Vatsa Nagaria 2019130041
Pranav Nair 2019130042
Vivek Poojari 2019130050

NextBinge

Problem Statement: A movie recommendation platform where the users can get recommendations based on two types of filtering: Demographic filtering and Content-based filtering(based on movie description and based on crew, cast, genres).

Theory:

Types of recommender systems

There are three primary types of recommender systems.

- Content-based filtering uses similarities in products, services, or content features, as well as information accumulated about the user to make recommendations.
- Collaborative filtering relies on the preferences of similar users to offer recommendations to a particular user.
- Demographic Filtering (DF) technique uses the demographic data of a user to determine which items may be appropriate for recommendation.

Content-based filtering:

- Content-based filtering is a type of recommender system that attempts to guess what a user may like based on that user's activity.
- Content-based filtering makes recommendations by using keywords and attributes assigned to objects in a database (e.g., items in an online marketplace) and matching them to a user profile. The user profile is created based on data derived from a user's actions, such as purchases, ratings (likes and dislikes), downloads, items searched for on a website and/or placed in a cart, and clicks on product links.
- For example, suppose you're recommending accessories to a user that just purchased a smartphone from your website and has previously bought smartphone accessories. Aside from keywords such as the smartphone manufacturer, make, and model, the user profile indicates prior purchases include phone holders with sleeves for credit cards. Based on this information, the recommender system may suggest similar phone holders for the new phone with attributes such as an RFID blocking fabric layer to help prevent unauthorized credit card scanning. In this example, the user would expect recommendations for similar phone holders, but the RFID blocking feature may be something they didn't expect yet appreciate.

Code:

```
#!/usr/bin/env python

# coding: utf-8



# In[4]:




import pandas as pd

import numpy as np

df1=pd.read_csv('tmdb_5000_credits.csv')

df2=pd.read_csv('tmdb_5000_movies.csv')




# In[5]:




df1.columns = ['id','tittle','cast','crew']

df2= df2.merge(df1,on='id')




# In[6]:
```

```
df2.head(5)
```

```
# In[7]:
```

```
C= df2['vote_average'].mean()
C
```

```
# In[8]:
```

```
m= df2['vote_count'].quantile(0.9)
m
```

```
# In[9]:
```

```
q_movies = df2.copy().loc[df2['vote_count'] >= m]
q_movies.shape
```

```
# In[10]:



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)



# In[11]:



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



# In[12]:



q_movies = q_movies.sort_values('score', ascending=False)

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

```
# In[13]:



from ast import literal_eval


features = ['cast', 'crew', 'keywords', 'genres']
for feature in features:
    df2[feature] = df2[feature].apply(literal_eval)




# In[14]:



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



# In[15]:
```

```
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[16]:



df2['director'] = df2['crew'].apply(get_director)

features = ['cast', 'keywords', 'genres']
for feature in features:
    df2[feature] = df2[feature].apply(get_list)



# In[17]:
```

```
df2[['title', 'cast', 'director', 'keywords', 'genres']].head(3)
```

```
# In[18]:
```

```
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[19]:
```

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

for feature in features:
```

```
    df2[feature] = df2[feature].apply(clean_data)




# In[20]:




def create_soup(x):

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

df2['soup'] = df2.apply(create_soup, axis=1)




# In[21]:




df2['soup']




# In[22]:




from sklearn.feature_extraction.text import CountVectorizer


count = CountVectorizer(stop_words='english')
```

```
count_matrix = count.fit_transform(df2['soup'])



# In[23]:



from sklearn.metrics.pairwise import cosine_similarity


cosine_sim2 = cosine_similarity(count_matrix, count_matrix)



# In[24]:



df2 = df2.reset_index()

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



# In[54]:



def get_recommendations(title, 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 df2[['title']].iloc[movie_indices]



# In[55]:



get_recommendations('The Dark Knight Rises', cosine_sim2)



# In[56]:



get_recommendations('Pirates of the Caribbean: The Curse of the Black Pearl', cosine_sim2)
```

```
# PLOT BASED RECOMMENDER


# In[28]:



df2['overview'].head(5)



# In[29]:



from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(stop_words='english')

df2['overview'] = df2['overview'].fillna('')

tfidf_matrix = tfidf.fit_transform(df2['overview'])

tfidf_matrix.shape
```

```
# In[30]:



from sklearn.metrics.pairwise import linear_kernel

cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)



# In[31]:



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



# In[57]:



get_recommendations('The Dark Knight Rises',cosine_sim)



# In[58]:
```

```
get_recommendations('The Avengers',cosine_sim)
```

```
# In[45]:
```

```
import pickle
```

```
# In[51]:
```

```
pickle.dump(df2[['id','title']],open('movies_list.pkl','wb'))
```

```
# In[52]:
```

```
pickle.dump(cosine_sim2,open('similarity.pkl','wb'))
```

```
# In[53]:
```

```
pickle.dump(cosine_sim,open('similarity1.pkl','wb'))



# In[61]:



pickle.dump(q_movies[['title', 'id']],open('top_movies.pkl', 'wb'))



# In[ ]:
```

Demo Screenshots:

Recommendations based on movie cast, director, genres-

Pirates of the Caribbean: At World's End

Show recommendations based on movie cast, director, genres

Show recommendations based on movie plot

Show top rated movies

Pirates of the (

Pirates of the (

The Lone Ranger

Pirates of the (

The Mummy: Tomb

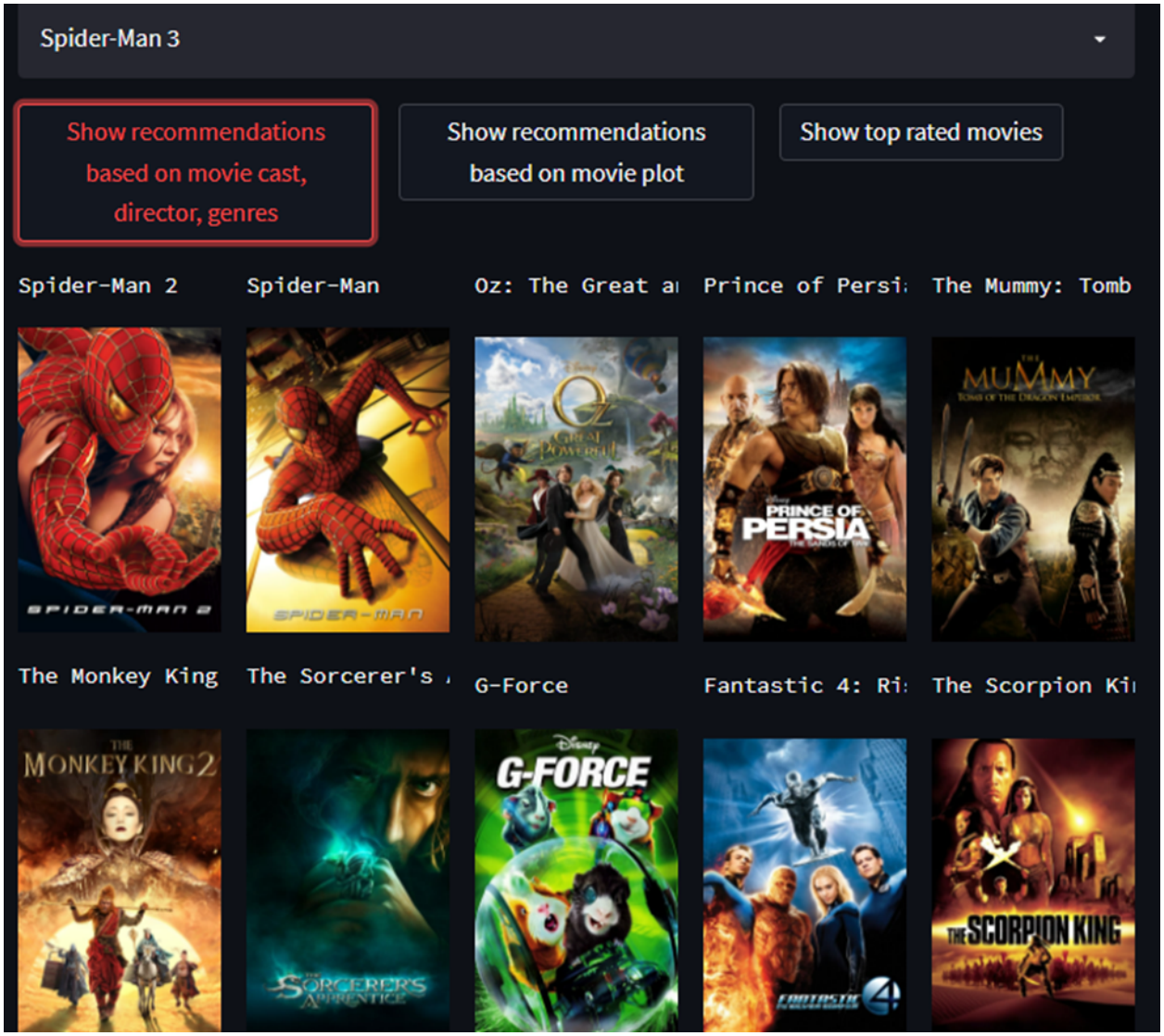
The Monkey King

Waterworld

The Sorcerer's

G-Force

Fantastic 4: Ri



Recommendations based on plot-

Search or select a movie

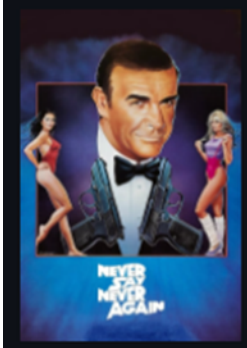
Spectre

Show recommendations based on movie cast, director, genres

Show recommendations based on movie plot

Show top rated movies

Never Say Never



From Russia wit

Thunderball

Safe Haven

Quantum of Sola

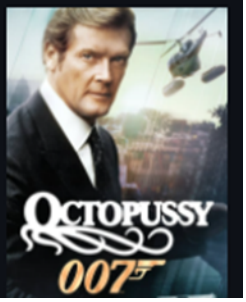
Dr. No

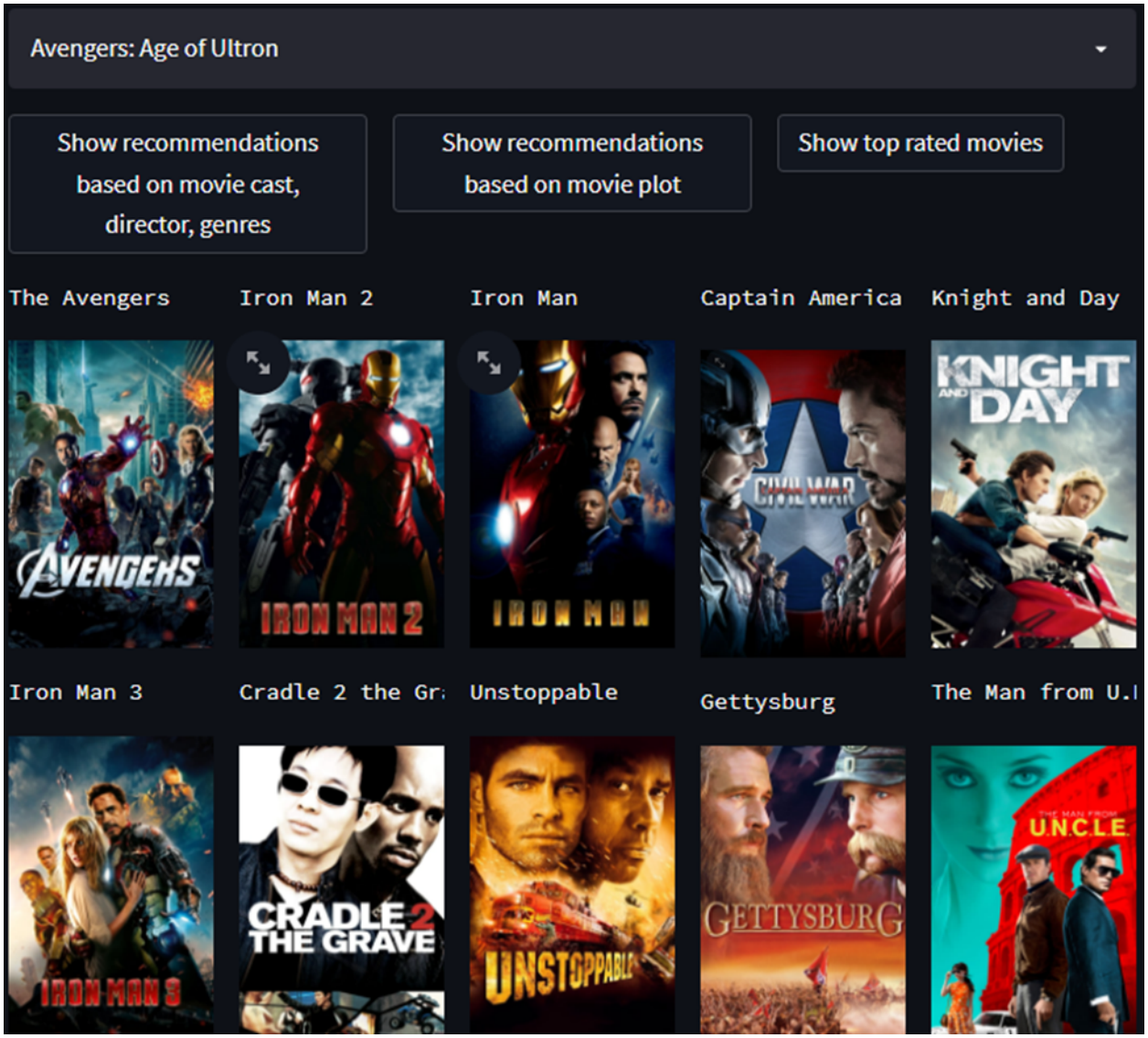
Skyfall

Dance Flick

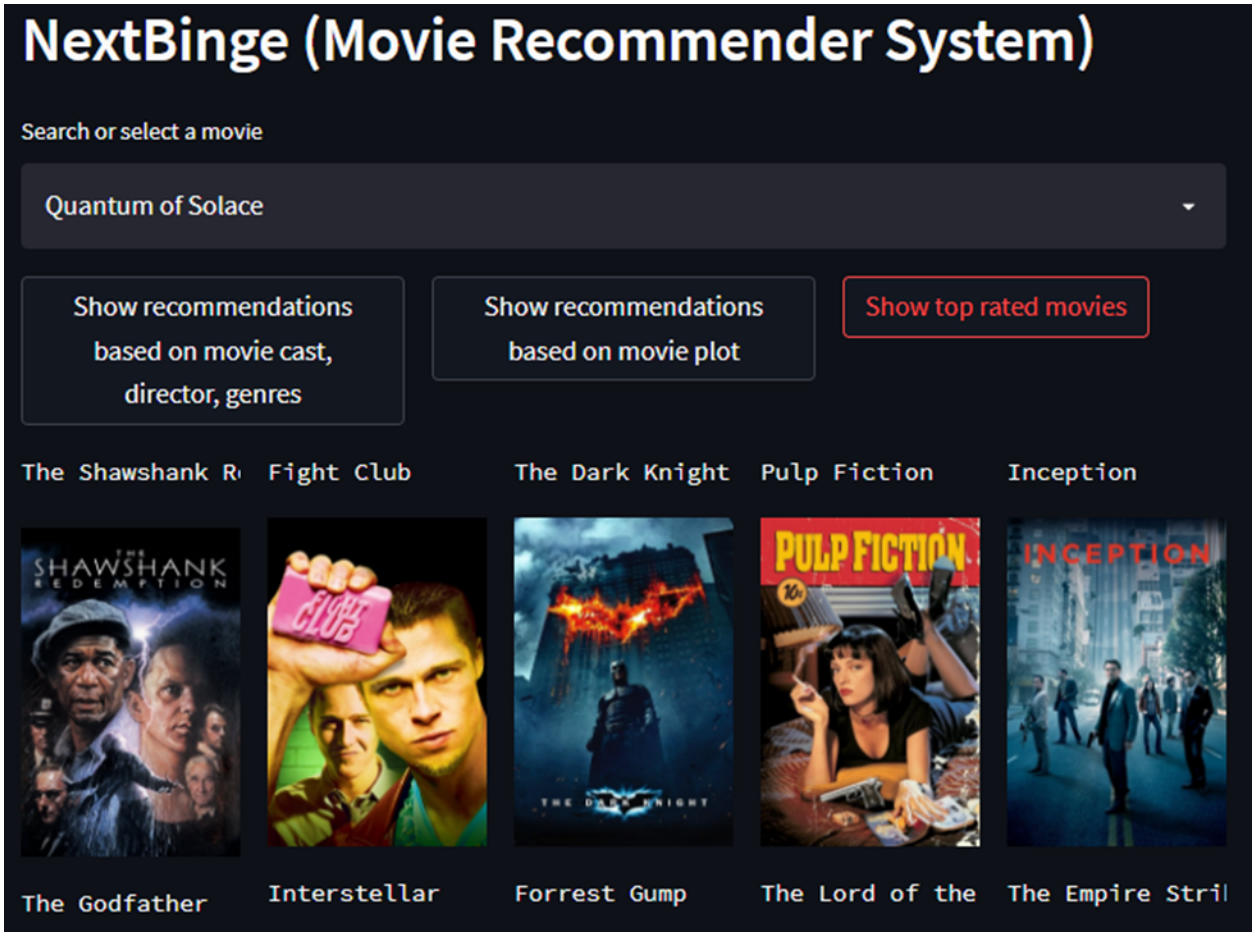
Diamonds Are Fo

Octopussy





Top rated movies-



Conclusion:

For our mini project, we built a movie recommendation system for which we used demographic filtering and content-based filtering technique. Demographic filtering technique uses the demographic data of a user to determine which items may be appropriate for recommendation, while Content-based filtering uses similarities in products, services, or content features, as well as information accumulated about the user to make recommendations. In Content based filtering, we further used 2 different methods to filter out the results. One method was to show recommendations based on the movie cast, directors and genre. To do this we created a single string containing the cast,directors, and genre which was named soup, for all the movies and then found out the cosine similarity between them. Here we found out that the movies which should have low similarity were getting high similarity, the reason for this was because of certain cast members having the same first name or last name or other similar words, It would match the similar name and may result in we getting high similarity so while making the string we ensured that we removed the spaces between names, genres. For eg. names like Chris Evans and Chris Hemsworth have “Chris” in common and there were many such examples. The second method was to show recommendations based on the movie plot. Here we do the similar thing, we vectorize the movie description and then find the cosine similarity. So given the input of the name of the movie, the model would find the movies with high

similarity and return the top ten movies with high similarity. In Demographic filtering, we find a score for each movie and then sort them in descending order and display the top ten movies. This score is calculated using the votes a movie has received , average vote count, popularity of the movie and a value ‘m’ which denotes the minimum number of votes a movie requires to be selected to be considered as a recommendation. The variable ‘m’ is important since a movie with 3 votes can have high rating and should not be considered as a high rated movie since it has very few votes and it may not be better than a movie with a lower rated movie with high votes. To correct this we took the value of ‘m’ such that the number of votes for each movie is more than 90% of the movies in the dataset. Only movies which satisfied this value of m were considered and then the score of each movie was calculated. At the end, we came across Streamlit, which we used to create a single page where the user can enter the movie name and get recommendations by clicking on the buttons below the input bar, and the top ten recommendations that are returned are displayed below.