PRESTIGE INSTITUTE OF MANAGEMENT AND RESEARCH, INDORE

(An Autonomous Institution Established in 1994, Accredited Twice
Consecutively with Grade A ‟NAAC’’ (UGC) ISO 9001: 2008 Certified
Institute, AICTE / UGC Approved Programs affiliated to DAVV, Indore)

(Session 2022 – 2022)

Multivariate Data Analysis Project Report
on

“Movie Recommendation System”

Faculty Mentor:
Mr. Makhan Kumbhkar

Submitted By:
Sakaar Mathur
Arin Jaiswal

Class:
BBA Business Analytics
V Semester (A)

MOVIE RECOMMENDATION SYSTEM

Index

Acknowledgement

We would like to thank Mr. Makhan Kumbhkar, our Professor-in-charge and our Director, Dr. Subramanian Raman Iyer for their support and guidance in completing our project on the topic, “Movie Recommendation System”. It was a great learning experience.

We would like to take this opportunity to express our gratitude to all of the group members Sakaar Mathur and Arin Jaiswal. The project would not have been successful without their cooperation and inputs.

Introduction

A movie recommendation system, or a movie recommender system, is an ML-based approach to filtering or predicting the users' film preferences based on their past choices and behavior. It's an advanced filtration mechanism that predicts the possible movie choices of the concerned user and their preferences towards a domain-specific item, aka movie.

The basic concept behind a movie recommendation system is quite simple. In particular, there are two main elements in every recommender system: users and items. The system generates movie predictions for its users, while items are the movies themselves.

The primary goal of movie recommendation systems is to filter and predict only those movies that a corresponding user is most likely to want to watch. The ML algorithms for these recommendation systems use the data about this user from the system's database. This data is used to predict the future behavior of the user concerned based on the information from the past.

Filtration Strategies for Movie Recommendation Systems:

Movie recommendation systems use a set of different filtration strategies and algorithms to help users find the most relevant films. The most popular categories of the ML algorithms used for movie recommendations include content-based filtering and collaborative filtering systems.

Content-Based Filtering:

A filtration strategy for movie recommendation systems, which uses the data provided about the items (movies). This data plays a crucial role here and is extracted from only one user. An ML algorithm used for this strategy recommends motion pictures that are similar to the user's preferences in the past. Therefore, the similarity in content-based filtering is generated by the data about the past film selections and likes by only one user.

The recommendation system analyzes the past preferences of the user concerned, and then it uses this information to try to find similar movies. This information is available in the database (e.g., lead actors, director, genre, etc.). After that, the system provides movie recommendations for the user. That said, the core element in content-based filtering is only the data of only one user that is used to make predictions.

Collaborative Filtering:

As the name suggests, this filtering strategy is based on the combination of the relevant user's and other users' behaviors. The system compares and contrasts these behaviors for the most optimal results. It's a collaboration of the multiple users' film preferences and behaviors.

What's the mechanism behind this strategy? The core element in this movie recommendation system and the ML algorithm it's built on is the history of all users in the database. Basically, collaborative filtering is based on the interaction of all users in the system with the items (movies). Thus, every user impacts the final outcome of this ML-based recommendation system, while content-based filtering depends strictly on the data from one user for its modeling.

Collaborative filtering algorithms are divided into two categories:

User-based collaborative filtering. The idea is to look for similar patterns in movie preferences in the target user and other users in the database.

Item-based collaborative filtering. The basic concept here is to look for similar items (movies) that target users rate or interact with.

The modern approach to the movie recommendation systems implies a mix of both strategies for the most gradual and explicit results.

<u>The Top Movie Recommendation System Use Cases:</u>

One can spot the increased use of the recommendation systems on almost every popular streaming service, social media, or e-commerce platform. These include Amazon, YouTube, Netflix, Facebook, to name a few. These recommendation systems help different industries provide more personalized experiences to their users.

Machine learning Steps for our Project

- Collecting Data:
- Preparing the Data:
- Data preprocessing
- Building Movie Recommendation System

Collecting the data

This dataset was generated from The Movie Database API. This product uses the TMDb API but is not endorsed or certified by TMDb.

Preparing the Data

1. Merging the data
2. Treating null values

Data preprocessing

1. Data-Processing for Movie Tags
2. Stemming
3. Vectorization

Building Movie Recommendation System

- Cosine Similarity
- Recommender

Code

Movies Recommendation System

Importing Required Libraries

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Importing Required Libraries
# for Movies Poster Reuest
import requests
from IPython.display import Image, HTML, display
import warnings
def warns(*args,**kwargs): pass
warnings.warn=warns
```

Importing Dataset

```
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving tmdb_5000_credits.csv to tmdb_5000_credits.csv
Saving tmdb_5000_movies.csv to tmdb_5000_movies.csv
```

```
dfmovies = pd.read_csv("tmdb_5000_movies.csv")
dfcredits = pd.read_csv("tmdb_5000_credits.csv")
print("Movies Shape: ", dfmovies.shape)
print("Credits Shape: ", dfcredits.shape)
```

```
Movies Shape:  (4803, 20)
Credits Shape:  (4803, 4)
```

Exploratory Data Analysis

```
dfmovies.head()
```

```
      budget                                             genres  \
0  237000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam...
1  300000000  [{"id": 12, "name": "Adventure"}, {"id": 14, "...
2  245000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam...
3  250000000  [{"id": 28, "name": "Action"}, {"id": 80, "nam...
4  260000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam...

                                       homepage      id  \
0                    http://www.avatarmovie.com/   19995
1  http://disney.go.com/disneypictures/pirates/     285
```

```
2   http://www.sonypictures.com/movies/spectre/  206647
3            http://www.thedarkknightrises.com/   49026
4          http://movies.disney.com/john-carter   49529

                                            keywords original_language  \
0  [{"id": 1463, "name": "culture clash"}, {"id":...                en
1  [{"id": 270, "name": "ocean"}, {"id": 726, "na...                en
2  [{"id": 470, "name": "spy"}, {"id": 818, "name...                en
3  [{"id": 849, "name": "dc comics"}, {"id": 853,...                en
4  [{"id": 818, "name": "based on novel"}, {"id":...                en

                            original_title  \
0                                   Avatar
1  Pirates of the Caribbean: At World's End
2                                  Spectre
3                    The Dark Knight Rises
4                              John Carter

                                            overview  popularity  \
0  In the 22nd century, a paraplegic Marine is di...  150.437577
1  Captain Barbossa, long believed to be dead, ha...  139.082615
2  A cryptic message from Bond’s past sends him o...  107.376788
3  Following the death of District Attorney Harve...  112.312950
4  John Carter is a war-weary, former military ca...   43.926995

                                production_companies  \
0  [{"name": "Ingenious Film Partners", "id": 289...
1  [{"name": "Walt Disney Pictures", "id": 2}, {"...
2  [{"name": "Columbia Pictures", "id": 5}, {"nam...
3  [{"name": "Legendary Pictures", "id": 923}, {"...
4        [{"name": "Walt Disney Pictures", "id": 2}]

                                production_countries release_date     revenue
\
0  [{"iso_3166_1": "US", "name": "United States o...   2009-12-10  2787965087
1  [{"iso_3166_1": "US", "name": "United States o...   2007-05-19   961000000
2  [{"iso_3166_1": "GB", "name": "United Kingdom"...   2015-10-26   880674609
3  [{"iso_3166_1": "US", "name": "United States o...   2012-07-16  1084939099
4  [{"iso_3166_1": "US", "name": "United States o...   2012-03-07   284139100

   runtime                                   spoken_languages    status  \
0    162.0  [{"iso_639_1": "en", "name": "English"}, {"iso...  Released
1    169.0           [{"iso_639_1": "en", "name": "English"}]  Released
2    148.0  [{"iso_639_1": "fr", "name": "Fran\u00e7ais"},...  Released
3    165.0           [{"iso_639_1": "en", "name": "English"}]  Released
4    132.0           [{"iso_639_1": "en", "name": "English"}]  Released

                                          tagline  \
0                      Enter the World of Pandora.
1  At the end of the world, the adventure begins.
```

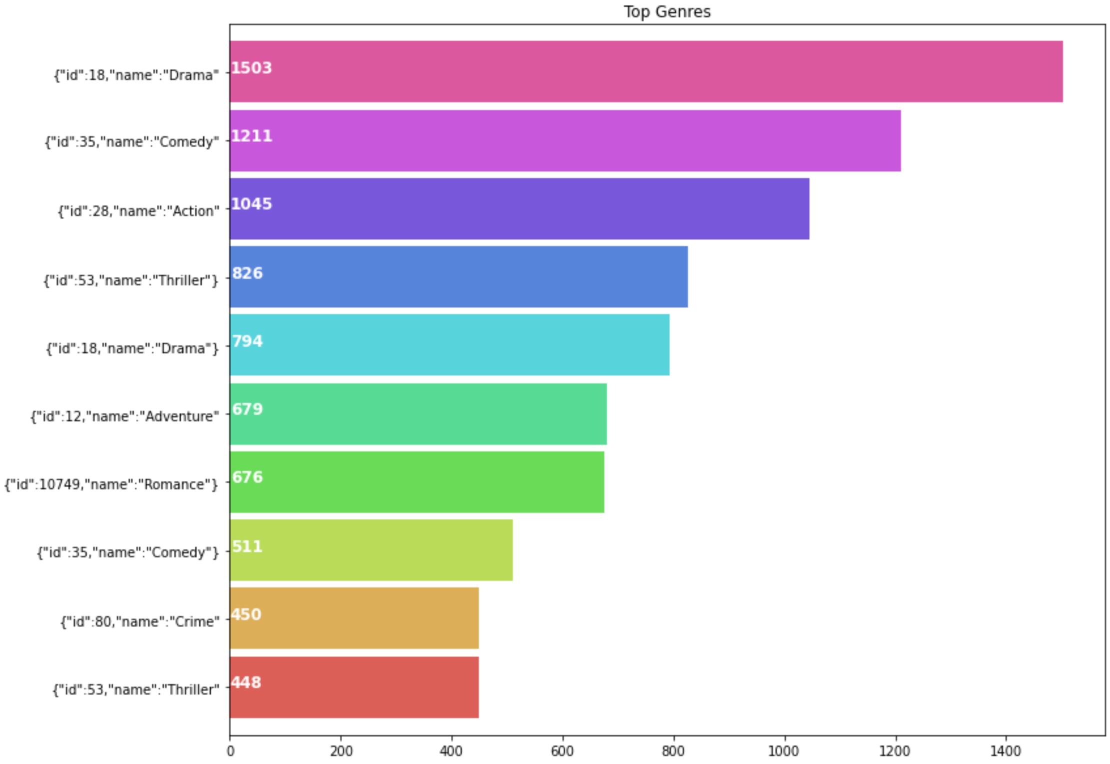
```
2                            A Plan No One Escapes
3                                  The Legend Ends
4             Lost in our world, found in another.

                                      title  vote_average  vote_count
0                                    Avatar           7.2       11800
1  Pirates of the Caribbean: At World's End           6.9        4500
2                                   Spectre           6.3        4466
3                     The Dark Knight Rises           7.6        9106
4                               John Carter           6.1        2124

dfcredits.head()

   movie_id                                     title  \
0     19995                                    Avatar
1       285  Pirates of the Caribbean: At World's End
2    206647                                   Spectre
3     49026                     The Dark Knight Rises
4     49529                               John Carter

                                                cast  \
0  [{"cast_id": 242, "character": "Jake Sully", "...
1  [{"cast_id": 4, "character": "Captain Jack Spa...
2  [{"cast_id": 1, "character": "James Bond", "cr...
3  [{"cast_id": 2, "character": "Bruce Wayne / Ba...
4  [{"cast_id": 5, "character": "John Carter", "c...

                                                crew
0  [{"credit_id": "52fe48009251416c750aca23", "de...
1  [{"credit_id": "52fe4232c3a36847f800b579", "de...
2  [{"credit_id": "54805967c3a36829b5002c41", "de...
3  [{"credit_id": "52fe4781c3a36847f81398c3", "de...
4  [{"credit_id": "52fe479ac3a36847f813eaa3", "de...
```

```
print("-"*20, "Movies Columns", "-"*20, "\n", dfmovies.columns)
print("\n", "-"*20, "Credites Columns", "-"*20, "\n", dfcredits.columns)
```

```
-------------------- Movies Columns --------------------
 Index(['budget', 'genres', 'homepage', 'id', 'keywords',
'original_language',
       'original_title', 'overview', 'popularity', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'vote_average',
       'vote_count'],
      dtype='object')

 -------------------- Credites Columns --------------------
 Index(['movie_id', 'title', 'cast', 'crew'], dtype='object')
```

Data Visualisation

```
tg = dfmovies['genres']
tg = tg.str.strip('[]').str.replace(' ','').str.replace("'",'')
tg = tg.str.split('},')

plt.subplots(figsize=(12,10))
list1 = []
for i in tg:
    list1.extend(i)
ax =
pd.Series(list1).value_counts()[:10].sort_values(ascending=True).plot.barh(wi
dth=0.9,color=sns.color_palette('hls',10))
for i, v in
enumerate(pd.Series(list1).value_counts()[:10].sort_values(ascending=True).va
lues):
    ax.text(.8, i, v,fontsize=12,color='white',weight='bold')
plt.title('Top Genres')
plt.show()
```



```
list1[0:50]
```

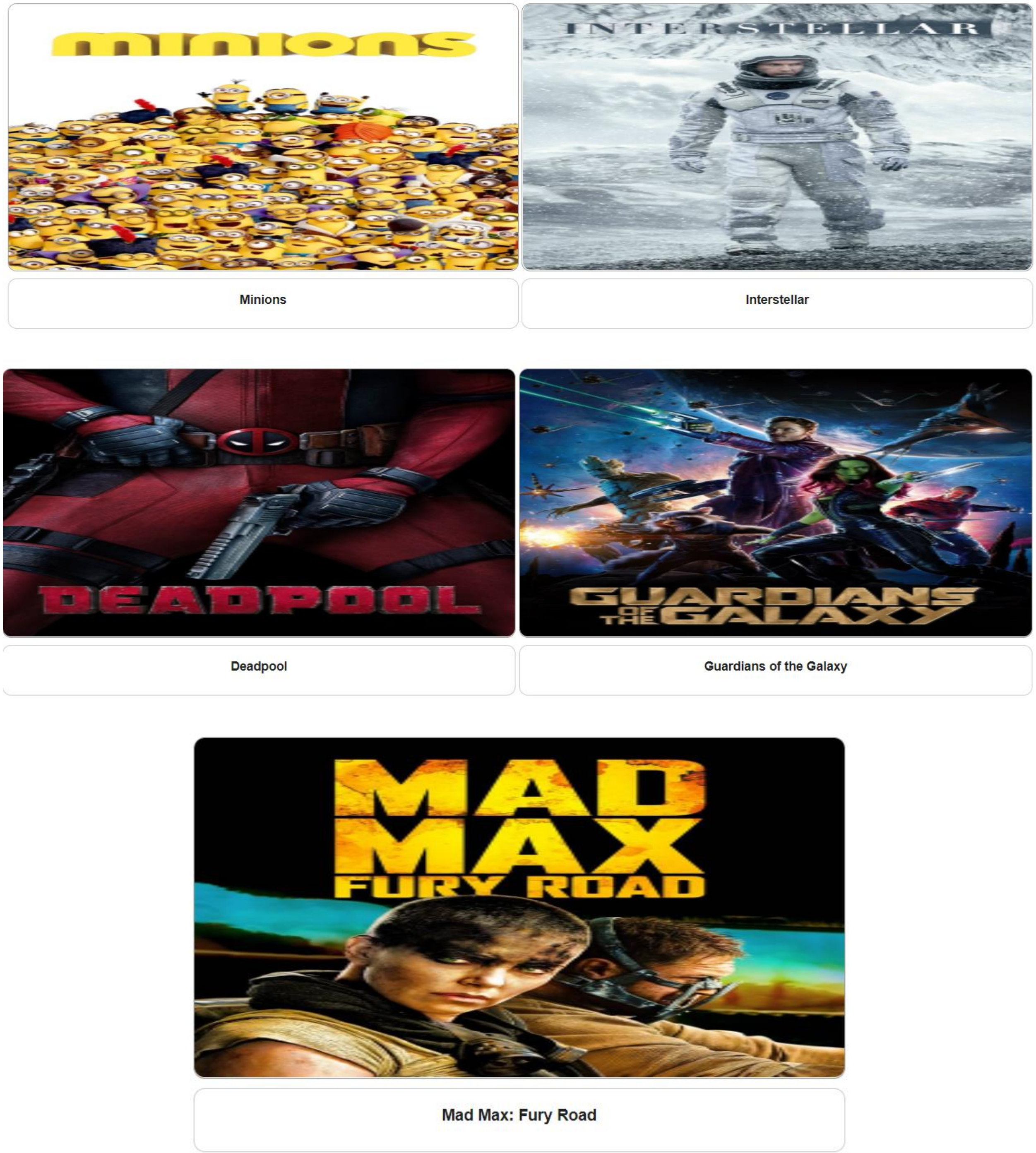
```
['{"id":28,"name":"Action"',
 '{"id":12,"name":"Adventure"',
 '{"id":14,"name":"Fantasy"',
 '{"id":878,"name":"ScienceFiction"}',
 '{"id":12,"name":"Adventure"',
 '{"id":14,"name":"Fantasy"',
 '{"id":28,"name":"Action"}',
 '{"id":28,"name":"Action"',
 '{"id":12,"name":"Adventure"',
 '{"id":80,"name":"Crime"}',
 '{"id":28,"name":"Action"',
 '{"id":80,"name":"Crime"',
 '{"id":18,"name":"Drama"',
 '{"id":53,"name":"Thriller"}',
 '{"id":28,"name":"Action"',
 '{"id":12,"name":"Adventure"',
 '{"id":878,"name":"ScienceFiction"}',
 '{"id":14,"name":"Fantasy"',
 '{"id":28,"name":"Action"',
 '{"id":12,"name":"Adventure"}',
 '{"id":16,"name":"Animation"',
 '{"id":10751,"name":"Family"}',
 '{"id":28,"name":"Action"',
 '{"id":12,"name":"Adventure"',
 '{"id":878,"name":"ScienceFiction"}',
 '{"id":12,"name":"Adventure"',
 '{"id":14,"name":"Fantasy"',
 '{"id":10751,"name":"Family"}',
 '{"id":28,"name":"Action"',
 '{"id":12,"name":"Adventure"',
 '{"id":14,"name":"Fantasy"}',
 '{"id":12,"name":"Adventure"',
 '{"id":14,"name":"Fantasy"',
 '{"id":28,"name":"Action"',
 '{"id":878,"name":"ScienceFiction"}',
 '{"id":12,"name":"Adventure"',
 '{"id":28,"name":"Action"',
 '{"id":53,"name":"Thriller"',
 '{"id":80,"name":"Crime"}',
 '{"id":12,"name":"Adventure"',
 '{"id":14,"name":"Fantasy"',
 '{"id":28,"name":"Action"}',
 '{"id":28,"name":"Action"',
 '{"id":12,"name":"Adventure"',
 '{"id":37,"name":"Western"}',
 '{"id":28,"name":"Action"',
 '{"id":12,"name":"Adventure"',
 '{"id":14,"name":"Fantasy"',
 '{"id":878,"name":"ScienceFiction"}',
 '{"id":12,"name":"Adventure"']
```

Top 5-Movies as per Popularity Calculation

```
popular_movies = dfmovies.nlargest(n=5, columns=['popularity'])[['id',
'title']]
getList_name = {}
for x, xRows in popular_movies.iterrows():
    #print(xRows['id'])
    getResponse =
requests.get('https://api.themoviedb.org/3/movie/{}?api_key=c0bda0be71f7815fd
6ba2eb5f5c86fd8'.format(xRows['id']) )
    getData = getResponse.json()
    getPath = "http://image.tmdb.org/t/p/w500" + getData['poster_path']
    getList_name[xRows['title']] = getPath

display( HTML(f"""<table>
                    <tr>
                        <td><img src={list(getList_name.values())[0]}
style='border-radius:10px; height:300px; width:575px; border:1px solid
#999;'></td>
                        <td><img src={list(getList_name.values())[1]}
style='border-radius:10px; height:300px; width:575px; border:1px solid
#999;'></td>
                        <td><img src={list(getList_name.values())[2]}
style='border-radius:10px; height:300px; width:575px; border:1px solid
#999;'></td>
                        <td><img src={list(getList_name.values())[3]}
style='border-radius:10px; height:300px; width:575px; border:1px solid
#999;'></td>
                        <td><img src={list(getList_name.values())[4]}
style='border-radius:10px; height:300px; width:575px; border:1px solid
#999;'></td>
                    </tr>
                    <tr>
                        <td><div style="height:40px; padding-top:15px; text-
align:center; font-size:14px; font-weight:bold; border:1px solid #ccc;
border-radius:10px;">{list(getList_name.keys())[0]}</div></td>
                        <td><div style="height:40px; padding-top:15px; text-
align:center; font-size:14px; font-weight:bold; border:1px solid #ccc;
border-radius:10px;">{list(getList_name.keys())[1]}</div></td>
                        <td><div style="height:40px; padding-top:15px; text-
align:center; font-size:14px; font-weight:bold; border:1px solid #ccc;
border-radius:10px;">{list(getList_name.keys())[2]}</div></td>
                        <td><div style="height:40px; padding-top:15px; text-
align:center; font-size:14px; font-weight:bold; border:1px solid #ccc;
border-radius:10px;">{list(getList_name.keys())[3]}</div></td>
```

```
                        <td><div style="height:40px; padding-top:15px; text-
align:center; font-size:14px; font-weight:bold; border:1px solid #ccc;
border-radius:10px;">{list(getList_name.keys())[4]}</div></td>
                    </tr>
                </table>"""))
```

```
<IPython.core.display.HTML object>
```



Minions

Interstellar

Deadpool

Guardians of the Galaxy

Mad Max: Fury Road

Data Merging

```
# Combine the 2 dataframes in single on title base
dfcombine = dfmovies.merge(dfcredits, on='title')
print("Cobine dataframe Shape: ", dfcombine.shape)
```

```
Cobine dataframe Shape:  (4809, 23)
```

```
dfcombine.columns
```

```
Index(['budget', 'genres', 'homepage', 'id', 'keywords', 'original_language',
       'original_title', 'overview', 'popularity', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'vote_average',
       'vote_count', 'movie_id', 'cast', 'crew'],
      dtype='object')
```

```
# Re-Create the Dataset with the selected columns for recomender system
# ['id', 'title', 'original_title', 'tagline', 'genres', 'keywords', 'cast',
'crew', 'overview']

features_cols = ['id', 'title', 'original_title', 'tagline', 'genres',
'keywords', 'cast', 'crew', 'overview']

df_final = dfcombine[features_cols]
df_final.head()
```

```
       id                                     title  \
0   19995                                    Avatar   
1     285  Pirates of the Caribbean: At World's End   
2  206647                                   Spectre   
3   49026                     The Dark Knight Rises   
4   49529                               John Carter   

                             original_title  \
0                                    Avatar   
1  Pirates of the Caribbean: At World's End   
2                                   Spectre   
3                     The Dark Knight Rises   
4                               John Carter   

                                          tagline  \
0                     Enter the World of Pandora.   
1  At the end of the world, the adventure begins.   
2                           A Plan No One Escapes   
3                                 The Legend Ends   
4            Lost in our world, found in another.   

                                              genres  \
0  [{"id": 28, "name": "Action"}, {"id": 12, "nam...   
1  [{"id": 12, "name": "Adventure"}, {"id": 14, "...   
```

```
2  [{"id": 28, "name": "Action"}, {"id": 12, "nam...
3  [{"id": 28, "name": "Action"}, {"id": 80, "nam...
4  [{"id": 28, "name": "Action"}, {"id": 12, "nam...

                                          keywords  \
0  [{"id": 1463, "name": "culture clash"}, {"id":...
1  [{"id": 270, "name": "ocean"}, {"id": 726, "na...
2  [{"id": 470, "name": "spy"}, {"id": 818, "name...
3  [{"id": 849, "name": "dc comics"}, {"id": 853,...
4  [{"id": 818, "name": "based on novel"}, {"id":...

                                              cast  \
0  [{"cast_id": 242, "character": "Jake Sully", "...
1  [{"cast_id": 4, "character": "Captain Jack Spa...
2  [{"cast_id": 1, "character": "James Bond", "cr...
3  [{"cast_id": 2, "character": "Bruce Wayne / Ba...
4  [{"cast_id": 5, "character": "John Carter", "c...

                                              crew  \
0  [{"credit_id": "52fe48009251416c750aca23", "de...
1  [{"credit_id": "52fe4232c3a36847f800b579", "de...
2  [{"credit_id": "54805967c3a36829b5002c41", "de...
3  [{"credit_id": "52fe4781c3a36847f81398c3", "de...
4  [{"credit_id": "52fe479ac3a36847f813eaa3", "de...

                                          overview
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...
```

Treating Missing Values

```
# Checking Missing Values
df_final.isnull().sum()
```

```
id                  0
title               0
original_title      0
tagline           844
genres              0
keywords            0
cast                0
crew                0
overview            3
dtype: int64
```

```
#Verifying Missing Values
df_final['tagline'] = df_final['tagline'].fillna('unknown')
df_final.dropna(inplace=True)
df_final.isnull().sum()
```

```
id                0
title             0
original_title    0
tagline           0
genres            0
keywords          0
cast              0
crew              0
overview          0
dtype: int64
```

```
df_final.head()
```

```
       id                                     title  \
0   19995                                    Avatar   
1     285  Pirates of the Caribbean: At World's End   
2  206647                                   Spectre   
3   49026                     The Dark Knight Rises   
4   49529                               John Carter   

                             original_title  \
0                                    Avatar   
1  Pirates of the Caribbean: At World's End   
2                                   Spectre   
3                     The Dark Knight Rises   
4                               John Carter   

                                          tagline  \
0                       Enter the World of Pandora.   
1  At the end of the world, the adventure begins.   
2                             A Plan No One Escapes   
3                                  The Legend Ends   
4             Lost in our world, found in another.   

                                              genres  \
0  [{"id": 28, "name": "Action"}, {"id": 12, "nam...   
1  [{"id": 12, "name": "Adventure"}, {"id": 14, "...   
2  [{"id": 28, "name": "Action"}, {"id": 12, "nam...   
3  [{"id": 28, "name": "Action"}, {"id": 80, "nam...   
4  [{"id": 28, "name": "Action"}, {"id": 12, "nam...   

                                            keywords  \
0  [{"id": 1463, "name": "culture clash"}, {"id":...   
1  [{"id": 270, "name": "ocean"}, {"id": 726, "na...   
2  [{"id": 470, "name": "spy"}, {"id": 818, "name...   
```

```
3  [{"id": 849, "name": "dc comics"}, {"id": 853,...
4  [{"id": 818, "name": "based on novel"}, {"id":...

                                                cast  \
0  [{"cast_id": 242, "character": "Jake Sully", "...
1  [{"cast_id": 4, "character": "Captain Jack Spa...
2  [{"cast_id": 1, "character": "James Bond", "cr...
3  [{"cast_id": 2, "character": "Bruce Wayne / Ba...
4  [{"cast_id": 5, "character": "John Carter", "c...

                                                crew  \
0  [{"credit_id": "52fe48009251416c750aca23", "de...
1  [{"credit_id": "52fe4232c3a36847f800b579", "de...
2  [{"credit_id": "54805967c3a36829b5002c41", "de...
3  [{"credit_id": "52fe4781c3a36847f81398c3", "de...
4  [{"credit_id": "52fe479ac3a36847f813eaa3", "de...

                                            overview
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...
```

Data Preprocessing

Data-Processing for Movie Tags

```
#Verify the Overview of Index 0
df_final['overview'][0]
```

```
{"type":"string"}
```

```
#Verify Overview Column by Applying Split Function
df_final['overview'].apply(lambda x: x.lower() and x.split() )
```

```
0       [In, the, 22nd, century,, a, paraplegic, Marin...
1       [Captain, Barbossa,, long, believed, to, be, d...
2       [A, cryptic, message, from, Bond’s, past, send...
3       [Following, the, death, of, District, Attorney...
4       [John, Carter, is, a, war-weary,, former, mili...
                              ...                        
4804    [El, Mariachi, just, wants, to, play, his, gui...
4805    [A, newlywed, couple's, honeymoon, is, upended...
4806    ["Signed,, Sealed,, Delivered", introduces, a,...
4807    [When, ambitious, New, York, attorney, Sam, is...
4808    [Ever, since, the, second, grade, when, he, fi...
Name: overview, Length: 4806, dtype: object
```

```
# For Creating Tags, add new columns with name [tags]
#Apply Split Function on Overview Column and Save in New Columns [tags]
df_final['tags'] = df_final['overview'].apply(lambda x: x.lower() and
x.split() )
df_final['tags'][0]
```

```
['In',
 'the',
 '22nd',
 'century,',
 'a',
 'paraplegic',
 'Marine',
 'is',
 'dispatched',
 'to',
 'the',
 'moon',
 'Pandora',
 'on',
 'a',
 'unique',
 'mission,',
 'but',
 'becomes',
 'torn',
 'between',
 'following',
 'orders',
 'and',
 'protecting',
 'an',
 'alien',
 'civilization.']
```

```
# Verify the [genres] values
df_final.iloc[0]['genres']
```

```
[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14,
"name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]
```

```
# Verify the [keywords] values
df_final.iloc[0]['keywords']
```

```
[{"id": 1463, "name": "culture clash"}, {"id": 2964, "name": "future"},
{"id": 3386, "name": "space war"}, {"id": 3388, "name": "space colony"},
{"id": 3679, "name": "society"}, {"id": 3801, "name": "space travel"}, {"id":
9685, "name": "futuristic"}, {"id": 9840, "name": "romance"}, {"id": 9882,
"name": "space"}, {"id": 9951, "name": "alien"}, {"id": 10148, "name":
"tribe"}, {"id": 10158, "name": "alien planet"}, {"id": 10987, "name":
"cgi"}, {"id": 11399, "name": "marine"}, {"id": 13065, "name": "soldier"},
{"id": 14643, "name": "battle"}, {"id": 14720, "name": "love affair"}, {"id":
165431, "name": "anti war"}, {"id": 193554, "name": "power relations"},
{"id": 206690, "name": "mind and soul"}, {"id": 209714, "name": "3d"}]
```

```
# import module [ast]
# ast module have fuction [literal_eval] that converts the string in list
# Create function for [genres] and [keywords]

import ast
def colData_Transform(obj):
    lst = []
    for i in ast.literal_eval(obj):
        lst.append(i['name'].replace(" ","").lower())
    return lst

# Apply Function
df_final['tags'] = df_final['tags'] +
df_final['genres'].apply(colData_Transform)
df_final['tags'] = df_final['tags'] +
df_final['keywords'].apply(colData_Transform)
df_final['tags'][0]
```

```
['In',
 'the',
 '22nd',
 'century,',
 'a',
 'paraplegic',
 'Marine',
 'is',
 'dispatched',
 'to',
 'the',
 'moon',
 'Pandora',
 'on',
 'a',
 'unique',
 'mission,',
 'but',
 'becomes',
 'torn',
 'between',
 'following',
 'orders',
 'and',
 'protecting',
 'an',
 'alien',
 'civilization.',
 'action',
 'adventure',
 'fantasy',
 'sciencefiction',
 'cultureclash',
 'future',
 'spacewar',
 'spacecolony',
 'society',
 'spacetravel',
 'futuristic',
 'romance',
 'space',
 'alien',
 'tribe',
 'alienplanet',
 'cgi',
 'marine',
 'soldier',
 'battle',
 'loveaffair',
 'antiwar',
 'powerrelations',
 'mindandsoul',
 '3d']
```

```
# Function to Extract the first 5-names from cast
# Create Function for [Cast]
def fetch_cast_data(obj):
    lst = []
    counter = 0
    for i in ast.literal_eval(obj):
        if counter < 5:
            lst.append(i['name'].replace(" ","").lower())
            counter += 1

    return lst

df_final['tags'] = df_final['tags'] + df_final['cast'].apply(fetch_cast_data)
df_final['tags'][0]
```

```
['In',
 'the',
 '22nd',
 'century,',
 'a',
 'paraplegic',
 'Marine',
 'is',
 'dispatched',
 'to',
 'the',
 'moon',
 'Pandora',
 'on',
 'a',
 'unique',
 'mission,',
 'but',
 'becomes',
 'torn',
 'between',
 'following',
 'orders',
 'and',
 'protecting',
 'an',
 'alien',
 'civilization.',
 'action',
 'adventure',
 'fantasy',
 'sciencefiction',
 'cultureclash',
 'future',
 'spacewar',
 'spacecolony',
 'society',
 'spacetravel',
 'futuristic',
 'romance',
 'space',
 'alien',
 'tribe',
 'alienplanet',
 'cgi',
 'marine',
 'soldier',
 'battle',
 'loveaffair',
 'antiwar',
 'powerrelations',
 'mindandsoul',
 '3d',
 'samworthington',
 'zoesaldana',
 'sigourneyweaver',
 'stephenlang',
 'michellerodriguez']
```

```
df_final['crew'][0]
```

```
{"type":"string"}
```

```
# Function to Extract the Director Name from Crew
# Create Function for [Crew]
def get_crew_data(obj):
    lst = []
    for x in ast.literal_eval(obj):
        if x['job'] == "Director":
            lst.append(x['name'].replace(" ","").lower())

    return lst

df_final['tags'] = df_final['tags'] + df_final['crew'].apply(get_crew_data)
df_final['tags'][0]
```

```
['In',
 'the',
 '22nd',
 'century,',
 'a',
 'paraplegic',
 'Marine',
 'is',
 'dispatched',
 'to',
 'the',
 'moon',
 'Pandora',
 'on',
 'a',
 'unique',
 'mission,',
 'but',
 'becomes',
 'torn',
 'between',
 'following',
 'orders',
 'and',
 'protecting',
 'an',
 'alien',
 'civilization.',
 'action',
 'adventure',
 'fantasy',
 'sciencefiction',
 'cultureclash',
 'future',
 'spacewar',
 'spacecolony',
 'society',
 'spacetravel',
 'futuristic',
 'romance',
 'space',
 'alien',
 'tribe',
 'alienplanet',
 'cgi',
 'marine',
 'soldier',
 'battle',
 'loveaffair',
 'antiwar',
 'powerrelations',
 'mindandsoul',
 '3d',
 'samworthington',
 'zoesaldana',
 'sigourneyweaver',
 'stephenlang',
 'michellerodriguez',
 'jamescameron']
```

```
# Convert movie title in list and concatenate with tags
df_final['tags'] = df_final['tags'] + df_final['title'].apply(lambda x:
x.lower() and x.split() )

# Convert movie original_title in list and concatenate with tags
# if original_title not equal to title then concatenate otherwise skip

for x, xRow in df_final.iterrows():
    if (xRow['original_title'] != xRow['title'] ):
        df_final.at[x, 'tags'] = xRow['tags'] +
xRow['original_title'].lower().split()

df_final['tags'][0]
```

```
['In',
 'the',
 '22nd',
 'century,',
 'a',
 'paraplegic',
 'Marine',
 'is',
 'dispatched',
 'to',
 'the',
 'moon',
 'Pandora',
 'on',
 'a',
 'unique',
 'mission,',
 'but',
 'becomes',
 'torn',
 'between',
 'following',
 'orders',
 'and',
 'protecting',
 'an',
 'alien',
 'civilization.',
 'action',
 'adventure',
 'fantasy',
 'sciencefiction',
 'cultureclash',
 'future',
 'spacewar',
 'spacecolony',
 'society',
 'spacetravel',
 'futuristic',
 'romance',
 'space',
 'alien',
 'tribe',
 'alienplanet',
 'cgi',
 'marine',
 'soldier',
 'battle',
 'loveaffair',
 'antiwar',
 'powerrelations',
 'mindandsoul',
 '3d',
 'samworthington',
 'zoesaldana',
 'sigourneyweaver',
 'stephenlang',
 'michellerodriguez',
 'jamescameron',
 'Avatar']
```

```
# Converting taglines in list for concatenation with tags
df_final['tagline'].apply(lambda x: x.lower() and x.split() )
```

```
0                        [Enter, the, World, of, Pandora.]
1        [At, the, end, of, the, world,, the, adventure...
2                              [A, Plan, No, One, Escapes]
3                                      [The, Legend, Ends]
4             [Lost, in, our, world,, found, in, another.]
                               ...                        
4804     [He, didn't, come, looking, for, trouble,, but...
4805     [A, newlywed, couple's, honeymoon, is, upended...
4806                                             [unknown]
4807                        [A, New, Yorker, in, Shanghai]
4808                                             [unknown]
Name: tagline, Length: 4806, dtype: object
```

```
# Convert movie tagline in list and concatenate with tags
# if tagline 'unknown' then skip

for x, xRow in df_final.iterrows():
    if (xRow['tagline'] != 'unknown'):
        df_final.at[x, 'tags'] = xRow['tags'] + 
xRow['tagline'].lower().split()

df_final['tags'][0]
```

```
['In',
 'the',
 '22nd',
 'century,',
 'a',
 'paraplegic',
 'Marine',
 'is',
 'dispatched',
 'to',
 'the',
 'moon',
 'Pandora',
 'on',
 'a',
 'unique',
 'mission,',
 'but',
 'becomes',
 'torn',
 'between',
 'following',
 'orders',
 'and',
 'protecting',
 'an',
 'alien',
 'civilization.',
 'action',
 'adventure',
 'fantasy',
 'sciencefiction',
 'cultureclash',
 'future',
 'spacewar',
 'spacecolony',
 'society',
 'spacetravel',
 'futuristic',
 'romance',
 'space',
 'alien',
 'tribe',
 'alienplanet',
 'cgi',
 'marine',
 'soldier',
 'battle',
 'loveaffair',
 'antiwar',
 'powerrelations',
 'mindandsoul',
 '3d',
 'samworthington',
 'zoesaldana',
 'sigourneyweaver',
 'stephenlang',
 'michellerodriguez',
 'jamescameron',
 'Avatar',
 'enter',
 'the',
 'world',
 'of',
 'pandora.']
```

```
df_final.columns
```

```
Index(['id', 'title', 'original_title', 'tagline', 'genres', 'keywords',
       'cast', 'crew', 'overview', 'tags'],
      dtype='object')
```

```
# Create new DataFrame with required columns only
df_new_final = df_final[['id', 'title', 'tags']]
df_new_final.head()
```

```
       id                                     title  \
0   19995                                    Avatar   
1     285  Pirates of the Caribbean: At World's End   
2  206647                                   Spectre   
3   49026                     The Dark Knight Rises   
4   49529                               John Carter   

                                                tags  
0  [In, the, 22nd, century,, a, paraplegic, Marin...  
1  [Captain, Barbossa,, long, believed, to, be, d...  
2  [A, cryptic, message, from, Bond’s, past, send...  
3  [Following, the, death, of, District, Attorney...  
4  [John, Carter, is, a, war-weary,, former, mili...  
```

```
# Transform list of tags in the string and in lowercase
df_new_final['tags'] = df_new_final['tags'].apply(lambda x: "
".join(x).lower())
df_new_final['tags'] = df_new_final['tags'].str.lower()
df_new_final.head()
```

```
       id                                     title  \
0   19995                                    Avatar   
1     285  Pirates of the Caribbean: At World's End   
2  206647                                   Spectre   
3   49026                     The Dark Knight Rises   
4   49529                               John Carter   

                                                tags  
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...  
```

Steming

```
# Apply Steming to remove similarities/duplications in words list
import nltk
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
# Apply steming on selected column and return in string
def xStem(txt):
    y = []
    for x in txt.split():
        y.append(ps.stem(x))
    return " ".join(y)
```

```
df_new_final['tags'] = df_new_final['tags'].apply(xStem)
df_new_final['tags'][0]
```

```
in the 22nd century, a parapleg marin is dispatch to the moon pandora on a
uniqu mission, but becom torn between follow order and protect an alien
civilization. action adventur fantasi sciencefict cultureclash futur spacewar
spacecoloni societi spacetravel futurist romanc space alien tribe alienplanet
cgi marin soldier battl loveaffair antiwar powerrel mindandsoul 3d
samworthington zoesaldana sigourneyweav stephenlang michellerodriguez
jamescameron avatar enter the world of pandora.
```

Vectorization

```
# Removing Stop-words
from sklearn.feature_extraction.text import CountVectorizer
countVec = CountVectorizer(max_features=10000, stop_words='english')

# Tags of Movies Convert in Vectors
dataVectors = countVec.fit_transform(df_new_final['tags']).toarray()
dataVectors
```

```
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]])
```

```
dataVectors[0]
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
countVec.get_feature_names()[130:250]
```

```
['aarontaylor',
 'abandon',
 'abbi',
 'abbiecornish',
 'abduct',
 'abhishekbachchan',
 'abigailbreslin',
 'abil',
 'abl',
 'abo',
 'aboard',
 'aborigin',
 'abort',
 'abov',
 'abraham',
 'abram',
 'abroad',
 'absolut',
 'absurd',
 'abus',
 'abuse',
 'academ',
 'academi',
 'academy',
 'accept',
 'access',
 'accid',
 'accident',
 'acclaim',
 'accompani',
 'accomplish',
 'account',
 'accus',
 'ace',
 'achiev',
 'acid',
 'acquaint',
 'acquir',
 'acr',
 'act',
 'action',
 'actionhero',
 'activ',
 'activist',
 'activities',
 'activity',
 'actor',
 'actress',
 'actual',
 'ad',
 'ada',
 'adam',
 'adambeach',
 'adambrodi',
 'adamdriv',
 'adamgarcia',
 'adamgoldberg',
 'adammckay',
 'adams',
 'adamsandl',
 'adamscott',
 'adamshankman',
 'adapt',
 'add',
 'addict',
 'addiction',
 'addit',
 'addl',
 'adjust',
 'admir',
 'admiss',
 'admit',
 'adolesc',
 'adolf',
 'adolfhitl',
 'adopt',
 'adoptivefath',
 'ador',
 'adrenaline',
 'adrian',
 'adrianlyn',
 'adrienbrodi',
 'adrift',
 'adult',
 'adultanim',
 'adulteri',
 'adulthood',
 'adults',
 'advanc',
 'advantag',
 'advantage',
 'adventur',
 'adventure',
 'adventures',
 'adversari',
 'advertis',
 'advic',
 'advice',
 'advis',
 'affair',
 'affect',
 'affection',
 'afflict',
 'affluent',
 'afford',
 'afghanistan',
 'afraid',
 'africa',
 'african',
 'africanamerican',
 'afro',
 'aftercreditssting',
 'afterlif',
 'afterlife',
 'aftermath',
 'afternoon',
 'ag',
 'age',
 'agediffer',
 'agenc']
```

Building Movie Recommendation System

Cosine Similarity

```
# Import cosine_similarity to Calulate Distance between the Vectors
from sklearn.metrics.pairwise import cosine_similarity

# Calculate the Distance of all Movies with eachothers
similarity = cosine_similarity(dataVectors)

# Shape
similarity.shape
```

```
(4806, 4806)
```

```
# Get the Array of Arrays for all Movies
similarity
```

```
array([[1.        , 0.11013346, 0.06598588, ..., 0.05458155, 0.        ,
        0.        ],
       [0.11013346, 1.        , 0.06419407, ..., 0.05309942, 0.03241019,
        0.01785714],
       [0.06598588, 0.06419407, 1.        , ..., 0.02120949, 0.        ,
        0.        ],
       ...,
       [0.05458155, 0.05309942, 0.02120949, ..., 1.        , 0.03212463,
        0.05309942],
       [0.        , 0.03241019, 0.        , ..., 0.03212463, 1.        ,
        0.08102547],
       [0.        , 0.01785714, 0.        , ..., 0.05309942, 0.08102547,
        1.        ]])
```

```
# Similarity of 1st Movies with all
# Similarity with itself will be Index of 0

print("1st Sub Array Shape: ",similarity[0].shape)
print("\n", "-"*35, "\n Similarity of First Movie [Avatar] ","\n","-"*35,
"\n", similarity[0])
```

```
1st Sub Array Shape:  (4806,)

 -----------------------------------
 Similarity of First Movie [Avatar]
 -----------------------------------
 [1.         0.11013346 0.06598588 ... 0.05458155 0.         0.        ]
```

```
#Conver list of tuples and index created that help in sorting
sorted(list(enumerate(similarity[0])), reverse=True, key=lambda x:x[1])[1:6]
# from 1-5, 0 will be same movies
```

```
[(2409, 0.2442273269282182),
 (1216, 0.23235407831517718),
 (1089, 0.21346619809405212),
 (539, 0.20947289622255397),
 (507, 0.20349795765537632)]
```

Recommender

```
# Recommender Function to Return Movie Names Only
def recommend_movies_names(xMovie):
    # Get Index of given Movie
    movie_index = df_new_final[df_new_final["title"].str.lower() ==
xMovie.lower()].index[0]
    distances = similarity[movie_index]
    listofMovies = sorted(list(enumerate(distances)), reverse=True,
key=lambda x:x[1])[1:6]

    for i in listofMovies:
        print(df_new_final.iloc[i[0]]["title"])
```

```
# Recommended Movies Name with Example of 3-Movies
movies_list = {"Avatar", "Batman", "Titanic"}
for x in movies_list:
    print("-"*20, "Recommendations for [", x,"]", "-"*20)
    recommend_movies_names(x)
```

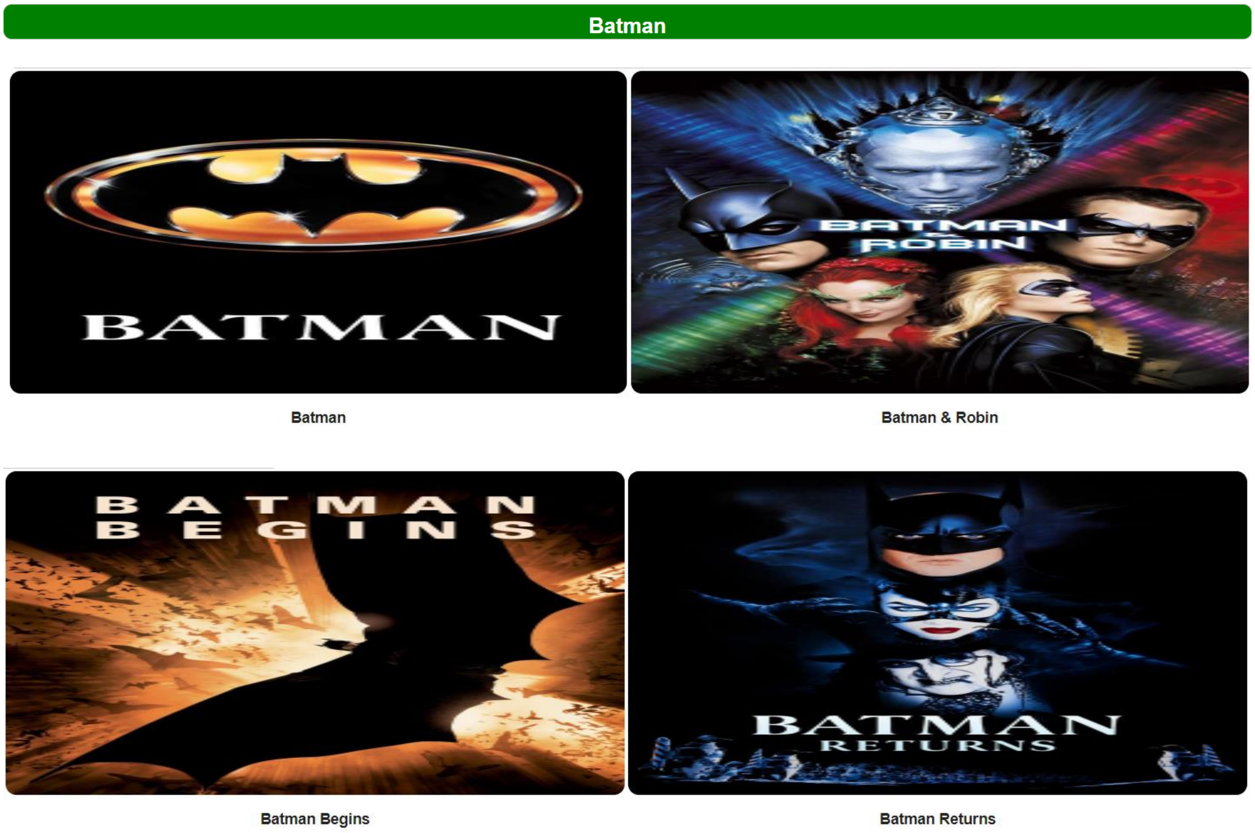
```
-------------------- Recommendations for [ Batman ] --------------------
Batman
Batman & Robin
Batman Begins
Batman Returns
The Dark Knight
-------------------- Recommendations for [ Avatar ] --------------------
Aliens
Aliens vs Predator: Requiem
Aliens in the Attic
Titan A.E.
Independence Day
-------------------- Recommendations for [ Titanic ] --------------------
Ghost Ship
The Chambermaid on the Titanic
Under the Same Moon
The Notebook
The Bounty
```

```
# Function that fetch Recommended Movies Poster Path
# with the help of API
def fetch_movies_poster(movieID):
    response = 
requests.get('https://api.themoviedb.org/3/movie/{}?api_key=c0bda0be71f7815fd
6ba2eb5f5c86fd8'.format(movieID) )
    data = response.json()
    posterPath = "http://image.tmdb.org/t/p/w500" + data['poster_path']
    return posterPath



# Movie Recomender Function,
# Also call movies poster function
# Return list of Recommended Movie Names and poster path
def getRecommended_movies_name(xMovie):
    # Get Index of given Movie
    movie_index = df_new_final[df_new_final["title"].str.lower() == 
xMovie.lower()].index[0]
    distances = similarity[movie_index]
    listofMovies = sorted(list(enumerate(distances)), reverse=True, 
key=lambda x:x[1])[1:6]

    recomended_movies = []
    movies_poster = []
    for i in listofMovies:
        recomended_movies.append(df_new_final.iloc[i[0]]["title"])
        # API required to fetch the poster_path
        movies_poster.append( 
fetch_movies_poster(df_new_final.iloc[i[0]]["id"]) )

    return recomended_movies, movies_poster


# Function that Display the Recommended Movie Names and Posters
def show_poster(sel_movie, mov_name, posterPath):
    display( HTML(f"""<div style="font-size:24px; font-weight:Bold; 
color:#fff; text-align:center; padding-top:8px; height:12%; width: 100%; 
border:1px solid #ccc; border-radius:10px; margin-top:10px; background-
color:green;">{sel_movie}</div>
                   <table>
                    <tr>
                        <td><img src={posterPath[0]} style='border-
radius:10px; width:575px; height:300px;'></td>
                        <td><img src={posterPath[1]} style='border-
radius:10px; width:575px; height:300px;'></td>
                        <td><img src={posterPath[2]} style='border-
radius:10px; width:575px; height:300px;'></td>
                        <td><img src={posterPath[3]} style='border-
radius:10px; width:575px; height:300px;'></td>
                        <td><img src={posterPath[4]} style='border-
radius:10px; width:575px; height:300px;'></td>
```

```
                    </tr>
                    <tr>
                        <td><p style="text-align:center; font-size:14px;
font-weight:bold">{mov_name[0]}</p></td>
                        <td><p style="text-align:center; font-size:14px;
font-weight:bold">{mov_name[1]}</p></td>
                        <td><p style="text-align:center; font-size:14px;
font-weight:bold">{mov_name[2]}</p></td>
                        <td><p style="text-align:center; font-size:14px;
font-weight:bold">{mov_name[3]}</p></td>
                        <td><p style="text-align:center; font-size:14px;
font-weight:bold">{mov_name[4]}</p></td>
                    </tr>
                   </table>""") )
```

```
# Show the Recommended Name and Poster of Movies in List
movies_list = {"Avatar", "Batman"}
for x in movies_list:
    names, poster = getRecommended_movies_name(x)
    show_poster(x, names, poster)
```

```
<IPython.core.display.HTML object>
```



Batman

Batman

Batman & Robin

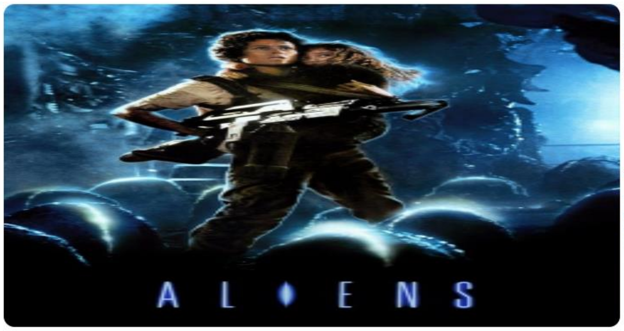
Batman Begins

Batman Returns

The Dark Knight

```
<IPython.core.display.HTML object>
```

Avatar



Aliens

Aliens vs Predator: Requiem

Aliens in the Attic

Titan A.E.

Independence Day

```
<IPython.core.display.HTML object>
```

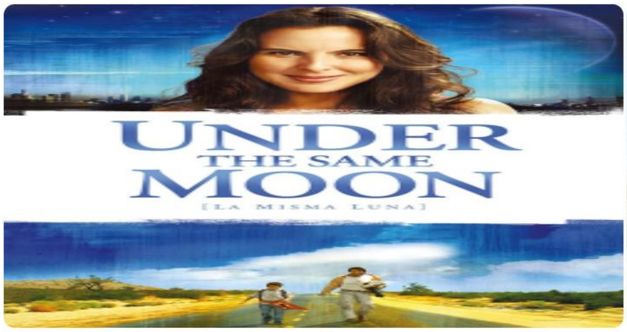
```
# Show the Recommended Movies of Selected Movie Name [Titanic]
movie_name = "Titanic"
names, poster = getRecommended_movies_name(movie_name)

# Code to Show the recommended movie names and poster
display( HTML(f"""<div style="font-size:24px; font-weight:Bold; color:#fff;
text-align:center; padding-top:8px; height:12%; width: 100%; border:1px solid
#ccc; border-radius:10px; margin-top:10px; background-
color:green;">{movie_name}</div>
                  <table>
                    <tr>
                        <td><img src={poster[0]} style='border-radius:10px;
width:575px; height:300px;'></td>
                        <td><img src={poster[1]} style='border-radius:10px;
width:575px; height:300px;'></td>
                        <td><img src={poster[2]} style='border-radius:10px;
width:575px; height:300px;'></td>
                        <td><img src={poster[3]} style='border-radius:10px;
width:575px; height:300px;'></td>
                        <td><img src={poster[4]} style='border-radius:10px;
width:575px; height:300px;'></td>
                    </tr>
                    <tr>
                        <td><p style="text-align:center; font-size:14px;
font-weight:bold">{names[0]}</p></td>
                        <td><p style="text-align:center; font-size:14px;
font-weight:bold">{names[1]}</p></td>
                        <td><p style="text-align:center; font-size:14px;
font-weight:bold">{names[2]}</p></td>
                        <td><p style="text-align:center; font-size:14px;
font-weight:bold">{names[3]}</p></td>
                        <td><p style="text-align:center; font-size:14px;
font-weight:bold">{names[4]}</p></td>
                    </tr>
                   </table>""") )
```

```
<IPython.core.display.HTML object>
```

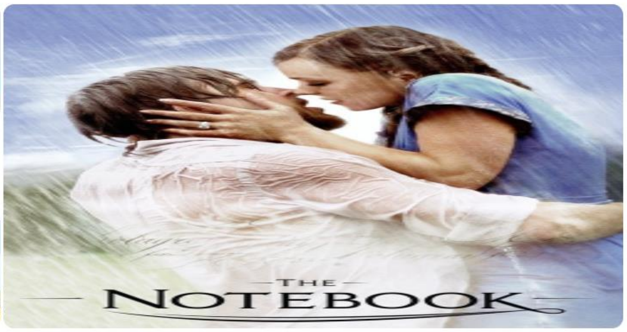
Titanic

Ghost Ship

The Chambermaid on the Titanic



Under the Same Moon



The Notebook

The Bounty

Result

The content-based recommendation is best in situations where there is known data to the item rather than the user as it analyses the attributes of items for generating predictions.

The function ‘getRecommended_movies_name()’ has been created to retrieve the similar movies. Users can enter the movie name in movie_name variable and execute the recommender function to get movie recommendations. The recommender returns 5 similar movies with their poster and name.

Conclusion

To conclude, a recommender system powered by content-based filtering performed using the cosine similarity algorithm can make better recommendations for users by suggesting them movies that have similar key features like popularity, genre, overview, tagline, tags etc.

Recommendations

This project is solely based on content based filtering, this project does not include collaborative based and neural network based filtering as these are beyond the scope of this project.

Some limitations of content based filtering are that it can only make recommendations based on existing interests of the user, it does not consider the fact that what do other users think of an item, thus low quality item recommendations may occur sometimes.

References

https://pdfs.semanticscholar.org/767e/ed55d61e3aba4e1d0e175d61f65ec0dd6c08.pdf

https://medium.com/@bkexcel2014/building-movie-recommender-systems-using-cosine-similarity-in-python-eff2d4e60d24

https://link.springer.com/chapter/10.1007/978-3-642-21793-7_63

https://www.kdnuggets.com/2019/04/building-recommender-system.html

https://towardsdatascience.com/how-to-build-from-scratch-a-content-based-movie-recommender-with-natural-language-processing-25ad400eb243

https://www.geeksforgeeks.org/using-countvectorizer-to-extracting-features-from-text/