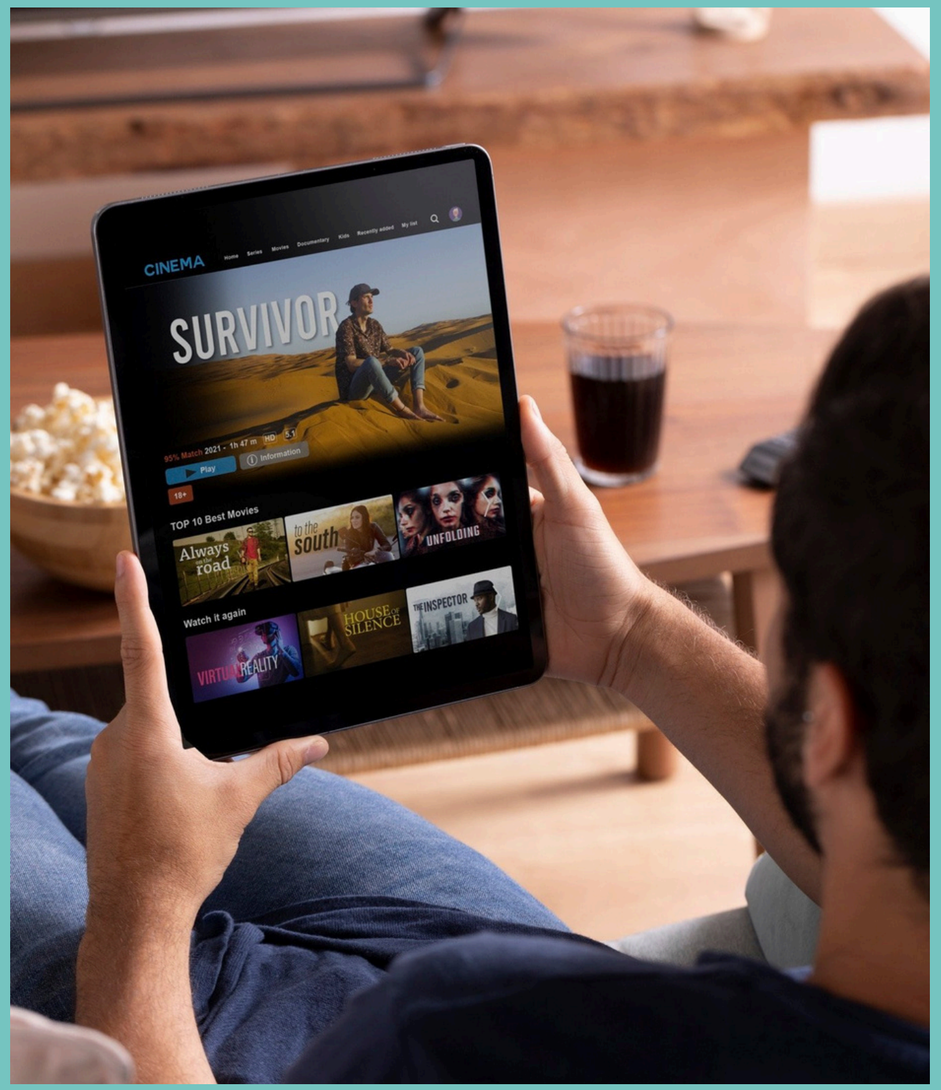
CONTENT BASED MOIVIE RECOMMENDATION SYSTEM

The Power of Recommender Systems

Recommender System is a tool which helps users find the required content and overcome information overload. It predicts interests of users by using Machine Learning algorithms and makes recommendation according to the interest of users.
Real-World Examples - Amazon, Netflix, Youtube

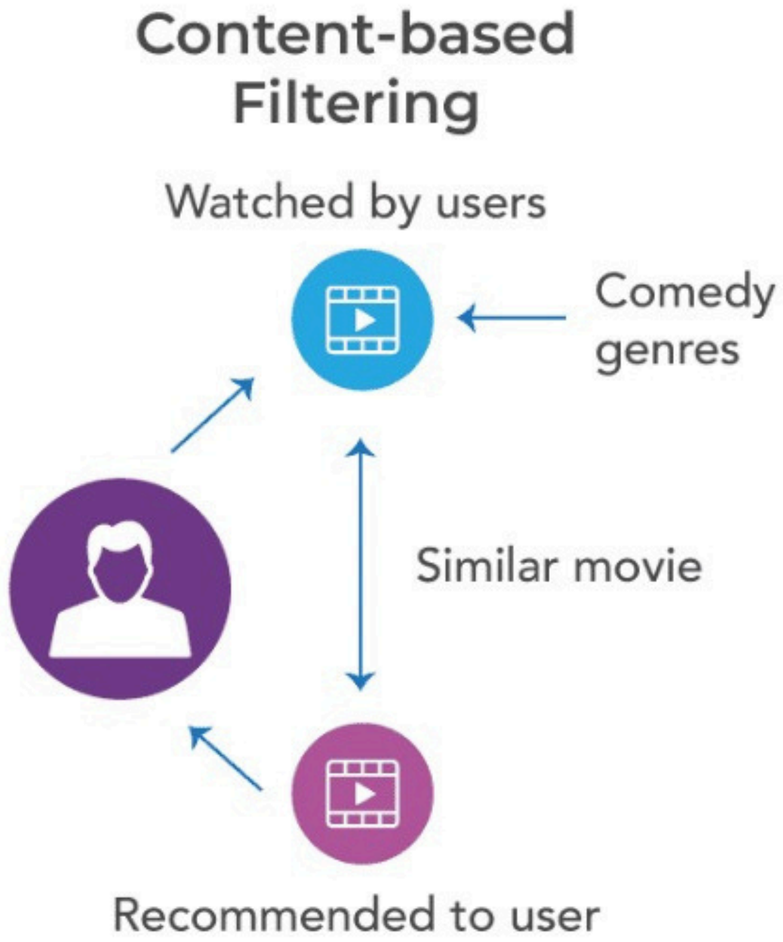


WHAT IS A MOVIE RECOMMENDER SYSTEM?

It's a tool that suggests movies based on user preferences. There are three types: content-based, collaborative filtering, and hybrid. Content-based recommends movies based on similarities in genre, cast, and plot. Collaborative filtering recommends movies based on user behavior. Hybrid combines both methods.

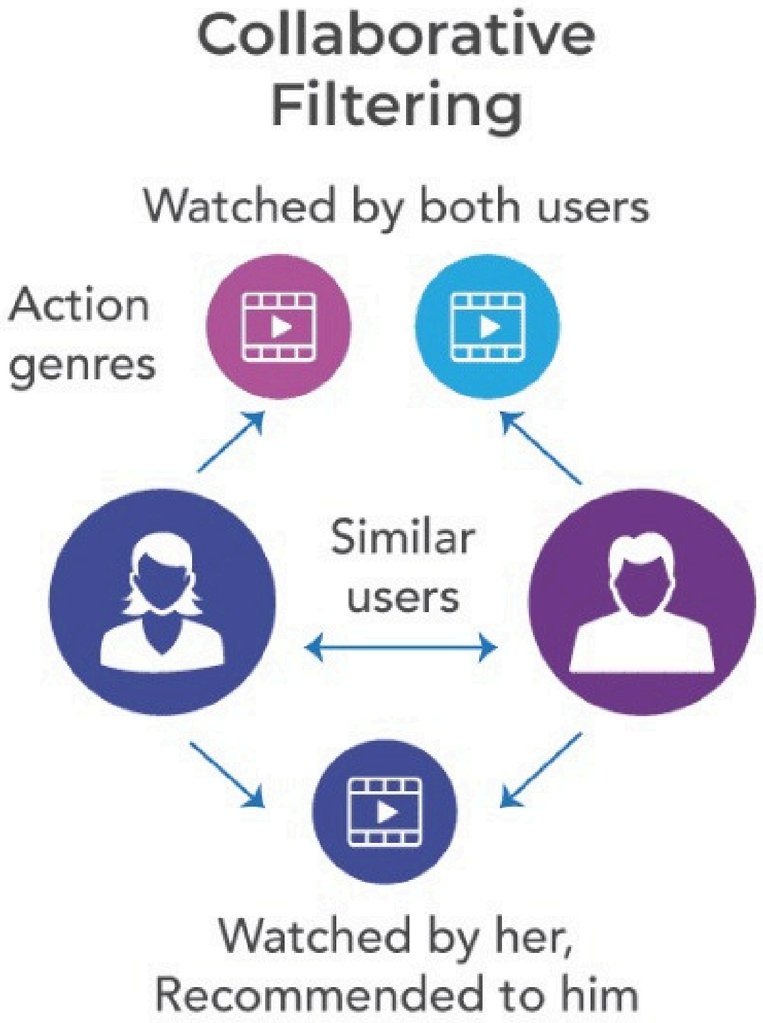
CONTENT-BASED RECOMMENDER SYSTEMS

Content-based recommender systems analyze the attributes of movies and TV shows, such as genre, director, overview, actors, etc. to make suggestions for the users. The intuition behind this sort of recommendation system is that if a user liked a particular movie or show, he/she might like a movie or a show similar to it.



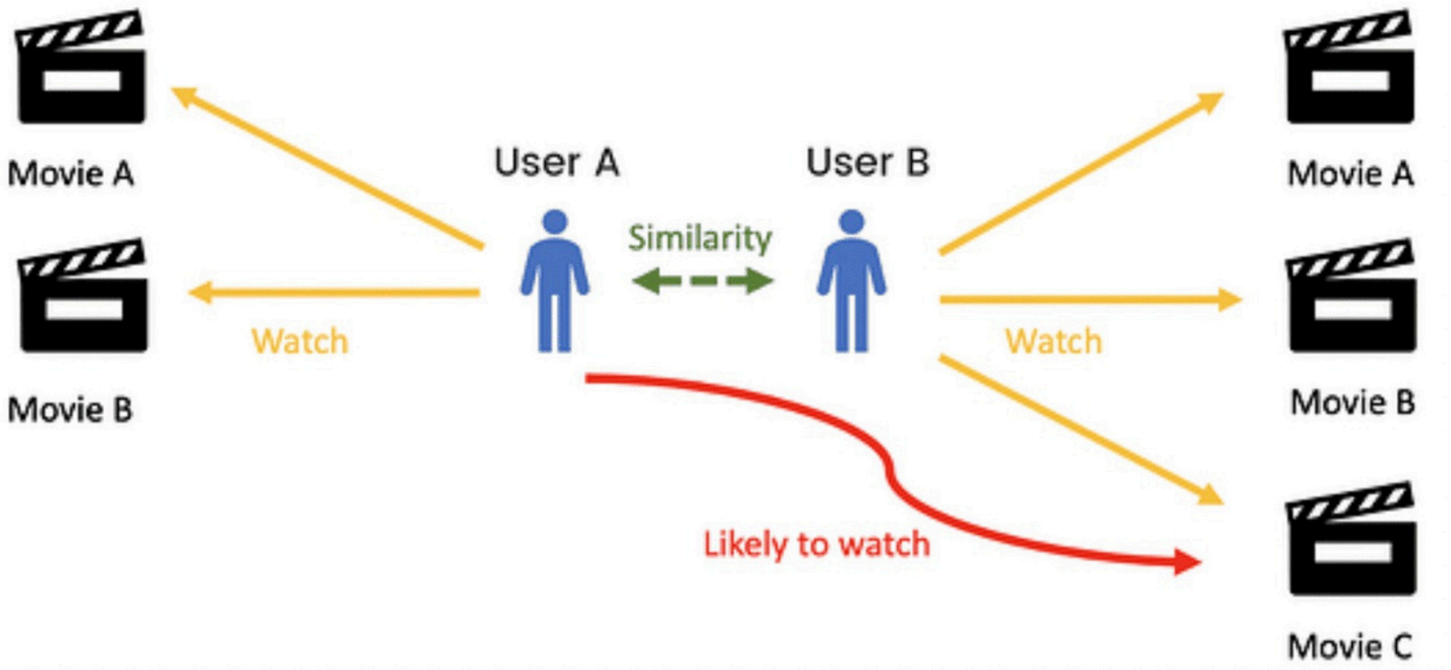
COLLABORATIVE FILTERING

Collaborative filtering recommender system recommends movies based on user behavior. It looks at what movies you've watched, rated, and searched for. It then suggests movies that other users with similar behavior patterns have enjoyed.



COLLABORATIVE FILTERING

For Genre Comedy



BENEFITS

Personalized Recommendations

Overcoming Cold Start Problem

Avoiding Filter Bubbles

Discovery of Niche Movies

Enhanced User Engagement

IMPORTING LIBRARIES AND MODULES

1. **pandas:** A powerful library for data manipulation and analysis. It provides data structures like DataFrame (tabular data) and Series (1D labeled array) along with functions to read, write, filter, and transform data easily.

2. **matplotlib.pyplot:** Part of the Matplotlib library,it's used for creating static, interactive, and animated visualizations in Python. It offers a simple interface for generating plots, charts, and graphs.

3. **ast (Abstract Syntax Trees) :** This module provides tools to work with Python code at an abstract level. It's often used to analyze and manipulate code programmatically.

4. **NLTK (Natural Language Toolkit):** a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.

IMPORTING LIBRARIES AND MODULES

5. **sklearn:** The scikit-learn library is used for machine learning and data analysis tasks in Python. It provides a wide range of tools and algorithms for tasks like classification, regression, clustering, dimensionality reduction, and model evaluation.

6. **pickle:** A built-in module for serializing and deserializing Python objects. It's used to convert complex Python objects (like dictionaries) into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network.

7. **streamlit:** A library used to create interactive web applications for data science and machine learning tasks.

8. **requests:** A popular library for making HTTP requests in Python.It simplifies sending HTTP requests and handling responses, making it easy to interact with web services and APIs.

```
#LOADING DATASETS
movies = pd.read_csv("/Users/daaminibatra/Downloads/archive (1)/tmdb_5000_movies.csv")
credits = pd.read_csv("/Users/daaminibatra/Downloads/archive (1)/tmdb_5000_credits.csv")
```

```
movies.head(1)
```

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_companies	production_
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": "Ingenious Film Partners", "id": 289...	[{"iso_3166 "nam

```
credits.head(1)
```

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...

```
movies.shape
```

```
(4803, 20)
```

```
credits.shape
```

```
(4803, 4)
```

MERGEING THE TWO DATASETS/DATAFRAMES

```
movies=movies.merge(credits,on="title")
movies.head(1)
```

	budget	genres	homepage	id	keywords	original_language	original_titl
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avata

1 rows × 23 columns

```
movies.shape #title column is not repeated
```

```
(4809, 23)
```

```
#relevant columns from the dataset for recommendation system
#genres
#movie_id
#keywords
#title
#overview
#cast
#crew
#release_date
#popularity
#vote_average
#vote_count

movies=movies[["movie_id","title","overview","genres","keywords","cast","cr
```

```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4809 entries, 0 to 4808
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   budget                4809 non-null   int64  
 1   genres                4809 non-null   object 
 2   homepage              1713 non-null   object 
 3   id                    4809 non-null   int64  
 4   keywords              4809 non-null   object 
 5   original_language     4809 non-null   object 
 6   original_title        4809 non-null   object 
 7   overview              4806 non-null   object 
 8   popularity            4809 non-null   float64
 9   production_companies  4809 non-null   object 
 10  production_countries  4809 non-null   object 
 11  release_date          4808 non-null   object 
 12  revenue               4809 non-null   int64  
 13  runtime               4807 non-null   float64
 14  spoken_languages      4809 non-null   object 
 15  status                4809 non-null   object 
 16  tagline               3965 non-null   object 
 17  title                 4809 non-null   object 
 18  vote_average          4809 non-null   float64
 19  vote_count            4809 non-null   int64  
 20  movie_id              4809 non-null   int64  
 21  cast                  4809 non-null   object 
 22  crew                  4809 non-null   object 
dtypes: float64(3), int64(5), object(15)
memory usage: 901.7+ KB
```

DATA CLEANING (REMOVING THE MISSING AND DUPLICATE DATA)

```
#checking missing data exists or not

movies.isnull().sum()
```

```
movie_id        0
title           0
overview        3
genres          0
keywords        0
cast            0
crew            0
popularity      0
release_date    1
vote_average    0
vote_count      0
dtype: int64
```

```
#removing the movies data whose overview or release_date don't exist
movies.dropna(inplace=True)
```

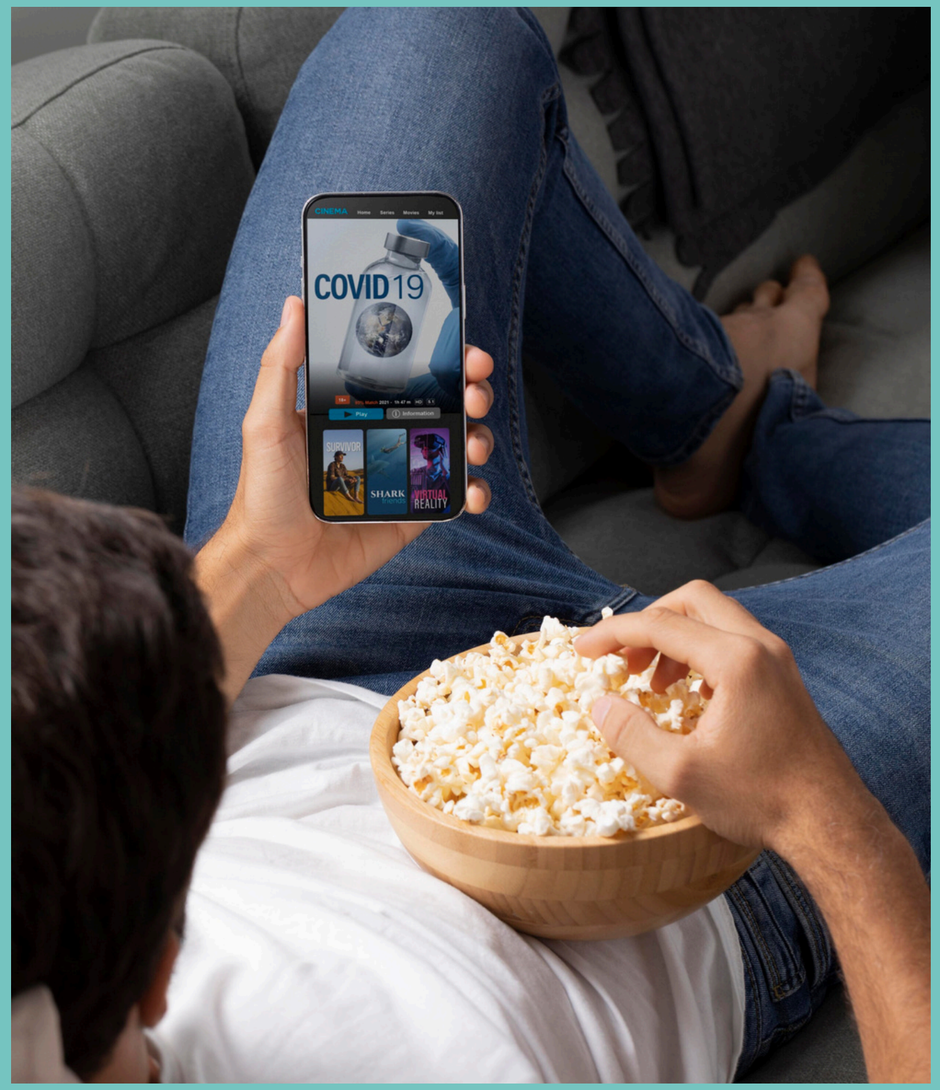
```
movies.isnull().sum()
```

```
movie_id        0
title           0
overview        0
genres          0
keywords        0
cast            0
crew            0
popularity      0
release_date    0
vote_average    0
vote_count      0
dtype: int64
```

```
#checking for duplicate data

movies.duplicated().sum()
```

```
0
```



DEMOGRAPHIC FILTERING

They offer generalized recommendations to every user, based on movie popularity and/or genre. The System recommends the same movies to users with similar demographic features. Since each user is different , this approach is considered to be too simple. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience.

DEMOGRAPHIC FILTERING

IMDB's weighted rating (wr) which is given as :-

$$\text{Weighted Rating (WR)} = \left(\frac{v}{v+m}.R\right) + \left(\frac{m}{v+m}.C\right)$$

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

We already have v(vote_count) and R (vote_average) and C can be calculated

DEMOGRAPHIC FILTERING

```
#C is the mean vote across the whole report
C= movies['vote_average'].mean()
C
```

```
6.094526534859521
```

```
#m is the minimum votes required to be listed in the chart
m= movies['vote_count'].quantile(0.8)
m
```

```
959.0
```

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

```
(962, 11)
```

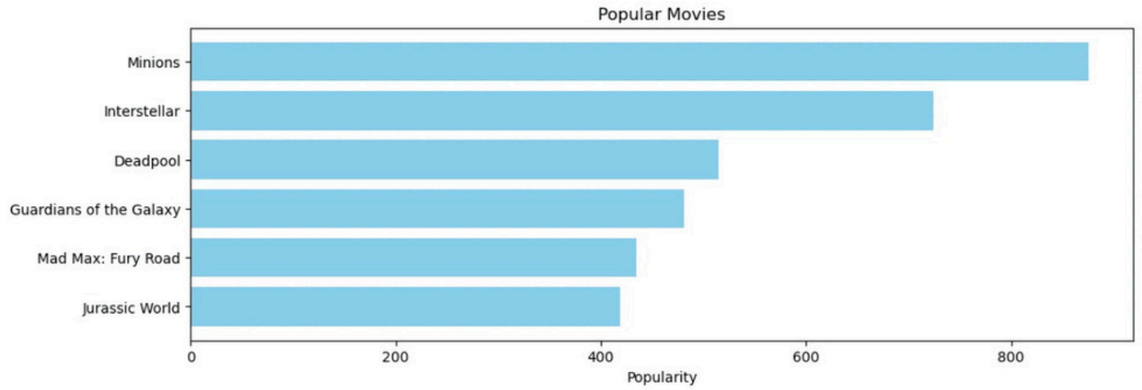
```
#v is the number of votes for the movie
#R is the average rating of the movie
def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']
    # Calculation based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)
```

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

```
#TOP 6 POPULAR MOVIES
pop= movies.sort_values('popularity', ascending=False)
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))

plt.barh(pop['title'].head(6),pop['popularity'].head(6), align='center',
        color='skyblue')
plt.gca().invert_yaxis()
plt.xlabel("Popularity")
plt.title("Popular Movies")
```

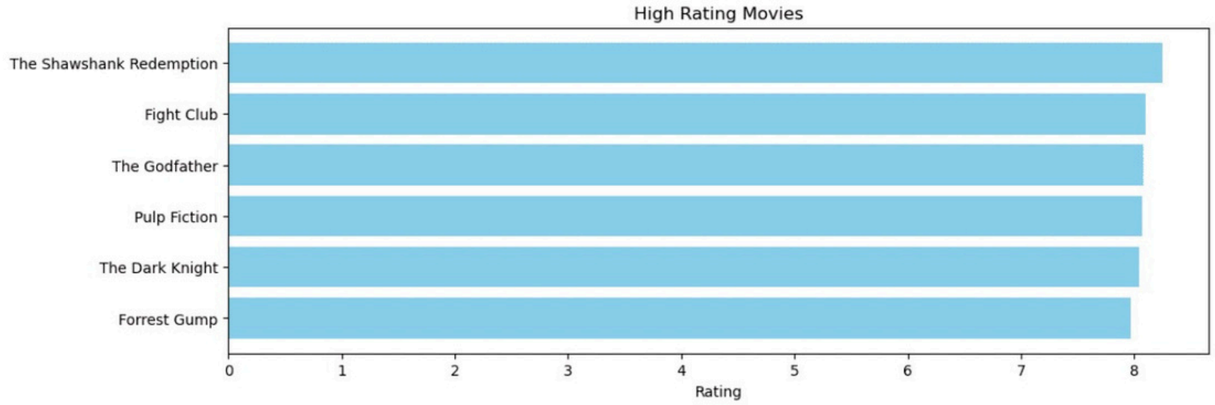
```
Text(0.5, 1.0, 'Popular Movies')
```




```
#TOP 6 HIGH RATING MOVIES
pop= movies.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='skyblue')
plt.gca().invert_yaxis()
plt.xlabel("Rating")
plt.title("High Rating Movies")
```

```
Text(0.5, 1.0, 'High Rating Movies')
```



DATA TRANSFORMATION

FEATURE ENGINEERING

```
movies.iloc[0].genres
```

```
'[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 1
4, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]'
```

```
#convert
#[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14,
#to
#["Action","Adventure","Fantasy","Science Fiction"]
```

```
#converting string of list to list using literal_eval from ast library
```

```
import ast
```

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

```
movies["genres"]=movies["genres"].apply(convert)

#applying convert method to keywords attribute as well
movies["keywords"]=movies["keywords"].apply(convert)
```

```
def top3cast(obj):
    L=[]
    count=0
    for i in ast.literal_eval(obj):
        if count!=3:
            L.append(i["name"])
            count+=1
        else:
            break
    return L
```

```
movies["cast"]=movies["cast"].apply(top3cast)
```

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

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

```
temp=movies["genres"]+movies["keywords"]+movies["cast"]+movies["crew"]
temp.head(3)
```

```
0    [Action, Adventure, Fantasy, Science Fiction, ...
1    [Adventure, Fantasy, Action, ocean, drug abuse...
2    [Action, Adventure, Crime, spy, based on novel...
dtype: object
```

```
movies.head(3)
```

	movie_id	title	overview	genres	keywords	cast	crew	popularity	releas
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 Weaver]	[James Cameron]	150.437577	200
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]	139.082615	200
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]	[Sam Mendes]	107.376788	201

```
#converting "Sam Worthington" into "SamWorthington" to improve the recommer
for i in ["genres","keywords","cast","crew"]:
    temp[i]=movies[i].apply(lambda x:[j.replace(" ","") for j in x])
```

```
movies["tags"]= movies["overview"].apply(lambda x:x.split())+temp["genres"]
temp["keywords"]+temp["cast"]+temp["crew"]
```

```
temp.cast.head(2)
```

```
0    [SamWorthington, ZoeSaldana, SigourneyWeaver]
1       [JohnnyDepp, OrlandoBloom, KeiraKnightley]
Name: cast, dtype: object
```

```
movies["tags"]=movies["tags"].apply(lambda x:" ".join(x))
```

```
movies.head(3)
```

overview	genres	keywords	cast	crew	popularity	release_date	vote_average	vote_count	score	tags
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 Weaver]	[James Cameron]	150.437577	2009-12-10	7.2	11800	7.116910	in the 22nd century, a parapleg marin is dispa...
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]	139.082615	2007-05-19	6.9	4500	6.758500	captain barbossa, long believ to be dead, ha c...
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]	[Sam Mendes]	107.376788	2015-10-26	6.3	4466	6.263678	a cryptic messag from bond' past send him on a...

CountVectorizer class used for converting collections of text documents into a matrix of token counts.

USING STEMMING

```
import nltk
```

```
from nltk.stem.porter import PorterStemmer
ps=PorterStemmer()
```

```
def stem(text):
    y=[]
    for i in text.split():
        y.append(ps.stem(i))
    return " ".join(y)
```

```
movies["tags"]=movies["tags"].apply(stem)
```

Example

```
print(stem("Lovely Loving"))
```

```
love love
```

Stemming is a text preprocessing technique used in natural language processing to reduce words to their base or root form.

USING CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
cv=CountVectorizer(max_features=5000,stop_words="english")
```

```
vector=cv.fit_transform(movies["tags"]).toarray()
```

```
vector
```

```
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]])
```

```
vector.shape
```

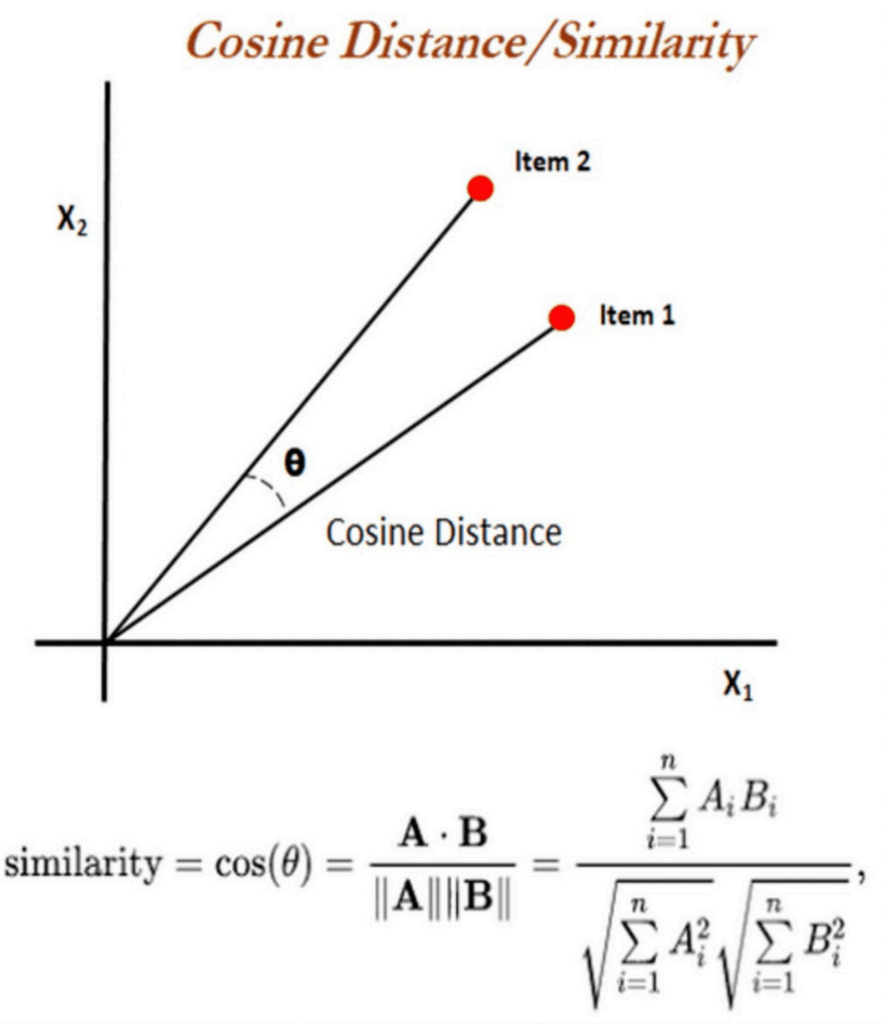
```
(962, 5000)
```

```
len(cv.get_feature_names_out())
```

```
5000
```

COSINE SIMILARITY

Cosine Similarity measures the similarity between two vectors by calculating the cosine of the angle between them. In movie terms, we can represent each movie as a vector of ratings for different features (e.g. genre, actors, plot). By comparing these vectors, we can find movies that are similar to each other.



USING COSINE SIMILARITY

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
cosine_similarity(vector).shape
```

```
(962, 962)
```

```
similarity=cosine_similarity(vector)
```

```
sorted(list(enumerate(similarity[0])), reverse=True, key=lambda x:x[1])[1:6]
```

```
[(785, 0.26452002850644324),
 (576, 0.23002185311411805),
 (389, 0.22886885410853178),
 (358, 0.22621385432622335),
 (60, 0.2204792759220492)]
```

BASIC SEARCH ENGINE FOR RECOMMENDING TOP 5 MOVIES SIMILAR TO THE INPUT MOVIE

```
def recommend(movie):
    movie_index=movies[movies["title"]==movie].index[0]
    distances=similarity[movie_index]
    movies_list=sorted(list(enumerate(distances)), reverse=True, key=lambda x:x[1])[1:6]

    for i in movies_list:
        print(i[0],movies.iloc[i[0]].title)
```

```
recommend("Avatar")
```

```
785 Aliens
576 Predators
389 Battle: Los Angeles
358 Independence Day
60 Jupiter Ascending
```

```
movie_index=movies[movies["title"]=="Avatar"].index[0]
```

```
distances=similarity[movie_index]
movies_list=sorted(list(enumerate(distances)), reverse=True, key=lambda x:x[1])[1:6]
```

```
movies_list
```

```
[(785, 0.26452002850644324),
 (576, 0.23002185311411805),
 (389, 0.22886885410853178),
 (358, 0.22621385432622335),
 (60, 0.2204792759220492)]
```

USING PICKLE

```
import pickle
```

```
pickle.dump(movies.to_dict(),open("movie_dict2.pkl","wb"))
```

```
pickle.dump(similarity,open("similarity2.pkl","wb"))
```

```
unique_genres = set()
```

```
unique_genres_list = sorted(list(movies['genres'].explode().unique()))
```

```
pickle.dump(unique_genres_list,open("unique_genres2.pkl","wb"))
```

```
actors = set()
actors = sorted(list(movies['cast'].explode().unique()))
```

```
pickle.dump(actors,open("actors.pkl","wb"))
```

```
actors=pickle.load(open('actors.pkl','rb'))
```

```
directer=set()
directer = sorted(list(movies['crew'].explode().unique()))
```

```
pickle.dump(directer,open("directer.pkl","wb"))
```

PyCharm

Importing Libraries and Modules

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

Deserializing

```
movie_dict=pickle.load(open('movie_dict2.pkl','rb'))
movies = pd.DataFrame(movie_dict)

similarity=pickle.load(open('similarity2.pkl','rb'))

unique_genres=pickle.load(open('unique_genres2.pkl','rb'))

actors=pickle.load(open('actors.pkl','rb'))

directers=pickle.load(open('directer.pkl','rb'))
```

Function for fetching Posters

```
def fetch_poster(movie_id):
    response = requests.get("https://api.themoviedb.org/3/movie/{}?api_key=ce07f8f5c01025d46d1d86d7b3e63fa3&language=en-US".format(movie_id))
    data = response.json()
    print(data)
    return "https://image.tmdb.org/t/p/w500/" + data['poster_path']
```

```python
def recommend(option):
    if option in movies['title'].values:
        movie_index = movies[movies['title'] == option].index[0]
        distance = similarity[movie_index]
        movie_list = sorted(list(enumerate(distance)), reverse=True, key=lambda x: x[1])[1:6]
        recommended_movies = []
        recommended_movies_posters=[]
        for i in movie_list:
            movie_id=movies.iloc[i[0]].movie_id
            recommended_movies.append(movies.iloc[i[0]])
            # fetch poster from api
            recommended_movies_posters.append(fetch_poster(movie_id))
    elif option in unique_genres:
        mask = movies.genres.apply(lambda x: option in x)
        filtered_movie = movies[mask]
        filtered_movie = filtered_movie.sort_values(by='popularity', ascending=False)[0:5]
        suggested_movies = [(idx, row['movie_id']) for idx, row in filtered_movie.iterrows()]

        recommended_movies = []
        recommended_movies_posters=[]
        for i in suggested_movies:
            movie_id = i[1]
            recommended_movie = movies[movies['movie_id'] == movie_id].iloc[0]
            recommended_movies.append(recommended_movie)
            recommended_movies_posters.append(fetch_poster(movie_id))
    elif option in actors:
        mask = movies.cast.apply(lambda x: option in x)
        filtered_movie = movies[mask]
        filtered_movie = filtered_movie.sort_values(by='popularity', ascending=False)[0:5]
        suggested_movies = [(idx, row['movie_id']) for idx, row in filtered_movie.iterrows()]

        recommended_movies = []
        recommended_movies_posters=[]
        for i in suggested_movies:
            movie_id = i[1]
            recommended_movie = movies[movies['movie_id'] == movie_id].iloc[0]
            recommended_movies.append(recommended_movie)
            recommended_movies_posters.append(fetch_poster(movie_id))
    else:
        mask = movies.crew.apply(lambda x: option in x)
        filtered_movie = movies[mask]
        filtered_movie = filtered_movie.sort_values(by='popularity', ascending=False)[0:5]
        suggested_movies = [(idx, row['movie_id']) for idx, row in filtered_movie.iterrows()]

        recommended_movies = []
        recommended_movies_posters=[]
        for i in suggested_movies:
            movie_id = i[1]
            recommended_movie = movies[movies['movie_id'] == movie_id].iloc[0]
            recommended_movies.append(recommended_movie)
            recommended_movies_posters.append(fetch_poster(movie_id))
    return recommended_movies,recommended_movies_posters
```

Movie Title

Genre

Actor

Directer

Content Based Recommendation

Popularity Based Recommendation

CONCLUSION

Content-based recommendations are revolutionizing the way we choose movies to watch. By analyzing the features of movies we've enjoyed in the past, we can enjoy a more personalized and time-saving movie selection experience. While there are limitations to this method, it's clear that content-based recommendations are here to stay.

Thanks!

Daamini Batra