Eunbin Yoo

Data Science Project

Movie Recommendation System

Content-Based Movie
Recommender System
using movies data

TABLE OF CONTENTS

1 INTRODUCTION & PROJECT GOAL

2 DATA SOURCE USED

3 PIPELINE

4 DATA PRE-PROCESSING

5 MODELLING BUILDING

6 EVALUATION & CONCLUSION

INTRODUCTION

Netflix is undeniably the biggest leader in the streaming world with a total global subscriber count of 232.5 million.

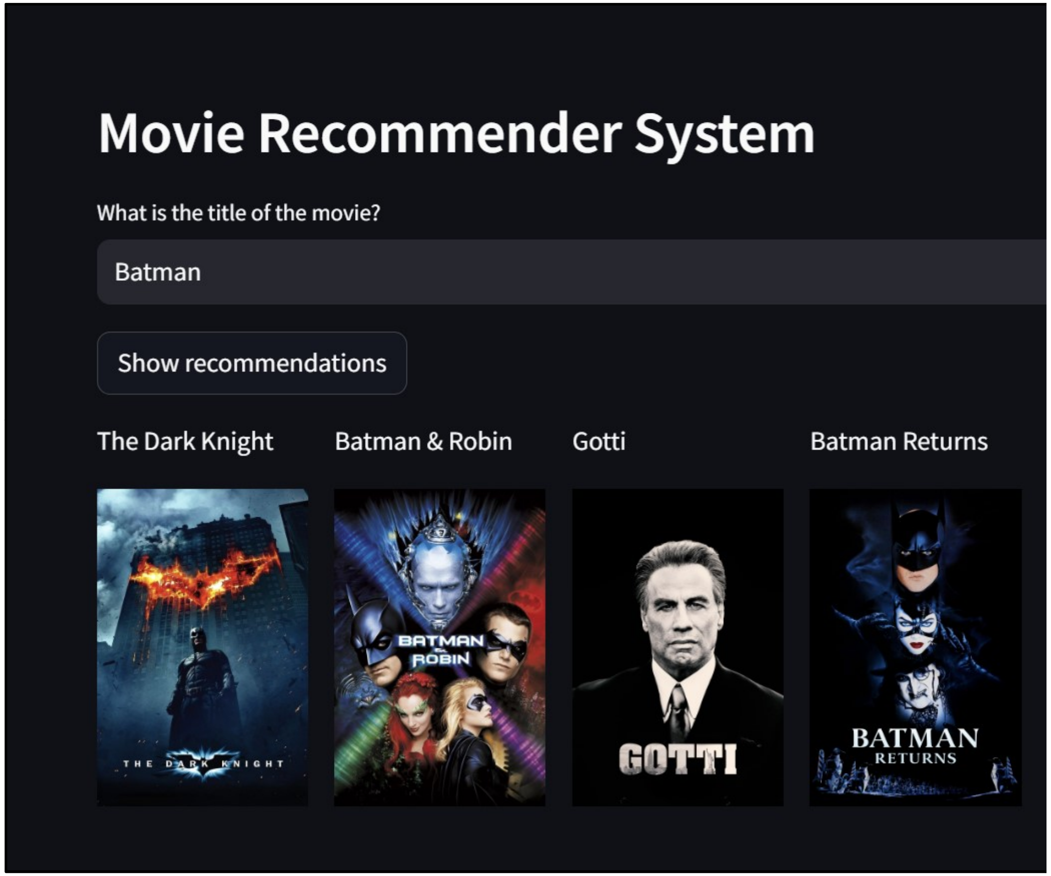
One key behind Netflix's success is its innovative utilization of data science. By analyzing viewing habits, time spent, and content preferences, Netflix offers tailored recommendations of movies and TV shows for each user.

These predictions lead to improved user experiences and increased engagement resulting in user retention, which is crucial for every business.

PROJECT GOAL

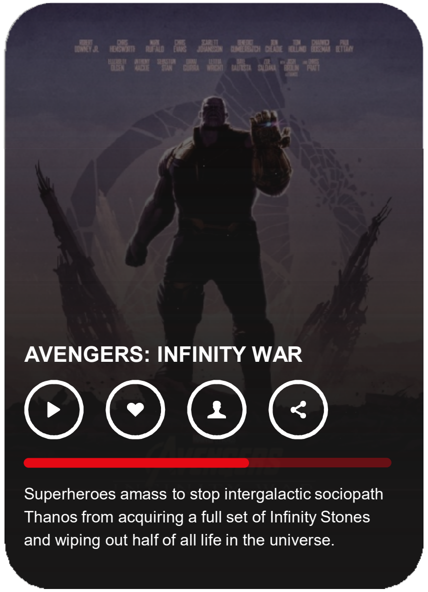
The goal of this project is to develop a recommendation engine that find similar movies to a user's input. For this Movie Recommender System, we will be using “content-based filtering” method.

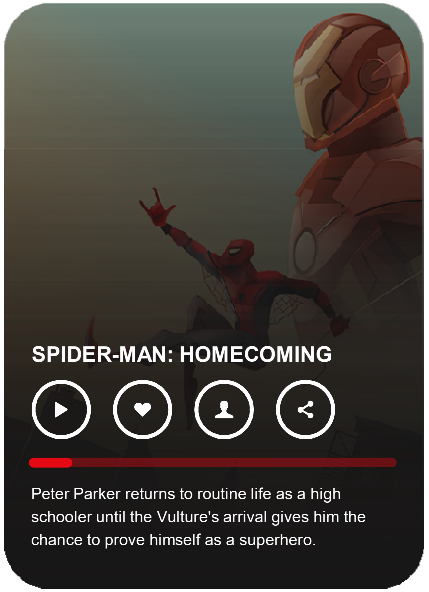
Content-based filtering uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit feedback. For example, if user A watched two science fiction movies, another science fiction movie will be proposed to them.



CONTENT-BASED FILTERING

A content-based recommendation engine is a subset of recommender systems that aims to infer the user's preferences in order to recommend items that are similar in content to items they have previously liked. Unlike collaborative recommendation systems, this approach does not require other user data to be able to give recommendation.





SUMMARY OF ANALYSIS

This is the summary of my model building process.

1. Fetching Dataset

Coding Environment
- Jupyter Notebook and Python

Data Source
- ""Movies Daily Update Dataset" from Kaggle

2. Pre-processing Data

Data Manipulation and Cleaning
- Remove duplicates
- Remove null values
- Feature selection

3. Vectorizer

Text Vectorization
- CountVectorizer to convert text data to numbers

4. Model Building

Similarity Score
- Cosine similarity model was used to build the recommender system algorithm

5. Testing Model

Evaluation
- Test a model on a dataset
- Identify any issues or limitations that may affect its performance.

6. Deployment

Streamlit
- Host a local server with Streamlit and configure the site

DATA SOURCE

For this project we fetch dataset from Kaggle's "Movies Daily Update Dataset". This dataset was made available through Akshay Pawar on Kaggle. This data is deemed credible as it operates under a public domain.
The dataset contains metadata for more than 700k movies listed in the TMDB Dataset.

"Movies Daily Update Dataset"

```
# first five rows of dataset
movie.head()
```

	id	title	genres	original_language	overview	popularity	production_companies	release_date	budget	revenue	runtime	
0	823464	Godzilla x Kong: The New Empire	Science Fiction-Action-Adventure	en	Following their explosive showdown Godzilla an...	10484.676	Legendary Pictures-Warner Bros. Pictures	2024-03-27	150000000.0	558503759.0	115.0	Re
1	615656	Meg 2: The Trench	Action-Science Fiction-Horror	en	An exploratory dive into the deepest depths of...	8763.998	Apelles Entertainment-Warner Bros. Pictures-di...	2023-08-02	129000000.0	352056482.0	116.0	Re
2	758323	The Pope's Exorcist	Horror-Mystery-Thriller	en	Father Gabriele Amorth Chief Exorcist of the V...	5953.227	Screen Gems-2.0 Entertainment-Jesus & Mary-Wor...	2023-04-05	18000000.0	65675816.0	103.0	Re
3	667538	Transformers: Rise of the Beasts	Action-Adventure-Science Fiction	en	When a new threat capable of destroying the en...	5409.104	Skydance-Paramount-di Bonaventura Pictures-Bay...	2023-06-06	200000000.0	407045464.0	127.0	Re
4	693134	Dune: Part Two	Science Fiction-Adventure	en	Follow the mythic journey of Paul Atreides as ...	4742.163	Legendary Pictures	2024-02-27	190000000.0	683813734.0	167.0	Re

△ First 5 rows of original dataset

DATA CHECK

HOW MANY MOVIES AND FEATURES ARE GIVEN?

```python
# shape of dataset
print(movie.shape)
```

```
(722444, 20)
```

We have about 722K movies in the dataset and about 20 features of each movies.

WHAT ARE THE DATA-TYPES AND COLUMN NAMES?

```python
# column names and data tyles
movie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 722444 entries, 0 to 722443
Data columns (total 20 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   id                    722444 non-null  int64
 1   title                 722440 non-null  object
 2   genres                511992 non-null  object
 3   original_language     722444 non-null  object
 4   overview              604118 non-null  object
 5   popularity            722444 non-null  float64
 6   production_companies  337285 non-null  object
 7   release_date          670657 non-null  object
 8   budget                722444 non-null  float64
 9   revenue               722444 non-null  float64
 10  runtime               688084 non-null  float64
 11  status                722444 non-null  object
 12  tagline               108359 non-null  object
 13  vote_average          722444 non-null  float64
 14  vote_count            722444 non-null  float64
 15  credits               497621 non-null  object
 16  keywords              210495 non-null  object
 17  poster_path           537760 non-null  object
 18  backdrop_path         222969 non-null  object
 19  recommendations       35179 non-null   object
dtypes: float64(6), int64(1), object(13)
memory usage: 110.2+ MB
```

The dataset mainly consists of text data.

DATA PRE-PROCESSING

DATA CLEANING:

- Drop unnecessary columns and duplicated movies
- Drop movies with *Vote_Count* < 350, movies with no *Genres* and *Overview*
- Remove "-" sign from *Genre*, *Keywords* and *Credits*
- Replace all the null value from *Genres* and *Overview* with “ “ (empty string)

CODE:

```python
# drop unnecessary columns
movie = movie.drop(["production_companies", "popularity", "budget", "revenue", "status",
                    "recommendations", "runtime", "vote_average", "backdrop_path", "tagline"], axis=1)
```

```python
# drop the driplicates in whole dataset
movie.drop_duplicates(inplace = True)
```

```python
# check if duplicated titles have same release date
movie[['title', 'release_date']].duplicated().sum()
```

```python
# get rid of duplicated titles with same release date
movie.drop_duplicates(subset = ["title","release_date"], inplace=True)
```

```python
movie = movie[movie.vote_count >= 350].reset_index()
```

```python
movie.isnull().sum()
```

```python
# replacing all the null value from genres adn overview with " " (empty string)
movie.fillna("", inplace = True)
```

```python
index = movie[(movie.genres == "") & (movie.overview == "")].index
movie.drop(index, inplace=True)
```

```python
# replacing genres, credits and keywords - with " " (empty strings)
movie.genres = movie.genres.apply(lambda x: " ".join(x.split("-")))
movie.keywords = movie.keywords.apply(lambda x: " ".join(x.split("-")))
movie.credits = movie.credits.apply(lambda x: " ".join(x.split("-")))
```

FEATURE SELECTION

To predict similar movies using natural language processing techniques, we will utilize text-based data as input for our machine learning model. To facilitate this process, we will create a new column called "Tags" that encompasses all the crucial text features such as Overview, Genres, Keywords, and Original Language. This will enable us to make accurate predictions of similar movies.

CODE:

```
# create a new column 'tag'
movie["tag"] = movie["overview"] + " " + movie["genres"] + " " + movie["keywords"] +
               " " + movie["credits"] + " " + movie["original_language"]
```

```
# make new framework for important features
movie_new = movie[["id", "title", "tag", 'poster_path']]
movie_new.shape
```

```
(7720, 4)
```

```
# convert to lower case form
movie_new = movie_new.copy()
movie_new.tag = movie_new.tag.apply(lambda x: x.lower())
```

```
movie_new.tag[0]
```

```
'following their explosive showdown godzilla and kong must reunite against
rld challenging their very existence - and our own. science fiction action
sy world giant ape godzilla king kong mongkey rebecca hall brian tyree henr
chel house ron smyck chantelle jamieson greg hatton kevin copeland tess dot
bain chika ikogwe vincent b. gorce yeye zhou jamaliah othman nick lawler jc
roney en'
```

▽ First 5 rows of the final dataset

```
movie_new.head()
```

	id	title	tag	poster_path
0	823464	Godzilla x Kong: The New Empire	Following their explosive showdown Godzilla an...	/v4uvGFAkKuYfyKLGZnYj6I47ERQ.jpg
1	615656	Meg 2: The Trench	An exploratory dive into the deepest depths of...	/4m1Au3YkjqsxF8iwQy0fPYSxE0h.jpg
2	758323	The Pope's Exorcist	Father Gabriele Amorth Chief Exorcist of the V...	/9JBEPLTPSm0d1mbEcLxULjJq9Eh.jpg
3	667538	Transformers: Rise of the Beasts	When a new threat capable of destroying the en...	/gPbM0MK8CP8A174rmUwGsADNYKD.jpg
4	693134	Dune: Part Two	Follow the mythic journey of Paul Atreides as ...	/czembW0Rk1Ke7ICJGahbOhdCuhV.jpg

TEXT VECTORIZATION

A vectorizer is also a part of pre-processing used in natural language processing (NLP) and text analysis to convert a collection of text documents into numerical feature vectors. This is done by analyzing the occurrence of words or terms within the documents and encoding them as numerical values.

For this project, *CountVectorizer* function from scikit-learn library was used.

CODE:

```
# set for 5000 most repeated words, and exclude stop words
cv = CountVectorizer(stop_words = 'english', max_features = 5000)
```

```
# fit tags in count vector and change it to array to use
vectors = cv.fit_transform(movie_new['tag']).toarray()
```

```
vectors
```

```
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, 1, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

```
cv.get_feature_names_out()[60:65]
```

```
array(['addison', 'adel', 'adkin', 'adler', 'adolesc'], dtype=object)
```

▽ First 5 rows of the final dataset

```
movie_new.head()
```

	id	title	tag	poster_path
0	823464	Godzilla x Kong: The New Empire	follow their explos showdown godzilla and kong...	/v4uvGFAkKuYfyKLGZnYj6l47ERQ.jpg
1	615656	Meg 2: The Trench	an exploratori dive into the deepest depth of ...	/4m1Au3YkjqsxF8iwQy0fPYSxE0h.jpg
2	758323	The Pope's Exorcist	father gabriel amorth chief exorcist of the va...	/9JBEPLTPSm0d1mbEcLxULjJq9Eh.jpg
3	667538	Transformers: Rise of the Beasts	when a new threat capabl of destroy the entir ...	/gPbM0MK8CP8A174rmUwGsADNYKD.jpg
4	693134	Dune: Part Two	follow the mythic journey of paul atreid as he...	/czembW0Rk1Ke7lCJGahbOhdCuhV.jpg

SIMILARITY SCORE

For this project's machine learning model, cosine similarity model was used to build a recommender system algorithm. *cosine_similarity* function from scikit-learn library was used.

COSINE SIMILARITY

Cosine similarity is the measure of similarity between two vectors, by computing the cosine of the angle between two vectors projected into multidimensional space. It can be applied to items available on a dataset to compute similarity to one another via keywords or other metrics.

```
# Cosine similarity

from sklearn.metrics.pairwise import cosine_similarity
similarity = cosine_similarity(vector, vector)
```

```
similarity
```

```
array([[1.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 1.        , 0.13363062, ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.13363062, 1.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 1.        , 0.18257419,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.18257419, 1.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        1.        ]])
```

RECOMMENDER 1

The resulted recommendations do not seem bad, but we see some unrelated titles.
Including more features like “cast”, “director” and “country” may improve the performance of the model.

RECOMMENDER ALGORITHM

```
netflix_new = netflix_new.reset_index()
indices = pd.Series(netflix_new.index, index = netflix_new['title'])

def get_recommendations(title):
    title = title.replace(' ', '').lower()
    index = indices[title]
    sim_scores = list(enumerate(similarity[index]))
    sim_scores = sorted(sim_scores, key = lambda x: x[1], reverse = True)
    for i in sim_scores[1:11]:
        print(netflix.iloc[i[0]].title)
```

RESULTS

```
get_recommendations('Twilight')
```

```
Adrift
The Tourist
Free State of Jones
In Search of Fellini
The Legacy of a Whitetail Deer Hunter
Birth of the Dragon
Django Unchained
Company of Heroes
Felon
The Outpost
```

The resulted recommendation for the movie ‘Twilight’ do not seem bad, but we see some unrelated titles.
Including more features like “cast”, “director” and “country” may improve the performance of the model.

RECOMMENDER 2

The new recommender engine was built with additional features “cast”, “director” and “country”.

VECTORIZATION

```
features_new = ['title', 'director', 'cast', 'listed_in', 'country', 'description']
netflix_new2 = netflix_filled[features_new]

netflix_new2 = netflix_new2.copy()

for feature in features_new:
    netflix_new2[feature] = netflix_new2[feature].apply(data_cleaning)
```

```
# Combining all features in one column
def content_include(x):
    return x['title']+ ' ' + x['director'] + ' ' + x['cast']
          + ' ' +x['listed_in']+' '+ x['country'] + ' ' + x['description']

netflix_new2['tag'] = netflix_new2.apply(content_include, axis=1)
```

```
# Removing stop words
cv_2 = CountVectorizer(max_features = 8807, stop_words = 'english')
vector_2 = cv_2.fit_transform(netflix_new2['tag'].values.astype('U')).toarray()
vector_2.shape
```

```
(8807, 8807)
```

SIMILARITY SCORE

```
# Cosine similarity
similarity_2 = cosine_similarity(vector_2)
```

```
similarity_2
```

```
array([[1.        , 0.        , 0.        , ..., 0.23570226, 0.20412415,
        0.        ],
       [0.        , 1.        , 0.14285714, ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.14285714, 1.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.23570226, 0.        , 0.        , ..., 1.        , 0.19245009,
        0.        ],
       [0.20412415, 0.        , 0.        , ..., 0.19245009, 1.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        1.        ]])
```

RECOMMENDER 2

We apply the new similarity score calculated to the recommender algorithm.

RECOMMENDER ALGORITHM

```
# Cosine similarity
similarity_2 = cosine_similarity(vector_2)
```

```
def get_recommendations_new(title):
    title = title.replace(' ', '').lower()
    index = indices[title]
    sim_scores = list(enumerate(similarity_2[index]))
    sim_scores = sorted(sim_scores, key = lambda x: x[1], reverse = True)
    for i in sim_scores[1:11]:
        print(netflix.iloc[i[0]].title)
```

RESULTS

```
get_recommendations_new('Twilight')
```

```
The Twilight Saga: Breaking Dawn: Part 2
The Twilight Saga: Breaking Dawn: Part 1
The Twilight Saga: Eclipse
The Twilight Saga: New Moon
Mosul
Black Site Delta
Expo
Adrift
The Legacy of a Whitetail Deer Hunter
Remember Me
```

The result from the recommender with additional features shows an improvement in the recommendation for the movie ‘Twilight’. We see more titles that are similar to ‘Twilight’ in terms of genre, story, cast and etc. This will be our final model for our Movie Recommender System.

RECOMMENDATIONS

These are the recommendations generated by our final recommender system.

‘Stranger Things’

```
get_recommendations_new('Stranger things')
```

```
Beyond Stranger Things
Manifest
Helix
Warrior Nun
The Umbrella Academy
The Messengers
The Twilight Zone (Original Series)
Chilling Adventures of Sabrina
Nightflyers
The 4400
```

‘Peaky Blinders’

```
get_recommendations_new('Peaky Blinders')
```

```
Giri / Haji
Get Even
Hinterland
Happy Valley
The Frankenstein Chronicles
Kiss Me First
Retribution
Murder Maps
Secrets of Great British Castles
Behind Her Eyes
```

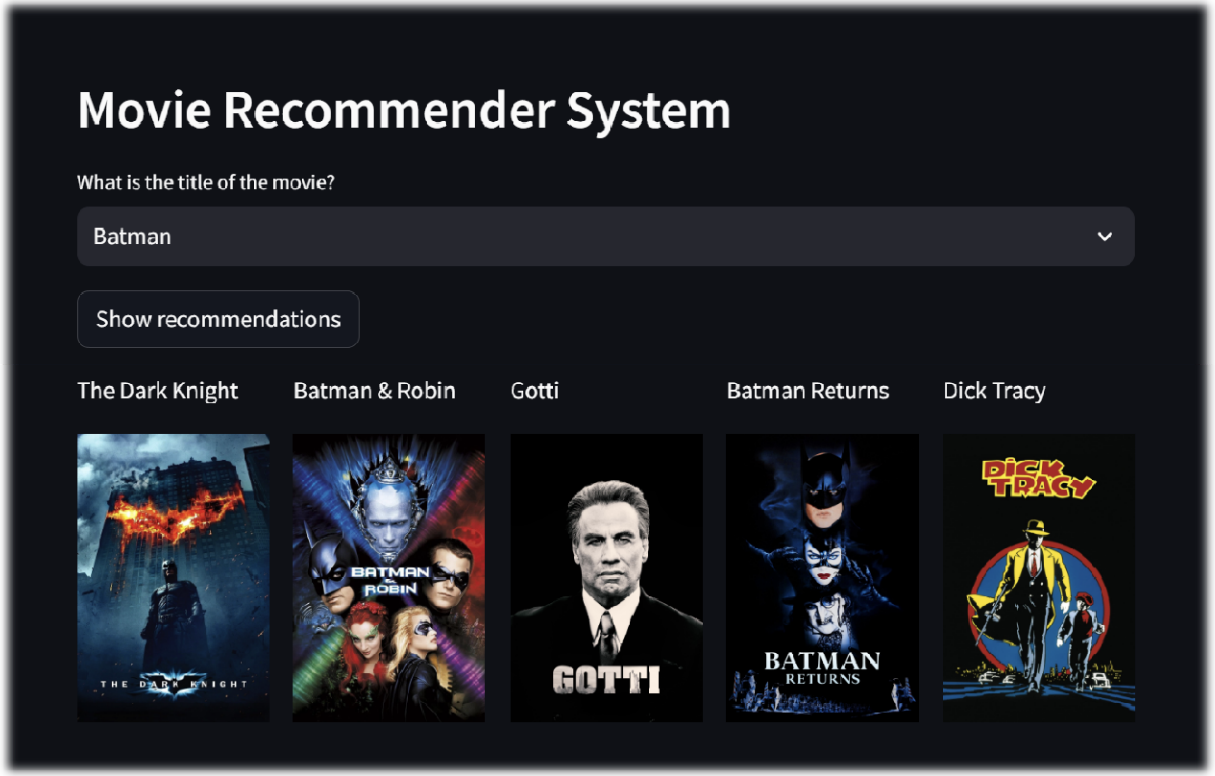
‘Friends’

```
get_recommendations_new('Friends')
```

```
The Andy Griffith Show
Man with a Plan
Episodes
Astronomy Club: The Sketch Show
Adam Ruins Everything
Frasier
Dad's Army
The Twilight Zone (Original Series)
Toast of London
Girlfriends
```

DEPLOYMENT

Deploy the recommender model to a web application using streamlit package. The web page shows the titles, posters and brief descriptions of 5 recommended movies similar to a user's selected movie. The screenshot image shows the recommendations for 'Batman'.



CODE

Python code for deployment

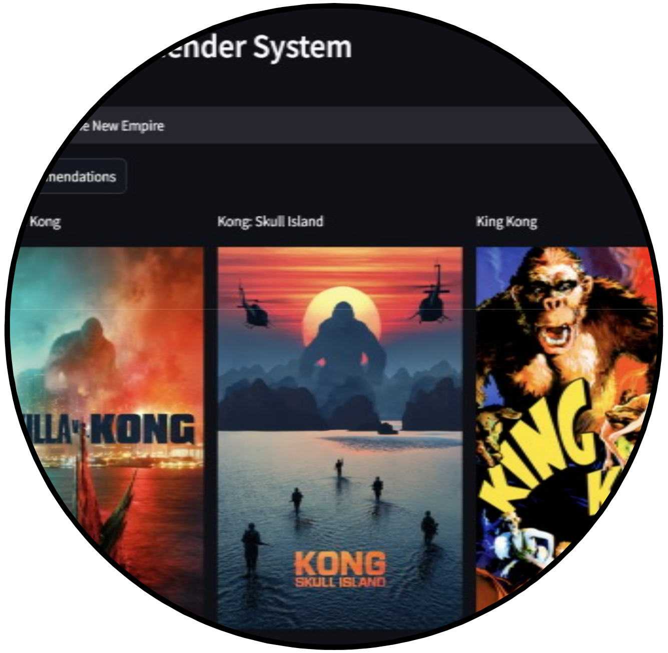
```python
# import libraries
import streamlit as st
import pickle


# import files
movies = pickle.load(open('movies.pkl', 'rb'))
movies_df = pickle.load(open('movies_df.pkl', 'rb'))
similarity = pickle.load(open('similarity.pkl', 'rb'))
movies_list = movies['title'].values

# create title for stream lit page
st.title("""Movie Recommender System
This is a content-based movie recommender system based on features of movies :smile: """)


# create a input box for a movie name
selected_movie = st.selectbox('What is your favorite movie?', movies_list)
```

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

    recommended_movies = []
    recommended_posters = []

    for i in sorted_movie_list:
        poster_path = movies["poster_path"][i[0]]
        recommended_movies.append(movies.iloc[i[0]].title)
        recommended_posters.append("https://image.tmdb.org/t/p/original"+poster_path)

    return recommended_movies, recommended_posters

# details of movies
movie_info = ["title", "genres", "overview", "release_date", "credits", "original_language"]
mv_dataframe = movies_df[movie_info]
```

CODE

Python code for deployment

```python
# create a recommend button with function of displaying recommended movies and movie posters
if st.button('Show recommendations'):
    recommendation, posters = recommend(selected_movie)

    col1, col2, col3, col4, col5 = st.columns(5)
    with col1:
        st.write(recommendation[0])
        st.image(posters[0])
        st.write("Genre: " + mv_dataframe[mv_dataframe['title'] == recommendation[0]].genres.values[0])
        st.write("Release date: " + mv_dataframe[mv_dataframe['title'] == recommendation[0]].release_date.values[0])
        st.write("Language: " + mv_dataframe[mv_dataframe['title'] == recommendation[0]].original_language.values[0])
        st.write("Overview: " + mv_dataframe[mv_dataframe['title'] == recommendation[0]].overview.values[0])

    with col2:
        st.write(recommendation[1])
        st.image(posters[1])
        st.write("Genre: " + mv_dataframe[mv_dataframe['title'] == recommendation[1]].genres.values[0])
        st.write("Release date: " + mv_dataframe[mv_dataframe['title'] == recommendation[1]].release_date.values[0])
        st.write("Language: " + mv_dataframe[mv_dataframe['title'] == recommendation[1]].original_language.values[0])
        st.write("Overview: " + mv_dataframe[mv_dataframe['title'] == recommendation[1]].overview.values[0])

    with col3:
        st.write(recommendation[2])
        st.image(posters[2])
        st.write("Genre: " + mv_dataframe[mv_dataframe['title'] == recommendation[2]].genres.values[0])
        st.write("Release date: " + mv_dataframe[mv_dataframe['title'] == recommendation[2]].release_date.values[0])
        st.write("Language: " + mv_dataframe[mv_dataframe['title'] == recommendation[2]].original_language.values[0])
        st.write("Overview: " + mv_dataframe[mv_dataframe['title'] == recommendation[2]].overview.values[0])

    with col4:
        st.write(recommendation[3])
        st.image(posters[3])
        st.write("Genre: " + mv_dataframe[mv_dataframe['title'] == recommendation[3]].genres.values[0])
        st.write("Release date: " + mv_dataframe[mv_dataframe['title'] == recommendation[3]].release_date.values[0])
        st.write("Language: " + mv_dataframe[mv_dataframe['title'] == recommendation[3]].original_language.values[0])
        st.write("Overview: " + mv_dataframe[mv_dataframe['title'] == recommendation[3]].overview.values[0])

    with col5:
        st.write(recommendation[4])
        st.image(posters[4])
        st.write("Genre: " + mv_dataframe[mv_dataframe['title'] == recommendation[4]].genres.values[0])
        st.write("Release date: " + mv_dataframe[mv_dataframe['title'] == recommendation[4]].release_date.values[0])
        st.write("Language: " + mv_dataframe[mv_dataframe['title'] == recommendation[4]].original_language.values[0])
        st.write("Overview: " + mv_dataframe[mv_dataframe['title'] == recommendation[4]].overview.values[0])
```



LIMITATIONS

Our recommendation system seems to have succeeded in providing similar movies and TV shows to a user's input.

However, it revealed some limitations in diversifying suggestions due to its reliance on inherent features of the items themselves. Consequently, recommendations tended to converge on similar content, potentially limiting exposure to new items.

Additionally, the model's dependency on keywords could lead to compromised recommendations if certain keywords were undervalued by the algorithm.

CONCLUSION

In this project, I built a content-based recommendation system that suggests similar movies and TV shows. First, I went through data cleaning and analysis process and selected key features such as title, genre and description. Then, I converted these text data into vectors using CountVectorizer tool under Scikit-learn library to convert these cleaned text into numerical features. Lastly, I created a recommender function that generates recommendations based on cosine similarity.

Moving forward, I aim to enhance recommendation personalization by integrating users' individual preferences and viewing history. This approach promises to deliver more tailored and enriching suggestions, overcoming the current system's limitations. In conclusion, this project has been an invaluable learning experience, fueling my curiosity to delve deeper into similar projects in the future.

THANK YOU

- Link to Original Dataset: https://www.kaggle.com/datasets/akshaypawar7/millions-of-movies
- Github Link for code, resources and more information: https://github.com/bebe5004/Eunbin-Yoo-s-Portfolio/tree/main/Movie%20Recommender%20System
- Github page for more projects: https://github.com/bebe5004/Eunbin-Yoo-s-Portfolio/tree/main?tab=readme-ov-file#readme