

Movie Recommendation System-SE

CSE1005

The past year and a half locked up in our homes made us long for entertainment indoors and what better way to get entertained than watching whole lotta movies!

With N number of movies to choose from various streaming sites, what movie will we choose at the very moment when we want to watch one? We usually select a movie based on actors, directors, etc or a movie which is similar to the one we like or the one our friend recommends, don't we?

What if... we build our own recommendation system?

In this article, I'll be explaining the procedures that are required to build an end-to-end movie recommendation system from scratch. Here is a summary of all the steps:

- Getting dataset from Kaggle TMDB dataset which is given by IMDB in 2010.
- Building a simple but effective **Content-based recommendation system**
- Designing a **front-end web app** for our model (using streamlit) to see movie recommendations
- **Deploy** containerized app using streamlit as web application

Data Extraction

Data Source Transfer Summary

We (Kaggle) have removed the original version of this dataset per a DMCA takedown request from IMDB. In order to minimize the impact, we're replacing it with a similar set of films and data fields from [The Movie Database \(TMDb\)](#) in accordance with their terms of use. The bad news is that kernels built on the old dataset will most likely no longer work.

The good news is that:

- You can port your existing kernels over with a bit of editing. This kernel offers functions and examples for doing so. You can also find a general introduction to the new format [here](#).
- The new dataset contains full credits for both the cast and the crew, rather than just the first three actors.
- Actor and actresses are now listed in the order they appear in the credits. It's unclear what ordering the original dataset used; for the movies I spot checked it didn't line up with either the credits order or IMDB's stars order.
- The revenues appear to be more current. For example, IMDB's figures for Avatar seem to be from 2010 and understate the film's global revenues by over \$2 billion.

- Some of the movies that we weren't able to port over (a couple of hundred) were just bad entries. For example, this IMDB entry has basically no accurate information at all. It lists Star Wars Episode VII as a documentary.


Detail

Compact

Column

About this file

movies and credits card datasets

🔍 movie_id	📄 title	📄 cast	📄 crew
 <div>5459k</div>	<div>4800</div> <div>unique values</div>	<div>4761</div> <div>unique values</div>	<div>4776</div> <div>unique values</div>
19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "credit_id": "5602a8a7c3a3685532001c9a", "gender": 2, "...	[{"credit_id": "52fe48009251416c750aca23", "department": "Editing", "gender": 0, "id": 1721, "job": ...
285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Sparrow", "credit_id": "52fe4232c3a36847f800b50d", "gende...	[{"credit_id": "52fe4232c3a36847f800b579", "department": "Camera", "gender": 2, "id": 120, "job": "D...
206647	Spectre	[{"cast_id": 1, "character": "James Bond", "credit_id": "52fe4d22c3a368484e1...	[{"credit_id": "54805967c3a36829b5002c41", "department":

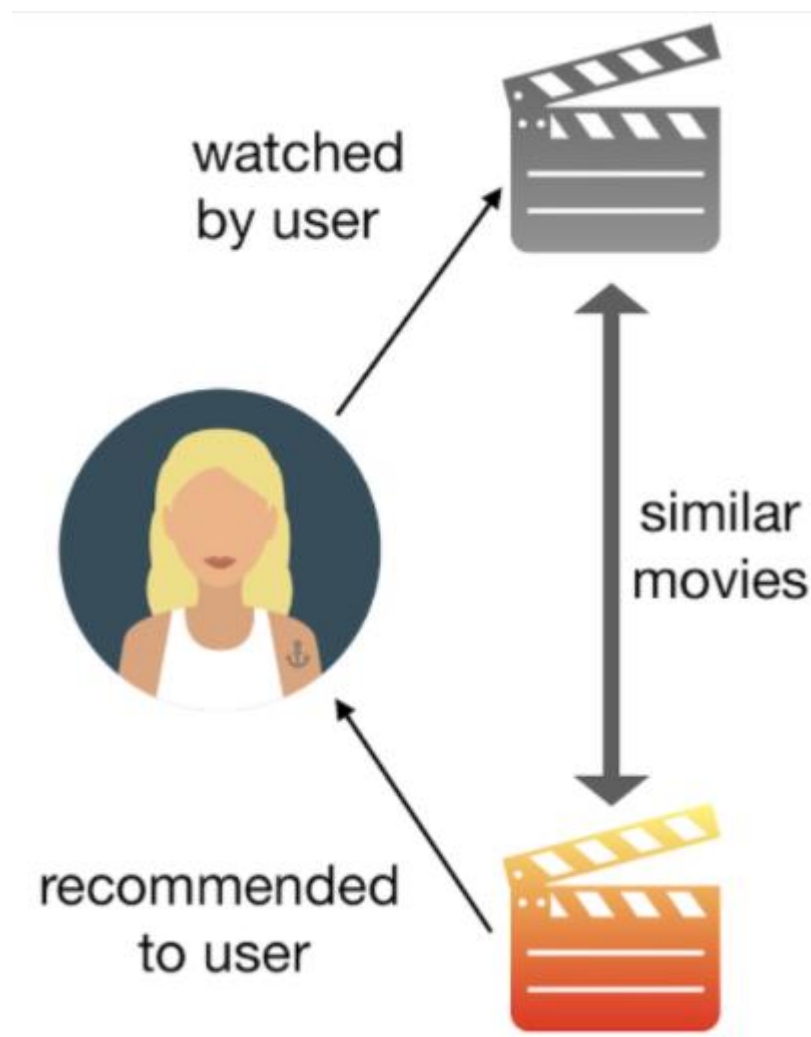
Here is an example of the data that was scraped from the site,

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

The dataset that is defined above will be used for modelling.

Building a content-based recommendation system

This method uses attributes of the content to recommend similar content. It doesn't have a cold-start problem because it works through attributes or tags of the content, such as actors, genres, or directors so that new movies can be recommended right away. For example, User X likes LOTR, Star Trek movies (action, adventure) then our model will recommend movies of similar content like Star Wars, etc to User X.



Captions from offerzen.com

Modeling

Import all the dependencies and load the scraped data.

▼ Import libraries

```
import os
import ast
import pickle
import numpy as np
import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

▼ Load data

Concatenate title,director,genre,description in to a single feature.

Merge credits data with movies on title column																		
[] movies = movies.merge(credits,on='title')																		
movies.head()																		
	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_companies	production_countries	release_date	revenue	runtime	spoken_languages	status	tagline	title
0	237000000	[[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}]]	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_1": "US", "name": "United States o...	2009-12-10	2787965087	162.0	[{"iso_639_1": "en", "name": "English"}, {"iso... "name": "English"}]]	Released	Enter the World of Pandora.	Avatar
1	300000000	[[{"id": 12, "name": "Adventure"}, {"id": 14, "name": "Action"}]]	http://disney.go.com/disneypictures/pirates/	285	[[{"id": 270, "name": "ocean"}, {"id": 726, "name": "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	[{"name": "Walt Disney Pictures", "id": 2}, {"name": "United States o...	[{"iso_3166_1": "US", "name": "United States o..."}, {"iso_3166_1": "GB", "name": "United Kingdom"}]]	2007-05-19	961000000	169.0	[{"iso_639_1": "en", "name": "English"}, {"iso_639_1": "fr", "name": "Fran\u00e7ais"}]]	Released	At the end of the world, the adventure begins.	Pirates of the Caribbean: At World's End
2	245000000	[[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}]]	http://www.sonymovies.com/movies/spectre/	206647	[[{"id": 470, "name": "spy"}, {"id": 818, "name": "James Bond"}]]	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	[{"name": "Columbia Pictures", "id": 5}, {"name": "United States o..."}, {"name": "United Kingdom"}]]	[{"iso_3166_1": "US", "name": "United States o..."}, {"iso_3166_1": "GB", "name": "United Kingdom"}]]	2015-10-26	880674609	148.0	[{"iso_639_1": "en", "name": "English"}, {"iso_639_1": "fr", "name": "Fran\u00e7ais"}]]	Released	A Plan No One Escapes	Spectre
3	250000000	[[{"id": 28, "name": "Action"}, {"id": 80, "name": "Adventure"}]]	http://www.thedarkknighttrises.com/	49026	[[{"id": 849, "name": "dc comics"}, {"id": 853, "name": "superhero"}]]	en	The Dark Knight Rises	Following the death of District Attorney Harvey...	112.312950	[{"name": "Legendary Pictures", "id": 923}, {"name": "United States o..."}, {"name": "United Kingdom"}]]	[{"iso_3166_1": "US", "name": "United States o..."}, {"iso_3166_1": "GB", "name": "United Kingdom"}]]	2012-07-16	1084939099	165.0	[{"iso_639_1": "en", "name": "English"}, {"iso_639_1": "fr", "name": "Fran\u00e7ais"}]]	Released	The Legend Ends	The Dark Knight Rises
4	260000000	[[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}]]	http://movies.disney.com/john-carter	49529	[[{"id": 818, "name": "based on novel"}, {"id": ...	en	John Carter	John Carter is a war-weary, former military ca...	43.926995	[{"name": "Walt Disney Pictures", "id": 2}], [{"name": "United States o..."}, {"name": "United Kingdom"}]]	[{"iso_3166_1": "US", "name": "United States o..."}, {"iso_3166_1": "GB", "name": "United Kingdom"}]]	2012-03-07	284139100	132.0	[{"iso_639_1": "en", "name": "English"}, {"iso_639_1": "fr", "name": "Fran\u00e7ais"}]]	Released	Lost in our world, found in another.	John Carter

This is done because our recommendation system works not only by recommending movies of a similar genre but also based on a combination of cast, director, and plot of the movie.

The next step is to vectorize this feature to calculate similarity.

TF-IDF Vectorizer

The **TF (term frequency)** is the number of times a given term occurs in a document.

$TF(t) = (\text{Number of times term } t \text{ appears in a document}) / (\text{Total number of terms in the document})$

The **IDF (inverse document frequency)** is the measure of how significant that term is in the whole corpus.

$\text{IDF}(t) = \log \left(\frac{\text{Total number of documents}}{\text{Number of documents containing the term } t \text{ in it}} \right)$.

$$w_{i,j} = t f_{i,j} \times \log \left(\frac{N}{df_i} \right)$$

TF*IDF

The reason behind choosing TF-IDF is because it gives more weightage to terms that are not frequent and less weightage to the frequent terms which is important for our model.

```
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(data['important_features'])
tfidf_matrix.shape

# generating the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

▼ Build count vectorizer model

```
[ ] cv = CountVectorizer(max_features=5000, stop_words='english')
```

```
[ ] vector = cv.fit_transform(new['tags']).toarray()
```

```
[ ] vector.shape
```

```
(4806, 5000)
```

Cosine Similarity

The next step is to calculate the similarity between movies. We can use cosine similarity to calculate the distance between two vectors.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

▼ Calculate cosine similarity

```
[ ] similarity = cosine_similarity(vector)
```

▶ similarity

```
array([[1.          , 0.08964215, 0.06071767, ..., 0.02519763, 0.0277885 ,
        0.          ],
       [0.08964215, 1.          , 0.06350006, ..., 0.02635231, 0.          ,
        0.          ],
       [0.06071767, 0.06350006, 1.          , ..., 0.02677398, 0.          ,
        0.          ],
       ...,
       [0.02519763, 0.02635231, 0.02677398, ..., 1.          , 0.07352146,
        0.04774099],
       [0.0277885 , 0.          , 0.          , ..., 0.07352146, 1.          ,
        0.05264981],
       [0.          , 0.          , 0.          , ..., 0.04774099, 0.05264981,
        1.          ]])
```

```
[ ] new[new['title'] == 'The Lego Movie'].index[0]
```

744

We need to write a logic that takes a movie title as input and returns the top 5 similar movies based on cosine similarity.

For this, we can create a function that calculates the similarity score of our input with other movies in the corpus then sort the scores out in descending order to get the top 5 movies with the highest similarity scores. Here index 0 is discarded in sim_scores variable so that the function does not return the same movie that's been entered in the input.

▼ Create recommender

```
▶ def recommend(movie):
    index = new[new['title'] == movie].index[0]
    distances = sorted(list(enumerate(similarity[index])), reverse=True, key = lambda x: x[1])
    for i in distances[1:6]:
        print(new.iloc[i[0]].title)
```

Test our recommendation engine

```
▶ recommend('Gandhi')
```

```
↳ Gandhi, My Father  
The Wind That Shakes the Barley  
A Passage to India  
Guiana 1838  
Ramanujan
```

```
✓ s ▶ recommend('Avatar')
```

```
↳ Titan A.E.  
Small Soldiers  
Ender's Game  
Aliens vs Predator: Requiem  
Independence Day
```

Pickle is a serialization/deserialization module that is already built-in in Python: using it we can save an arbitrary Python object (with a few exceptions) to a file. Once we have a file, we can load the model from there in a different process.

▼ Save the movie list and similarity matrix as pickle

```
[ ] pickle.dump(new, open('movie_list.pkl', 'wb'))  
    pickle.dump(similarity, open('similarity.pkl', 'wb'))
```

```
[ ] pickle.dump(new.to_dict(), open('movie_dict.pkl', 'wb'))
```

```
▶ new['title'].values
```

```
↳ array(['Avatar', 'Pirates of the Caribbean: At World's End', 'Spectre',  
        ..., 'Signed, Sealed, Delivered', 'Shanghai Calling',  
        'My Date with Drew'], dtype=object)
```

```
pickle.dump(new, open('movie_list.pkl', 'wb'))  
pickle.dump(cosine_sim, open('similarity.pkl', 'wb'))
```

,

Designing a front end

Now that we have completed building a model, we can start creating a web application.

Our model's frontend uses Streamlit. Streamlit is an open-source Python library that makes it easy to build beautiful custom web apps for machine learning and data science.

```
pip install streamlit
```

There are already many good tutorials on streamlit like this one. So, I don't want to go into detail on all my processes. In general, my goal was to build a frontend that gets user input and recommends a list of similar movies.

The following code is app file

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

def fetch_poster(movie_id):
    response = requests.get(
        'https://api.themoviedb.org/3/movie/{}?api_key=cf642fdb4c5208a52a64475f29673ebd&language=en-US'.format(
            movie_id))
    data = response.json()
    return "https://image.tmdb.org/t/p/w500/" + data['poster_path']

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]
    recommended_movies = []
    recommended_movies_posters = []
    for i in movies_list:
        movie_id = movies.iloc[i[0]].movie_id

        recommended_movies.append(movies.iloc[i[0]].title)
        # fetch poster from API
        recommended_movies_posters.append(fetch_poster(movie_id))
    return recommended_movies , recommended_movies_posters

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

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

st.title('Movie Recommender System')

selected_movie_name = st.selectbox(
    'Select or type movies below',
    movies['title'].values)

if st.button('Recommend Movies'):
    names, posters = recommend(selected_movie_name)
    col1, col2, col3, col4, col5 = st.columns(5)
    with col1:
        st.text(names[0])
        st.image(posters[0])
    with col2:
        st.text(names[1])
        st.image(posters[1])
    with col3:
```



```

        st.text(names[2])
        st.image(posters[2])
    with col4:
        st.text(names[3])
        st.image(posters[3])
    with col5:
        st.text(names[4])
        st.image(posters[4])

```

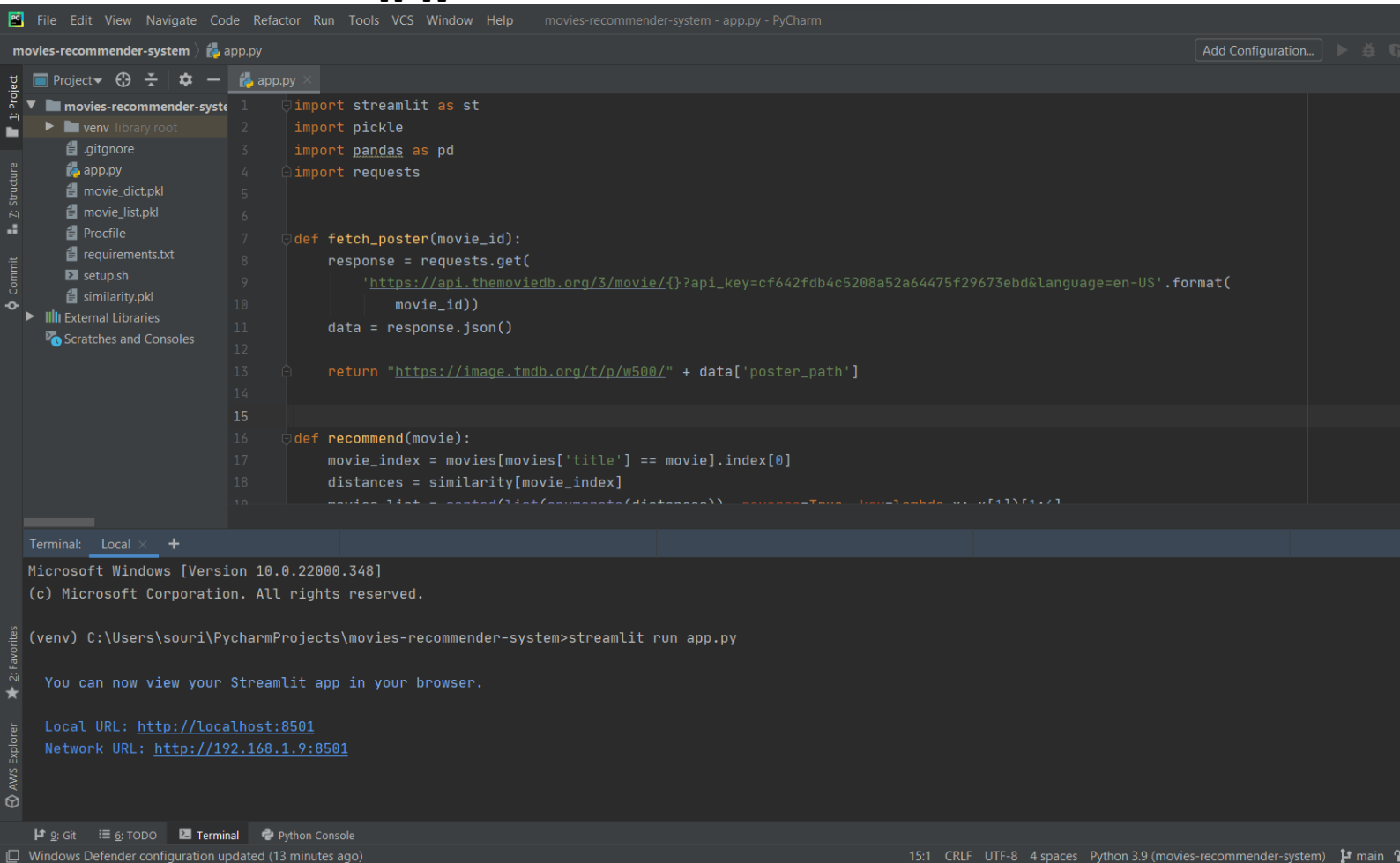
Execution in local environment using PyCharm

Once you have built the frontend you can test it out locally in the command prompt.

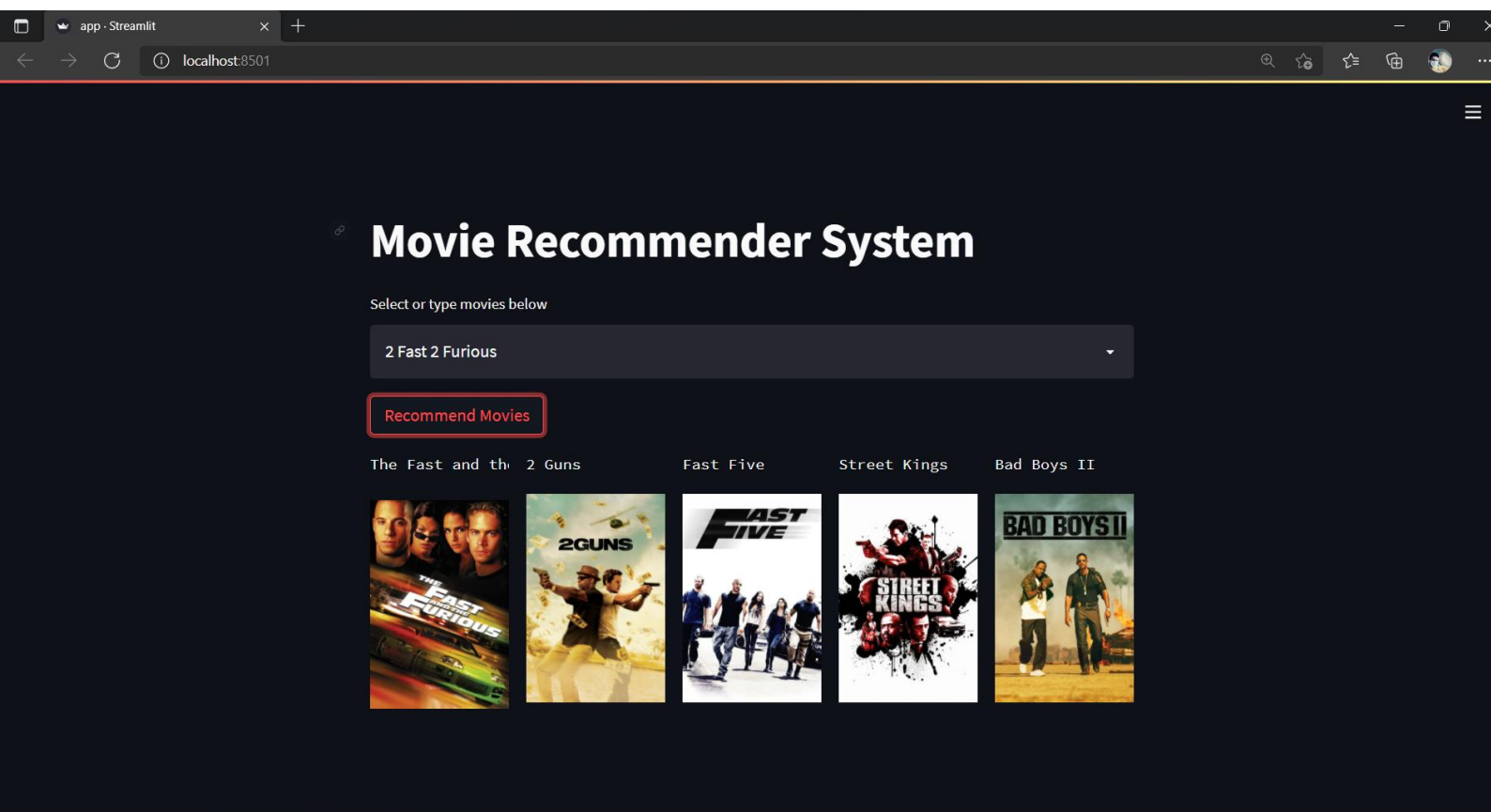
Make sure you are present in the directory that contains all the files, then type in the following command in the cmd,

```
pip install streamlit
```

```
streamlit run app.py
```



In the above screenshot we can see that the execution is done in local virtual environment using PyCharm where I already installed streamlit and running the app using the above command so that a direct local host link will be available. Where our app output is shown below,



Group Members:

- 1.Souri Yaswanth Krishna -19bci7070
- 2.Abdul Samad -19bcn7159
- 3.Narendranath Budda -19bc37774
- 4.Netra -19bce7484
- 5.Ashok Vardhan Reddy -19bcn7013

References:

<https://www.kaggle.com/tmdb/tmdb-movie-metadata>

<https://medium.com/geekculture/end-to-end-movie-recommendation-system-49b29a8b57ac>