III - Mini Project

Design and Implement a Movie Recommendation System.

A recommender system aims to suggest relevant content or products to users that might be liked or purchased by them. It helps to find items that the user is looking for — and they don't even realize it until the recommendation is displayed. Different strategies have to be applied for different clients and they are determined by available data

---

There are two main techniques used in recommendation system, known as content-based filtering and collaborative filtering

> ***Content Based :*** Content-based filtering uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit feedback.For example, if user A watched two horror movies, another horror movie will be proposed to him.

> ***Collaborative Filtering :*** To address some of the limitations of content-based filtering, collaborative filtering uses similarities between users and items simultaneously to provide recommendations. Collaborative filtering models can recommend an item to user A based on the interests of a similar user B. Main advantage is that they learn users' embeddings automatically, without the need for hand-engineering.

Importing Libraries & Datasets

In [1]:
```
import pandas as pd # Python library for data analysis and data frame
import numpy as np # Numerical Python library for linear algebra and computations
pd.set_option('display.max_columns', None) # code to display all columns

# Visualisation libraries
import matplotlib.pyplot as plt
import seaborn as sns

# libraries for text processing
import nltk
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# to display images
from skimage import io

# to save the required files
import pickle

import warnings
warnings.filterwarnings('ignore') # To prevent kernel from showing any warning
```

In [2]:
```
df = pd.read_csv('movies.csv\movies.csv')
```

Initial Analysis

Understand The Data 🧠

In [3]:
```
# How does the data look like?
df.head()
```

Out[3]:

	id	title	genres	original_language	overview	popularity	production_companies	release_date	budget	
0	760161	Orphan: First Kill	Horror-Thriller	en	After escaping from an Estonian psychiatric fa...	8098.027	Dark Castle Entertainment-Entertainment One-Ea...	2022-07-27	0.0	9
1	718930	Bullet Train	Action-Comedy-Thriller	en	Unlucky assassin Ladybug is determined to do h...	7949.491	Columbia Pictures-87North Productions	2022-07-03	90000000.0	23
2	744276	After Ever Happy	Romance-Drama	en	As a shocking truth about a couple's families ...	4017.342	Voltage Pictures-Vertical Entertainment-CalMap...	2022-08-24	0.0	
3	579974	RRR	Action-Drama	te	A fictional history of two legendary revolutio...	3416.323	Lyca Productions-DVV Entertainment-Pen Studios	2022-03-24	69000000.0	16
4	532639	Pinocchio	Fantasy-Adventure-Family	en	A wooden puppet embarks on a thrilling adventu...	3239.378	Walt Disney Pictures-Depth of Field	2022-09-07	0.0	

In [4]:
```
#How big is data?
df.shape
```

Out[4]:
```
(744493, 20)
```

In [5]:
```
# What is the data type of cols?
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 744493 entries, 0 to 744492
Data columns (total 20 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   id                    744493 non-null  int64  
 1   title                 744489 non-null  object 
 2   genres                523793 non-null  object 
 3   original_language     744493 non-null  object 
 4   overview              623360 non-null  object 
 5   popularity            744493 non-null  float64
 6   production_companies  345206 non-null  object 
 7   release_date          687712 non-null  object 
 8   budget                744493 non-null  float64
 9   revenue               744493 non-null  float64
 10  runtime               706032 non-null  float64
 11  status                744493 non-null  object 
 12  tagline               109989 non-null  object 
 13  vote_average          744493 non-null  float64
 14  vote_count            744493 non-null  float64
 15  credits               513164 non-null  object 
 16  keywords              215024 non-null  object 
 17  poster_path           545394 non-null  object 
 18  backdrop_path         225367 non-null  object 
 19  recommendations       43687 non-null   object 
dtypes: float64(6), int64(1), object(13)
memory usage: 113.6+ MB
```

In [6]:
```
# Are there any missing values?
df.isnull().sum()
```

Out[6]:
```
id                          0
title                       4
genres                 220700
original_language           0
overview               121133
popularity                  0
production_companies   399287
release_date            56781
budget                      0
revenue                     0
runtime                 38461
status                      0
tagline                634504
vote_average                0
vote_count                  0
credits                231329
keywords               529469
poster_path            199099
backdrop_path          519126
recommendations        700806
dtype: int64
```

In [7]:
```
# Are there duplicate values?
df.duplicated().sum()
```

Out[7]:
```
173
```

1. There are more than 7 lakh rows and 20 columns
2. Data consists of 6 numeric columns and 14 object/string columns
3. There are many columns with lots of missing valuest
4. There are duplicate values present in data

Preprocessing

In [8]:
```
# lets get rid of the duplicate values
df.drop_duplicates(inplace=True)
```

Let's check if there are any movies with same title

In [9]:
```
df['title'].duplicated().sum()
```

Out[9]:
```
168414
```

There are 168580 movies with same title. Now these might be duplicate movies but there's possibility that some might be different movies with same title
Thats why Let's check if there are any movies with same title and same release date

In [10]:
```
df[['title','release_date']].duplicated().sum()
```

Out[10]:
```
84371
```

In [11]:
```
# lets get rid of the duplicate movies
df.drop_duplicates(subset=['title','release_date'], inplace=True)
```

In [12]:
```
df.shape
```

Out[12]:
```
(659949, 20)
```

Now we have 6 lakh movies but most of the movies have 0 vote count. so we will consider only those movies which have at least more than 20 vote counts.

In [13]:
```
# filtering the movies
df1 = df[df.vote_count >= 20].reset_index()
```

In [14]:
```
df1.isnull().sum()
```

Out[14]:
```
index                      0
id                         0
title                      0
genres                   179
original_language          0
overview                 475
popularity                 0
production_companies    3306
release_date               2
budget                     0
revenue                    0
runtime                   16
status                     0
tagline                20420
vote_average               0
vote_count                 0
credits                  647
keywords                9576
poster_path              138
backdrop_path           2436
recommendations        11658
dtype: int64
```

In [15]:
```
# Replace the Nan with ''
df1.fillna('', inplace=True)
```

We are making content based recommendation system and genres , overview are very important to find similar movies. So i will delete movies which don't have genres and overview.

In [16]:
```
# finding index with '' genres and overview
index = df1[(df1['genres']=='') & (df1['overview']=='')].index
```

In [17]:
```
# droping those index
df1.drop(index, inplace=True)
```

- genres, keywords and credits are seperated by '-'
- So replacing that with space
- and from credits only extracting first values words

In [18]:
```
df1['genres'] = df1['genres'].apply(lambda x: ' '.join(x.split('-')))
df1['keywords'] = df1['keywords'].apply(lambda x: ' '.join(x.split('-')))
df1['credits'] = df1['credits'].apply(lambda x: ' '.join(x.replace(' ', '').split('-')[:5]))
```

Creating Tags

Lets create a column with all the important columns which describe a movie, so we can create tags out of it

In [19]:
```
df1['tags'] = df1['overview'] +' '+ df1['genres'] +' '+ df1['keywords'] +' '+ df1['credits'] +' '+ df1
```

In [20]:
```
df1.tags[0]
```

Out[20]:
```
'After escaping from an Estonian psychiatric facility Leena Klammer travels to America by impersonati
ng Esther the missing daughter of a wealthy family. But when her mask starts to slip she is put again
st a mother who will protect her family from the murderous “child” at any cost. Horror Thriller psych
opath family secrets prequel murder impersonator mental patient psycho killer escaped mental patient
missing daughter estonia female psychopath IsabelleFuhrman JuliaStiles RossifSutherland MatthewFinlan
HiroKanagawa en'
```

Let's apply stemming on tags column

Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes.

In [21]:
```
stemmer = SnowballStemmer("english")
def stem(text):
    y = []

    for i in text.split():
        y.append(stemmer.stem(i))

    return ' '.join(y)

df1['tags'] = df1['tags'].apply(stem)
```

In [22]:
```
# Removing punctuation
df1['tags'] = df1['tags'].str.replace('[^\w\s]','')
```

TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction.

In [23]:
```
tfidf = TfidfVectorizer(stop_words='english')
```

In [24]:
```
tfidf_matrix = tfidf.fit_transform(df1['tags'])
```

In [25]:
```
df1.tags[0]
```

Out[25]:
```
'after escap from an estonian psychiatr facil leena klammer travel to america by imperson esther the
miss daughter of a wealthi family but when her mask start to slip she is put against a mother who wil
l protect her famili from the murder child at ani cost horror thriller psychopath famili secret prequ
el murder imperson mental patient psycho killer escap mental patient miss daughter estonia femal psyc
hopath isabellefuhrman juliastil rossifsutherland matthewfinlan hirokanagawa en'
```

Recommendation System

In [39]:
```
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations(title):
    # Get the index of the movie that matches the title
    idx = df1.index[df1['title'] == title][0]
    # show given movie poster
    try:
        a = io.imread(f'https://image.tmdb.org/t/p/w500/{df1.loc[idx, "poster_path"]}')
        plt.imshow(a)
        plt.axis('off')
        plt.title(title)
        plt.show()
    except:pass

    print('Recommendations\n')


    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(
        cosine_similarity(
            tfidf_matrix,
            tfidf_matrix[idx])))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:10]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    result = df1.iloc[movie_indices]

    # show reco. movie posters
    fig, ax = plt.subplots(2, 4, figsize=(15,15))
    ax=ax.flatten()
    for i, j in enumerate(result.poster_path):
        try:
            ax[i].axis('off')
            ax[i].set_title(result.iloc[i].title)
            a = io.imread(f'https://image.tmdb.org/t/p/w500/{j}')
            ax[i].imshow(a)
        except: pass
    fig.tight_layout()
    fig.show()
```

Testing the Recommender

In [40]:
```
get_recommendations("Avengers: Endgame")
```

Avengers: Endgame

Recommendations

The Avengers

Avengers: Infinity War

Team Thor

Avengers: Age of Ultron

Captain America: Civil War

Justice League: The Flashpoint Paradox

Marvel Studios: Assembling a Universe

Captain Marvel

In [28]:
```
pickle.dump(df1,open('movie_list.pkl','wb'))
pickle.dump(tfidf_matrix,open('tfidf_matrix.pkl','wb'))
```

Deployment

The Deployment of this WEB APP is done using streamlit. Forst we install streamlit on our device and then give commands to run the app

In [29]:
```
# Install streamlit
# NB : If any error occurs during installation process , run it again
# !pip install -q streamlit
# !npm install -g localtunnel -U
```

In [30]:

```
%%writefile movie_recommendation_app.py

import pickle
import streamlit as st
from sklearn.metrics.pairwise import cosine_similarity
from PIL import Image

@st.cache_data
def get_recommendation(title):
    idx = df1.index[df1['title'] == title][0]
    poster = f'https://image.tmdb.org/t/p/w500/{df1.loc[idx, "poster_path"]}'

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(
        cosine_similarity(
            tfidf_matrix,
            tfidf_matrix[idx])))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:13]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    result = df1.iloc[movie_indices]

    recommended_movie_names = []
    recommended_movie_posters = []
    recommended_movie_overview = []

    for i, j in enumerate(result.poster_path):
        recommended_movie_names.append(result.iloc[i].title)
        recommended_movie_posters.append(f'https://image.tmdb.org/t/p/w500/{j}')
        recommended_movie_overview.append(result.iloc[i].overview)
    return poster, recommended_movie_names, recommended_movie_posters, recommended_movie_overview


image = Image.open('Movie recommender system.png')
st.image(image)

st.markdown('You might have wondered sometime or at some point that how do platforms like Netflix or Am
st.markdown('There are two main techniques used in recommendation system, known as content-based filter
st.markdown('For this project I have used Content Based Recommendation System, It uses attributes such

df1 = pickle.load(open('movie_list.pkl ', 'rb'))
tfidf_matrix = pickle.load(open('tfidf_matrix.pkl ', 'rb'))

movies_list = df1['title'].values
selected_movie = st.selectbox('Type and Choose The Movie',movies_list)

if st.button('Show Recommendation'):
    poster,recommended_movie_names,recommended_movie_posters,recommended_movie_overview = get_recommend
    st.image(poster,width=160)
    col1, col2, col3, col4 = st.columns(4)
    with col1:
        st.image(recommended_movie_posters[0])
        st.markdown(recommended_movie_names[0])
        with st.expander("OverView"):
            st.write(recommended_movie_overview[0])

        st.image(recommended_movie_posters[4])
        st.markdown(recommended_movie_names[4])
        with st.expander("OverView"):
            st.write(recommended_movie_overview[4])

        st.image(recommended_movie_posters[8])
        st.markdown(recommended_movie_names[8])
        with st.expander("OverView"):
            st.write(recommended_movie_overview[8])

    with col2:
        st.image(recommended_movie_posters[1])
        st.markdown(recommended_movie_names[1])
```

```
        with st.expander("OverView"):
            st.write(recommended_movie_overview[1])

        st.image(recommended_movie_posters[5])
        st.markdown(recommended_movie_names[5])
        with st.expander("OverView"):
            st.write(recommended_movie_overview[5])

        st.image(recommended_movie_posters[9])
        st.markdown(recommended_movie_names[9])
        with st.expander("OverView"):
            st.write(recommended_movie_overview[9])

    with col3:
        st.image(recommended_movie_posters[2])
        st.markdown(recommended_movie_names[2])
        with st.expander("OverView"):
            st.write(recommended_movie_overview[2])

        st.image(recommended_movie_posters[6])
        st.markdown(recommended_movie_names[6])
        with st.expander("OverView"):
            st.write(recommended_movie_overview[6])

        st.image(recommended_movie_posters[10])
        st.markdown(recommended_movie_names[10])
        with st.expander("OverView"):
            st.write(recommended_movie_overview[10])

    with col4:
        st.image(recommended_movie_posters[3])
        st.markdown(recommended_movie_names[3])
        with st.expander("OverView"):
            st.write(recommended_movie_overview[3])

        st.image(recommended_movie_posters[7])
        st.markdown(recommended_movie_names[7])
        with st.expander("OverView"):
            st.write(recommended_movie_overview[7])

        st.image(recommended_movie_posters[11])
        st.markdown(recommended_movie_names[11])
        with st.expander("OverView"):
            st.write(recommended_movie_overview[11])
```

```
Overwriting movie_recommendation_app.py
```

The App will display a UI in your terminal with public URL of the tunnel and other status and metrics information about connections made over the tunnel.

In [31]:
```
!streamlit run movie_recommendation_app.py & npx localtunnel --port 8501
```

```
^C
```