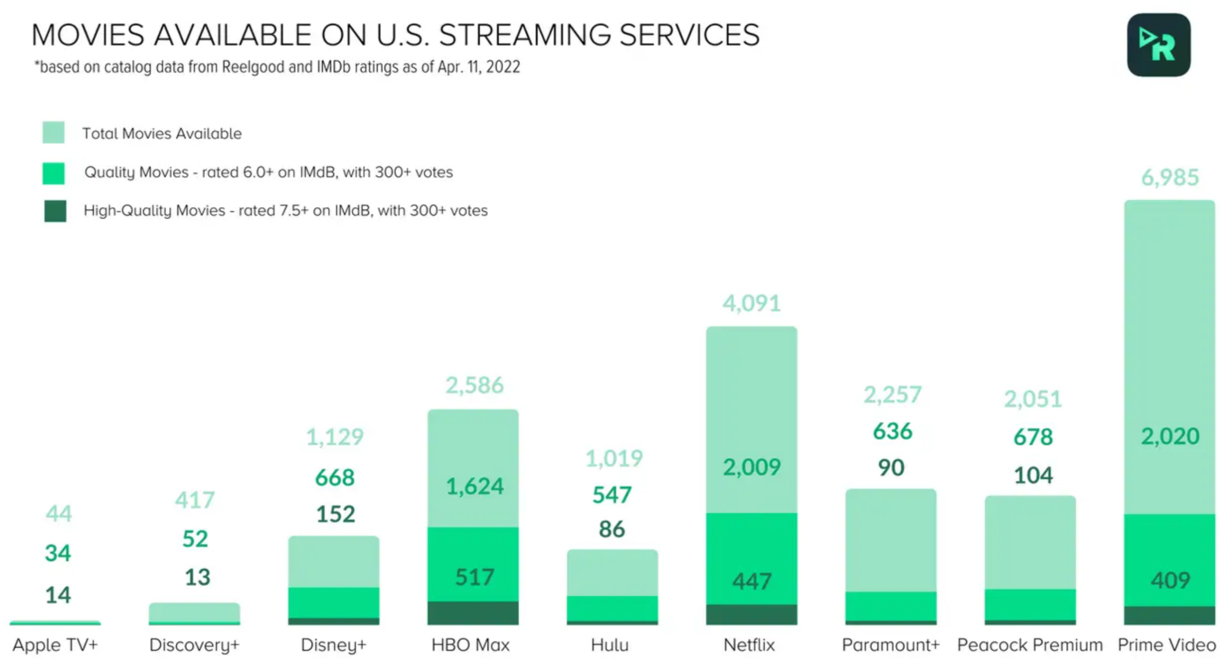
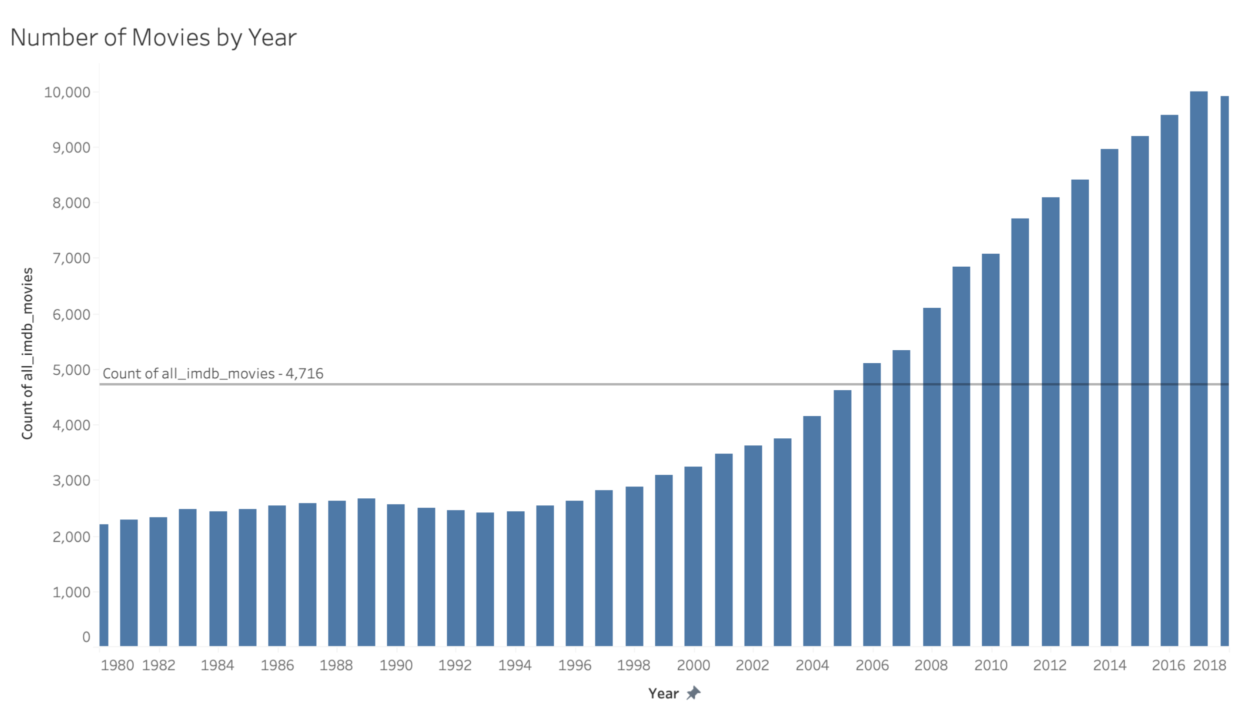
Movie Recommender System

Abstract

We are living in the era of abundance. There are an more than 18 thousand movies available on the top 5 streaming services and this number keeps growing every year.



Each year there an estimated 5K movies released globally and this trend is growing. (Source: IMDB Dataset for 1980 - 2020, Note this visualization was built in Tableau)



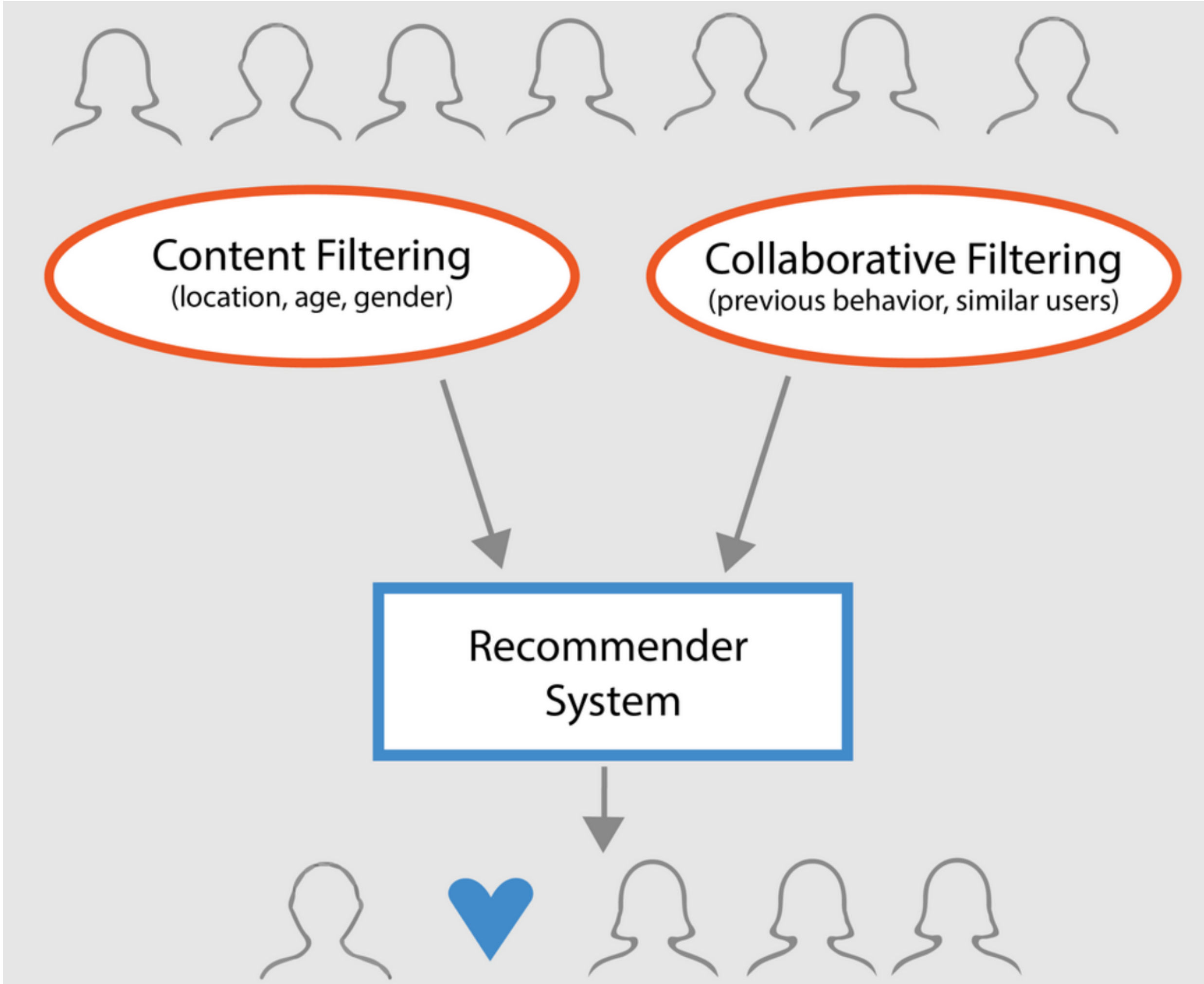
In this study, we examined 40K movies, with features such as Genre, Keywords, Movie Popularity (based on Ratings and number of votes), Director and Cast of the movie to build a content based recommender system. The system uses a combination of feature extraction, vectorization, and cosine similarity to compare the similarity between movies and provide recommendations based on user preferences. The advantage of the proposed recommender system is the ability to recommend niche movies that may not be popular (since factors such as genre, cast and keywords are given a huge weight in the model). Overall, the proposed content-based movie recommender system provides a promising approach for personalized and accurate movie recommendations.

Introduction

About Movie Recommender Systems: With the rapid growth of online movie platforms, the amount of available movie content has become overwhelming, making it difficult for users to find movies that match their preferences. This has resulted in an increasing demand for movie recommendation systems that can provide personalized recommendations to users. Movie recommender systems have become a popular research topic in the field of data science and machine learning due to their ability to help users discover new movies that they may enjoy and increase user engagement on online movie platforms.

Types of Recommender Systems: There are three main types of recommender systems: collaborative filtering, content-based filtering, and hybrid recommender systems.

1. **Collaborative Filtering:** This type of recommender system recommends items based on the preferences and behavior of similar users. Collaborative filtering algorithms analyze user data, such as ratings and purchases, and recommend items to users based on the preferences of similar users. Collaborative filtering is effective in finding items that are popular and have high ratings, but it can suffer from the "cold-start problem" when there are new users or new items without sufficient data.
2. **Content-Based Filtering:** This type of recommender system recommends items based on the features or attributes of the items themselves. Content-based filtering algorithms analyze the features of items, such as genre, director, and cast in the case of movies, and recommend similar items to users based on their past preferences. Content-based filtering is effective in finding niche or less popular items, but it may not be able to capture the user's changing preferences over time.
3. **Hybrid Recommender Systems:** This type of recommender system combines both collaborative filtering and content-based filtering to overcome the limitations of each method. Hybrid recommender systems leverage the strengths of each method to provide more accurate and personalized recommendations to users. Hybrid recommender systems can be designed in many ways, such as using collaborative filtering to provide initial recommendations and content-based filtering to refine them.



In this study, we built a Content Based Filtering Recommender System. Content-based recommender systems have several advantages over other types of recommender systems. They are able to handle cold-start problems, where a user has no or limited rating history, by recommending movies based on their content attributes. Additionally, they can recommend niche or less popular movies to users based on their content, which may not be possible with collaborative filtering-based systems that rely on user ratings and popularity.

The proposed a content-based movie recommender system utilizes movie features such as genre, director, cast, keywords and movie popularity to provide personalized recommendations to users. The proposed system has several advantages over other types of recommender systems and provides a promising approach for personalized and accurate movie recommendations.

Building out the dataset

Step 1 - Building out the Base Dataset

I built out the base of the IMDb movie dataset by downloading the datasets from the IMDb website (https://datasets.imdbws.com/ (https://datasets.imdbws.com/)).
The final output of the base dataset was ~275K movies with the following features

- movie_id
- primaryTitle
- originalTitle
- isAdult
- Year
- runtimeMinutes
- genres
- averageRating
- numVotes
- director_name

Step 2 - Adding Additional Features

I built out another dataset based on the following datasets from Kaggle (https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset (https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset))

- **movies_metadata.csv:** The main Movies Metadata file. Contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.
- **keywords.csv:** Contains the movie plot keywords for our MovieLens movies. Available in the form of a stringified JSON Object.
- **credits.csv:** Consists of Cast and Crew Information for all our movies. Available in the form of a stringified JSON Object.

The final output of this dataset was ~45K movies with the following features:

- imdb_id
- original_title
- original_language
- overview
- keywords
- cast

Step 3 - Combining the Datasets & Data Cleaning

I joined the two datasets from Step 1 & 2 based on the the IMDb ID.
The final output of this step was a combined dataset with ~40K movie records with the following features:

- movie_id
- primaryTitle
- originalTitle
- isAdult
- Year
- runtimeMinutes
- genres
- averageRating
- numVotes
- director_name
- original_title
- original_language
- overview
- keywords
- cast

Step 4 - Data Cleaning

I cleaned the cast & keywords fields to take the top 3 cast members and top 5 keywords for the movie
I added a feature for movie popularity based on averageRatings and numVotes

The final output of this step was my final dataset for the project with ~40K movie records with the following features:

- title
- Year
- runtimeMinutes
- genres
- moviePopularity (based on averageRatings and numVotes)
- director_name
- cast (top 3 actors of the movie)
- keywords (top 5 keywords for the movie)
- overview (Storyline)
- original_language

Step 1: Building out the Base Table: (~275K Movies)

I downloaded the following files from the IMDb website ([https://datasets.imdbws.com/ (https://datasets.imdbws.com/)](https://datasets.imdbws.com/)).

title.basics.tsv.gz - Contains the following information for titles:

- tconst (string) - alphanumeric unique identifier of the title
- titleType (string) – the type/format of the title (e.g. movie, short, tvseries, tvepisode, video, etc)
- primaryTitle (string) – the more popular title / the title used by the filmmakers on promotional materials at the point of release
- originalTitle (string) - original title, in the original language
- isAdult (boolean) - 0: non-adult title; 1: adult title
- startYear (YYYY) – represents the release year of a title. In the case of TV Series, it is the series start year
- endYear (YYYY) – TV Series end year. '\N' for all other title types
- runtimeMinutes – primary runtime of the title, in minutes
- genres (string array) – includes up to three genres associated with the title

    Note: I filtered on titleType to get only movie data

title.ratings.tsv.gz – Contains the IMDb rating and votes information for titles:

- tconst (string) - alphanumeric unique identifier of the title
- averageRating – weighted average of all the individual user ratings
- numVotes - number of votes the title has received

    Note: I did an inner join between the title.basics and title.ratings because I need the rating for each movie

title.crew.tsv.gz – Contains the director and writer information for all the titles in IMDb. Fields include:

- tconst (string) - alphanumeric unique identifier of the title
- directors (array of nconsts) - director(s) of the given title
- writers (array of nconsts) – writer(s) of the given title

    Note: I only added the directors to the movie dataset not the writers

name.basics.tsv.gz – Contains the following information for names:

- nconst (string) - alphanumeric unique identifier of the name/person
- primaryName (string)– name by which the person is most often credited
- birthYear – in YYYY format
- deathYear – in YYYY format if applicable, else '\N'
- primaryProfession (array of strings)– the top-3 professions of the person

- knownForTitles (array of tconsts) – titles the person is known for

  Note: I did a left join with crew to get the name of the director, I only need the names from this dataset, not the other columns

Importing the datasets

In [1]:
```
import pandas as pd
```

In [2]:
```
title = pd.read_csv('title.basics.tsv',sep= '\t')
```

```
/Users/asadimam270/opt/anaconda3/lib/python3.7/site-packages/IPython/cor
e/interactiveshell.py:3063: DtypeWarning: Columns (4,5) have mixed types.
Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [3]:
```
title.shape
```

Out[3]: (8710313, 9)

Note: We have 8.7 million records in the title dataset

In [4]:
```
title.head()
```

Out[4]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	
0	tt0000001	short	Carmencita	Carmencita	0	1894	\N	1	
1	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5	
2	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4	Animat
3	tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	12	
4	tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	1	

Importing the rating dataset

In [5]:
```
rating = pd.read_csv('title.ratings.tsv',sep= '\t')
```

In [6]:
```
rating.shape
```

Out[6]: (1217067, 3)

Note: We have 1.2 million records in the rating dataset.

In [7]:
```
rating.head()
```
Out[7]:

	tconst	averageRating	numVotes
0	tt0000001	5.7	1863
1	tt0000002	6.0	243
2	tt0000003	6.5	1632
3	tt0000004	6.0	158
4	tt0000005	6.2	2458

We will do an inner join between the rating and title dataset on tconst (which is the primary key for each record) because we need the rating for each record. Our new dataset will have 1.2 million records

In [8]:
```
title_rating = title.merge(rating, on='tconst', how='inner')
```
In [9]:
```
title_rating
```
Out[9]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes
0	tt0000001	short	Carmencita	Carmencita	0	1894	\N	1
1	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5
2	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4
3	tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	12
4	tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	1
...	...	...	...	...	...	...	...	...
1217062	tt9916690	tvEpisode	Horrid Henry Delivers the Milk	Horrid Henry Delivers the Milk	0	2012	\N	10
1217063	tt9916720	short	The Nun 2	The Nun 2	0	2019	\N	10
1217064	tt9916730	movie	6 Gunn	6 Gunn	0	2017	\N	116
1217065	tt9916766	tvEpisode	Episode #10.15	Episode #10.15	0	2019	\N	43
1217066	tt9916778	tvEpisode	Escape	Escape	0	2019	\N	\N

1217067 rows × 11 columns

Now we will import the other datasets (crew and names) to add director name for each record

In [10]:
```
crew = pd.read_csv('title.crew.tsv',sep= '\t')
```

In [11]:
```
crew.shape
```
Out[11]:
```
(8710313, 3)
```
In [12]:
```
crew.head()
```
Out[12]:

	tconst	directors	writers
0	tt0000001	nm0005690	\N
1	tt0000002	nm0721526	\N
2	tt0000003	nm0721526	\N
3	tt0000004	nm0721526	\N
4	tt0000005	nm0005690	\N

We will only add the director for each record (we can add the writer as well if needed)

In [13]:
```
directors = crew[['tconst', 'directors']].copy()
```
In [14]:
```
directors.head()
```
Out[14]:

	tconst	directors
0	tt0000001	nm0005690
1	tt0000002	nm0721526
2	tt0000003	nm0721526
3	tt0000004	nm0721526
4	tt0000005	nm0005690

Note: for directors we have the director id not the names, we will join with the names dataset to get the director name

In [15]:
```
names = pd.read_csv('name.basics.tsv',sep= '\t')
```
In [16]:
```
names.shape
```
Out[16]:
```
(11412150, 6)
```

In [17]:
```
names.head()
```
Out[17]:

	nconst	primaryName	birthYear	deathYear	primaryProfession	
0	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt0053137,tt007
1	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt0037382,tt011
2	nm0000003	Brigitte Bardot	1934	\N	actress,soundtrack,music_department	tt0057345,tt005
3	nm0000004	John Belushi	1949	1982	actor,soundtrack,writer	tt0077975,tt008
4	nm0000005	Ingmar Bergman	1918	2007	writer,director,actor	tt0060827,tt005

Since the names dataset has the name nconst to represent the id of the person and the director table uses the id of director in the directors column, we will rename it to nconst to match with the names dataset

In [18]:
```
directors.rename(columns={"directors": "nconst"},inplace = True)
```

Now we will join the directors dataset with the names dataset to get the names of the directors

In [19]:
```
directors_names = directors.merge(names, on='nconst', how='left')
```

In [20]:
```
directors_names.head()
```
Out[20]:

	tconst	nconst	primaryName	birthYear	deathYear	primaryProfession	
0	tt0000001	nm0005690	William K.L. Dickson	1860	1935	cinematographer,director,producer	tt1
1	tt0000002	nm0721526	Émile Reynaud	1844	1918	director,animation_department,writer	tt000
2	tt0000003	nm0721526	Émile Reynaud	1844	1918	director,animation_department,writer	tt000
3	tt0000004	nm0721526	Émile Reynaud	1844	1918	director,animation_department,writer	tt000
4	tt0000005	nm0005690	William K.L. Dickson	1860	1935	cinematographer,director,producer	tt1

Since we only need the names we will drop everything else (We can change this if other columns are important

In [21]:
```
directors_names.drop(['birthYear', 'deathYear',
        'primaryProfession', 'knownForTitles'], axis=1,inplace = True)
```

We will rename primaryName to director_name

In [22]:
```
directors_names.rename(columns={"primaryName": "director_name"},inplace
```

In [23]:
```
directors_names.head()
```
Out[23]:

	tconst	nconst	director_name
0	tt0000001	nm0005690	William K.L. Dickson
1	tt0000002	nm0721526	Émile Reynaud
2	tt0000003	nm0721526	Émile Reynaud
3	tt0000004	nm0721526	Émile Reynaud
4	tt0000005	nm0005690	William K.L. Dickson

Now we will add the director name to the title_rating dataset

In [24]:
```
title_rating_directors = title_rating.merge(directors_names, on='tconst
```
In [25]:
```
title_rating_directors.shape
```
Out[25]:
```
(1217067, 13)
```
In [26]:
```
title_rating_directors.head()
```
Out[26]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	
0	tt0000001	short	Carmencita	Carmencita	0	1894	\N	1	
1	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5	
2	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4	Animat
3	tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	12	
4	tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	1	

Checking how many different titleTypes we have

In [27]:
```
title_rating_directors.titleType.unique()
```
Out[27]:
```
array(['short', 'movie', 'tvEpisode', 'tvSeries', 'tvShort', 'tvMovie',
       'tvMiniSeries', 'tvSpecial', 'video', 'videoGame'], dtype=object)
```

In [28]:
```
title_rating_directors.titleType.value_counts()
```
Out[28]:
```
tvEpisode       585985
movie           275128
short           139524
tvSeries         80297
video            49228
tvMovie          48356
tvMiniSeries     12987
videoGame        12926
tvSpecial        10225
tvShort           2411
Name: titleType, dtype: int64
```

Our dataset had mostly tvEpisodes,shorts etc. We will filter our dataset for movie only

In [29]:
```
movies = title_rating_directors[title_rating_directors.titleType == 'mo
```
In [30]:
```
movies.shape
```
Out[30]:
```
(275128, 13)
```
In [31]:
```
movies.head()
```
Out[31]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	
339	tt0000502	movie	Bohemios	Bohemios	0	1905	\N	100	
373	tt0000574	movie	The Story of the Kelly Gang	The Story of the Kelly Gang	0	1906	\N	70	Actic
382	tt0000591	movie	The Prodigal Son	L'enfant prodigue	0	1907	\N	90	
396	tt0000615	movie	Robbery Under Arms	Robbery Under Arms	0	1907	\N	\N	
404	tt0000630	movie	Hamlet	Amleto	0	1908	\N	\N	

Checking if we need the endYear column

In [32]:
```
movies.endYear.value_counts()
```
Out[32]:
```
\N    275128
Name: endYear, dtype: int64
```

Since all the values are missing. We can drop the endYear column

In [33]:
```
movies.drop('endYear', axis=1,inplace = True)
```

In [34]:
```
movies.titleType.value_counts()
```

Out[34]:
```
movie    275128
Name: titleType, dtype: int64
```

Since the titleType is only movie we can delete this column as well

In [35]:
```
movies.drop('titleType', axis=1,inplace = True)
```

Renaming the columns of the movie dataset

In [36]:
```
movies.columns
```

Out[36]:
```
Index(['tconst', 'primaryTitle', 'originalTitle', 'isAdult', 'startYear',
       'runtimeMinutes', 'genres', 'averageRating', 'numVotes', 'nconst',
       'director_name'],
      dtype='object')
```

In [37]:
```
movies.rename(columns={'tconst':'movie_id','startYear':'Year','nconst':
```

In [38]:
```
movies.head()
```

Out[38]:

	movie_id	primaryTitle	originalTitle	isAdult	Year	runtimeMinutes	genres	a
339	tt0000502	Bohemios	Bohemios	0	1905	100	\N	
373	tt0000574	The Story of the Kelly Gang	The Story of the Kelly Gang	0	1906	70	Action,Adventure,Biography	
382	tt0000591	The Prodigal Son	L'enfant prodigue	0	1907	90	Drama	
396	tt0000615	Robbery Under Arms	Robbery Under Arms	0	1907	\N	Drama	
404	tt0000630	Hamlet	Amleto	0	1908	\N	Drama	

Reseting the index of the movie df

In [39]:
```
movies = movies.reset_index(drop=True)
```

In [41]:
```
movies
```

Out[41]:

	movie_id	primaryTitle	originalTitle	isAdult	Year	runtimeMinutes	genre
0	tt0000502	Bohemios	Bohemios	0	1905	100	\N
1	tt0000574	The Story of the Kelly Gang	The Story of the Kelly Gang	0	1906	70	Action,Adventure,Biography
2	tt0000591	The Prodigal Son	L'enfant prodigue	0	1907	90	Drama
3	tt0000615	Robbery Under Arms	Robbery Under Arms	0	1907	\N	Drama
4	tt0000630	Hamlet	Amleto	0	1908	\N	Drama
...	...	...	...	...	...	...	..
275123	tt9916270	Il talento del calabrone	Il talento del calabrone	0	2020	84	Thrille
275124	tt9916362	Coven	Akelarre	0	2020	92	Drama,History
275125	tt9916428	The Secret of China	Hong xing zhao yao Zhong guo	0	2019	\N	Adventure,History,Wa
275126	tt9916538	Kuambil Lagi Hatiku	Kuambil Lagi Hatiku	0	2019	123	Drama
275127	tt9916730	6 Gunn	6 Gunn	0	2017	116	\N

275128 rows × 11 columns

saving the output of this step as a csv file

In [41]:
```
movies.to_csv(r'movie_dataset.csv')
```

Step 2: Adding additional features: (~46K Movies)

In [11]:
```
import pandas as pd
```

link for the dataset - [https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset (https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset)](https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset)

This dataset consists of the following files:

movies_metadata.csv: The main Movies Metadata file. Contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.

keywords.csv: Contains the movie plot keywords for our MovieLens movies. Available in the form of a stringified JSON Object.

credits.csv: Consists of Cast and Crew Information for all our movies. Available in the form of a stringified JSON Object.

In [12]:
```
keywords = pd.read_csv('keywords.csv')
```

In [13]:
```
keywords
```

Out[13]:

	id	keywords
0	862	[{'id': 931, 'name': 'jealousy'}, {'id': 4290,...
1	8844	[{'id': 10090, 'name': 'board game'}, {'id': 1...
2	15602	[{'id': 1495, 'name': 'fishing'}, {'id': 12392...
3	31357	[{'id': 818, 'name': 'based on novel'}, {'id':...
4	11862	[{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n...
...	...	...
46414	439050	[{'id': 10703, 'name': 'tragic love'}]
46415	111109	[{'id': 2679, 'name': 'artist'}, {'id': 14531,...
46416	67758	[]
46417	227506	[]
46418	461257	[]

46419 rows × 2 columns

In [ ]:
```
movies_metadata = pd.read_csv('movies_metadata.csv')
```

In [17]:
```
movies_metadata
```

Out[17]:

	adult	belongs_to_collection	budget	genres	homepage	i
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	86
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	884
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	NaN	1560
3	False	NaN	16000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	3135
4	False	{'id': 96871, 'name': 'Father of the Bride Col...	0	[{'id': 35, 'name': 'Comedy'}]	NaN	1186
...	...	...	...	...	...	.
45461	False	NaN	0	[{'id': 18, 'name': 'Drama'}, {'id': 10751, 'n...	http://www.imdb.com/title/tt6209470/	43905
45462	False	NaN	0	[{'id': 18, 'name': 'Drama'}]	NaN	11110
45463	False	NaN	0	[{'id': 28, 'name': 'Action'}, {'id': 18, 'nam...	NaN	6775
45464	False	NaN	0	[]	NaN	22750
45465	False	NaN	0	[]	NaN	46125

45466 rows × 24 columns

Joining the Movies Metadata with Keywords Dataset

In [18]:
```
new_df = movies_metadata.merge(keywords, on='id', how='left')
```

Adding Credits Datasets (to get Cast and Crew Members for the movies)

In [6]:
```
credits = pd.read_csv('credits.csv')
```

In [21]:
```
credits['id'] = credits['id'].astype(str)
```

Joining all 3 Datasets

In [24]:
```
new_df_1 = new_df.merge(credits, on='id', how='left')
```

Creating a final dataset for this step with only the relevant fields

In [29]:
```
new_movie_df = new_df_1[['imdb_id','original_title','original_language'
```

In [30]:
```
new_movie_df['imdb_id'].dropna(inplace= True)
```

In [31]:
```
new_movie_df
```

Out[31]:

	imdb_id	original_title	original_language	overview	keywords	cast
0	tt0114709	Toy Story	en	Led by Woody, Andy's toys live happily in his ...	[{'id': 931, 'name': 'jealousy'}, {'id': 4290,...	[{'cast_id': 14, 'character': 'Woody (voice)',...
1	tt0113497	Jumanji	en	When siblings Judy and Peter discover an encha...	[{'id': 10090, 'name': 'board game'}, {'id': 1...	[{'cast_id': 1, 'character': 'Alan Parrish', '...
2	tt0113228	Grumpier Old Men	en	A family wedding reignites the ancient feud be...	[{'id': 1495, 'name': 'fishing'}, {'id': 12392...	[{'cast_id': 2, 'character': 'Max Goldman', 'c...
3	tt0114885	Waiting to Exhale	en	Cheated on, mistreated and stepped on, the wom...	[{'id': 818, 'name': 'based on novel'}, {'id':...	[{'cast_id': 1, 'character': "Savannah 'Vannah...
4	tt0113041	Father of the Bride Part II	en	Just when George Banks has recovered from his ...	[{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n...	[{'cast_id': 1, 'character': 'George Banks', '...
...	...	...	...	...	...	...
46627	tt6209470	رگ خواب	fa	Rising and falling between a man and woman.	[{'id': 10703, 'name': 'tragic love'}]	[{'cast_id': 0, 'character': '', 'credit_id': ...
46628	tt2028550	Siglo ng Pagluluwal	tl	An artist struggles to finish his work while a...	[{'id': 2679, 'name': 'artist'}, {'id': 14531,...	[{'cast_id': 1002, 'character': 'Sister Angela...
46629	tt0303758	Betrayal	en	When one of her hits goes wrong, a professiona...	[]	[{'cast_id': 6, 'character': 'Emily Shaw', 'cr...
46630	tt0008536	Satana likuyushchiy	en	In a small town live two brothers, one a minis...	[]	[{'cast_id': 2, 'character': '', 'credit_id': ...
46631	tt6980792	Queerama	en	50 years after decriminalisation of homosexual...	[]	[]

46632 rows × 6 columns

saving the output of this step as a csv file

In [32]:
```
new_movie_df.to_csv('movie_metadata_imdb_updated.csv')
```

Step 3: Combining the Datasets: (~40K Movies)

Importing the Base Dataset from Step 1

In [1]:
```
import pandas as pd
```

In [2]:
```
df_base = pd.read_excel('IMDB Dataset 3.13.23.xlsx',sheet_name= 'all_im
```

In [5]:
```
df_base.drop(columns = ['director_id','number'],inplace = True)
```

Importing the dataset from Step 2

In [6]:
```
df_overview = pd.read_csv('movie_metadata_imdb_updated.csv')
```

In [8]:
```
df_overview.drop(columns = ['Unnamed: 0'],inplace = True)
```

In [9]:
```
df_overview.rename(columns = {'imdb_id': 'movie_id'},inplace = True)
```

Joining the two Datasets

In [12]:
```
new_df = df_base.merge(df_overview, on='movie_id', how='left')
```

In [14]:
```
df = new_df.dropna(subset=['overview'])
```

In [6]:
```
df[['movie_id','primaryTitle','genres','overview','keywords','cast']].h
```

Out[6]:

	movie_id	primaryTitle	genres	overview	keywords	cast
0	tt0111161	The Shawshank Redemption	Drama	Framed in the 1940s for the double murder of h...	[{'id': 378, 'name': 'prison'}, {'id': 417, 'n...	[{'cast_id': 3, 'character': 'Andy Dufresne', ...
1	tt0468569	The Dark Knight	Action,Crime,Drama	Batman raises the stakes in his war on crime. ...	[{'id': 849, 'name': 'dc comics'}, {'id': 853,...	[{'cast_id': 35, 'character': 'Bruce Wayne / B...
2	tt1375666	Inception	Action,Adventure,Sci-Fi	Cobb, a skilled thief who commits corporate es...	[{'id': 1014, 'name': 'loss of lover'}, {'id':...	[{'cast_id': 1, 'character': 'Dom Cobb', 'cred...
3	tt0137523	Fight Club	Drama	A ticking-time-bomb insomniac and a slippery s...	[{'id': 825, 'name': 'support group'}, {'id': ...	[{'cast_id': 4, 'character': 'The Narrator', '...
4	tt0109830	Forrest Gump	Drama,Romance	A man with a low IQ has accomplished great thi...	[{'id': 422, 'name': 'vietnam veteran'}, {'id'...	[{'cast_id': 7, 'character': 'Forrest Gump', '...
5	tt0110912	Pulp Fiction	Crime,Drama	A burger-loving hit man, his philosophical par...	[{'id': 396, 'name': 'transporter'}, {'id': 14...	[{'cast_id': 2, 'character': 'Vincent Vega', '...
6	tt0133093	The Matrix	Action,Sci-Fi	Set in the 22nd century, The Matrix tells the ...	[{'id': 83, 'name': 'saving the world'}, {'id'...	[{'cast_id': 34, 'character': 'Thomas "Neo" An...
7	tt0120737	The Lord of the Rings: The Fellowship of the Ring	Action,Adventure,Drama	Young hobbit Frodo Baggins, after inheriting a...	[{'id': 603, 'name': 'elves'}, {'id': 604, 'na...	[{'cast_id': 28, 'character': 'Frodo Baggins',...
8	tt0167260	The Lord of the Rings: The Return of the King	Action,Adventure,Drama	Aragorn is revealed as the heir to the ancient...	[{'id': 603, 'name': 'elves'}, {'id': 606, 'na...	[{'cast_id': 12, 'character': 'Frodo Baggins',...
9	tt0068646	The Godfather	Crime,Drama	Spanning the years 1945 to 1955, a chronicle o...	[{'id': 131, 'name': 'italy'}, {'id': 699, 'na...	[{'cast_id': 5, 'character': 'Don Vito Corleon...

Saving the results to a csv file

In [17]:
```
df.to_csv('final_dataset_40K_movies_4.17.23.csv')
```

Step 4: Data Cleaning : (~40K Movies)

In [1]:
```python
import pandas as pd
```

In [2]:
```python
df = pd.read_csv('final_dataset_40K_movies.csv')
```

In [3]:
```python
#Data Cleaning
df.drop(columns = ['Unnamed: 0'],inplace = True)
df.rename(columns = {'primaryTitle':'title'},inplace=True)
df['keywords'] = df['keywords'].fillna('[]')
df['original_language'] = df['original_language'].fillna('')
df['overview'] = df['overview'].fillna('')
df['director_name'] = df['director_name'].fillna('')
```

In [6]:
```python
df.head()
```

Out[6]:

	movie_id	title	originalTitle	isAdult	Year	runtimeMinutes	genres	averageR
0	tt0111161	The Shawshank Redemption	The Shawshank Redemption	0	1994	142	Drama	
1	tt0468569	The Dark Knight	The Dark Knight	0	2008	152	Action,Crime,Drama	
2	tt1375666	Inception	Inception	0	2010	148	Action,Adventure,Sci-Fi	
3	tt0137523	Fight Club	Fight Club	0	1999	139	Drama	
4	tt0109830	Forrest Gump	Forrest Gump	0	1994	142	Drama,Romance	

In [7]:
```python
import ast
df.keywords = df.keywords.apply(ast.literal_eval)
```

Defining a function to extract the top 3 cast and Keywords

In [8]:
```python
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
        if len(names) > 3:
            names = names[:3]
        return names

    #Return empty list in case of missing/malformed data
    return []
```

In [9]:
```python
df['keywords'] = df['keywords'].apply(get_list)
```

In [21]:
```python
df['keywords'] = df['keywords'].apply(lambda x: ','.join(map(str, x)))
```

In [7]:
```python
df.head()
```

Out[7]:

	movie_id	title	originalTitle	isAdult	Year	runtimeMinutes	genres	averageR
0	tt0111161	The Shawshank Redemption	The Shawshank Redemption	0	1994	142	Drama	
1	tt0468569	The Dark Knight	The Dark Knight	0	2008	152	Action,Crime,Drama	
2	tt1375666	Inception	Inception	0	2010	148	Action,Adventure,Sci-Fi	
3	tt0137523	Fight Club	Fight Club	0	1999	139	Drama	
4	tt0109830	Forrest Gump	Forrest Gump	0	1994	142	Drama,Romance	

In [2]:

```python
df = pd.read_csv('final_recommender_system_df_4.17.23.csv')
```

data cleaning

In [5]:

```python
# Function to convert all strings to lower case and strip names of spac
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 ''
```

In [6]:

```python
df['director_name'] = df['director_name'].apply(clean_data)
```

In [8]:

```python
df['cast'] = df['cast'].apply(clean_data)
```

In [10]:

```python
df['genres'] = df['genres'].apply(clean_data)
```

Creating a field for Movie Popularity

In [13]:

```python
# Calculate mean of vote average column
C = df['averageRating'].mean()
```

```
6.305530528619835
```

In [22]:

```python
# Calculate the minimum number of votes required to be in the chart, m
m = df['numVotes'].quantile(0.0)
```

```
6.0
```

In [23]:

```python
# Function that computes the weighted rating of each movie
def weighted_rating(x, m=m, C=C):
    v = x['numVotes']
    R = x['averageRating']
    # Calculation based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)
```

In [24]:

```python
# Define a new feature 'score' and calculate its value with `weighted_r
df['movie_popularity'] = df.apply(weighted_rating, axis=1)
```

In [26]:

```python
df = df[['title','movie_popularity','genres','keywords','cast','directo
```

In [27]:
```
df
```

Out[27]:

genres	keywords	
drama	prison,corruption,police brutality,prison cell...	timrobbins,morganfreeman,bobgunton,clancybr
action,crime,drama	dc comics,crime fighter,secret identity,scarec...	christianbale,michaelcaine,heathledger,aarc
action,adventure,sci-fi	loss of lover,dream,kidnapping,sleep,subconsci...	leonardodicaprio,josephgordon-levitt,ellenp
drama	support group,dual identity,nihilism,rage and ...	edwardnorton,bradpitt,meatloaf,jaredleto,h
drama,romance	vietnam veteran,hippie,mentally disabled,runni...	tomhanks,robinwright,garysinise,mykeltiwil
...	...	
iography,documentary	NaN	
documentary	demonstration,political activism,protest,wall ...	
documentary	NaN	
documentary	NaN	roberthenderson,krishaunb
iography,documentary	NaN	

In [28]:
```
df.to_csv('cleaned_df_4.17.23.csv', index=False)
```

Methodology

The proposed content-based movie recommender system utilizes a combination of feature extraction, vectorization, and cosine similarity to compare the similarity between movies and provide recommendations based on user preferences. Feature extraction is used to extract relevant features such as genre, director, and cast from the movie database. Vectorization is used to convert these features into a numerical format that can be used for similarity calculations. Cosine similarity is used to compare the similarity between movies based on their vectorized features.

TF-IDF is a technique used to evaluate the importance of words in a document based on their frequency in the document and the entire corpus. The technique calculates a score for each word in a document, with the score increasing proportionally to the frequency of the word in the document but decreasing based on the frequency of the word in the entire corpus. This helps to identify words that are unique to a document and, therefore, more informative in distinguishing it from other documents. TF-IDF is widely used in document classification, information retrieval, and search engine ranking.

Cosine similarity, is a measure of the similarity between two non-zero vectors of an inner product space. In text mining, the vectors represent the frequency of words in a document, and cosine similarity measures the angle between the two vectors. A value of 1 indicates that the two vectors are identical, while a value of 0 indicates that the two vectors are orthogonal and have no similarity. Cosine similarity is commonly used in text classification, clustering, and recommendation systems.

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

In [8]:
```python
import pandas as pd
#Import TfIdfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer
# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel
```

In [9]:
```python
df = pd.read_csv('cleaned_df_4.17.23.csv')
```

In [10]:
```python
df.fillna('', inplace=True)
```

In [11]:
```
1 df.head()
```

Out[11]:

	title	movie_popularity	genres	keywords	
0	The Shawshank Redemption	9.299993	drama	prison,corruption,police brutality,prison cell...	timrobbir
1	The Dark Knight	8.999994	action,crime,drama	dc comics,crime fighter,secret identity,scarec...	chris
2	Inception	8.799993	action,adventure,sci-fi	loss of lover,dream,kidnapping,sleep,subconsci...	leon
3	Fight Club	8.799993	drama	support group,dual identity,nihilism,rage and ...	edw
4	Forrest Gump	8.799992	drama,romance	vietnam veteran,hippie,mentally disabled,runni...	tom

Weights for Recommender System

- Genre - 30%
- Keywords - 25%
- Movie Popularity - 20%
- Movie Director - 15%
- Movie Cast - 10%

In [12]:
```
1 indices = pd.Series(df.index, index=df['title']).drop_duplicates()
```

In [13]:
```
1 new_df = df.copy()
```

In [14]:
```
1 new_df['weighted_movie_popularity']= new_df['movie_popularity'] * 2
```

Recommender System based on Genre

In [15]:

```python
#Define a TF-IDF Vectorizer Object. Remove all english stop words such
genre_tfidf = TfidfVectorizer(stop_words='english')

#Construct the required TF-IDF matrix by fitting and transforming the d
genre_tfidf_matrix = genre_tfidf.fit_transform(df['genres'])

#Output the shape of tfidf_matrix
genre_tfidf_matrix.shape
```

Out[15]: (39291, 29)

In [16]:

```python
#Array mapping from feature integer indices to feature name.
genre_tfidf.get_feature_names()[10:15]
```

Out[16]: ['fantasy', 'fi', 'film', 'history', 'horror']

In [17]:

```python
# Compute the cosine similarity matrix
genre_cosine_sim = linear_kernel(genre_tfidf_matrix, genre_tfidf_matrix
genre_cosine_sim.shape
```

Out[17]: (39291, 39291)

```python
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations_genre(title, genre_cosine_sim=genre_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
    genre_sim_scores = list(enumerate(genre_cosine_sim[idx]))

    new_df['genre_similarity_score'] = [i[1] for i in genre_sim_scores]
    new_df['weighted_genre_similarity_score']=new_df['genre_similarity_score'] * 30
```

```python
get_recommendations_genre('The Dark Knight')
```

```python
new_df
```

Recommender System Based on Keywords

In [18]:

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

#Construct the required TF-IDF matrix by fitting and transforming the d
keywords_tfidf_matrix = keywords_tfidf.fit_transform(df['keywords'])

#Output the shape of tfidf_matrix
keywords_tfidf_matrix.shape
```

Out[18]: (39291, 8719)

In [19]:

```
# Compute the cosine similarity matrix
keywords_cosine_sim = linear_kernel(keywords_tfidf_matrix, keywords_tfi
keywords_cosine_sim.shape
```

Out[19]: (39291, 39291)

```
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations_keywords(title, keywords_cosine_sim=keywords_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
    keywords_sim_scores = list(enumerate(keywords_cosine_sim[idx]))

    new_df['keywords_similarity_score'] = [i[1] for i in keywords_sim_scores]
new_df['weighted_keywords_similarity_score']=new_df['keywords_similarity_score'] * 25
```

```
get_recommendations_keywords('The Dark Knight')
```

```
new_df
```

Recommender System Based on Director

In [20]:

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

#Construct the required TF-IDF matrix by fitting and transforming the d
director_tfidf_matrix = director_tfidf.fit_transform(df['director_name'

#Output the shape of tfidf_matrix
director_tfidf_matrix.shape
```

Out[20]: (39291, 15367)

In [21]:

```
# Compute the cosine similarity matrix
director_cosine_sim = linear_kernel(director_tfidf_matrix, director_tfi
director_cosine_sim.shape
```

Out[21]: (39291, 39291)

```
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations_director(title, director_cosine_sim=director_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
    director_sim_scores = list(enumerate(director_cosine_sim[idx]))

    new_df['director_similarity_score'] = [i[1] for i in director_sim_scores]
new_df['weighted_director_similarity_score']=new_df['director_similarity_score'] * 15
```

```
get_recommendations_director('The Dark Knight')
```

```
new_df
```

Recommender System based on Cast

In [22]:

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

#Construct the required TF-IDF matrix by fitting and transforming the d
cast_tfidf_matrix = cast_tfidf.fit_transform(df['cast'])

#Output the shape of tfidf_matrix
cast_tfidf_matrix.shape
```

Out[22]: (39291, 66774)

In [23]:

```
# Compute the cosine similarity matrix
cast_cosine_sim = linear_kernel(cast_tfidf_matrix, cast_tfidf_matrix)
cast_cosine_sim.shape
```

Out[23]: (39291, 39291)

```
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations_cast(title, cast_cosine_sim=cast_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
    cast_sim_scores = list(enumerate(cast_cosine_sim[idx]))

```

```
    new_df['cast_similarity_score'] = [i[1] for i in cast_sim_scores]

new_df['weighted_cast_similarity_score']=new_df['cast_similarity_score
'] * 10
```

```
get_recommendations_cast('The Dark Knight')
```

```
new_df
```

Recommender System Function

In [24]:

```
# Function that takes in movie title as input and outputs most similar
def get_recommendations(title, genre_cosine_sim=genre_cosine_sim,keywor
    # Get the index of the movie that matches the title
    idx = indices[title]

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

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

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

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

    # Adding Similarity Scores to New DF
    new_df['genre_similarity_score'] = [i[1] for i in genre_sim_scores]
    new_df['weighted_genre_similarity_score']=new_df['genre_similarity_

    new_df['keywords_similarity_score'] = [i[1] for i in keywords_sim_s
    new_df['weighted_keywords_similarity_score']=new_df['keywords_simil

    new_df['director_similarity_score'] = [i[1] for i in director_sim_s
    new_df['weighted_director_similarity_score']=new_df['director_simil

    new_df['cast_similarity_score'] = [i[1] for i in cast_sim_scores]
    new_df['weighted_cast_similarity_score']=new_df['cast_similarity_sc

    # Computing Movie Similarity Score
    new_df['movie_similarity_score'] = new_df['weighted_movie_popularit

    #Sorting the Df based on Similarity
    result = new_df.sort_values(by=['movie_similarity_score'], ascendin

    #Returns top 5 similar movies
    return result[1:6]
```

Results

In [25]:

```
get_recommendations('Avengers: Age of Ultron')
```

Out[25]:

	title	movie_popularity	genres	keywords	
19	The Avengers	7.999992	action,adventure,sci-fi	new york,shield,marvel comic,superhero,based o...	robertdowneyjr.,chrisevar
108	Captain America: Civil War	7.799988	action,adventure,sci-fi	civil war,war,marvel comic,sequel,superhero	chrisevans,robertdowney
93	Iron Man 2	6.899995	action,adventure,sci-fi	malibu,marvel comic,superhero,based on comic,r...	robertdowneyjr.,gwynethpal
182	X-Men	7.399989	action,adventure,sci-fi	mutant,marvel comic,superhero,based on comic,s...	patrickstewart,hughjackma
84	Captain America: The Winter Soldier	7.699990	action,adventure,sci-fi	washington d.c.,future,shield,marvel comic,sup...	chrisevans,samuell.jacks

In [26]:
```
1 get_recommendations("Harry Potter and the Sorcerer's Stone")
```

Out[26]:

	title	movie_popularity	genres	keywords
175	Harry Potter and the Chamber of Secrets	7.499988	adventure,family,fantasy	flying car,witch,magic,cutting the cord,child ...
177	Harry Potter and the Prisoner of Azkaban	7.899984	adventure,family,fantasy	flying,traitor,magic,cutting the cord,child hero
203	Harry Potter and the Order of the Phoenix	7.499987	action,adventure,family	prophecy,witch,loss of lover,magic,cutting the...
1017	Percy Jackson & the Olympians: The Lightning T...	5.900013	adventure,family,fantasy	monster,greek mythology,god,poseidon ,lightni...
226	Harry Potter and the Deathly Hallows: Part 1	7.699984	adventure,family,fantasy	corruption,isolation,radio,magic,teleportation

In [27]:
```
get_recommendations("Skyfall")
```

Out[27]:

	title	movie_popularity	genres	keywords	
320	Spectre	6.799993	action,adventure,thriller	spy,based on novel,secret agent,sequel,mi6	danielcraig
306	Quantum of Solace	6.599996	action,adventure,thriller	killing,undercover,secret agent,british secret...	danielcraig,
1831	Diamonds Are Forever	6.599983	action,adventure,thriller	spy,fight,secret organization,satellite,secret...	seanconn
155	Casino Royale	7.999984	action,adventure,thriller	italy,poker,casino,terrorist,banker	danielcraig,e
32134	The Venetian Affair	5.410932	action,thriller	spy,secret agent	robertvau

Conclusion

In this study, we built a content-based movie recommender system that utilizes movie features such as genre, director, keywords, popularity and cast to provide personalized recommendations to users. The system uses feature extraction, vectorization, and cosine similarity to compare the similarity between movies and provide recommendations based on user preferences. We evaluated the proposed system using a dataset of 40K movies.

By using the following weights for the features we computed a similarity score which we used to recommend similar movies:

- Genre - 30%
- Keywords - 25%
- Movie Popularity - 20%
- Movie Director - 15%
- Movie Cast - 10%

The benefit of giving a majority of the weight (80% of total) to features such as genre, keywords, director and cast was that the system was effective in handling cold-start problems and can recommend niche movies that may not be popular.

As future work, we plan to further improve our system's accuracy and explore other advanced techniques, such as deep learning, to enhance its performance. Additionally, we aim to integrate a hybrid approach to this recommender system and get inputs from users, to make the system even more personalized. Overall, we believe that our proposed system can be a good recommender system not just for movies but also for other applications of recommendation systems.

References

1. Number of Movies on Streaming Services (https://www.businessinsider.com/major-streaming-services-compared-cost-number-of-movies-and-shows-2022-4#prime-video-has-the-most-movies-of-any-service-but-hbo-max-has-the-most-high-quality-movies-2)
2. IMDb Datasets (https://developer.imdb.com/non-commercial-datasets/)
3. Article on How to Build a Movie Recommendation System (https://towardsdatascience.com/how-to-build-a-movie-recommendation-system-67e321339109)
4. The second dataset for adding additional features (https://grouplens.org/datasets/movielens/latest/)
5. Kaggle - Movie Recommender System Project (https://www.kaggle.com/code/rounakbanik/movie-recommender-systems)

Apendix

1. Project Milestones

This project was part of CS 7920 - Analytics Project 2 during Spring 2023. As part of our progress we built a project plan with the expected deliverables for the project. The expected timeline for this project can be seen below:

Recommender System Project Expected Timeline

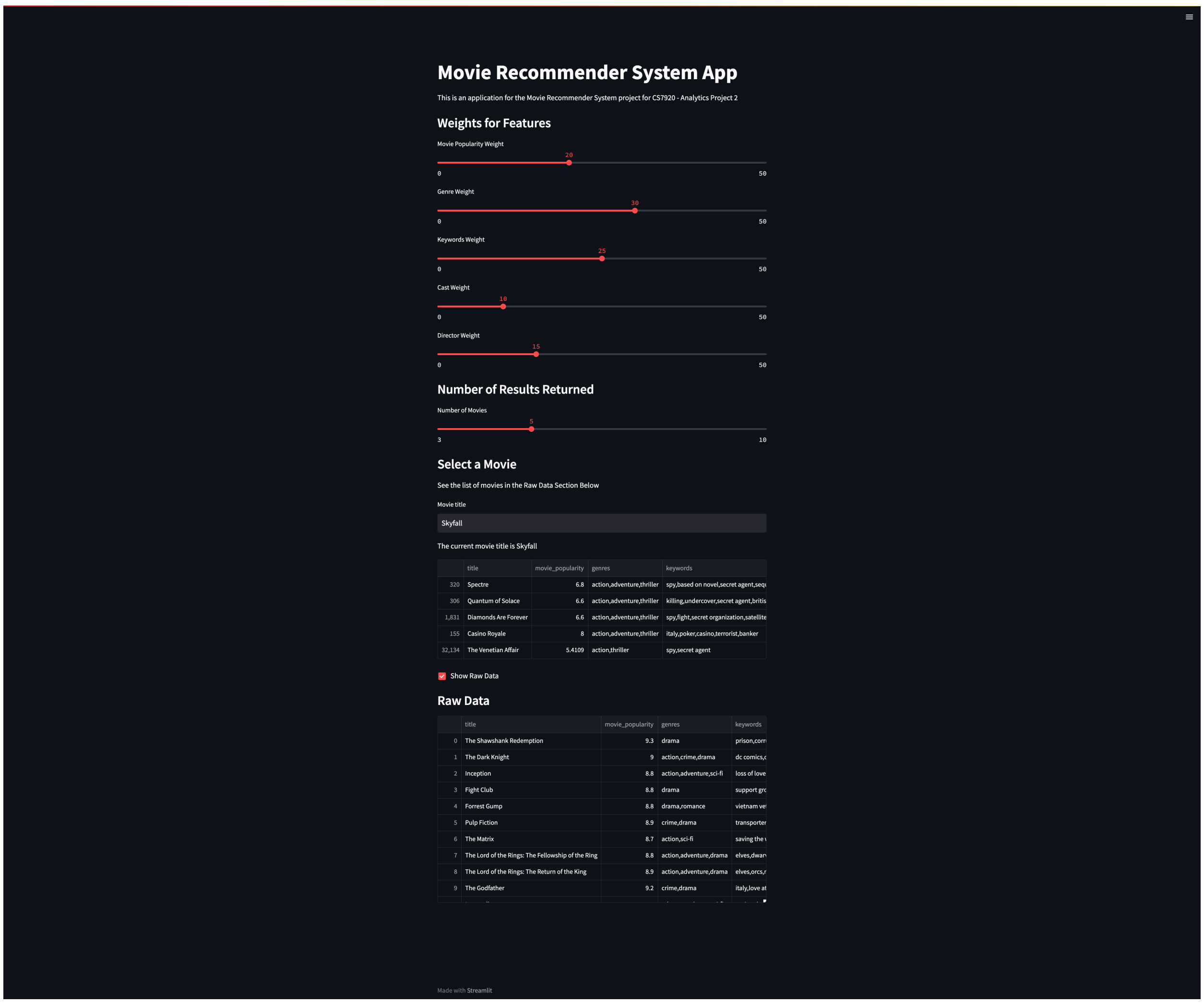
Date	Deliverables	Status
3.13.23	1. Expanded the dataset from top 250 movies to a 1000 movies	1. Completed
	2. Used the ChatGPT API to come up with Synthetic dataset for bank transactions	2. Completed
3.27.23	1. Expanding the dataset from 1000 movies to 2000 movies	1. The dataset consists of 1765 movies now
	2. Building V1 of the Recommender System Model	2. Built out a V1 of the Recommender System, still unsure of how it works
	3. Creating an estimated timeline of the project (this sheet)	3. This Sheet
4.3.23	1. Use a larger dataset for the model - around 40K movies (try to add storyline and keywords if possible).	1. Completed
	2. Improve the Recommender System model by using the other features in the dataset	2. Model only used Storyline. Need to add other features
	3. Adding measures for accuracy in the model	3. Added Similarity Score based on Cosine Similarity (Scores were very low - less than 10%)
4.10.23	Improve the Recommender System model by using the other features in the dataset	Built out 3 models for the Recommender System 1. Only based on Genre (cosine similarity very high - but not too useful) 2. Only based on Keywords (cosine similarity averaged 40% - plus results not too accurate) 3. Model based on Genre, Keywords, Storyline, Director and Language (cosine similarity less than 30% - not acccurate)
4.17.23	Add features of Movie Popularity (based on Average Rating and Number of Reviews) and Names of top 5 Cast Members for the movie	Completed
	Finalize the model of the Recommender System	Built out the final model by giving the following weights to features Genre - 30% Keywords - 25% Movie Popularity - 20% Movie Director - 15% Movie Cast - 10%
	Improving the performance of Recommender System	Movie Accuracy for top few movies around 80% - very accurate
4.24.23	Start working on an interface for picking the movie (maybe using Plotly Dash or Streamlit)	
	Start the Write Up for the Project	
5.1.23	Finalize the App for the Project	
	Final Application for Recommender System Due	
	Final Write Up for the Project due	

2. Recommender System App

We built a UI for our Recommender System. We used the Streamlit Library in Python to built a web application that lets the user picks a movie and the recommender system shows movies that are similar to the selected movie. The user also has the control to filter for the weights of the model as well as how many similar movies they want to see.

The code and a screenshot of the App can be seen below

Screenshot of the App



Code for the App

```
import streamlit as st
import pandas as pd
#Import TfIdfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer
# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel

```

```python
st.title("Movie Recommender System App")
st.markdown("This is an application for the Movie Recommender System project for CS7920 - Analytics Project 2")


st.subheader("Weights for Features")
# Weights
movie_popularity_weight = st.slider("Movie Popularity Weight", 0, 50,value=20)
genre_weight = st.slider("Genre Weight", 0, 50,value=30)
keywords_weight = st.slider("Keywords Weight", 0, 50,value=25)
cast_weight = st.slider("Cast Weight", 0, 50,value=10)
director_weight = st.slider("Director Weight", 0, 50,value=15)

st.subheader("Number of Results Returned")
num_results = st.slider("Number of Movies", 3, 10,value=5)

def load_data():
    data = pd.read_csv("/Users/asadimam270/Desktop/MS Data Science/Analytics Project 2/5.8.23 - Streamlit Application/cleaned_df_4.17.23.csv")
    data.fillna('', inplace=True)
    return data

data = load_data()
df = data.copy()
new_df = data.copy()

# Code for Recommender System
indices = pd.Series(data.index, index=data['title']).drop_duplicates()


##########################################
#Genre
##########################################

genre_tfidf = TfidfVectorizer(stop_words='english')
genre_tfidf_matrix = genre_tfidf.fit_transform(df['genres'])
genre_cosine_sim = linear_kernel(genre_tfidf_matrix, genre_tfidf_matrix)


##########################################
#Keywords
##########################################

keywords_tfidf = TfidfVectorizer(stop_words='english')
keywords_tfidf_matrix = keywords_tfidf.fit_transform(df['keywords'])
keywords_cosine_sim = linear_kernel(keywords_tfidf_matrix, keywords_tfidf_matrix)


##########################################
#Director
##########################################

```

```
director_tfidf = TfidfVectorizer(stop_words='english')
director_tfidf_matrix = director_tfidf.fit_transform(df['director_name'])
director_cosine_sim = linear_kernel(director_tfidf_matrix, director_tfidf_matrix)


########################################
#Cast
########################################

cast_tfidf = TfidfVectorizer(stop_words='english')
cast_tfidf_matrix = cast_tfidf.fit_transform(df['cast'])
cast_cosine_sim = linear_kernel(cast_tfidf_matrix, cast_tfidf_matrix)


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

def get_recommendations(title, genre_cosine_sim=genre_cosine_sim,keywords_cosine_sim=keywords_cosine_sim,director_cosine_sim=director_cosine_sim,cast_cosine_sim=cast_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
    genre_sim_scores = list(enumerate(genre_cosine_sim[idx]))

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

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

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

    # Adding Similarity Scores to New DF
    new_df['genre_similarity_score'] = [i[1] for i in genre_sim_scores]
    new_df['weighted_genre_similarity_score']=new_df['genre_similarity_score'] * genre_weight

    new_df['keywords_similarity_score'] = [i[1] for i in keywords_sim_scores]
    new_df['weighted_keywords_similarity_score']=new_df['keywords_similarity_score'] * keywords_weight

    new_df['director_similarity_score'] = [i[1] for i in director_sim_scores]
```

```
    new_df['weighted_director_similarity_score']=new_df['director_similarity_score'] * director_weight

    new_df['cast_similarity_score'] = [i[1] for i in cast_sim_scores]
    new_df['weighted_cast_similarity_score']=new_df['cast_similarity_score'] * cast_weight

    new_df['weighted_movie_popularity']= (new_df['movie_popularity']/10)* movie_popularity_weight

    # Computing Movie Similarity Score
    new_df['movie_similarity_score'] = new_df['weighted_movie_popularity'] + new_df['weighted_genre_similarity_score'] + new_df['weighted_keywords_similarity_score'] + new_df['weighted_director_similarity_score'] + new_df['weighted_cast_similarity_score']

    #Sorting the Df based on Similarity
    result = new_df.sort_values(by=['movie_similarity_score'], ascending=False)

    #Returns top 5 similar movies
    return result[1:num_results+1]




st.subheader("Select a Movie")
st.markdown("See the list of movies in the Raw Data Section Below")
title = st.text_input('Movie title', 'Skyfall')
st.write('The current movie title is', title)

st.write(get_recommendations(title))


if st.checkbox("Show Raw Data", False):
    st.subheader('Raw Data')
    st.write(data)
```