**Movie Recommender Systems**

**Introduction:**

A Movie Recommender System is a machine learning project that predicts and suggests movies to users based on their preferences. It leverages algorithms like collaborative filtering, content-based filtering, or hybrid methods to recommend movies. Collaborative filtering uses user interaction data (ratings or reviews) to find similar users or movies, while content-based filtering suggests films with similar attributes, such as genre or cast. In Python, libraries like Pandas, NumPy, and scikit-learn are commonly used to preprocess data and build recommendation models. Such systems enhance user experience by providing personalized movie suggestions.

**About Dataset:**

These files contain metadata for all 45,000 movies listed in the Full Movie Lens Dataset. The dataset consists of movies released on or before July 2017. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages.

This dataset also has files containing 26 million ratings from 270,000 users for all 45,000 movies. Ratings are on a scale of 1-5 and have been obtained from the official Group Lens website. This dataset is an ensemble of data collected from TMDB and Group Lens.  
The Movie Details, Credits and Keywords have been collected from the TMDB Open API. This product uses the TMDB API but is not endorsed or certified by TMDB. Their API also provides access to data on many additional movies, actors and actresses, crew members, and TV shows.

**Content**

This dataset consists of the following files:

**movies\_metadata.csv:** The main Movies Metadata file. Contains information on 45,000 movies featured in the Full Movie Lens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.

**keywords.csv:** Contains the movie plot keywords for our Movie Lens movies. Available in the form of a stringified JSON Object.

**credits.csv:** Consists of Cast and Crew Information for all our movies. Available in the form of a stringified JSON Object.

**links.csv:** The file that contains the TMDB and IMDB IDs of all the movies featured in the Full Movie Lens dataset.

**links\_small.csv:** Contains the TMDB and IMDB IDs of a small subset of 9,000 movies of the Full Dataset.

**ratings\_small.csv:** The subset of 100,000 ratings from 700 users on 9,000 movies.

The Full Movie Lens Dataset consisting of 26 million ratings and 750,000 tag applications from 270,000 users on all the 45,000 movies in this dataset can be accessed.

**How to Prepare and Process the Movies Dataset**

The first step of any machine learning project is collecting and preparing the data. As our goal is to build a movie recommendation engine, I have chosen a movie credits and keyword dataset. The dataset is publicly available for free on [Kaggle](https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset/).

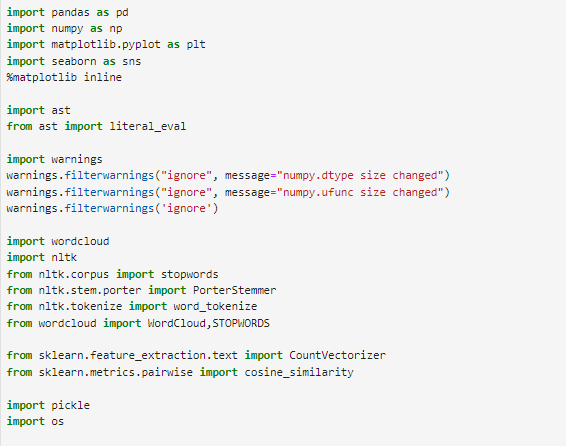
The dataset has two main files in the format of CSV:

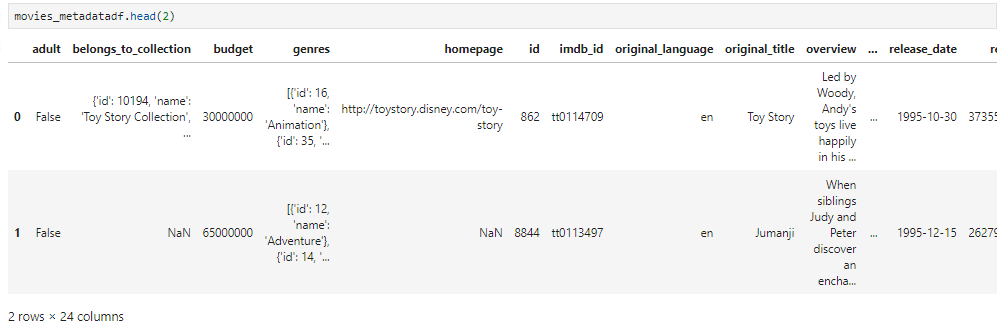
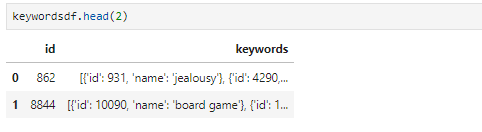
**movies\_metadata.csv:** The main Movies Metadata file. Contains information on 45,000 movies featured in the Full Movie Lens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.

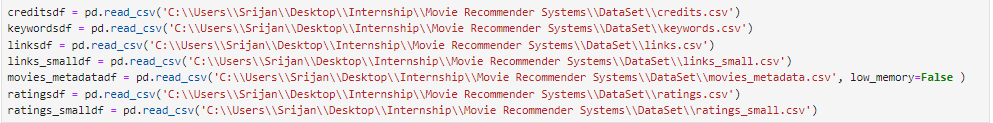
**keywords.csv:** Contains the movie plot keywords for our Movie Lens movies. Available in the form of a stringified JSON Object.

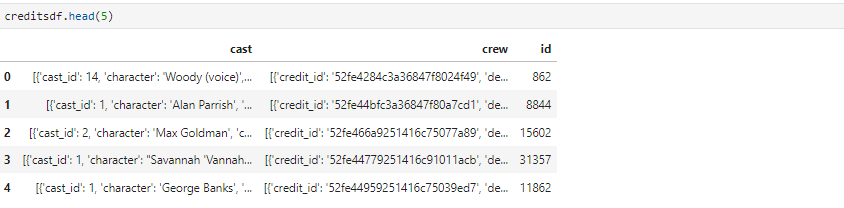
**credits.csv:** Consists of Cast and Crew Information for all our movies. Available in the form of a stringified JSON Object.

Let us first import the Python packages needed to read the CSV files.



Next, read the whole file into Panda’s data frames and look at the columns.





Join the movies and credited through the ID and ID has the unique identification number for every movie. Now create the new data frame.

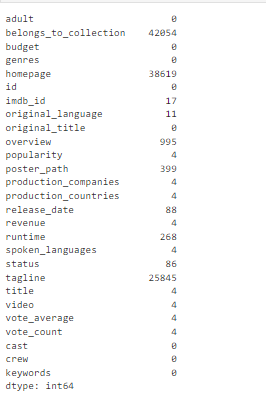
Now the this created data frame is join through the ID with the Keyword data frame and ID has the unique identification and created a new data frame.

Now copy the new created data frame is in the new data frame for comfortable use.

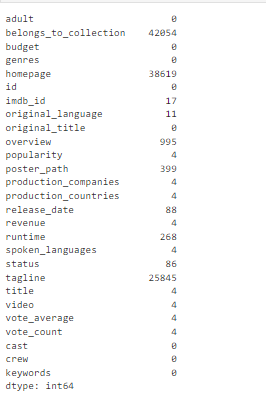


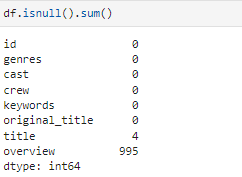
New data frame to check all EDA process to clean the hole dataset which has been created.



Now check the shape of the data frame

Now select the columns and which is columns are useful to my work. In this data frame these columns are present.

Now check the shape of the data frame

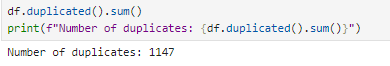
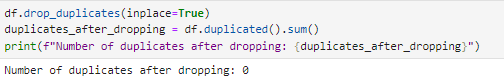
Check the null values

Drop the NA values in the data frame

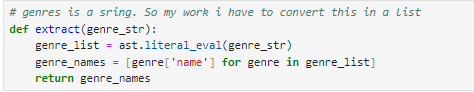
Now check the shape of the data frame



Now Check the duplicate value and drop the duplicate value in the data frame

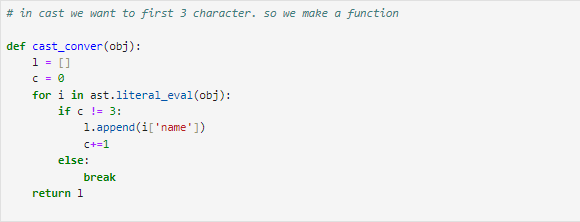


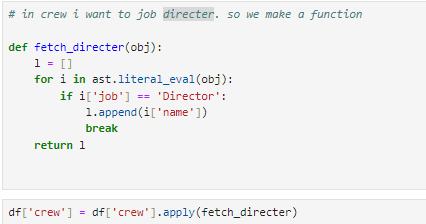
Now all EDA process is complete then we have to some columns are modification as per my work.

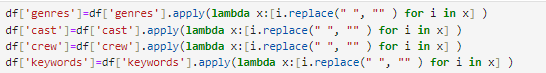
In the Genres column modify and in modification show only name. In Genres column are show id and name I want to only name. Same as the keywords column.



In Cast column we have want to first 3 characters. So, we make the function and get the first 3 characters.

In Crew column we have want to job director. So we make the function and get the director.



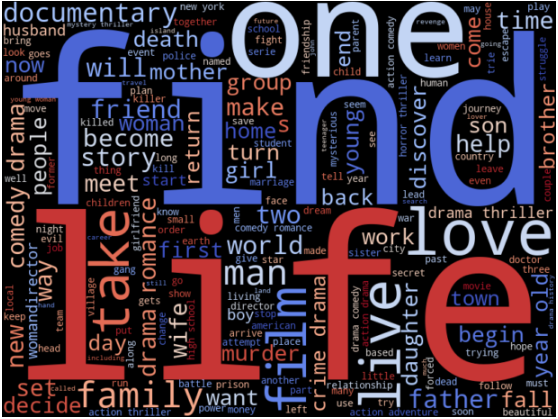
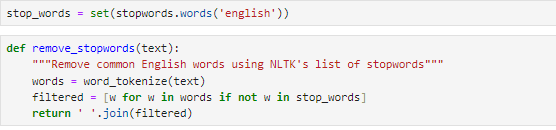
Now to replace the space in no space.

Now make a Tag column to add or Concatenation the Overview, Genres, Keyword, Cast and Crew.

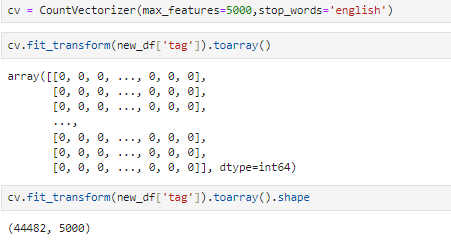
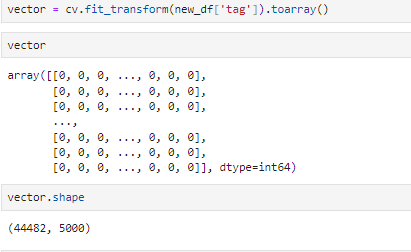


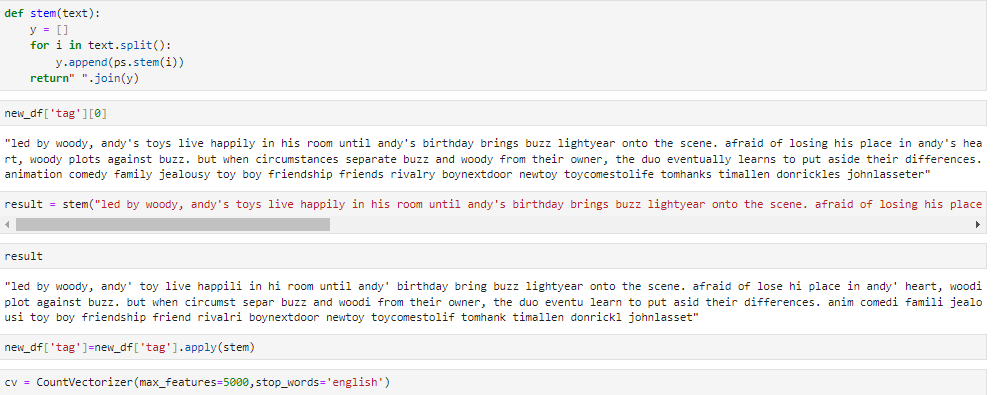
Make a new data frame New\_DF with some of the column.

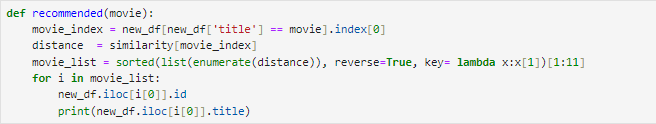
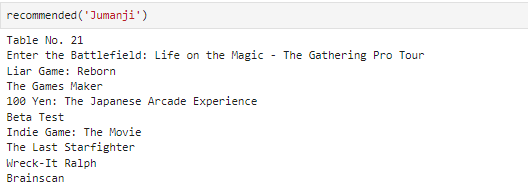
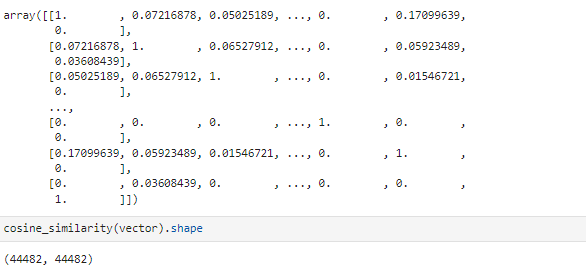
Now download the Nltk library and remove the stop words in the Tag column and generate the word cloud.

Now we must create the model and this model to convert the .pkl file to use in the flask.

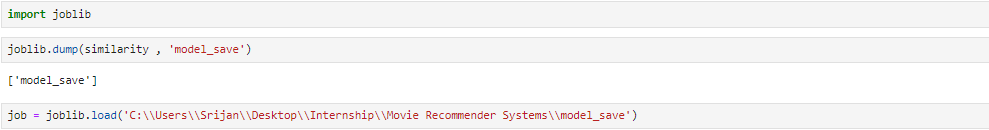




Now Create the function to check my model is work or not

Now my model is ready then we create the one is .pkl and other one is . joblib file one is movie list and other one is recommended.





Now all ready now ready to deploy in the flask.

**from** flask **import** Flask, render\_template, request, jsonify

**import** pickle

**import** requests

**import** pandas **as** pd

**import** numpy **as** np

**import** joblib

app = Flask(\_\_name\_\_)

**with** **open**('C:\\Users\\Srijan\\Desktop\\Internship\\Movie Recommender Systems\\Website\\PickleFile\\movies\_name.pkl', 'rb') **as** file:

    movie\_dict = pickle.load(file)

movies = pd.DataFrame(movie\_dict)

data = joblib.load(

    'C:\\Users\\Srijan\\Desktop\\Internship\\Movie Recommender Systems\\Website\\PickleFile\\model\_save')

**def** **fetch\_poster**(movieId):

    url = "https://api.themoviedb.org/3/movie/{}?api\_key=ee46d7f2824234db2ff7c05c6c4f665d&language=en-US".format(

        movieId)

    response = requests.get(url)

    data = response.json()

    poster\_path = data["poster\_path"]

**if** poster\_path:

        full\_path = "https://image.tmdb.org/t/p/original/" + poster\_path

**return** full\_path

**else**:

**return** **None**

@app.route('/get\_genres', methods=['GET'])

**def** **get\_genres**():

    url = "https://api.themoviedb.org/3/genre/movie/list?language=en&api\_key=ee46d7f2824234db2ff7c05c6c4f665d"

    response = requests.get(url)

**if** response.status\_code == 200:

        data = response.json()

        genres = [{'id': genre['id'], 'name': genre['name']}

**for** genre **in** data.get('genres', [])]

**return** jsonify(genres)

**else**:

**return** jsonify({"error": "Failed to fetch genres"}), response.status\_code

@app.route('/get\_movies\_by\_genre', methods=['GET'])

**def** **get\_movies\_by\_genre**():

    genre\_id = request.args.get('genre\_id')

    page = **int**(request.args.get('page', 1))

    url = **f**"https://api.themoviedb.org/3/discover/movie?api\_key=ee46d7f2824234db2ff7c05c6c4f665d&with\_genres={

        genre\_id}&language=en-US&page={page}"

    response = requests.get(url)

**if** response.status\_code != 200:

**return** jsonify({'error': 'Failed to fetch movies', 'status\_code': response.status\_code}), response.status\_code

    data = response.json()

    movies = data.get('results', [])

    total\_pages = data.get('total\_pages', 1)

    movie\_list = []

**for** movie **in** movies:

        movie\_list.append({

            'title': movie.get('title'),

            'poster\_path': **f**"https://image.tmdb.org/t/p/original/{movie.get('poster\_path', '')}"

        })

**return** jsonify({'movies': movie\_list, 'total\_pages': total\_pages})

movie\_titles = movies['title'].tolist()

@app.route('/')

**def** **home**():

**return** render\_template('index.html', movie\_titles=movie\_titles)

@app.route('/get\_movie\_titles', methods=['GET'])

**def** **get\_movie\_titles**():

    movie\_titles = movies['title'].tolist()

    page = **int**(request.args.get('page', 1))

    limit = **int**(request.args.get('limit', 10))

    start = (page - 1) \* limit

    end = start + limit

    titles = movie\_titles[start:end]

**return** jsonify(titles)

**def** **recommended**(movie):

**if** **not** **isinstance**(data, np.ndarray):

**print**("Error: Data is not a NumPy array.")

**return** [], []

**try**:

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

        distance = data[movie\_index]

**if** **not** **isinstance**(distance, (**list**, np.ndarray)):

**raise** **TypeError**("Expected distance to be a list or numpy array.")

        movie\_list = **sorted**(**list**(**enumerate**(distance)),

                            reverse=**True**, key=**lambda** x: x[1])[1:11]

        recommended\_movies = []

**for** i **in** movie\_list:

            movie\_id = movies.iloc[i[0]]['id']

            title = movies.iloc[i[0]]['title']

            poster\_path = fetch\_poster(movie\_id)

            recommended\_movies.append({

                'id': movie\_id,

                'title': title,

                'poster\_path': poster\_path

            })

**print**("recommended\_movies==>", recommended\_movies)

**return** recommended\_movies

**except** **Exception** **as** e:

**print**(**f**"An error occurred: {e}")

**return** [], []

@app.route('/recommend', methods=['GET'])

**def** **recommend1**():

**print**("similarity==>", data)

    movie\_title = request.args.get('movie\_title')

**if** **not** movie\_title:

**return** "Movie title not provided", 400

**print**("movie\_title===>", movie\_title)

**if** data **is** **None**:

**return** "Error: Recommendation system not available due to pickle file corruption", 500

    recommended\_movies = recommended(movie\_title)

**print**("Recommended==>", recommended\_movies)

**return** render\_template('recommended.html',

                           selected\_movie=movie\_title,

                           recommended\_movies=recommended\_movies)

**if** \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=**True**, port=4000)

