SIMPLE AND CONTENT BASED MOVIE RECOMMENDER

We started with a simple recommendation system followed by content based . For the simple recommender we need to take vote count and vote average into account. The following are the steps involved: • Decide on the metric or score to rate movies on. • Calculate the score for every movie. • Sort the movies based on the score and output the top results. • To calculate the score, we have weighted rating that takes into account the average rating and the number of votes it has accumulated. Such a system will make sure that a movie with a 9 rating from 100,000 voters gets a (far) higher score than a movie with the same rating but a mere few hundred voters. Since we have tried to build a clone of IMDB's Top 250, we used its weighted rating formula as a metric/score. Mathematically, it is represented as follows:

WeightedRating(WR)=(vv+m·R)+(mv+m·C)WeightedRating(WR)=(vv+m·R)+(mv+m·C) In the above equation, • 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; • C is the mean vote across the whole report.

SIMPLE RECOMMENDER

In [1]:

```
import pandas as pd
import numpy as np
C:\Users\sahit\Anaconda3\lib\site-
packages\pandas\compat\_optional.py:138: UserWarning: Pandas
requires version '2.7.0' or newer of 'numexpr' (version '2.6.9'
currently installed).
  warnings.warn(msg, UserWarning)
```

In [2]:

```
metadata = pd.read_csv('movies_metadata.csv', low_memory=False)
```

In [3]:

```
metadata.head(3)
```

Out[3]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	original_title	overview	...	release_date	revenue	runtime	spoken_languages	status	tagline	title	video	vote_average	vote_count
0	False	{'id': 10194, 'name': 'Toy Story Colle	30000000	[{'id': 16, 'name': 'Anima	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy Story	Led by Woody,	...	1995-10-30	373554033.0	81.0	[{'iso_639_1': 'en', 'name': 'Eng	Released	NaN	Toy Story	False	7.7	5415.0

		ction', ...		tion'}, {'id': 35, '...					Andy's toys live happily in his ...								lish'}]				
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt0113497	en	Jumanji	When siblings Judy and Peter discover an encha...	...	1995-12-15	2627972490.0	104.0	[{'iso_639_1': 'en', 'name': 'English'}, {'iso...	Released	Roll the dice and unleash the excitement!	Jumanji	False	6.9	2413.0
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	NaN	15602	tt0113228	en	Grumpier Old Men	A family wedding reignites the ancien	...	1995-12-22	0.0	101.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Still Yelling. Still Fighting. Still Ready for ...	Grumpier Old Men	False	6.5	92.0

t
fe
u
d
be
...

3 rows × 24 columns

In [4]:

```
C = metadata['vote_average'].mean()
print(C)
```

```
5.618207215133889
```

In [5]:

```
m = metadata['vote_count'].quantile(0.90)
print(m)
```

```
160.0
```

In [6]:

```
q_movies = metadata.copy().loc[metadata['vote_count'] >= m]
q_movies.shape
```

Out[6]: (4555, 24)

In [7]:

```
metadata.shape
```

Out[7]: (45466, 24)

In [8]:

```
def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']

    return (v/(v+m) * R) + (m/(m+v) * C)
```

This function defined the weighted_rating which is needed to calculate the score .Here we have the recommendations for the top 10 movies in IMDB based on the score we have calculated. Based on the score calculated we have the top recommendation as The Shawshank redemption and the top 10thmovie as Life is Beautiful.Though this recommender works fine it's not accurate nor customized. It's just based on the vote average and count. So, for customized recommendations we built the other two models' content and collaborative filtering.

In [9]:

```
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```

In [10]:

```
q_movies = q_movies.sort_values('score', ascending=False)
pd.set_option('precision',2)
```

In [11]:

```
q_movies[['title', 'vote_count', 'vote_average',
'score']].head(10)
```

Out[11]:

	title	vote_count	vote_average	score
314	The Shawshank Redemption	8358.0	8.5	8.45

834	The Godfather	6024.0	8.5	8.43
10309	Dilwale Dulhania Le Jayenge	661.0	9.1	8.42
12481	The Dark Knight	12269.0	8.3	8.27
2843	Fight Club	9678.0	8.3	8.26
292	Pulp Fiction	8670.0	8.3	8.25
522	Schindler's List	4436.0	8.3	8.21
23673	Whiplash	4376.0	8.3	8.21
5481	Spirited Away	3968.0	8.3	8.20
2211	Life Is Beautiful	3643.0	8.3	8.19

CONTENT BASED

Content-based recommenders: 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. And to recommend that, it will make use of the user's past item metadata. A good example could be YouTube, where based on your history, it suggests you new videos that you could potentially watch.

We had to compute Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each document. This will give us a matrix where each column represents a word in the overview vocabulary and each column represents a movie, as before.

In [12]:
```
metadata['overview'].head()
```
Out[12]:
```
0    Led by Woody, Andy's toys live happily in his ...
1    When siblings Judy and Peter discover an encha...
2    A family wedding reignites the ancient feud be...
3    Cheated on, mistreated and stepped on, the wom...
4    Just when George Banks has recovered from his ...
Name: overview, dtype: object
```

In [13]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [14]:
```
tfidf = TfidfVectorizer(stop_words='english')
```

In [15]:
```
metadata['overview'] = metadata['overview'].fillna('')
```

In [16]:
```
tfidf_matrix = tfidf.fit_transform(metadata['overview'])
```

In [17]:
```
tfidf_matrix.shape
```
Out[17]:
```
(45466, 75827)
```

In [18]:

```
tfidf.get_feature_names()[5000:5010]
```

Out[18]:
```
['avails',
 'avaks',
 'avalanche',
 'avalanches',
 'avallone',
 'avalon',
 'avant',
 'avanthika',
 'avanti',
 'avaracious']
```

In [19]:
```
from sklearn.metrics.pairwise import linear_kernel
```

We computed the cosine similarity score using the the tf-idf matrix obtained

In [20]:
```
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [21]:
```
cosine_sim.shape
```

Out[21]:
```
(45466, 45466)
```

In [22]:
```
cosine_sim[1]
```

Out[22]:
```
array([0.01504121, 1.        , 0.04681953, ..., 0.        ,
       0.02198641,
       0.00929411])
```

In [23]:
```
indices = pd.Series(metadata.index,
index=metadata['title']).drop_duplicates()
```

In [24]:
```
indices[:10]
```

Out[24]:
```
title
Toy Story                    0
Jumanji                      1
Grumpier Old Men             2
Waiting to Exhale   3
Father of the Bride Part II    4
Heat                         5
Sabrina                      6
Tom and Huck                 7
Sudden Death                 8
GoldenEye                    9
dtype: int64
```

In [25]:
```
def get_recommendations(title, cosine_sim=cosine_sim):

    idx = indices[title]
```

```
    sim_scores = list(enumerate(cosine_sim[idx]))

    sim_scores = sorted(sim_scores, key=lambda x: x[1],
reverse=True)

    sim_scores = sim_scores[1:11]

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

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

We 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.

In [26]:

```
get_recommendations('The Dark Knight Rises')
```

Out[26]:

```
12481                            The Dark Knight
150                               Batman Forever
1328                              Batman Returns
15511                 Batman: Under the Red Hood
585                                      Batman
21194    Batman Unmasked: The Psychology of the Dark Kn...
9230                 Batman Beyond: Return of the Joker
18035                               Batman: Year One
19792     Batman: The Dark Knight Returns, Part 1
3095                  Batman: Mask of the Phantasm
Name: title, dtype: object
```

In [27]:

```
get_recommendations('The Godfather')
```

Out[27]:

```
1178                The Godfather: Part II
44030    The Godfather Trilogy: 1972-1990
1914 The Godfather: Part III
23126                          Blood Ties
11297                    Household Saints
34717                   Start Liquidation
10821                            Election
38030            A Mother Should Be Loved
17729                   Short Sharp Shock
26293                  Beck 28 - Familjen
Name: title, dtype: object
```

The recommendations have been obtained for the movie "GodFather" 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 your present system.

Credits, Genres, and Keywords Based Recommender

The quality of our recommender would be increased with the usage of better metadata and by capturing more of the finer details. That is precisely what you are going to do in this section. You will build a recommender system based on the following metadata: the 3 top actors, the director, related genres, and the movie plot keywords.

The keywords, cast, and crew data are not available in your current dataset, so the first step would be to load and merge them into your main DataFrame

We had to compute Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each document. This will give us a matrix where each column represents a word in the overview vocabulary and each column represents a movie, as before.

In [28]:

```
credits = pd.read_csv('credits.csv')
keywords = pd.read_csv('keywords.csv')
```

In [29]:

```
metadata = metadata.drop([19730, 29503, 35587])
```

In [30]:

```
keywords['id'] = keywords['id'].astype('int')
credits['id'] = credits['id'].astype('int')
metadata['id'] = metadata['id'].astype('int')
```

In [31]:

```
metadata = metadata.merge(credits, on='id')
metadata = metadata.merge(keywords, on='id')
```

In [32]:

```
metadata.head(5)
```

Out[32]:

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	original_title	overview	...	spoken_languages	status	tagline	title	video	vote_average	vote_count	cast	crew	keywords
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy Story	Led by Woody, An	...	[{'iso_639_1': 'en', 'name': 'English'}]	Released	NaN	Toy Story	False	7.7	5415.0	[{'cast_id': 14, 'character	[{'credit_id': '52fe4284c3a36847f8024f49', 'de...	[{'id': 931, 'name': 'jeal

				, {'id': 35, '...						dy's toys live happily in his ...									er': 'Woody (voice)',...		ousy'}, {'id': 4290,...
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt0113497	en	Jumanji	When siblings Judy and Peter discover an enchanted cha...	...	[{'iso_639_1': 'en', 'name': 'English'}, {'iso...	Released	Roll the dice and unleash the excitement!	Jumanji	False	6.9	2413.0	[{'cast_id': 1, 'character': 'Alan Parrish', '...	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'de...	[{'id': 10090, 'name': 'board game'}, {'id': 1...
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	NaN	15602	tt0113228	en	Grumpier Old Men	A family wedding reignites the ancient fe	...	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Still Yelling. Still Fighting. Still Ready for...	Grumpier Old Men	False	6.5	92.0	[{'cast_id': 2, 'character': 'Max Goldman',	[{'credit_id': '52fe466a9251416c75077a89', 'de...	[{'id': 1495, 'name': 'fishing'}, {'id': 12392...

											ud be...												'c...			
3	False	NaN	16000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	3135 7	tt0114885	en	Waiting to Exhale	Cheated on, mistreated and stepped on, the wom...	...	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Friends are the people who let you be yourself...	Waiting to Exhale	False	6.1	34.0	[{'cast_id': 1, 'character': "Savannah 'Vannah...	[{'credit_id': '52fe447792 51416c9101 1acb', 'de...	[{'id': 818, 'name': 'based on novel'}, {'id': ...
4	False	{'id': 96871, 'name': 'Father of the Bride Col...	0	[{'id': 35, 'name': 'Comedy'}]	NaN	11862	tt0113041	en	Father of the Bride Part II	Just when George Banks has recovered from his ...	...	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Just When His World Is Back To Normal... He's ...	Father of the Bride Part II	False	5.7	173.0	[{'cast_id': 1, 'character': 'George Banks', '...	[{'credit_id': '52fe449592 51416c7503 9ed7', 'de...	[{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n...

5 rows × 27 columns

In [33]:

```
from ast import literal_eval
```

In [34]:

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

```
for feature in features:
    metadata[feature] = metadata[feature].apply(literal_eval)
```

In [35]:

```
def get_director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
    return np.nan
```

In [36]:

```
def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]

        if len(names) > 3:
            names = names[:3]
        return names

    return []
```

In [37]:

```
metadata['director'] = metadata['crew'].apply(get_director)

features = ['cast', 'keywords', 'genres']
for feature in features:
    metadata[feature] = metadata[feature].apply(get_list)
```

In [38]:

```
metadata[['title', 'cast', 'director', 'keywords',
'genres']].head(5)
```

Out[38]:

	title	cast	director	keywords	genres
0	Toy Story	[Tom Hanks, Tim Allen, Don Rickles]	John Lasseter	[jealousy, toy, boy]	[Animation, Comedy, Family]
1	Jumanji	[Robin Williams, Jonathan Hyde, Kirsten Dunst]	Joe Johnston	[board game, disappearance, based on children'...	[Adventure, Fantasy, Family]
2	Grumpier Old Men	[Walter Matthau, Jack Lemmon, Ann-Margret]	Howard Deutch	[fishing, best friend, duringcreditsstinger]	[Romance, Comedy]
3	Waiting to Exhale	[Whitney Houston, Angela Bassett, Loretta Devine]	Forest Whitaker	[based on novel, interracial relationship, sin...	[Comedy, Drama, Romance]
4	Father of the Bride Part II	[Steve Martin, Diane Keaton, Martin Short]	Charles Shyer	[baby, midlife crisis, confidence]	[Comedy]

Removing the spaces between words is an important preprocessing step. It is done so that our vectorizer doesn't count the Johnny of "Johnny Depp" and "Johnny Galecki" as the same. After this processing step, the aforementioned actors will be represented as "johnnydepp" and "johnnygalecki" and will be distinct to your vectorizer.

In [39]:

```
def clean_data(x):
```

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

        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''
```

In [40]:

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

for feature in features:
    metadata[feature] = metadata[feature].apply(clean_data)
```

In [41]:

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

In [42]:

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

In [43]:

```
metadata[['soup']].head(2)
```

Out[43]:

	soup
0	jealousy toy boy tomhanks timallen donrickles ...
1	boardgame disappearance basedonchildren'sbook ...

The major difference between CountVectorizer() and TF-IDF is the inverse document frequency (IDF) component which is present in later and not in the former.

In [44]:

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

In [45]:

```
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(metadata['soup'])
```

In [46]:

```
count_matrix.shape
```

Out[46]: (46628, 73881)

In [47]:

```
from sklearn.metrics.pairwise import cosine_similarity

cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

In [48]:

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

In [49]:

```
get_recommendations('The Dark Knight Rises', cosine_sim2)
```

Out[49]: 12589 The Dark Knight

```
10210        Batman Begins
9311                Shiner
9874       Amongst Friends
7772 Mitchell
516      Romeo Is Bleeding
11463         The Prestige
24090            Quicksand
25038             Deadfall
41063                 Sara
Name: title, dtype: object
```

In [50]:

```
get_recommendations('The Godfather', cosine_sim2)
```

Out[50]:

```
1934            The Godfather: Part III
1199             The Godfather: Part II
15609                   The Rain People
18940                        Last Exit
34488                             Rege
35802            Manuscripts Don't Burn
35803            Manuscripts Don't Burn
8001     The Night of the Following Day
18261                The Son of No One
28683            In the Name of the Law
Name: title, dtype: object
```

Our recommender has been successful in capturing more information due to more metadata and has given us better recommendations.