**ECM3001 – DATA ANALYTICS AND VISUALIZATION**

**J Component Report**

**A project report titled**

**Movie Recommendation System**

*By*

Reg. No: 19blc1049 Name: Jerin Mathew

Reg. No: 19blc1003 Name:Vaibhav Manglani

Reg. No: 19blc1146 Name: Kaushal Goyal

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGENEERING

*Submitted to*

**Dr.R.Rajalakshmi**

**School of Computer Science and Engineering**

****

*November 2021*

*­­­*

**DECLARATION BY THE CANDIDATE**

I hereby declare that the report titled “**Movie Recommendation System”** submitted by me to VIT Chennai is a record of bona-fide work undertaken by me under the supervision of **Dr. R. Rajalakshmi, Associate Professor, SCOPE, Vellore Institute of Technology, Chennai.**

Signature of the Candidate

**ACKNOWLEDGEMENT**

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr.R. Rajalakshmi,** School of Computer Science and Engineeringfor her consistent encouragement and valuable guidance offered to us throughout the course of the project work.

We are extremely grateful to **Dr.R. Ganesan, Dean,** School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai, for extending the facilities of the School towards our project and for his unstinting support.

We express our thanks to our **Head of the Department** for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the courses.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

**BONAFIDE CERTIFICATE**

Certified that this project report entitled “**Movie Recommendation System”** is a bona-fide work of **Jerin Mathew (19BLC1049), Vaibhav Manglani (19BLC1003) , Kaushal Goyal (19BLC1146)** carried out the “J”-Project work under my supervision and guidance for **ECM3001 – DATA ANALYTICS AND VISUALIZATION**

.

**Dr.R. Rajalakshmi**

SCOPE

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Ch. No** | **Chapter** | **Page Number** |
| 1 | Introduction | 7 |
| 2 | Literature Survey / Requirements | 8 |
| 3 | Proposed System / Module(s) description | 10 |
| 4 | Results and Discussion | 12 |
| 5 | Conclusion | 19 |
| 6 | Reference | 20 |

**ABSTRACT**

Everyone loves movies irrespective of age, gender, race, colour, or geographical location. We all in a way are connected to each other via this amazing medium. Yet what most interesting is the fact that how unique our choices and combinations are in terms of movie preferences. Some people like genre-specific movies be it a thriller, romance, or sci-fi, while others focus on lead actors and directors. When we take all that into account, it’s astoundingly difficult to generalize a movie and say that everyone would like it.

So we plan on creating a movie recommendation system where the movies will be recommended with the help of behavioural patterns of not only the audience but also the movies themselves.

**INTRODUCTION**

Recommender systems are the systems that are designed to recommend things to the user based on many different factors. These systems predict the most likely product that the users are most likely to purchase and are of interest to. Companies like Netflix, Amazon, etc. use recommender systems to help their users to identify the correct product or movies for them.

We have created a movie recommender system where the movies are recommended to the user based on:

1. TOP RATED MOVIES: Where the user is presented with a generalized list of the movies which have been highly rated by the reviewers.
2. CONTENT BASED FILTERING: Where the user is presented with similar types of movies based on a selected movie.

**………………**

**Literature Survey / Requirements**

# Movie Recommender Systems

# By ROUNAK BANIK

# Few recommendation algorithms are used to try and build an ensemble of these models to come up with a recommendation system.

# Simple recommender: The Simple Recommender offers generalized recommendations to every user based on movie popularity. The basic idea behind this recommender is that movies that are more popular and more critically acclaimed will have a higher probability of being liked by the average audience. This model does not give personalized recommendations based on the user.

# The implementation of this model is extremely trivial. All we have to do is sort our movies based on ratings and popularity and display the top movies of our list. As an added step, we can pass in a genre argument to get the top movies of a particular genre.

# Getting Started with a Movie Recommendation System

# By IBTESAM AHMED

**Content Based Filtering**

In this recommender system the content of the movie (overview, cast, crew, keyword, tagline etc) is used to find its similarity with other movies. Then the movies that are most likely to be similar are recommended.

**Plot description based Recommender**

We will compute pair wise similarity scores for all movies based on their plot descriptions and recommend movies based on that similarity score. The plot description is given in the overview feature of our dataset.

We need to convert the word vector of each overview. Now we'll compute Term Frequency-Inverse Document Frequency (TF-IDF) vectors for each overview.

Now if you are wondering what is term frequency, it is the relative frequency of a word in a document and is given as (term instances/total instances). Inverse Document Frequency is the relative count of documents containing the term is given as log(number of documents/documents with term) The overall importance of each word to the documents in which they appear is equal to TF \* IDF

This will give you a matrix where each column represents a word in the overview vocabulary (all the words that appear in at least one document) and each row represents a movie, as before. This is done to reduce the importance of words that occur frequently in plot overviews and therefore, their significance in computing the final similarity score.

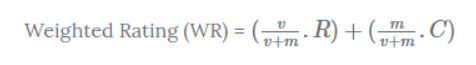
**Proposed System / Module(s) description**

TOP RATED MOVIES:

* We need a metric to score or rate movie
* Calculate the score for every movie
* Sort the scores and recommend the best rated movie to the users.

We use weighted average to calculate the score based on the ratings which is later used to sort the movies.

Weighted average is a calculation that takes into account the varying degrees of importance of the numbers in a data set. In calculating a weighted average, each number in the data set is multiplied by a predetermined weight before the final calculation is made.

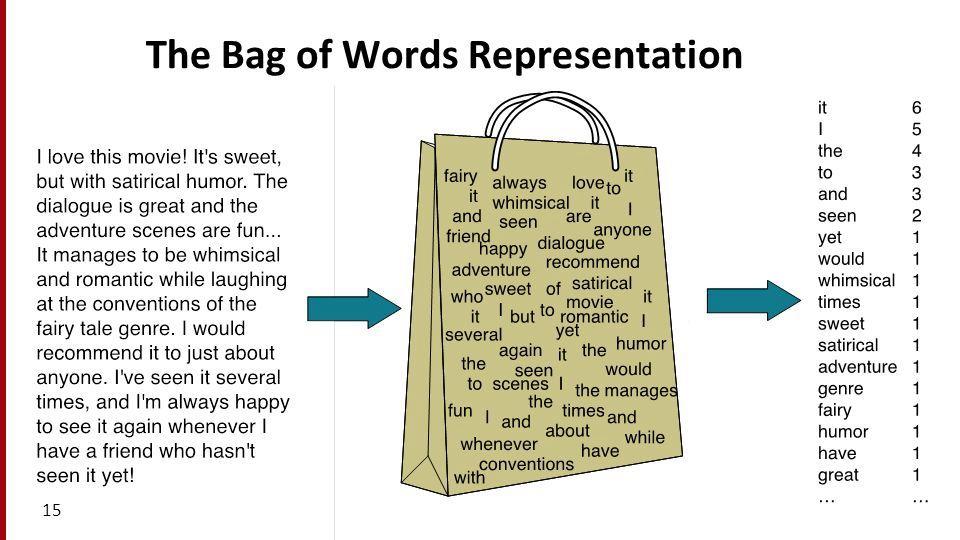


Where,

* v is the number of votes for the movie;
* m is the minimum votes required to be listed in the chart;
* R is the average rating of the movie; And
* C is the mean vote across the whole report

CONTENT BASED FILTERING:

* The user must select a movie and related movies will be recommended to the user. Tags are used to perform content based filtering.
* In information systems, a tag is a keyword or term assigned to a piece of information (such as an Internet bookmark, digital image, database record, or computer file). This kind of metadata helps describe an item and allows it to be found again by browsing or searching. Tags are generally chosen informally and personally by the item's creator or by its viewer, depending on the system
* Once tags are created for each movie, bag-of-words model is used to create a vector out of these tags for each movie.
* The Bag of Words (Bow) model is the simplest form of text representation in numbers. Like the term itself, we can represent a sentence as a bag of words vector (a string of numbers).
* The bag-of-words model is very simple to understand and implement and offers a lot of flexibility for customization on your specific text data.
* It has been used with great success on prediction problems like language modelling and documentation classification.



**Results and Discussion**

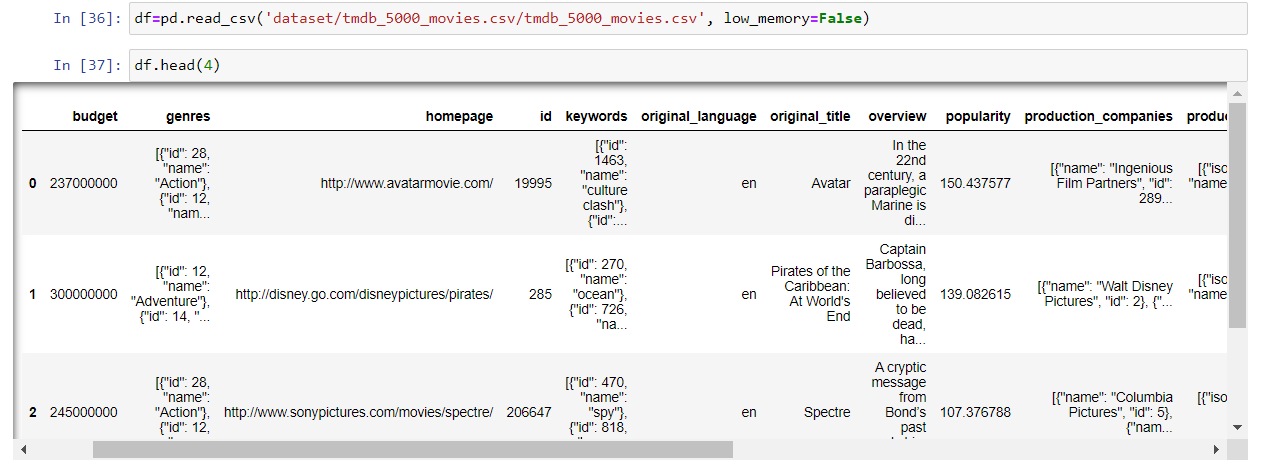
**PROJECT FLOW FOR TOP RATED MOVIES:**

* Pre-processing of the dataset

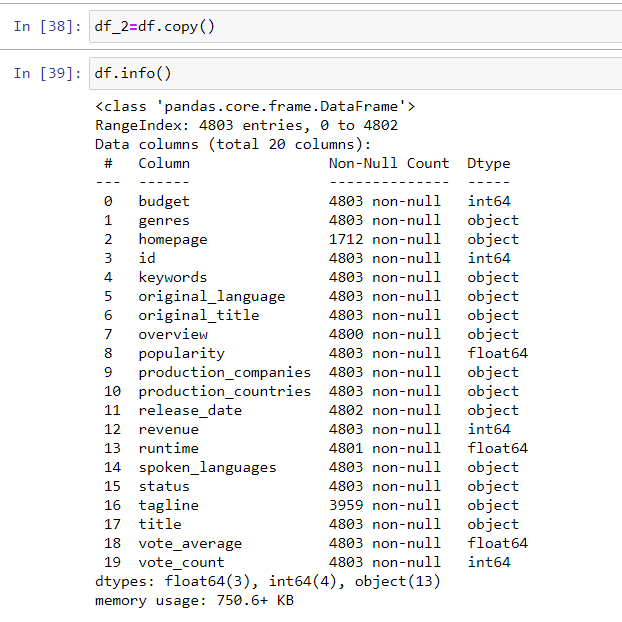
Dataset used: TMDB 5000 Movie Dataset

<https://www.kaggle.com/tmdb/tmdb-movie-metadata?select=tmdb_5000_movies.csv>

Initial dataset

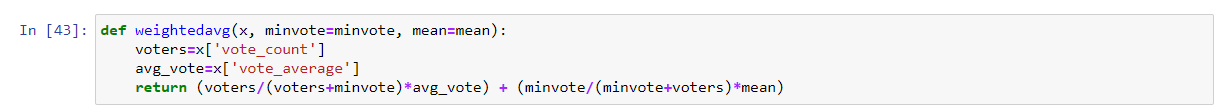


Data available in the dataset.



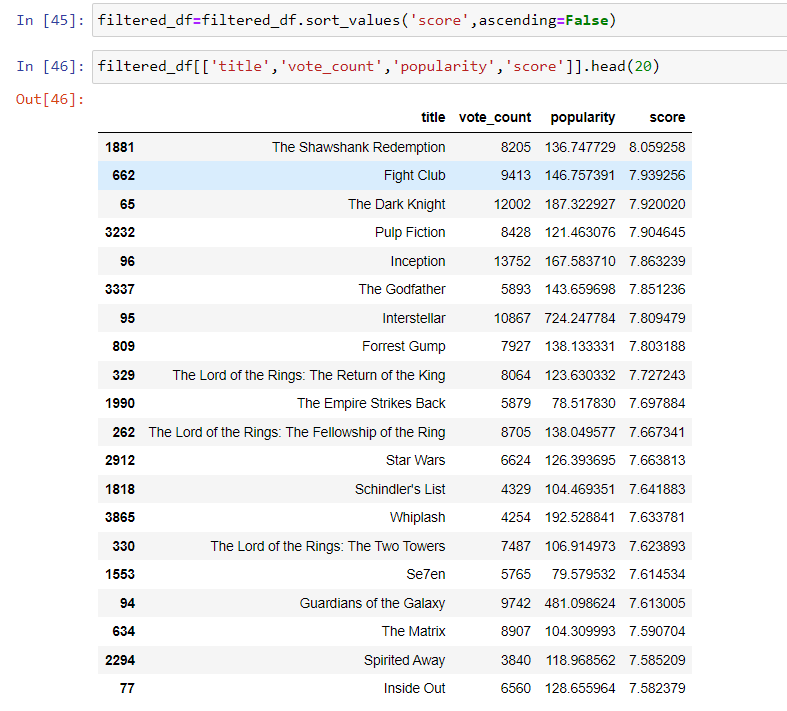
* Applying weighted average

A function is created to calculate the score for each movie using weighted average



The function is then applied to the dataset and the dataset is sorted in descending order based on the score value calculated

The dataset returned is the top 20 movies which are sorted based on our own scoring system.



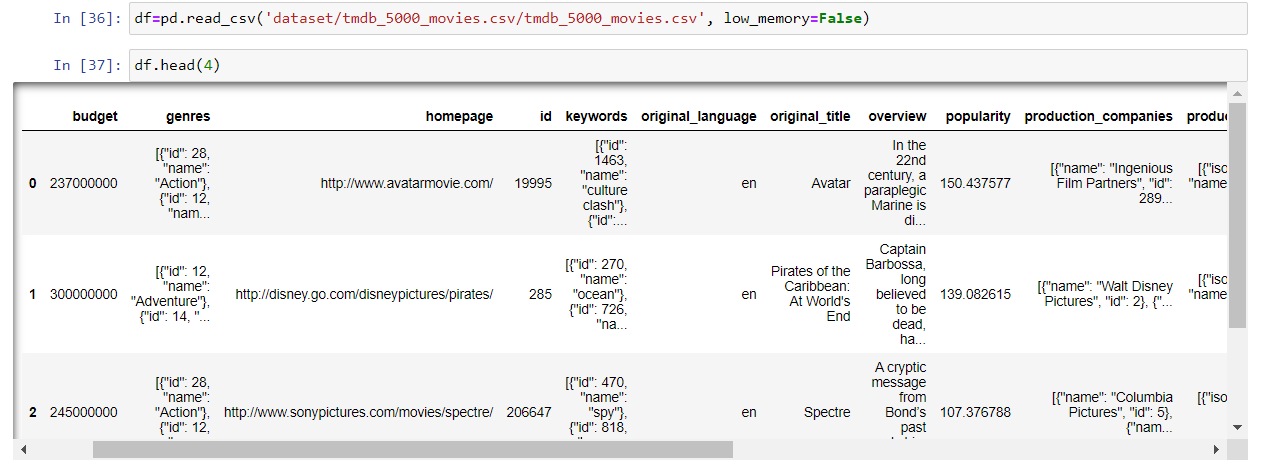
PROJECT FLOW FOR CONTENT FILTERING:

* Pre-processing of the dataset

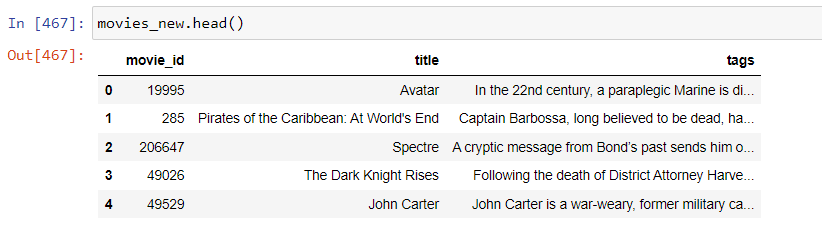
Dataset used: TMDB 5000 Movie Dataset

<https://www.kaggle.com/tmdb/tmdb-movie-metadata?select=tmdb_5000_movies.csv>

Initial dataset



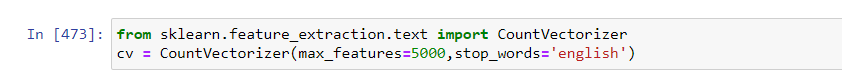
Dataset after pre-processing



Here the columns overview, genres, keywords, crew and cast are used to create the tags column, where the tags are created by converting the format of the initial data in the columns into meaningful words which are later concatenated to form a string which can be later used to find similar movies.

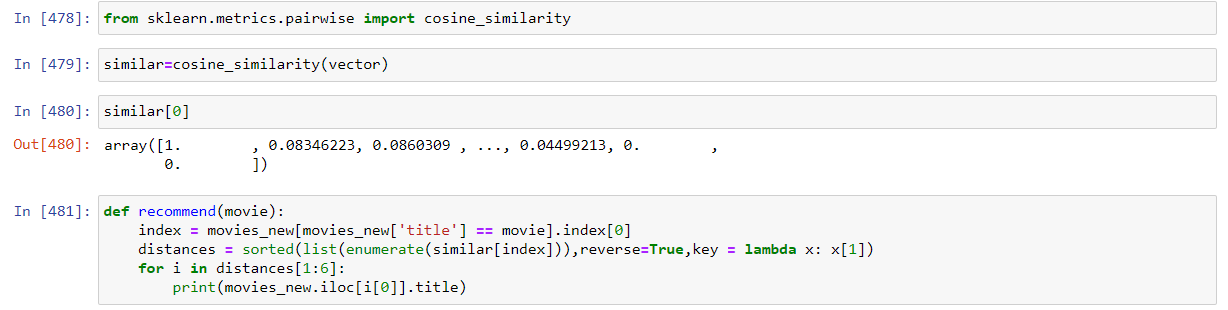
* Applying bag-of-words

Scikit-learn library is used to implement bag-of-words. Using this library stop words are also removed from the bag of words.

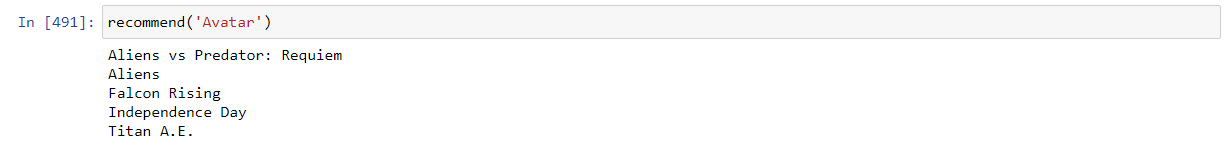


Scikit-learn library is also used to apply cosine similarity to the vectors formed by applying bag of words.

And later a function called recommend is created to return the top 5 movies which are similar to the selected movie with the help of cosine similarity.

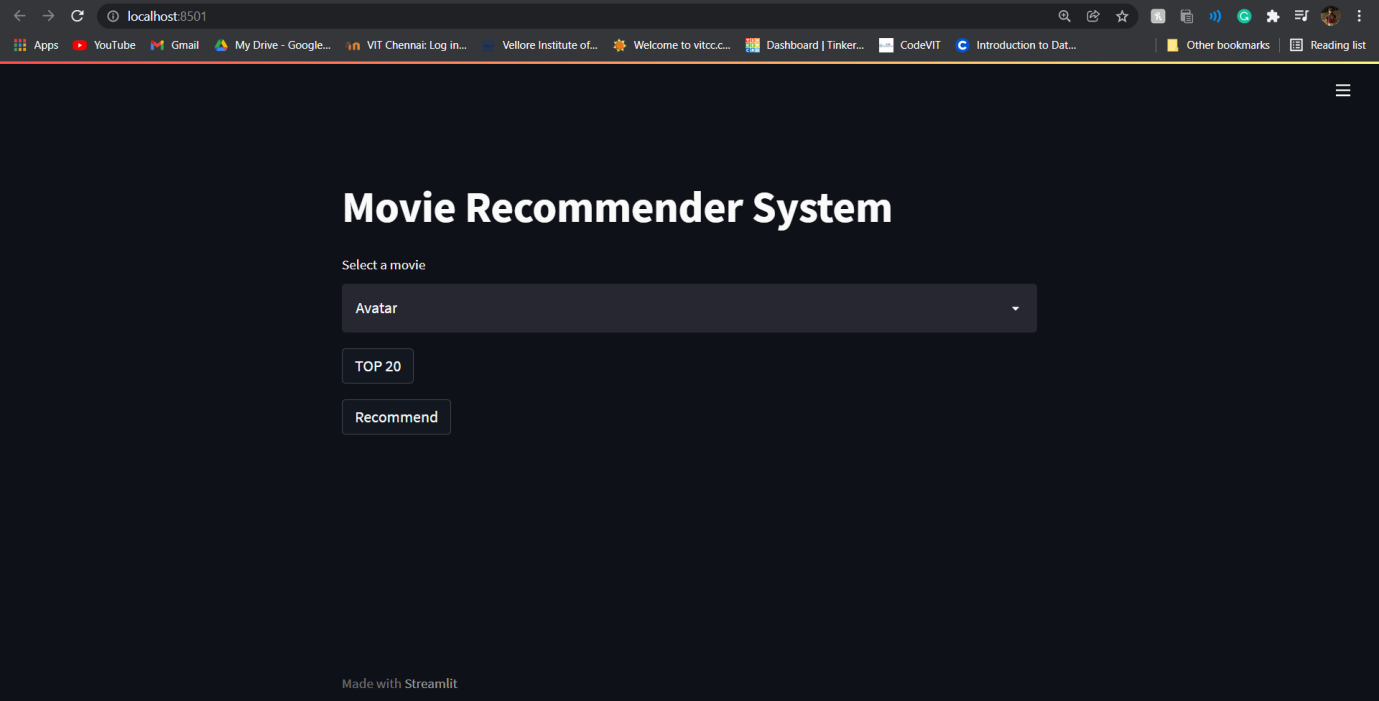


* Working of the function (recommend).

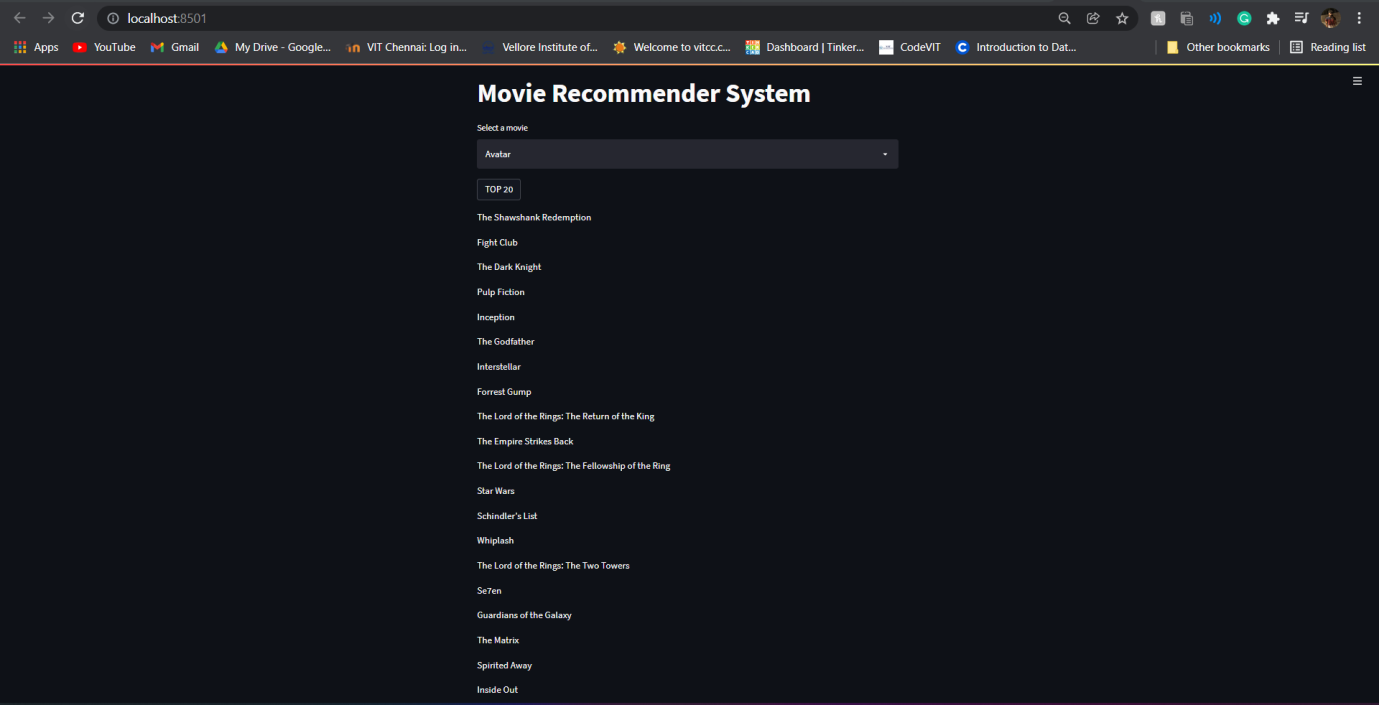


Project Demo:

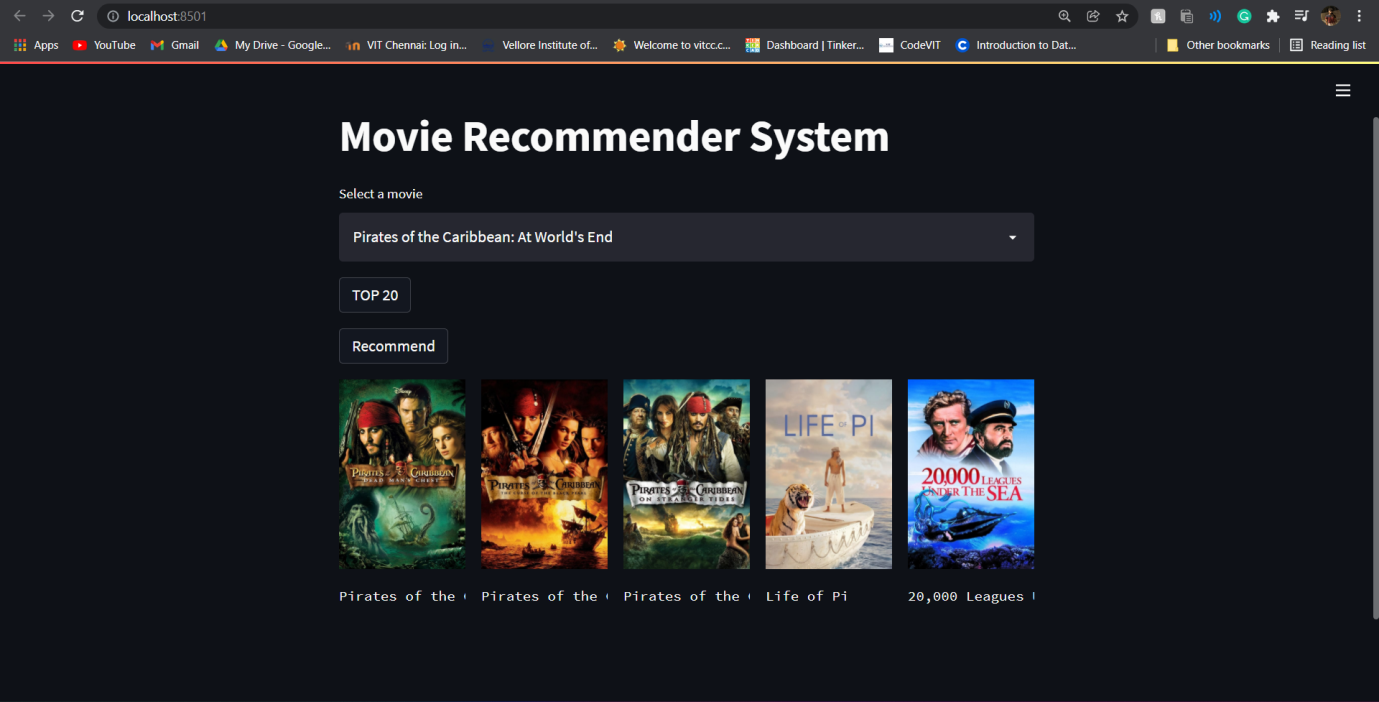
Website outlook

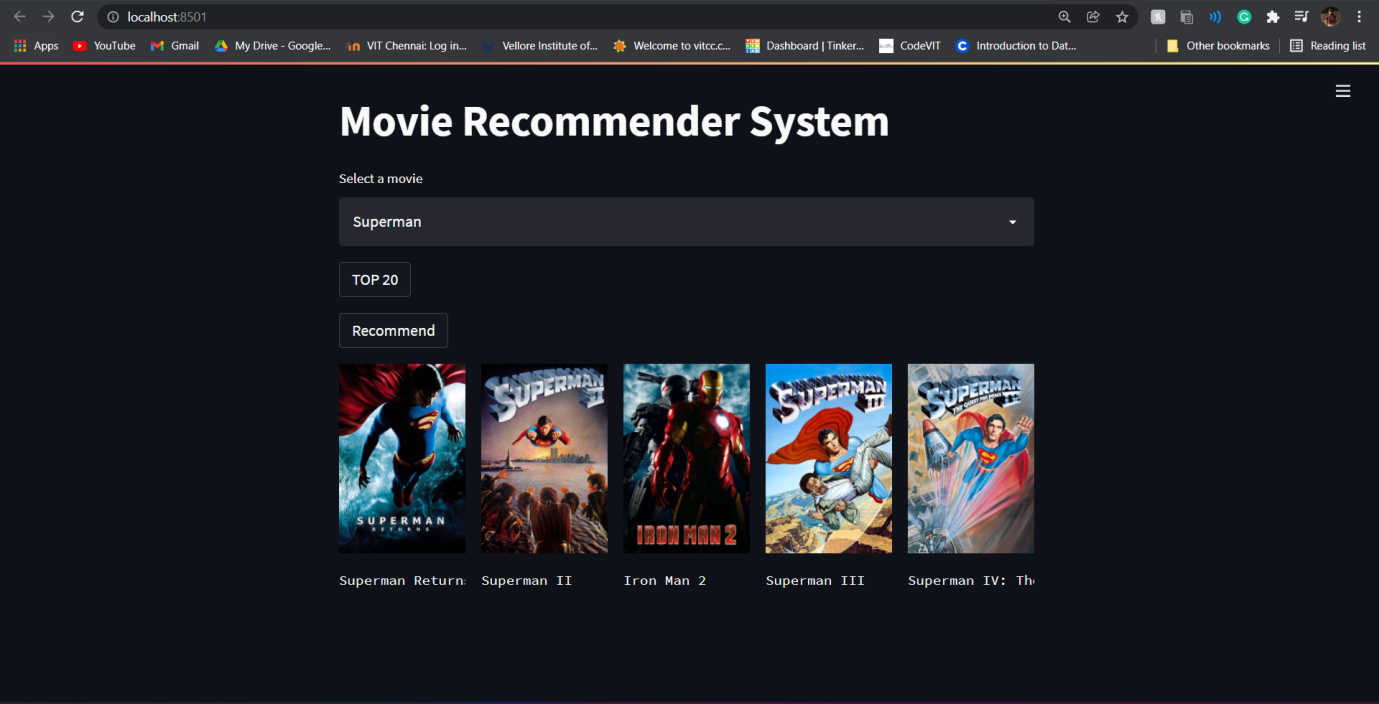


Implementation of score based filtering:



Implementation of content based filtering:





**Conclusion**

So with this we can conclude that the movie recommendation system has been successfully made and is working as we expected. There are also a lot of new avenues that can be appended to the recommendation system and which can be explored to make the recommendation system more efficient and functional.

**Reference**

[**https://www.kaggle.com/tmdb/tmdb-movie-metadata?select=tmdb\_5000\_movies.csv**](https://www.kaggle.com/tmdb/tmdb-movie-metadata?select=tmdb_5000_movies.csv)

[**https://www.kaggle.com/ibtesama/getting-started-with-a-movie-recommendation-system**](https://www.kaggle.com/ibtesama/getting-started-with-a-movie-recommendation-system)

[**https://www.kaggle.com/rounakbanik/movie-recommender-systems**](https://www.kaggle.com/rounakbanik/movie-recommender-systems)

[**https://machinelearningmastery.com/gentle-introduction-bag-words-model/**](https://machinelearningmastery.com/gentle-introduction-bag-words-model/)

[**https://www.investopedia.com/terms/w/weightedaverage.asp**](https://www.investopedia.com/terms/w/weightedaverage.asp)

[**https://scikit-learn.org/stable/modules/feature\_extraction.html**](https://scikit-learn.org/stable/modules/feature_extraction.html)

[**https://scikit-learn.org/stable/tutorial/text\_analytics/working\_with\_text\_data.html**](https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html)

[**https://www.indeed.com/career-advice/career-development/how-to-calculate-weighted-average**](https://www.indeed.com/career-advice/career-development/how-to-calculate-weighted-average)

[**https://www.themoviedb.org/**](https://www.themoviedb.org/)

**APPENDIX**

**Implementation / Code**

**Score based filtering code:**

import pandas as pd

df=pd.read\_csv('dataset/tmdb\_5000\_movies.csv/tmdb\_5000\_movies.csv', low\_memory=False)

df.head(4)

df\_2=df.copy()

df.info()

mean=df['vote\_average'].mean()

mean

```python

minvote=df['vote\_count'].quantile(0.9)

minvote

```

filtered\_df=df.copy().loc[df['vote\_count']>=minvote]

filtered\_df.info()

def weightedavg(x, minvote=minvote, mean=mean):

voters=x['vote\_count']

avg\_vote=x['vote\_average']

return (voters/(voters+minvote)\*avg\_vote) + (minvote/(minvote+voters)\*mean)

filtered\_df['score']=filtered\_df.apply(weightedavg,axis=1)

filtered\_df=filtered\_df.sort\_values('score',ascending=False)

filtered\_df[['title','vote\_count','popularity','score']].head(20)

filtered\_df['title'].values[0]

pop= filtered\_df.sort\_values('score', ascending=False)

import matplotlib.pyplot as plt

plt.figure(figsize=(12,4))

plt.barh(pop['title'].head(6),pop['score'].head(6), align='center',

color='blue')

plt.gca().invert\_yaxis()

plt.xlabel("rating")

plt.title("TOP RATED MOVIES")

import pickle

pickle.dump(filtered\_df,open('top20.pkl','wb'))

**Content based filtering code:**

In [ ]:

import numpy as np

import pandas as pd

In [ ]:

movies=pd.read\_csv('dataset/tmdb\_5000\_movies.csv/tmdb\_5000\_movies.csv', low\_memory=False)

credits=pd.read\_csv('dataset/tmdb\_5000\_credits.csv/tmdb\_5000\_credits.csv', low\_memory=False)

In [ ]:

movies.shape

In [ ]:

credits.head()

In [ ]:

movies = movies.merge(credits,on='title')

In [ ]:

movies.head()

In [ ]:

movies = movies[['movie\_id','title','overview','genres','keywords','cast','crew']]

In [ ]:

movies.head()

In [ ]:

import ast

In [ ]:

movies.dropna(inplace=True)

In [ ]:

movies.iloc[0].genres

In [ ]:

def fetch(text):

l=[]

for i in ast.literal\_eval(text):

l.append(i['name'])

return l

In [ ]:

movies['genres']=movies['genres'].apply(fetch)

In [ ]:

movies['keywords']=movies['keywords'].apply(fetch)

In [ ]:

movies.iloc[0].cast

In [ ]:

def fetch3(text):

l=[]

counter=0

for i in ast.literal\_eval(text):

if counter < 3:

l.append(i['name'])

counter+=1

return l

In [ ]:

movies['cast']=movies['cast'].apply(fetch3)

In [ ]:

movies.head()

In [ ]:

movies.iloc[0].crew

In [ ]:

def director(text):

l=[]

for i in ast.literal\_eval(text):

if i['job']=='Director':

l.append(i['name'])

return l

In [ ]:

movies['crew']=movies['crew'].apply(director)

In [ ]:

movies.head()

In [ ]:

def spacing(k):

l=[]

for i in k:

l.append(i.replace(" ",""))

return l

In [ ]:

movies['cast']=movies['cast'].apply(spacing)

movies['crew']=movies['crew'].apply(spacing)

movies['genres']=movies['genres'].apply(spacing)

movies['keywords']=movies['keywords'].apply(spacing)

In [ ]:

movies.head()

In [ ]:

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

In [ ]:

movies['tags']=movies['overview']+movies['genres']+movies['keywords']+movies["cast"]+movies["crew"]

In [ ]:

movies\_new=movies.drop(columns=['overview','genres','keywords','cast','crew'])

In [ ]:

movies\_new.head()

In [ ]:

movies\_new['tags']=movies\_new['tags'].apply(lambda x: " ".join(x))

In [ ]:

movies\_new.head()

In [ ]:

import nltk

In [ ]:

from nltk.stem.porter import PorterStemmer

ps = PorterStemmer()

In [ ]:

def stem(text):

y=[]

for i in text.split():

y.append(ps.stem(i))

return " ".join(y)

In [ ]:

movies\_new['tags']=movies\_new['tags'].apply(stem)

In [ ]:

movies\_new.iloc[0].tags

In [ ]:

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer(max\_features=5000,stop\_words='english')

In [ ]:

vector = cv.fit\_transform(movies\_new['tags']).toarray()

In [ ]:

vector

In [ ]:

vector[0]

In [ ]:

cv.get\_feature\_names()

In [ ]:

from sklearn.metrics.pairwise import cosine\_similarity

In [ ]:

similar=cosine\_similarity(vector)

In [ ]:

similar[0]

In [ ]:

def recommend(movie):

index = movies\_new[movies\_new['title'] == movie].index[0]

distances = sorted(list(enumerate(similar[index])),reverse=True,key = lambda x: x[1])

for i in distances[1:6]:

print(movies\_new.iloc[i[0]].title)

In [ ]:

recommend('Batman')

In [ ]:

import pickle

pickle.dump(movies\_new,open('movies.pkl','wb'))

pickle.dump(similar,open('similars.pkl','wb'))

In [ ]:

In [ ]:

**Web app code:**

import streamlit as st  
import pickle  
import pandas as pd  
import requests  
import numpy as np  
  
def fetch\_poster(movie\_id):  
 url = "https://api.themoviedb.org/3/movie/{}?api\_key=8265bd1679663a7ea12ac168da84d2e8&language=en-US".format(movie\_id)  
 data = requests.get(url)  
 data = data.json()  
 print(data)  
 poster\_path = data['poster\_path']  
 full\_path = "https://image.tmdb.org/t/p/w500/" + poster\_path  
 return full\_path  
  
  
def recommend(movie):  
 index = movies[movies['title'] == movie].index[0]  
 distances = sorted(list(enumerate(similar[index])), reverse=True, key=lambda x: x[1])  
 recommended\_movies=[]  
 recommended\_movie\_posters=[]  
 for i in distances[1:6]:  
 movie\_id=movies.iloc[i[0]].movie\_id  
 recommended\_movies.append(movies.iloc[i[0]].title)  
 recommended\_movie\_posters.append(fetch\_poster(movie\_id))  
 return recommended\_movies,recommended\_movie\_posters  
  
movies = pickle.load(open('movies.pkl','rb'))  
top20 = pickle.load(open('top20.pkl','rb'))  
movies\_list=movies['title'].values  
similar = pickle.load(open('similars.pkl','rb'))  
  
  
st.title("Movie Recommender System")  
  
selected\_movie = st.selectbox( 'Select a movie',  
 movies\_list)  
if st.button('TOP 20'):  
 for i in range(0,20):  
 st.write(top20['title'].values[i])  
  
  
  
if st.button('Recommend'):  
 names,posters = recommend(selected\_movie)  
  
 col1, col2, col3, col4, col5 = st.columns(5)  
 with col1:  
 st.image(posters[0])  
 st.text(names[0])  
 with col2:  
 st.image(posters[1])  
 st.text(names[1])  
 with col3:  
 st.image(posters[2])  
 st.text(names[2])  
 with col4:  
 st.image(posters[3])  
 st.text(names[3])  
 with col5:  
 st.image(posters[4])  
 st.text(names[4])