A PROJECT REPORT

On

Movie Recommendation System

Submitted by

K. JANVEE RAO (32201220026)
IRFAN AHMAD (32201220055)
CHIRANJIB GHOSH (32201220072)
SHREYASI DAS (32201220082)
ANINDYA KUMAR GHOSH (32201220090)

Under the Guidance of

Mr. Subhas Chandra Nath
Assistant Professor

Computer Applications

Asansol Engineering College
Asansol

Affiliated to

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

May 2023

Department of Computer Applications

Asansol Engineering College

Kalyanpur, Sen Raleigh Road, Asansol – 713304

CERTIFICATE

This is to certify that the project work entitled ***"Movie Recommendation System"*** *is a bonafide record of work carried out in the* ***Department of Computer Applications****, Asansol Engineering College, Asansol.*

Name	Roll Number	Registration Number
K. Janvee Rao	32201220026	203221001210068
Irfan Ahmad	32201220055	203221001210039
Chiranjib Ghosh	32201220072	203221001210022
Shreyasi Das	32201220082	203221001210012
Anindya Kumar Ghosh	32201220090	203221001210004

The students of 6th Semester BCA 2022-23 under my / our supervision in requirement of partial fulfillment of the Award of Degree of BCA from Maulana Abul Kalam Azad University Of Technology (MAKAUT), Kolkata (W.B).

Signature of
The Project Internal Guide
Mr. Subhas Chandra Nath
Assistant Professor

Recommendation &Signature
of
Internal / External Examiner

Recommendation &Signature of
The Head of Department
Dr. P. PAL

Recommendation & Signature of
The Principal
Dr. Anilesh Dey

ACKNOWLEDGEMENT

It is our great privilege to express my profound and sincere gratitude to our Project Supervisor **Mr. Subhas Chandra Nath**, **Assistant Professor** for providing me with very cooperative and precious guidance at every stage of the present project work being carried out under his/her supervision. His valuable advice and instructions in carrying out the present study have been a very rewarding and pleasurable experience that has greatly benefitted us throughout our work.

We would also like to pay our heartiest thanks and gratitude to **Dr. Pintu Pal, HoD** and all the faculty members of the Computer Application, Asansol Engineering College for various suggestions being provided in attaining success in our work.

We would like to express our earnest thanks to all technical staff of the Computer Application, and Higher authorities of Asansol Engineering College for their valuable assistance being provided during our project work.

Finally, we would like to express our deep sense of gratitude to our parents for their constant motivation and support throughout our work.

……
Anindya Kumar Ghosh

……
Shreyasi Das

……
Chiranjib Ghosh

……
Irfan Ahmad

……
K. Janvee Rao

Date: 26/05/2023
Place: Asansol

3rd Year
Computer Applications

Table Of Contents

Objective Of The Project

The largest movie libraries in the world are all digitized and transferred to online streaming services, like Netflix, HBO, or YouTube. Enhanced with AI-powered tools, these platforms can now assist us with probably the most difficult chore of all — picking a movie.

The objective of this project is to design and implement a movie recommendation system that suggests movies to users based on their genre preferences and past movie-watching history.

Scope Of The Project

The broad scope of the Movie Recommendation System project includes:

- Data Sources:

  Primary Data Reference: MovieLens Dataset
  Secondary Data Reference: IMDb

- The dataset includes data about thousands of movies with about 20 different genres.

- The system uses content based filtering method to suggest high-rated movies based on the genres and other details of the entered movie.

- The system will be available on Google Colab for 24x7 access.

Methodology

Content-Based Filtering:

Content-based filtering is a technique used in recommender systems to provide personalized recommendations to users based on the characteristics or content of items. It relies on analysing the attributes or features of items that users have interacted with or shown interest in, and then suggesting similar items with comparable content.

Agile Methodology:

- Collecting The Datasets:

  We collected all the required datasets from the Kaggle website. For this project, we require movie.csv and ratings.csv.

- Data Analysis:

  We ensured that the collected datasets are correct and analysed the data in the csv files.

- Algorithms:

  We have used bag-of-words and cosine similarity algorithms for our machine learning recommendation model.

- Training & Testing The Model:

  After the implementation of the algorithm, we trained the model and tested it several times.

Data Description

Source of data:

- Primary data reference – MovieLens
- Secondary data reference – IMDb

List of data:

- movieId: Movie Ids are allocated to all the movie titles so that they can be identified. It is a categorical data.

- title: Movie titles are available in the dataset so that the user knows what titles are available. It is a categorical data.

- year: The year in which the movie was released. It is a discrete numerical column.

- rating: The ratings for movies given by the viewers. It is a numerical, continuous column.

- genres: Movie genres usually help to identify and match the user's taste in movies so that similar movies could be recommended. It is a categorical data.

- tags: Tags are short, descriptive keywords or phrases that users can associate with movies to express their opinions, provide information, or categorize them in a personal way. They can reflect various aspects such as the movie's genre, theme, mood, style, or specific elements like actors, directors, or plot details. The tags are assigned by the users.

System Requirements Specification

Hardware Requirements:

- A PC with Windows/Linux OS
- Processor with 1.7-2.4 gHz speed
- Minimum of 8 gb RAM
- 2 gb Graphic card

Software Specifications:

- Google Colab
- Python libraries: NumPy, Pandas, Matplotlib, Seaborn, Scikit-learn

Software Requirements:

- Google Colab: Colaboratory, or Colab for short, is a product from Google Research which allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

- NumPy: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

- Pandas: Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

- Matplotlib: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

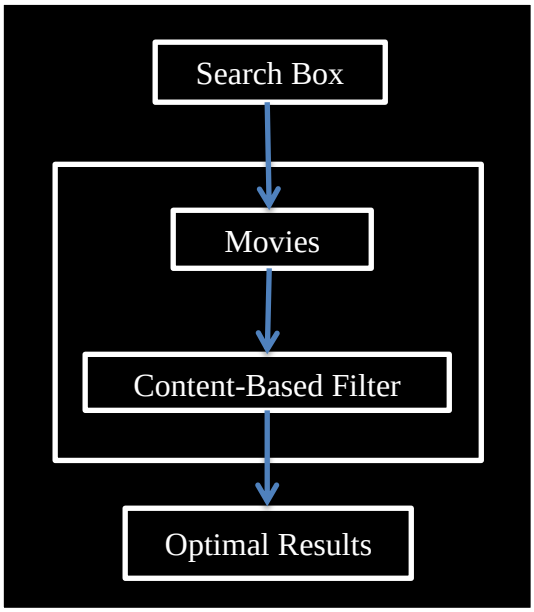
- Seaborn: Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

- Scikit-learn is a open-source software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is a NumFOCUS fiscally sponsored project.

System Analysis & Design

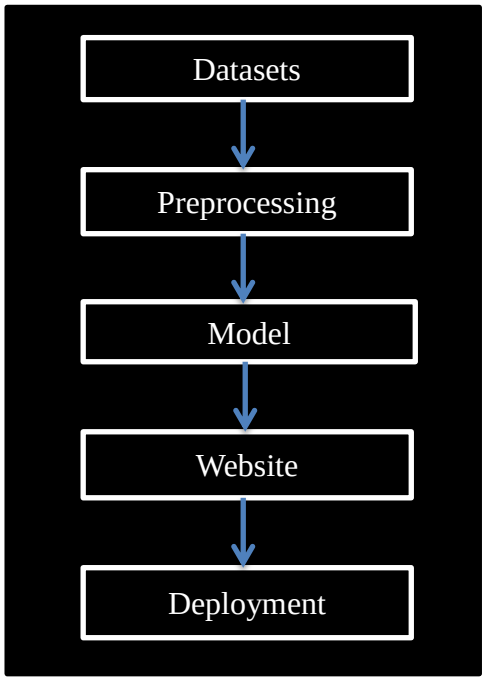
System Architecture Of The Proposed System:

The system will recommend 10 similar movies based on the genres of the movie entered by the user in the search box.



Dataflow:

First, the datasets required to build the model are loaded. Preprocessing of datasets will take place and then by applying content-based filtering, users will get 10 movie recommendations similar to the entered movie.



Implementation

Model Building:

- Bag-Of-Words model:

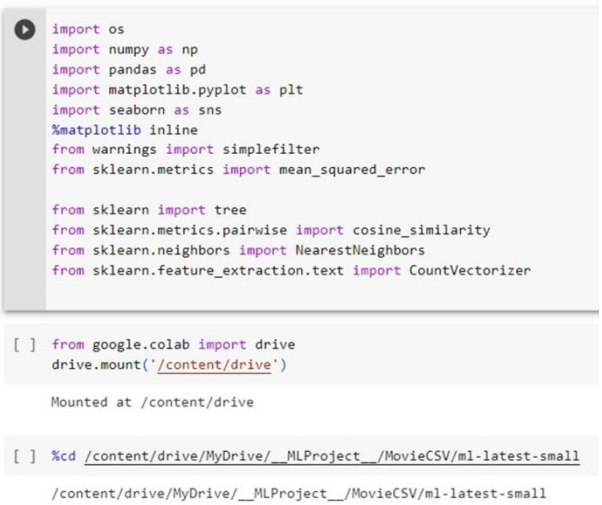
  Whenever we apply any algorithm in NLP, it works on numbers. We cannot directly feed our text into that algorithm. Hence, Bag of Words model is used to preprocess the text by converting it into a *bag of words*, which keeps a count of the total occurrences of most frequently used words.

  This model can be visualized using a table, which contains the count of words corresponding to the word itself.

  Using this model, we created a cosine similarity matrix for the 'genre' column which consisted of multiple genres belonging to the movies.

Coding (Back-end):

For back-end, we have used Google Colab as the coding platform and python as the programming language.



```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from warnings import simplefilter
from sklearn.metrics import mean_squared_error

from sklearn import tree
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors
from sklearn.feature_extraction.text import CountVectorizer
```

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

```
[ ] %cd /content/drive/MyDrive/__MLProject__/MovieCSV/ml-latest-small

    /content/drive/MyDrive/__MLProject__/MovieCSV/ml-latest-small
```

- Importing necessary modules for the movie recommending system.
- Mounting google drive and changing directory to current working directory.

```
[ ]  mv_data = pd.read_csv('MovieS.csv')
     link = pd.read_csv('links.csv')
     rt_data = pd.read_csv('ratings.csv')
     tag = pd.read_csv('tags.csv')
```

```
[ ]  mv_data.head(10)
```

	Unnamed: 0	movieId	genres	title	year
0	0	1.0	Adventure	Toy Story	1995
1	1	1.0	Animation	Toy Story	1995
2	2	1.0	Children	Toy Story	1995
3	3	1.0	Comedy	Toy Story	1995
4	4	1.0	Fantasy	Toy Story	1995
5	5	2.0	Adventure	Jumanji	1995
6	6	2.0	Children	Jumanji	1995
7	7	2.0	Fantasy	Jumanji	1995
8	8	3.0	Comedy	Grumpier Old Men	1995
9	9	3.0	Romance	Grumpier Old Men	1995

```
[ ]  mv_data = mv_data.drop('Unnamed: 0', axis =1)
```

- Loading the datasets from the csv files using the pandas module.
- Displaying the first 10 rows of the dataset using the head() function.

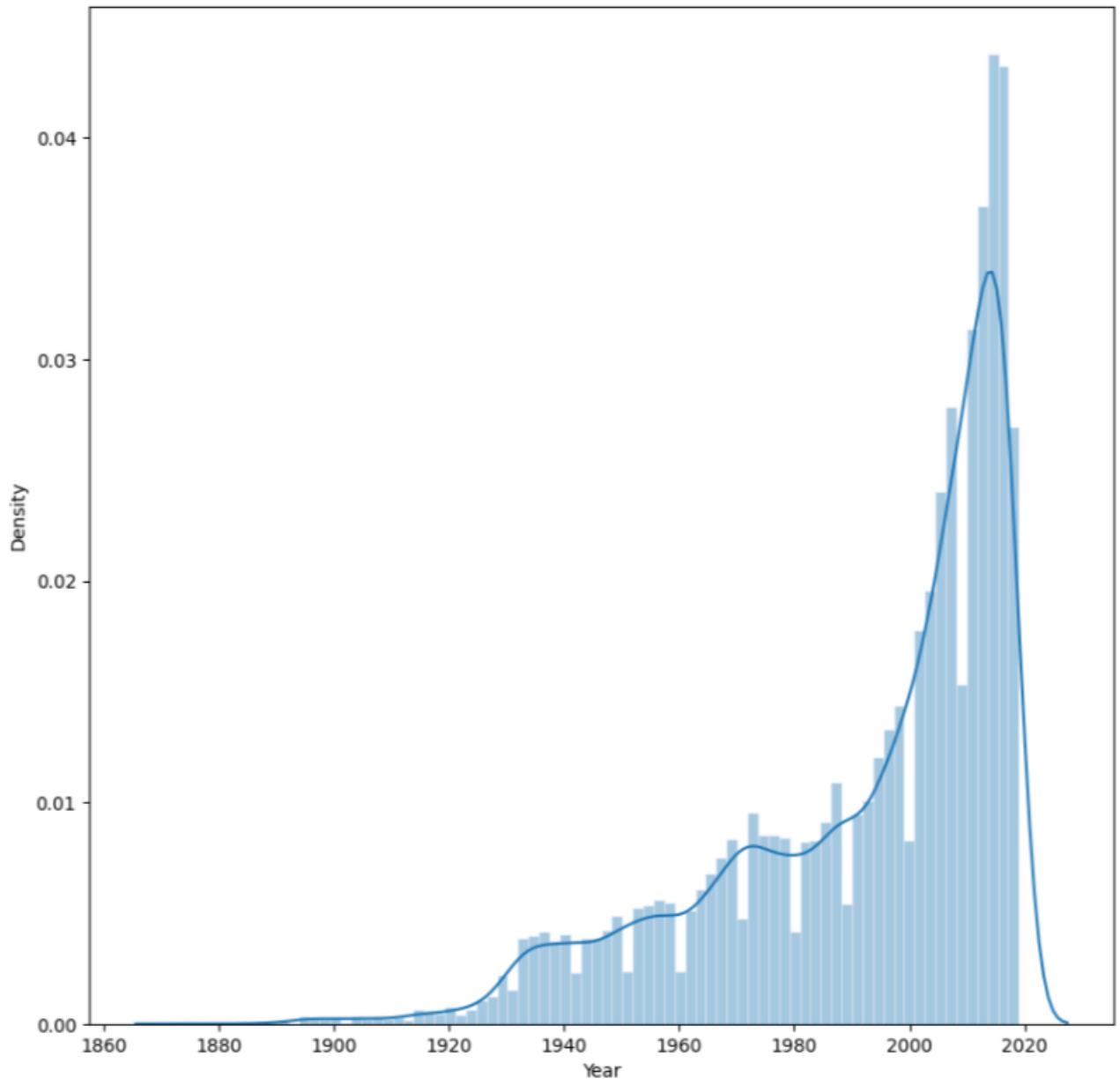
```
[ ]  1  for xz in [mv_data, rt_data,link]:
     2    print (xz.head())
     3    print("+" * 55)
```

```
   movieId                    title     genres    Year
0        1                ToyStory  Adventure  1995.0
1        2                 Jumanji  Adventure  1995.0
2        3          GrumpierOldMen     Comedy  1995.0
3        4         WaitingtoExhale     Comedy  1995.0
4        5  FatheroftheBridePartII     Comedy  1995.0
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   userId  movieId  rating   timestamp
0       1      296     5.0  1147880044
1       1      306     3.5  1147868817
2       1      307     5.0  1147868828
3       1      665     5.0  1147878820
4       1      899     3.5  1147868510
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   movieId  imdbId   tmdbId
0        1  114709    862.0
1        2  113497   8844.0
2        3  113228  15602.0
3        4  114885  31357.0
4        5  113041  11862.0
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

- Dropping unnamed columns.
- Displaying the first 5 rows of all the datasets using the head() function.

```python
# sns.distplot(mv_data['Year'] )
plt.figure(figsize = (10,10))
sns.histplot(
    mv_data["Year"], kde=True,
    stat="density", kde_kws=dict(cut=3),
    alpha=.4, edgecolor=(1, 1, 1, .4),
)
```

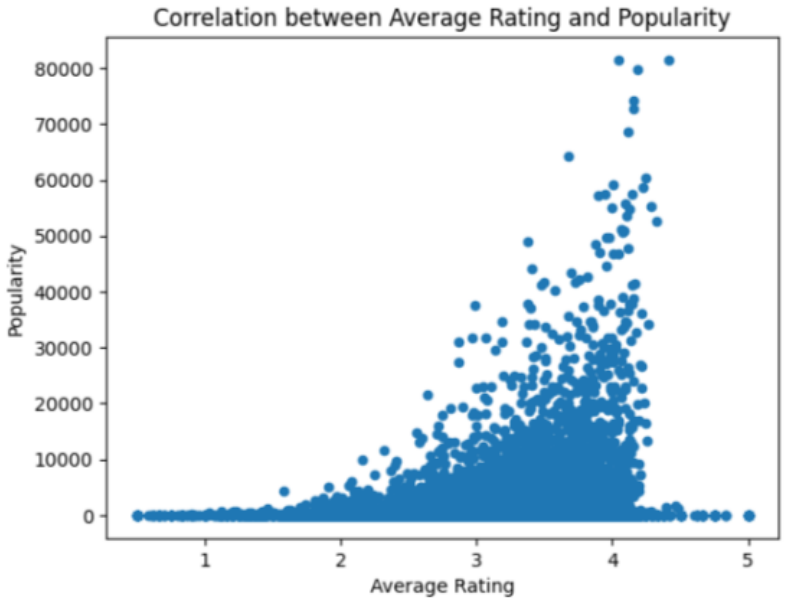
```
<Axes: xlabel='Year', ylabel='Density'>
```



- Hist plot for movie release year.

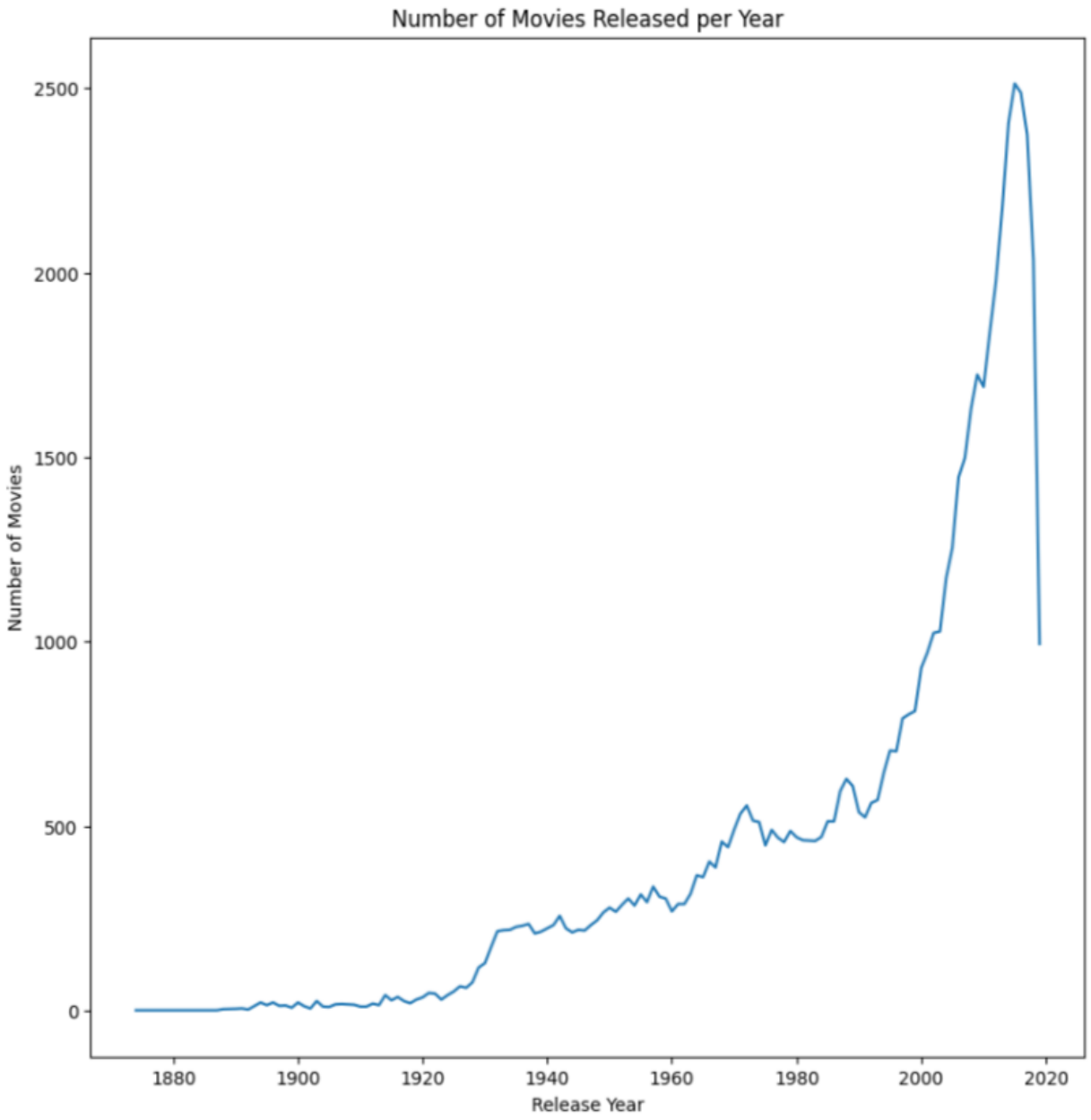
```
popularity = rt_data.groupby('movieId').size()
avg_ratings_per_movie = rt_data.groupby('movieId')['rating'].mean()
ratings_popularity = pd.concat([avg_ratings_per_movie, popularity], axis=1, keys=['Average Rating', 'Popularity'])
plt.figure(figsize = (10,10))
ratings_popularity.plot.scatter(x='Average Rating', y='Popularity')
plt.xlabel('Average Rating')
plt.ylabel('Popularity')
plt.title('Correlation between Average Rating and Popularity')
plt.show()
```

```
<Figure size 1000x1000 with 0 Axes>
```



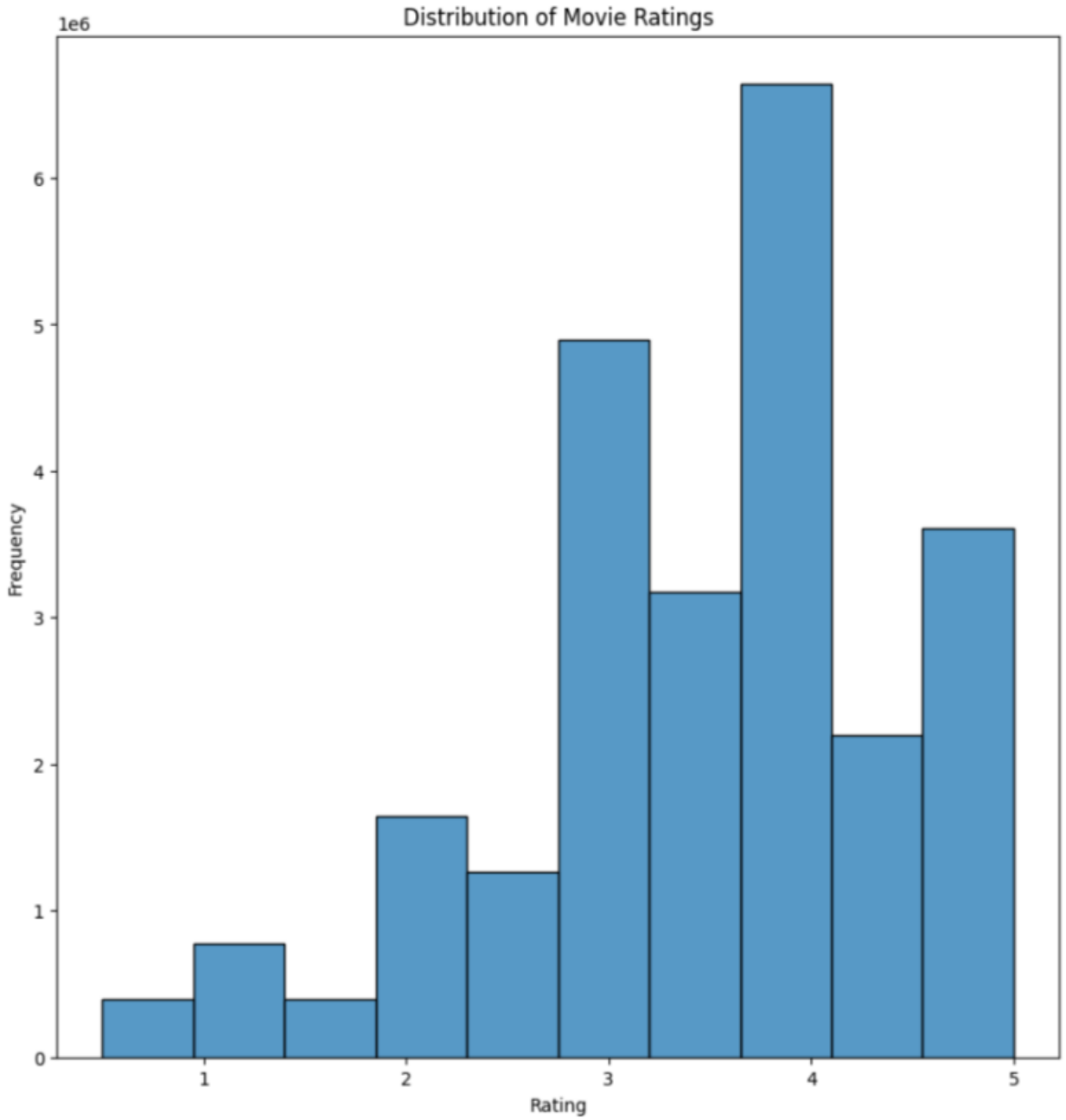
- Scatter Plot for Average Rating & Popularity.

```
[27]  1  plt.figure(figsize = (10,10))
      2  mv_data['Year'].value_counts().sort_index().plot()
      3  plt.xlabel('Release Year')
      4  plt.ylabel('Number of Movies')
      5  plt.title('Number of Movies Released per Year')
      6  plt.show()
      7
```



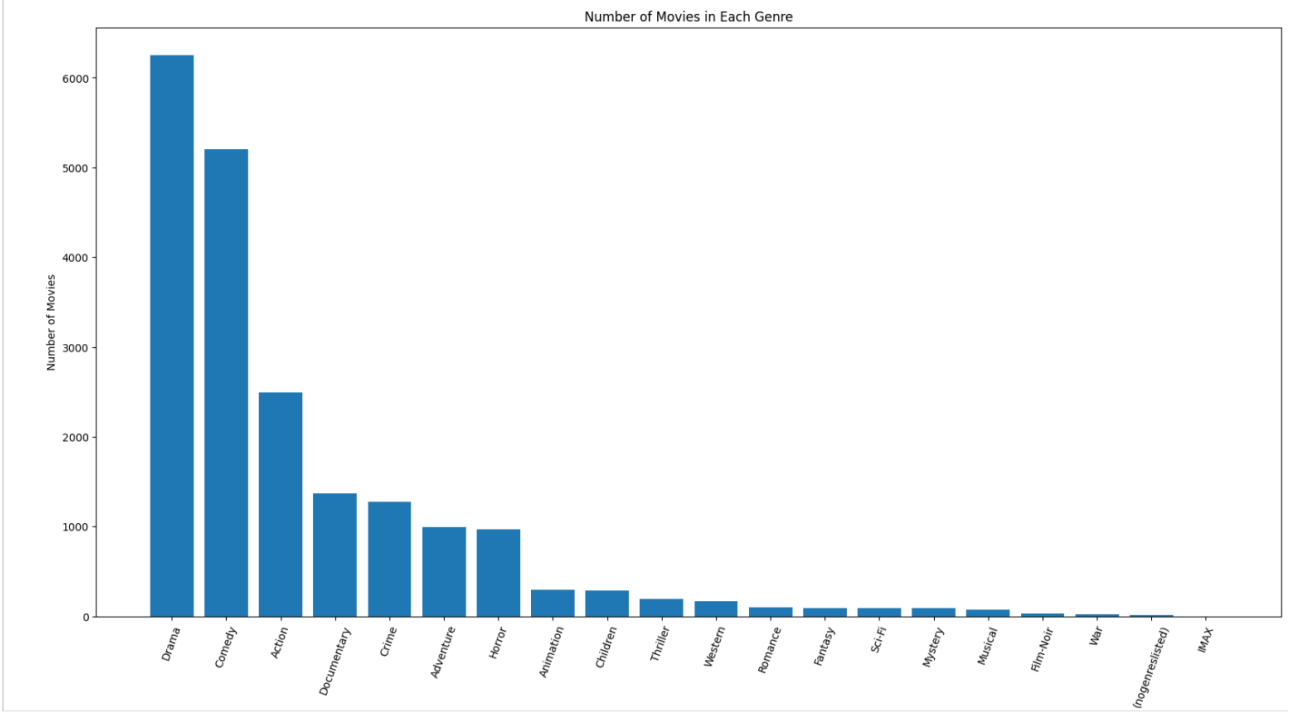
- Plot Showing Count for Number of Movies Released Per Year

```
[29]  1  plt.figure(figsize = (10,10))
      2  sns.histplot(data=rt_data.rating, bins=10)
      3  plt.xlabel('Rating')
      4  plt.ylabel('Frequency')
      5  plt.title('Distribution of Movie Ratings')
      6  plt.show()
```



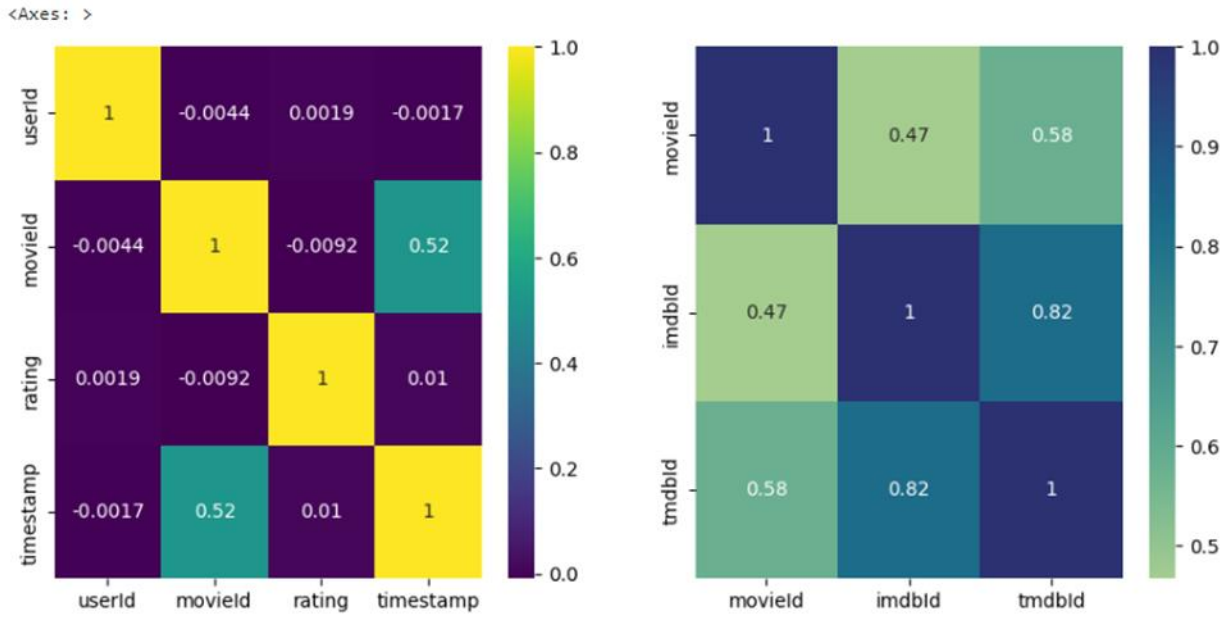
- HistPlot Showing Frequency of The Ratings

```
plt.figure(figsize=(20,10))
genre_counts = mv_data.genres.value_counts()
plt.bar(genre_counts.index, genre_counts.values)
plt.xlabel('Genre')
plt.ylabel('Number of Movies')
plt.title('Number of Movies in Each Genre')
plt.xticks(rotation=70)
plt.show()
```



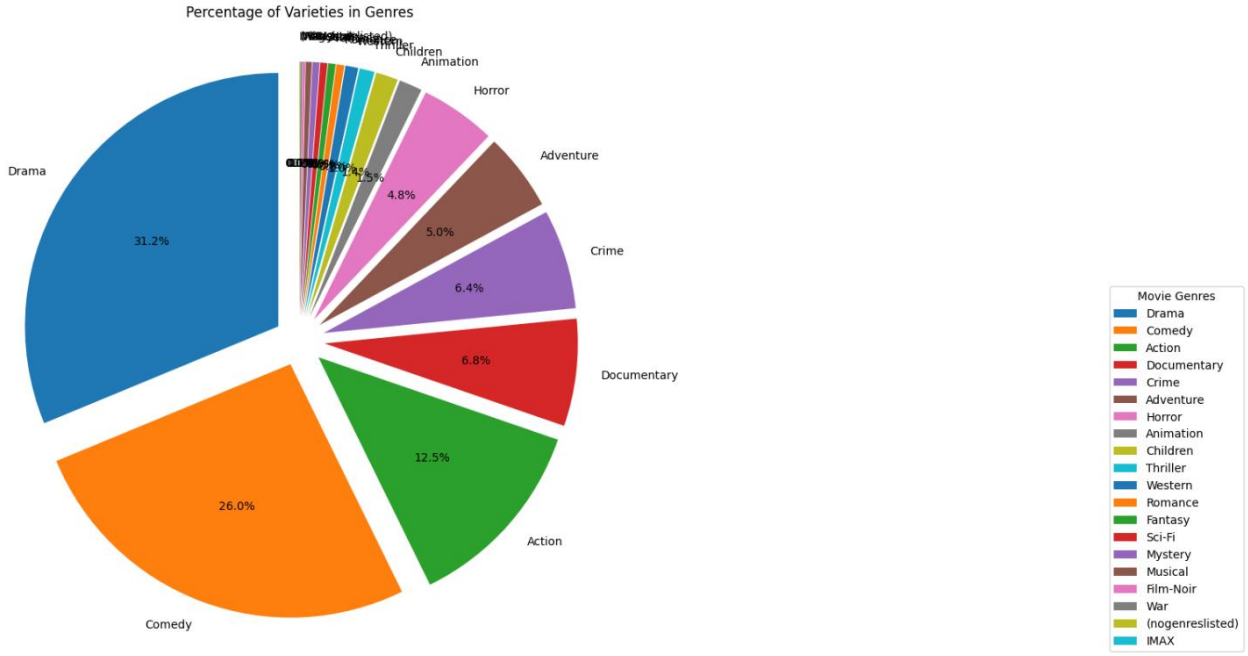
- Bar plot for genre count.

```
[15]  plt.figure(figsize=(11, 11))
      plt.subplot(2,2,1)
      sns.heatmap(data = rt_data.corr(),annot = True, cmap = 'viridis' )
      plt.subplot(2,2,2)
      sns.heatmap(data = link.corr(),annot = True, cmap = 'crest' )
```



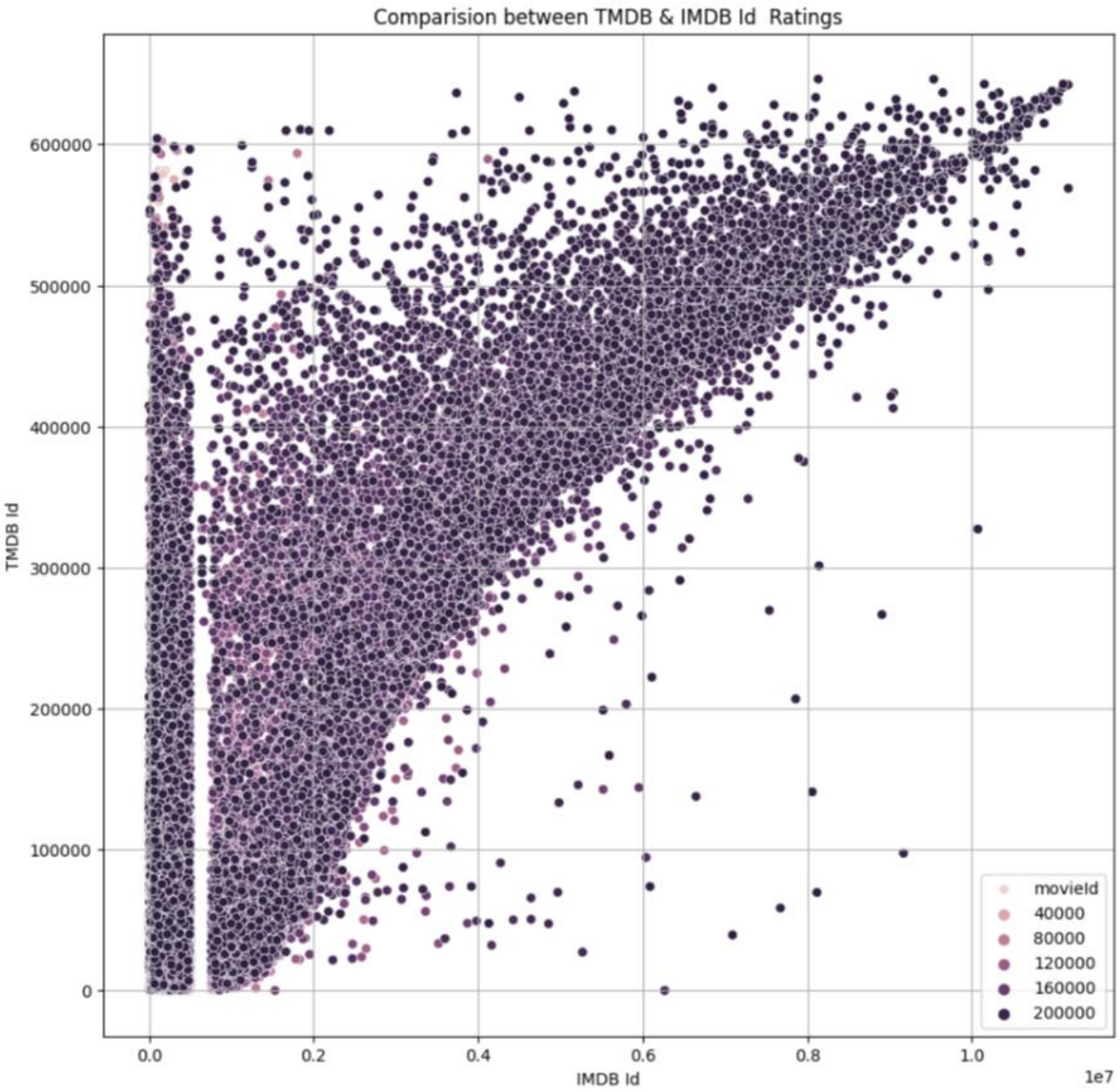
- Heatmap for the correlation between columns.

```
[ ]  1  plt.figure(figsize = (10,10))
     2  data = mv_data['genres']
     3  # Count the frequency of each string
     4  counts = pd.Series(data).value_counts()
     5  explode = explode = [0.1] * len(counts)
     6  plt.pie(counts, labels=counts.index, explode=explode, autopct='%1.1f%%', startangle=90)
     7  plt.title('Percentage of Varieties in Genres')
     8  plt.legend(title='Movie Genres', loc='lower right', bbox_to_anchor=(2, 0))
     9  plt.show()
    10
```



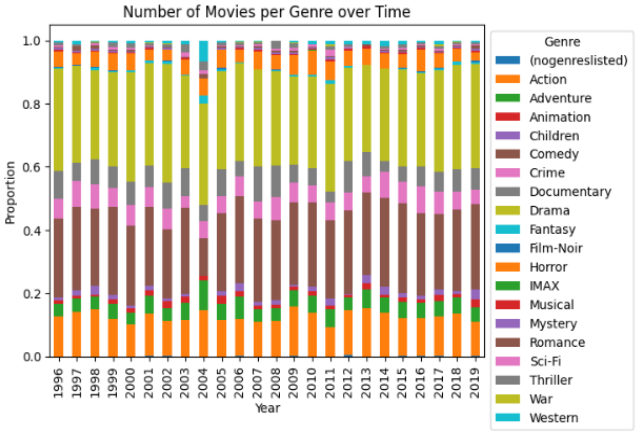
- Pie chart for genre percentage calculation.

```
plt.figure(figsize = (11, 11))
plt.title("Comparision between TMDB & IMDB Id  Ratings")
plt.grid()
sns.scatterplot(data=link, x="imdbId", y="tmdbId", hue="movieId", label="movieId")
plt.xlabel('IMDB Id')
plt.ylabel('TMDB Id')
plt.legend()
plt.show()
```



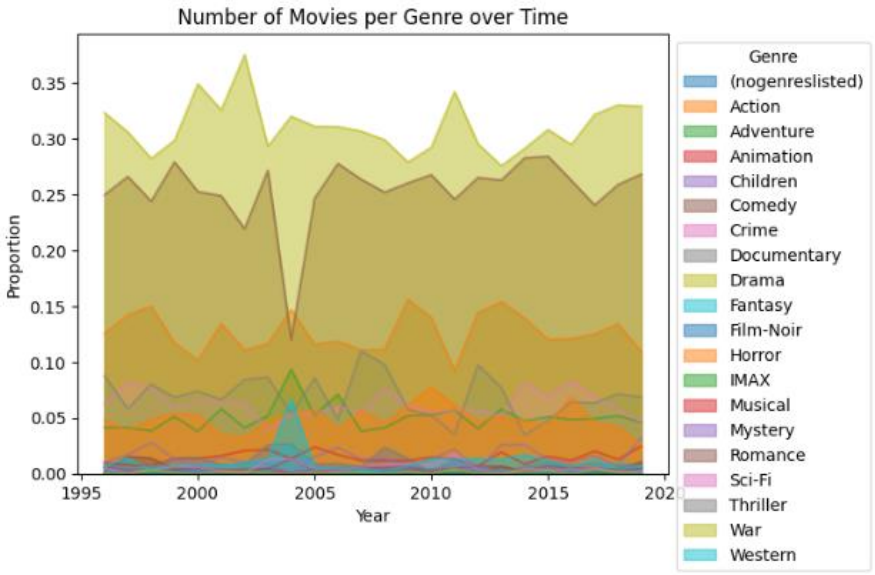
- Scatter plot for TMDB and IMDB ratings comparison.

```
[ ]  genre_counts_over_time = pd.crosstab(pd.to_datetime(rt_data.timestamp, unit='s').dt.year, mv_data.genres).apply(lambda x: x / x.sum(), axis=1)
     genre_counts_over_time.plot(kind='bar', stacked=True)
     plt.xlabel('Year')
     plt.ylabel('Proportion')
     plt.title('Number of Movies per Genre over Time')
     plt.legend(title='Genre', bbox_to_anchor=(1, 1))
     plt.show()
```



- Stacked bar plot for number of movies per genre over time.

```
[ ]  genre_counts_over_time.plot(kind='area', stacked=False)
     plt.xlabel('Year')
     plt.ylabel('Proportion')
     plt.title('Number of Movies per Genre over Time')
     plt.legend(title='Genre', bbox_to_anchor=(1, 1))
     plt.show()
```



- Stacked area plot for number of movies per genre over time.

```
[ ] cv = CountVectorizer()
    # Create the bag-of-words model for the genres column
    genres_matrix = cv.fit_transform(mv_data['genres'].values.astype('U'))
    # Compute the cosine similarity matrix for the genres
    cosine_sim = cosine_similarity(genres_matrix)

    # Define a function to get the top recommended movies based on a movie title
    def get_similar_movies(title, cosine_sim, mv_data):
        # Get the index of the movie title
        index = mv_data[mv_data['title'] == title].index[0]

        # Get the cosine similarity scores for all movies
        sim_scores = list(enumerate(cosine_sim[index]))

        # Sort the scores by descending order
        sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

        # Get the top 10 similar movies
        recommender = [mv_data.iloc[score[0]]['title'] for score in sim_scores[0:10]]
        return recommender

    # Asking the user input for a movie title
    title = input("Enter a movie title: ")

    # Getting the top 10 similar movies
    recommender = get_similar_movies(title, cosine_sim, mv_data)

    rec_movies = []

    # Print the top similar movies
    print(f"Top 10 similar movies to '{title}':")
    for movie in recommender:
        rec_movies.append(movie)
        print(movie)

    print ('___Recommended Movies Generated___')
```

```
Enter a movie title: Toy Story
Top 10 similar movies to 'Toy Story ':
Toy Story
Jumanji
Tom and Huck
GoldenEye
Balto
Cutthroat Island
City of Lost Children, The
Dead Presidents
Usual Suspects, The
Big Green, The
___Recommended Movies Generated___
```

- Creating a bag-of-words model for the genres column.
- Computing the cosine similarity matrix for the genres.
- Defining a function to get the top recommended movies based on a movie title.
- Taking a movie title as user input and displaying top 10 movies with similar genres.

```
[ ] # Load the mv_data.csv and tag.csv files and add a size custom size for reduced load
    mv_data = pd.read_csv('mv_yr.csv', nrows = 10000)
    tag = pd.read_csv('tags.csv', nrows = 10000)

    # Merge the mv_data and tag dataframes based on 'movieId'
    merged_df = pd.merge(tag, mv_data, on='movieId')

    # Create the bag-of-words model for the 'tag' column
    cv = CountVectorizer()
    tag_matrix = cv.fit_transform(merged_df['tag'].values.astype('U'))

    # Compute the cosine similarity matrix for the 'tag' column
    cosine_sim = cosine_similarity(tag_matrix)

    # Define a function to get the top recommended mv_data based on a movie tag
    def get_similar_movies(tag, cosine_sim, movies_data):
        try:
            # Get the index of the movie tag
            index = merged_df[merged_df['tag'] == tag].index[0]

            # Get the cosine similarity scores for all mv_data
            sim_scores = list(enumerate(cosine_sim[index]))

            # Sort the scores by descending order
            sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

            # Get the top 10 similar mv_data
            top_similar_movies = sim_scores[1:11]  # Exclude the first movie itself

            # Get the movie titles of the top similar mv_data
            movie_titles = [movies_data[movies_data['movieId'] == merged_df.iloc[score[0]]['movieId']]['title'].values[0] for score in top_similar_movies]

            return movie_titles

        except IndexError:
            return []  # Return an empty list if no matching mv_data are found
```

```
    # Get the user input for a movie tag
    tag_input = input("Enter a movie tag: ")

    # Get the top 10 similar mv_data
    recommended_movies = get_similar_movies(tag_input, cosine_sim, mv_data)

    if recommended_movies:
        # Print the top similar mv_data
        print(f"Top 10 similar Movies to '{tag_input}':")
        for movie in recommended_movies:
            print(movie)
    else:
        print("No similar  found for the provided tag.")

    print (f"___Recommended Movies By Tags Generated___")
```

```
Enter a movie tag: boring
Top 10 similar Movies to 'boring':
DrivingMissDaisy
AmoresPerros(Love'saBitch)
BeforeSunset
BadEducation(Lamalaeducación)
3-Iron(Bin-jip)
ForrestGump
CityofAngels
LordoftheRings:TheFellowshipoftheRing,The
LostinTranslation
BreakingtheWaves
```

- Creating a bag-of-words model for the tag column.
- Computing the cosine similarity matrix for the tag.
- Defining a function to get the top recommended movies based on a movie tag.
- Taking a movie tag as user input and displaying top 10 movies with similar tags.

```
[ ] mv_data = pd.read_csv('mv_yr.csv', nrows=10000)
    rating = pd.read_csv('ratings.csv', nrows=10000)

    # Merge the mv_data and rating dataframes based on 'movieId'
    merged_df = pd.merge(rating, mv_data, on='movieId')

    # Pivot the merged dataframe to create a user-movie rating matrix
    user_rating_matrix = merged_df.pivot_table(index='userId', columns='title', values='rating').fillna(0)

    # Compute the cosine similarity matrix for the user-rating matrix
    cosine_sim = cosine_similarity(user_rating_matrix)

    # Define a function to get the top recommended movies based on user ratings
    def get_similar_movies(movie_title, cosine_sim, movies_data):
        try:
            # Get the index of the movie title
            index = movies_data[movies_data['title'] == movie_title].index[0]

            # Get the cosine similarity scores for all movies
            sim_scores = list(enumerate(cosine_sim[index]))

            # Sort the scores by descending order
            sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

            # Get the top 10 similar movies
            top_similar_movies = sim_scores[1:11]  # Exclude the first movie itself

            # Get the titles of the top similar movies
            similar_movie_titles = [movies_data.iloc[score[0]]['title'] for score in top_similar_movies]

            return similar_movie_titles

        except IndexError:
            return []  # Return an empty list if no matching movies are found
```

```
    # Get the user input for a movie title
    movie_input = input("Enter a movie title: ")

    # Get the top 10 similar movies based on user ratings
    recommended_movies = get_similar_movies(movie_input, cosine_sim, mv_data)

    if recommended_movies:
        # Print the top similar movies
        print(f"Top 10 similar movies to '{movie_input}':")
        for movie in recommended_movies:
            print(movie)
    else:
        print("No similar movies found for the provided title.")

    print (f"_____Recommended Movies By Ratings Generated_____")
```

```
Enter a movie title: ToyStory
Top 10 similar movies to 'ToyStory':
Georgia
FrenchTwist(Gazonmaudit)
AcrosstheSeaofTime
ItTakesTwo
UsualSuspects,The
HomefortheHolidays
Mr.Holland'sOpus
AceVentura:WhenNatureCalls
Nixon
WingsofCourage
```

- Merging the mv_data and rating dataframes based on movieId and pivoting them to create a user-rating matrix.
- Computing the cosine similarity matrix for the user-rating matrix.
- Defining a function to get the top recommended movies based on a user rating.
- Taking a movie title as user input and displaying top 10 movies with similar ratings.

```python
# Load the mv_data.csv and rating.csv files
mv_data = pd.read_csv('mv_yr.csv', nrows=10000)
rating = pd.read_csv('ratings.csv', nrows=10000)

# Merge the mv_data and rating dataframes based on 'movieId'
merged_df = pd.merge(rating, mv_data, on='movieId')

# Pivot the merged dataframe to create a movie-user rating matrix
movie_rating_matrix = merged_df.pivot_table(index='title', columns='userId', values='rating').fillna(0)

# Compute the cosine similarity matrix for the movie-rating matrix
cosine_sim = cosine_similarity(movie_rating_matrix)

# Define a function to get the top recommended movies based on ratings
def get_similar_movies(rating_input, cosine_sim, movies_data):
    try:
        # Get the index of the rating input
        index = merged_df[merged_df['rating'] == rating_input].index[0]

        # Get the cosine similarity scores for all movies
        sim_scores = list(enumerate(cosine_sim[index]))

        # Sort the scores by descending order
        sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

        # Get the top 10 similar movies
        top_similar_movies = sim_scores[1:11]  # Exclude the first movie itself

        # Get the titles of the top similar movies
        similar_movie_titles = [movies_data.iloc[score[0]]['title'] for score in top_similar_movies]

        return similar_movie_titles

    except IndexError:
        return []  # Return an empty list if no matching movies are found
```

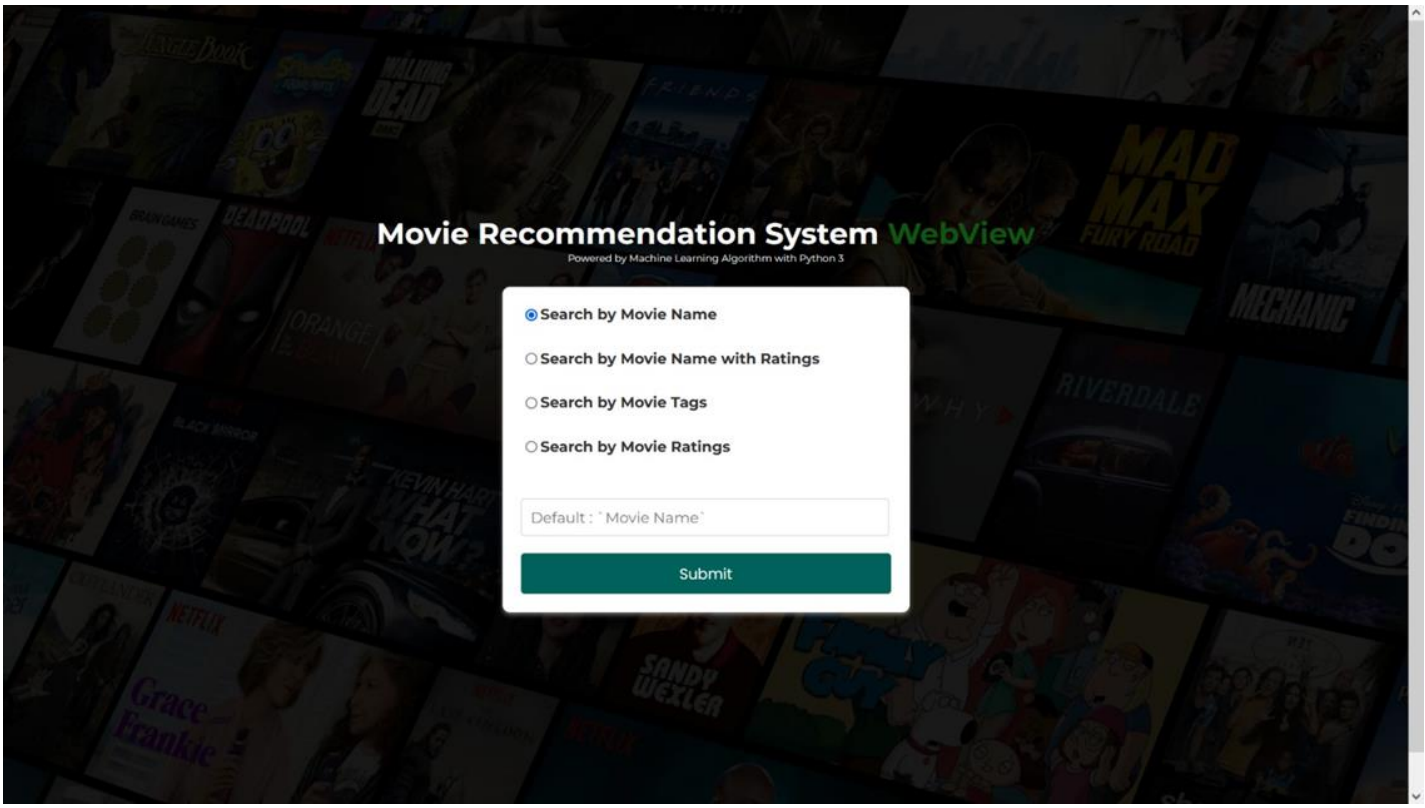
```python
# Get the user input for a rating
rating_input = float(input("Enter a rating: "))

# Get the top 10 similar movies based on ratings
recommended_movies = get_similar_movies(rating_input, cosine_sim, mv_data)

if recommended_movies:
    # Print the top similar movies
    print(f"Top 10 similar movies to '{rating_input}':")
    for movie in recommended_movies:
        print(movie)
else:
    print("No similar movies found for the provided rating.")


print (f"___Recommended Movies By Rating {rating_input} Generated___")
```

```
Enter a rating: 4.5
Top 10 similar movies to '4.5':
FreeWilly2:TheAdventureHome
Reckless
TotalEclipse
PictureBride(Bijophoto)
QueenMargot(ReineMargot,La)
Vanyaon42ndStreet
Crow,The
Cobb
AddamsFamilyValues
IntheLineofFire
```

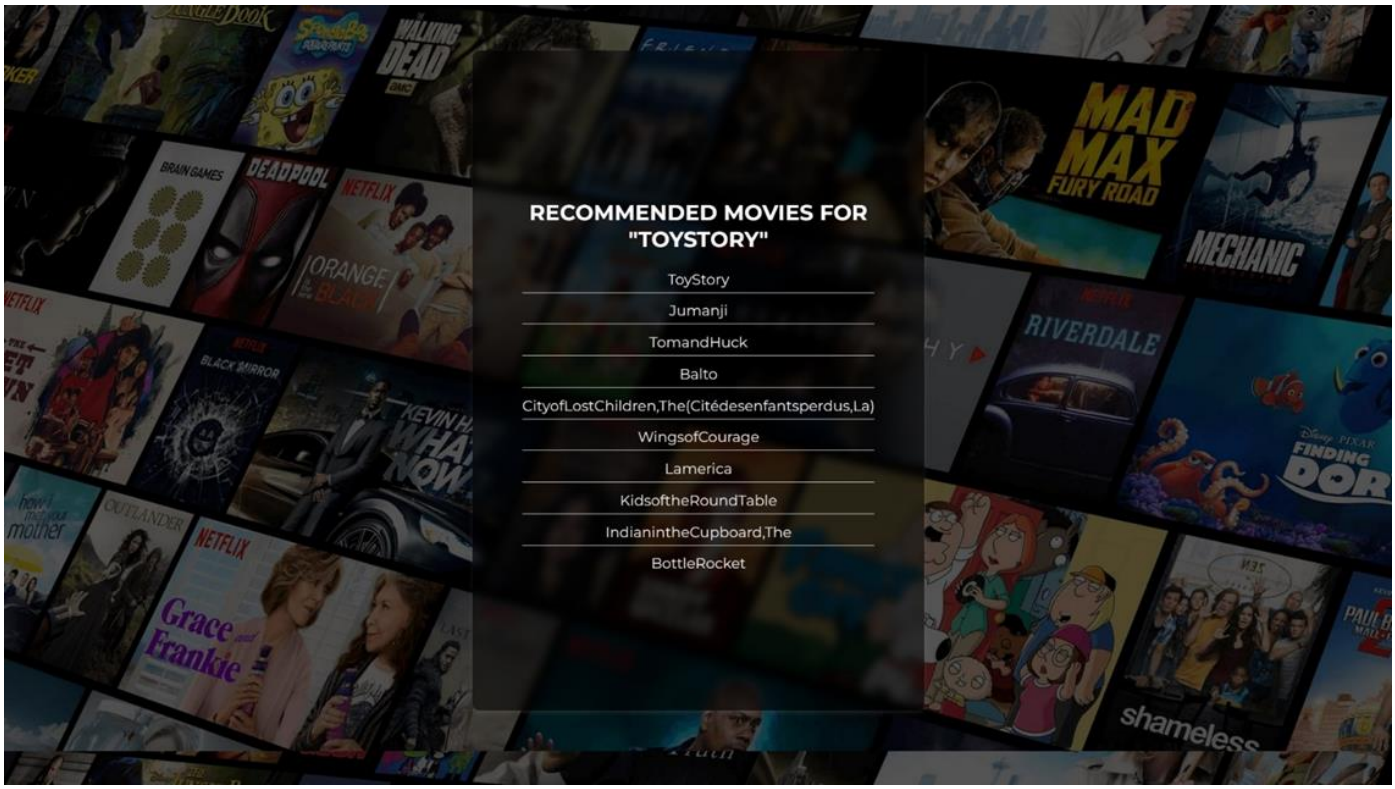
- Merging the mv_data and rating dataframes based on movieId and pivoting them to create a user-rating matrix.
- Computing the cosine similarity matrix for the movie-rating matrix.
- Defining a function to get the top recommended movies based on the user rating of a movie.
- Taking a rating as user input and displaying top 10 movies with similar ratings.

User Interface (Front-end):

For front-end, we have used Visual Studio Code as the coding platform and HTML & CSS as the markup languages.



The webpage users will interact with to get movie recommendations based on genres, tags and ratings.



Top 10 genre based recommendations for Toy Story.

Testing

The goal of system testing, which consists of a variety of tests, is to completely exercise the computer-based system. Despite the fact that each test has a distinct objective, they all aim to ensure that all system components have been correctly integrated and carry out their assigned functions. The testing procedure is actually used to ensure that the product performs exactly as it is intended to.

Unit Testing:

Unit testing is the first level of testing and is performed by the developers themselves.

Integration Testing:

After each unit is thoroughly tested, they are integrated with each other to create modules or components that are designed to perform specific tasks or activities.

System Testing:

System testing is a black box testing method used to evaluate the completed and integrated system as a whole, to ensure it meets the specified requirements.

Conclusion & Future Scope

Conclusion:

In this project, we have successfully made a movie recommending system which recommends movies based on the content that is most similar to what the user searches on the site. We have used cosine similarity to find out which top 5 movies would be the closest to what the user searches. This system tries to save up time for the users who want to watch a movie similar to what they had watched before. We have used movie datasets from Kaggle.com which contained information like genre, cast, movie title and any more to preprocess the data and filter and gather all the information that would to be required to find out all the relevant data for content-based filtering.

Future Scope:

In the proposed approach, we have used content based filtering, further we can move onto the hybrid filtering which would contain both content based and collaborative based filtering in the model. We can also put up and options for register/login so that the information of the users would be stored in the database and can be further used for collaborative filtering.