Machine Learning

Project Report

Pouria Malek Khayat (99222097) - Farimah Rashidi (99222040)

Abstract

Introduction

In this project, we are required to develop a movie recommendation system for the Movie dataset. In order to create personalized movie suggestions, this project analyzes this large movie collection and incorporates user preferences. So, we are not just using clustering methods.

The model is built using Python libraries and it uses Streamlit in order to create a user-friendly user interface. It is deployed on Hugging face to use and test.

First, we clean the data and analyze it to decide which features to use and see how we can build an optimal model.

Then, we can go on developing a UI for it for the best experience.

Data Analysis - Movies

Features

The data contains the following features:

- Adult: true if the movie is suited for adult people
- Budget: amount of budget spent for the production
- Genres: a json format that includes all movie's genres
- Homepage: a link to the movie's official site
- Id: a unique id for the movie
- Imdb_id: a unique id for the movie in imdb
- Original_language: the initial language spoken in the movie.
- Original_title: the movie's title
- Overview: the movie summary
- Popularity: how popular the movie became after its release
- Poster_path: a link to the movie's poster
- Production_companies: the companies involved in the production.
- Production_countries: the countries involved in the production.
- Release_date: the date that movies was published in.
- Revenue: the profit that the movie made
- Runtime: the movie's length
- Spoken_languages: all languages spoken throughout the movie in json format.
- Status: the movie's current status
- Tagline: a short sentence about the movie
- Title: the movie's title
- Video:
- Vote_average: the movie's average score
- Vote_count: number of votes

Now, we need to decide which features are useful to us and then go on with cleaning them since they have missing values.

Dropping Features

There are some features that are not useful for training. First, we must identify them and then drop them.

- imdb_id: both ratings.csv and keywords.csv have an id column to match with movies metadata dataset, so we do not need this column.
- homepage: this column is useless for movie recommendation analysis and there will be no analysis depending on the homepage of the movie.
- original_title: this column can be removed because the title column is also included and the original_title column has non-ASCII characters.
- spoken_languages: original_language is included, so we do not need this column.
- video & poster_path: we won't do any image or video related processing. So, we can drop this column.

Removing rows with missing values:

In this step we remove rows from the movies DataFrame that contain all missing values (NaN) across all columns.

There are 6 rows Out of 45449 rows, with no title so we drop them.

Object to numeric:

The types of id, popularity and budget is object, although they had to be numeric. Errors will be handled with coerce option; thus, invalid parsing will be set as NaN. Also converting release_date to datetime instead of object and extracting the year data may be helpful.

Null values in belongs_to_collection:

Due to the excessive number of null values in the belongs_to_collection column, we can transform the data to 0 and 1 instead of the collection name, with 0 denoting not belonging and 1 denoting belonging.

Removing adult column

Only 9 True values are present in the adult column, which means that information will not be useful to us, and that column has also been removed.

Filling null values of status column:

Less than 100 entries in the status column are null, thus it could be a good idea to fill these with the most common data.

Filling null values of runtime column:

Less than 100 entries in the runtime column are null. So, we can fill NaN values with the mean.

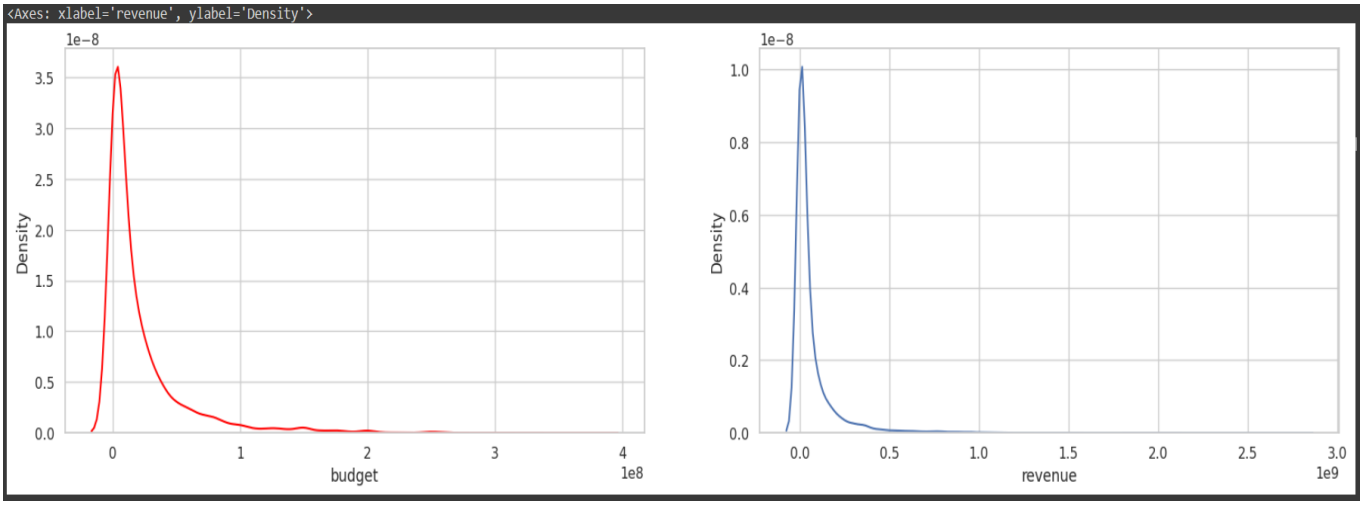
Handling Json Features

As mentioned earlier, there are some features that are in Json format, such as such as production_companies, production_countries, and genres. And we must convert them to lists in order to make use of them in the training process. So, we implement a function which converts json list to list of inputs. Then we use this function to genres, production_countries and production_companies columns.

Convert 0 to NaN in budget and revenue entries:

Many budget and revenue entries are zero. To see how many entries are genuinely available, NaN makes more sense than 0, so we replace 0 with NaN.

checking the distribution of revenue and budget:



Scaling budget and revenue rows:

Some rows that have a budget and revenue value are not scaled, so we decide to implement a scaling function.

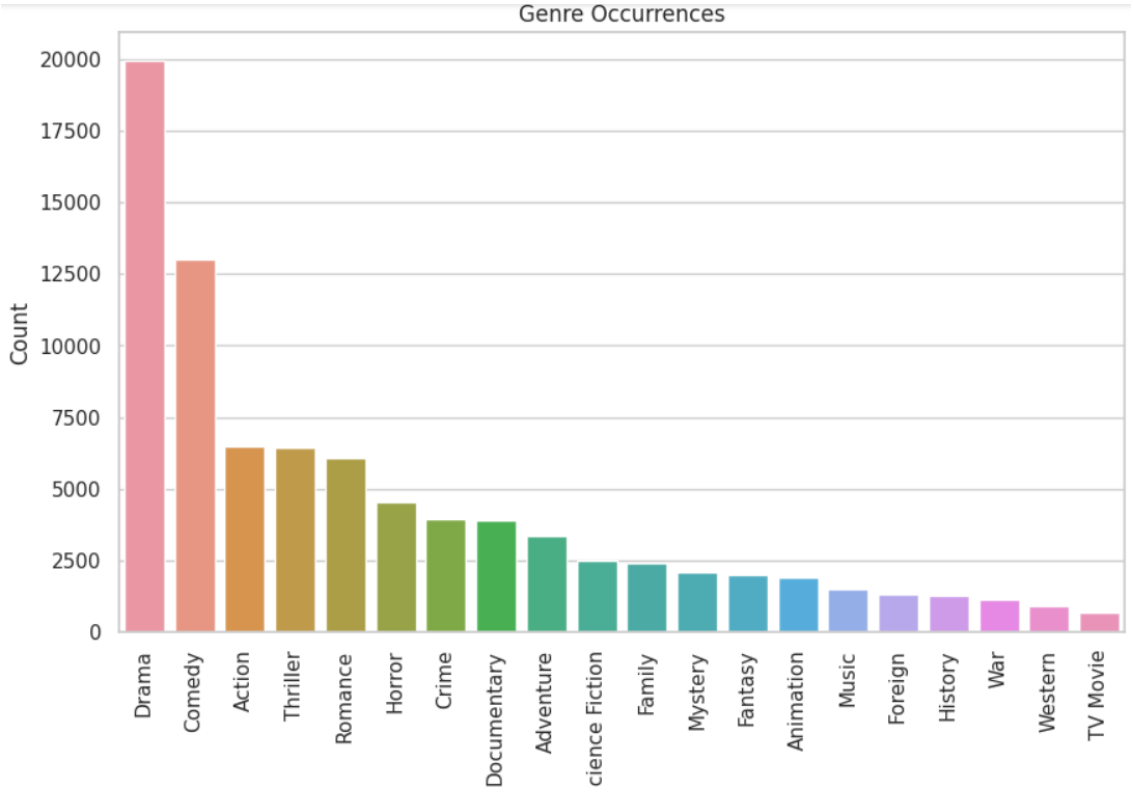
- If the value is less than 100, it is multiplied by 1,000,000 (1 million).
- If the value is between 100 and 1000 (inclusive), it is multiplied by 10,000.
- If the value is between 1000 and 10000 (inclusive), it is multiplied by 100.
- For values greater than or equal to 10000, no scaling is applied, and the original value is returned.

List counter function:

For genres and country, we can fill the values with the most appearing entry or mean. The function analyzes the most occurring values for each column in lists. It iterates over each cell in the column and counts the occurrence of each element in the lists. The function returns a list of the most frequently appearing elements, sorted in descending order of their occurrence counts. The function also provides an option to limit the number of returned elements by specifying a 'limiter' parameter. Additionally, the function calculates the total count of selected elements and the count of remaining elements that exceed the

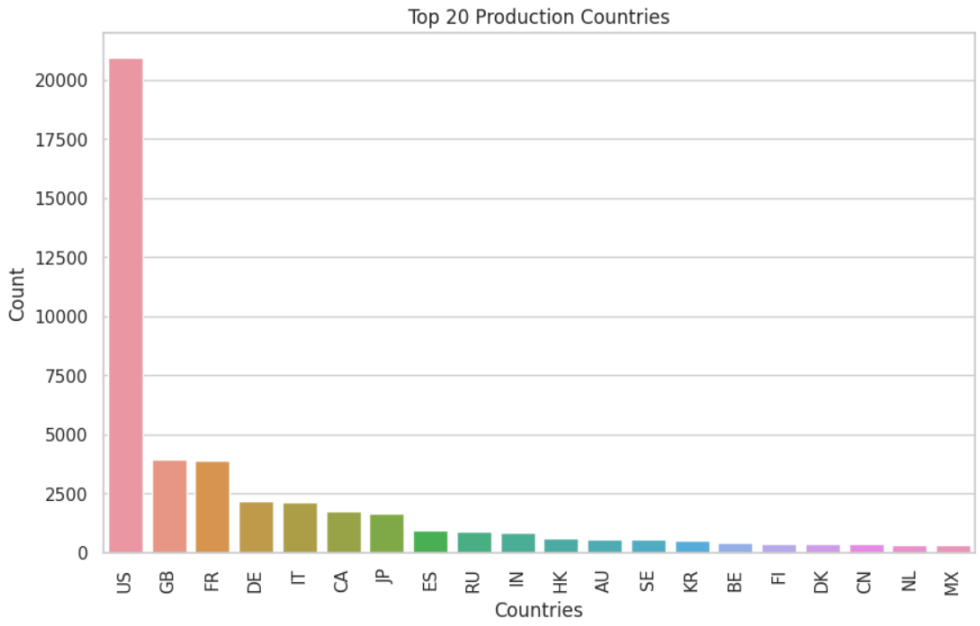
specified limit. The purpose of this function is to analyze the distribution of values in a column and identify the most common elements, which can be useful for filling missing values with the most frequent or average values in the respective column.

Genre Occurrences (using list counter function):



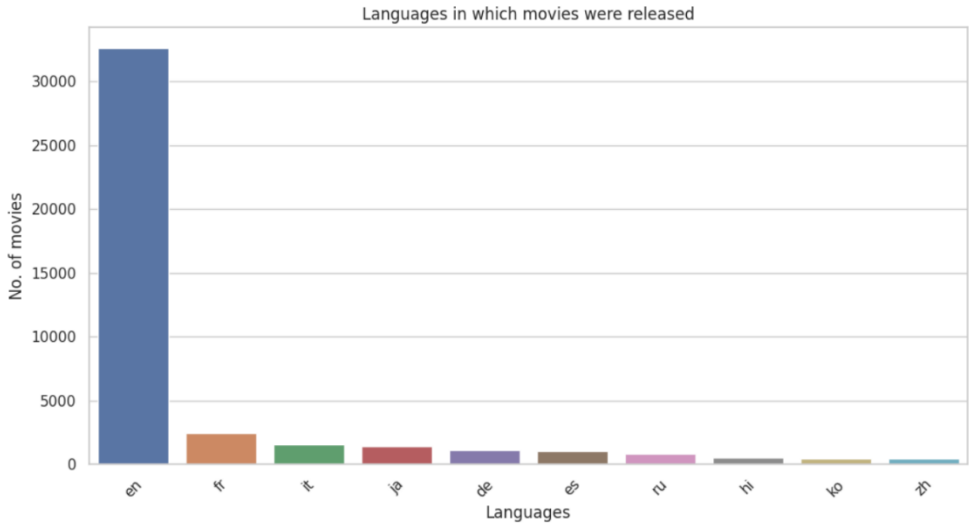
Drama is liked in 1st place.

Top 20 production Countries:



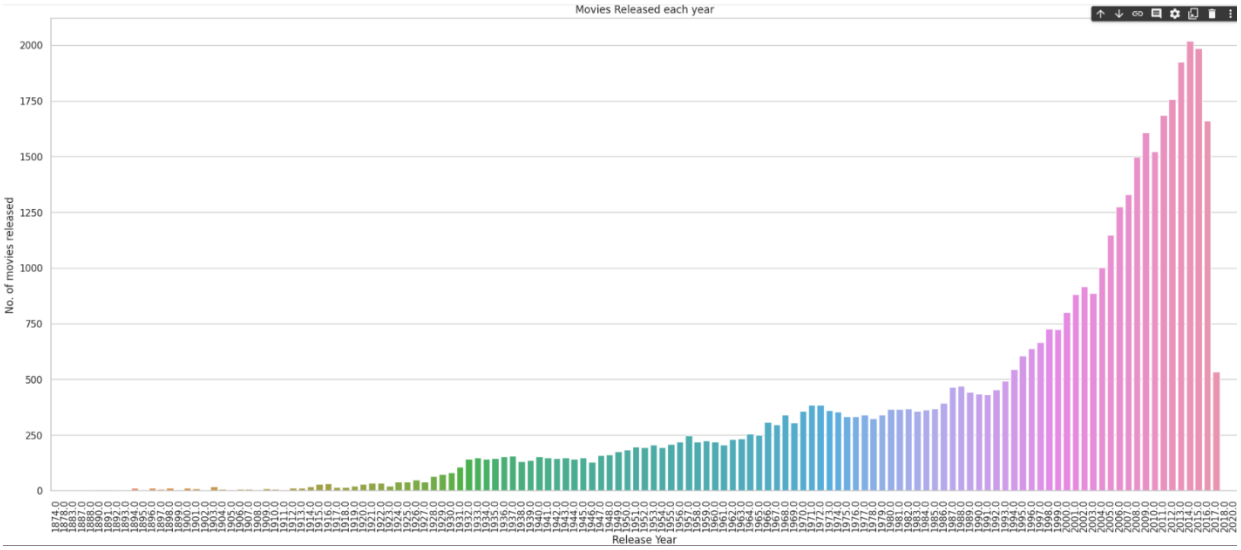
According to plot, we can see US is the most occurring one in production countries.

Number of movies made in each language:



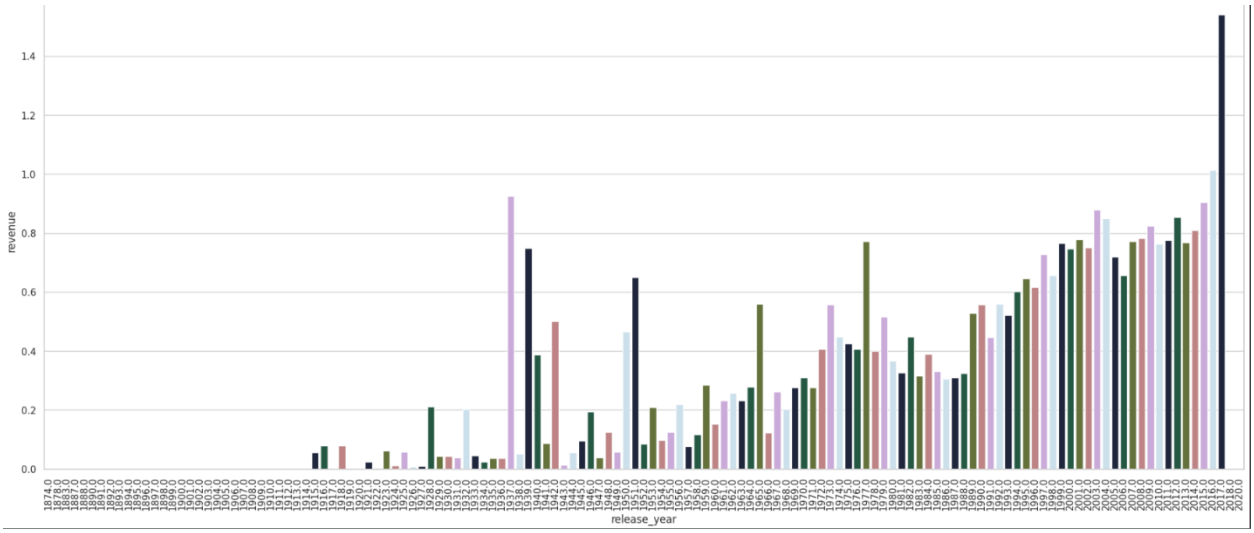
Major number of movies are produced in English, with French and Italian following.

Number of movies released each year:



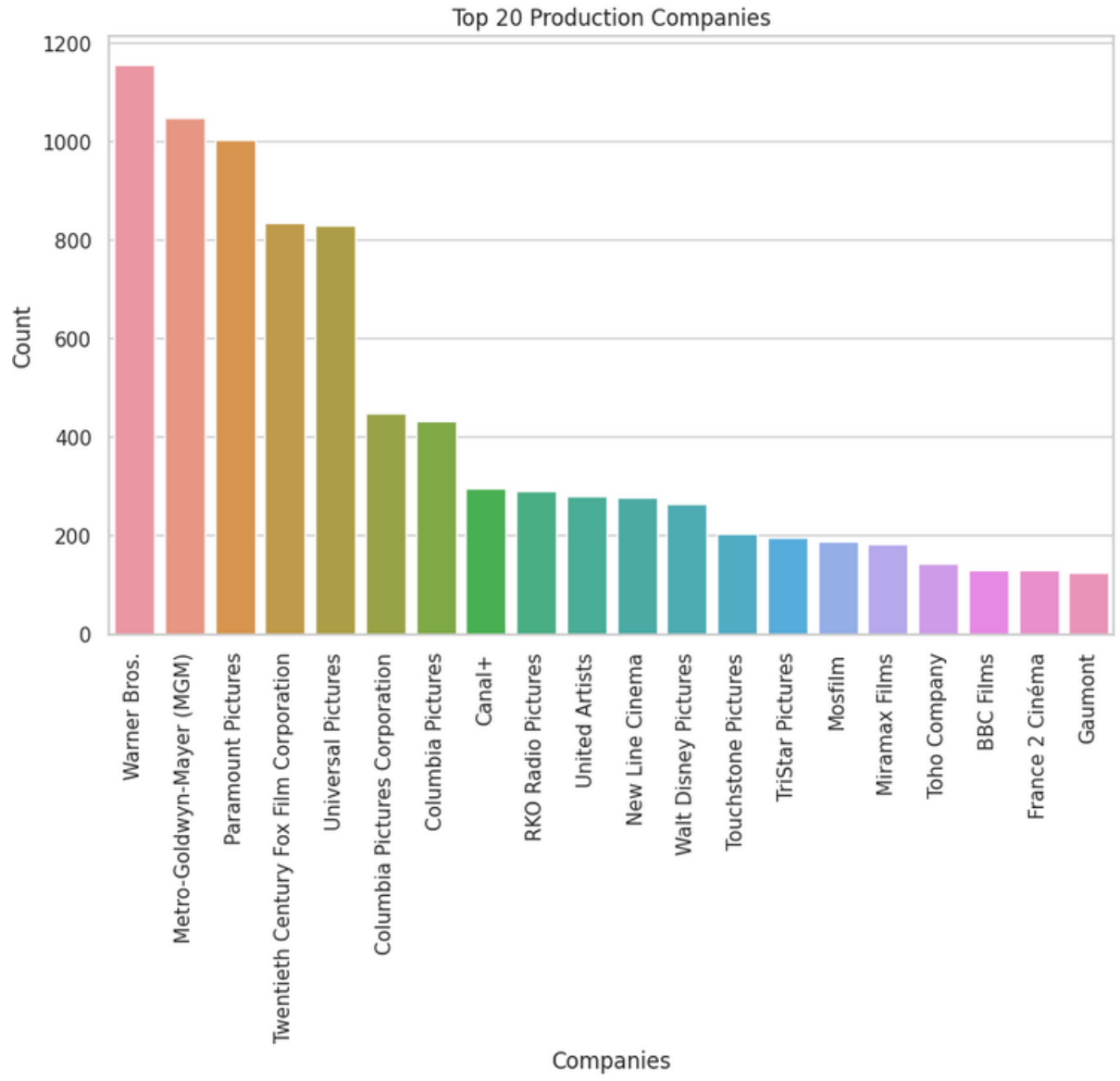
As you can see, the year 2014 had the most released movies and after 1900, there has been a dramatic increase in the number of movies released each year.

Revenue of movies in each year:



The most revenue was earned by movies in 2017.

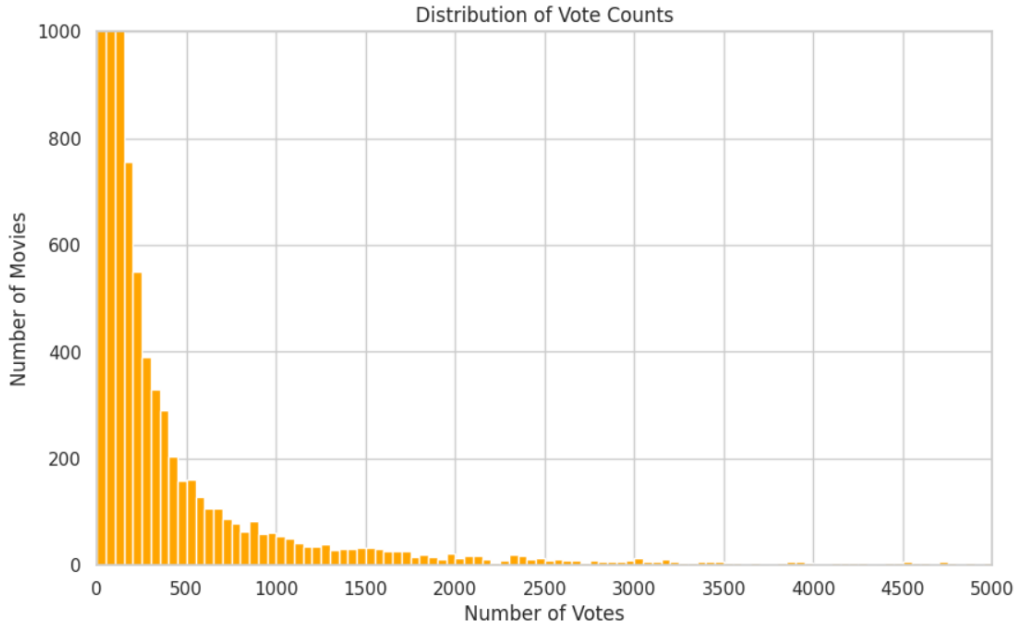
Top 20 Production Companies:



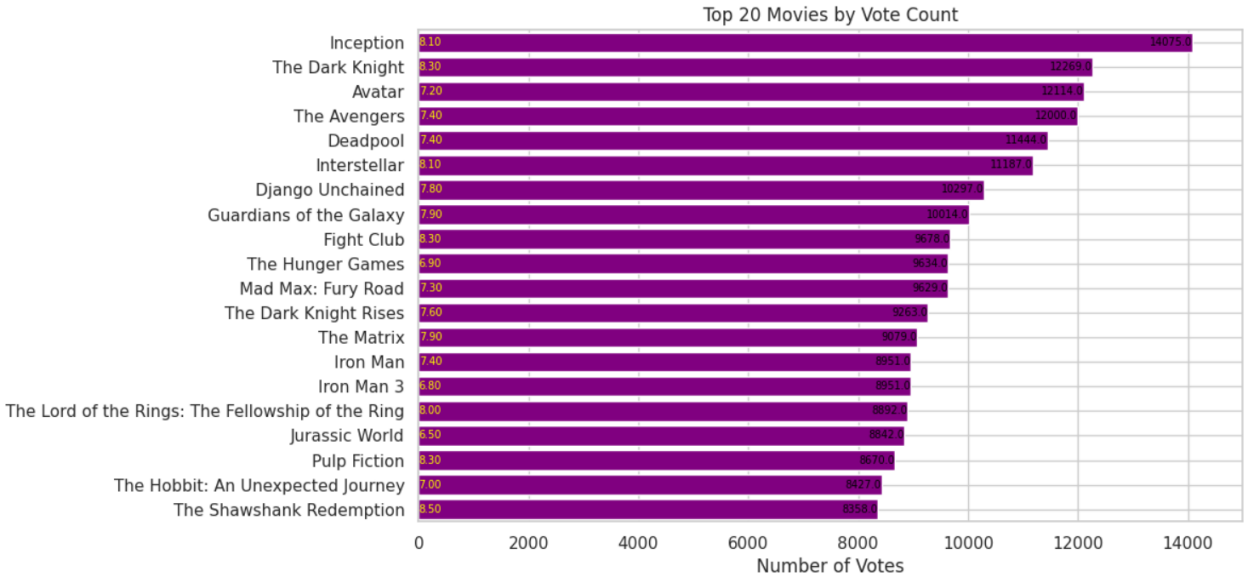
In production_countries, US is the most occurring one and in genres Drama is the most occurring one. We can be place these into NA cells of these columns.

Histogram of number of votes:

The number of votes per movie ranges from 0 to 13,752.



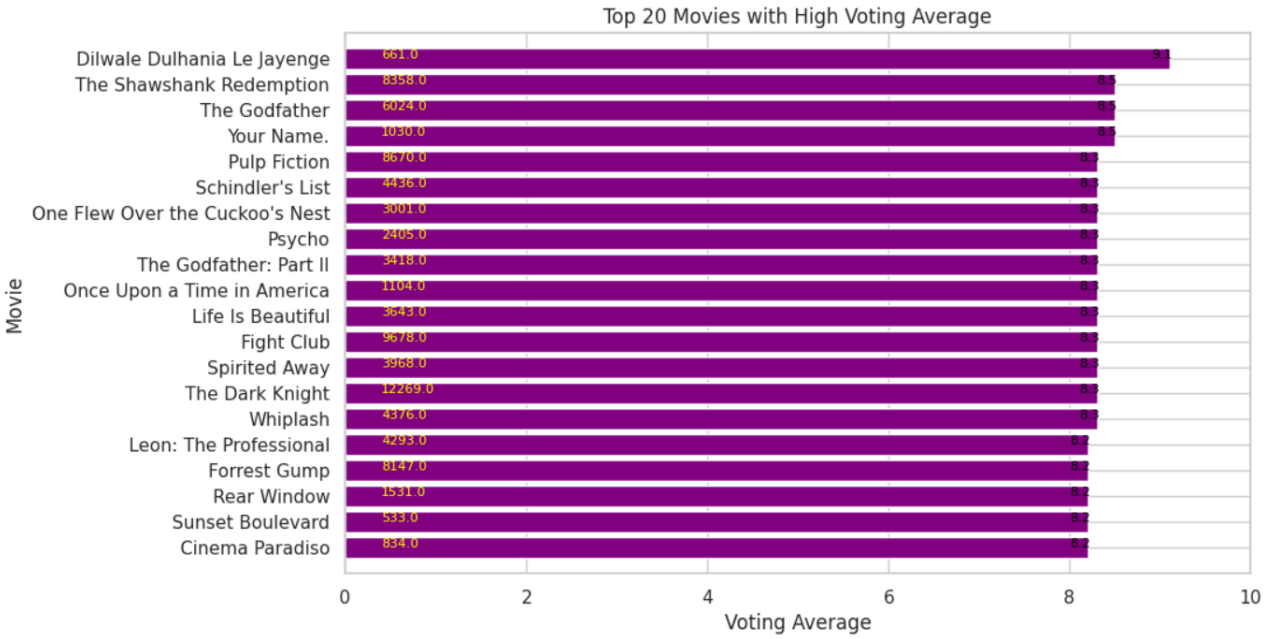
Movies with the highest number of votes:



As you see the lowest vote_average within the Top20 movies with most votes is 6.8. most of the movies had good voting average.

Movies with the highest vote average:

Now we can see the top20 of movies with the highest vote_average that received at least 250 votes (first column, yellow numbers):



Data Analysis – Keywords

Handling Json:

Keywords format is stringified list of json. We must convert them to lists in order to make use of them in the training process. So, we implement a function which converts json list to list of inputs.

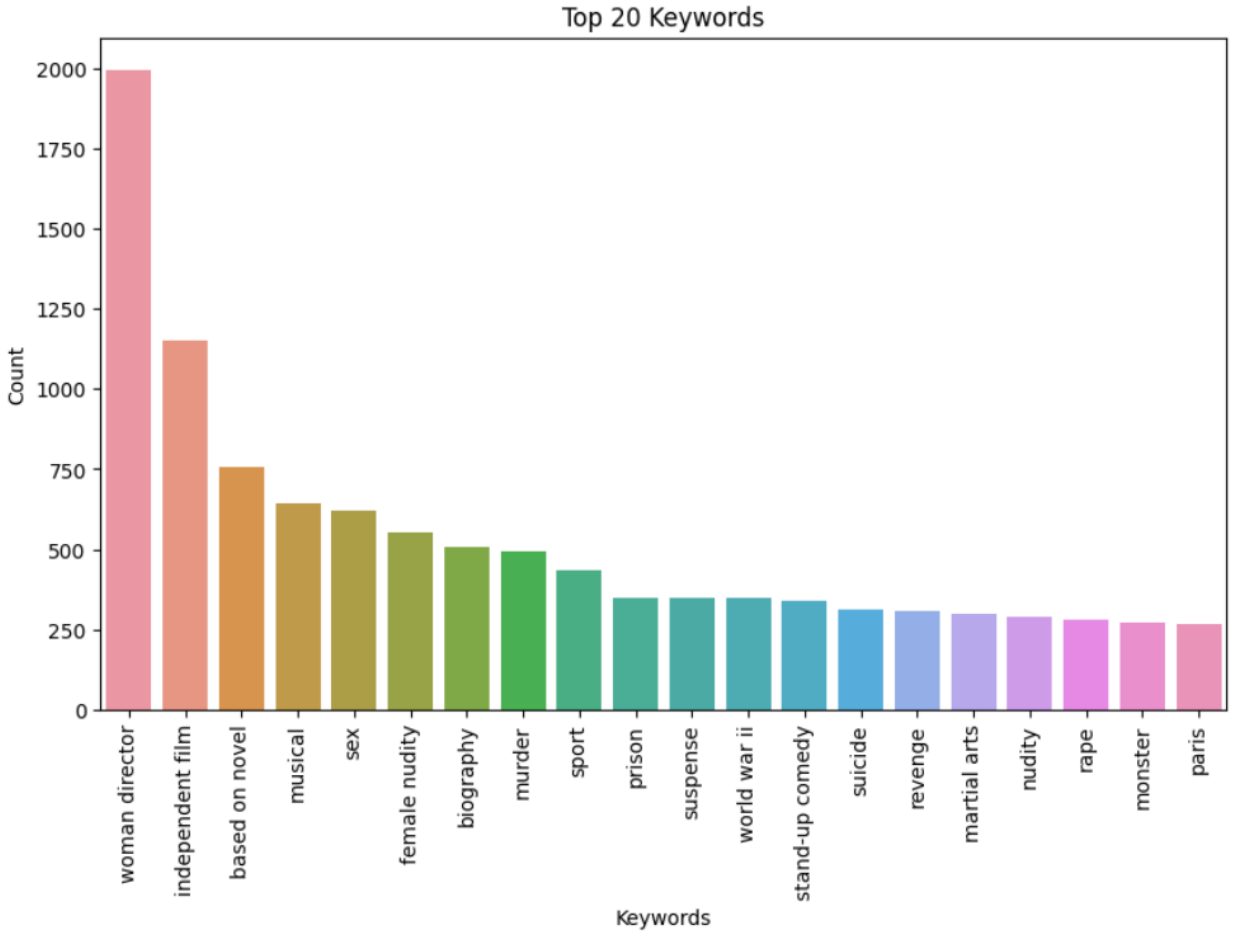
removing any rows with missing values:

First, we calculate and print the number of missing values in each column of the keywords DataFrame. And then removes any rows with missing values from the DataFrame.

List counter function:

For genres and country, we can fill the values with the most appearing entry or mean. The function analyzes the most occurring values for each column in lists. (We explain this function before.)

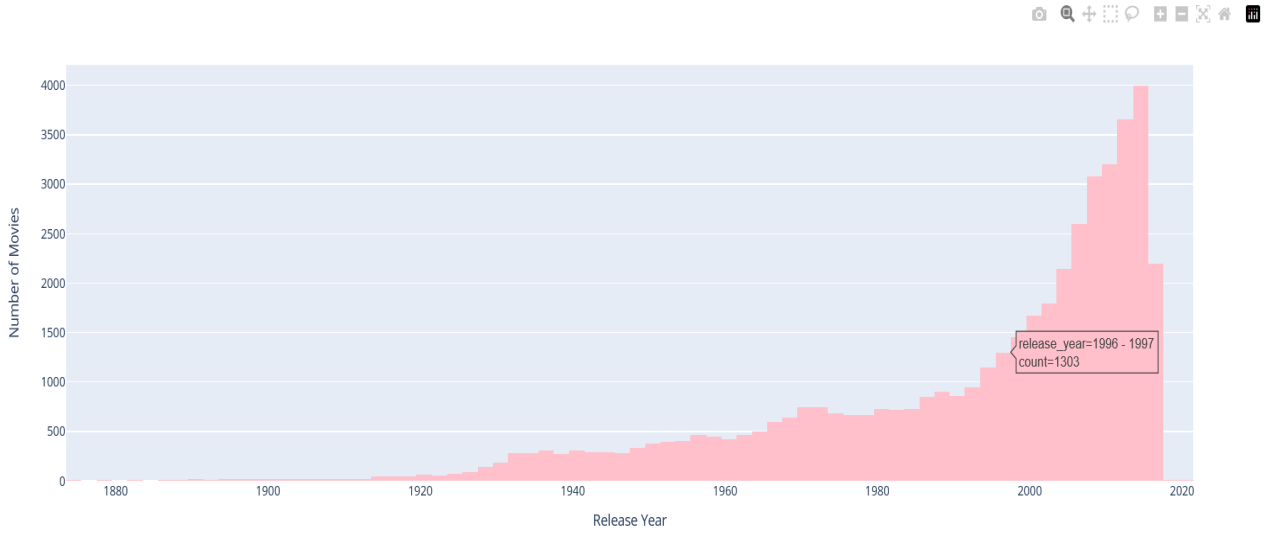
Top 20 Keywords:



As you can see in plot, ‘woman director’ is the top keyword which used around 2000 times. And after that, the most used keyword is ‘independent film’. Other details are shown in plot.

Release Year:

you can hover over the years to see the details in notebook.



Movies distribution shows that over the years the movie industry has developed and got more attention.

Merge Movies and Keywords Dataset:

As you know, id parameters in both metadata and keywords are directing to the same movie. So, we can merge these two datasets.

Data Analysis – Credits

Cast and crew columns type in this dataset is stringified list of json. We first apply the json_to_arr function that we implemented before. Then, we extract the names from the cast and directors from the crew.

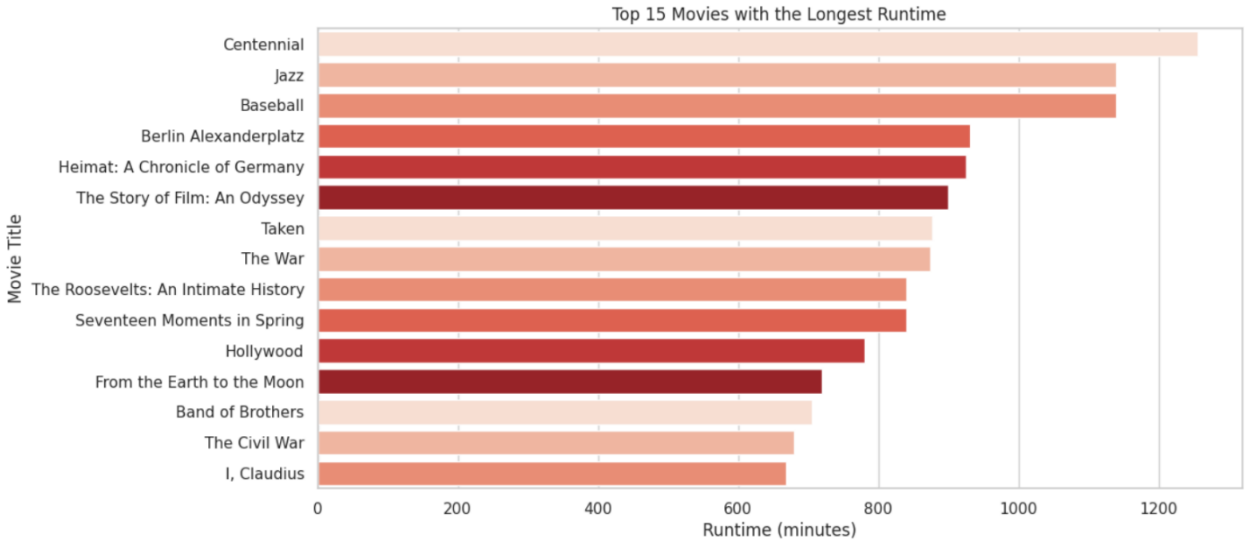
In next step we drop cells with missing both cast and director columns.

Merge Movies and Credits Dataset:

As you know, id parameters in both movies metadata and credits are directing to the same movie. So, we can merge these two datasets.

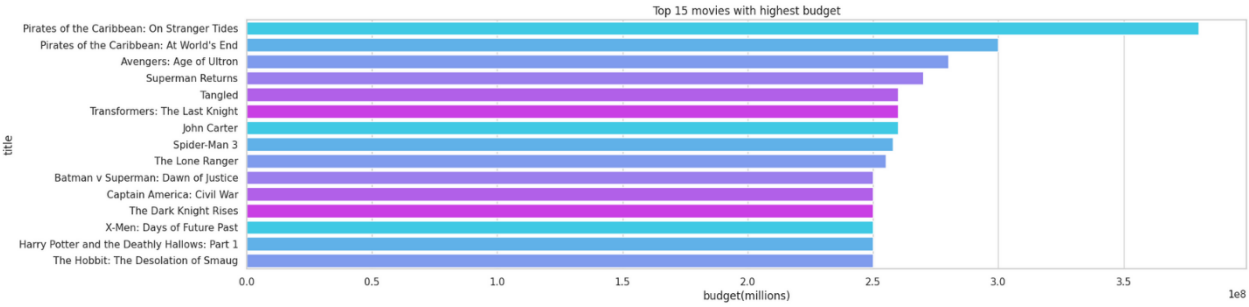
More Plots for Better understanding Of Data:

Top 15 largest runtime movies:



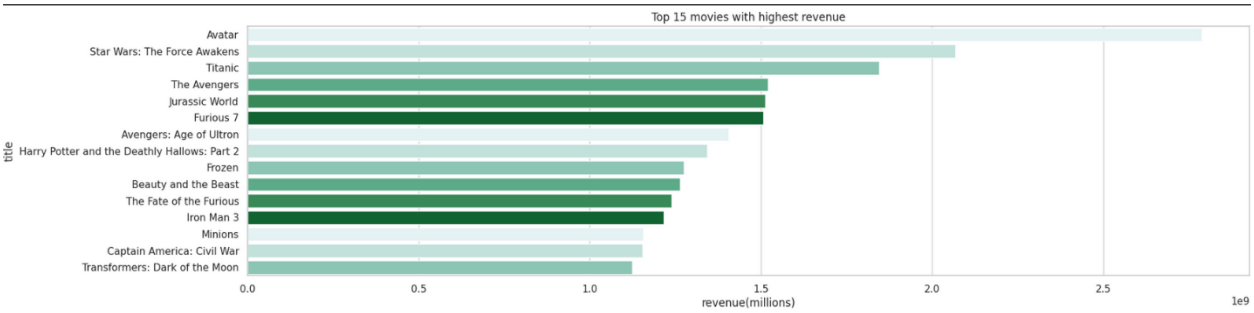
Centennial is the longest runtime movie.

Top 15 movies with highest budget:



Pirates of the Caribbean has the highest budget.

Top 15 movies with highest revenue:



Avatar has the highest revenue.

Movie Recommendation / Content-Based:

We are going to build an engine that computes the similarity between movies based on some metrics and recommends movies that are most similar to a specific movie that a user liked. Since we are going to use movie metadata (or content) to build this engine, this is also known as content-based filtering.

Content Based Recommenders based on:

- Movie Overviews and Taglines
- Improved version using Rating and Popularity

Also, as mentioned in the introduction, I will be using a subset of all the movies available to us due to limiting computing power available to me.

Movie overview Based Recommender:

In this section we will build a recommender using movie descriptions and taglines. We must assess our machine's performance qualitatively because we lack a quantitative metric for doing so.

Cosine Similarity:

We will use the Cosine Similarity to calculate a numeric quantity that denotes the similarity between two movies. Mathematically, it is defined as follows:

$$cosine(x, y) = \frac{x.y^T}{||x||.||y||}$$

Since the TF-IDF Vectorizer was employed, calculating the Dot Product will immediately provide us with the Cosine Similarity Score. Because it is quicker, we will use sklearn's linear_kernel instead of cosine_similarities.

Then we will have a pairwise cosine similarity matrix for movies in our dataset.

Recommendation Function:

based on the cosine similarity score, the 30 most similar movies will be returned by this function.

```
def get_recommendations(title):
    idx = indices[title]
    print(idx)
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:31]
    movie_indices = [i[0] for i in sim_scores]
    return titles.iloc[movie_indices]
```

Some examples to test our recommendation system:

```
get_recommendations('Made').head(10)
```

```
3524
4217                          Johnny Dangerously
3120                          The Way of the Gun
621                                     Thinner
8440                                The Family
695                               The Godfather
6239                       The Constant Gardener
7184                           The International
3627                               Harlem Nights
7042    The Sisterhood of the Traveling Pants 2
6436                                 Renaissance
Name: title, dtype: object
```

```
[ ]  get_recommendations('Rustom').head(10)

     9190
     9098                  Manson Family Vacation
     5003                      That Touch of Mink
     7028                                   Felon
     4528                             Life Stinks
     6954                            Street Kings
     1484                          Doctor Dolittle
     560                       Courage Under Fire
     7699    The Agony and Ecstasy of Phil Spector
     4517                     The Thrill of It All
     5253                        To Hell and Back
     Name: title, dtype: object
```

Improved version using Rating and Popularity:

Now we want to improve our recommendation system and we will add a mechanism to return movies which are popular and have had a good critical response and remove bad movies.

Until now, our recommendation system recommends movies regardless of ratings and popularity. Now we will use these two.

In this part we calculated a weighted rating for each movie based on its vote count, average rating, and the IMDB formula, and adds the calculated ratings to the 'movies' DataFrame for further analysis and comparison.

The 60th percentile movie's vote was calculated using the top 25 movies based on similarity scores. The weighted rating of each movie was determined using the IMDB formula, using this as the value of m.

Some examples to test our recommendation system:

```
improved_recommendations('Se7en')
```

	title	vote_count	vote_average	release_year	wr
7141	Seven Pounds	2092	7	2008.0	6.700930
6677	Zodiac	2080	7	2007.0	6.699500
5051	Monster	500	7	2003.0	6.189523
3440	The Magnificent Seven	472	7	1960.0	6.164392
4125	Insomnia	1181	6	2002.0	5.799284
1773	Cube	1101	6	1997.0	5.788803
2422	The Bone Collector	843	6	1999.0	5.746033
9093	Solace	740	6	2015.0	5.723694
3097	The Cell	442	6	2000.0	5.629376
8064	The Raven	432	6	2012.0	5.625082

```
improved_recommendations('Mean Girls')
```

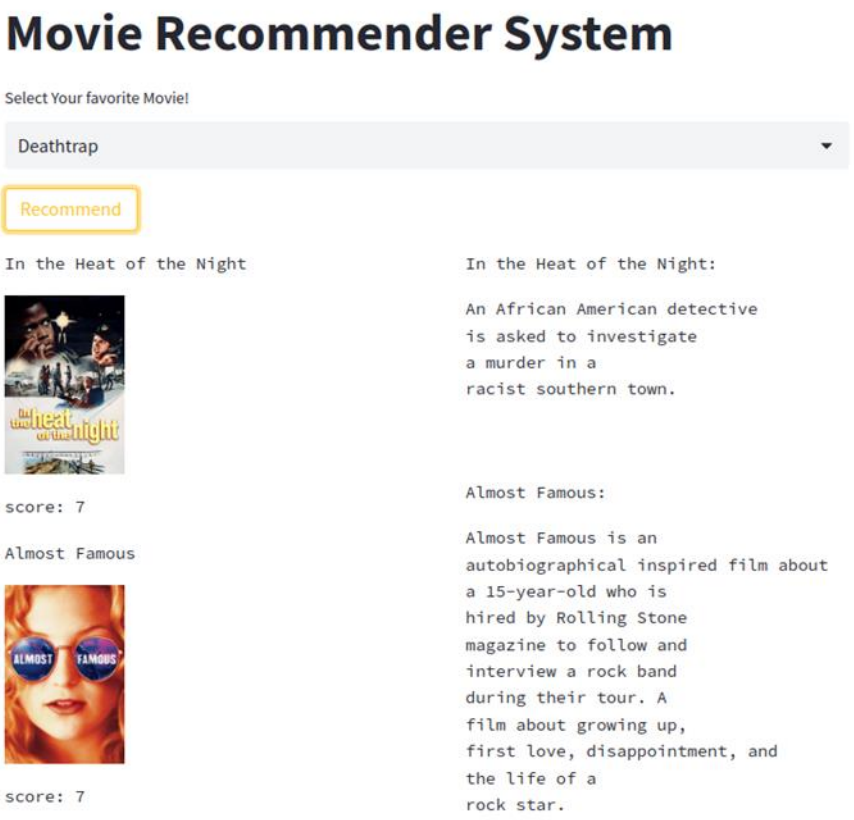
	title	vote_count	vote_average	release_year	wr
9152	Sing Street	669	8	2016.0	6.922223
8154	Pitch Perfect	2310	7	2012.0	6.724716
1092	Cape Fear	692	7	1991.0	6.328091
6358	District B13	572	6	2004.0	5.677413
4948	The Last Boy Scout	502	6	1991.0	5.653211
7157	Wild Child	421	6	2008.0	5.620241
611	The Craft	388	6	1996.0	5.604940
1686	Nineteen Eighty-Four	311	6	1984.0	5.563943
3361	Revenge of the Nerds	204	6	1984.0	5.490466
7541	Death at a Funeral	248	5	2010.0	5.158241

Streamlit

We create the UI using this framework. The application consists of two main columns which show the overview, photo, rating and the name of the movie.

It also has a select box to pick from 10000 movies to get recommendations from.

The application uses **smd** and **movies_raw** datasets to make predictions. We use a modified version of the improved_recommendations to get the photos and other data. This is the final result.



Hugging Face

The application is deployed on the following link:

https://huggingface.co/spaces/Pouriamlk/FinalProjectRecommendationSystem

The dependencies are in the requirements.txt file.

Running the App

The application is currently deployed on Hugging Face to use. To test the application on your local machine, clone the project and run the following command:

```
→ final project python3 -m streamlit run app.py
```

This will run the app on the browser.

Conclusion

To sum up, the goal of this machine learning research was to implement a system for recommending movies using the Movies Dataset retrieved from The Movie Database (TMDb) API. The goal was to examine the dataset, take into account user preferences, and apply algorithms to produce suggestions for movies that might be of interest. A recommendation-specific method (content-based filtering) was the main approache that was investigated.

We can increase the model accuracy by using the other dataset with much more samples, but it requires more computational power.

In order to improve the user experience, interactive interfaces that enable users to engage with the recommendation engine and visualize results were made using Streamlit, a well-known Python library for developing user interfaces. For displaying the models and getting user reaction, this library provides customisable choices.

Finally, the trained model and user interface were deployed on HuggingFace Spaces, a platform that enables developers and researchers to showcase their models and allow others to easily interact with and benefit from their work.

Finally, HuggingFace Spaces, a platform that lets developers and researchers to present their models and allow others to easily interact with and benefit from their work, was used to deploy the trained model and user interface.

Notebook Link:

https://colab.research.google.com/drive/1gNi1eWaFkniqeKkkmv64w1BmndHWsu4k#scrollTo=JPcALtSwIrPe