Abstract

In practical recommendation systems, the predicted rating of a target movie is used to compare it to the user's current ratings. The higher the predicted rating of the target movie, the more chances it has to be recommended. In this research paper, we propose a recommendation method that not only predicts the rated target movie's ratings, but also gives its closest estimation. The closer a similarity of those two values are, then higher chances the target movie has to be recommended. In addition, other features like genres were also used to make recommendations. For this, we compared two models, content-based and collaborative filtering. The experimental results proved that the accuracy rate of the collaborative-filtering model is significantly higher than that of the content-based learning model.

Introduction

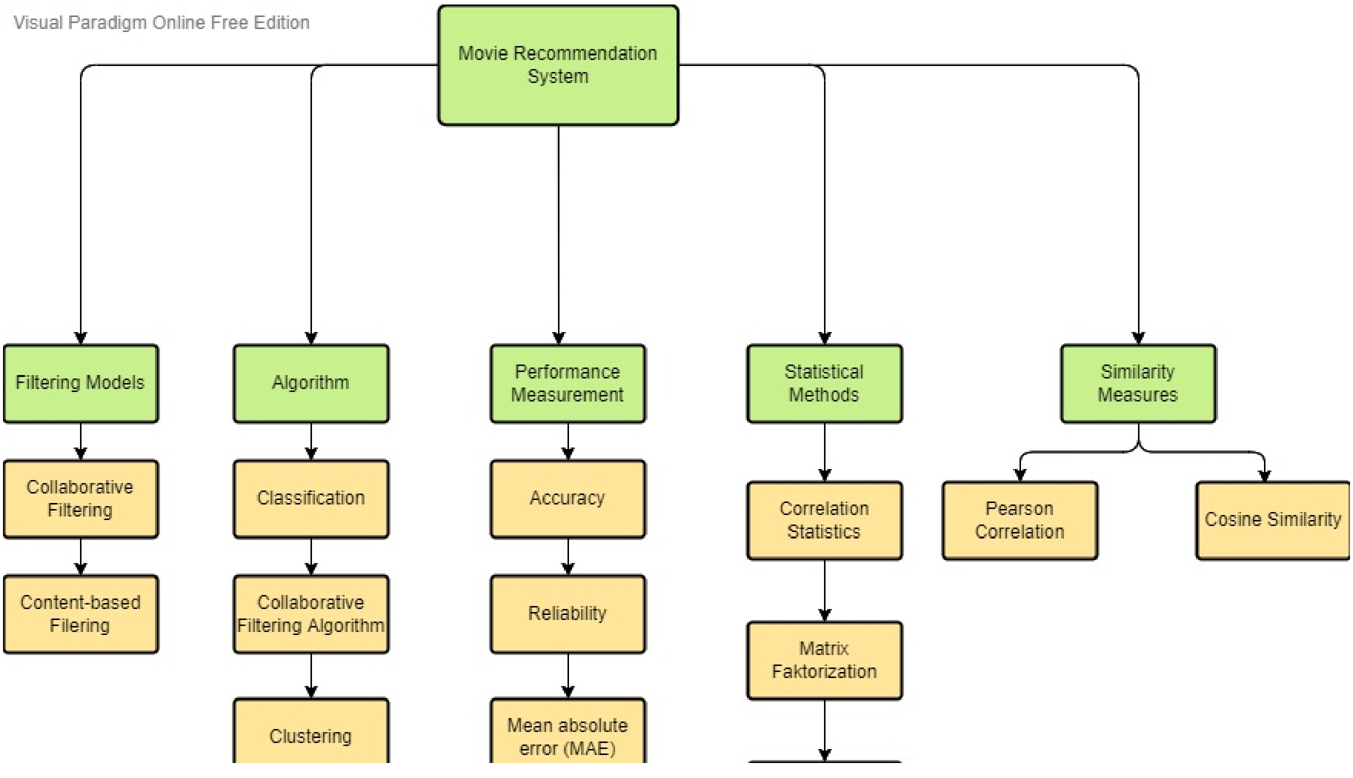
A movie recommendation system is a computer system that generates recommendations for films to users based on their preferences. These recommendations are personalized and based on the user's past behavior, such as what films they have watched in the past, which films they liked and disliked, or their location. A movie recommendation system could recommend similar movies to a user based on what they have watched before or recommend different movies that are similar to those previously liked. Movie recommendation systems will help you choose a film based on things you like so that your experience as an audience member is more pleasant and enjoyable. It also helps with binge watching because it saves time by just circling through your list of favourite suggestions instead of having to go through every single title individually. The workings and systems of movie recommendation systems vary depending on what type of data the system is using to make recommendations. There are two main types: Content-based and collaborative filtering approaches. In content-based approaches, the movie recommendations are generated based on a user viewing a specific film or rating them in some way. In collaborative filtering approaches, a user's ratings and other statistics are used to produce recommendations.

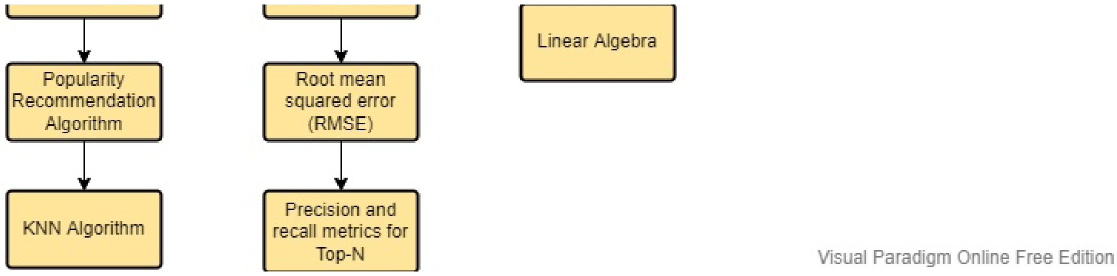
Content-based approaches use information from the user to produce recommendations. In this method, the system checks if the film already exists in a database, or is available from an online store such as Netflix or iTunes. If the film does not exist in the database, then it is recommended to someone based on if that person has watched similar films in the past. If there are multiple users who have watched a certain film but haven't rated it yet on IMDb, then a recommendation could be made by looking at their ratings on that film and comparing them with other users' ratings on that film.

Collaborative Filtering approaches use information from other users to recommend movies. This is a more advanced method of movie recommendation systems because it relies on current ratings by other users on this film, but often results in more accurate recommendations. In this method, additional information may be collected to help with the recommendation process, such as rating the user's general personality or comments about the film made by other users for example. Once a collaborative filtering approach has been used to generate ratings, then movies are recommended based on how similar they are in some way with other movies that have been rated by other people (users).

Related Work

Taxanomy Mapping





Introduction

The related work can be categorized into 5 categories which are filtering models, algorithm, performance measurements, statistical method and similarity measures. The taxonomy mapping for this related work is shown in the figure below.

Filtering Models

For the movie recommender system, we used collaborative filtering and content based-filtering. In content-based filtering, the movies are recommended based on similarities by filtering movies similar to what the user likes. In collaborative filtering, it relies on how other users responded to the same movies based on the preferences of other users. These behaviors are roughly divided into implicit feedback behaviour and explicit feedback behaviour, the latter can directly present user preferences. However, there is a hybrid approach of content-based filtering and collaborative filtering combined. The hybrid approach overcomes the drawbacks of combination in the content-based filtering and collaborative filtering. The hybrid approach is assisted by using a graph-based model with a combination of CF and CB filtering. Also, by using combination of similarity measure a better user similarity can be generated rather than using single similarity measure and efficiency of the system is also increased.

Algorithm

Machine learning algorithms are utilized in the movie recommender system in order to turn the dataset into a model. Various algorithms are used to get the accurate results and precision. Algorithms such as KNN algorithm, collaborative filtering algorithm, clustering, classification, the calculation of score of movie for a particular user which utilizes the preference shown by users to movies of a range of genres and the association of those genres to the genres of the movie to be recommended to the user and the popularity recommendation algorithm were used for the movie recommender system. Since machine learning comprises algorithms, it learns the pattern and the behaviour of a recommender system given its input and output. There are two types of machine learning algorithms and they are supervised and unsupervised. For the movie recommender system, unsupervised learning is used. There are no suitable algorithms as it depends on how the system works, however, based on various studies, many implement deep-learning algorithms, content-based algorithms and collaborative filtering algorithms.

Performance Measurement

Performance measurement is the process of determining how well a system or model performs. It is usually done in order to compare that performance against some reference point, such as a desired level of accuracy or speed. In this case, performance measurements are used to measure the accuracy and reliability of the system in recommending movies based on the user's preferences. Using an ensemble of algorithms, the system provides a set of solutions that are averaged together to determine a final ranking of movies for the user. Based on several research papers, the metrics used in performance measurement are usually accuracy metrics, reliability metrics, precision and recall metrics, mean absolute error (MAE) and root mean squared error (RMSE).

Statistical Methods

Statistical methods are used in collecting, analyzing, and drawing conclusions from quantitative data. Statistic methods in movie recommendation used to give reliable conclusions from the behavior and characteristics of small samples. The author, Prem Melville wrote that the recommender systems were built based on correlation statistics and predictive modeling. Based on his research study, matrix factorization used in collaborative filtering uses linear algebra and statistical matrix analysis which is known as a state of the art technique.

Similarity Measures

Similarity measures are used to measure the similarity between users and items. Pearson Correlation is used to find similarity between different users to obtain recommendation for another user with similar movie interests. Mahesh Goyani and Neha Chaurasiya, the authors from ‘Review of movie recommendation’ used Pearson Correlation for collaborative filtering. The authors also used an algorithm called Log Likelihood Similarity to find item similarity. Besides that, Cosine Similarity is a common method used for Collaborative filtering and Content-based filtering. It determines the similarity between two vectors using the cosine angle. If the cosine angles between two vectors are smaller, this indicates that the vectors are more alike to each other. For examples, when there is a movie that has smaller cosine angle with another movie, then the recommendation systems will recommend the movie to the user.

Conclusion

Several recommendation systems have been proposed that are based on collaborative filtering and content-based filtering. Majority of the recommended systems have been able to solve the problems while providing better recommendations. However, due to information explosion, it is required to work on this research area to explore and provide new methods that can provide recommendations while considering the quality and privacy aspects. Thus, the current recommendation system needs improvement for present and future requirements of better recommendation qualities.

Critical Review

Machine learning can be used to solve the problem of recommendation. Some recommender systems use Machine learning in order to provide better recommendations. Machine Learning algorithms are used in order to predict the preferences based on movies that are similar with other users or movies that have rated higher. The recommender system provides a model that learns from past data and predicts future data for a user. However, there is still uncertainty about the reliability of such data in predicting future behaviour of users due to many factors such as privacy, accuracy, principle applications and many more. Privacy issues arise due to the fact that users will share their analysis data with other parties for better recommendations. However, the recommender system is required to provide recommendations for the user while taking into account those factors. Thus, there are many research papers and studies that concentrate on the use of recommender systems in providing better recommendations based on Machine Learning algorithms.

Problem Statement

Everyone loves movies irrespective of age, gender, race, colour or geographical location and all of us are connected to each other through this amazing medium. Moreover, to achieve customer loyalty, the system provides relevant content and maximizes the time spent by a user. However, the one unique thing about movies is that we have our own preferences of movies to watch such as different and unique genres. The movie recommendation system is to predict and filter preferences according to users' choices of movies. The business problem exists because, in order for the recommender system to work, it needs enough users(more data). For instance, if we want to find similar movies, we will first match the users with a set of similar movies to generate the similarity. The problem for the recommender arises as the user or rating matrix becomes sparse. This problem happens when it becomes complex to find users that rated the same movies because generally, most users do not bother to even rate the movies they have watched. Moreover, in terms of scalability, it requires the right amount of resources as it uses a massive amount of data. Since big data is involved, the cost needed for the recommender system becomes costly in order to build and maintain data warehouses. The stakeholders involved in this problem are the users and also the developer(s) of the recommender system. This impacts the users since they may have a hard time finding movies of their taste and as for the developers, this affects their system negatively as users will shy away from the system, thus, affecting the system's marketability.

Data Acquisition

The TMDB 5000 Movie Dataset was extracted from kaggle and was generated from The Movie Database API. The link is https://www.kaggle.com/tmdb/tmdb-movie-metadata. This dataset has three files, tmdb_5000_credits.csv, tmdb_5000_movies.csv and rating_small.csv.

There are eight quantitative variables namely budget, popularity, revenue, runtime, vote_average, vote_count, rating and timestamp while there are 18 qualitative variables namely genres, homepage, id, keywords, original_language, original_title, overview, production_companies, production_countries, release_date, spoken_languages, status, tagline, title, movie_id, cast, crew and userId.

In [1]:

```
# Functions to import the data
import numpy as np
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
credits = pd.read_csv("tmdb_5000_credits.csv")
movies = pd.read_csv("tmdb_5000_movies.csv")
ratings = pd.read_csv('ratings_small.csv')
```

In [3]:

```
credits.head(4)
```

Out[3]:

	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...

In [4]:

```
movies.head(4)
```

Out[4]:

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity

0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	en	The Dark Knight Rises	Following the death of District Attorney Harve...	112.312950

In [5]:

```
ratings.head(4)
```

Out[5]:

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185

Data Preparation

Data Cleaning

In order to 'clean' the data, we need to find the information of the data by using the shape function in order to find the details of each dataset. In this case, we have 4804 rows and 4 columns in the credits dataset, 4803 rows and 20 columns in the movies dataset while there are 100 004 rows and 4 columns in ratings dataset.

In [6]:

```
print("Dataset Details:- ")
print("Credits Details: ")
print(credits.shape)
print("Movies Details: ")
print(movies.shape)
print("Rating Details: ")
print(ratings.shape)
```

```
Dataset Details:-
Credits Details:
(4803, 4)
Movies Details:
(4803, 20)
Rating Details:
(100004, 4)
```

The dtype describes how the bytes in the fixed-size block of memory corresponding to an array item should be interpreted.

In [7]:

```
#credits dtype
print(credits.dtypes)
```

```
movie_id     int64
title       object
cast        object
crew        object
dtype: object
```

In [8]:
```
#movies dtype
print(movies.dtypes)
```

```
budget                    int64
genres                   object
homepage                 object
id                        int64
keywords                 object
original_language        object
original_title           object
overview                 object
popularity              float64
production_companies     object
production_countries     object
release_date             object
revenue                   int64
runtime                 float64
spoken_languages         object
status                   object
tagline                  object
title                    object
vote_average            float64
vote_count                int64
dtype: object
```

In [9]:
```
#ratings dtype
print(ratings.dtypes)
```

```
userId         int64
movieId        int64
rating       float64
timestamp      int64
dtype: object
```

We will list out all of the columns available in the credits, movies and ratings dataset because the datasets above shown are quite long to be seen.

In [10]:
```
credits.columns
```

Out[10]:
```
Index(['movie_id', 'title', 'cast', 'crew'], dtype='object')
```

In [11]:
```
movies.columns
```

Out[11]:
```
Index(['budget', 'genres', 'homepage', 'id', 'keywords', 'original_language',
       'original_title', 'overview', 'popularity', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'vote_average',
       'vote_count'],
      dtype='object')
```

In [12]:
```
ratings.columns
```

Out[12]:
```
Index(['userId', 'movieId', 'rating', 'timestamp'], dtype='object')
```

In [13]:
```
print(credits.describe(include="all"))
```

```
             movie_id     title  cast  crew
count     4803.000000      4803  4803  4803
unique            NaN      4800  4761  4776
top               NaN  The Host    []    []
freq              NaN         2    43    28
mean     57165.484281       NaN   NaN   NaN
std      88694.614033       NaN   NaN   NaN
min          5.000000       NaN   NaN   NaN
```

```
25%       9014.500000       NaN    NaN    NaN
50%      14629.000000       NaN    NaN    NaN
75%      58610.500000       NaN    NaN    NaN
max     459488.000000       NaN    NaN    NaN
```

Selecting useful attributes:-

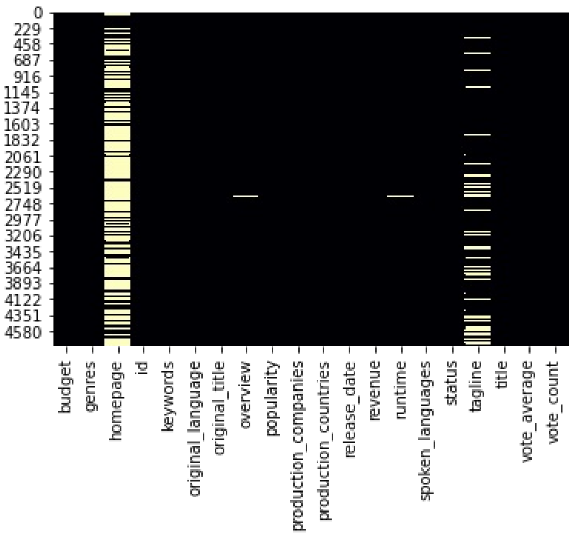
1. movie_id
2. title
3. cast
4. id
5. genres
6. overview
7. popularity
8. runtime
9. status
10. vote_average
11. vote_count

In [14]:

```python
#Here are the missing values in the movie dataset and the interpretation is shown in the heatmap
print(movies.isnull().sum())
sns.heatmap(movies.isnull(),cbar=False, cmap='magma')
```

```
budget                     0
genres                     0
homepage                3091
id                         0
keywords                   0
original_language          0
original_title             0
overview                   3
popularity                 0
production_companies       0
production_countries       0
release_date               1
revenue                    0
runtime                    2
spoken_languages           0
status                     0
tagline                  844
title                      0
vote_average               0
vote_count                 0
dtype: int64
```

Out[14]: <AxesSubplot:>



In order to remove the missing values from the movies dataset, we drop the unused features from the movies dataset and put it in a new variable called movies1 dataset that only contain the relevant attributes.

In [15]:

```python
movies1 = movies.drop(['budget','homepage', 'keywords','original_language', 'original_title', 'production_compani
```

```
movies1.head()
```

Out[15]:

	genres	id	overview	popularity	runtime	status	title	vote_average	vote_count
0	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	19995	In the 22nd century, a paraplegic Marine is di...	150.437577	162.0	Released	Avatar	7.2	11800
1	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	285	Captain Barbossa, long believed to be dead, ha...	139.082615	169.0	Released	Pirates of the Caribbean: At World's End	6.9	4500
2	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	206647	A cryptic message from Bond's past sends him o...	107.376788	148.0	Released	Spectre	6.3	4466
3	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	49026	Following the death of District Attorney Harve...	112.312950	165.0	Released	The Dark Knight Rises	7.6	9106
4	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	49529	John Carter is a war-weary, former military ca...	43.926995	132.0	Released	John Carter	6.1	2124

What has been applied to movies dataset is the same goes to credits dataset. We drop the crew feature from the credits dataset and put the relevant features into a new variable name called credits1.

In [16]:

```
credits1 = credits.drop(['crew'], axis=1)
credits1.head()
```

Out[16]:

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

In [17]:

```
ratings = ratings.drop(['timestamp'], axis=1)
ratings.head()
```

Out[17]:

	userId	movieId	rating
0	1	31	2.5
1	1	1029	3.0
2	1	1061	3.0
3	1	1129	2.0
4	1	1172	4.0

In [18]:

```
#dropping missing values from the movies1 and credits1 dataset
movies1.dropna(inplace=True)
```

In [19]:

```
print(movies1.isnull().sum())
```

```
genres          0
id              0
overview        0
popularity      0
runtime         0
status          0
title           0
vote_average    0
vote_count      0
dtype: int64
```

In [20]:

```
print(credits1.isnull().sum())
```

```
movie_id    0
title       0
cast        0
dtype: int64
```

In [21]:
```
# There are no missing values in ratings table
print(ratings.isnull().sum())
```

```
userId     0
movieId    0
rating     0
dtype: int64
```

Data Integration

For the data integration, we will then be merging credits1 and movies1 datasets together based on the 'titles' since they share the same 'title' feature.

In [22]:
```
df = pd.merge(credits1,movies1, on = 'title')
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', 25)

df.head()
```

Out[22]:

	movie_id	title	cast	genres	id	overview	popularity	runtime	status	vote_average	vote_count
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	19995	In the 22nd century, a paraplegic Marine is di...	150.437577	162.0	Released	7.2	11800
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	285	Captain Barbossa, long believed to be dead, ha...	139.082615	169.0	Released	6.9	4500
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	206647	A cryptic message from Bond's past sends him o...	107.376788	148.0	Released	6.3	4466
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	49026	Following the death of District Attorney Harve...	112.312950	165.0	Released	7.6	9106
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	49529	John Carter is a war-weary, former military ca...	43.926995	132.0	Released	6.1	2124

In [23]:
```
# Changing the column name to movie_id so that can merge with df table
ratings.rename(columns={"movieId": "movie_id"}, inplace=True)
ratings.columns
```

Out[23]:
```
Index(['userId', 'movie_id', 'rating'], dtype='object')
```

In [24]:
```
df = df.merge(ratings, on="movie_id")
df.head()
```

Out[24]:

	movie_id	title	cast	genres	id	overview	popularity	runtime	status	vote_average	vote_count	userId	rating
0	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	285	Captain Barbossa, long believed to be dead, ha...	139.082615	169.0	Released	6.9	4500	39	4.0
1	559	Spider-Man 3	[{"cast_id": 30, "character": "Peter Parker / ...	[{"id": 14, "name": "Fantasy"}, {"id": 28, "na...	559	The seemingly invincible Spider-Man goes up ag...	115.699814	139.0	Released	5.9	3576	492	5.0
2	767	Harry Potter and the Half-Blood Prince	[{"cast_id": 3, "character": "Harry Potter", "...	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	767	As Harry begins his sixth year at Hogwarts, he...	98.885637	153.0	Released	7.4	5293	30	4.0
		Pirates of	[{"cast_id":			Captain							

3	58	the Caribbean: Dead Man's Chest	37, "character": "Captain Jack Sp...	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	58	Jack Sparrow works his way out of a bl...	145.847379	151.0	Released	7.0	5246	19	3.0
4	58	Pirates of the Caribbean: Dead Man's Chest	[{"cast_id": 37, "character": "Captain Jack Sp...	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	58	Captain Jack Sparrow works his way out of a bl...	145.847379	151.0	Released	7.0	5246	23	3.5

In [25]:

```
df[df.duplicated(keep=False)]
#no duplicates
```

Out[25]:

movie_id	title	cast	genres	id	overview	popularity	runtime	status	vote_average	vote_count	userId	rating

In [26]:

```
# drop rows with missing values
df.dropna(inplace=True)
# summarize the shape of the data with missing rows removed
print(df.shape)
```

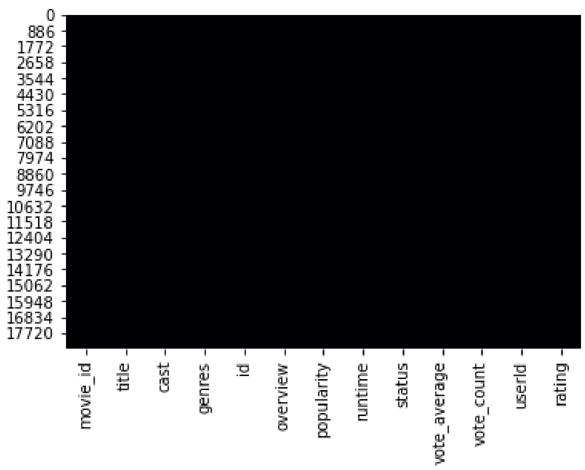
```
(18589, 13)
```

In [27]:

```
#there are no missing values in the merged datasets
print(df.isnull().sum())
sns.heatmap(df.isnull(),cbar=False, cmap='magma')
```

```
movie_id        0
title           0
cast            0
genres          0
id              0
overview        0
popularity      0
runtime         0
status          0
vote_average    0
vote_count      0
userId          0
rating          0
dtype: int64
```

Out[27]: <AxesSubplot:>



Extracting columns for genres

In [28]:

```
import ast
def convert(obj):
    L = []
    for i in ast.literal_eval(obj):
        L.append(i['name'])
```

```
    return L
```

In [29]:
```
df['genres'] = df['genres'].apply(convert)
```

Extracting column for cast

In [30]:
```
def convert3(obj):
    L = []
    counter = 0
    for i in ast.literal_eval(obj):
        if counter !=3:
            L.append(i['name'])
            counter+=1
        else:
            break
    return L
```

In [31]:
```
df['cast'] = df['cast'].apply(convert3)
```

In [32]:
```
df.head()
```

Out[32]:

	movie_id	title	cast	genres	id	overview	popularity	runtime	status	vote_average	vote_count	userId	rating
0	285	Pirates of the Caribbean: At World's End	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	285	Captain Barbossa, long believed to be dead, ha...	139.082615	169.0	Released	6.9	4500	39	4.0
1	559	Spider-Man 3	[Tobey Maguire, Kirsten Dunst, James Franco]	[Fantasy, Action, Adventure]	559	The seemingly invincible Spider-Man goes up ag...	115.699814	139.0	Released	5.9	3576	492	5.0
2	767	Harry Potter and the Half-Blood Prince	[Daniel Radcliffe, Rupert Grint, Emma Watson]	[Adventure, Fantasy, Family]	767	As Harry begins his sixth year at Hogwarts, he...	98.885637	153.0	Released	7.4	5293	30	4.0
3	58	Pirates of the Caribbean: Dead Man's Chest	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	58	Captain Jack Sparrow works his way out of a bl...	145.847379	151.0	Released	7.0	5246	19	3.0
4	58	Pirates of the Caribbean: Dead Man's Chest	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	58	Captain Jack Sparrow works his way out of a bl...	145.847379	151.0	Released	7.0	5246	23	3.5

Data Standardization

We will now standardize the format of the datasets.

In [33]:
```
df.describe()
```

Out[33]:

	movie_id	id	popularity	runtime	vote_average	vote_count	userId	rating
count	18589.000000	18589.000000	18589.000000	18589.000000	18589.000000	18589.000000	18589.000000	18589.000000
mean	2123.953306	2170.338049	36.357556	117.993114	6.801662	1364.354080	345.634138	3.535397
std	6531.440755	6772.980138	28.665135	22.915972	0.790860	1549.348063	195.757797	1.050293
min	5.000000	5.000000	0.034135	0.000000	2.300000	3.000000	1.000000	0.500000
25%	364.000000	364.000000	16.871194	102.000000	6.300000	334.000000	176.000000	3.000000
50%	801.000000	801.000000	28.969151	114.000000	6.900000	765.000000	358.000000	4.000000
75%	1961.000000	1961.000000	48.110909	128.000000	7.400000	1895.000000	518.000000	4.000000
max	116977.000000	116977.000000	271.972889	216.000000	8.500000	12002.000000	671.000000	5.000000

In [34]:
```python
#shows the statistical summary of the dataframe df (movie_id, popularity, vote_average,rating)
df[['movie_id', 'popularity', 'userId', 'vote_average','rating']].describe()
```

Out[34]:

	movie_id	popularity	userId	vote_average	rating
count	18589.000000	18589.000000	18589.000000	18589.000000	18589.000000
mean	2123.953306	36.357556	345.634138	6.801662	3.535397
std	6531.440755	28.665135	195.757797	0.790860	1.050293
min	5.000000	0.034135	1.000000	2.300000	0.500000
25%	364.000000	16.871194	176.000000	6.300000	3.000000
50%	801.000000	28.969151	358.000000	6.900000	4.000000
75%	1961.000000	48.110909	518.000000	7.400000	4.000000
max	116977.000000	271.972889	671.000000	8.500000	5.000000

StandardScaler removes the mean and scales each feature/variable to unit variance. This operation is performed feature-wise in an independent way.

In [35]:
```python
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

# Take a subset of the DataFrame you want to scale
df_subset = df[['movie_id', 'popularity','userId','vote_average','rating']]

print(df_subset.iloc[:5])

# Apply the scaler to the DataFrame subset
df_subset_scaled = ss.fit_transform(df_subset)

print(df_subset_scaled[:5])
```

```
   movie_id  popularity  userId  vote_average  rating
0       285  139.082615      39           6.9     4.0
1       559  115.699814     492           5.9     5.0
2       767   98.885637      30           7.4     4.0
3        58  145.847379      19           7.0     3.0
4        58  145.847379      23           7.0     3.5
[[-0.28156158  3.58372014 -1.56643767  0.12434614  0.44236725]
 [-0.23960952  2.76797551  0.74770865 -1.14013435  1.39450813]
 [-0.20776271  2.18138734 -1.61241409  0.75658638  0.44236725]
 [-0.31631748  3.81971924 -1.66860748  0.25079419 -0.50977364]
 [-0.31631748  3.81971924 -1.64817352  0.25079419 -0.0337032 ]]
```

In [36]:
```python
moviecreds_df_subset_scaled=pd.DataFrame(df_subset_scaled)
```

In [37]:
```python
moviecreds_df_subset_scaled.head(5)
```

Out[37]:

	0	1	2	3	4
0	-0.281562	3.583720	-1.566438	0.124346	0.442367
1	-0.239610	2.767976	0.747709	-1.140134	1.394508
2	-0.207763	2.181387	-1.612414	0.756586	0.442367
3	-0.316317	3.819719	-1.668607	0.250794	-0.509774
4	-0.316317	3.819719	-1.648174	0.250794	-0.033703

Data Transformation

We will be changing the categorical of merged datasets 'status' into numerical format.

In [38]:
```python
df_categorical = df.select_dtypes(exclude=[np.number])
df_categorical.head()
```

Out[38]:

	title	cast	genres	overview	status
	Pirates of the Caribbean: At World's	[Johnny Depp, Orlando Bloom, Keira	[Adventure, Fantasy,	Captain Barbossa, long believed to be	

0	End	Knightley]	Action]	dead, ha...	Released
1	Spider-Man 3	[Tobey Maguire, Kirsten Dunst, James Franco]	[Fantasy, Action, Adventure]	The seemingly invincible Spider-Man goes up ag...	Released
2	Harry Potter and the Half-Blood Prince	[Daniel Radcliffe, Rupert Grint, Emma Watson]	[Adventure, Fantasy, Family]	As Harry begins his sixth year at Hogwarts, he...	Released
3	Pirates of the Caribbean: Dead Man's Chest	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	Captain Jack Sparrow works his way out of a bl...	Released
4	Pirates of the Caribbean: Dead Man's Chest	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	Captain Jack Sparrow works his way out of a bl...	Released

In [39]:
```
df_categorical['status'].unique()
```

Out[39]:
```
array(['Released'], dtype=object)
```

In [40]:
```
df_categorical.status.value_counts()
```

Out[40]:
```
Released    18589
Name: status, dtype: int64
```

In [41]:
```
df_categorical.status.replace({"Released":1, "Post Production":2, "Rumored":3}, inplace= True)
```

In [42]:
```
df_categorical.head()
```

Out[42]:

	title	cast	genres	overview	status
0	Pirates of the Caribbean: At World's End	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	Captain Barbossa, long believed to be dead, ha...	1
1	Spider-Man 3	[Tobey Maguire, Kirsten Dunst, James Franco]	[Fantasy, Action, Adventure]	The seemingly invincible Spider-Man goes up ag...	1
2	Harry Potter and the Half-Blood Prince	[Daniel Radcliffe, Rupert Grint, Emma Watson]	[Adventure, Fantasy, Family]	As Harry begins his sixth year at Hogwarts, he...	1
3	Pirates of the Caribbean: Dead Man's Chest	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	Captain Jack Sparrow works his way out of a bl...	1
4	Pirates of the Caribbean: Dead Man's Chest	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	Captain Jack Sparrow works his way out of a bl...	1

Exploratory Data Analysis

To understand exploratory data analysis, it is critical to have a grasp of statistics. Statistics is the process of using math to summarize and interpret data. In exploratory data analysis, we are looking for relationships between variables, outliers, and other interesting structures.

Basic Statistics

First, we will observe the numerical and categorical variables. Then, we will search for outliers and use visualization to gain insight into the data.

In [43]:
```
df.describe()
```

Out[43]:

	movie_id	id	popularity	runtime	vote_average	vote_count	userId	rating
count	18589.000000	18589.000000	18589.000000	18589.000000	18589.000000	18589.000000	18589.000000	18589.000000
mean	2123.953306	2170.338049	36.357556	117.993114	6.801662	1364.354080	345.634138	3.535397
std	6531.440755	6772.980138	28.665135	22.915972	0.790860	1549.348063	195.757797	1.050293
min	5.000000	5.000000	0.034135	0.000000	2.300000	3.000000	1.000000	0.500000
25%	364.000000	364.000000	16.871194	102.000000	6.300000	334.000000	176.000000	3.000000
50%	801.000000	801.000000	28.969151	114.000000	6.900000	765.000000	358.000000	4.000000
75%	1961.000000	1961.000000	48.110909	128.000000	7.400000	1895.000000	518.000000	4.000000

max	116977.000000	116977.000000	271.972889	216.000000	8.500000	12002.000000	671.000000	5.000000

From what we can observe, the rating variable ranges from 0.5 to 5.0 as well as that there are at least more than one movie to have gotten a 4 ratings. As for the movies' popularity, it ranges from approximately 0.03 to 271.9 in popularity.

In [44]:
```
df_categorical.describe()
```

Out[44]:

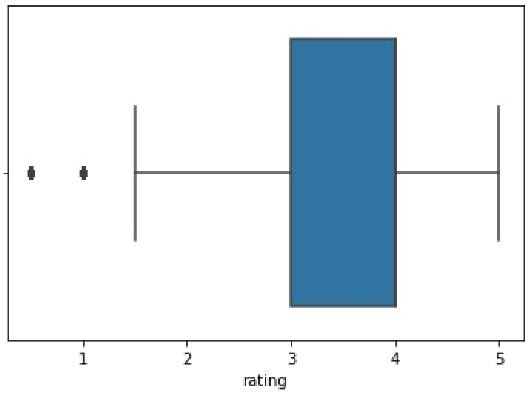
	status
count	18589.0
mean	1.0
std	0.0
min	1.0
25%	1.0
50%	1.0
75%	1.0
max	1.0

And finally based on the status variable, we can assume that all movies has been released.

Finding Outliers

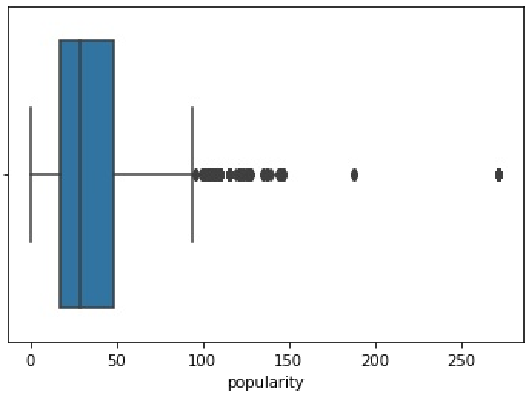
In [45]:
```
sns.boxplot(x=df['rating'])
```

Out[45]:
```
<AxesSubplot:xlabel='rating'>
```



In [46]:
```
sns.boxplot(x=df['popularity'])
```

Out[46]:
```
<AxesSubplot:xlabel='popularity'>
```



There are more outliers found in the popularity column. So, we will identify the outliers using that column.

In [47]:
```
mean = np.mean(df['popularity'])
```

```
mean = np.mean(df['popularity'])
std = np.std(df['popularity'])
df['popularity_zscore'] = (df['popularity'] - mean)/std
print(df.head(5))
```

```
   movie_id                                     title  \
0       285  Pirates of the Caribbean: At World's End
1       559                              Spider-Man 3
2       767    Harry Potter and the Half-Blood Prince
3        58  Pirates of the Caribbean: Dead Man's Chest
4        58  Pirates of the Caribbean: Dead Man's Chest

                                         cast  \
0  [Johnny Depp, Orlando Bloom, Keira Knightley]
1   [Tobey Maguire, Kirsten Dunst, James Franco]
2  [Daniel Radcliffe, Rupert Grint, Emma Watson]
3  [Johnny Depp, Orlando Bloom, Keira Knightley]
4  [Johnny Depp, Orlando Bloom, Keira Knightley]

                        genres   id  \
0  [Adventure, Fantasy, Action]  285
1  [Fantasy, Action, Adventure]  559
2  [Adventure, Fantasy, Family]  767
3  [Adventure, Fantasy, Action]   58
4  [Adventure, Fantasy, Action]   58

                                            overview  popularity  runtime  \
0  Captain Barbossa, long believed to be dead, ha...  139.082615    169.0
1  The seemingly invincible Spider-Man goes up ag...  115.699814    139.0
2  As Harry begins his sixth year at Hogwarts, he...   98.885637    153.0
3  Captain Jack Sparrow works his way out of a bl...  145.847379    151.0
4  Captain Jack Sparrow works his way out of a bl...  145.847379    151.0

     status  vote_average  vote_count  userId  rating  popularity_zscore
0  Released           6.9        4500      39     4.0           3.583720
1  Released           5.9        3576     492     5.0           2.767976
2  Released           7.4        5293      30     4.0           2.181387
3  Released           7.0        5246      19     3.0           3.819719
4  Released           7.0        5246      23     3.5           3.819719
```

In [48]:

```
# create empty list
outlier_index = []

# Take 3 as the thres value
outlier_index.extend(df.index[df['popularity_zscore']>3].tolist())
outlier_index.extend(df.index[df['popularity_zscore']<-3].tolist())


print(outlier_index)
```

```
[0, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 62, 63, 64, 65, 66, 67, 347, 348, 349, 350, 351,
352, 353, 354, 355, 1510, 1511, 1512, 1513, 1514, 1515, 1516, 1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 152
5, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1540, 1541, 1542, 1543, 15
44, 1545, 1546, 1547, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2
366, 2367, 2368, 2369, 4150, 4151, 4152, 4153, 4154, 4155, 4156, 4157, 4158, 4159, 4160, 4839, 4840, 4841, 4842,
4843, 4844, 4845, 4846, 7928, 7929, 7930, 7931, 7932, 7933, 7934, 7935, 7936, 7937, 7938, 7939, 7940, 7941, 10971
, 10972, 10973, 10974, 10975, 13898, 13899, 13900, 13901, 13902, 13903, 13904, 13905, 13906, 13907, 13908, 13909,
13910, 13911, 13912, 13913, 13914, 13915, 13916, 13917, 13918, 13919, 13920, 13921, 13922, 13923, 13924, 13925, 1
3926, 13927, 13928, 13929, 13930, 13931, 13932, 13933, 13934, 13935, 13936, 13937, 13938, 13939, 13940, 13941, 13
942, 13943, 13944, 13945, 13946, 13947, 13948, 13949, 13950, 13951, 13952, 13953, 13954, 13955, 13956, 13957, 139
58, 13959, 13960, 13961, 13962, 13963, 13964, 13965, 13966, 13967, 13968, 13969, 13970, 13971, 13972, 13973, 1397
4, 13975, 13976, 13977, 13978, 13979, 14779, 14780, 14781, 14782, 14783, 15665, 15666, 15667, 15668, 15669]
```

In [49]:

```
# Removing outliers
new_df = df.drop(df.index[outlier_index])
```

In [50]:

```
# Check rows in previous and new dataset
print(df.shape[0], new_df.shape[0])
```

```
18589 18341
```

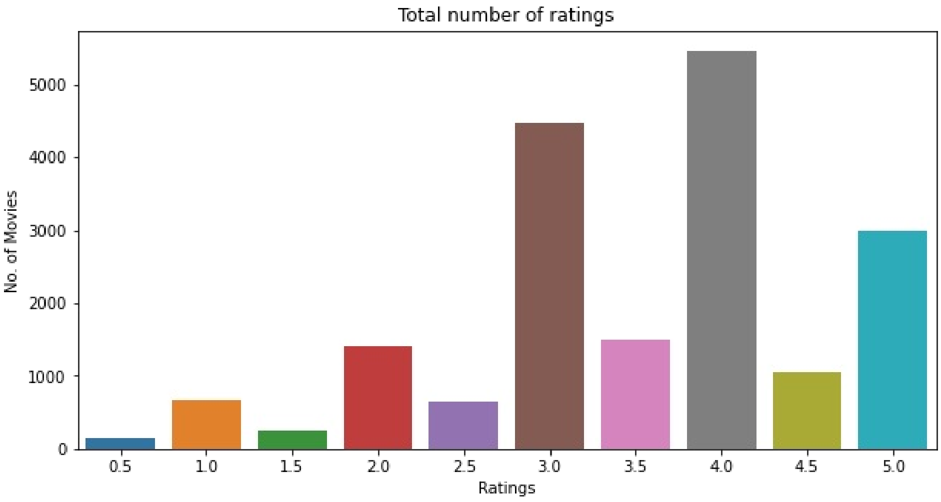
When the threshold value is taken as 3, there are 248 outliers removed from the dataset.

Univariate analysis

In [51]:

```
plt.figure(figsize=(10, 5))
sns.countplot(df['rating'])
plt.title('Total number of ratings')
plt.xlabel('Ratings')
plt.ylabel('No. of Movies')
```

Out[51]: Text(0, 0.5, 'No. of Movies')

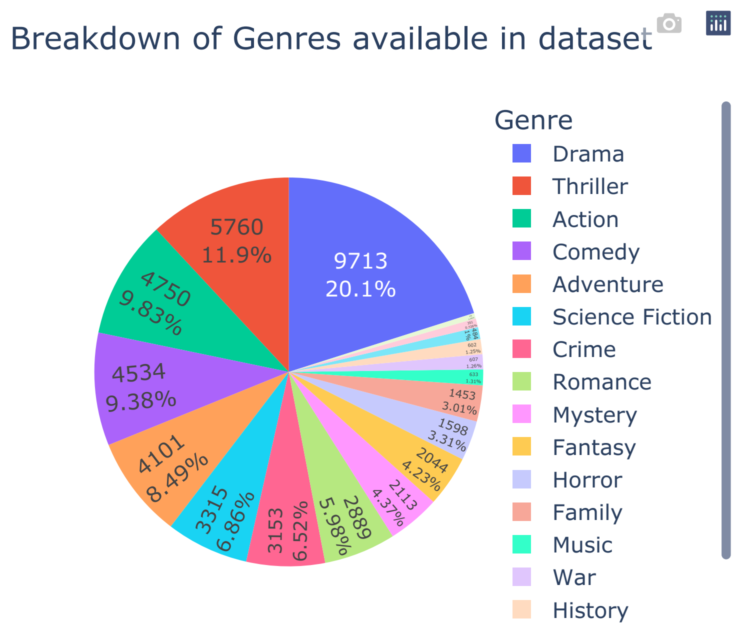


Through the bar graph, we can observe that most movies have gotten a rating of 4.0. We can also observe that the number of movies with the highest ratings are approximately 3000 while the number of movies with the lowest ratings are less than 1000.

In [52]:

```
import plotly.express as px
import plotly.graph_objects as go
from collections import Counter
genres = Counter()
for i in range(df.shape[0]):
    for j in df.genres[i]:
        genres[j]+=1
Genres = pd.DataFrame.from_dict(genres, orient='index').reset_index()
Genres = Genres.rename(columns = {'index': 'Genres' ,0: 'Total Number'})

Genres.loc[Genres['Total Number'] < 100, 'Genres'] = 'Others'
fig = px.pie(Genres, values='Total Number', names='Genres',width=500,height=500)
fig.update_layout(
    title="Breakdown of Genres available in dataset",
    legend_title="Genre",
    font=dict(
        size=14
    )
)
fig.layout.template = 'plotly'
fig.update_traces(textposition='inside', textinfo='value+percent')
fig.show()
```



Here is a breakdown of all the genres available in this dataset. The highest percentage is Drama at 20.1% with the total number of 9713 movies. Whereas the lowest percentage is Western at 0.455% with the total number of 220 movies.

Bivariate analysis

In [53]:
```
df['movie_id'].shape
```

Out[53]: (18589,)

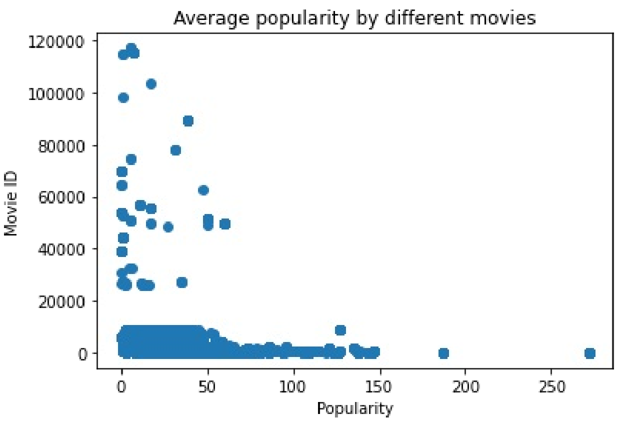
In [54]:
```
df['title'].shape
```

Out[54]: (18589,)

To make the visualization clear and understandable, we decided to use movie_id instead of title as the first variable. There is no issue using it interchangeably as every title has it's own movie_id.

In [55]:
```
y = df['movie_id']
x = df['popularity']
plt.scatter(x,y)
plt.title("Average popularity by different movies")
plt.ylabel('Movie ID', fontsize=10)
plt.xlabel('Popularity',fontsize=10)
```

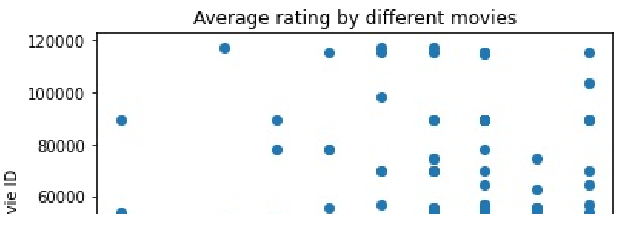
Out[55]: Text(0.5, 0, 'Popularity')

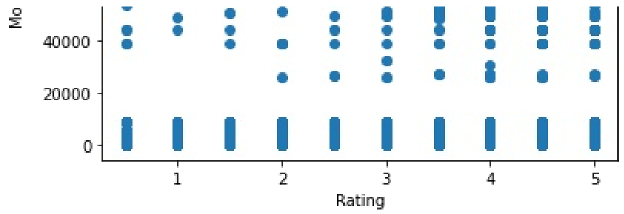


From the above scatter plot, we can observe that only a few movies are most popular. However, most movies are can be observed to be ranging from the popularity of 0 to 50.

In [56]:
```
y = df['movie_id']
x = df['rating']
plt.scatter(x,y)
plt.title("Average rating by different movies")
plt.ylabel('Movie ID', fontsize=10)
plt.xlabel('Rating',fontsize=10)
```

Out[56]: Text(0.5, 0, 'Rating')





From the above scatter plot, we can observe that there are many movies with ratings higher than 2. As oppose to the rating below 2, there are lesser movies.

Feature Selection

I will be using the filter method to reduce the dataset's dimensionality. Below, I have implemented the Pearson Correlation Heatmap to select the necessary features.

In [57]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.metrics import confusion_matrix
```

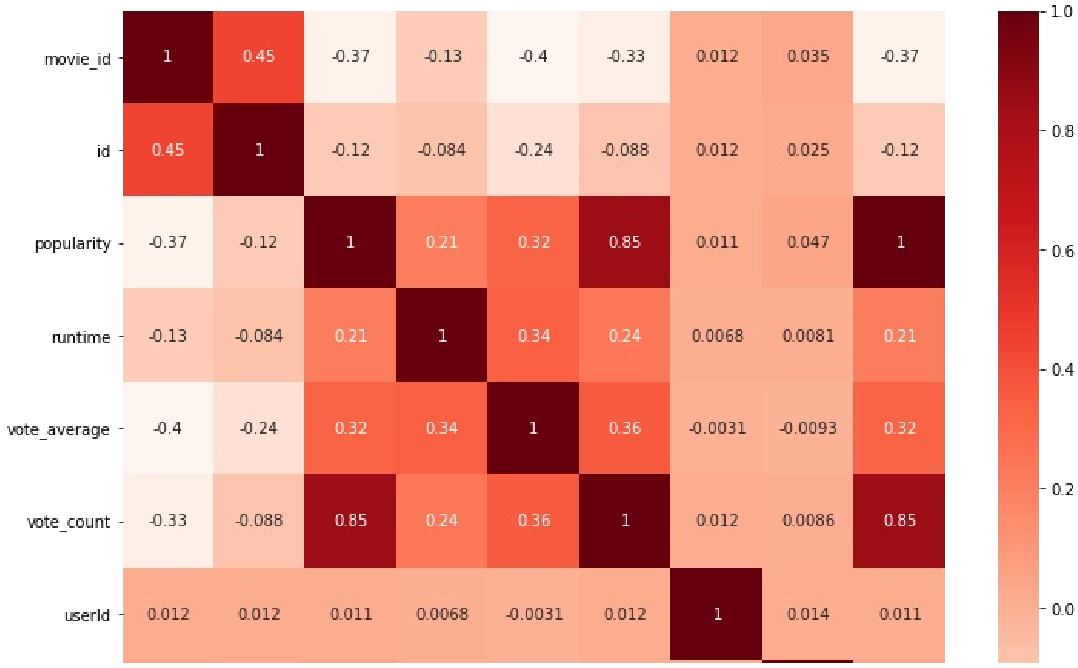
In [58]:
```
label_encoder = LabelEncoder()
df.iloc[:,0] = label_encoder.fit_transform(df.iloc[:,0]).astype('float64')
```

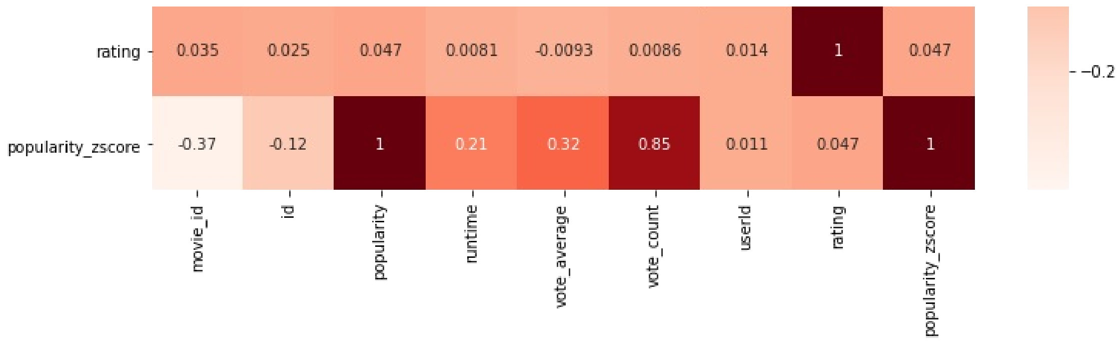
In [59]:
```
corr = df.corr()
corr
```

Out[59]:

	movie_id	id	popularity	runtime	vote_average	vote_count	userId	rating	popularity_zscore
movie_id	1.000000	0.446143	-0.367734	-0.127336	-0.399240	-0.333730	0.011780	0.034581	-0.367734
id	0.446143	1.000000	-0.115801	-0.083803	-0.235238	-0.087740	0.011517	0.024988	-0.115801
popularity	-0.367734	-0.115801	1.000000	0.214505	0.322582	0.850417	0.011450	0.047056	1.000000
runtime	-0.127336	-0.083803	0.214505	1.000000	0.343658	0.244022	0.006753	0.008089	0.214505
vote_average	-0.399240	-0.235238	0.322582	0.343658	1.000000	0.356024	-0.003090	-0.009336	0.322582
vote_count	-0.333730	-0.087740	0.850417	0.244022	0.356024	1.000000	0.012100	0.008568	0.850417
userId	0.011780	0.011517	0.011450	0.006753	-0.003090	0.012100	1.000000	0.013908	0.011450
rating	0.034581	0.024988	0.047056	0.008089	-0.009336	0.008568	0.013908	1.000000	0.047056
popularity_zscore	-0.367734	-0.115801	1.000000	0.214505	0.322582	0.850417	0.011450	0.047056	1.000000

In [60]:
```
#Using Pearson Correlation
plt.figure(figsize=(12,10))
sns.heatmap(corr, annot=True, cmap=plt.cm.Reds)
plt.show()
```





From the above scatter plot, we can deduce that vote_count and popularity are positively correlated (0.85) with each other. However, we have decided to take the rating variable. Thus, it will be the target for deciding the correlated variables.

In [61]:
```
#Correlation with output variable
corr_target = abs(corr["rating"])
#Selecting highly correlated features
relevant_features1 = corr_target[corr_target>0.01]
relevant_features1
```

Out[61]:
```
movie_id             0.034581
id                   0.024988
popularity           0.047056
userId               0.013908
rating               1.000000
popularity_zscore    0.047056
Name: rating, dtype: float64
```

In [62]:
```
print(df[["id","movie_id"]].corr())
```

```
                id  movie_id
id        1.000000  0.446143
movie_id  0.446143  1.000000
```

The above code shows that the variables movie_id and id are closely correlated with one another (0.446143). As a result, we would maintain one variable and discard the other. movie_id will be maintained since it's correlation to rating is higher than id. After eliminating the unnecessary variables, we are left with movie_id, popularity and userId and rating.

Model Development

In [63]:
```
import math
import re
from scipy.sparse import csr_matrix
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
sns.set_style("darkgrid")
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error,mean_absolute_error
```

In [64]:
```
df_status = df_categorical.drop(['cast','genres','overview'],axis=1)
```

In [65]:
```
#Meging the table with categorical value which is status
df_movies=df.merge(df_status,on='title')
df_movies.head()
```

Out[65]:

	movie_id	title	cast	genres	id	overview	popularity	runtime	status_x	vote_average	vote_count	userId	rating	popular
0	118.0	Pirates of the Caribbean: At World's End	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	285	Captain Barbossa, long believed to be dead, ha...	139.082615	169.0	Released	6.9	4500	39	4.0	
			[Tobey Maguire,			The seemingly								

1	206.0	Spider-Man 3	Kirsten Dunst, James Franco]	[Fantasy, Action, Adventure]	559	invincible Spider-Man goes up ag...	115.699814	139.0	Released	5.9	3576	492	5.0
2	280.0	Harry Potter and the Half-Blood Prince	[Daniel Radcliffe, Rupert Grint, Emma Watson]	[Adventure, Fantasy, Family]	767	As Harry begins his sixth year at Hogwarts, he...	98.885637	153.0	Released	7.4	5293	30	4.0
3	16.0	Pirates of the Caribbean: Dead Man's Chest	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	58	Captain Jack Sparrow works his way out of a bl...	145.847379	151.0	Released	7.0	5246	19	3.0
4	16.0	Pirates of the Caribbean: Dead Man's Chest	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	58	Captain Jack Sparrow works his way out of a bl...	145.847379	151.0	Released	7.0	5246	19	3.0

Content-Based Filtering

In [66]:

```
#drop unwanted columns for the model
df_movies=df_movies.drop(['id','status_x','popularity_zscore'],axis=1)
```

In [67]:

```
#Getting all numerical values only
df2_movies = df_movies.select_dtypes(exclude = 'object')
X = df2_movies.drop('rating',axis=1)
y = df2_movies['rating'].astype(int)
```

In [68]:

```
#Splitting into test and train dataset with 20% test size
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=10)
```

In [69]:

```
#ratings are in integers
print(ytrain.head())
```

```
279357     3
112083     4
1338120    3
1459856    5
770555     4
Name: rating, dtype: int32
```

In [70]:

```
#only numerical variables used
print(Xtrain.head())
```

```
         movie_id  popularity  runtime  vote_average  vote_count  userId  \
279357       61.0   60.929352    116.0           7.2        3783     311
112083      123.0   69.405188    109.0           5.9        2143     534
1338120     223.0   22.139842    129.0           8.0         657      85
1459856     179.0    8.109958    114.0           6.8          57     125
770555      620.0   14.503568     91.0           5.2         253     228

         status_y
279357          1
112083          1
1338120         1
1459856         1
770555          1
```

In [71]:

```
#Fitting the dataset into KNN model
knn=KNeighborsClassifier(n_neighbors=99)
knn_model=knn.fit(Xtrain,ytrain)
ypred = knn_model.predict(Xtest)
```

In [72]:

```
# Last row from the data will be used as test
```

```
display(df2_movies.iloc[-1:])
# Get movie id from last row
print("Validation set (Movie id): ", df2_movies['movie_id'].values[-1])
```

	movie_id	popularity	runtime	vote_average	vote_count	userId	rating	status_y
1567608	257.0	4.553644	93.0	6.2	110	247	2.0	1

```
Validation set (Movie id):  257.0
```

Showing the title for movie id 257 below

In [73]:

```
df_show= df_movies[(df_movies['movie_id']==257)]
df_show
```

Out[73]:

	movie_id	title	cast	genres	overview	popularity	runtime	vote_average	vote_count	userId	rating	status_y
1567608	257.0	Pink Flamingos	[Divine, David Lochary, Mary Vivian Pearce]	[Horror, Comedy, Crime]	Notorious Baltimore criminal and underground f...	4.553644	93.0	6.2	110	247	2.0	1

In [74]:

```
testset = df2_movies.iloc[-1:,:-2]
# validation set from the df2_movies table (excluding the last row)
Xval =  df2_movies.iloc[:-1,:-2]
yval = y.iloc[:-1]
val_knn = knn.fit(Xval, yval)
```

In [75]:

```
# finding distances between validation set and other movies based on their similar features
distances, indeces = val_knn.kneighbors(testset)
# create table distances and indeces from "Movie id=257"
distance_table = pd.DataFrame(val_knn.kneighbors(testset)[0][0], columns = ['distance'])
distance_table['index'] = val_knn.kneighbors(testset)[1][0]
distance_table.set_index('index').head(10) # distances between movies with validation set
```

Out[75]:

index	distance
800925	58.239125
800947	58.239125
800939	58.239125
800943	58.239125
800946	58.239125
800927	58.239125
800920	58.239125
800935	58.239125
800931	58.239125
800934	58.239125

The result below shows recommendations for Movie id=257 but only 2 movies are recommended.Maybe remaining was removed due to outliers in the data.

In [76]:

```
result = distance_table.join(df_movies,on='index')
result=result.sort_values('distance', ascending=True )
result = result.groupby('distance')
result_=result.first()
result_=result_.drop_duplicates(subset='movie_id', keep="first")
result_[['index','movie_id','title','genres','rating']] # showing recommended movies that has the nearest distanc
```

Out[76]:

distance	index	movie_id	title	genres	rating
58.239125	800925	306.0	Krull	[Fantasy, Action, Adventure]	4.0
62.751556	1307948	198.0	D.E.B.S.	[Action]	2.0

Collaborative Filtering

In [77]:

```
# Split into test and train dataset
df3_movies=df_movies.copy()
df3_movies= df3_movies.drop(['cast','overview','popularity'],axis=1)
train_data, test_data = train_test_split(df3_movies, test_size = 0.20)
print("Train size:", train_data.shape)
print("Test size:", test_data.shape)
```

```
Train size: (1254087, 9)
Test size: (313522, 9)
```

In [78]:

```
df3_movies.head()
```

Out[78]:

	movie_id	title	genres	runtime	vote_average	vote_count	userId	rating	status_y
0	118.0	Pirates of the Caribbean: At World's End	[Adventure, Fantasy, Action]	169.0	6.9	4500	39	4.0	1
1	206.0	Spider-Man 3	[Fantasy, Action, Adventure]	139.0	5.9	3576	492	5.0	1
2	280.0	Harry Potter and the Half-Blood Prince	[Adventure, Fantasy, Family]	153.0	7.4	5293	30	4.0	1
3	16.0	Pirates of the Caribbean: Dead Man's Chest	[Adventure, Fantasy, Action]	151.0	7.0	5246	19	3.0	1
4	16.0	Pirates of the Caribbean: Dead Man's Chest	[Adventure, Fantasy, Action]	151.0	7.0	5246	19	3.0	1

In [79]:

```
reader = Reader(rating_scale = (1, 5))
trainData = Dataset.load_from_df(train_data[['userId','movie_id','rating']], reader)
testData = Dataset.load_from_df(test_data[['userId', 'movie_id','rating']], reader)
```

In [80]:

```
#Build full trainset
trainset = trainData.build_full_trainset()
testset =  testData.build_full_trainset()
# Create the trainset and testset
data_trainset = trainset.build_testset()
data_testset =  testset.build_testset()
```

In [81]:

```
svd = SVD(n_factors=100, n_epochs=40, lr_all=0.005, reg_all=0.2, verbose=False)
```

In [82]:

```
#fitting SVD to train data
svd_model = svd.fit(trainset)
```

In [83]:

```
# Movies liked in past by user with ID=30
user_30 = df3_movies[(df3_movies['userId'] == 30) & (df3_movies['rating'] == 5.0)]
user_30 =user_30.set_index('movie_id')
user_30 = user_30.groupby('movie_id')
user_30_=user_30.first()
user_30_.head(10)
```

Out[83]:

movie_id	title	genres	runtime	vote_average	vote_count	userId	rating	status_y
5.0	Dancer in the Dark	[Drama, Crime, Music]	140.0	7.6	377	30	5.0	1
61.0	Ocean's Eleven	[Thriller, Crime]	116.0	7.2	3783	30	5.0	1
123.0	Terminator 3: Rise of the Machines	[Action, Thriller, Science Fiction]	109.0	5.9	2143	30	5.0	1
144.0	Alien	[Horror, Action, Thriller, Science Fiction]	117.0	7.9	4470	30	5.0	1
172.0	Romeo + Juliet	[Drama, Romance]	120.0	6.7	1374	30	5.0	1
187.0	Love Actually	[Comedy, Romance, Drama]	135.0	7.0	1869	30	5.0	1
188.0	Notting Hill	[Romance, Comedy, Drama]	124.0	7.0	1262	30	5.0	1
218.0	Silent Hill	[Horror, Mystery]	125.0	6.3	1067	30	5.0	1
219.0	The Hours	[Drama]	114.0	7.0	451	30	5.0	1
223.0	To Kill a Mockingbird	[Crime, Drama]	129.0	8.0	657	30	5.0	1

In [84]:

```
#Removing movies with least reviews or less vote count
```

```
f = ['count','mean']
movie_summary = df_movies.groupby('movie_id')['vote_average'].agg(f)
movie_summary.index = movie_summary.index.map(int)
movie_benchmark = round(movie_summary['count'].quantile(0.7),0)
drop_movie_list = movie_summary[movie_summary['count'] < movie_benchmark].index
print('Movie minimum times of review: {}'.format(movie_benchmark))
```

```
Movie minimum times of review: 361.0
```

In [85]:

```
# Getting shape of dataset after removing the rows with least reviews
print('Original Shape: {}'.format(df_movies.shape))
df_movies = df_movies[~df_movies['movie_id'].isin(drop_movie_list)]
print('After Trim Shape: {}'.format(df_movies.shape))
```

```
Original Shape: (1567609, 12)
After Trim Shape: (1537447, 12)
```

In [90]:

```
# Recommeding movies that user with ID=30 likes to watch
movie_30 = df3_movies.copy()
movie_30 = movie_30[~movie_30['movie_id'].isin(drop_movie_list)]
movie_30['Estimate_Score'] = movie_30['movie_id'].apply(lambda x: svd.predict(30, x).est)
movie_30 = movie_30.sort_values(by='Estimate_Score', ascending=True )
movie_30=movie_30.groupby('movie_id')
movie_30_=movie_30.first()
movie_30_.head(10)
```

Out[90]:

movie_id	title	genres	runtime	vote_average	vote_count	userId	rating	status_y	Estimate_Score
0.0	Four Rooms	[Crime, Comedy]	98.0	6.5	530	318	3.0	1	3.602157
1.0	Star Wars	[Adventure, Action, Science Fiction]	121.0	8.1	6624	311	5.0	1	4.180389
4.0	American Beauty	[Drama]	122.0	7.9	3313	15	2.5	1	3.691430
5.0	Dancer in the Dark	[Drama, Crime, Music]	140.0	7.6	377	221	3.0	1	4.029424
6.0	The Fifth Element	[Adventure, Fantasy, Action, Thriller, Science...	126.0	7.3	3885	571	5.0	1	3.575024
7.0	Metropolis	[Drama, Science Fiction]	153.0	8.0	657	500	3.0	1	3.184590
9.0	Pirates of the Caribbean: The Curse of the Bla...	[Adventure, Fantasy, Action]	143.0	7.5	6985	282	3.0	1	3.625370
10.0	Kill Bill: Vol. 1	[Action, Crime]	111.0	7.7	4949	185	4.0	1	3.457530
11.0	Jarhead	[Drama, War]	125.0	6.6	765	461	2.5	1	4.076173
16.0	Pirates of the Caribbean: Dead Man's Chest	[Adventure, Fantasy, Action]	151.0	7.0	5246	19	3.0	1	4.210848

The table shows movie that will be liked by user with id=30. Estimate_score column shows the estimated value that was given by SVD, to see if it gives the correct prediction with actual rating of the movies.

Model Evaluation

KNN model evaluation

In [87]:

```
# Test Data
print('Mean Absolute Error:', mean_absolute_error(ytest, ypred))
print('Mean Squared Error:', mean_squared_error(ytest, ypred))
print('Root Mean Squared Error:', np.sqrt(mean_squared_error(ytest, ypred)))
```

```
Mean Absolute Error: 0.11917823948558634
Mean Squared Error: 0.2277958165615172
Root Mean Squared Error: 0.47727959998466013
```

SVD model evaluation

In [88]:

```
#Test Data
cross_validate(svd,testData, measures=['RMSE','MAE','MSE'])
```

Out[88]:

```
{'test_rmse': array([0.75640783, 0.74949944, 0.7572576 , 0.75110691, 0.75329323]),
```

```
 'test_mae': array([0.58597646, 0.58243844, 0.5875602 , 0.58209875, 0.58362724]),
 'test_mse': array([0.57215281, 0.56174942, 0.57343907, 0.56416159, 0.56745069]),
 'fit_time': (39.32354211807251,
  36.401416540145874,
  39.82191061973572,
  42.25337862968445,
  40.02106475830078),
 'test_time': (3.6417415142059326,
  0.9060494899749756,
  1.009962797164917,
  0.6564834117889404,
  0.9732513427734375)}
```

In [92]:

```python
#Mean value of Root Mean Square Error,Mean Absolute Error and Mean Squared Error
rmse2=[0.75640783, 0.74949944, 0.7572576 , 0.75110691, 0.75329323]
mae2=[0.58597646, 0.58243844, 0.5875602 , 0.58209875, 0.58362724]
mse2=[0.57215281, 0.56174942, 0.57343907, 0.56416159, 0.56745069]
rmse2 = pd.Series(rmse2)
mae2 = pd.Series(mae2)
mse2 = pd.Series(mse2)
print('Mean Absolute Error:', mae2.mean())
print('Mean Squared Error:', mse2.mean())
print('Root Mean Squared Error:', rmse2.mean())
```

```
Mean Absolute Error: 0.584340218
Mean Squared Error: 0.567790716
Root Mean Squared Error: 0.7535130019999999
```

Lower MSE,RMSE,MAE the better the model fits. It means the predicted values are closer to actual values. KNN model has smaller MSE,RMSE and MAE compared to SVD model, but this does not mean that KNN can give accurate movie recommendations for content-based learning.This is because the movies recommended by content-based learning using KNN does not seem similar with the validation movie set.Most of the movies recommended using collaborative filtering are showing similaritites with the selected movie.

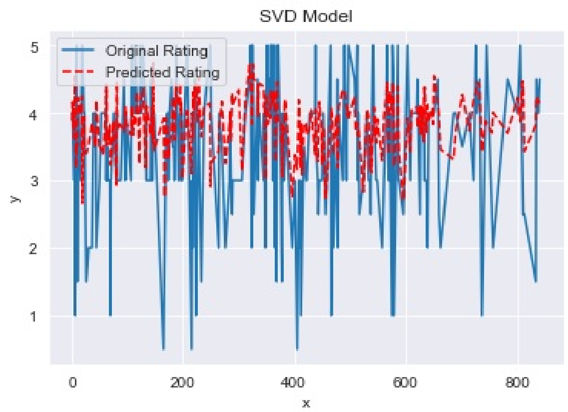
Visualization and Communication

Discussing the results for collaborative filtering model

The graph below shows the actual and predicted ratings of the model.

In [90]:

```python
fig, ax = plt.subplots()
original=movie_30_['rating']
predicted=movie_30_['Estimate_Score']
ax.plot(original, label="Original Rating")
ax.plot(predicted,color='r', ls='--', label="Predicted Rating")
ax.legend(loc=2); # upper left corner
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('SVD Model');
```



It shows that the model can give almost accurate results for ratings of the movies.The prediction using SVD shown below

In [91]:

```python
svd.predict(30,1.0,5.0)  # svd predicts score of user 30 for movie id=5
```

Out[91]: Prediction(uid=30, iid=1.0, r_ui=5.0, est=4.180389157286113, details={'was_impossible': False})

User id=311 gave a rating of 5.0 for the movie id=1. When predicting the rating for user id=30 with the same movie, the estimated rating is 4.18 which is quite close with the rating given by user 311. This shows that the system is giving accurate movie recommendations because same type of users should like the same type of movies.By looking at the estimated score, we can conclude that user 30 will give almost similar ratings with user 311 and also the other users from the movie*30* table.

Below shown discussion for movies recommended by the collaborative filtering and content-based filtering model

In [92]:
```
def countGenre(char, list): # function to count occurences of each genres
   return sum([i.count(char) for i in list])
```

In [93]:
```
#For content-based
user30_genres= user_30_['genres']
userList_genres = user30_genres.tolist() #converting arrays to lists
```

In [94]:
```
Drama=countGenre('Drama',userList_genres); Adventure=countGenre('Adventure',userList_genres);Action=countGenre('A
Crime=countGenre('Crime',userList_genres);Fantasy=countGenre('Fantasy',userList_genres);Sf=countGenre('Science Fi
Comedy=countGenre('Comedy',userList_genres);Music=countGenre('Music',userList_genres);Animation=countGenre('Anima
War=countGenre('War',userList_genres);Family=countGenre('Family',userList_genres);Romance=countGenre('Romance',us
Horror=countGenre('Horror',userList_genres);Mystery=countGenre('Mystery',userList_genres);Thriller=countGenre('Th
Western=countGenre('Western',userList_genres);History=countGenre('History',userList_genres)
```

In [95]:
```
#For collaborative filtering
movie30_genres= movie_30_['genres']
list2_genres = movie30_genres.tolist() #converting arrays to lists
```

In [96]:
```
Drama2=countGenre('Drama',list2_genres); Adventure2=countGenre('Adventure',list2_genres);Action2=countGenre('Acti
Crime2=countGenre('Crime',list2_genres);Fantasy2=countGenre('Fantasy',list2_genres);Sf2=countGenre('Science Ficti
Comedy2=countGenre('Comedy',list2_genres);Music2=countGenre('Music',list2_genres);Animation2=countGenre('Animatic
War2=countGenre('War',list2_genres);Family2=countGenre('Family',list2_genres);Romance2=countGenre('Romance',list2
Horror2=countGenre('Horror',list2_genres);Mystery2=countGenre('Mystery',list2_genres);Thriller2=countGenre('Thril
Western2=countGenre('Western',list2_genres);History2=countGenre('History',list2_genres)
```

In [97]:
```
drama_col=[Drama,Drama2];ad_col=[Adventure,Adventure2];ac_col=[Action,Action2];cr_col=[Crime,Crime2];f_col=[Fanta
sf_col=[Sf,Sf2];co_col=[Comedy,Comedy2];mu_col=[Music,Music2];an_col=[Animation,Animation2];war_col=[War,War2];fa
rom_col=[Romance,Romance2];hor_col=[Horror,Horror2];mys_col=[Mystery,Mystery2];thr_col=[Thriller,Thriller2];wes_c
his_col=[History,History2]
```

In [98]:
```
Genres=['Liked by User','Prediction']
CL_table = pd.DataFrame({'Genres':Genres,'Drama':drama_col,'Adventure':ad_col,'Action':ac_col,'Crime':cr_col,'Fan
                         'Music':mu_col,'Animation':an_col,'Romance':rom_col,'Horror':hor_col,'War':war_col,'Fami
                         'Mystery':mys_col,'Thriller':thr_col,'Western':wes_col,'History':his_col})
print("Collaborative Filtering Genres Table")
CL_table
```

```
Collaborative Filtering Genres Table
```

Out[98]:

	Genres	Drama	Adventure	Action	Crime	Fantasy	Sf	Comedy	Music	Animation	Romance	Horror	War	Family	Mystery	Thriller	We
0	Liked by User	20	7	7	7	6	7	9	2	2	7	7	1	3	2	9	
1	Prediction	137	56	68	44	31	46	64	9	7	48	17	11	22	25	84	

This collaborative filtering gets the movies liked by a user. Then the recommendation system gets users with similar interests with the selected user and displays the movies that liked by the similar users. Since, they have similar interests , the model will most likely display movie names that will be liked by the selected user. The table above shows the genres liked by user with user id=30 and the prediction made by the model.The first row shows that the user likes movie with Drama genre the most, followed by Comedy and Thriller. The model also recommended Drama genre the most, with 137 movies and second highest is the Thriller.It also recommended quite a number of Comedy movies. This shows that the model is giving accurate results by recommending movies that will be liked by the user.It seems that collaborative filtering model gives more accuracte results compared to content-based learning.

In [99]:
```
#Content-based learning
df_genre = df3_movies[(df3_movies['movie_id'] == 257)]#Comparing the genres of validation set with recommended mo
df_genre.head(1)
```

Out[99]:

movie_id	title	genres	runtime	vote_average	vote_count	userId	rating	status_y

1567608	257.0	Pink Flamingos	[Horror, Comedy, Crime]	93.0	6.2	110	247	2.0	1

In [100…

```
#Listing the genres
result_genres= result_['genres']
list_genres = result_genres.tolist()
print(list_genres)
```

```
[['Fantasy', 'Action', 'Adventure'], ['Action']]
```

In [101…

```
drama= countGenre('Drama',list_genres) ; action = countGenre('Action',list_genres);comedy= countGenre('Comedy',li
music= countGenre('Music',list_genres); fantasy = countGenre('Fantasy',list_genres);thriller=countGenre('Thriller
history=countGenre('History',list_genres);
print("Drama:",drama," Action:",action, " Comedy:",comedy," Music:", music," Fantasy:",fantasy," Thriller:",thril
      history)
```

```
Drama: 0  Action: 2  Comedy: 0  Music: 0  Fantasy: 1  Thriller: 0  History: 0
```

One movie that was selected for the model is Pink Flamingos with movie id=257. The movie's genres are Horror,Comedy and Crime.By counting movies that have smaller distance with Pink Flamingos, we obtain the results for movies that most likely will be liked by users that liked Pink Flamingos. The movie with nearest distance to Pink Flamingos is Krull and D.E.B.S. Content-based learning model only gave 2 movies as recommendation and it is recommending movies with different genres. None of the movie has the same genre as Pink Flamingos. This shows that content-based learning model is not so accuracte compared to collabortive filtering model.

Deploy and Maintain

Before we deploy the movie recommender system, we first need to test whether the system works as intended to make sure that it is free from defects. There are three steps when testing is carried out. Firstly, we need to do a review of the requirements and test the usability of the system. For instance, we will need several users to use the recommender system and see if the movies are accurately matched to the users. When it is accurately matched, we will need to see whether the subsystems of the tested components work together. In this case, users can try rating the movies they watched and see how the recommender system filters the movie and suggests them movies based on their likes. Now, since we have a functional recommender system, validation is needed in order for the system to be optimized for performance and being stable for a period of time. This is to check the accuracy of the data as incorrect data can negatively impact the system in making inaccurate results. The process of data validation is also known as data cleaning. Since the movie recommender uses a large amount of data, it needs a sample rather than a complete dataset. We need to decide the volume of the data sample and find the error rate to ensure the system works. When the errors are identified, improvement can be made to fix the system components. The final step in the deployment process is streamlining. Since the environment changes throughout the model, its accuracy will fall. In order to understand the relation between accuracy and environment, we need to automate and streamline the process, decrease the deployment time and ensure that we are using a quality data sample.

Limitations

To build an efficient recommender system a hybrid combination of different methods of recommendation is must. It is concluded that by using a combination of similarity measures, a better user similarity can be generated rather than using a single similarity measure and efficiency of the system is also increased. The author stated that the dataset used in the model test in this paper is very small compared with the calculation required for actual application, and there is still a certain gap in the amount of data. Accuracy of the movie recommender system can be improved if we add extra movie features. Generally, most of the papers have shown the combination of collaborative filtering and content-based filtering. Another limitation is that the search engine design will directly show the output rather than showing the list of movie recommendations which we all know the process is very time consuming. Hence, designing the information of all the movies present till date will not be possible. This explains why the recommendation system does not use machine learning as their domain.

For future studies, the movie recommendation system can be designed better once the user recommendations could be obtained from users' search history, thus, machine learning will be used in order to obtain the results of the user based search history. In addition, we will use deep-learning based approaches to apply movie recommendations as various studies have been inspired by multimodal-based approaches with efficient deep-learning and BigDL framework. The movie recommendation system also needs to be deployed in a real-life scenario to receive manual responses of accuracy to judge the full extent of the system.

Conclusion

In our research, we provided two types of recommendation models, which are content-based learning and collaborative-filtering. Our problem statement discusses the issues of not having enough user data, such as ratings to predict the movies. This is because not everyone will give their ratings after watching a movie. From our experiment, it is shown that a collaborative filtering model could solve this issue because the SVD algorithm gives estimated ratings for each movie which helps the system to recommend movies with high estimated ratings. The SVD algorithm also gives the nearest estimation to the real ratings for the movies. Content-based learning model does not give rating estimations, and it predicts movies by calculating the nearest distance between the movies.Collaborative-filtering model in our experiment gives estimated ratings and also gives the correct movie predictions.When we did genre comparisons for movies liked by user with the predicted movies, we can see that the recommender system gives movies with the genre liked most by the user.Thus, we can prove that collaborative filtering models can solve the issue addressed in our problem statement.