.9)
    b



```
1838.4000000000015
```

Now, we can filter out the movies that qualify for the chart

6.) q_movies = f2.copy().loc[f2['vote_count'] >= m]
    q_movies.shape

```
(481, 23)
```

We see that there are 481 movies which qualify to be in this list. Now, we need to calculate our metric for each qualified movie. To do this, we will define a function, **weighted_rating()** and define a new feature **score**, of which we'll calculate the value by applying this function to our DataFrame of qualified movies:

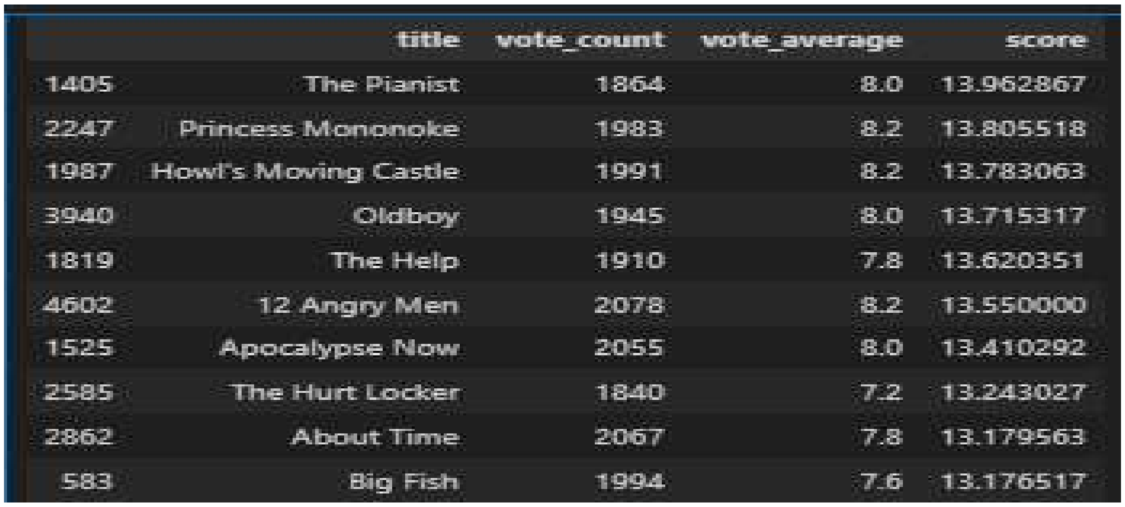
7.) **def weighted_rating(x, m=a, C=b):**
```
    v = x['vote_count']
    R = x['vote_average']
    # Calculation based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)
```

8.) **# Define a new feature 'score' and calculate its value with `weighted_rating()`**
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)

Finally, let's sort the DataFrame based on the score feature and output the title, vote count, vote average and weighted rating or score of the top 10 movies.

9.) **#Sort movies based on score calculated above**
q_movies = q_movies.sort_values('score', ascending=False)

#Print the top 15 movies
q_movies[['title', 'vote_count', 'vote_average', 'score']].head(10)

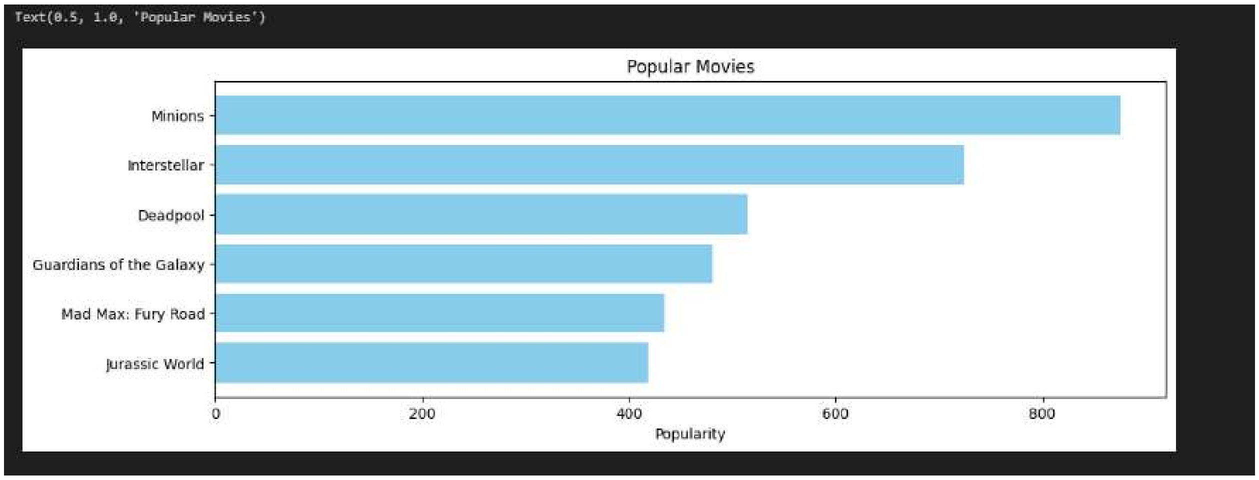


	title	vote_count	vote_average	score
1405	The Pianist	1864	8.0	13.962867
2247	Princess Mononoke	1983	8.2	13.805518
1987	Howl's Moving Castle	1991	8.2	13.783063
3940	Oldboy	1945	8.0	13.715317
1819	The Help	1910	7.8	13.620351
4602	12 Angry Men	2078	8.2	13.550000
1525	Apocalypse Now	2055	8.0	13.410292
2585	The Hurt Locker	1840	7.2	13.243027
2862	About Time	2067	7.8	13.179563
583	Big Fish	1994	7.6	13.176517

Hurray! We have made our first(though very basic) recommender. Under the Trending Now tab of these systems we find movies that are very popular and they can just be obtained by sorting the dataset by the popularity column.

10.)

```
pop= f2.sort_values('popularity', ascending=False)
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))
plt.barh(pop['title'].head(6),pop['popularity'].head(6), align='center',
color='skyblue')
plt.gca().invert_yaxis()
plt.xlabel("Popularity")
plt.title("Popular Movies")
```

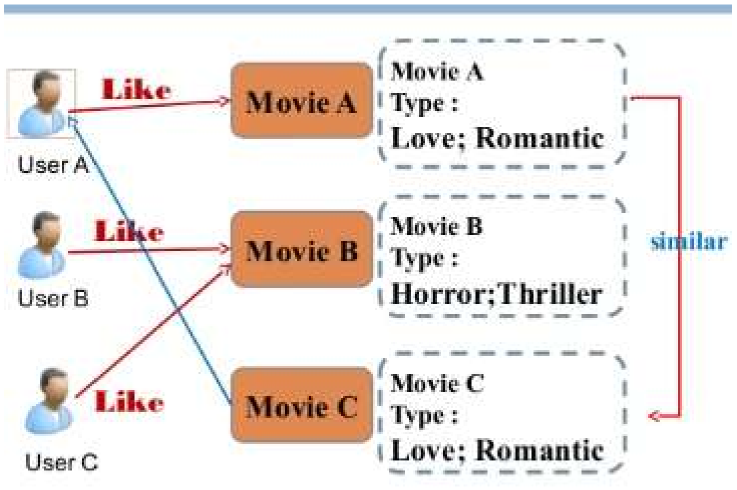


```
Text(0.5, 1.0, 'Popular Movies')
```

Now something to keep in mind is that these demographic recommender provide a general chart of recommended movies to all the users. They are not sensitive to the interests and tastes of a particular user. This is when we move on to a more refined system- Content Based Filtering.

Content Based Filtering

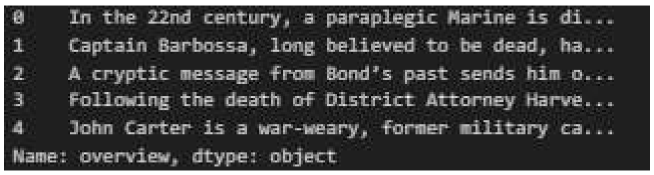
In this recommender system the content of the movie (overview, cast, crew, keyword, tagline etc) is used to find its similarity with other movies. Then the movies that are most likely to be similar are recommended.



Plot description based Recommender

We will compute pairwise similarity scores for all movies based on their plot descriptions and recommend movies based on that similarity score. The plot description is given in the **overview** feature of our dataset. Let's take a look at the data.

11.) f2['overview'].head(5)



```
0    In the 22nd century, a paraplegic Marine is di...
1    Captain Barbossa, long believed to be dead, ha...
2    A cryptic message from Bond's past sends him o...
3    Following the death of District Attorney Harve...
4    John Carter is a war-weary, former military ca...
Name: overview, dtype: object
```

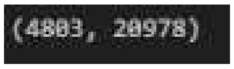
For any of you who has done even a bit of text processing before knows we need to convert the word vector of each overview. Now we'll compute Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each overview.

Now if you are wondering what is term frequency , it is the relative frequency of a word in a document and is given as **(term instances/total instances)**. Inverse Document Frequency is the relative count of documents containing the term is given as **log(number of documents/documents with term)** The overall importance of each word to the documents in which they appear is equal to **TF * IDF**

This will give you a matrix where each column represents a word in the overview vocabulary (all the words that appear in at least one document) and each row represents a movie, as before.This is done to reduce the importance of words that occur frequently in plot overviews and therefore, their significance in computing the final similarity score.

Fortunately, scikit-learn gives you a built-in TfIdfVectorizer class that produces the TF-IDF matrix in a couple of lines. That's great, isn't it?

12.)
```
#import TfIdfVectorizer from scikit-learn

import sklearn

from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'

tfidf = TfidfVectorizer(stop_words='english')

#Replace NaN with an empty string

f2['overview'] = f2['overview'].fillna('')

#Construct the required TF-IDF matrix by fitting and transforming the data

tfidf_matrix = tfidf.fit_transform(f2['overview'])

#Output the shape of tfidf_matrix

tfidf_matrix.shape
```



```
(4803, 20978)
```

We see that over 20,000 different words were used to describe the 4800 movies in our dataset.

With this matrix in hand, we can now compute a similarity score. There are several candidates for this; such as the euclidean, the Pearson and the cosine similarity scores. There is no right answer to which score is the best. Different scores work well in different scenarios and it is often a good idea to experiment with different metrics.

We will be using the cosine similarity to calculate a numeric quantity that denotes the similarity between two movies. We use the cosine similarity score since it is independent of magnitude and is relatively easy and fast to calculate. Mathematically, it is defined as follows:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

Since we have used the TF-IDF vectorizer, calculating the dot product will directly give us the cosine similarity score. Therefore, we will use sklearn's linear_kernel() instead of cosine_similarities() since it is faster.

```
13.) # Import linear_kernel

     from sklearn.metrics.pairwise import linear_kernel

     # Compute the cosine similarity matrix

     cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

We are going to define a function that takes in a movie title as an input and outputs a list of the 10 most similar movies. Firstly, for this, we need a reverse mapping of movie titles and DataFrame indices. In other words, we need a mechanism to identify the index of a movie in our metadata DataFrame, given its title.

14.) #Construct a reverse map of indices and movie titles

```
indices = pd.Series(df2.index, index=df2['title']).drop_duplicates()
```

We are now in a good position to define our recommendation function. These are the following steps we'll follow :-

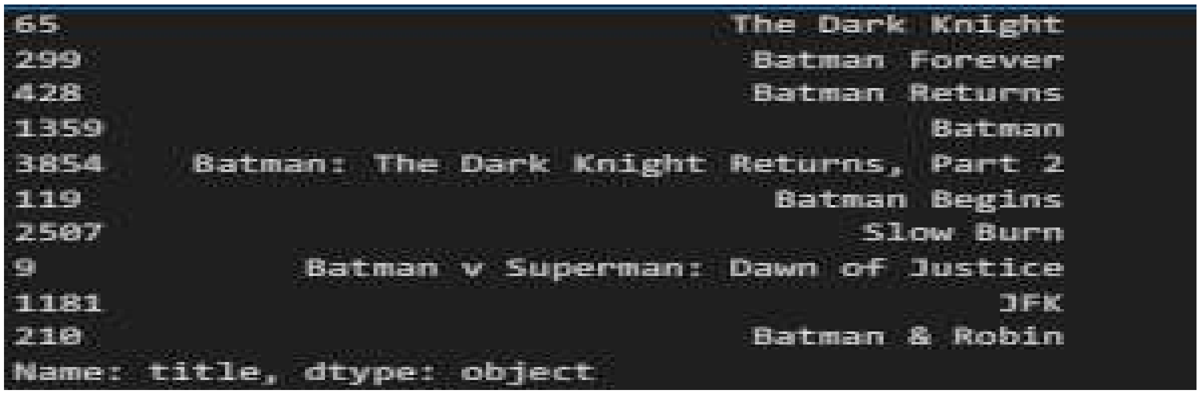
- Get the index of the movie given its title.
- Get the list of cosine similarity scores for that particular movie with all movies. Convert it into a list of tuples where the first element is its position and the second is the similarity score.
- Sort the aforementioned list of tuples based on the similarity scores; that is, the second element.
- Get the top 10 elements of this list. Ignore the first element as it refers to self (the movie most similar to a particular movie is the movie itself).
- Return the titles corresponding to the indices of the top elements.

15.) # Function that takes in movie title as input and outputs most similar movies

```
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]))
```

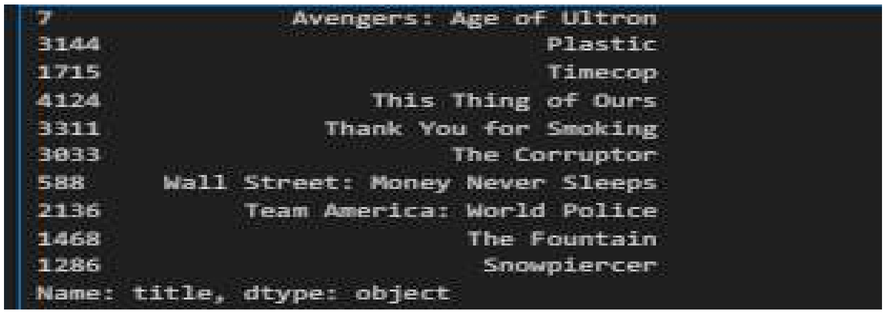
```
        # Sort the movies based on the similarity scores

        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[1:11]

         # Get the movie indices

          movie_indices = [i[0] for i in sim_scores]

          # Return the top 10 most similar movies

          return f2['title'].iloc[movie_indices]
```

16.) get_recommendations('The Dark Knight Rises')



```
65                                   The Dark Knight
299                                   Batman Forever
428                                   Batman Returns
1359                                          Batman
3854    Batman: The Dark Knight Returns, Part 2
119                                    Batman Begins
2507                                       Slow Burn
9             Batman v Superman: Dawn of Justice
1181                                             JFK
210                                   Batman & Robin
Name: title, dtype: object
```

17.)get_recommendations('The Avengers')



```
7                Avengers: Age of Ultron
3144                             Plastic
1715                             Timecop
4124                  This Thing of Ours
3311               Thank You for Smoking
3033                       The Corruptor
588        Wall Street: Money Never Sleeps
2136          Team America: World Police
1468                        The Fountain
1286                         Snowpiercer
Name: title, dtype: object
```

While our system has done a decent job of finding movies with similar plot descriptions, the quality of recommendations is not that great. "The Dark Knight Rises" returns all Batman movies while it is more likely that the people who liked that movie are more inclined to enjoy other Christopher Nolan movies. This is something that cannot be captured by the present system.

Credits, Genres and Keywords Based Recommender¶

It goes without saying that the quality of our recommender would be increased with the usage of better metadata. That is exactly what we are going to do in this section. We are going to build a recommender based on the following metadata: the 3 top actors, the director, related genres and the movie plot keywords.

From the cast, crew and keywords features, we need to extract the three most important actors, the director and the keywords associated with that movie. Right now, our data is present in the form of "stringified" lists , we need to convert it into a safe and usable structure

18.) # Parse the stringified features into their corresponding python objects

```
from ast import literal_eval

features = ['cast', 'crew', 'keywords', 'genres']

for feature in features:

f2[feature] = f2[feature].apply(literal_eval)
```

Next, we'll write functions that will help us to extract the required information from each feature.

19.) # Get the director's name from the crew feature. If director is not listed, return NaN

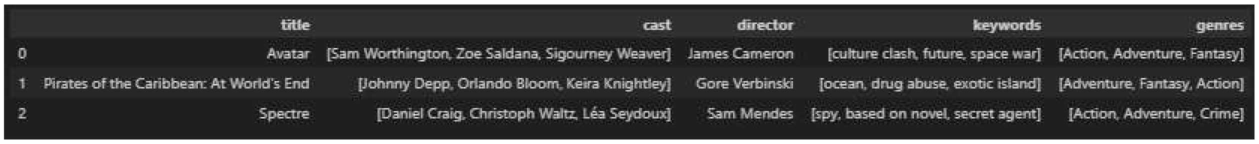
```
    def get_director(x):

    for i in x:

        if i['job'] == 'Director':

            return i['name']

    return np.nan
```

20.) # Returns the list top 3 elements or entire list; whichever is more.

```
def get_list(x):

    if isinstance(x, list):

        names = [i['name'] for i in x]

 #Check if more than 3 elements exist. If yes, return only first three. If no,
return entire list.

    if len(names) > 3:

        names = names[:3]

    return names

  return []
```

```
21.) # Define new director, cast, genres and keywords features that are in a suitable form.

f2['director'] = f2['crew'].apply(get_director)

features = ['cast', 'keywords', 'genres']

for feature in features:

    f2[feature] = f2[feature].apply(get_list)
```

```
22.) # Print the new features of the first 3 films

f2[['title', 'cast', 'director', 'keywords', 'genres']].head(3)
```



	title	cast	director	keywords	genres
0	Avatar	[Sam Worthington, Zoe Saldana, Sigourney Weaver]	James Cameron	[culture clash, future, space war]	[Action, Adventure, Fantasy]
1	Pirates of the Caribbean: At World's End	[Johnny Depp, Orlando Bloom, Keira Knightley]	Gore Verbinski	[ocean, drug abuse, exotic island]	[Adventure, Fantasy, Action]
2	Spectre	[Daniel Craig, Christoph Waltz, Léa Seydoux]	Sam Mendes	[spy, based on novel, secret agent]	[Action, Adventure, Crime]

The next step would be to convert the names and keyword instances into lowercase and strip all the spaces between them. This is done so that our vectorizer doesn't count the Johnny of "Johnny Depp" and "Johnny Galecki" as the same.

```
23.) # Function to convert all strings to lower case and strip names of spaces

def clean_data(x):

    if isinstance(x, list):

        return [str.lower(i.replace(" ", "")) for i in x]
    else:

        #Check if director exists. If not, return empty string

        if isinstance(x, str):

            return str.lower(x.replace(" ", ""))

        else:

            return ''
```

24.) # Apply clean_data function to your features.

```
features = ['cast', 'keywords', 'director', 'genres']

for feature in features:

    f2[feature] = f2[feature].apply(clean_data)
```

We are now in a position to create our "metadata soup", which is a string that contains all the metadata that we want to feed to our vectorizer (namely actors, director and keywords).

25.) def create_soup(x):

```
          return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director'] + ' ' + '
'.join(x['genres'])

     f2['soup'] = f2.apply(create_soup, axis=1)
```

The next steps are the same as what we did with our plot description based recommender. One important difference is that we use the CountVectorizer() instead of TF-IDF. This is because we do not want to down-weight the presence of an actor/director if he or she has acted or directed in relatively more movies. It doesn't make much intuitive sense.

```
26.) # Import CountVectorizer and create the count matrix

from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer(stop_words='english')

count_matrix = count.fit_transform(f2['soup'])
```

```
27.) # Compute the Cosine Similarity matrix based on the count_matrix

from sklearn.metrics.pairwise import cosine_similarity

cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

```
28.) # Reset index of our main DataFrame and construct reverse mapping as before

f2 = f2.reset_index()

indices = pd.Series(f2.index, index=f2['title'])
```

We can now reuse our get_recommendations() function by passing in the new cosine_sim2 matrix as your second argument.