MOVIE RECOMMENDATION ENGINE

Project Report Submitted

To

Gujarat University

In partial fulfilment of the requirements for

the award to the Degree of

MASTER OF SCIENCE

(Artificial Intelligence & Machine Learning)

SEMESTER – III

GUIDED BY:

Mr. ANKAN MAJUMDAR

SUBMITTED BY:

VICKEY PATEL (30010)

PRIYANK SHIMPY (30004)

DEPARTMENT OF COMPUTER SCIENCE

GUJARAT UNIVERSITY, AHMEDABAD

YEAR: 2022-23

Department Of Computer Science

Gujarat University

Certificate

Roll No : 04 *Seat No : 30004*

This is to certify that Mr. /~~Ms~~. PRIYANK SHIMPY student of Third Semester of M.Sc (Artificial Intelligence and Machine Learning) – Defence Specific has duly completed his/her project titled ***MOVIE RECOMMENDATION ENGINE*** *for the semester ending in December 2022, towards partial fulfillment of degree of Master of Science (Artificial Intelligence and Machine Learning) – Defence Specific.*

Date of Submission *Internal Project Guide*

Course Coordinator

Head of Department

Department Of Computer Science

Gujarat University

Certificate

Roll No : 10   Seat No : 30010

This is to certify that Mr. /~~Ms~~. VICKEY SOMABHAI PATEL student of Third Semester of M.Sc (Artificial Intelligence and Machine Learning) has duly completed his/her project titled **MOVIE RECOMMENDATION ENGINE** for the semester ending in December 2022, towards partial fulfillment of degree of Master of Science (Artificial Intelligence and Machine Learning).

Date of Submission   Internal Project Guide

Course Coordinator

Head of Department

Acknowledgement

We would like to thank our guide **Mr. Ankan Majumdar** for their valuable time and provided support to our team. We owe a debt of gratitude to towards our guide for guiding us throughout the project and for her invaluable guidance at every stage of the preparation of this project. Without their constant guidance, support, help and encouragement the present study would not have been possible. We are grateful to our guide for their kindness, support, and feedback in various stages of this research work.

Abstract

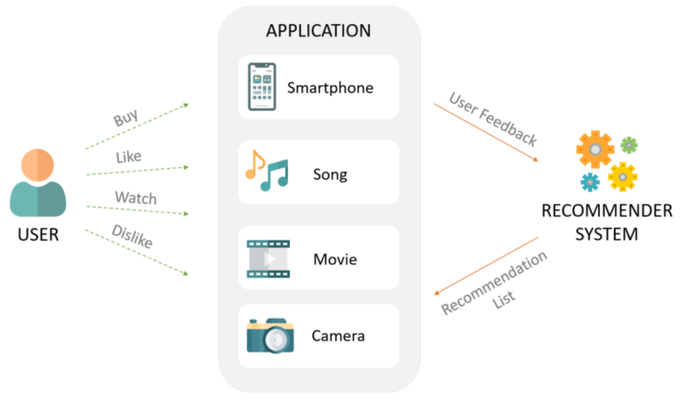
Now a day’s recommendation system has changed the style of searching the things of our interest. This is information filtering approach that is used to predict the preference of that user. This Report presents a movie recommendation system that is based on collaborative filtering , content-based methods and hybrid based filtering method. The system utilizes user ratings and content-based features such as genre, cast and crew, plot, and other related information to generate personalized recommendations for users. We used a hybrid approach to combine the collaborative filtering and content-based methods and applied them to a large movie dataset from TMDB. Our experimental results show that the hybrid approach outperforms the collaborative filtering approach, and the content-based approach, in terms of accuracy and coverage. Furthermore, the hybrid approach is able to capture user preferences in a more effective way. The user can then browse the recommendations easily and find a movie of their choice

Index

Figure Index

Introduction and Problem Statement

In today's world where internet has become an important part of human life, users often face the problem of too much choice. Right from looking for a motel to looking for good investment options, there is too much information available. To help the users cope with this information explosion, companies have deployed recommendation systems to guide their users. The research in the area of recommendation systems has been going on for several decades now, but the interest still remains high because of the abundance of practical applications and the problem rich domain the user may accept them according to their choice and may also provide, immediately or at a next stage, an implicit or explicit feedback. The actions of the users and their feedbacks can be stored in the recommender database and may be used for generating new recommendations in the next user-system interactions. High quality personalized recommendations add another dimension to user experience. The web personalized recommendation systems are recently applied to provide different types of customized information to their respective users.



These systems can be applied in various types of applications and are very common now day. We can classify the recommender systems in three broad categories:

- Collaborative filtering approach
- Content-based filtering approach
- Hybrid based filtering approach

Content-based filtering approach:

Content-based filtering is a type of recommendation system that uses the content of the item to recommend other items. It looks at the attributes of an item, such as genre, author, or actor, and uses them to recommend similar items. This approach is based on the assumption that if a user likes one item, they are likely to like other items with similar attributes. The content-based filtering approach is often used in music, movie, and book recommendation systems. It is also used in e-commerce websites to recommend similar products.

Collaborative filtering approach:

The collaborative filtering approach is a type of recommendation system that is used to make recommendations for an individual user based on the preferences of many other users. This approach uses the ratings and reviews of other users to make recommendations for a particular user. It works by creating a user-item matrix that stores the ratings of each user for each item. This matrix is then used to find similarities between users and items, in order to make recommendations. This approach is often used in online stores, social media sites, and other online platforms.

Hybrid based filtering approach:

The hybrid-based filtering approach combines the techniques of various types of filters (such as content-based, collaborative, and rule-based) to provide better filtering results. This type of filtering uses multiple techniques to analyze user input, such as analyzing content and context, user ratings, and user preferences. The hybrid-based filtering approach can also make use of machine learning algorithms to determine the best results for the user. This type of filtering is especially useful for personalizing content and providing better recommendations to users.

Problem Statement:

Design and implement a movie recommendation engine that can recommend movies to users based on their past movie preferences and ratings. The engine should be able to recommend movies to users based on their past movie preferences and ratings as well as any other relevant information about them (such as age, gender, location, etc.). The engine should be able to accurately recommend movies that the user will find interesting, based on their past movie ratings and preferences.

Tools, Technologies and Principles used

- **Tools:**

Machine Learning: Machine Learning algorithms such as collaborative filtering, content-based filtering, and hybrid approaches are used to build the movie recommendation system.

Natural Language Processing: NLP techniques, such as sentiment analysis and text classification, can be used to analyze user feedback and reviews to determine user preferences.

- **Technologies:**

Web Technologies: Python flask, HTML, CSS, JavaScript web technologies are used to build the user interface for the system

Hardware Requirements: A PC with Windows/Linux OS , Processor with 1.7-2.4gHz speed ,Minimum of 8gb RAM ,2gb Graphic card

Software Specification: VS-code , Anaconda distribution package (Jupyter notebook), Python libraries(Flask,gunicorn,Jinja2,MarkupSafe,Werkzeug,numpy,scipy,nltk,scikit-learn,pandas,beautifulsoup4,jsonschema.tmdbv3api.lxml,urllib3requests,pickleshare)

- **Principles:**

Personalization: The system should be able to personalize recommendations based on user preferences, interests, and history.

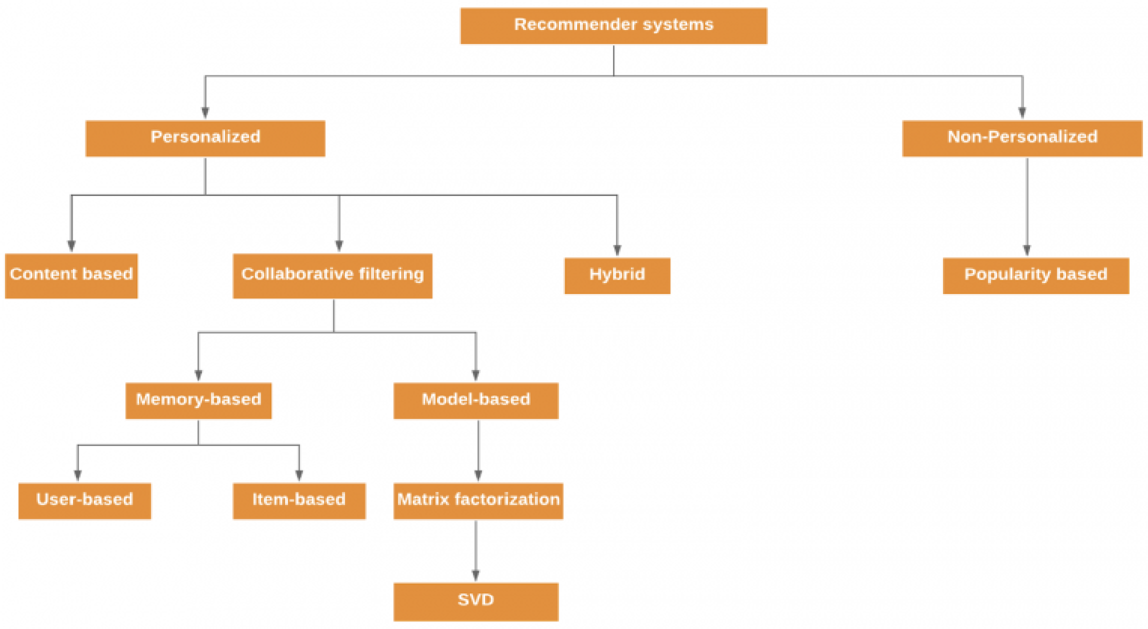
Relevance: The system should be able to recommend relevant movies to users based on their interests and preferences.

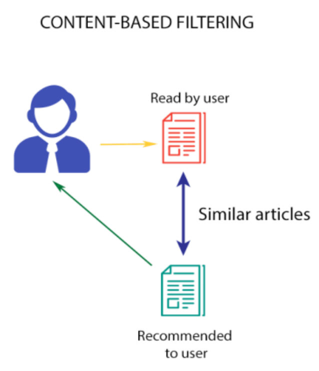
Diversity: The system should be able to recommend a diverse selection of movies to users to ensure a wide variety of choices.

Accuracy: The system should be able to provide accurate and reliable recommendations to users.

Methodology

The problem we have picked for the project is to design and train an efficient movie recommendation model which will recommend movies to users based on the User interaction matrix which would contain the user details with the movies the users liked and vice versa. Also, we will design another model which will recommend movies based on the context, title, genre, and such other attributes of the movies liked by the user and would recommend similar movies to the user. The proposed movie recommendation system gives finer similarity metrics and quality than the existing Movie recommendation system but the computation time which is taken by the proposed recommendation system is more than the existing recommendation system. This problem can be fixed by taking the clustered data points as an input dataset The proposed approach is for improving the scalability and quality of the movie recommendation system .We use a Hybrid approach , by unifying Content-Based Filtering and Collaborative Filtering, so that the approaches can be profited from each other. For computing similarity between the different movies in the given dataset efficiently and in least time and to reduce computation time of the movie recommender engine we used cosine similarity measure.





- **What is Content Based Filtering?**

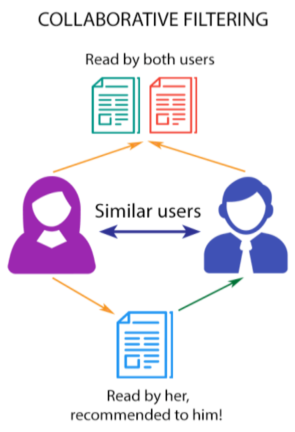
  Content-Based recommender system tries to guess the features or behaviour of a user given the item's features, he/she reacts positively to. The last two columns Action and Comedy Describe the Genres of the movies. Now, given these genres, we can know which users like which genre, as a result, we can obtain features corresponding to that particular user, depending on how he/she reacts to movies of that genre. Once, we know the likings of the user we can embed him/her in an embedding space using the feature vector generated and recommend him/her according to his/her choice.

Movies	User 1	User 2	User 3	User 4	Action	Comedy
Item 1	1		4	5	Yes	No
Item 2	5	4	1	2	No	Yes
Item 3	4	4		3	Yes	Yes
Item 4	2	2	4	4	No	Yes

During recommendation, the similarity metrics (We will talk about it in a bit) are calculated from the item's feature vectors and the user's preferred feature vectors from his/her previous records. Then, the top few are recommended.

- **What is Collaborative Filtering?**

  Collaborative does not need the features of the items to be given. Every user and item is described by a feature vector or embedding. It creates embedding for both users and items on its own. It embeds both users and items in the same embedding space. It considers other users' reactions while recommending a particular user. It notes which items a particular user likes and also the items that the users with behavior and likings like him/her likes, to recommend items to that user. It collects user feedbacks on different items and uses them for recommendations.



Type of collaborative Recommender System we are Using

we am using a Item-Item filtering type. Here, if user A likes an item x, then, the items y and z which are similar to x in property, then y and z are recommended to the user. As a statement, it can be said, “Because you liked this, you may also like those”.

Now, if one user A behaves like other users B, C, and D, then for a product x, A’s rating is given by:

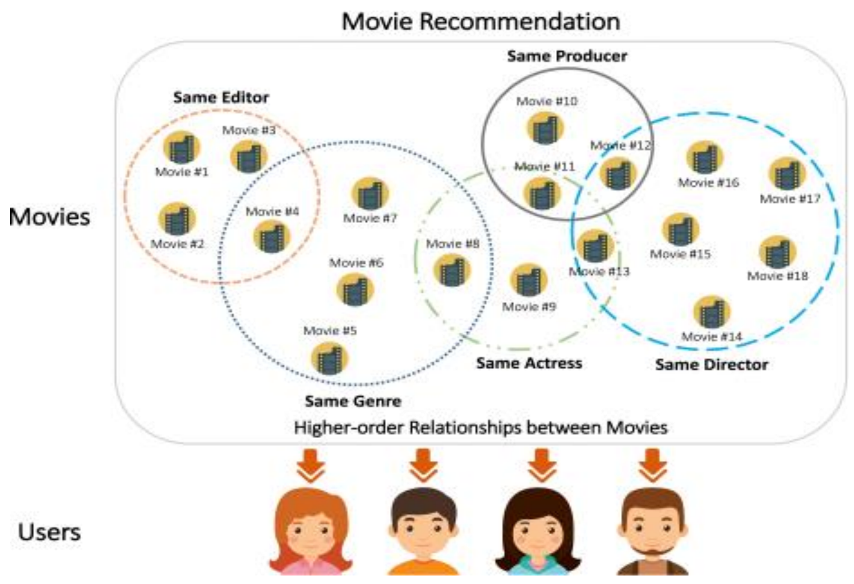
$$R_{xu}=(\Sigma_{i=0}{}^{n}R_i) / n$$

Where R is the rating user u gives to the product x, and it is the average of the ratings u gave to products like x. Here also, we take a weighted average.

$$R_{xu}=(\Sigma_{i=0}{}^{n}R_iW_i) / \Sigma_{i=0}{}^{n}W_i$$

- **User-based filtering:** User-based preferences are very common in the field of designing personalized systems. This approach is based on the user's likings. The process starts with users giving ratings to some movies. These ratings can be implicit or explicit. Often explicit ratings are hard to gather as not every user is much interested in providing feedbacks. In these scenarios, we gather implicit ratings based on their behaviour. For instance, if a user buys a product more than once, it indicates a positive preference. In context to movie systems, we can imply that if a user watches the entire movie, he/she has some likeability to it. Note that there are no clear rules in determining implicit ratings. Next, for each user, we first find some defined number of nearest neighbours. The assumption that if two users' ratings are highly correlated, then these two users must enjoy similar items and products is used to recommend items to users.

- **Item-based filtering:** Unlike the user-based filtering method, itembased focuses on the similarity between the item’s users like instead of the users themselves. The most similar items are computed ahead of time. Then for recommendation, the items that are most similar to the target item are recommended to the user.



Recommendation algorithms

	Advantages	**Disadvantages**
Content based	Recommendation result is intuitive and easy to interpret. No need for users access history data. No new item problem and no sparsity problem. Supported by the mature technology of classification learning.	Limited by the features extraction methods. New user problem. The training of classifier needs massive data. Poor scalability. Need more data.
Collaboration filtering	No need for professional knowledge. Performance improving as the increasing of the user number Automatic. Easy to find users new interesting point. Complex unstructured item can be processed. eg. Music, Video, etc.	Sparsity problem. Poor scalability. New user and new item problem. The recommendation quality limited by the history data set.
Hybrid Recommendation system	The recommendation accuracy is usually higher in hybrid systems. Make system robust, improve performance. Improve sparsity.	A little complex to apply on same dataset. Increase complexity, expensive in implementation.

DEEP LEARNING APPROACH FOR RECOMMENDATIONS

According to the study “Deep Neural Networks for YouTube Recommendations”, the YouTube recommendation system algorithm consists of two neural networks: one for candidate generation and one for ranking.

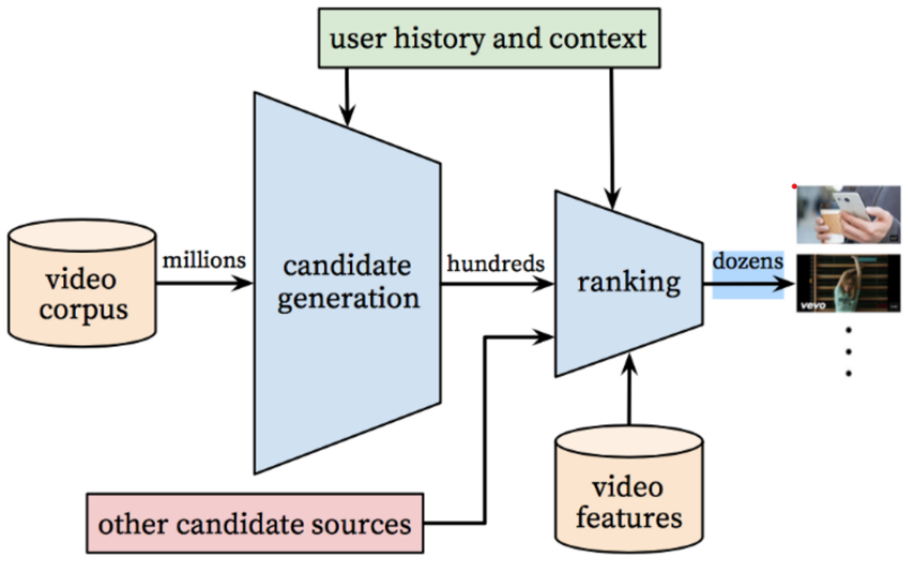
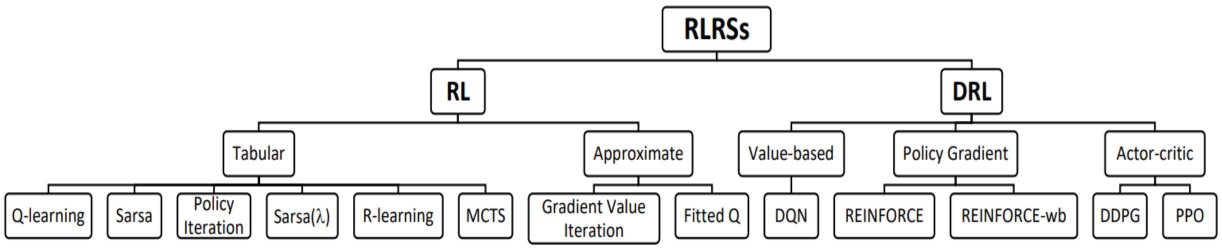


Figure-1

Taking events from a user’s history as input, the candidate generation network significantly decreases the amount of videos and makes a group of the most relevant ones from a large corpus. The generated candidates are the most relevant to the user, whose grades we are predicting. The goal of this network is only to provide a broad personalization via collaborative filtering.

REINFORCEMENT LEARNING APPROCH FOR RECOMMENDER SYSTEMS

The algorithms in a classified manner first we divide RLRSs into RL- and DRL-based methods. Then, we survey algorithms in each category with respect to the RLRS framework.



Its has Limitations

- **RL doesn't generalize well.** If new features or decisions are introduced, it often struggles to adapt.
- **RL doesn't scale well on combinatorial decisions spaces.** So, if we have lots of possible decisions, such as recommending lots of movies on Netflix's home screen, RL struggles to handle the volume of possible configurations.
- **RL doesn't handle low signal-to-noise ratio data.** RL is a very powerful model that can learn intricate rules and relationships from the data — if there are noisy features, RL will fit the noise.
- **RL doesn't handle long time-horizons.** Similar to the bullet above, if we're looking to optimize a long term decision, there's lots of opportunity to fit noise so RL can overfit if it's given a complex optimization task.

So, RL is really powerful in well-defined problem spaces, but how can we get it to generalize to more complex problems is a very difficult task.

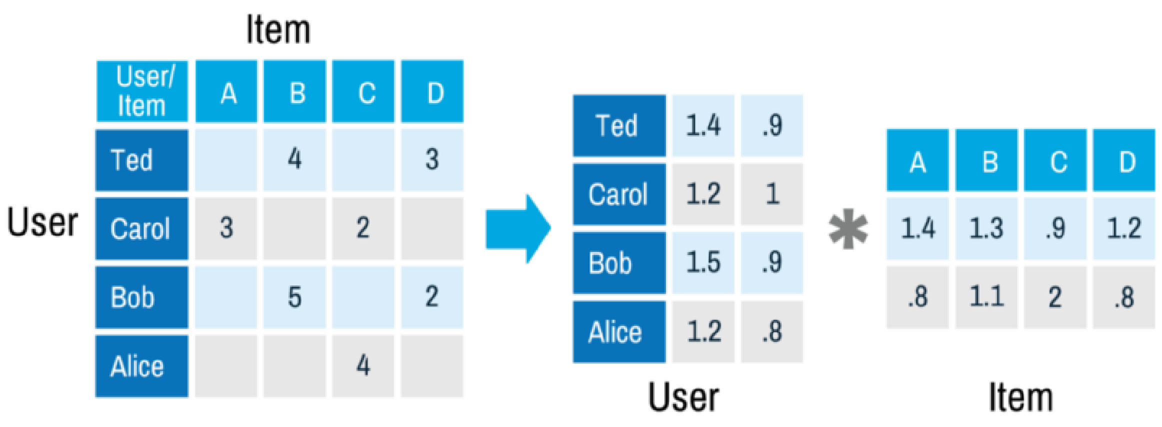
MATRIX DECOMPOSITION FOR RECOMMENDATIONS

In this approach uses matrix decompositions. It’s a very elegant recommendation algorithm because usually, when it comes to matrix decomposition, we don’t give much thought to what items are going to stay in the columns and rows of the resulting matrices. But using this recommender engine, we see clearly that **u** is a vector of interests of the i-thuser**,** and v is a vector of parameters for j-th film**.**

$$x_{ij} \approx \langle u_i, v_j \rangle$$

$$\sum_{i,j} \left(\langle u_i, v_j \rangle - x_{ij} \right)^2 \to min$$

So we can approximate **x** (grade from i-th user to j-th film) with dot product of **u** and **v**. We build these vectors by the known scores and use them to predict unknown grades. For example, after matrix decomposition we have vector (1.4; .9) for Ted and vector (1.4; .8) for film A, now we can restore the grade for film A−Ted just by calculating the dot product of (1.4; .9) and (1.4; .8). As a result, we get 2.68 grade**.**



User/ Item	A	B	C	D
Ted		4		3
Carol	3		2	
Bob		5		2
Alice			4	

Ted	1.4	.9
Carol	1.2	1
Bob	1.5	.9
Alice	1.2	.8

A	B	C	D
1.4	1.3	.9	1.2
.8	1.1	2	.8

Figure-2

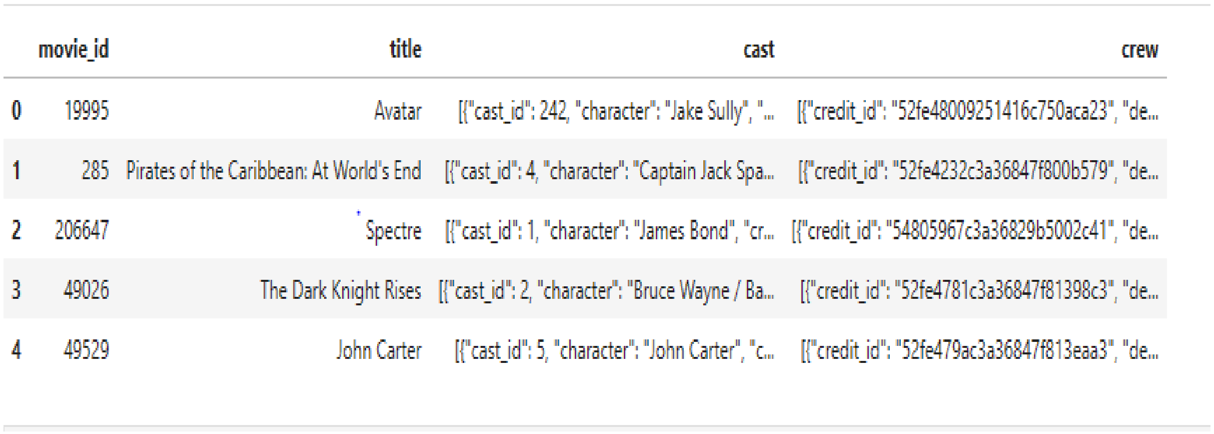
Dataset Usage

The TMDB dataset is a collection of data from the popular movie and television streaming service, The Movie Database (TMDB). The dataset contains information on over 5,000 movies and 5000 credits including title, cast, crew, budget, revenue, and release date. It also includes user ratings and information on the popularity of each movie. Additionally, the dataset contains metadata for each movie, such as the genre, language, and country of origin. The dataset is available for free download on the TMDB website, and can be used for a variety of research and analysis projects.

Dataset consist:

5000 Movies and 5000 credits are listed along with features include:

- Movie title
- Cast
- Crew
- Budget
- Genres
- Production companies
- Release date
- Runtime
- Revenue
- Tagline
- Overview
- Popularity
- Vote average - Vote count



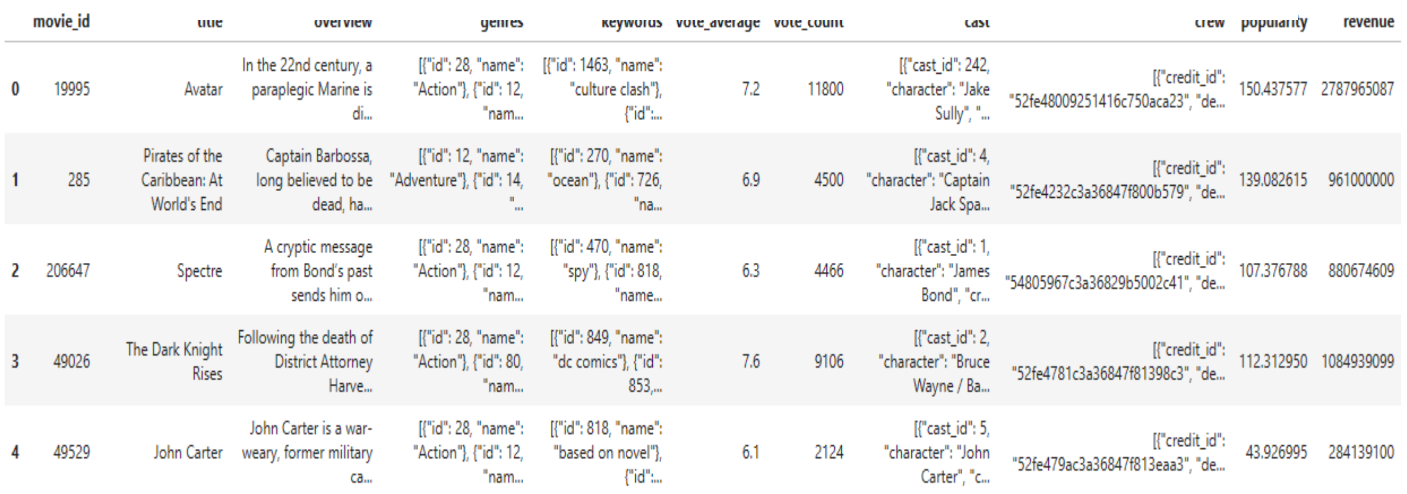
	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

Genres are listed along with their (Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western.) Certain columns, like 'cast' and 'genres', contain multiple values separated by pipe (|) characters.

This dataset download from this link:

https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata

- **Merging both csv of movie.csv and credit.csv:**



	movie_id	title	overview	genres	keywords	vote_average	vote_count	cast	crew	popularity	revenue
0	19995	Avatar	In the 22nd century, a paraplegic Marine is di...	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 1463, "name": "culture clash"}, {"id":...	7.2	11800	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...	150.437577	2787965087
1	285	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	6.9	4500	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...	139.082615	961000000
2	206647	Spectre	A cryptic message from Bond's past sends him o...	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 470, "name": "spy"}, {"id": 818, "name...	6.3	4466	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...	107.376788	880674609
3	49026	The Dark Knight Rises	Following the death of District Attorney Harve...	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	[{"id": 849, "name": "dc comics"}, {"id": 853,...	7.6	9106	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...	112.312950	1084939099
4	49529	John Carter	John Carter is a war-weary, former military ca...	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 818, "name": "based on novel"}, {"id":...	6.1	2124	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...	43.926995	284139100

Implementation Details

- **Load dataset and apply Pre-processing on data** : I have proposed a hybrid recommender which will be using the trained model from both the mentioned methods and then predict and derive the results for a specific movie and calculate the weighted average from the resultant matrices that we obtain and create a new hybrid matrix and extract the resultant predicted movies that we get from it.

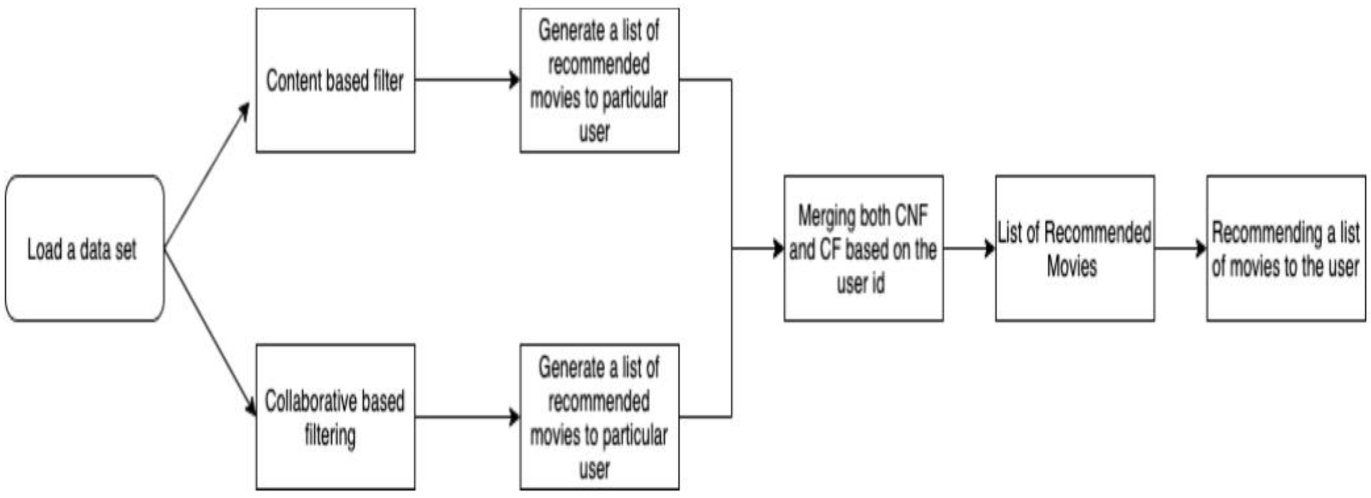
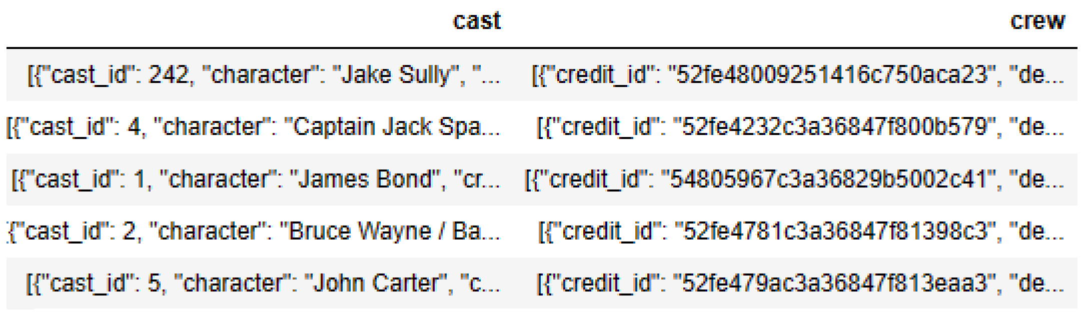

Load a data set
Content based filter
Collaborative based filtering
Generate a list of recommended movies to particular user
Generate a list of recommended movies to particular user
Merging both CNF and CF based on the user id
List of Recommended Movies
Recommending a list of movies to the user

Fig.3 Dataflow Diagram

Initially load the data sets that are required to build a model the data set that are required in this project are movies.csv, credit.csv, all the data sets are available in the Kaggle.com. Basically, two models are built in this project content based and collaborative filtering each produce a list of movies to a particular user by combining both based on the movieid a single final list of movies are recommended to the particular user.

Pre-processing: we have observed that our dataset features are in unstructured form

In our dataset our cast and crew data are in unstructured form.



cast	crew
[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

We use to convert well form in the data use AST and apply on the data

Abstract Syntax Tree (AST) is a data structure used to reason about the grammar of a programming language in the context of the instructions provided into source code

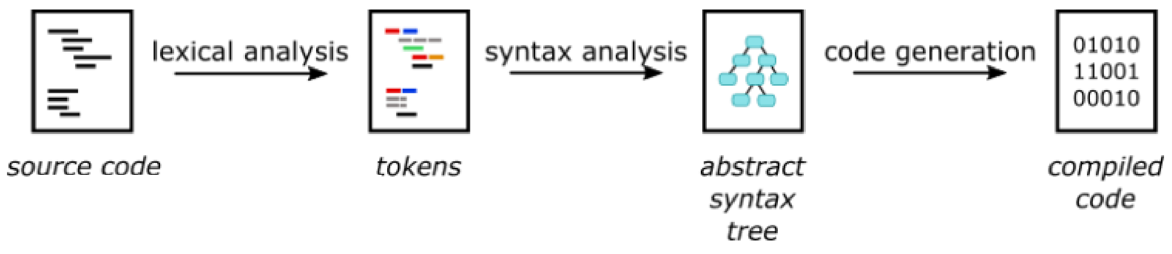
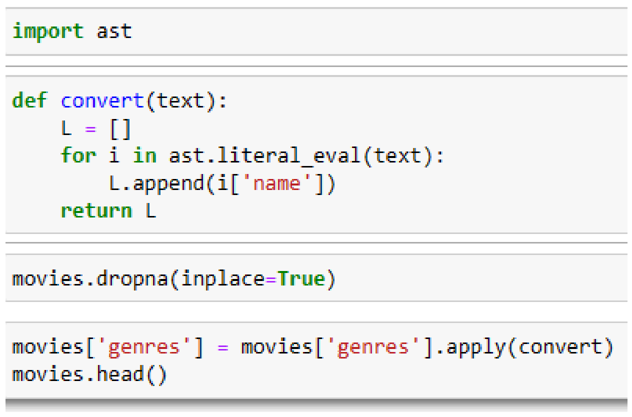


Figure 4 Abstract Syntax Tree

ast.literal_eval is a Python library function that can evaluate a string containing a Python expression. It is used to safely evaluate an expression node or a string containing a Python expression. It is used to evaluate strings containing Python expressions safely, avoiding the security issues of using eval().

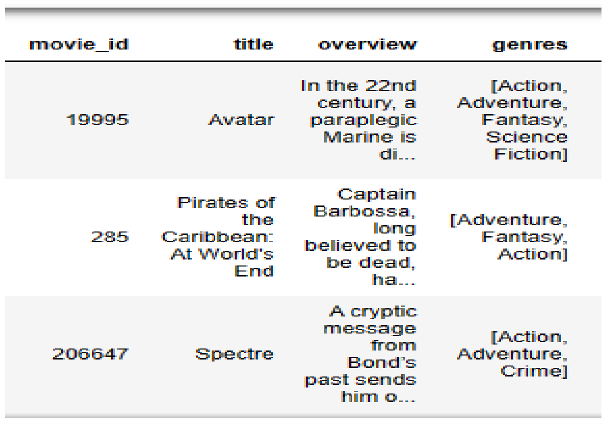


```
import ast
```

```
def convert(text):
    L = []
    for i in ast.literal_eval(text):
        L.append(i['name'])
    return L
```

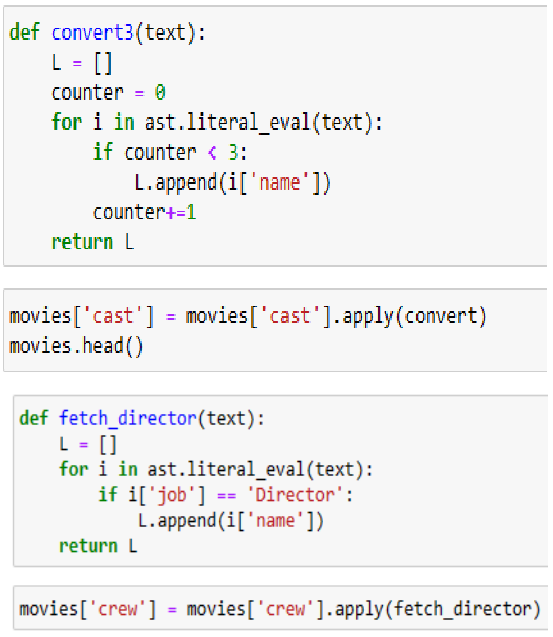
```
movies.dropna(inplace=True)
```

```
movies['genres'] = movies['genres'].apply(convert)
movies.head()
```



movie_id	title	overview	genres
19995	Avatar	In the 22nd century, a paraplegic Marine is di...	[Action, Adventure, Fantasy, Science Fiction]
285	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	[Adventure, Fantasy, Action]
206647	Spectre	A cryptic message from Bond's past sends him o...	[Action, Adventure, Crime]

Also pre-process on cast and crew

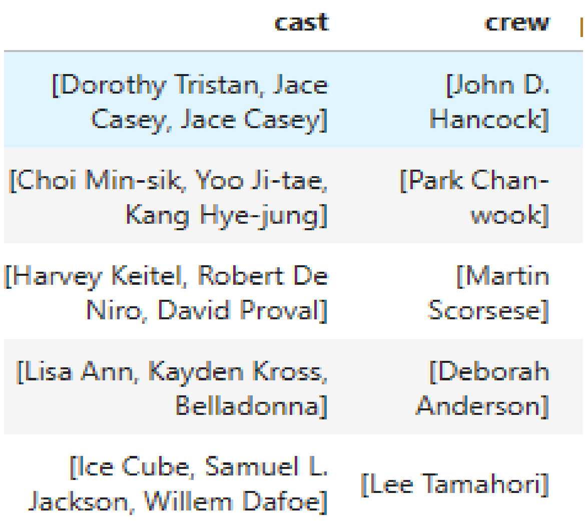


```
def convert3(text):
    L = []
    counter = 0
    for i in ast.literal_eval(text):
        if counter < 3:
            L.append(i['name'])
        counter+=1
    return L
```

```
movies['cast'] = movies['cast'].apply(convert)
movies.head()
```

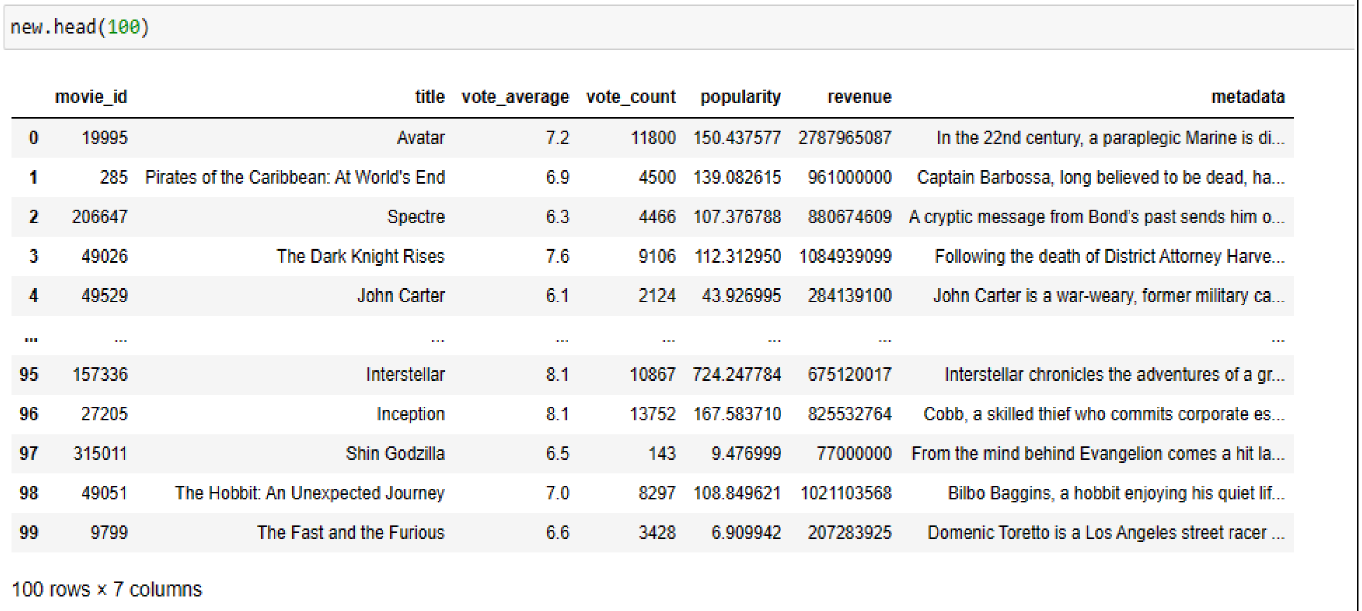
```
def fetch_director(text):
    L = []
    for i in ast.literal_eval(text):
        if i['job'] == 'Director':
            L.append(i['name'])
    return L
```

```
movies['crew'] = movies['crew'].apply(fetch_director)
```



cast	crew
[Dorothy Tristan, Jace Casey, Jace Casey]	[John D. Hancock]
[Choi Min-sik, Yoo Ji-tae, Kang Hye-jung]	[Park Chan-wook]
[Harvey Keitel, Robert De Niro, David Proval]	[Martin Scorsese]
[Lisa Ann, Kayden Kross, Belladonna]	[Deborah Anderson]
[Ice Cube, Samuel L. Jackson, Willem Dafoe]	[Lee Tamahori]

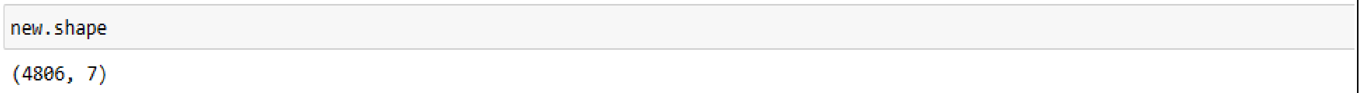
Final dataset



```
new.head(100)
```

	movie_id	title	vote_average	vote_count	popularity	revenue	metadata
0	19995	Avatar	7.2	11800	150.437577	2787965087	In the 22nd century, a paraplegic Marine is di...
1	285	Pirates of the Caribbean: At World's End	6.9	4500	139.082615	961000000	Captain Barbossa, long believed to be dead, ha...
2	206647	Spectre	6.3	4466	107.376788	880674609	A cryptic message from Bond's past sends him o...
3	49026	The Dark Knight Rises	7.6	9106	112.312950	1084939099	Following the death of District Attorney Harve...
4	49529	John Carter	6.1	2124	43.926995	284139100	John Carter is a war-weary, former military ca...
...	...	...	...	...	...	...	...
95	157336	Interstellar	8.1	10867	724.247784	675120017	Interstellar chronicles the adventures of a gr...
96	27205	Inception	8.1	13752	167.583710	825532764	Cobb, a skilled thief who commits corporate es...
97	315011	Shin Godzilla	6.5	143	9.476999	77000000	From the mind behind Evangelion comes a hit la...
98	49051	The Hobbit: An Unexpected Journey	7.0	8297	108.849621	1021103568	Bilbo Baggins, a hobbit enjoying his quiet lif...
99	9799	The Fast and the Furious	6.6	3428	6.909942	207283925	Domenic Toretto is a Los Angeles street racer ...

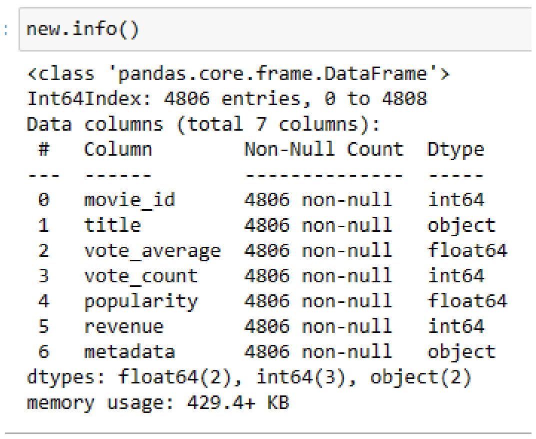
100 rows × 7 columns



```
new.shape
```

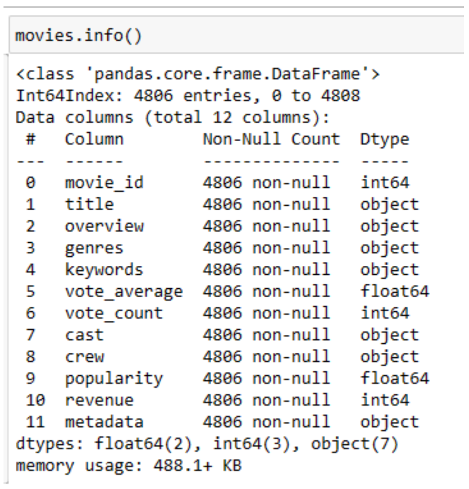
```
(4806, 7)
```

Exploratory Data Analysis(EDA)



```
new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4806 entries, 0 to 4808
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   movie_id      4806 non-null   int64  
 1   title         4806 non-null   object 
 2   vote_average  4806 non-null   float64
 3   vote_count    4806 non-null   int64  
 4   popularity    4806 non-null   float64
 5   revenue       4806 non-null   int64  
 6   metadata      4806 non-null   object 
dtypes: float64(2), int64(3), object(2)
memory usage: 429.4+ KB
```



```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4806 entries, 0 to 4808
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   movie_id      4806 non-null   int64  
 1   title         4806 non-null   object 
 2   overview      4806 non-null   object 
 3   genres        4806 non-null   object 
 4   keywords      4806 non-null   object 
 5   vote_average  4806 non-null   float64
 6   vote_count    4806 non-null   int64  
 7   cast          4806 non-null   object 
 8   crew          4806 non-null   object 
 9   popularity    4806 non-null   float64
 10  revenue       4806 non-null   int64  
 11  metadata      4806 non-null   object 
dtypes: float64(2), int64(3), object(7)
memory usage: 488.1+ KB
```

```
import matplotlib.pyplot as plt
import seaborn as sns
print(new['vote_average'].describe())
plt.figure(figsize=(9, 8))
sns.distplot(new['vote_average'],color='g', bins=100, hist_kws={'alpha': 0.4});
```

```
count    4806.000000
mean        6.093258
std         1.190846
min         0.000000
25%         5.600000
50%         6.200000
75%         6.800000
max        10.000000
Name: vote_average, dtype: float64
```

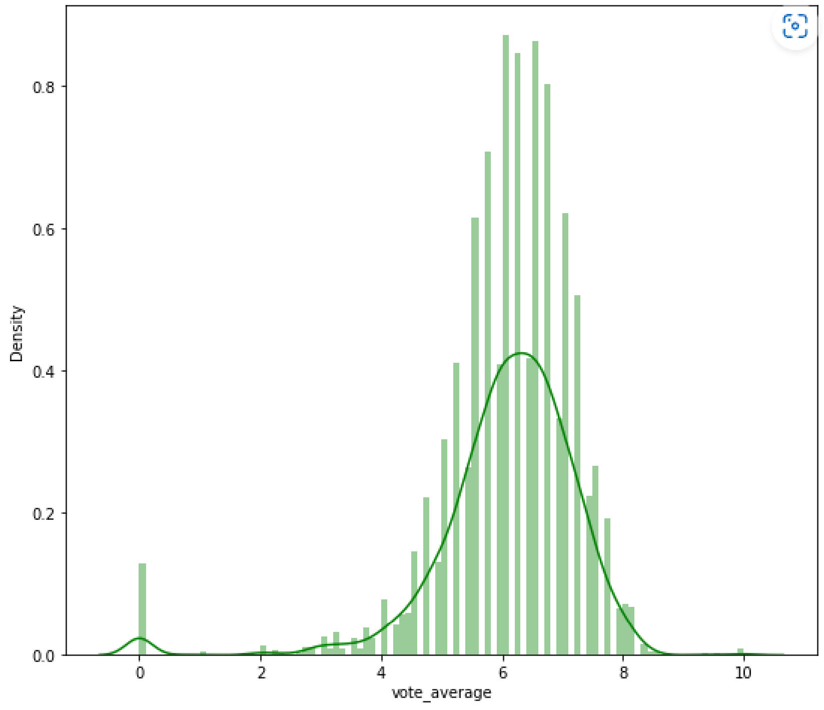


Figure-5. vote_avarage

```
movies_num = movies.select_dtypes(include = ['float64', 'int64'])
movies_num.head()
```

	movie_id	vote_average	vote_count	popularity	revenue
0	19995	7.2	11800	150.437577	2787965087
1	285	6.9	4500	139.082615	961000000
2	206647	6.3	4466	107.376788	880674609
3	49026	7.6	9106	112.312950	1084939099
4	49529	6.1	2124	43.926995	284139100

```
movies_num.hist(figsize=(10, 10), bins=50, xlabelsize=8, ylabelsize=8);
```

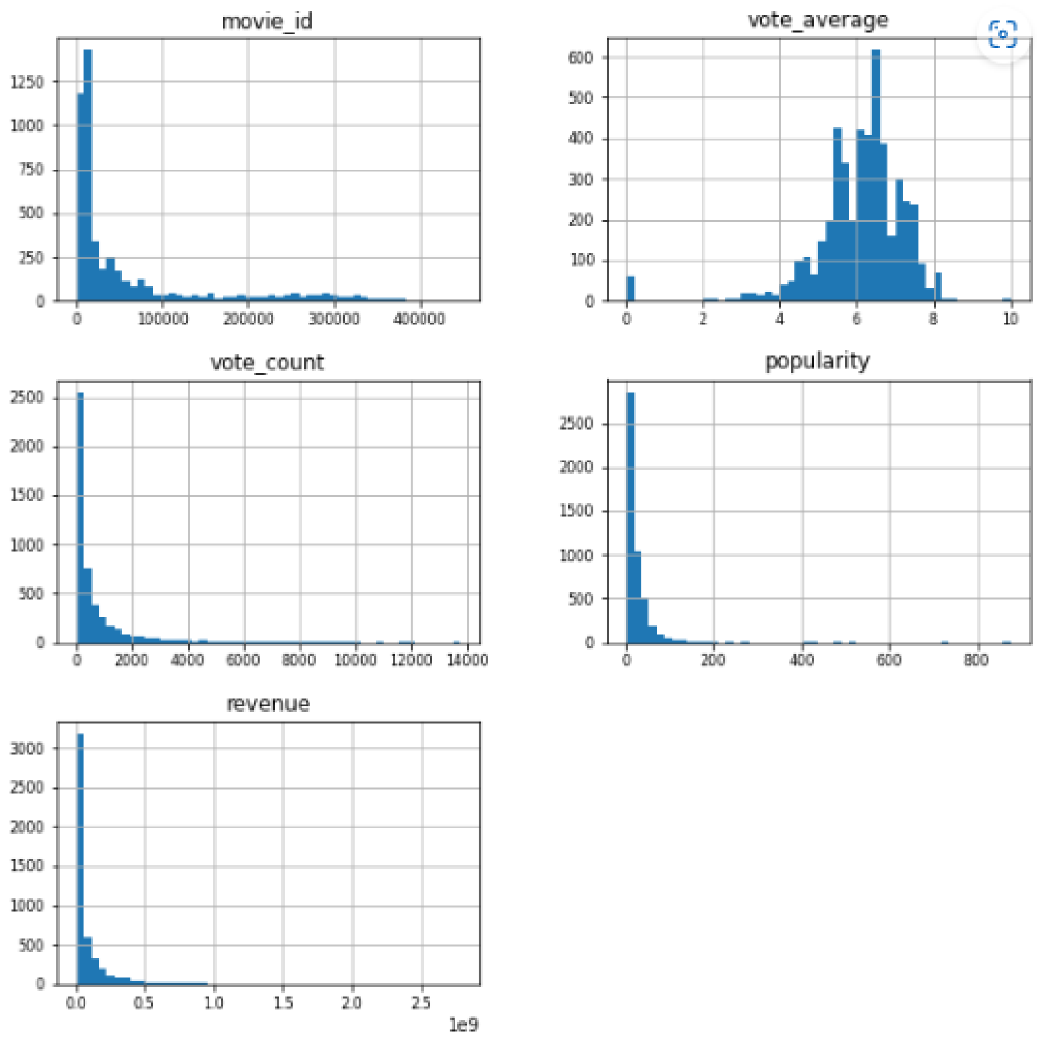
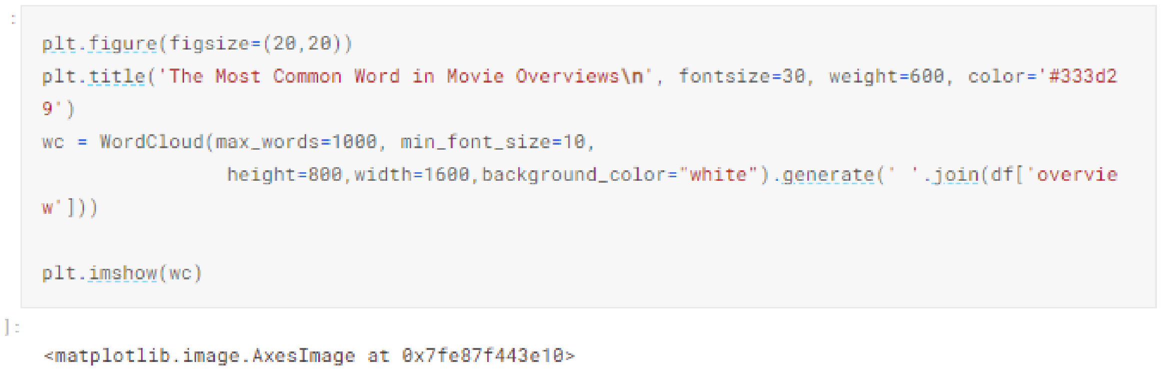


Fig6. Analysis of movie_id , vote_avarage, vote_count ,popularity revenue histogram

The most Common Word in Movie Overview:


```
plt.figure(figsize=(20,20))
plt.title('The Most Common Word in Movie Overviews\n', fontsize=30, weight=600, color='#333d2
9')
wc = WordCloud(max_words=1000, min_font_size=10,
                height=800,width=1600,background_color="white").generate(' '.join(df['overvie
w']))

plt.imshow(wc)
```

```
<matplotlib.image.AxesImage at 0x7fe87f443e10>
```

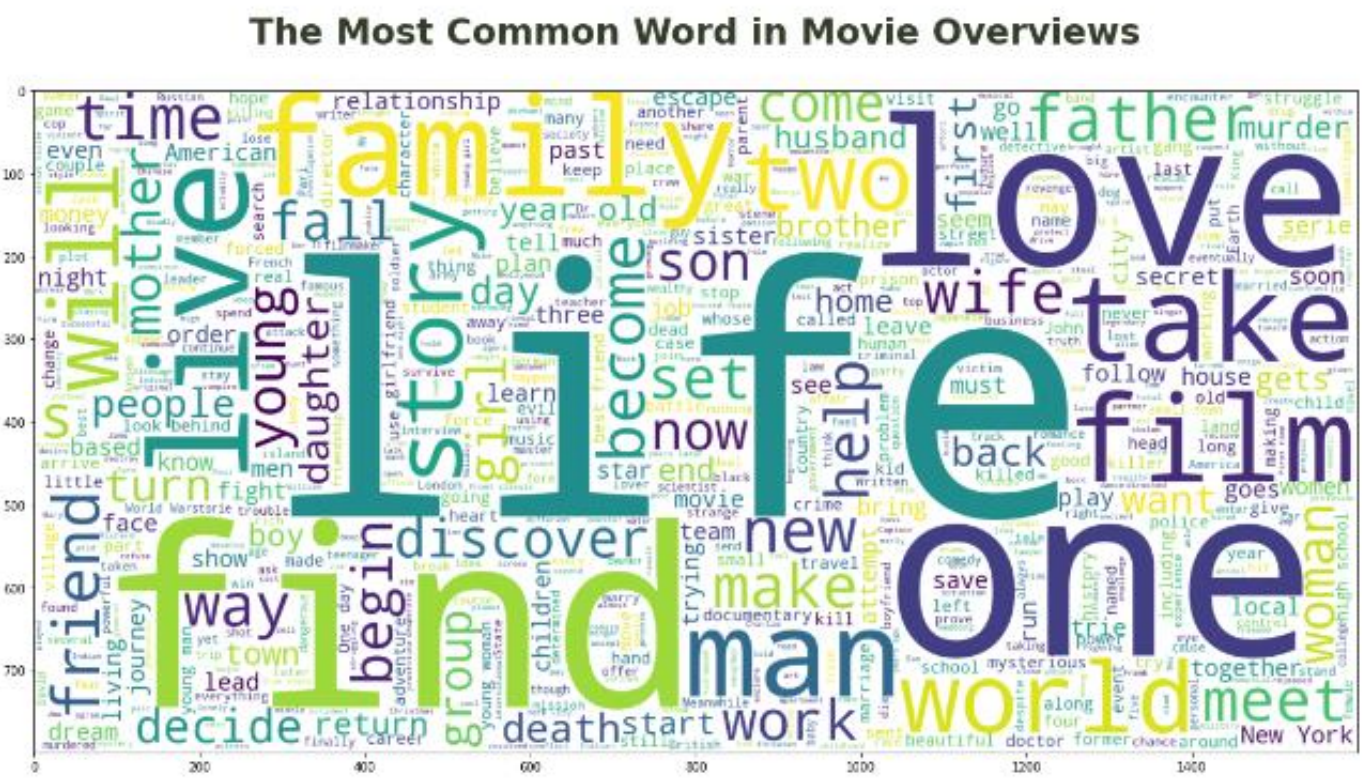
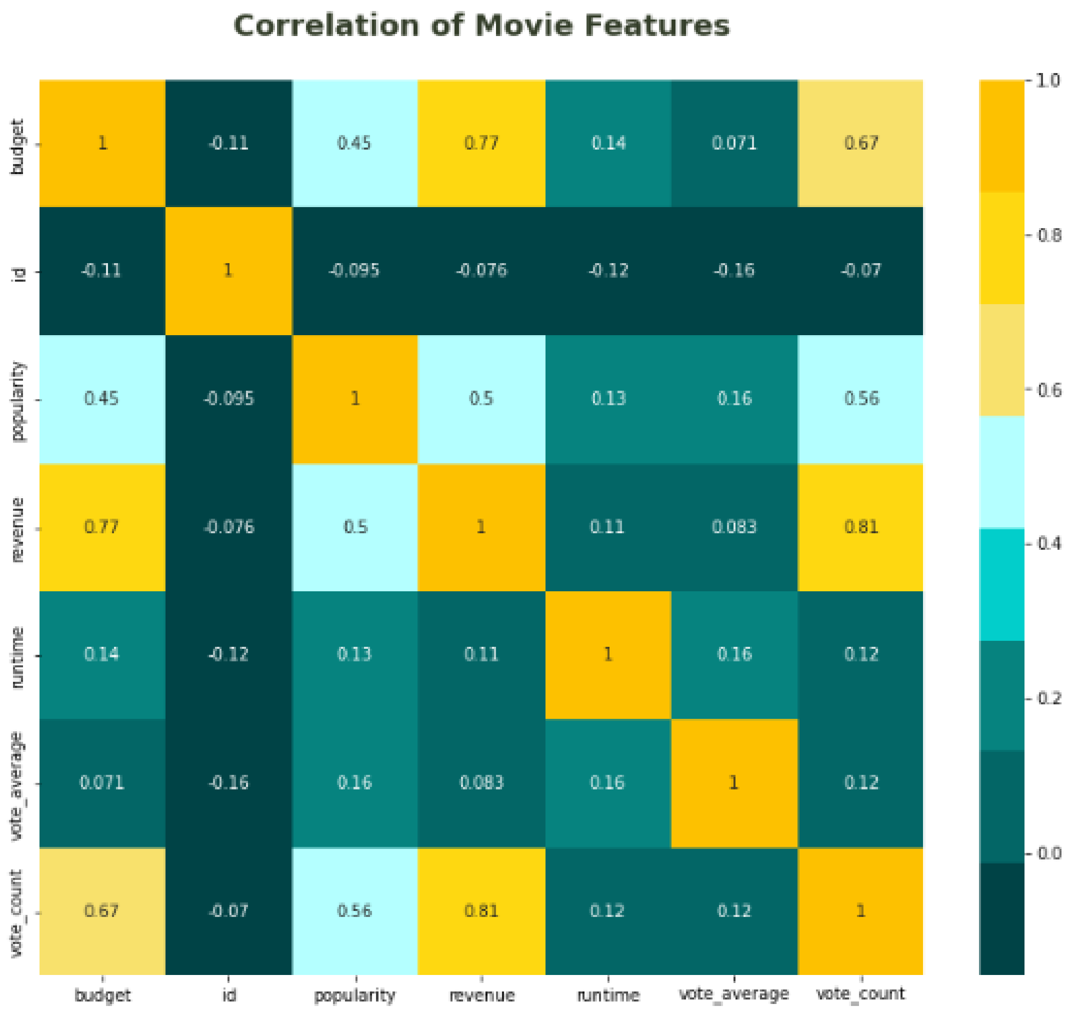


Fig7. Word MAP The most Common Word in Movie Overview:

Correlated value with average

```
plt.figure(figsize=(12,10))
plt.title('Correlation of Movie Features\n', fontsize=18, weight=600, color='#333d29')
sns.heatmap(df.corr(), annot=True, cmap=['#004346', '#036666', '#06837f', '#02cecb', '#b4fff
f', '#f8e16c', '#fed811', '#fdc100'])
```

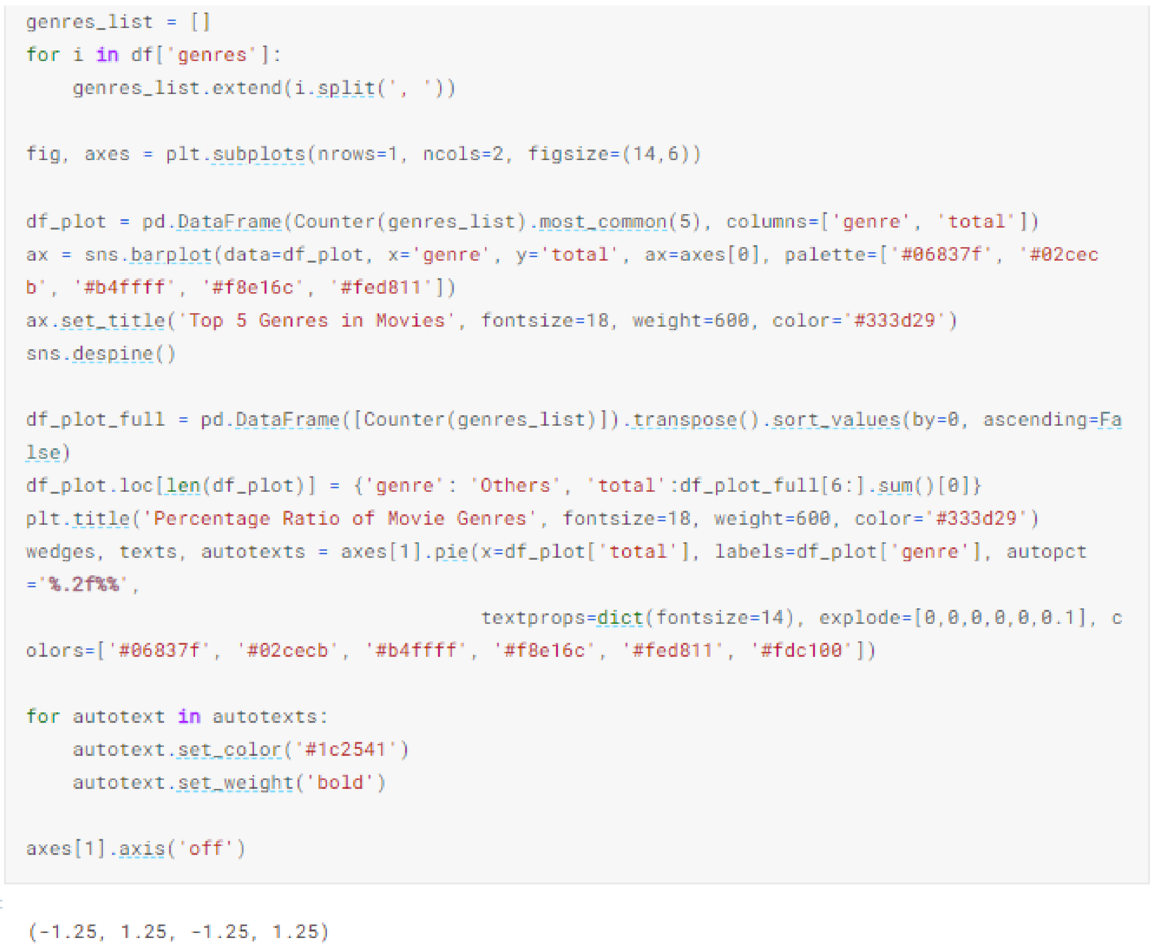
```
<AxesSubplot:title={'center':'Correlation of Movie Features\n'}>
```



	budget	id	popularity	revenue	runtime	vote_average	vote_count
budget	1	-0.11	0.45	0.77	0.14	0.071	0.67
id	-0.11	1	-0.095	-0.076	-0.12	-0.16	-0.07
popularity	0.45	-0.095	1	0.5	0.13	0.16	0.56
revenue	0.77	-0.076	0.5	1	0.11	0.083	0.81
runtime	0.14	-0.12	0.13	0.11	1	0.16	0.12
vote_average	0.071	-0.16	0.16	0.083	0.16	1	0.12
vote_count	0.67	-0.07	0.56	0.81	0.12	0.12	1

Fig7. Heatmap of correlated vote average.

Drama is the most dominant genre with over 5000 movies



```
genres_list = []
for i in df['genres']:
    genres_list.extend(i.split(', '))

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14,6))

df_plot = pd.DataFrame(Counter(genres_list).most_common(5), columns=['genre', 'total'])
ax = sns.barplot(data=df_plot, x='genre', y='total', ax=axes[0], palette=['#06837f', '#02cec
b', '#b4ffff', '#f8e16c', '#fed811'])
ax.set_title('Top 5 Genres in Movies', fontsize=18, weight=600, color='#333d29')
sns.despine()

df_plot_full = pd.DataFrame([Counter(genres_list)]).transpose().sort_values(by=0, ascending=Fa
lse)
df_plot.loc[len(df_plot)] = {'genre': 'Others', 'total':df_plot_full[6:].sum()[0]}
plt.title('Percentage Ratio of Movie Genres', fontsize=18, weight=600, color='#333d29')
wedges, texts, autotexts = axes[1].pie(x=df_plot['total'], labels=df_plot['genre'], autopct
='%.2f%%',
                                        textprops=dict(fontsize=14), explode=[0,0,0,0,0,0.1], c
olors=['#06837f', '#02cecb', '#b4ffff', '#f8e16c', '#fed811', '#fdc100'])

for autotext in autotexts:
    autotext.set_color('#1c2541')
    autotext.set_weight('bold')

axes[1].axis('off')
```

```
(-1.25, 1.25, -1.25, 1.25)
```

Out of 5 top genres, there are still many genres in the dataset. They hold 38.67% of total genres in the movies

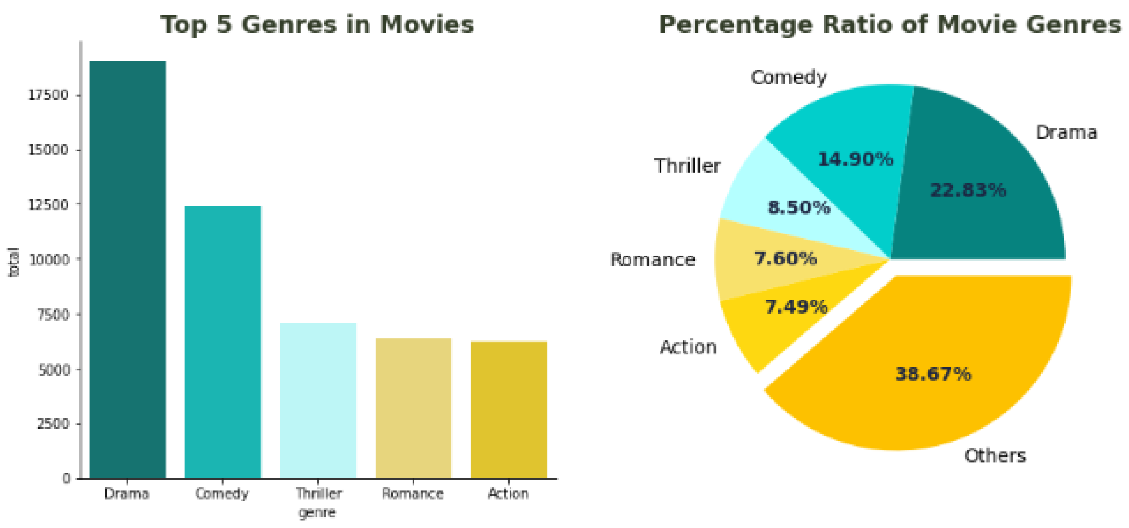


Fig8. Top 5 Genres in movie/ percentage ration of movie generes

Taking top 10 genres with highest counts

```
Counts = movies["genres"].value_counts() .head(10) #Taking top 10 genres with hi
Genres = Counts.index
Genres
```

```
Index([                    ['Drama'],                        ['Comedy'],
                 ['Drama', 'Romance'],             ['Comedy', 'Romance'],
                  ['Comedy', 'Drama'], ['Comedy', 'Drama', 'Romance'],
               ['Horror', 'Thriller'],                  ['Documentary'],
                           ['Horror'],             ['Drama', 'Thriller']],
      dtype='object')
```

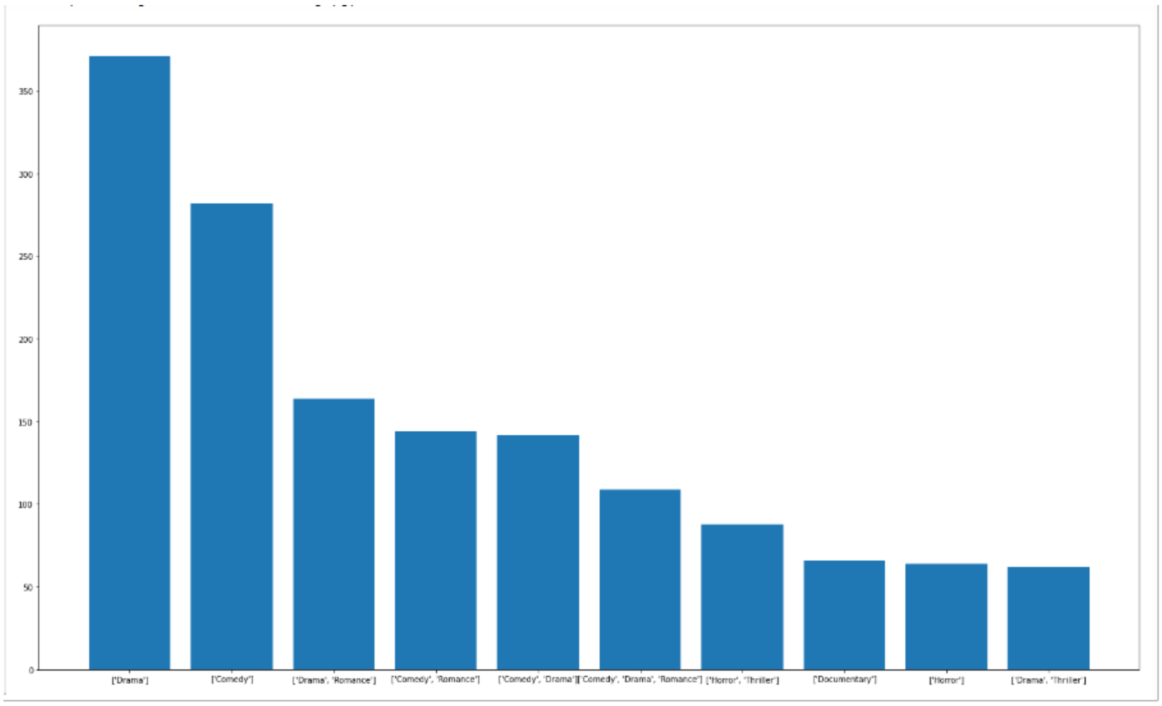


Figure 9. Taking top 10 genres with highest counts

Text Vectorization: Term Frequency — Inverse Document Frequency (TFIDF)

Term frequency — Inverse document frequency (TFIDF) is based on the Bag of Words (BoW) model, which contains insights about the less relevant and more relevant words in a document. The importance of a word in the text is of great significance in information retrieval.

Example — If you search something on the search engine, with the help of TFIDF values, search engines can give us the most relevant documents related to our search.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words="english")
tfidf_matrix = tfidf.fit_transform(new['metadata'])
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), index=new.index)
print(tfidf_df.shape)
```

```
(4806, 35547)
```

```
tfidf_df.loc[0]
```

```
0        0.0
1        0.0
2        0.0
3        0.0
4        0.0
        ...
35542    0.0
35543    0.0
35544    0.0
35545    0.0
35546    0.0
Name: 0, Length: 35547, dtype: float64
```

See for what dimension range we get atleast 80% of the variance of features

```
from sklearn.decomposition import TruncatedSVD
import matplotlib.pyplot as plt
svd = TruncatedSVD(n_components=2500)
latent_matrix = svd.fit_transform(tfidf_df)
```

```
#See for what dimension range we get atleast 80% of the variance of features
## for now it is at alomost 0.40 because of time and process related issues
explained  = svd.explained_variance_ratio_.cumsum()
plt.plot(explained,'.-',ms=10,color='red')
plt.xlabel('Singular value components', fontsize=12)
plt.ylabel('Cumulative percent of varience', fontsize=12)
plt.show()
```

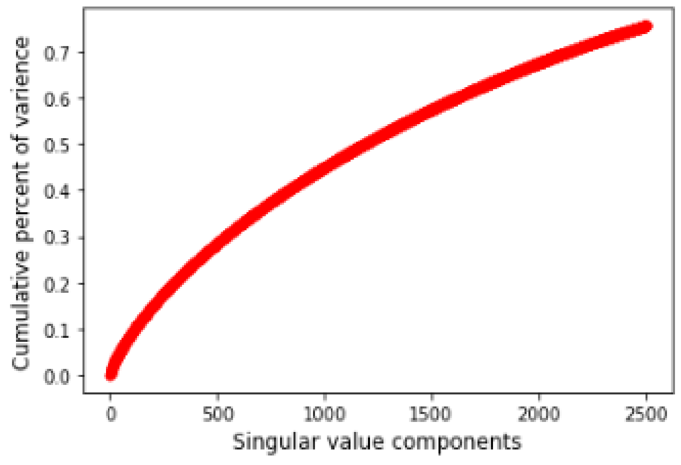
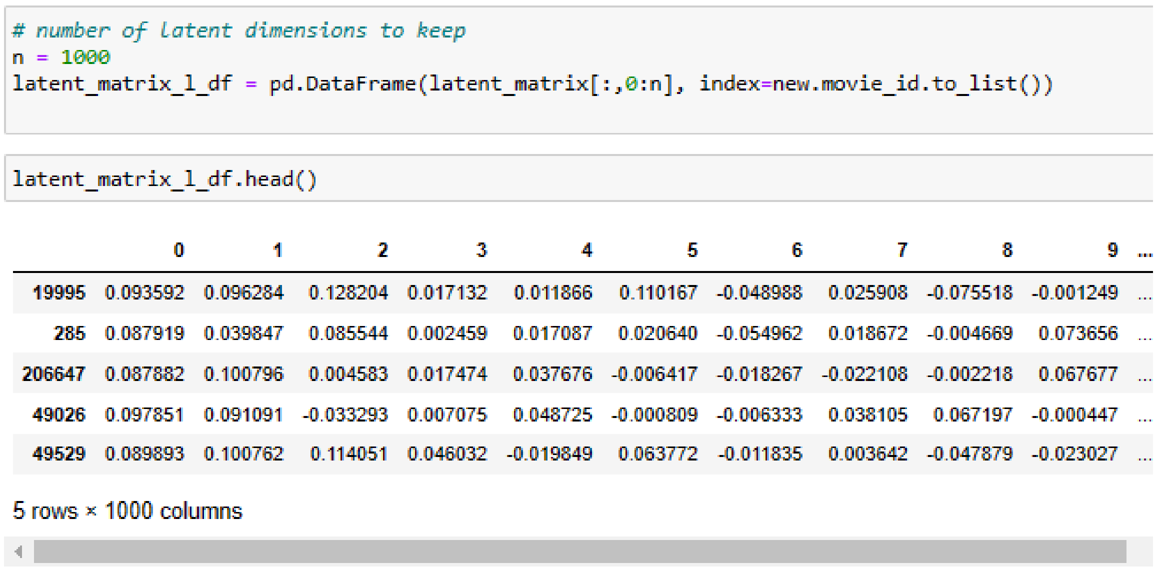


Figure 10

SVC vs. Cumulative percent varience

Number of latent dimensions to keep



```
# number of latent dimensions to keep
n = 1000
latent_matrix_1_df = pd.DataFrame(latent_matrix[:,0:n], index=new.movie_id.to_list())
```

```
latent_matrix_1_df.head()
```

	0	1	2	3	4	5	6	7	8	9	...
19995	0.093592	0.096284	0.128204	0.017132	0.011866	0.110167	-0.048988	0.025908	-0.075518	-0.001249	...
285	0.087919	0.039847	0.085544	0.002459	0.017087	0.020640	-0.054962	0.018672	-0.004669	0.073656	...
206647	0.087882	0.100796	0.004583	0.017474	0.037676	-0.006417	-0.018267	-0.022108	-0.002218	0.067677	...
49026	0.097851	0.091091	-0.033293	0.007075	0.048725	-0.000809	-0.006333	0.038105	0.067197	-0.000447	...
49529	0.089893	0.100762	0.114051	0.046032	-0.019849	0.063772	-0.011835	0.003642	-0.047879	-0.023027	...

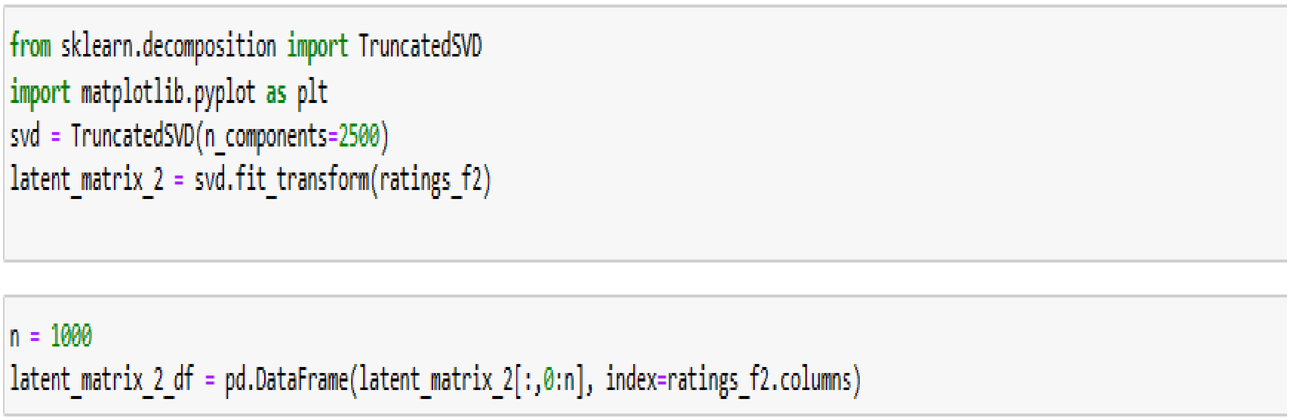
5 rows × 1000 columns

Reducing the dimenions of the collaborative matrix using SVD

Matrix decomposition, also known as matrix factorization, involves describing a given matrix using its constituent elements.

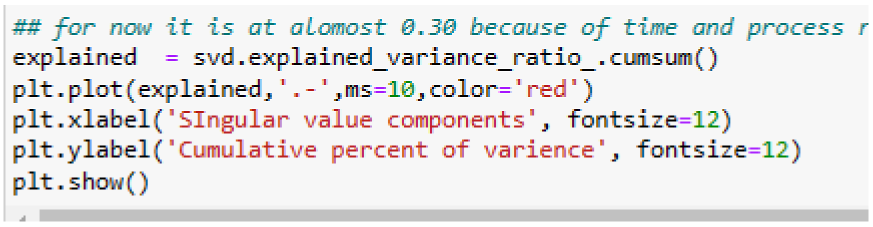
Perhaps the most known and widely used matrix decomposition method is the Singular-Value Decomposition, or SVD. All matrices have an SVD, which makes it more stable than other methods, such as the eigendecomposition. As such, it is often used in a wide array of applications including compressing, denoising, and data reduction.

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u}_\mathbf{i} \mathbf{v}_\mathbf{i}^T .$$

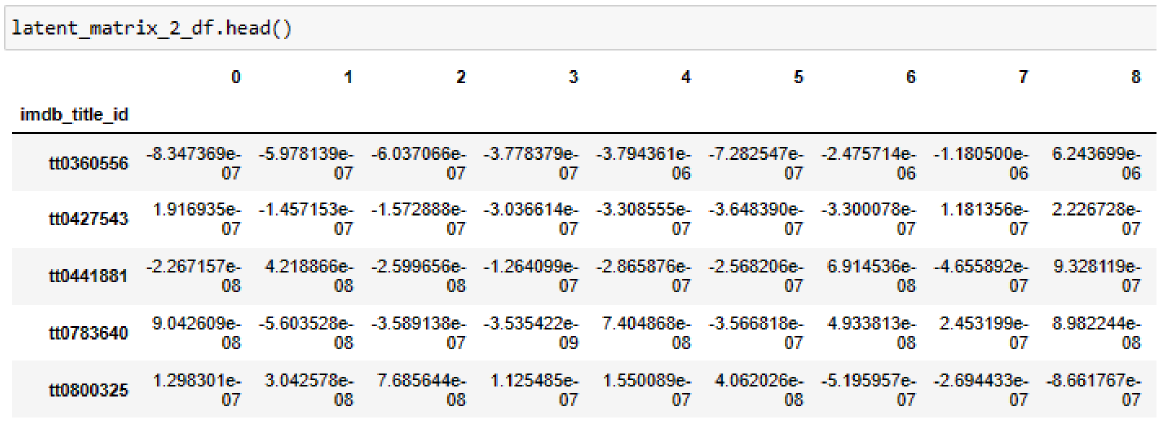


```
from sklearn.decomposition import TruncatedSVD
import matplotlib.pyplot as plt
svd = TruncatedSVD(n_components=2500)
latent_matrix_2 = svd.fit_transform(ratings_f2)
```

```
n = 1000
latent_matrix_2_df = pd.DataFrame(latent_matrix_2[:,0:n], index=ratings_f2.columns)
```



```python
## for now it is at alomost 0.30 because of time and process r
explained  = svd.explained_variance_ratio_.cumsum()
plt.plot(explained,'.-',ms=10,color='red')
plt.xlabel('SIngular value components', fontsize=12)
plt.ylabel('Cumulative percent of varience', fontsize=12)
plt.show()
```



```python
latent_matrix_2_df.head()
```

imdb_title_id	0	1	2	3	4	5	6	7	8
tt0360556	-8.347369e-07	-5.978139e-07	-6.037066e-07	-3.778379e-07	-3.794361e-06	-7.282547e-07	-2.475714e-06	-1.180500e-06	6.243699e-06
tt0427543	1.916935e-07	-1.457153e-07	-1.572888e-07	-3.036614e-07	-3.308555e-07	-3.648390e-07	-3.300078e-07	1.181356e-07	2.226728e-07
tt0441881	-2.267157e-08	4.218866e-08	-2.599656e-08	-1.264099e-07	-2.865876e-07	-2.568206e-07	6.914536e-08	-4.655892e-07	9.328119e-07
tt0783640	9.042609e-08	-5.603528e-08	-3.589138e-07	-3.535422e-09	7.404868e-08	-3.566818e-07	4.933813e-08	2.453199e-07	8.982244e-08
tt0800325	1.298301e-07	3.042578e-08	7.685644e-08	1.125485e-07	1.550089e-07	4.062026e-08	-5.195957e-07	-2.694433e-07	-8.661767e-07

Cosine Similarity: Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them.

Formula:

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

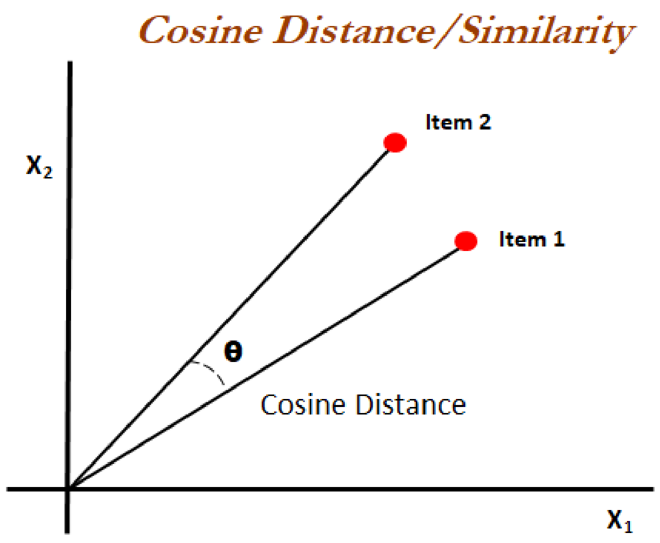
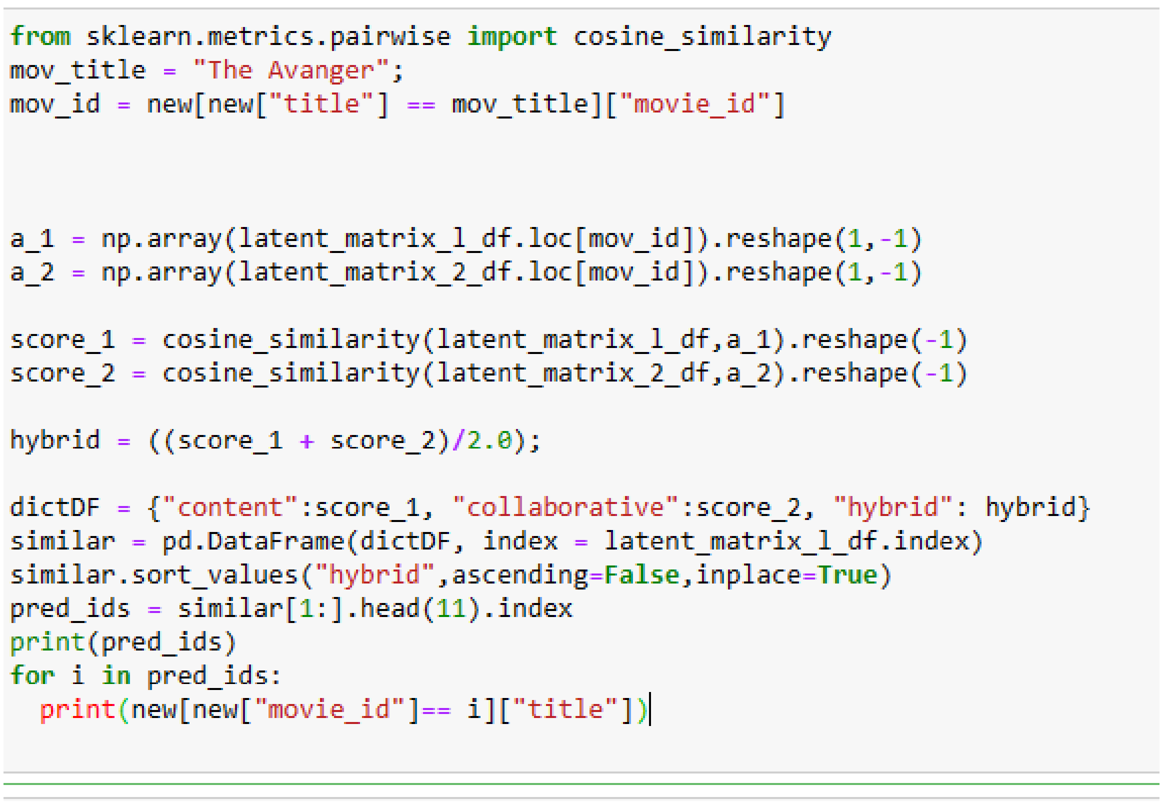
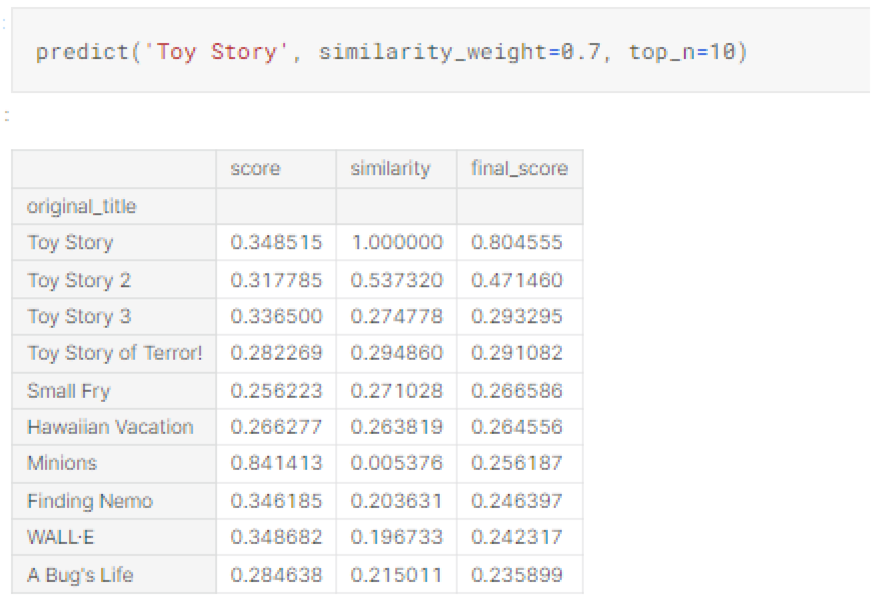


Figure 11. cosine distance



```python
from sklearn.metrics.pairwise import cosine_similarity
mov_title = "The Avanger";
mov_id = new[new["title"] == mov_title]["movie_id"]



a_1 = np.array(latent_matrix_l_df.loc[mov_id]).reshape(1,-1)
a_2 = np.array(latent_matrix_2_df.loc[mov_id]).reshape(1,-1)

score_1 = cosine_similarity(latent_matrix_l_df,a_1).reshape(-1)
score_2 = cosine_similarity(latent_matrix_2_df,a_2).reshape(-1)

hybrid = ((score_1 + score_2)/2.0);

dictDF = {"content":score_1, "collaborative":score_2, "hybrid": hybrid}
similar = pd.DataFrame(dictDF, index = latent_matrix_l_df.index)
similar.sort_values("hybrid",ascending=False,inplace=True)
pred_ids = similar[1:].head(11).index
print(pred_ids)
for i in pred_ids:
  print(new[new["movie_id"]== i]["title"])
```

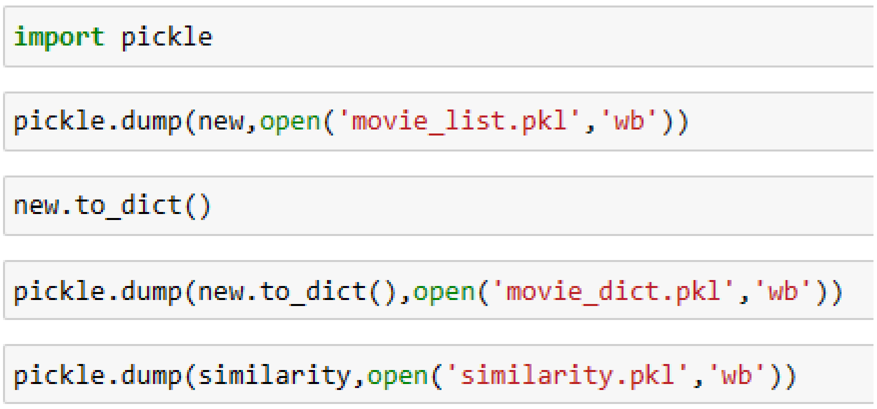


```python
predict('Toy Story', similarity_weight=0.7, top_n=10)
```

	score	similarity	final_score
original_title			
Toy Story	0.348515	1.000000	0.804555
Toy Story 2	0.317785	0.537320	0.471460
Toy Story 3	0.336500	0.274778	0.293295
Toy Story of Terror!	0.282269	0.294860	0.291082
Small Fry	0.256223	0.271028	0.266586
Hawaiian Vacation	0.266277	0.263819	0.264556
Minions	0.841413	0.005376	0.256187
Finding Nemo	0.346185	0.203631	0.246397
WALL·E	0.348682	0.196733	0.242317
A Bug's Life	0.284638	0.215011	0.235899

After we save model :

pickle module is a great way of storing python objects like tuple, dictionaries, lists, and even python classes and functions can be serialized and de-serialized. But it may not support cross-language, multiple python versions compatibility. Also, unpickling from unknown sources should be avoided as they may contain malicious, erroneous data.



```python
import pickle
```

```python
pickle.dump(new,open('movie_list.pkl','wb'))
```

```python
new.to_dict()
```

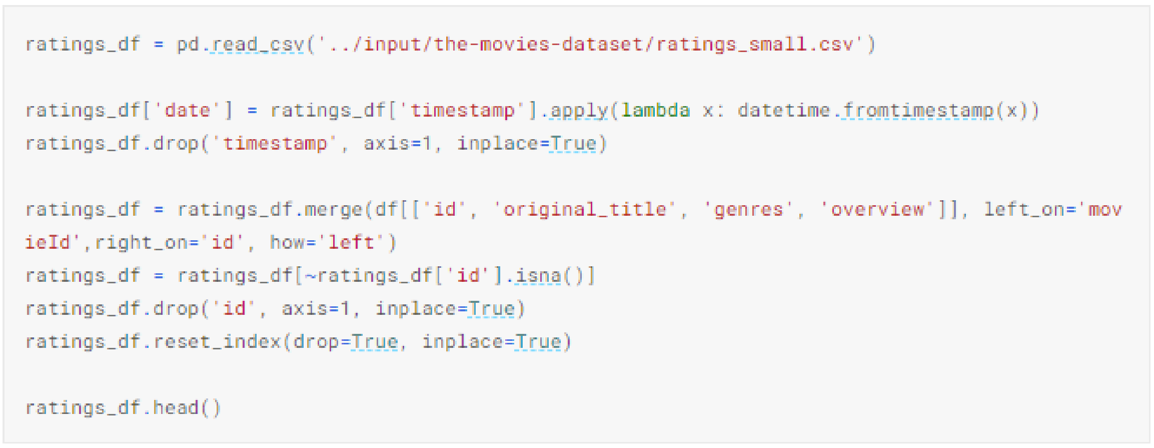
```python
pickle.dump(new.to_dict(),open('movie_dict.pkl','wb'))
```

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

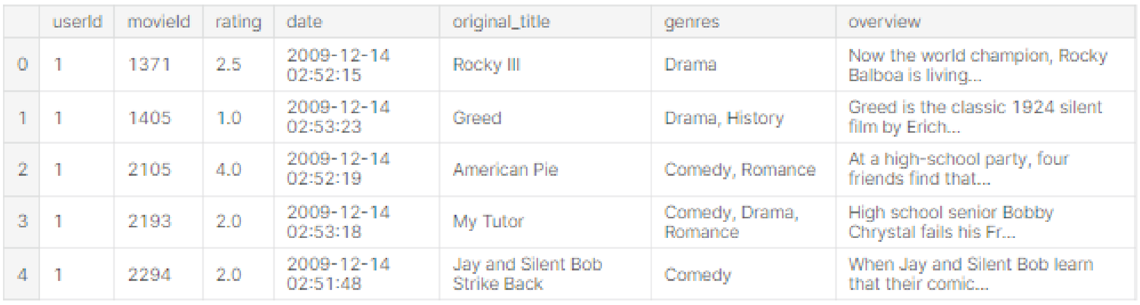
Deep Learning

Tensorflow comes with a library called TensorFlow Recommenders (TFRS) for building a recommender system. It's built on Keras and aims to have a gentle learning curve while still giving you the flexibility to build complex models.

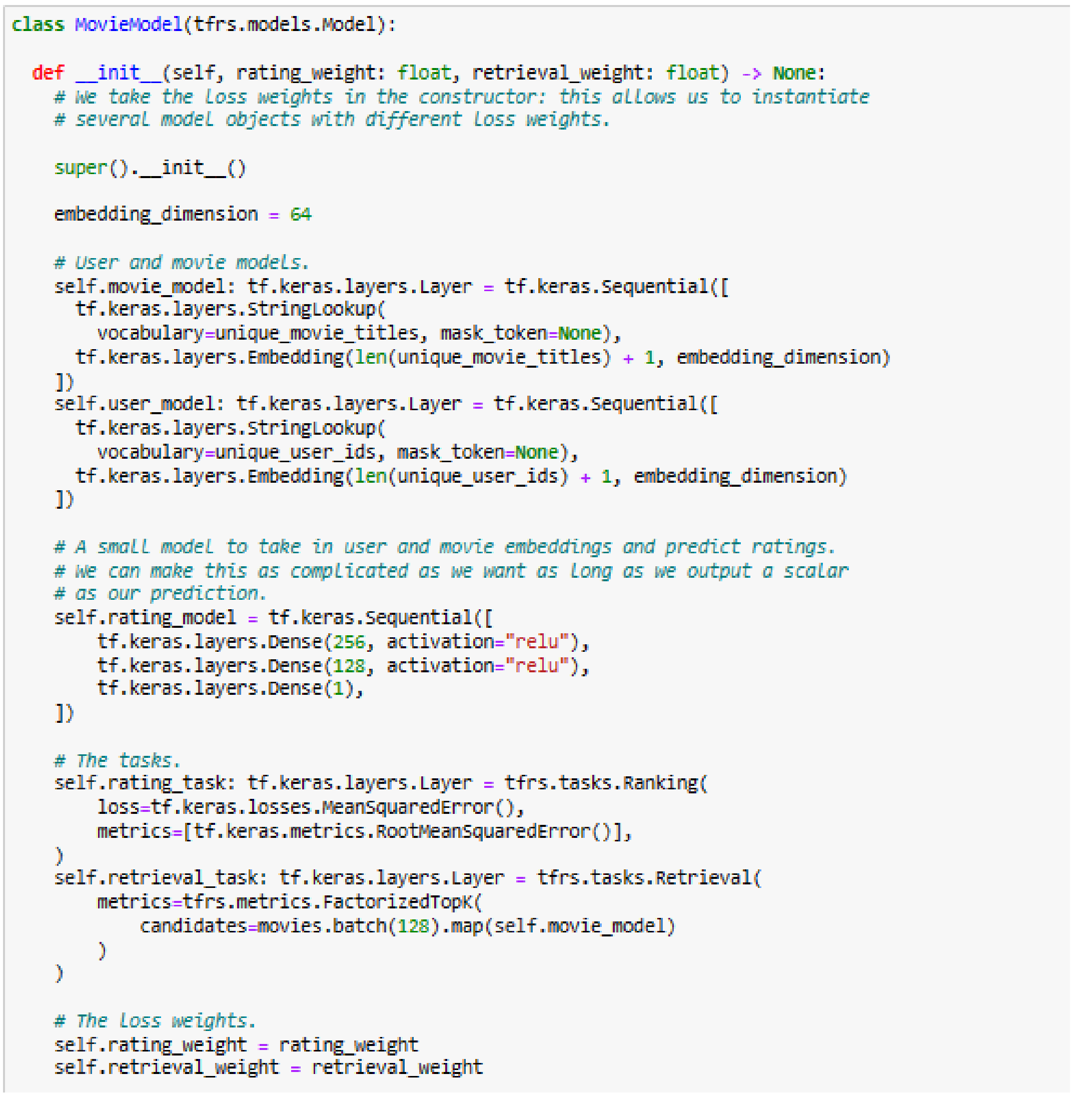
This time, we use multi-objective approach that applies both implicit (movie watches) and explicit signals (ratings). In the end, we can predict what movies should the user watch along with the given rating corresponds to historical data



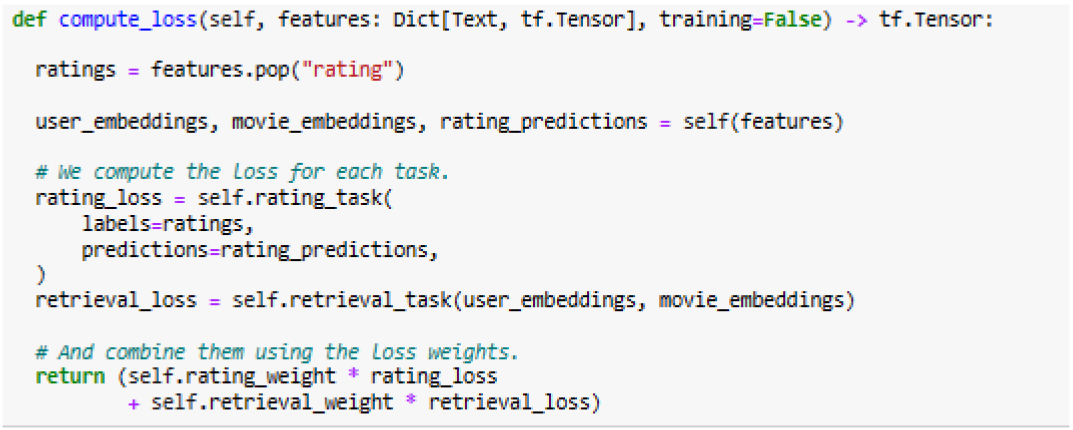
```python
ratings_df = pd.read_csv('../input/the-movies-dataset/ratings_small.csv')

ratings_df['date'] = ratings_df['timestamp'].apply(lambda x: datetime.fromtimestamp(x))
ratings_df.drop('timestamp', axis=1, inplace=True)

ratings_df = ratings_df.merge(df[['id', 'original_title', 'genres', 'overview']], left_on='mov
ieId',right_on='id', how='left')
ratings_df = ratings_df[~ratings_df['id'].isna()]
ratings_df.drop('id', axis=1, inplace=True)
ratings_df.reset_index(drop=True, inplace=True)

ratings_df.head()
```



	userId	movieId	rating	date	original_title	genres	overview
0	1	1371	2.5	2009-12-14 02:52:15	Rocky III	Drama	Now the world champion, Rocky Balboa is living...
1	1	1405	1.0	2009-12-14 02:53:23	Greed	Drama, History	Greed is the classic 1924 silent film by Erich...
2	1	2105	4.0	2009-12-14 02:52:19	American Pie	Comedy, Romance	At a high-school party, four friends find that...
3	1	2193	2.0	2009-12-14 02:53:18	My Tutor	Comedy, Drama, Romance	High school senior Bobby Chrystal fails his Fr...
4	1	2294	2.0	2009-12-14 02:51:48	Jay and Silent Bob Strike Back	Comedy	When Jay and Silent Bob learn that their comic...



```
class MovieModel(tfrs.models.Model):

  def __init__(self, rating_weight: float, retrieval_weight: float) -> None:
    # We take the loss weights in the constructor: this allows us to instantiate
    # several model objects with different loss weights.

    super().__init__()

    embedding_dimension = 64

    # User and movie models.
    self.movie_model: tf.keras.layers.Layer = tf.keras.Sequential([
      tf.keras.layers.StringLookup(
        vocabulary=unique_movie_titles, mask_token=None),
      tf.keras.layers.Embedding(len(unique_movie_titles) + 1, embedding_dimension)
    ])
    self.user_model: tf.keras.layers.Layer = tf.keras.Sequential([
      tf.keras.layers.StringLookup(
        vocabulary=unique_user_ids, mask_token=None),
      tf.keras.layers.Embedding(len(unique_user_ids) + 1, embedding_dimension)
    ])

    # A small model to take in user and movie embeddings and predict ratings.
    # We can make this as complicated as we want as long as we output a scalar
    # as our prediction.
    self.rating_model = tf.keras.Sequential([
        tf.keras.layers.Dense(256, activation="relu"),
        tf.keras.layers.Dense(128, activation="relu"),
        tf.keras.layers.Dense(1),
    ])

    # The tasks.
    self.rating_task: tf.keras.layers.Layer = tfrs.tasks.Ranking(
        loss=tf.keras.losses.MeanSquaredError(),
        metrics=[tf.keras.metrics.RootMeanSquaredError()],
    )
    self.retrieval_task: tf.keras.layers.Layer = tfrs.tasks.Retrieval(
        metrics=tfrs.metrics.FactorizedTopK(
            candidates=movies.batch(128).map(self.movie_model)
        )
    )

    # The loss weights.
    self.rating_weight = rating_weight
    self.retrieval_weight = retrieval_weight
```



```
def compute_loss(self, features: Dict[Text, tf.Tensor], training=False) -> tf.Tensor:

  ratings = features.pop("rating")

  user_embeddings, movie_embeddings, rating_predictions = self(features)

  # We compute the loss for each task.
  rating_loss = self.rating_task(
      labels=ratings,
      predictions=rating_predictions,
  )
  retrieval_loss = self.retrieval_task(user_embeddings, movie_embeddings)

  # And combine them using the loss weights.
  return (self.rating_weight * rating_loss
          + self.retrieval_weight * retrieval_loss)
```

```
model = MovieModel(rating_weight=1.0, retrieval_weight=1.0)
model.compile(optimizer=tf.keras.optimizers.Adagrad(0.1))

cached_train = train.shuffle(100_000).batch(1_000).cache()
cached_test = test.batch(1_000).cache()

model.fit(cached_train, epochs=3)
```

```
Epoch 1/3
35/35 [==============================] - 60s 2s/step - root_mean_squared_error: 1.5516 - fa
ctorized_top_k/top_1_categorical_accuracy: 4.8571e-04 - factorized_top_k/top_5_categorical_
accuracy: 0.0076 - factorized_top_k/top_10_categorical_accuracy: 0.0181 - factorized_top_k/
top_50_categorical_accuracy: 0.1027 - factorized_top_k/top_100_categorical_accuracy: 0.1715
- loss: 6811.1486 - regularization_loss: 0.0000e+00 - total_loss: 6811.1486
Epoch 2/3
35/35 [==============================] - 58s 2s/step - root_mean_squared_error: 1.0156 - fa
ctorized_top_k/top_1_categorical_accuracy: 0.0011 - factorized_top_k/top_5_categorical_accu
racy: 0.0194 - factorized_top_k/top_10_categorical_accuracy: 0.0449 - factorized_top_k/top_
50_categorical_accuracy: 0.2020 - factorized_top_k/top_100_categorical_accuracy: 0.3214 - l
oss: 6450.4905 - regularization_loss: 0.0000e+00 - total_loss: 6450.4905
Epoch 3/3
35/35 [==============================] - 57s 2s/step - root_mean_squared_error: 0.9882 - fa
ctorized_top_k/top_1_categorical_accuracy: 6.8571e-04 - factorized_top_k/top_5_categorical_
accuracy: 0.0257 - factorized_top_k/top_10_categorical_accuracy: 0.0568 - factorized_top_k/
top_50_categorical_accuracy: 0.2430 - factorized_top_k/top_100_categorical_accuracy: 0.3784
- loss: 6186.2205 - regularization_loss: 0.0000e+00 - total_loss: 6186.2205
```

```
metrics = model.evaluate(cached_test, return_dict=True)

print(f"\nRetrieval top-100 accuracy: {metrics['factorized_top_k/top_100_categorical_accurac
y']:.3f}")
print(f"Ranking RMSE: {metrics['root_mean_squared_error']:.3f}")
```

```
9/9 [==============================] - 12s 1s/step - root_mean_squared_error: 1.0954 - fact
orized_top_k/top_1_categorical_accuracy: 0.0011 - factorized_top_k/top_5_categorical_accura
cy: 0.0050 - factorized_top_k/top_10_categorical_accuracy: 0.0101 - factorized_top_k/top_50
_categorical_accuracy: 0.0459 - factorized_top_k/top_100_categorical_accuracy: 0.0859 - los
s: 5724.6355 - regularization_loss: 0.0000e+00 - total_loss: 5724.6355

Retrieval top-100 accuracy: 0.086
Ranking RMSE: 1.095
```

```
def predict_movie(user, top_n=3):
    # Create a model that takes in raw query features, and
    index = tfrs.layers.factorized_top_k.BruteForce(model.user_model)
    # recommends movies out of the entire movies dataset.
    index.index_from_dataset(
      tf.data.Dataset.zip((movies.batch(100), movies.batch(100).map(model.movie_model)))
    )

    # Get recommendations.
    _, titles = index(tf.constant([str(user)]))

    print('Top {} recommendations for user {}:\n'.format(top_n, user))
    for i, title in enumerate(titles[0, :top_n].numpy()):
        print('{}. {}'.format(i+1, title.decode("utf-8")))

def predict_rating(user, movie):
    trained_movie_embeddings, trained_user_embeddings, predicted_rating = model({
          "userId": np.array([str(user)]),
          "original_title": np.array([movie])
      })
    print("Predicted rating for {}: {}".format(movie, predicted_rating.numpy()[0][0]))
```

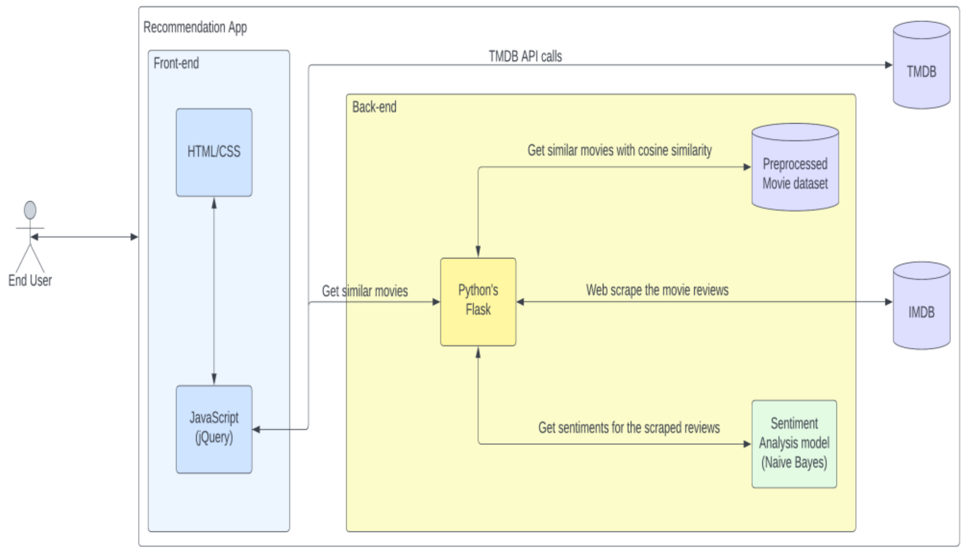
```
predict_movie(123, 5)
```

```
Top 5 recommendations for user 123:

1. Scary Movie
2. Anatomie de l'enfer
3. The Greatest Story Ever Told
4. Un long dimanche de fiançailles
5. Jezebel
```

```
predict_rating(123,'Minions')
```

```
Predicted rating for Minions: 3.088733196258545
```

The movies are recommended based on the content of the movie you entered or selected. The main parameters that are considered for the recommendations are the genre, director, and top 3 casts. The details of the movies, such as title, genre, runtime, rating, poster, casts, etc., are fetched from TMDB. The reviews of each individual movie given by the users are "web-scraped" from the IMDB website with the help of beautifulsoup4, and the reviews are subjected to sentiment analysis, where the model predicts whether the review is positive or negative.



Recommendation App
Front-end
TMDB API calls
TMDB
Back-end
HTML/CSS
Get similar movies with cosine similarity
Preprocessed Movie dataset
End User
Get similar movies
Python's Flask
Web scrape the movie reviews
IMDB
JavaScript (jQuery)
Get sentiments for the scraped reviews
Sentiment Analysis model (Naive Bayes)

Fig.12 Recommendation app flow

Code: Front-end

The Front end development involves the development of the visual components of a website. This includes the design and development of the web page layout, the colour scheme, the navigation, and the user interface elements. It also involves the creation of the HTML code that is used to structure the page, the CSS code that is used to style the page, and the JavaScript code that is used to add interactivity to the page.

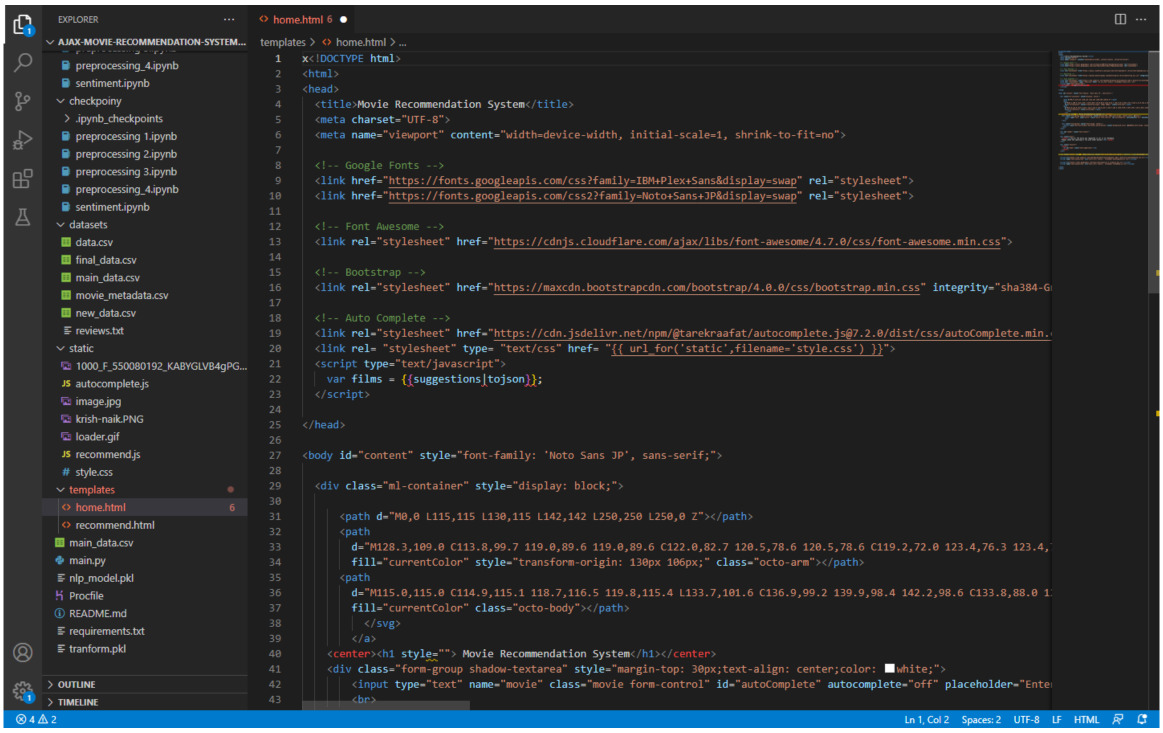
Html/css/javascript:

The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

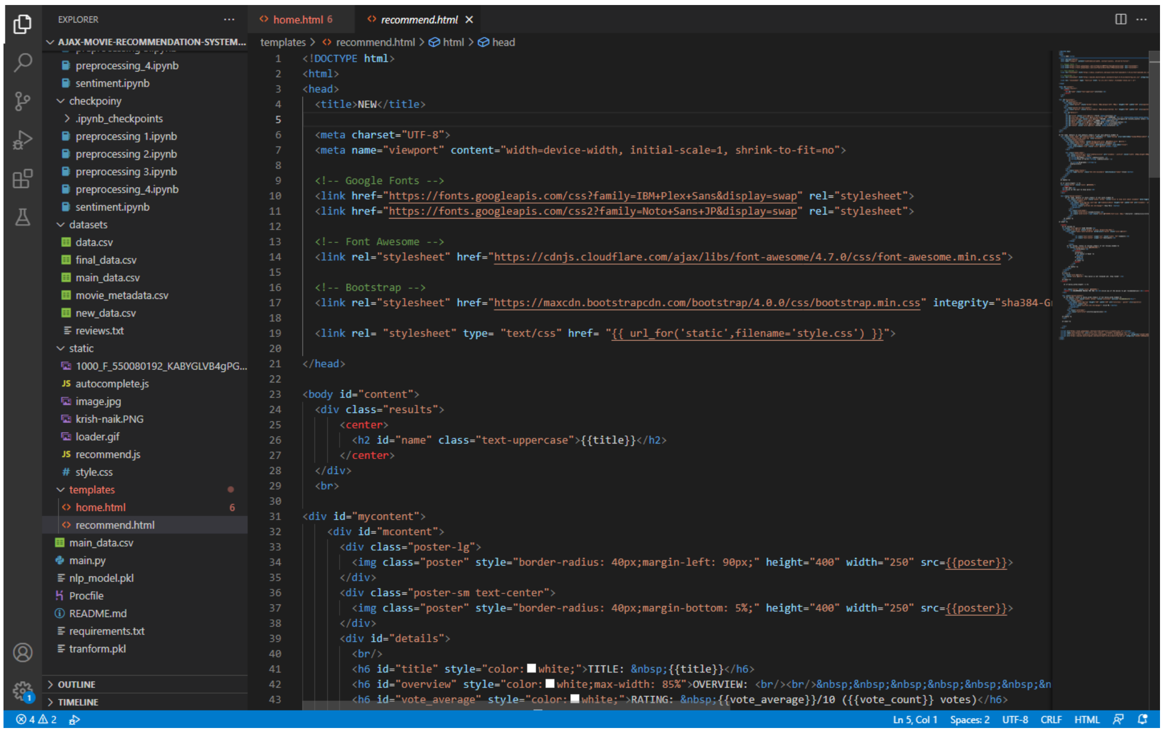
HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes, and other items. HTML elements are delineated by *tags*, written using angle brackets. Tags such as `<img />` and `<input />` directly introduce content into the page. Other tags such as `<p>` surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags but use them to interpret the content of the page.

Home.html



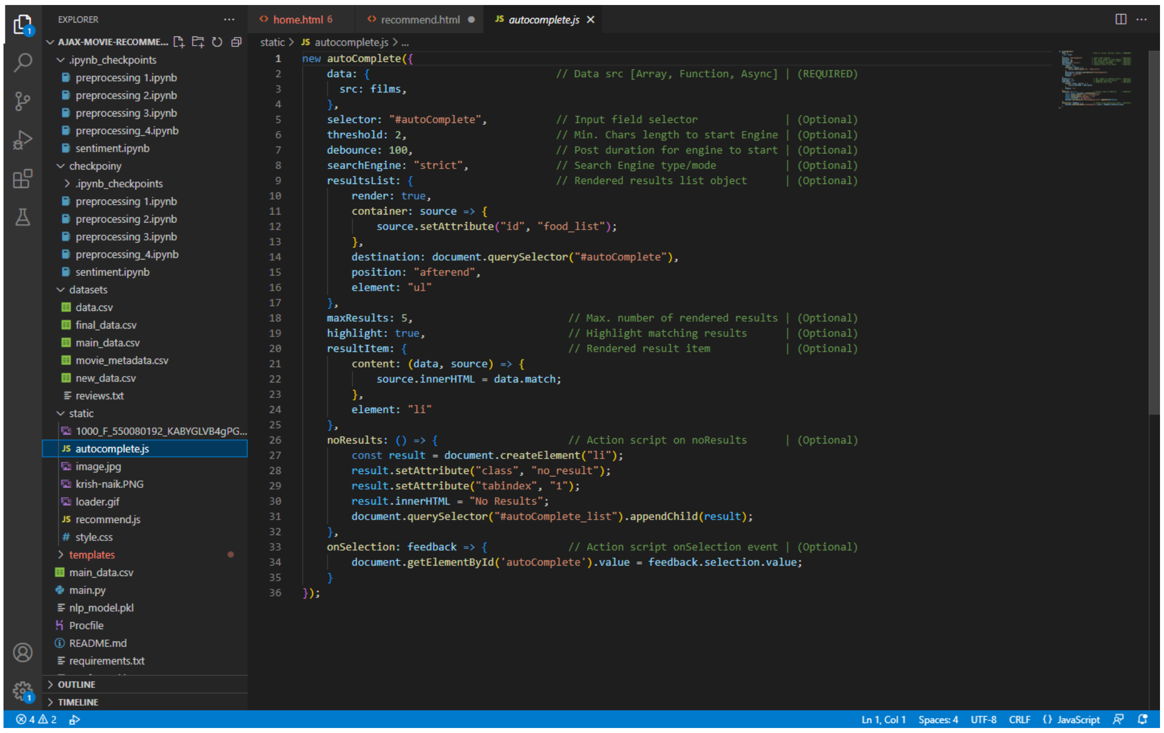
```html
x<!DOCTYPE html>
<html>
<head>
  <title>Movie Recommendation System</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Google Fonts -->
  <link href="https://fonts.googleapis.com/css?family=IBM+Plex+Sans&display=swap" rel="stylesheet">
  <link href="https://fonts.googleapis.com/css2?family=Noto+Sans+JP&display=swap" rel="stylesheet">

  <!-- Font Awesome -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">

  <!-- Bootstrap -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-G

  <!-- Auto Complete -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@tarekraafat/autocomplete.js@7.2.0/dist/css/autoComplete.min.
  <link rel= "stylesheet" type= "text/css" href= "{{ url_for('static',filename='style.css') }}">
  <script type="text/javascript">
    var films = {{suggestions|tojson}};
  </script>

</head>

<body id="content" style="font-family: 'Noto Sans JP', sans-serif;">

  <div class="ml-container" style="display: block;">

      <path d="M0,0 L115,115 L130,115 L142,142 L250,250 L250,0 Z"></path>
      <path
        d="M128.3,109.0 C113.8,99.7 119.0,89.6 119.0,89.6 C122.0,82.7 120.5,78.6 120.5,78.6 C119.2,72.0 123.4,76.3 123.4,
        fill="currentColor" style="transform-origin: 130px 106px;" class="octo-arm"></path>
      <path
        d="M115.0,115.0 C114.9,115.1 118.7,116.5 119.8,115.4 L133.7,101.6 C136.9,99.2 139.9,98.4 142.2,98.6 C133.8,88.0 1
        fill="currentColor" class="octo-body"></path>
          </svg>
        </a>
    <center><h1 style=""> Movie Recommendation System</h1></center>
    <div class="form-group shadow-textarea" style="margin-top: 30px;text-align: center;color: white;">
        <input type="text" name="movie" class="movie form-control" id="autoComplete" autocomplete="off" placeholder="Ente
        <br>
```

Recommend.html



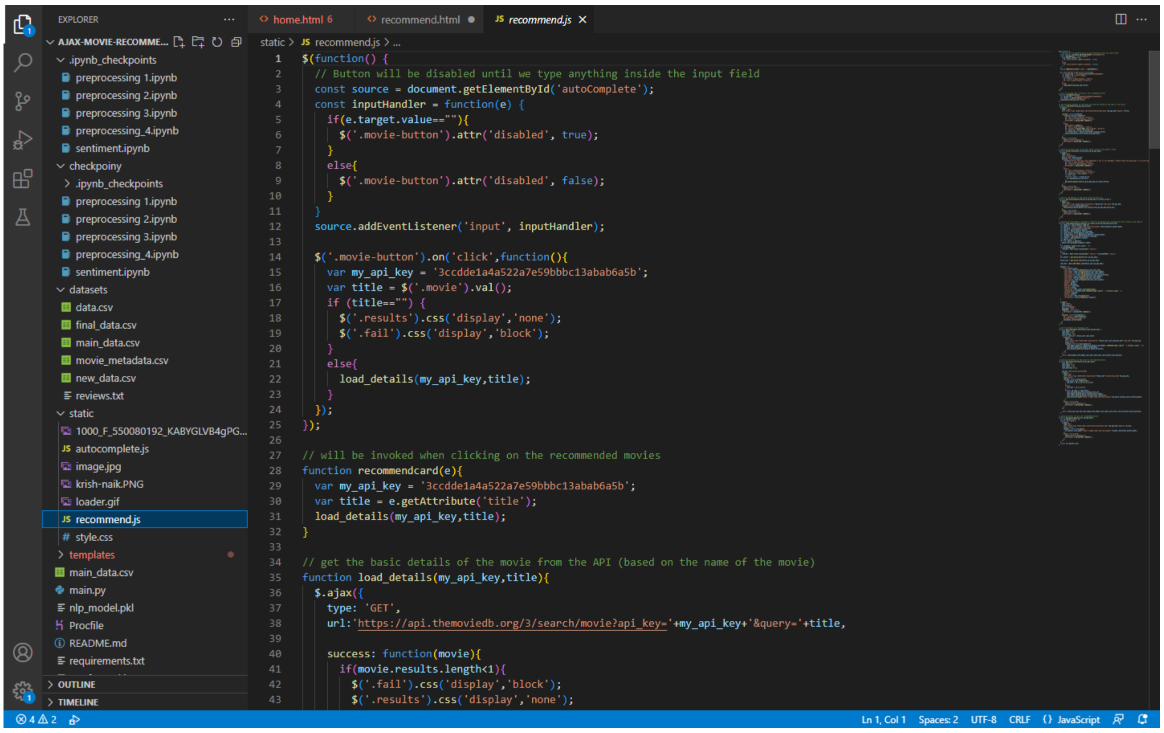
```html
<!DOCTYPE html>
<html>
<head>
  <title>NEW</title>

  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Google Fonts -->
  <link href="https://fonts.googleapis.com/css?family=IBM+Plex+Sans&display=swap" rel="stylesheet">
  <link href="https://fonts.googleapis.com/css2?family=Noto+Sans+JP&display=swap" rel="stylesheet">

  <!-- Font Awesome -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">

  <!-- Bootstrap -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-G

  <link rel= "stylesheet" type= "text/css" href= "{{ url_for('static',filename='style.css') }}">

</head>

<body id="content">
  <div class="results">
      <center>
        <h2 id="name" class="text-uppercase">{{title}}</h2>
      </center>
  </div>
  <br>

<div id="mycontent">
    <div id="mcontent">
      <div class="poster-lg">
        <img class="poster" style="border-radius: 40px;margin-left: 90px;" height="400" width="250" src={{poster}}>
      </div>
      <div class="poster-sm text-center">
        <img class="poster" style="border-radius: 40px;margin-bottom: 5%;" height="400" width="250" src={{poster}}>
      </div>
      <div id="details">
        <br/>
        <h6 id="title" style="color:white;">TITLE: &nbsp;{{title}}</h6>
        <h6 id="overview" style="color:white;max-width: 85%">OVERVIEW: <br/><br/>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&n
        <h6 id="vote_average" style="color:white;">RATING: &nbsp;{{vote_average}}/10 ({{vote_count}} votes)</h6>
```

Autocomplete.js



```javascript
new autoComplete({
    data: {                              // Data src [Array, Function, Async] | (REQUIRED)
      src: films,
    },
    selector: "#autoComplete",           // Input field selector              | (Optional)
    threshold: 2,                        // Min. Chars length to start Engine | (Optional)
    debounce: 100,                       // Post duration for engine to start | (Optional)
    searchEngine: "strict",              // Search Engine type/mode           | (Optional)
    resultsList: {                       // Rendered results list object      | (Optional)
        render: true,
        container: source => {
            source.setAttribute("id", "food_list");
        },
        destination: document.querySelector("#autoComplete"),
        position: "afterend",
        element: "ul"
    },
    maxResults: 5,                         // Max. number of rendered results | (Optional)
    highlight: true,                       // Highlight matching results      | (Optional)
    resultItem: {                          // Rendered result item            | (Optional)
        content: (data, source) => {
            source.innerHTML = data.match;
        },
        element: "li"
    },
    noResults: () => {                     // Action script on noResults      | (Optional)
        const result = document.createElement("li");
        result.setAttribute("class", "no_result");
        result.setAttribute("tabindex", "1");
        result.innerHTML = "No Results";
        document.querySelector("#autoComplete_list").appendChild(result);
    },
    onSelection: feedback => {             // Action script onSelection event | (Optional)
        document.getElementById('autoComplete').value = feedback.selection.value;
    }
});
```

Recommend.js



```
$(function() {
  // Button will be disabled until we type anything inside the input field
  const source = document.getElementById('autoComplete');
  const inputHandler = function(e) {
    if(e.target.value==""){
      $('.movie-button').attr('disabled', true);
    }
    else{
      $('.movie-button').attr('disabled', false);
    }
  }
  source.addEventListener('input', inputHandler);

  $('.movie-button').on('click',function(){
    var my_api_key = '3ccdde1a4a522a7e59bbbc13abab6a5b';
    var title = $('.movie').val();
    if (title=="") {
      $('.results').css('display','none');
      $('.fail').css('display','block');
    }
    else{
      load_details(my_api_key,title);
    }
  });
});

// will be invoked when clicking on the recommended movies
function recommendcard(e){
  var my_api_key = '3ccdde1a4a522a7e59bbbc13abab6a5b';
  var title = e.getAttribute('title');
  load_details(my_api_key,title);
}

// get the basic details of the movie from the API (based on the name of the movie)
function load_details(my_api_key,title){
  $.ajax({
    type: 'GET',
    url:'https://api.themoviedb.org/3/search/movie?api_key='+my_api_key+'&query='+title,

    success: function(movie){
      if(movie.results.length<1){
        $('.fail').css('display','block');
        $('.results').css('display','none');
```

API Overview

Our API is available for everyone to use. A TMDB user account is required to request an API key. Professional users are approved on a per application basis.

As always, you must attribute TMDB as the source of your data.

API Documentation

To view all the methods available, you should head over to developers.themoviedb.org. Everything outlined on this page is simply a high level overview to help you understand what is available.

What is TMDB's API?

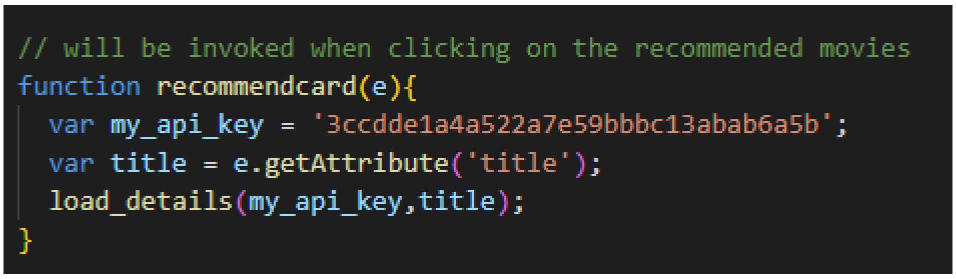
The API service is for those of you interested in using our movie, TV show or actor images and/or data in your application. Our API is a system we provide for you and your team to programmatically fetch and use our data and/or images.

Why would I need an API?

The API provides a fast, consistent and reliable way to get third party data.

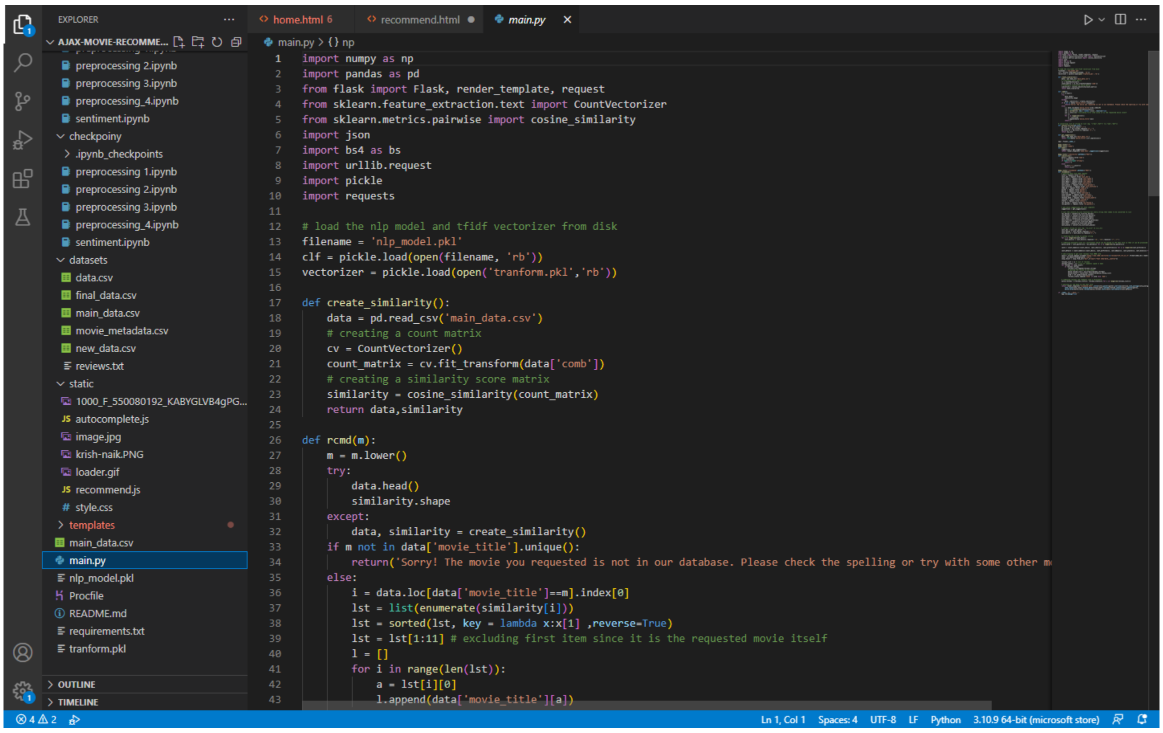
How to get the API key?

Create an account in https://www.themoviedb.org/. Once you successfully created an account, click on the API link from the left hand sidebar in your account settings and fill all the details to apply for an API key. If you are asked for the website URL, just give "NA" if you don't have one. You will see the API key in your API sidebar once your request has been approved.



```
// will be invoked when clicking on the recommended movies
function recommendcard(e){
  var my_api_key = '3ccdde1a4a522a7e59bbbc13abab6a5b';
  var title = e.getAttribute('title');
  load_details(my_api_key,title);
}
```

BackEnd code:



```
import numpy as np
import pandas as pd
from flask import Flask, render_template, request
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import json
import bs4 as bs
import urllib.request
import pickle
import requests

# load the nlp model and tfidf vectorizer from disk
filename = 'nlp_model.pkl'
clf = pickle.load(open(filename, 'rb'))
vectorizer = pickle.load(open('tranform.pkl','rb'))

def create_similarity():
    data = pd.read_csv('main_data.csv')
    # creating a count matrix
    cv = CountVectorizer()
    count_matrix = cv.fit_transform(data['comb'])
    # creating a similarity score matrix
    similarity = cosine_similarity(count_matrix)
    return data,similarity

def rcmd(m):
    m = m.lower()
    try:
        data.head()
        similarity.shape
    except:
        data, similarity = create_similarity()
    if m not in data['movie_title'].unique():
        return('Sorry! The movie you requested is not in our database. Please check the spelling or try with some other m
    else:
        i = data.loc[data['movie_title']==m].index[0]
        lst = list(enumerate(similarity[i]))
        lst = sorted(lst, key = lambda x:x[1] ,reverse=True)
        lst = lst[1:11] # excluding first item since it is the requested movie itself
        l = []
        for i in range(len(lst)):
            a = lst[i][0]
            l.append(data['movie_title'][a])
```

We use flask

- Flask is an API of Python that allows us to build up web-applications.
- Getting Started With Flask: Python 2.6 or higher is required for the installation of the Flask. You can start by import Flask from the flask package on any python IDE.
- By using flask, we generate resulting api which stores the data in the form of json format these data is retrieved in react by using axios npm mode and then displaying the data.

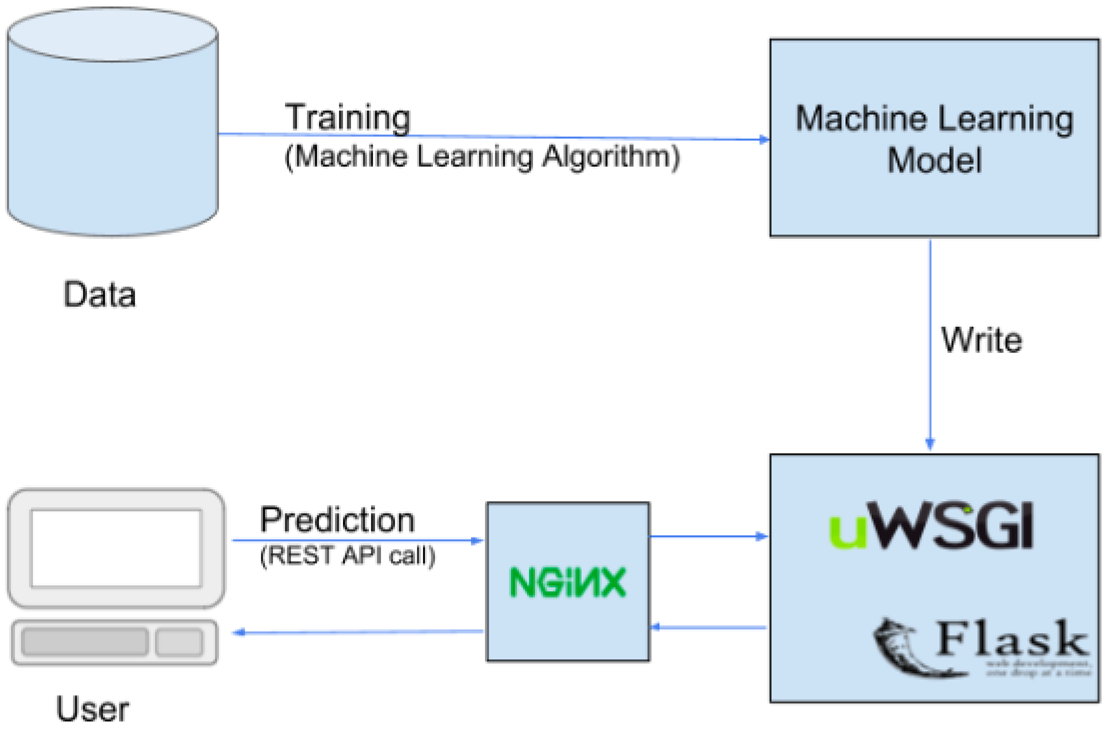
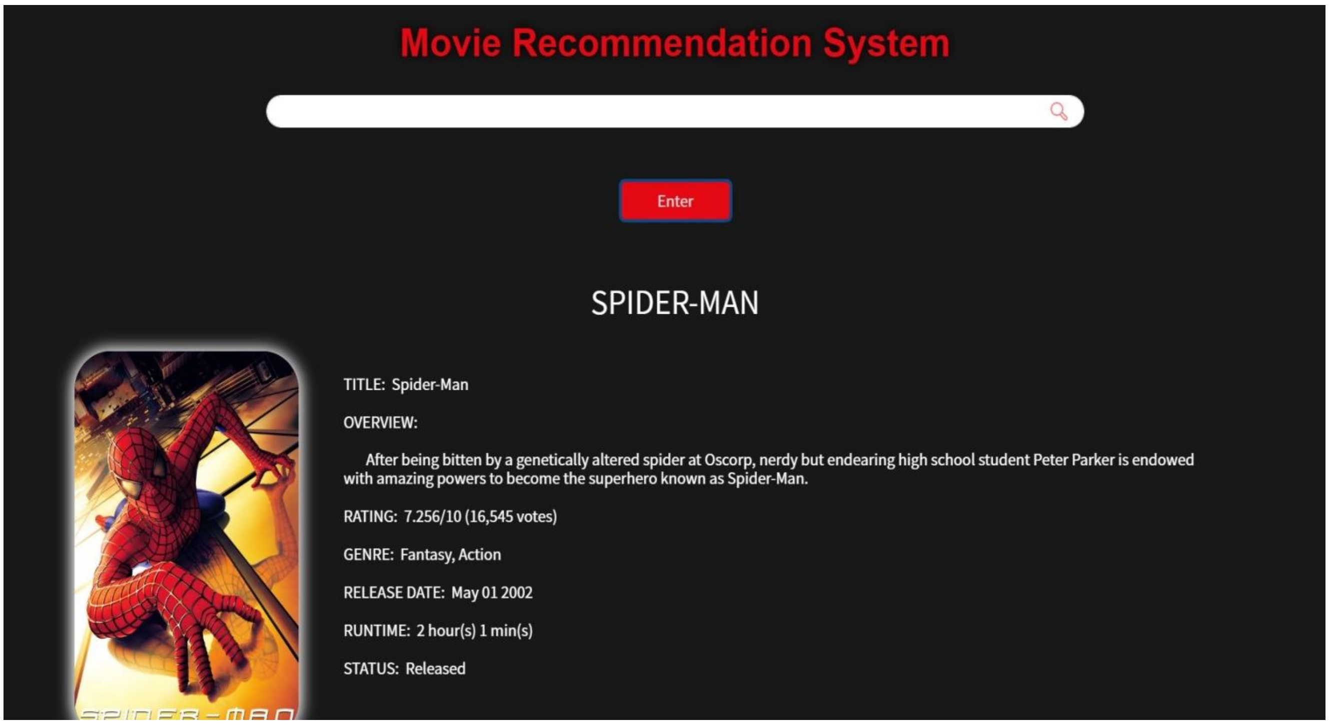


Fig.13 Flask Implementaion

Result



SPIDER-MAN

TITLE: Spider-Man

OVERVIEW:

After being bitten by a genetically altered spider at Oscorp, nerdy but endearing high school student Peter Parker is endowed with amazing powers to become the superhero known as Spider-Man.

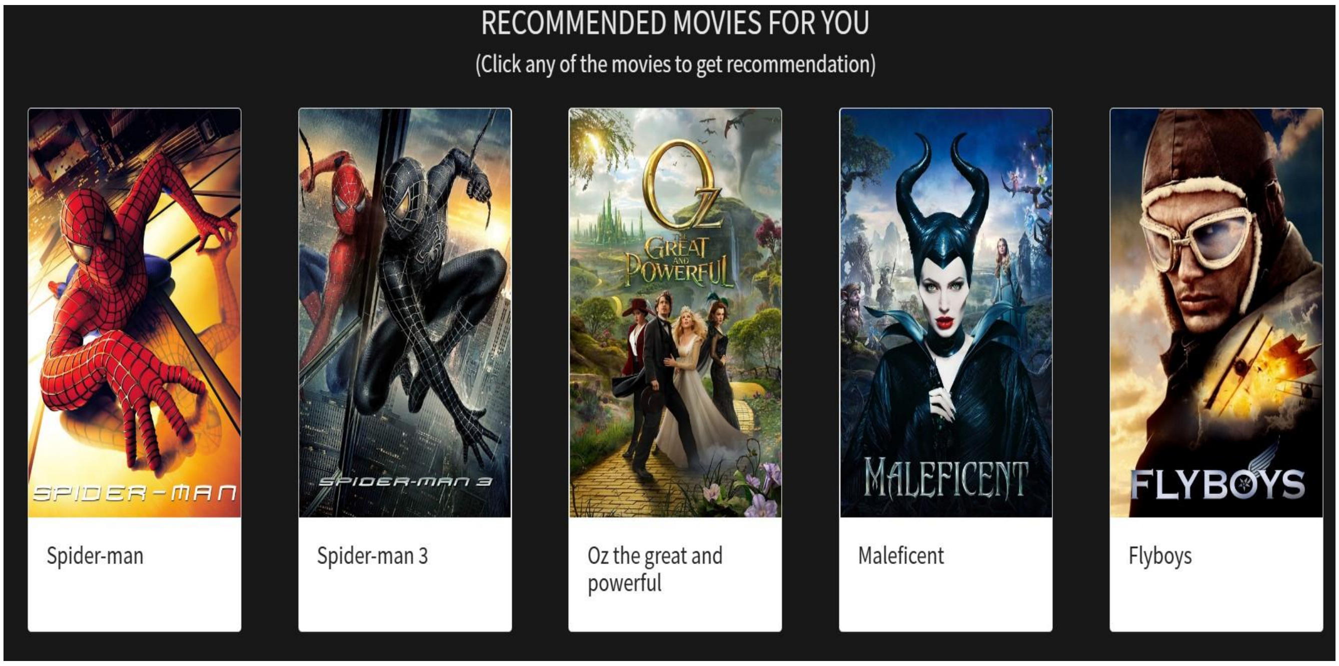
RATING: 7.256/10 (16,545 votes)

GENRE: Fantasy, Action

RELEASE DATE: May 01 2002

RUNTIME: 2 hour(s) 1 min(s)

STATUS: Released



RECOMMENDED MOVIES FOR YOU

(Click any of the movies to get recommendation)

Spider-man

Spider-man 3

Oz the great and powerful

Maleficent

Flyboys

Analysis

The hybrid approach will resolves all these limitations by combining both content and collaborative filtering.

No need domain knowledge because the embedding are automatically learned. The model can help users discover new interests. So, if an item is not seen during training, the system can't create an embedding for it and can't query the model with this item. This issue is often called the cold-start problem.

The main disadvantage in hybrid approach is it require high memory.

Conclusion

We conclude that movie recommendation system that combines both content-based and collaborative filtering techniques to provide users with accurate, personalized recommendations. It is capable of providing more accurate and detailed results than either technique alone, making it an ideal choice for movie recommendation systems. The hybrid-based system also has the capability to take into account user preferences and contextual information to provide more tailored and relevant results. By taking into account both user preferences and content-based information, the hybrid-based system is able to generate more accurate and meaningful recommendations.

References

- Suvir Bhargav. Efficient features for movie recommendation systems. 2014
- Manoj Kumar, D.K. Yadav, Ankur Singh and Vijay Kr. Gupta (2015), "A Movie Recommender System: MOVREC", International Journal of Computer Applications (0975 – 8887) Volume 124 – No.3.
- RyuRi Kim, Ye Jeong Kwak, HyeonJeong Mo, Mucheol Kim, Seungmin Rho,Ka Lok Man, Woon Kian Chong (2015),"Trustworthy Movie Recommender System with Correct Assessment and Emotion Evaluation", Proceedings of the International MultiConference of Engineers and Computer Scientists Vol II.
- Zan Wang, Xue Yu*, Nan Feng, Zhenhua Wang (2014), "An Improved Collaborative Movie Recommendation System using Computational Intelligence",Journal of Visual Languages & Computing,Volume 25, Issue 6.
- Debadrita Roy, Arnab Kundu, (2013), "Design of Movie Recommendation System by Means of Collaborative Filtering", International Journal of Emerging Technology and Advanced Engineering, Volume 3, Issue 4.
- Important links
  - https://www.themoviedb.org/documentation/api
  - https://www.themoviedb.org/settings/api
  - https://beta.openai.com/playground?model%3Ddavinci
  - https://www.ijcaonline.org/research
  - https://flask.palletsprojects.com/en/2.2.x/