PROJECT REPORT

MOVIE RECOMMENDATION SYSTEM USING MACHINE LEARNING

delivered by

Student name	Exam C. no.
Dhakane Pooja Shesharao	C22019772804
Pokharkar Sayali Dipak	C22019772801
Thombare Kiran Bhaskar	C22019772803

in partial fulfillment for the award of the degree of

Bachelor of Technology in

ELECTRONICS AND TELECOMMUNICATION of SAVITRIBAI PHULE PUNE UNIVERSITY,

under the guidance of
Prof. Mrs. Vidya Sisale

Sponsored by : -In-house

in the Department of Electronics and Telecommunication of CUMMINS COLLEGE OF ENGINEERING FOR WOMEN , KARVENAGAR, PUNE - 411052 (An Autonomous Institute affiliated to SAVITRIBAI PHULE PUNE UNIVERSITY)

Academic year
2020 - 2021

a) Project title : -Movie Recommendation System using ML

b) Subject area : -Machine Learning

c) Nature of the Project : - Software

CERTIFICATE

This is to certify that

Dhakane Pooja Shesharao
Pokharkar Sayali Dipak
Thombare Kiran Bhaskar

have presented a SEMINAR on their PROJECT TOPIC

Movie Recommendation System using Machine Learning

in partial fulfillment for the award of the degree of

Bachelor of Technology in ELECTRONICS AND TELECOMMUNICATION of SAVITRIBAI PHULE PUNE UNIVERSITY,

in

CUMMINS COLLEGE OF ENGINEERING FOR WOMEN , KARVENAGAR , PUNE-52 (An Autonomous Institute affiliated to SAVITRIBAI PHULE PUNE UNIVERSITY)

Internal Guide	**Head of the Department**	**Principal**
[Prof.(Mrs)Vidya Sisale]	[Dr.Prachi Mukherji]	[Dr.(Mrs.)M.B.Khambete]

Acknowledgment

I would like to express our sincere gratitude towards our internal guide Prof. Mrs. Vidya Sisale Madam for their constant encouragement and valuable guidance during the completion of this project work. We would also like to thank Dr. Prachi Mukherji (H.O.D., E&TC). For her continuous valuable guidance, support, valuable suggestions, and her precious time in every possible way in spite of her busy schedule throughout our project. I take this opportunity to express our sincere thanks to all the staff members of the E&TC Department for their constant help whenever required. Finally, we express our sincere thanks to all those who helped me directly or indirectly in many ways in the completion of this project.

Abstract

Algorithms are used in recommendation systems to give users with useful product or service recommendations. Machine learning techniques from the field of artificial intelligence have recently been used in recommendation systems. There are four types of recommendation systems: simple system based on popularity, collaborative filtering, content-based approach, and hybrid-based approach.

Because of its ability to provide increased amusement, a movie recommender system has become a significant component of our social lives. Based on different characteristics such as the popularity of the films, such a system will estimate what movies a user will enjoy. Several movie recommendation systems have been suggested and deployed.

Table of Contents

List of Figures

1. Introduction

Recommender systems are designed to anticipate customers' needs and recommend products that are likely to excite them. They are a few of the most successful system learning structures that online stores impose in order to boost sales.

Data for recommender systems comes from explicit consumer scores after watching a movie or listening to a song, from implicit search engine inquiries and purchase histories, or from other understandings about the customers or gadgets themselves.

Spotify, YouTube, and Netflix, for example, utilize these analytics to identify playlists, also known as Daily mixes, or to create video suggestions. The demand for guidance services has recently surged, resulting in a massive influx of fresh content material onto the internet.

In order for customers to be searching for the content material they desire, capable advice offerings are extraordinarily helpful. It’s difficult for a consumer to be searching for out ideal films appropriate for his or he tastes.

Different customers like distinct films or actors. It is critical to be searching for out a manner of filtering inappropriate films and discover an organization of applicable films.

Therefore, automatic advice offerings are applied to ease this task. There are but many methods to enforce those structures and organizations need to shape positive they enforce people who excellent match their business.

2. Literature Survey

Authors	Paper Title	Methodology	Pros	Cons
Ramni Harbir Singh	Movie Recommendation System using Cosine Similarity and KNN	Content-based Filtering Recommendation and KNN	It Increased accuracy with the help of cosine similarity	It has a Problem of slow start
Ashrita Kashyap	A Movie Recommender System: MOVREC using Machine Learning Techniques	and K-means algorithm and Collaborative filtering Recommendation	It Included movies database irrespective of their language and location	It Incorporates with different machine learning and clustering algorithms
Costin-Gabriel Chiru	Movie Recommender System Using the User's Psychological Profile	Collaborative based filtering	It Uses user psychological profile	Include additional sources of information, relevant for the user's preferences and also movie features.
Nicolas Hug	Surprise: A Python library for recommender systems	Using Surprise Library	Surprise contains Model evaluation like cross-validation iterators and built-in metrics as well as tools for Model selection and automatic Hyper-parameter search	It Cannot add custom features in their model
Utkarsh Gupta	Recommender system based on hierarchical clustering algorithm chameleon	Hierarchical clustering algorithm using chameleon	Produces less error as compared to K-means based Recommender System	Scope of reducing the running time of chameleon by using any parallel framework like map reduce

3. Specifications

3.1 Relevance:

The maximum apparent operational aim of a recommender System is to suggest objects which can be applicable to the consumer at hand. Users are much more likely to eat objects they discover interesting. Although relevance is the number one operational aim of a recommender machine, it isn't enough in isolation. Therefore, we talk numerous secondary dreams below, which aren't pretty as critical as relevance however are though critical sufficient to have a sizeable impact.

3.2 Novelty:

When the recommended item is something the user has never seen before, recommender systems are quite useful. Popular movies in a user's favored genre, for example, are rarely new to them. Recommendations of popular items on a regular basis can reduce sales diversity.

3.3 Serendipity:

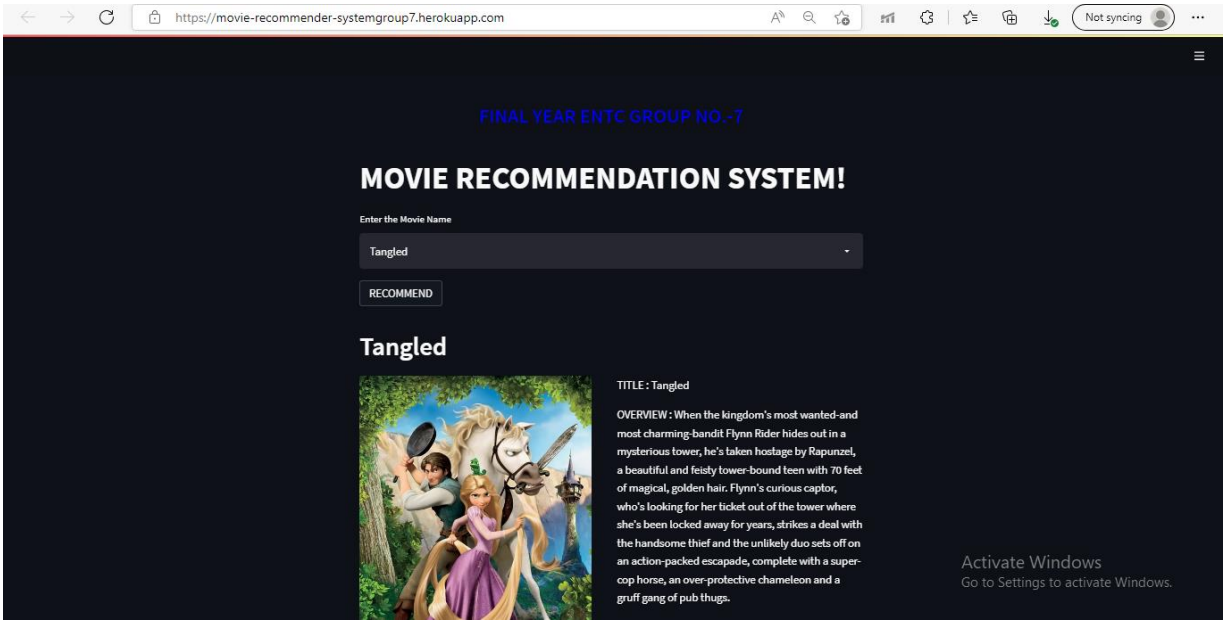
An associated belief is that of serendipity, in which the objects advocated are extremely unexpected, and consequently, there may be a modest detail of fortunate discovery, instead of apparent tips. Serendipity isn't the same as a novelty in that the tips are sincerely unexpected to the person, instead of actually something they did now no longer understand approximately before. It can also add frequently be the case that a specific person can also additionally best be eating objects of a particular type, even though a latent hobby in objects of different sorts can also additionally exist which the person may discover unexpected. Unlike novelty, serendipitous strategies cognizance of coming across such tips.

3.4 Increasing recommendation diversity:

A list of top-ok things is commonly proposed by recommender structures. When these types of endorsed objects are highly similar, the likelihood that the person will dislike all of them increases. On the other hand, because the recommended list includes a variety of objects, there is a greater chance that the person will like at least this type of item. Diversity has the advantage of ensuring that the person does not lose interest due to frequent suggestions of similar objects.

4. Methodology

a) Real-life (Pictorial) view of your System.



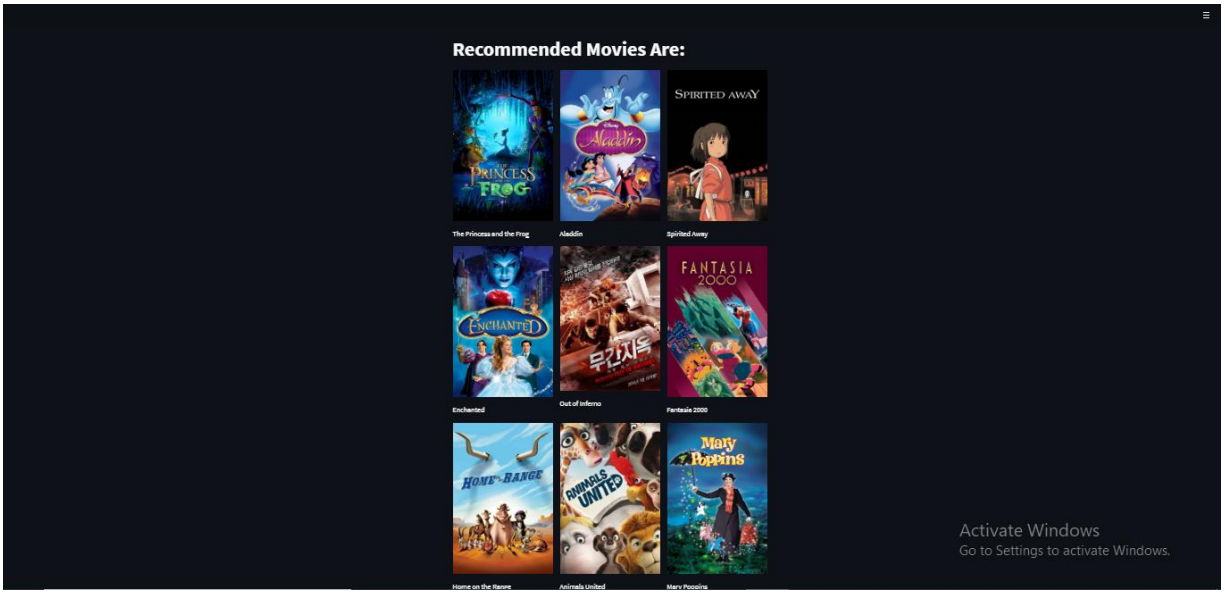


Fig 1: Pictorial Representation of Movie Recommendation System

b) Technical Block Diagram of the System you have developed.

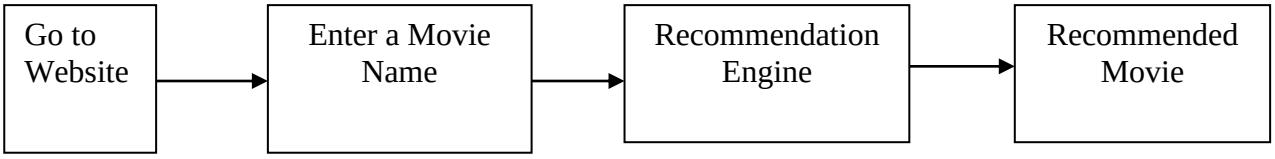


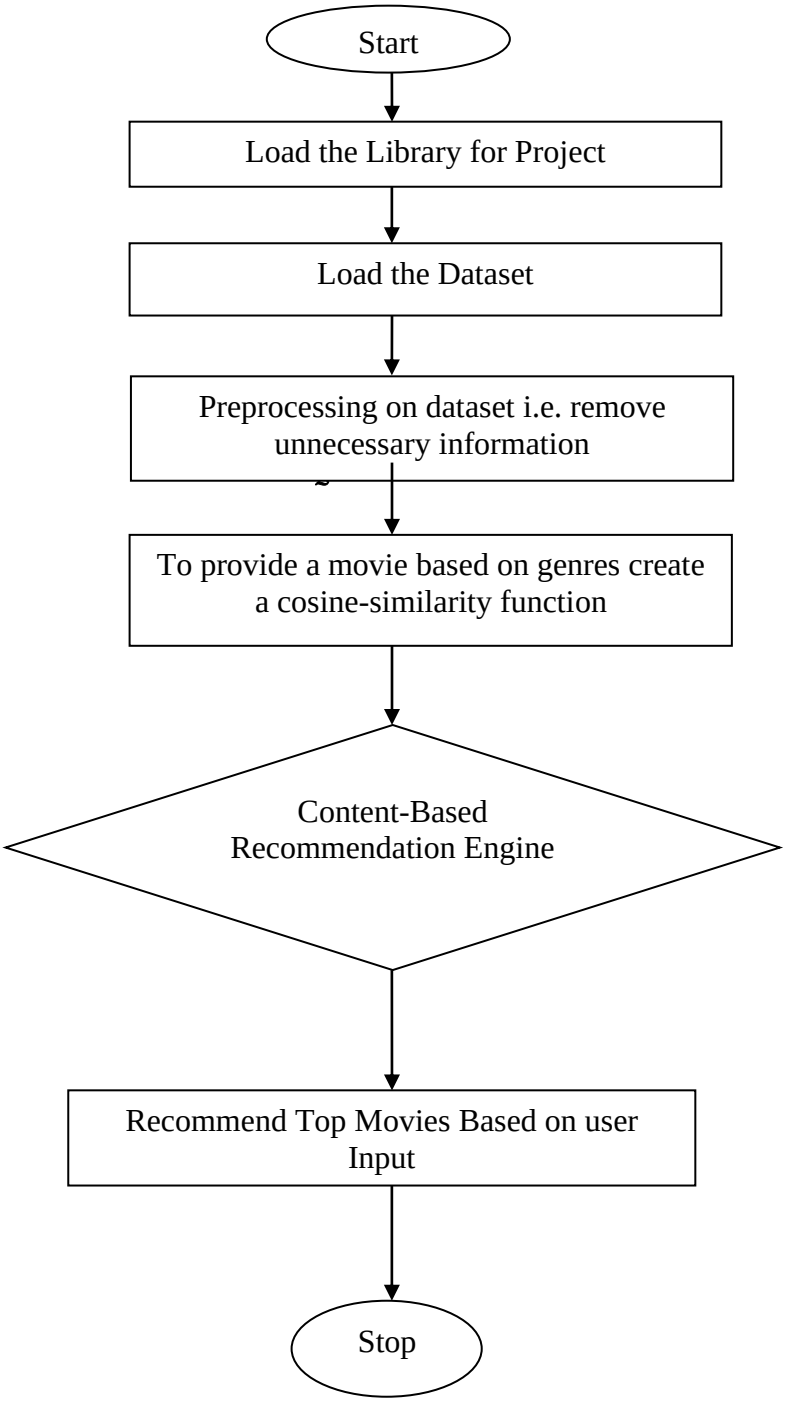
Fig 2: Block Diagram of Movie Recommendation System

1. Fig 2 shows the Block Diagram of Movie Recommendation System.
2. First go to website of Movie Recommendation System. After going to website enter the movie name which are present in Dataset.
3. In this project we are using TMDB Dataset which includes 5000 movies.
4. Enter the any movie name which are present in this Dataset.
5. In our case, we focused on Content-based Recommendation System which focus on keywords like Genres, Cast, Crew, Overview, Tags etc.
6. At last it Recommends Nine Movies based on user Input along with search Movie with its Overview.

5. Detail Design

5.1 Software design : -

Flow Chart



5.2 Methodology:

Content-based Recommender System:

Content-based totally Recommender System completely filtering technique which is one kind of recommendation system. The content material-primarily based totally attributes of the stuff you need are said as content material. Here, the system uses your abilities and likes if you want to advocate for topics that you might in all likelihood like. It uses the data furnished via you over the internet and people they'll be able to acquire and then they curate tips in step with that.

The intention inside the lower back of content material material-based totally completely filtering is to class products with precise keywords, examine what the consumer likes, look up those terms inside the database, and then recommend similar topics. This form of recommender system is exceedingly relying on the inputs furnished via users, some commonplace region examples covered Google, Wikipedia, etc.

For example, whilst a client searches for a fixed of keywords, then Google shows all the items consisting of those keywords. The under video explains how a content material material-based totally completely recommender works.

COSINE SIMILARITY:

Cosine similarity is a metric used to determine how comparable entities are irrespective of their size. Mathematically, it measures the cosine of the mindset among vectors projected in a multi-dimensional space. When we're saying vectors, they may be product descriptions, titles of articles, or actually arrays of words.

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

$$\|\vec{a}\| = \sqrt{a_1^2 + a_2^2 + a_3^2 + \cdots + a_n^2}$$

$$\|\vec{b}\| = \sqrt{b_1^2 + b_2^2 + b_3^2 + \cdots + b_n^2}$$

Mathematically, if ‘a’ and ‘b’ are two vectors, cosine equation gives the angle between the two.

For Example:

$$Consider\ following\ vectors$$

$$a:[1,1,0]$$

$$b:[1,0,1]$$

$$Norm\ of\ vector\ a\ is = \|\vec{a}\| = \sqrt{1^2+1^2+0^2} = \sqrt{2}$$

$$Norm\ of\ vector\ b\ is = \|\vec{b}\| = \sqrt{1^2+2^2+1^2} = \sqrt{2}$$

$$\vec{a}\cdot\vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + a_3 b_3 = 1\times 1 + 1\times 0 + 0\times 1 = 1 + 0 + 0 = 1$$

$$\cos\theta = \frac{1}{\sqrt{2}\times\sqrt{2}} = \frac{1}{2} = 0.5$$

$$\theta = \cos^{-1} 0.5 = 60^{\circ}$$

Streamlit Library:

Streamlit is a Python-based open-source mobile app framework. It enables us to quickly develop web apps for data science and device learning. Keras, Scikit-learn, PyTorch, NumPy, SymPy, Matplotlib, pandas and other Python libraries are well-suited.

Because it combines the back-end and front-end of the app, Streamlit makes life easier.

Streamlit is an open-source Python toolkit for building and sharing beautiful, unique web apps for machine learning and data research. You can create and deploy powerful data apps in as little as a few minutes.

7. Results (Simulation / Actual)

Step 1: Importing Important Library and Dataset for Project

importing Libraries

```
In [1]: import numpy as np
        import pandas as pd
        import warnings
        warnings.filterwarnings(action = 'ignore')
```

Load Dataset with read_csv Function

```
In [2]: movies = pd.read_csv("tmdb_5000_movies.csv")
```

```
In [3]: credits = pd.read_csv("tmdb_5000_credits.csv")
```

Step 2: Merging Both Datasets Together

Merging both dataFrames

```
In [4]: movies = movies.merge(credits , on = 'title')
```

```
In [5]: movies.head(3)
```

Out[5]:

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_comp
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	[{"name": "Ingo Film Partners
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	[{"name": "Walt D Pictures", "id": 2
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	[{"name": "Col Pictures", "i {"

3 rows × 23 columns

Step 3: Took important Column for Project

```
In [6]: movies = movies[['movie_id' , 'title', 'overview' , 'genres' , 'keywords' , 'cast' , 'crew']]
```

```
In [7]: movies.head(5)
```

Out[7]:

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

Step 4: Checking For Missing Value

checking for missing values with isnull()

```
In [8]: movies.isnull().sum()
```

```
Out[8]: movie_id    0
        title       0
        overview    3
        genres      0
        keywords    0
        cast        0
        crew        0
        dtype: int64
```

Step 5: Preprocessing On Genres

Preprocessing Genres

```
0]: import ast   # ast- It helps python applications to process trees of the python abstract syntax grammar.
    def process(genre):      # Process function for Genre
        final = []           # Creating one empty list
        for i in ast.literal_eval(genre): # literal_eval()-it helps to traverse an Abstract Syntax Tree
            final.append(i['name'])   #Append() function adds a single item to the existing list

        return final  #return all Genres
```

Step 6: Use Apply() on genres and keywords

```
In [11]: movies['genres'] = movies['genres'].apply(process)
```

```
In [12]: movies['keywords'] = movies['keywords'].apply(process)
```

```
In [13]: movies['cast'][0]
```

```
Out[13]: '[{"cast_id": 242, "character": "Jake Sully", "credit_id": "5602a8a7c3a3685532001c9a", "gender": 2, "id": 65731, "name": "Sam W
         orthington", "order": 0}, {"cast_id": 3, "character": "Neytiri", "credit_id": "52fe48009251416c750ac9cb", "gender": 1, "id": 86
         91, "name": "Zoe Saldana", "order": 1}, {"cast_id": 25, "character": "Dr. Grace Augustine", "credit_id": "52fe48009251416c750ac
         a39", "gender": 1, "id": 10205, "name": "Sigourney Weaver", "order": 2}, {"cast_id": 4, "character": "Col. Quaritch", "credit_i
         d": "52fe48009251416c750ac9cf", "gender": 2, "id": 32747, "name": "Stephen Lang", "order": 3}, {"cast_id": 5, "character": "Tru
         dy Chacon", "credit_id": "52fe48009251416c750ac9d3", "gender": 1, "id": 17647, "name": "Michelle Rodriguez", "order": 4}, {"cas
         t_id": 8, "character": "Selfridge", "credit_id": "52fe48009251416c750ac9e1", "gender": 2, "id": 1771, "name": "Giovanni Ribis
         i", "order": 5}, {"cast_id": 7, "character": "Norm Spellman", "credit_id": "52fe48009251416c750ac9dd", "gender": 2, "id": 5923
         1, "name": "Joel David Moore", "order": 6}, {"cast_id": 9, "character": "Moat", "credit_id": "52fe48009251416c750ac9e5", "gende
         r": 1, "id": 30485, "name": "CCH Pounder", "order": 7}, {"cast_id": 11, "character": "Eytukan", "credit_id": "52fe48009251416c7
         50ac9ed", "gender": 2, "id": 15853, "name": "Wes Studi", "order": 8}, {"cast_id": 10, "character": "Tsu\'Tey", "credit_id": "52
         fe48009251416c750ac9e9", "gender": 2, "id": 10964, "name": "Laz Alonso", "order": 9}, {"cast_id": 12, "character": "Dr. Max Pat
```

Step 7: Defining One Function to return the Cast of Movies.

Defining One function to return the Cast Of Movies

```
In [14]: def actors(cast):
             final = []
             count = 0
             for i in ast.literal_eval(cast):
                 if(count<5):
                     final.append(i['name'])
                     count+=1
                 else:
                     break

             return final
```

```
In [15]: movies['cast'] = movies['cast'].apply(actors)
```

```
In [16]: movies.head(3)
```

Step 8: Define Director Function which returns Director of Movie.

In [18]:
```
## Defining Director() which returns crew as a director
```

In [19]:
```
def director(crew):
    final = []
    for i in ast.literal_eval(crew):
        if i['job'] == 'Director':
            final.append(i['name'])

    return final
```

In [20]:
```
movies['crew'] = movies['crew'].apply(director)
```

In [21]:
```
movies.head()
```

Out[21]:

	movie_id	title	overview	genres	keywords	cast	crew
0	19995	Avatar	In the 22nd century, a paraplegic Marine is di...	[Action, Adventure, Fantasy, Science Fiction]	[culture clash, future, space war, space colon...	[Sam Worthington, Zoe Saldana, Sigourney Weave...	[James Cameron]
1	285	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	[Adventure, Fantasy, Action]	[ocean, drug abuse, exotic island, east india ...	[Johnny Depp, Orlando Bloom, Keira Knightley, ...	[Gore Verbinski]
2	206647	Spectre	A cryptic message from Bond's past sends him o...	[Action, Adventure, Crime]	[spy, based on novel, secret agent, sequel, mi...	[Daniel Craig, Christoph Waltz, Léa Seydoux, R...	[Sam Mendes]
3	49026	The Dark Knight Rises	Following the death of District Attorney Harve...	[Action, Crime, Drama, Thriller]	[dc comics, crime fighter, terrorist, secret i...	[Christian Bale, Michael Caine, Gary Oldman, A...	[Christopher Nolan]
4	49529	John Carter	John Carter is a war-weary, former military ca...	[Action, Adventure, Science Fiction]	[based on novel, mars, medallion, space travel...	[Taylor Kitsch, Lynn Collins, Samantha Morton,...	[Andrew Stanton]

Step 9: Covert String into List using Lambda function.

overview is in string format so we will be converting it into list

In [22]:
```
movies['overview'] = movies['overview'].apply(lambda x:x.split())
```

Step 10: Proprocessing on Genres, Keywords, Cast and Crew.

processing genres

In [23]:
```
movies['genres'] = movies['genres'].apply(lambda x:[i.replace(" " ,"") for i in x])
```

processing keywords

In [24]:
```
movies['keywords'] = movies['keywords'].apply(lambda x:[i.replace(" " ,"") for i in x])
```

processing cast

In [25]:
```
movies['cast'] = movies['cast'].apply(lambda x:[i.replace(" " ,"") for i in x])
```

processing crew

In [26]:
```
movies['crew'] = movies['crew'].apply(lambda x:[i.replace(" " ,"") for i in x])
```

Step 11:Combine different columns together

Now we have to concatenate all the columns to make a new columns which will have all the tags of a movie

```
In [27]: movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] + movies['cast'] + movies['crew']
```

len() returns the number of items in an object

```
In [28]: movies.head()
         len(movies)
Out[28]: 4806
```

Step 12: Create New Dataframe with the following Table.

Created one new Dataframe with following Columns

```
[31]: df = movies[['movie_id' , 'title' , 'tags']]
```

```
[32]: df
```

Out[32]:

	movie_id	title	tags
0	19995	Avatar	[In, the, 22nd, century,, a, paraplegic, Marin...
1	285	Pirates of the Caribbean: At World's End	[Captain, Barbossa,, long, believed, to, be, d...
2	206647	Spectre	[A, cryptic, message, from, Bond's, past, send...
3	49026	The Dark Knight Rises	[Following, the, death, of, District, Attorney...
4	49529	John Carter	[John, Carter, is, a, war-weary,, former, mili...
...	...	...	...
4804	9367	El Mariachi	[El, Mariachi, just, wants, to, play, his, gui...
4805	72766	Newlyweds	[A, newlywed, couple's, honeymoon, is, upended...
4806	231617	Signed, Sealed, Delivered	["Signed,, Sealed,, Delivered", introduces, a,...
4807	126186	Shanghai Calling	[When, ambitious, New, York, attorney, Sam, is...
4808	25975	My Date with Drew	[Ever, since, the, second, grade, when, he, fi...

4806 rows × 3 columns

Step 13: Convert Tags into String

#Converting Tags column into a string

```
In [33]: df['tags'] = df['tags'].apply(lambda x:" ".join(x))
```

printing first tag

```
In [34]:
         df['tags'][0]
Out[34]: 'In the 22nd century, a paraplegic Marine is dispatched to the moon Pandora on a unique mission, but becomes torn between follo
         wing orders and protecting an alien civilization. Action Adventure Fantasy ScienceFiction cultureclash future spacewar spacecol
         ony society spacetravel futuristic romance space alien tribe alienplanet cgi marine soldier battle loveaffair antiwar powerrela
         tions mindandsoul 3d SamWorthington ZoeSaldana SigourneyWeaver StephenLang MichelleRodriguez JamesCameron'
```

Step 14: Converting Strings into Lowercase

converting string of tags columns into lower case

In [35]:
```
df['tags'] = df['tags'].apply(lambda x:x.lower())
```

printing first tag

In [36]:
```
df['tags'][0]
```

Out[36]:
```
'in the 22nd century, a paraplegic marine is dispatched to the moon pandora on a unique mission, but becomes torn between follo
wing orders and protecting an alien civilization. action adventure fantasy sciencefiction cultureclash future spacewar spacecol
ony society spacetravel futuristic romance space alien tribe alienplanet cgi marine soldier battle loveaffair antiwar powerrela
tions mindandsoul 3d samworthington zoesaldana sigourneyweaver stephenlang michellerodriguez jamescameron'
```

In [37]:
```
len(df)
```

Out[37]:
```
4806
```

Step 15: Install nltk

nltk -Natural Language Toolkit- It uses to work with human language data.

39]:
```
pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\hp\appdata\local\programs\python\python39\lib\site-packages (3.7)
Requirement already satisfied: regex>=2021.8.3 in c:\users\hp\appdata\local\programs\python\python39\lib\site-packages (from nl
tk) (2022.4.24)
Requirement already satisfied: tqdm in c:\users\hp\appdata\local\programs\python\python39\lib\site-packages (from nltk) (4.64.
0)
Requirement already satisfied: click in c:\users\hp\appdata\local\programs\python\python39\lib\site-packages (from nltk) (8.1.
3)
Requirement already satisfied: joblib in c:\users\hp\appdata\local\programs\python\python39\lib\site-packages (from nltk) (1.1.
0)
Requirement already satisfied: colorama in c:\users\hp\appdata\local\programs\python\python39\lib\site-packages (from click->nl
tk) (0.4.4)
Note: you may need to restart the kernel to use updated packages.

WARNING: You are using pip version 21.2.4; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\Hp\AppData\Local\Programs\Python\Python39\python.exe -m pip install --upgrade p
ip' command.
```

Step 16: Define one function to find stopword

In [40]:
```
import re  # re-Regular Expression is a sequence of character that forms a search pattern.
import nltk
nltk.download('stopwords')# download function used to download several packages and STOPWORD- stopwords are words that you don't
from nltk.corpus import stopwords # Corpus is a collectin of Authentic text organized into dataset. Authentic means text written
from nltk.stem.porter import PorterStemmer # Stemmers remove morphological affixes from words leaving only the word stem.
corpus = []
for i in df['tags']:
  #removing all Punctuations.
  tag = re.sub('[^a-zA-Z]', ' ', i)

  #Splitting tags into different Words.
  tag = tag.split()

  #Stemming.
  ps = PorterStemmer()
  all_stopwords = stopwords.words('english')
  all_stopwords.remove('not')
  tag = [ps.stem(word) for word in tag if not word in set(all_stopwords)]
  #Now since we have applied stemming to all the words we again join them back
  tag   = ' '.join(tag)
  corpus.append(tag)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Hp\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Step 17: Import CountVectorizer and to create a Vector

```
In [42]: from sklearn.feature_extraction.text import CountVectorizer
         #limiting the number of different words to 5000
         cv = CountVectorizer(max_features = 5000)
         vectors = cv.fit_transform(corpus).toarray()

In [43]: # sparse Matrix
         vectors

Out[43]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=int64)

In [44]: #list of 5000 most common words in our corpus
         cv.get_feature_names()

Out[44]: ['aaron',
          'aaroneckhart',
          'aarontaylor',
          'abandon',
          'abbi',
          'abduct',
          'abern',
          'abigailbreslin',
          'abil',
          'abl',
          'aboard',
          'aborigin',
          'absenc',
```

Step 18: Find the Cosine Similarity of Movie

```
In [45]: from sklearn.metrics.pairwise import cosine_similarity

In [46]: similarity_matrix = cosine_similarity(vectors)

In [47]: similarity_matrix

Out[47]: array([[1.        , 0.07792865, 0.07897472, ..., 0.06254073, 0.0238705 ,
                 0.        ],
                [0.07792865, 1.        , 0.08000711, ..., 0.02111943, 0.        ,
                 0.        ],
                [0.07897472, 0.08000711, 1.        , ..., 0.04280585, 0.        ,
                 0.        ],
                ...,
                [0.06254073, 0.02111943, 0.04280585, ..., 1.        , 0.05822225,
                 0.03881483],
                [0.0238705 , 0.        , 0.        , ..., 0.05822225, 1.        ,
                 0.06666667],
                [0.        , 0.        , 0.        , ..., 0.03881483, 0.06666667,
                 1.        ]])
```

Step 19: Recommendation function to recommend Movie

```
In [48]: # now we will recommend 9 movies which has similarity score near to 1
         def recommendation_system(movie):
             try:
                 movie_index = df[df['title'] == movie].index[0]
                 cosine_angles = similarity_matrix[movie_index]
                 recommended_movies = sorted(list(enumerate(cosine_angles)) , reverse = True , key = lambda x:x[1])[1:10]
                 print("Top 9 recommended movies based on your choice:")
                 print()
                 k = 1
                 for i in recommended_movies:
                     print(k , "->", df.iloc[i[0]].title) # it enables us to select a particulat cell of the dataset.
                     k+=1
             except:
                 print("This movie is not in our DataBase")
```

Step 20: Result of Recommendation on Jupyter Notebook

```
In [49]: recommendation_system('Spectre')

Top 9 recommended movies based on your choice:

1 -> Quantum of Solace
2 -> Skyfall
3 -> Never Say Never Again
4 -> From Russia with Love
5 -> Thunderball
6 -> Octopussy
7 -> Safe Haven
8 -> Diamonds Are Forever
9 -> Licence to Kill
```

Step 21: Conver Our Dataframe into Dictionary

to_dict() is used to convert a dataframe into a dictionary of series.

```
In [50]: df.to_dict()

Out[50]: {'movie_id': {0: 19995,
                       1: 285,
                       2: 206647,
                       3: 49026,
                       4: 49529,
                       5: 559,
                       6: 38757,
                       7: 99861,
                       8: 767,
                       9: 209112,
                       10: 1452,
                       11: 10764,
                       12: 58,
                       13: 57201,
                       14: 49521,
                       15: 2454,
                       16: 24428,
                       17: 1865,
                       18: 41154,
```

Step 22: Creating a Pickle file of the movie and similarity matrix

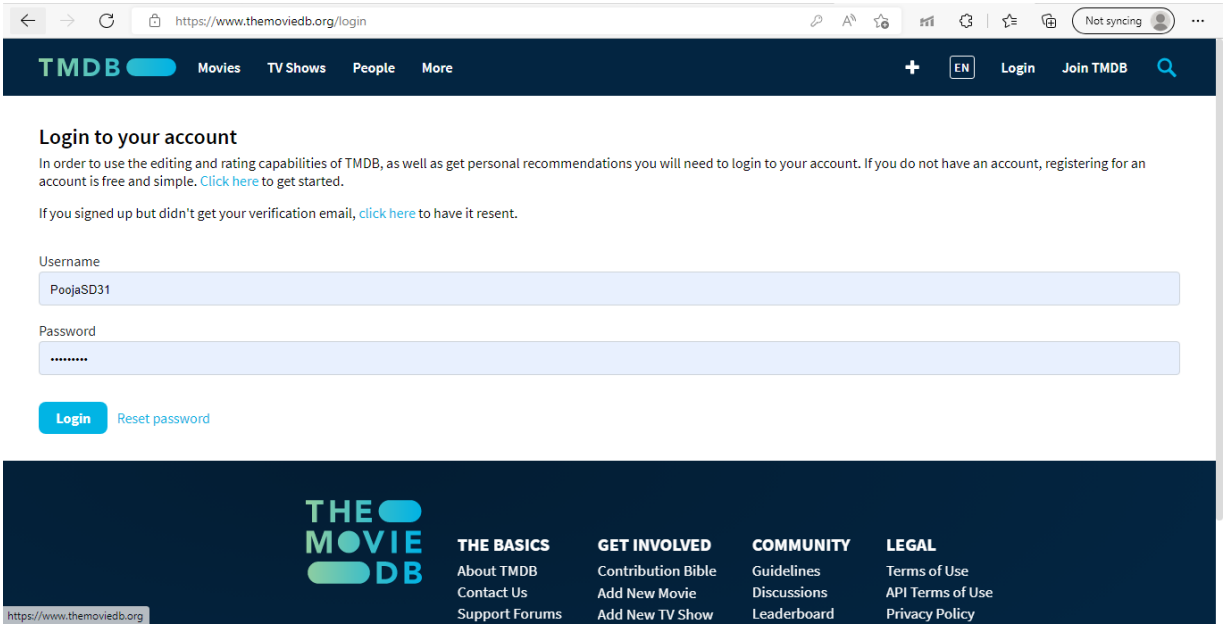
PICKLE library is used to converts any kind of python objects like list,dict etc into byte streams 0s and 1s is called Pickling.

```
In [51]: import pickle
```

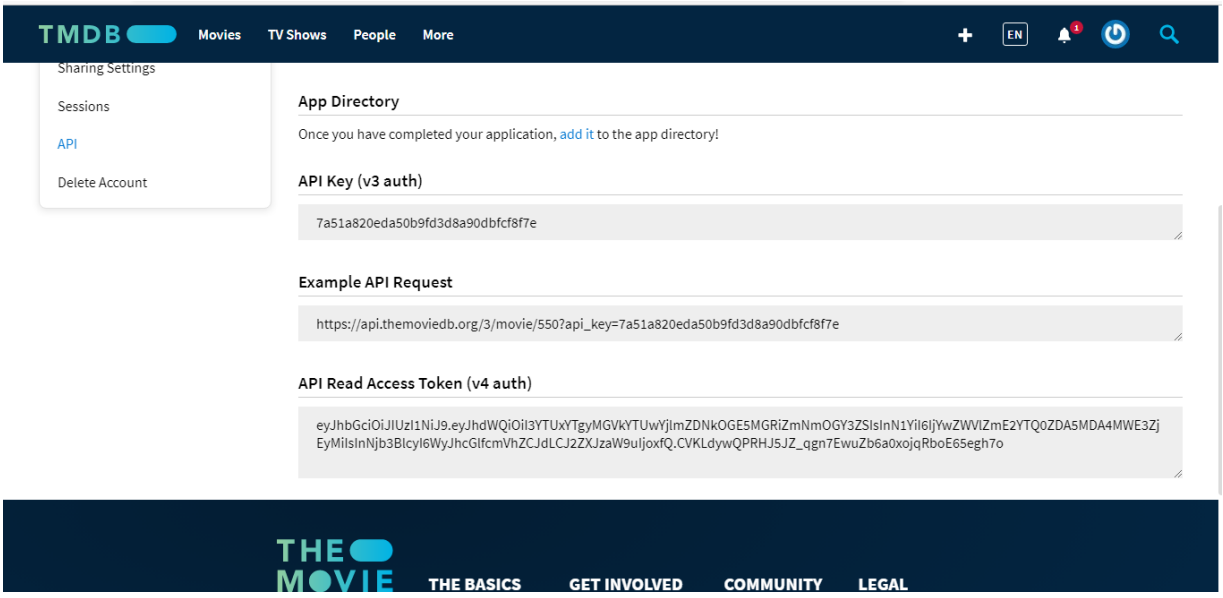
dump() function to store the object data to the file.

```
In [52]: pickle.dump(df.to_dict() , open('movie.pkl', 'wb'))

In [53]: pickle.dump(similarity_matrix , open('similaritymat.pkl' , 'wb'))
```

Step 23: Login to TMDB Account



Step 24: Go to Settings to take Unique API to fetch the poster of the project



Step 25: Install the important Library on Pycharm

```
# Importing the Library

import streamlit as st
import pickle
import pandas as pd
import requests
```

Step 26: Defined one crew function with API Key

```
def crew(movie_id):
    response = requests.get(
        "https://api.themoviedb.org/3/movie/{0}/credits?api_key=7a51a820eda50b9fd3d8a90dbfcf8f7e&language=en-US".format(
            movie_id))
    data = response.json()
    crew_names = []
    final_casts = []
    n = 0
    for i in data["cast"]:
        if(n!=6):
            crew_names.append(i['name'])
            final_casts.append("https://image.tmdb.org/t/p/w500/" + i['profile_path'])
            n+=1
        else:
            break
    return crew_names, final_casts

#Function which return release date of search movie
```

Step 27: Defined date, genres and Overview Function with API Key

movieapp.py | requirements.txt

```
def date(movie_id):
    response = requests.get(
        "https://api.themoviedb.org/3/movie/{}?api_key=7a51a820eda50b9fd3d8a90dbfcf8f7e&language=en-US".format(
            movie_id))
    data = response.json()
    return data['release_date']

#Function which return all genres of search movie
def genres(movie_id):
    response = requests.get(
        "https://api.themoviedb.org/3/movie/{}?api_key=7a51a820eda50b9fd3d8a90dbfcf8f7e&language=en-US".format(
            movie_id))
    data = response.json()
    return data['genres']

#Function which return Overview of Search Movie
def overview(movie_id):
    response = requests.get(
        "https://api.themoviedb.org/3/movie/{}?api_key=7a51a820eda50b9fd3d8a90dbfcf8f7e&language=en-US".format(
            movie_id))
    data = response.json()
    return data['overview']
```

Step 28: Defined Poster function to display poster of movie.

```
#Function which returns Poster of search movie
def poster(movie_id):
    response = requests.get("https://api.themoviedb.org/3/movie/{}?api_key=7a51a820eda50b9fd3d8a90dbfcf8f7e&language=en-US".f
    data = response.json()
    return "https://image.tmdb.org/t/p/w500/" + data['poster_path']

```

Step 29: Defined Recommendation function to recommend 9 Movies.

```
def recommendation(movie):
    movie_index = movies[movies['title'] == movie].index[0]
    cosine_angles = similarity[movie_index]
    recommended_movies = sorted(list(enumerate(cosine_angles)), reverse=True, key=lambda x: x[1])[0:10]

    finals = []
    finals_posters = []
    final_names, final_cast = crew(movies.iloc[movies[movies['title'] == movie].index[0]].movie_id)
    genre = genres(movies.iloc[movies[movies['title'] == movie].index[0]].movie_id)
    overviews_final = overview(movies.iloc[movies[movies['title'] == movie].index[0]].movie_id)
    release_date = date(movies.iloc[movies[movies['title'] == movie].index[0]].movie_id)
    for i in recommended_movies:
        finals.append(movies.iloc[i[0]].title)
        finals_posters.append(poster(movies.iloc[i[0]].movie_id))
    return final_names, final_cast, release_date, genre, overviews_final, finals, finals_posters

```

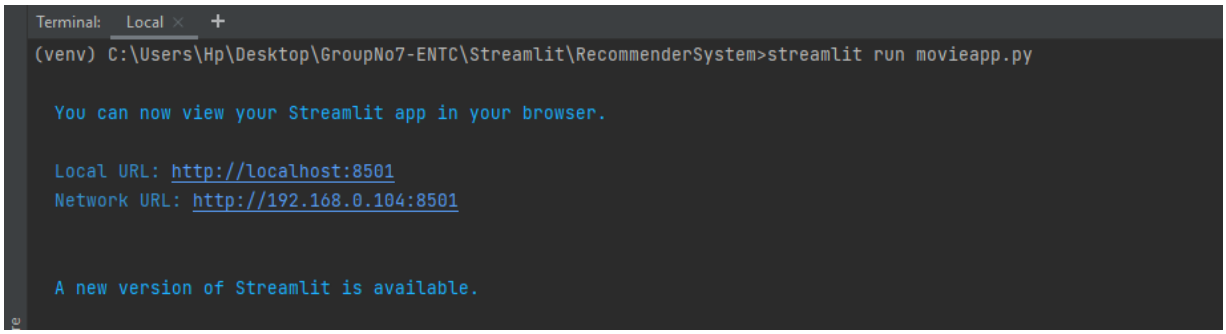
Step 30: Used the Pickle file in our Project

```
# movie.pkl file is loaded into moviedict
moviesdict = pickle.load(open('movie.pkl', 'rb'))
movies = pd.DataFrame(moviesdict)
# similaritymat.pkl is loaded into similarity
similarity = pickle.load(open('similaritymat.pkl', 'rb'))
st.title('MOVIE RECOMMENDATION SYSTEM!')


# selectbox to enter a movie name
selected_movie = st.selectbox(
    'Enter the Movie Name',
     movies['title'].values)

```

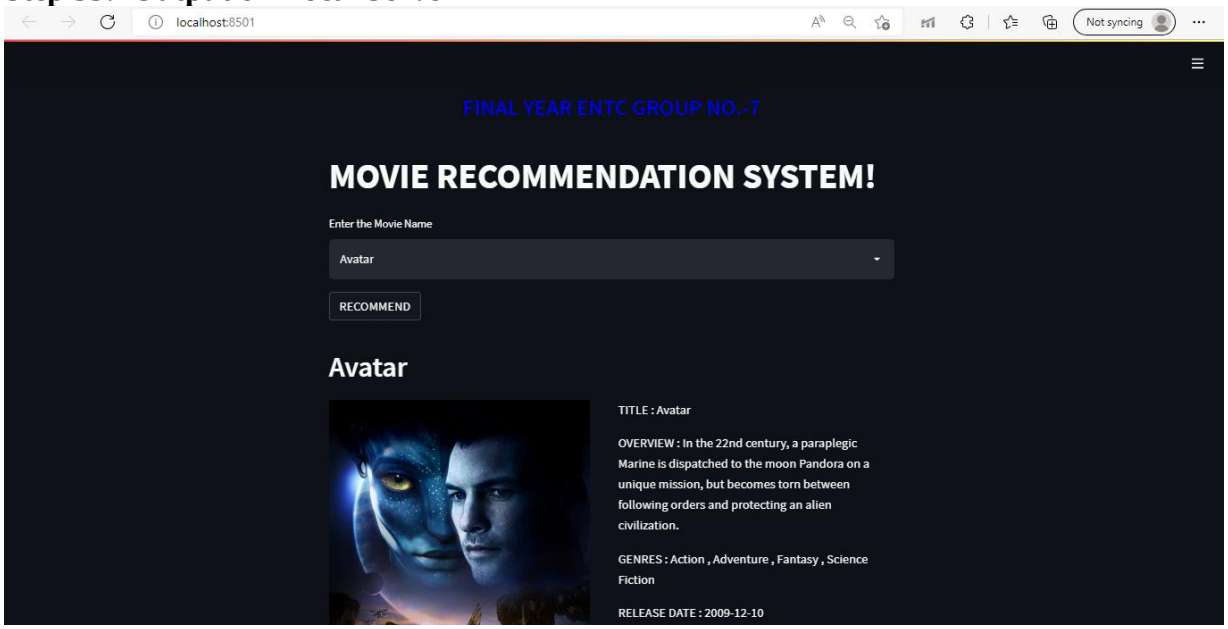
Step 31: To displayed the Movie on WebPage.

```
c1, c2, c3 = st.columns(3)
with c1:
    st.image(cast[0], width=225, use_column_width=225)
    st.caption(name[0])
with c2:
    st.image(cast[1], width=225, use_column_width=225)
    st.caption(name[1])
with c3:
    st.image(cast[2], width=225, use_column_width=225)
    st.caption(name[2])

c1, c2, c3 = st.columns(3)
with c1:
    st.image(cast[3], width=225, use_column_width=225)
    st.caption(name[3])

with c2:
    st.image(cast[4], width=225, use_column_width=225)
    st.caption(name[4])

with c3:
    st.image(cast[5], width=225, use_column_width=225)
    st.caption(name[5])
if st.button('RECOMMEND')
```

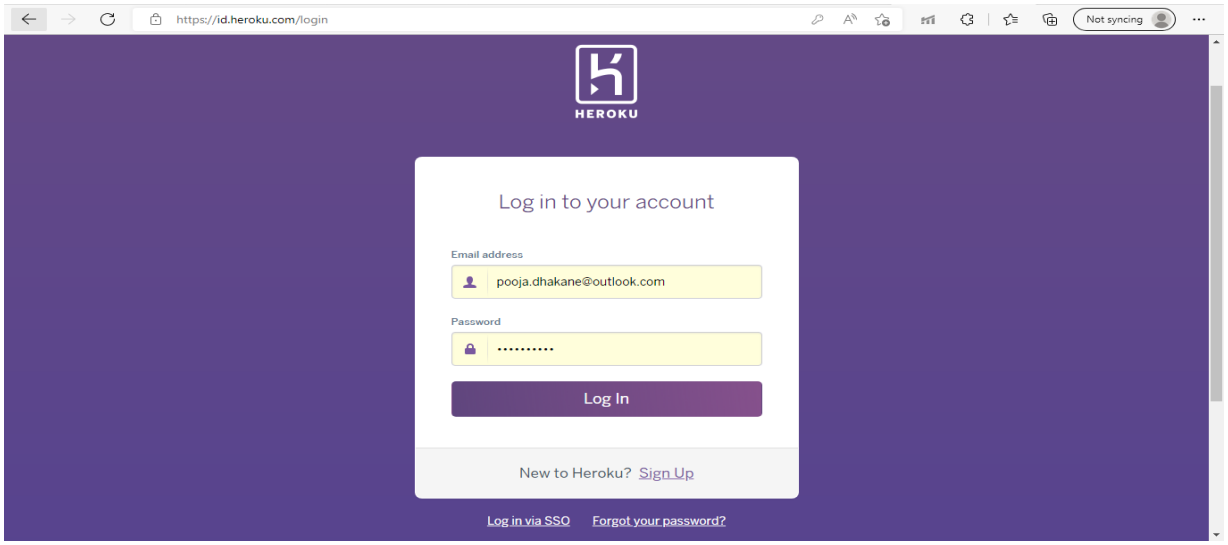
Step 32: Command to run Project on Local Server



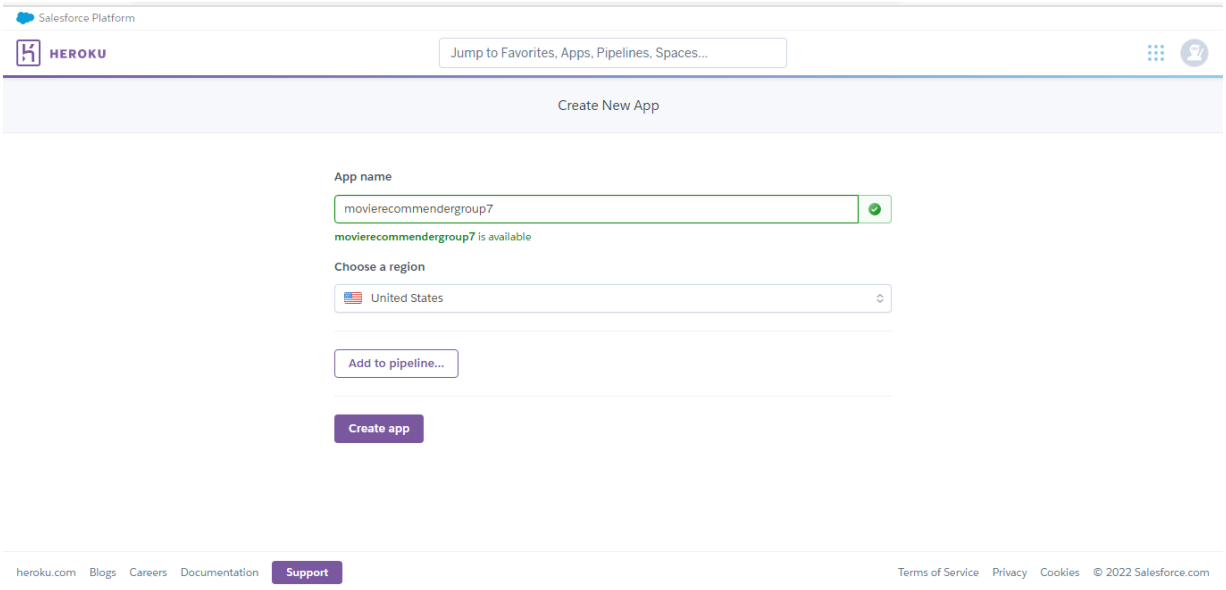
Step 33: Output on Local Server



Step 34: Log in to Heroku Account for Deployment



Step 35: Create new App with Heroku App



Step 36: git:clone –a movie-recommender-systemgroup7

```
(venv) C:\Users\Hp\Desktop\GroupNo7-ENTC\Streamlit\RecommenderSystem>heroku git:clone -a movie-recommender-systemgroup7
Cloning into 'movie-recommender-systemgroup7'...
remote: Counting objects: 22, done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 22 (delta 4), reused 0 (delta 0)
Unpacking objects: 100% (22/22), 44.75 MiB | 921.00 KiB/s, done.
```

Step 37: git commit –am “Uploading”

```
(venv) C:\Users\Hp\Desktop\GroupNo7-ENTC\Streamlit\RecommenderSystem>git commit -am "uploading"
[master 40ae7ab] uploading
 2 files changed, 3 insertions(+), 2 deletions(-)
```

Step 38: git push heroku master

```
(venv) C:\Users\Hp\Desktop\GroupNo7-ENTC\Streamlit\RecommenderSystem>git push heroku master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 470 bytes | 94.00 KiB/s, done.
Total 5 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Building on the Heroku-20 stack
remote: -----> Using buildpack: heroku/python
remote: -----> Python app detected
remote: -----> No Python version was specified. Using the same version as the last build: python-3.10.4
remote:        To use a different version, see: https://devcenter.heroku.com/articles/python-runtimes
remote: -----> No change in requirements detected, installing from cache
remote: -----> Using cached install of python-3.10.4
remote: -----> Installing pip 22.0.4, setuptools 60.10.0 and wheel 0.37.1
remote: -----> Installing SQLite3
remote: -----> Installing requirements with pip
remote: -----> Discovering process types
remote:        Procfile declares types -> web
```

```
remote:
remote: -----> Compressing...
remote:        Done: 215.1M
remote: -----> Launching...
remote:        Released v4
remote:        https://movie-recommender-systemgroup7.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/movie-recommender-systemgroup7.git
   ccecfc9..40ae7ab  master -> master
```

Step 39: URL to run on Remote Server

https://movie-recommender-systemgroup7.herokuapp.com/

7. Conclusion

a) Comments about the results obtained:

We used Jupyter Notebook and Pycharm to create a Movie Recommendation System. For our project, we used Jupyter Notebook for Python Programming and the TMDB 5000 Movies and TMDB 5000 Credits Dataset. We used the pickle file from a Python project in our deployment procedure. We utilised the Python language and the streamlit library for deployment. We used TMDB to get the Poster of that image.

b) Features:

Drive Traffic:

Through personalized e-mail messages and centered blasts, an advice engine can inspire accelerated quantities of visitors to your site, accordingly growing the possibility to scoop up extra information to in addition increase a patron profile.

Deliver Relevant Content:

A recommendation engine can give relevant product recommendations as the consumer shops based on the aforementioned profile by reading the customer's current internet web page usage and previous browsing history. The data is collected in real time so that the software application can respond to changes in shopping behaviour at the Fly.

Reduce Workload and Overhead:

The number of records required to create a personal buying enjoyment for each consumer is normally an extended manner too massive to be managed manually. Using an engine automates this process, easing the workload on your IT staff.

Engage Customers:

Consumers become being greater engaged inside the internet site while individualized object guidelines are made. They are capable of diving even greater deeply into the product line with no need to perform seek after seek.

Offer Recommendations and Direction:

A skilled provider can offer hints on methods to make use of the records amassed and pronounced to the customer. Acting as an accomplice and a consultant, the dealer desires to have the knowledge to help direct the e-trade web website online to a rich future.

c) Limitations:

Cold-Start Problem:

It needs enough customers within the tool to find out a suit. For instance, if we want to find out a similar patron or similar item, we suit them with the set of available clients or items. At the initial stage for today's patron, his profile is empty as he has now not rated any item and the tool does now not understand his taste, so it turns into difficult for a tool to provide him with recommendations about any item.

Data Sparsity:
The person or score matrix may be very sparse. It may be very tough to locate customers who have rated the equal objects due to the fact maximum of the person does now no longer fee the objects. So it turns into tough to locate a hard and fast of customers who fee the objects. To provide a piece of advice is genuinely hard whilst there may be much fewer facts approximately any person.

Scalability:
Collaborative Filtering uses a massive amount of data to make reliability better which requires more resources. As information grows exponentially processing becomes expensive and inaccurate results from this big data challenge.

d) Future scope:

A lot of research is going immediately to similarly enhance the output accuracy and upgrades in all dimensions of the recommender system. The search is centered on several areas to make the Recommender System more and more usable and practical in real-existence scenarios. Cosine similarity calculations do now not artwork properly at the same time as we do now no longer have enough rankings for movies or at the same time as a user's rating for some movie is as a substitute each immoderate or low. As an improvement on this project, some unique strategies which encompass adjusted cosine similarity can be used to compute similarity.

8. References

8.1 Charu C. Aggarwal's A Recommender System Textbook
8.2 "Movie recommendation system employing cosine similarity and knn," R. Singh, S. Maurya, T. Tripathi, T. Narula, and G. Srivastav.
8.3 "Article: A movie recommender system: Movrec," International Journal of Computer Applications, M. Kumar, D. Yadav, A. Singh, and V. K. Gupta.
8.4 N. Hug, Journal of Open Source Software, "Surprise: A Python Library for Recommender Systems."
8.5 L. M. de Campos, J. M. Fernandez-Luna, J. F. Huete, and M. A. Rueda-Morales, "Combining content-based and collaborative recommendations: A hybrid approach based on Bayesian networks," International Journal of Approximate Reasoning.
8.6 https://www.slideshare.net/CrossingMinds/recommendation-system-explained
8.7 https://www.mygreatlearning.com/blog/masterclass-on-movie-recommendation-system/