

# Movie Recommendation System



## Business Understanding

### 1.1) Overview

The vast and fast growing movie industry may be overwhelming for a movie platform user to decide on which movie to watch and actually enjoy it. This project aims to curb this challenge by developing a personalized movie recommendation system by leveraging a dataset containing movie information, user ratings, and interactions. The system will utilize content-based and collaborative filtering techniques to suggest movies to users based on their preferences and past interactions.

- Modules for movie and user profiling will be developed to analyze movie attributes and user preferences, respectively. Evaluation and optimization will be conducted to enhance the accuracy
- Furthermore, leveraging movie metadata, user ratings, and collaborative filtering algorithms, the system provides personalized movie recommendations to enhance the user's movie-watching experience.

### 1.2) Problem Statement

- The movie industry is vast and fast evolving, with countless movies and movie sequels released each year hence can be a challenge for the users to navigate through the vast amount of content and get to know which movies align with their preferences.

- To ease this, or rather improve the users' experience, we come up with a recommendation system that provides personalized movie recommendations based on user preferences and similarities with other users, and also aim to improve user satisfaction, increase user engagement, which ultimately drive user's retention on the platform.

## 1.3) Objectives

### 1.3.1) Specific Objectives

- To develop a demographic recommendation system that suggests popular movies based on user demographic attributes.
- To implement a content-based recommendation system that recommends movies based on movie overviews, cast, and keywords.
- To build a collaborative filtering recommendation system that suggests movies based on user similarities and their ratings.
- To create a hybrid recommendation system that combines the techniques from content-based and collaborative filtering approaches to provide personalized movie recommendations.

## Data Understanding

- TMDB is a popular database that provides comprehensive information about movies, that contains the following titles, release dates, genres, cast and crew information. Credit information is given as well about the cast and crew information whereby the cast and crew are involved in each movie. With the combination of the datasets, we gain valuable insights and perform various analyses related to the movie industry. you can get the data via API through: kaggle datasets download -d tmdb/tmdb-movie-metadata or using the following link:  
<https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata>

The dataset columns represent:

- **id**: Unique identifier for each movie
- **title**: Title of the movie
- **cast**: List of actors/actresses in the movie
- **crew**: List of crew members involved in the movie
- **budget**: Budget of the movie
- **genres**: List of genres associated with the movie
- **homepage**: Website URL of the movie
- **keywords**: List of keywords associated with the movie
- **original\_language**: Original language of the movie
- **original\_title**: Original title of the movie
- **production\_companies**: List of production companies involved in the movie
- **production\_countries**: List of countries where the movie was produced
- **release\_date**: Release date of the movie

- revenue: Revenue generated by the movie
- runtime: Duration of the movie in minutes
- spoken\_languages: List of languages spoken in the movie
- status: Current status of the movie (e.g., Released, Post Production)
- tagline: Tagline or slogan of the movie
- vote\_average: Average vote rating for the movie
- vote\_count: Number of votes received by the movie
- tags: List of tags associated with the movie
- Ratings Data File Structure (ratings.csv)

All ratings are contained in the file *ratings.csv*. Each line of this file after the header row represents one rating of one movie by one user, and has the following format:

userId, movieId, rating, timestamp

The lines within this file are ordered first by userId, then, within user, by movieId.

Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars).

Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

This dataset (ml-latest-small) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. These data were created by 610 users between March 29, 1996 and September 24, 2018. This dataset was generated on September 26, 2018.

Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

The data are contained in the files links.csv, movies.csv, ratings.csv and tags.csv. More details about the contents and use of all these files follows.

This is a development dataset. As such, it may change over time and is not an appropriate dataset for shared research results. See available benchmark datasets if that is your intent.

This and other GroupLens data sets are publicly available for download at <http://grouplens.org/datasets/>. we handled the separate file that is provided in the github repository

*Import/ Load the libraries required*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import re
```

```

import ast
import json
from collections.abc import Iterable

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import PrecisionRecallDisplay,
mean_squared_error, precision_recall_fscore_support,
precision_recall_curve
from sklearn.pipeline import Pipeline

from wordcloud import WordCloud

from surprise import SVD, Reader, Dataset
from surprise.model_selection import cross_validate, train_test_split,
GridSearchCV
from surprise import KNNWithMeans
from surprise import accuracy

from nltk import PorterStemmer

from my_functions import DatasetInfo, movie_score,
get_user_recommendations, recommend_movies, DataFrameFiller,
recommended_movies, update_crew_with_director, create_soup, stem,
recommended_movies, recommend, hybrid_recommendations

import warnings
warnings.filterwarnings('ignore', category=UserWarning,
module='IPython')

```

## Load the Datasets

- Movie Lens Dataset

```

new_data = pd.read_csv(r".data/movies_lens.csv")
new_data.head()

```

	movieId		title	genres	userId	rating	\
0	3	Grumpier Old Men (1995)	Comedy Romance	1.0	4.0		
1	3	Grumpier Old Men (1995)	Comedy Romance	1.0	4.0		
2	3	Grumpier Old Men (1995)	Comedy Romance	1.0	4.0		
3	3	Grumpier Old Men (1995)	Comedy Romance	1.0	4.0		
4	3	Grumpier Old Men (1995)	Comedy Romance	1.0	4.0		

	date	time	sentiment	\
0	2000-07-30	18:20:47	Positive	
1	2000-07-30	18:20:47	Positive	
2	2000-07-30	18:20:47	Positive	
3	2000-07-30	18:20:47	Positive	

4 2000-07-30 18:20:47 Positive

		review	tag
top_critic \			
0	A distinctly gallows take on contemporary fina...	moldy	
0.0			
1	It's an allegory in search of a meaning that n...	moldy	
0.0			
2	... life lived in a bubble in financial dealin...	moldy	
0.0			
3	Continuing along a line introduced in last yea...	moldy	
0.0			
4	... a perverse twist on neorealism...	moldy	
0.0			

	publisher	date.1
0	Patrick Nabarro	November 10, 2018
1	io9.com	May 23, 2018
2	Stream on Demand	January 4, 2018
3	MUBI	November 16, 2017
4	Cinema Scope	October 12, 2017

Applying the DataInfo object on our data

```
new_data_info = DatasetInfo(new_data)
new_data_info
```

```
<my_functions.DatasetInfo at 0x2244525f090>
```

```
new_data_info.check_dataset_info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11970 entries, 0 to 11969
Data columns (total 13 columns):
#   Column          Non-Null Count  Dtype
---  -
0   movieId         11970 non-null  int64
1   title           11970 non-null  object
2   genres          11970 non-null  object
3   userId          11970 non-null  float64
4   rating          11970 non-null  float64
5   date            11970 non-null  object
6   time            11970 non-null  object
7   sentiment       11970 non-null  object
8   review          11970 non-null  object
9   tag             11970 non-null  object
10  top_critic      11970 non-null  float64
11  publisher       11970 non-null  object
12  date.1          11970 non-null  object
dtypes: float64(3), int64(1), object(9)
memory usage: 1.2+ MB
```

```
new_data.isnull().sum()
```

```
movieId      0
title        0
genres       0
userId       0
rating       0
date         0
time         0
sentiment    0
review       0
tag          0
top_critic   0
publisher    0
date.1       0
dtype: int64
```

```
new_data_info.check_dataset_shape()
```

```
Dataset shape: (11970, 13)
```

```
new_data.drop_duplicates()
```

```
      movieId      title      genres  userId
rating \
0      3  Grumpier Old Men (1995)  Comedy|Romance    1.0
4.0
1      3  Grumpier Old Men (1995)  Comedy|Romance    1.0
4.0
2      3  Grumpier Old Men (1995)  Comedy|Romance    1.0
4.0
3      3  Grumpier Old Men (1995)  Comedy|Romance    1.0
4.0
4      3  Grumpier Old Men (1995)  Comedy|Romance    1.0
4.0
...      ...      ...      ...      ...      ..
.
11965   1997   Exorcist, The (1973)  Horror|Mystery    51.0
5.0
11966   1997   Exorcist, The (1973)  Horror|Mystery    51.0
5.0
11967   1997   Exorcist, The (1973)  Horror|Mystery    51.0
5.0
11968   1997   Exorcist, The (1973)  Horror|Mystery    51.0
5.0
11969   1997   Exorcist, The (1973)  Horror|Mystery    51.0
5.0

      date      time  sentiment \
0  2000-07-30  18:20:47  Positive
1  2000-07-30  18:20:47  Positive
```

2	2000-07-30	18:20:47	Positive
3	2000-07-30	18:20:47	Positive
4	2000-07-30	18:20:47	Positive
...	...	...	...
11965	2009-01-02	21:39:00	Positive
11966	2009-01-02	21:39:00	Positive
11967	2009-01-02	21:39:00	Positive
11968	2009-01-02	21:39:00	Positive
11969	2009-01-02	21:39:00	Positive

		review	tag
top_critic \			
0	A distinctly gallows take on contemporary fina...		moldy
0.0			
1	It's an allegory in search of a meaning that n...		moldy
0.0			
2	... life lived in a bubble in financial dealin...		moldy
0.0			
3	Continuing along a line introduced in last yea...		moldy
0.0			
4	... a perverse twist on neorealism...		moldy
0.0			
...		...	...
...			
11965	A respectably funny -- but small -- effort, a ...		demons
1.0			
11966	Curtin and Aykroyd rely on old standbys... Whe...		demons
1.0			
11967	A thoroughly updated and nicely fleshed-out co...		demons
0.0			
11968	The film itself is like the cinematic equivale...		demons
0.0			
11969	A new spin on the original skit with some fres...		demons
0.0			

		publisher	date.1
0		Patrick Nabarro	November 10, 2018
1		io9.com	May 23, 2018
2		Stream on Demand	January 4, 2018
3		MUBI	November 16, 2017
4		Cinema Scope	October 12, 2017
...		...	...
11965		Washington Post	January 1, 2000
11966		Washington Post	January 1, 2000
11967		Austin Chronicle	January 1, 2000
11968	Deseret News (Salt Lake City)		January 1, 2000
11969		Filmcritic.com	January 1, 2000

[11970 rows x 13 columns]

```
new_data = new_data.rename(columns={"userId_x":"userId", "rating_x":  
"rating", "timestamp_x":"timestamp", "date_x": "date", "time_x":  
"time", "sentiment_x": "sentiment"})
```

```
new_data.dropna(subset=['title', 'userId'], inplace=True)
```

```
new_data.info.check_dataset_shape()
```

```
Dataset shape: (11970, 13)
```

```
columns = ['review', 'tag', 'top_critic', 'publisher']
```

```
df_filler = DataFrameFiller(new_data)
```

```
new_data = df_filler.fillna_random(columns)
```

```
new_data.isnull().sum()
```

```
movieId      0  
title        0  
genres       0  
userId       0  
rating       0  
date         0  
time         0  
sentiment    0  
review       0  
tag          0  
top_critic   0  
publisher    0  
date.1       0  
dtype: int64
```

```
new_data['publisher'] = new_data['publisher'].fillna('').astype(str)
```

```
new_data['genres'] = new_data['genres'].fillna('').astype(str)
```

```
new_data['review'] = new_data['review'].fillna('').astype(str)
```

```
new_data['top_critic'] = new_data['top_critic'].fillna('').astype(str)
```

```
new_data['sentiment'] = new_data['sentiment'].fillna('').astype(str)
```

```
new_data['tag'] = new_data['publisher'] + new_data['genres'] +
```

```
new_data['review'] + new_data['top_critic'] + new_data['sentiment']
```

```
# Lambda Function to turn the strings to lower case and remove  
separators(|, (), ', ', '.')
```

```
new_data['tag'] = new_data['tag'].apply(lambda x: re.sub(r'[|() ,\d ]',  
+', ', x.lower()))
```

```
new_data['tag'][14]
```

```
'movie habitcomedyromancefor those who like their cronenberg thick and  
chewypositive'
```

```
new_data['title'] = new_data['title'].apply(lambda x: x.split('(')  
[0].strip())
```

```
new_data['title']
```



```

0      Grumpier Old Men
1      Grumpier Old Men
2      Grumpier Old Men
3      Grumpier Old Men
4      Grumpier Old Men
...
11965   Exorcist, The
11966   Exorcist, The
11967   Exorcist, The
11968   Exorcist, The
11969   Exorcist, The
Name: title, Length: 11970, dtype: object

```

```
new_data['review'] = new_data['review'].str.lower()
```

- Movie Credits Dataset

```

tmdb_movie_credits = pd.read_csv(r".data/tmdb_5000_credits.csv")
tmdb_movie_credits

```

```

      movie_id      title \
0      19995      Avatar
1      285  Pirates of the Caribbean: At World's End
2     206647      Spectre
3      49026  The Dark Knight Rises
4      49529  John Carter
...
4798     9367      El Mariachi
4799     72766      Newlyweds
4800     231617  Signed, Sealed, Delivered
4801     126186  Shanghai Calling
4802     25975  My Date with Drew

```

```

      cast \
0  [{"cast_id": 242, "character": "Jake Sully", "...
1  [{"cast_id": 4, "character": "Captain Jack Spa...
2  [{"cast_id": 1, "character": "James Bond", "cr...
3  [{"cast_id": 2, "character": "Bruce Wayne / Ba...
4  [{"cast_id": 5, "character": "John Carter", "c...
...
4798 [{"cast_id": 1, "character": "El Mariachi", "c...
4799 [{"cast_id": 1, "character": "Buzzy", "credit_...
4800 [{"cast_id": 8, "character": "Oliver 0\u2019To...
4801 [{"cast_id": 3, "character": "Sam", "credit_id...
4802 [{"cast_id": 3, "character": "Herself", "credi...

```

```

      crew
0  [{"credit_id": "52fe48009251416c750aca23", "de...
1  [{"credit_id": "52fe4232c3a36847f800b579", "de...
2  [{"credit_id": "54805967c3a36829b5002c41", "de...
3  [{"credit_id": "52fe4781c3a36847f81398c3", "de...

```

```

4      [{"credit_id": "52fe479ac3a36847f813eaa3", "de...
...
4798 [{"credit_id": "52fe44eec3a36847f80b280b", "de...
4799 [{"credit_id": "52fe487dc3a368484e0fb013", "de...
4800 [{"credit_id": "52fe4df3c3a36847f8275ecf", "de...
4801 [{"credit_id": "52fe4ad9c3a368484e16a36b", "de...
4802 [{"credit_id": "58ce021b9251415a390165d9", "de...

```

[4803 rows x 4 columns]

-Movies Dataset

```

tmdb_movies = pd.read_csv(r".data/tmdb_5000_movies.csv")
tmdb_movies

```

```

      budget      genres \
0      237000000 [{"id": 28, "name": "Action"}, {"id": 12, "nam...
1      300000000 [{"id": 12, "name": "Adventure"}, {"id": 14, "...
2      245000000 [{"id": 28, "name": "Action"}, {"id": 12, "nam...
3      250000000 [{"id": 28, "name": "Action"}, {"id": 80, "nam...
4      260000000 [{"id": 28, "name": "Action"}, {"id": 12, "nam...
...
4798      220000 [{"id": 28, "name": "Action"}, {"id": 80, "nam...
4799      9000 [{"id": 35, "name": "Comedy"}, {"id": 10749, "...
4800      0 [{"id": 35, "name": "Comedy"}, {"id": 18, "nam...
4801      0 []
4802      0 [{"id": 99, "name": "Documentary"}]

```

```

      homepage      id \
0      http://www.avatarmovie.com/ 19995
1      http://disney.go.com/disneypictures/pirates/ 285
2      http://www.sonypictures.com/movies/spectre/ 206647
3      http://www.thedarkknighttrises.com/ 49026
4      http://movies.disney.com/john-carter 49529
...
4798      NaN 9367
4799      NaN 72766
4800      http://www.hallmarkchannel.com/signedsealeddel... 231617
4801      http://shanghaicalling.com/ 126186
4802      NaN 25975

```

```

      keywords
original_language \
0      [{"id": 1463, "name": "culture clash"}, {"id":...
en
1      [{"id": 270, "name": "ocean"}, {"id": 726, "na...
en
2      [{"id": 470, "name": "spy"}, {"id": 818, "name...
en
3      [{"id": 849, "name": "dc comics"}, {"id": 853,...
en

```

```

4      [{"id": 818, "name": "based on novel"}, {"id": ...
en
...
...
4798 [{"id": 5616, "name": "united states\u2013mexi...
es
4799
en
4800 [{"id": 248, "name": "date"}, {"id": 699, "nam...
en
4801
en
4802 [{"id": 1523, "name": "obsession"}, {"id": 224...
en

```

```

                                original_title \
0                                Avatar
1      Pirates of the Caribbean: At World's End
2                                Spectre
3                                The Dark Knight Rises
4                                John Carter
...
4798                                El Mariachi
4799                                Newlyweds
4800                                Signed, Sealed, Delivered
4801                                Shanghai Calling
4802                                My Date with Drew

```

```

                                overview popularity \
0      In the 22nd century, a paraplegic Marine is di... 150.437577
1      Captain Barbossa, long believed to be dead, ha... 139.082615
2      A cryptic message from Bond's past sends him o... 107.376788
3      Following the death of District Attorney Harve... 112.312950
4      John Carter is a war-weary, former military ca... 43.926995
...
4798      El Mariachi just wants to play his guitar and ... 14.269792
4799      A newlywed couple's honeymoon is upended by th... 0.642552
4800      "Signed, Sealed, Delivered" introduces a dedic... 1.444476
4801      When ambitious New York attorney Sam is sent t... 0.857008
4802      Ever since the second grade when he first saw ... 1.929883

```

```

                                production_companies \
0      [{"name": "Ingenious Film Partners", "id": 289...
1      [{"name": "Walt Disney Pictures", "id": 2}, {"...
2      [{"name": "Columbia Pictures", "id": 5}, {"nam...
3      [{"name": "Legendary Pictures", "id": 923}, {"...
4      [{"name": "Walt Disney Pictures", "id": 2}]
...
4798      [{"name": "Columbia Pictures", "id": 5}]
4799

```

```

4800 [{"name": "Front Street Pictures", "id": 3958}...
4801                                     []
4802 [{"name": "rusty bear entertainment", "id": 87...

```

```

                                production_countries
release_date \
0 [{"iso_3166_1": "US", "name": "United States o... 2009-12-10
1 [{"iso_3166_1": "US", "name": "United States o... 2007-05-19
2 [{"iso_3166_1": "GB", "name": "United Kingdom"... 2015-10-26
3 [{"iso_3166_1": "US", "name": "United States o... 2012-07-16
4 [{"iso_3166_1": "US", "name": "United States o... 2012-03-07
...                                     ...
4798 [{"iso_3166_1": "MX", "name": "Mexico"}, {"iso... 1992-09-04
4799                                     [] 2011-12-26
4800 [{"iso_3166_1": "US", "name": "United States o... 2013-10-13
4801 [{"iso_3166_1": "US", "name": "United States o... 2012-05-03
4802 [{"iso_3166_1": "US", "name": "United States o... 2005-08-05

```

```

        revenue runtime
spoken_languages \
0 2787965087 162.0 [{"iso_639_1": "en", "name": "English"},
{"iso...
1 961000000 169.0 [{"iso_639_1": "en", "name":
"English"}]
2 880674609 148.0 [{"iso_639_1": "fr", "name": "Fran\
u00e7ais"},...
3 1084939099 165.0 [{"iso_639_1": "en", "name":
"English"}]
4 284139100 132.0 [{"iso_639_1": "en", "name":
"English"}]
...
...
4798 2040920 81.0 [{"iso_639_1": "es", "name": "Espa\
u00f1ol"}]
4799 0 85.0
[]
4800 0 120.0 [{"iso_639_1": "en", "name":
"English"}]

```

```

4801      0      98.0      [{"iso_639_1": "en", "name":
"English"}]
4802      0      90.0      [{"iso_639_1": "en", "name":
"English"}]

      status      tagline \
0      Released      Enter the World of Pandora.
1      Released      At the end of the world, the adventure begins.
2      Released      A Plan No One Escapes
3      Released      The Legend Ends
4      Released      Lost in our world, found in another.
...      ...      ...
4798      Released      He didn't come looking for trouble, but troubl...
4799      Released      A newlywed couple's honeymoon is upended by th...
4800      Released      NaN
4801      Released      A New Yorker in Shanghai
4802      Released      NaN

      title      vote_average
vote_count
0      Avatar      7.2
11800
1      Pirates of the Caribbean: At World's End      6.9
4500
2      Spectre      6.3
4466
3      The Dark Knight Rises      7.6
9106
4      John Carter      6.1
2124
...      ...      ...
..
4798      El Mariachi      6.6
238
4799      Newlyweds      5.9
5
4800      Signed, Sealed, Delivered      7.0
6
4801      Shanghai Calling      5.7
7
4802      My Date with Drew      6.3
16

```

[4803 rows x 20 columns]

### Merge the Datasets

- To avoid overlapping of the tilte columns we drop because we already have the title in the movie dataset

```
# Drop the Title column in Movies Dataset
```

```
tmdb_movies.drop(['title'], axis = 1, inplace = True )
```

```
# Identify the columns that are common and need to be merged
```

```
tmdb_movie_credits.columns=['id', 'title', 'cast', 'crew']
```

```
movies_credits = pd.merge(tmdb_movie_credits, tmdb_movies, on = 'id')
```

```
movies_credits.head()
```

```

      id                                title \
0   19995                                Avatar
1    285  Pirates of the Caribbean: At World's End
2  206647                                Spectre
3   49026                The Dark Knight Rises
4   49529                John Carter

```

```

                                cast \
0 [{"cast_id": 242, "character": "Jake Sully", "...
1 [{"cast_id": 4, "character": "Captain Jack Spa...
2 [{"cast_id": 1, "character": "James Bond", "cr...
3 [{"cast_id": 2, "character": "Bruce Wayne / Ba...
4 [{"cast_id": 5, "character": "John Carter", "c...

```

```

                                crew    budget \
0 [{"credit_id": "52fe48009251416c750aca23", "de...  237000000
1 [{"credit_id": "52fe4232c3a36847f800b579", "de...  300000000
2 [{"credit_id": "54805967c3a36829b5002c41", "de...  245000000
3 [{"credit_id": "52fe4781c3a36847f81398c3", "de...  250000000
4 [{"credit_id": "52fe479ac3a36847f813eaa3", "de...  260000000

```

```

                                genres \
0 [{"id": 28, "name": "Action"}, {"id": 12, "nam...
1 [{"id": 12, "name": "Adventure"}, {"id": 14, "...
2 [{"id": 28, "name": "Action"}, {"id": 12, "nam...
3 [{"id": 28, "name": "Action"}, {"id": 80, "nam...
4 [{"id": 28, "name": "Action"}, {"id": 12, "nam...

```

```

                                homepage \
0                                http://www.avatarmovie.com/
1 http://disney.go.com/disneypictures/pirates/
2 http://www.sonypictures.com/movies/spectre/
3                                http://www.thedarkknightises.com/
4                                http://movies.disney.com/john-carter

```

```

                                keywords original_language
\
0 [{"id": 1463, "name": "culture clash"}, {"id":... en
1 [{"id": 270, "name": "ocean"}, {"id": 726, "na... en

```

2	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en
3	[{"id": 849, "name": "dc comics"}, {"id": 853,...	en
4	[{"id": 818, "name": "based on novel"}, {"id":...	en

	original_title	...	\
0	Avatar	...	
1	Pirates of the Caribbean: At World's End	...	
2	Spectre	...	
3	The Dark Knight Rises	...	
4	John Carter	...	

	production_companies	\
0	[{"name": "Ingenious Film Partners", "id": 289...	
1	[{"name": "Walt Disney Pictures", "id": 2}, {"...	
2	[{"name": "Columbia Pictures", "id": 5}, {"nam...	
3	[{"name": "Legendary Pictures", "id": 923}, {"...	
4	[{"name": "Walt Disney Pictures", "id": 2}]	

	production_countries	release_date
revenue	\	
0	[{"iso_3166_1": "US", "name": "United States o...	2009-12-10
2787965087		
1	[{"iso_3166_1": "US", "name": "United States o...	2007-05-19
961000000		
2	[{"iso_3166_1": "GB", "name": "United Kingdom"...	2015-10-26
880674609		
3	[{"iso_3166_1": "US", "name": "United States o...	2012-07-16
1084939099		
4	[{"iso_3166_1": "US", "name": "United States o...	2012-03-07
284139100		

	runtime	spoken_languages	status
\			
0	162.0	[{"iso_639_1": "en", "name": "English"}, {"iso...	Released
1	169.0	[{"iso_639_1": "en", "name": "English"}]	Released
2	148.0	[{"iso_639_1": "fr", "name": "Fran\u00e7ais"},...	Released
3	165.0	[{"iso_639_1": "en", "name": "English"}]	Released
4	132.0	[{"iso_639_1": "en", "name": "English"}]	Released

tagline	vote_average
---------	--------------

vote_count		
0	Enter the World of Pandora.	7.2
11800		
1	At the end of the world, the adventure begins.	6.9
4500		
2	A Plan No One Escapes	6.3
4466		
3	The Legend Ends	7.6
9106		
4	Lost in our world, found in another.	6.1
2124		

[5 rows x 22 columns]

- Movie ratings Dataset

```
movie_rating= pd.read_csv(r".data/ratings.csv")
movie_rating.head()
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

## Data Cleaning and Preparation

In this section, we create functions to explore the following characteristics of our dataset

- Info
- Columns, Column Names
- Datatypes
- Statistics

```
data_info = DatasetInfo(movies_credits)
print(data_info)
```

```
<my_functions.DatasetInfo object at 0x000002244676DBD0>
```

```
data_info.check_dataset_info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 4803 entries, 0 to 4802
```

```
Data columns (total 22 columns):
```

#	Column	Non-Null Count	Dtype
0	id	4803 non-null	int64
1	title	4803 non-null	object
2	cast	4803 non-null	object
3	crew	4803 non-null	object
4	budget	4803 non-null	int64
5	genres	4803 non-null	object



```

6  homepage          1712 non-null  object
7  keywords          4803 non-null  object
8  original_language  4803 non-null  object
9  original_title     4803 non-null  object
10 overview          4800 non-null  object
11 popularity         4803 non-null  float64
12 production_companies 4803 non-null  object
13 production_countries 4803 non-null  object
14 release_date       4802 non-null  object
15 revenue            4803 non-null  int64
16 runtime            4801 non-null  float64
17 spoken_languages    4803 non-null  object
18 status             4803 non-null  object
19 tagline             3959 non-null  object
20 vote_average        4803 non-null  float64
21 vote_count         4803 non-null  int64
dtypes: float64(3), int64(4), object(15)
memory usage: 825.6+ KB

```

```
data_info.check_dataset_shape()
```

```
Dataset shape: (4803, 22)
```

```
data_info.get_dataset_statistics_describe()
```

	id	budget	popularity	revenue
runtime \				
count	4803.000000	4.803000e+03	4803.000000	4.803000e+03
mean	57165.484281	2.904504e+07	21.492301	8.226064e+07
std	88694.614033	4.072239e+07	31.816650	1.628571e+08
min	5.000000	0.000000e+00	0.000000	0.000000e+00
25%	9014.500000	7.900000e+05	4.668070	0.000000e+00
50%	14629.000000	1.500000e+07	12.921594	1.917000e+07
75%	58610.500000	4.000000e+07	28.313505	9.291719e+07
max	459488.000000	3.800000e+08	875.581305	2.787965e+09

	vote_average	vote_count
count	4803.000000	4803.000000
mean	6.092172	690.217989
std	1.194612	1234.585891
min	0.000000	0.000000
25%	5.600000	54.000000
50%	6.200000	235.000000

```
75%          6.800000    737.000000
max         10.000000   13752.000000
```

```
movies_credits.duplicated().sum()
```

```
0
```

```
movies_credits.isnull().sum()
```

```
id                0
title             0
cast              0
crew              0
budget            0
genres            0
homepage          3091
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
vote_average      0
vote_count        0
dtype: int64
```

Some columns within our dataset contain a list of dictionaries. In the cells below, we apply the function from my\_functions to extract the desired attribute from each element in the columns.

```
# For the genres column we have to convert it but first we have to implement
```

```
movies_credits.iloc[0].genres
```

```
'[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]'
```

Here we will apply the functions from our python file to clean the dataset columns

```
# Cleaning the genres column
```

```
movies_credits['genres']=movies_credits['genres'].apply(data_info.convert)
```

```
# Cleaning the keywords column
```

```
movies_credits['keywords']=movies_credits['keywords'].apply(data_info.get_keywords)
```

```
# Cleaning the production companies column
```

```
movies_credits['production_companies']=movies_credits['production_companies'].apply(data_info.convert)
```

```
# Cleaning the production countries column
```

```
movies_credits['production_countries'] =  
movies_credits['production_countries'].apply(data_info.convert)
```

```
# Cleaning the cast column
```

```
movies_credits['cast']=movies_credits['cast'].apply(data_info.convert3)
```

```
# Cleaning the crew column
```

```
movies_credits['crew']=movies_credits['crew'].apply(data_info.get_directors)
```

In the cell below, we split the text in the overview column into a list of words for each row where the value is a string. For rows where the value is not a string, it assigns np.nan to indicate a missing value.

```
movies_credits['overview'] = movies_credits['overview'].apply(lambda  
x: x.split() if isinstance(x, str) else np.nan)
```

```
movies_credits.head()
```

	id	title \	cast	crew \	budget	genres \
0	19995	Avatar		[Sam Worthington, Zoe Saldana, Sigourney Weave... [James Cameron]		
1	285	Pirates of the Caribbean: At World's End		[Johnny Depp, Orlando Bloom, Keira Knightley, ... [Gore Verbinski]		
2	206647	Spectre		[Daniel Craig, Christoph Waltz, Léa Seydoux, R... [Sam Mendes]		
3	49026	The Dark Knight Rises		[Christian Bale, Michael Caine, Gary Oldman, A... [Christopher Nolan]		
4	49529	John Carter		[Taylor Kitsch, Lynn Collins, Samantha Morton,... [Andrew Stanton]		
0	237000000	[Action, Adventure, Fantasy, Science Fiction]				

1	300000000	[Adventure, Fantasy, Action]
2	245000000	[Action, Adventure, Crime]
3	250000000	[Action, Crime, Drama, Thriller]
4	260000000	[Action, Adventure, Science Fiction]

	homepage \
0	<a href="http://www.avatarmovie.com/">http://www.avatarmovie.com/</a>
1	<a href="http://disney.go.com/disneypictures/pirates/">http://disney.go.com/disneypictures/pirates/</a>
2	<a href="http://www.sonypictures.com/movies/spectre/">http://www.sonypictures.com/movies/spectre/</a>
3	<a href="http://www.thedarkknightriserises.com/">http://www.thedarkknightriserises.com/</a>
4	<a href="http://movies.disney.com/john-carter">http://movies.disney.com/john-carter</a>

	keywords	original_language
\		
0	[culture clash, future, space war, space colon...	en
1	[ocean, drug abuse, exotic island, east india ...	en
2	[spy, based on novel, secret agent, sequel, mi...	en
3	[dc comics, crime fighter, terrorist, secret i...	en
4	[based on novel, mars, medallion, space travel...	en

	original_title	...	\
0	Avatar	...	
1	Pirates of the Caribbean: At World's End	...	
2	Spectre	...	
3	The Dark Knight Rises	...	
4	John Carter	...	

	production_companies	\
0	[Ingenious Film Partners, Twentieth Century Fo...	
1	[Walt Disney Pictures, Jerry Bruckheimer Films...	
2	[Columbia Pictures, Danjaq, B24]	
3	[Legendary Pictures, Warner Bros., DC Entertai...	
4	[Walt Disney Pictures]	

	production_countries	release_date	revenue
\			
0	[United States of America, United Kingdom]	2009-12-10	2787965087
1	[United States of America]	2007-05-19	961000000
2	[United Kingdom, United States of America]	2015-10-26	880674609
3	[United States of America]	2012-07-16	1084939099

```
4 [United States of America] 2012-03-07 284139100
```

	runtime	spoken_languages	status
\			
0	162.0	[{"iso_639_1": "en", "name": "English"}, {"iso...	Released
1	169.0	[{"iso_639_1": "en", "name": "English"}]	Released
2	148.0	[{"iso_639_1": "fr", "name": "Fran\u00e7ais"},...	Released
3	165.0	[{"iso_639_1": "en", "name": "English"}]	Released
4	132.0	[{"iso_639_1": "en", "name": "English"}]	Released

	tagline	vote_average
vote_count		
0	Enter the World of Pandora.	7.2
11800		
1	At the end of the world, the adventure begins.	6.9
4500		
2	A Plan No One Escapes	6.3
4466		
3	The Legend Ends	7.6
9106		
4	Lost in our world, found in another.	6.1
2124		

```
[5 rows x 22 columns]
```

The code below applies a lambda function to iterate over each element *i* (column value) in the input list *x* (column). It replaces any occurrence of whitespace (" ") in each element with an empty string, effectively removing any spaces.

```
# Remove spaces from the elements in the 'genres' column
movies_credits['genres'] = movies_credits['genres'].apply(lambda x:
[i.replace(" ", "") for i in x])
```

```
# Remove spaces from the elements in the 'keywords' column
movies_credits['keywords'] = movies_credits['keywords'].apply(lambda
x: [i.replace(" ", "") for i in x])
```

```
# Remove spaces from the elements in the 'crew' column
movies_credits['crew'] = movies_credits['crew'].apply(lambda x:
[i.replace(" ", "") for i in x])
```

```
# Remove spaces from the elements in the 'cast' column
```

```
movies_credits['cast'] = movies_credits['cast'].apply(lambda x:
[i.replace(" ", "") for i in x])
```

Concatenating the modified columns into one named tags

```
movies_credits['tags'] = movies_credits['overview'] +
movies_credits['genres'] + movies_credits['keywords'] +
movies_credits['cast'] + movies_credits['crew']
```

```
movies_credits['tags']
```

```
0      [In, the, 22nd, century,, a, paraplegic, Marin...
1      [Captain, Barbossa,, long, believed, to, be, d...
2      [A, cryptic, message, from, Bond's, past, send...
3      [Following, the, death, of, District, Attorney...
4      [John, Carter, is, a, war-weary,, former, mili...
...
4798   [El, Mariachi, just, wants, to, play, his, gui...
4799   [A, newlywed, couple's, honeymoon, is, upended...
4800   ["Signed,, Sealed,, Delivered", introduces, a,...
4801   [When, ambitious, New, York, attorney, Sam, is...
4802   [Ever, since, the, second, grade, when, he, fi...
Name: tags, Length: 4803, dtype: object
```

## EDA

### I. Univariate Analysis

- Vote Count

```
# Vote Count description
```

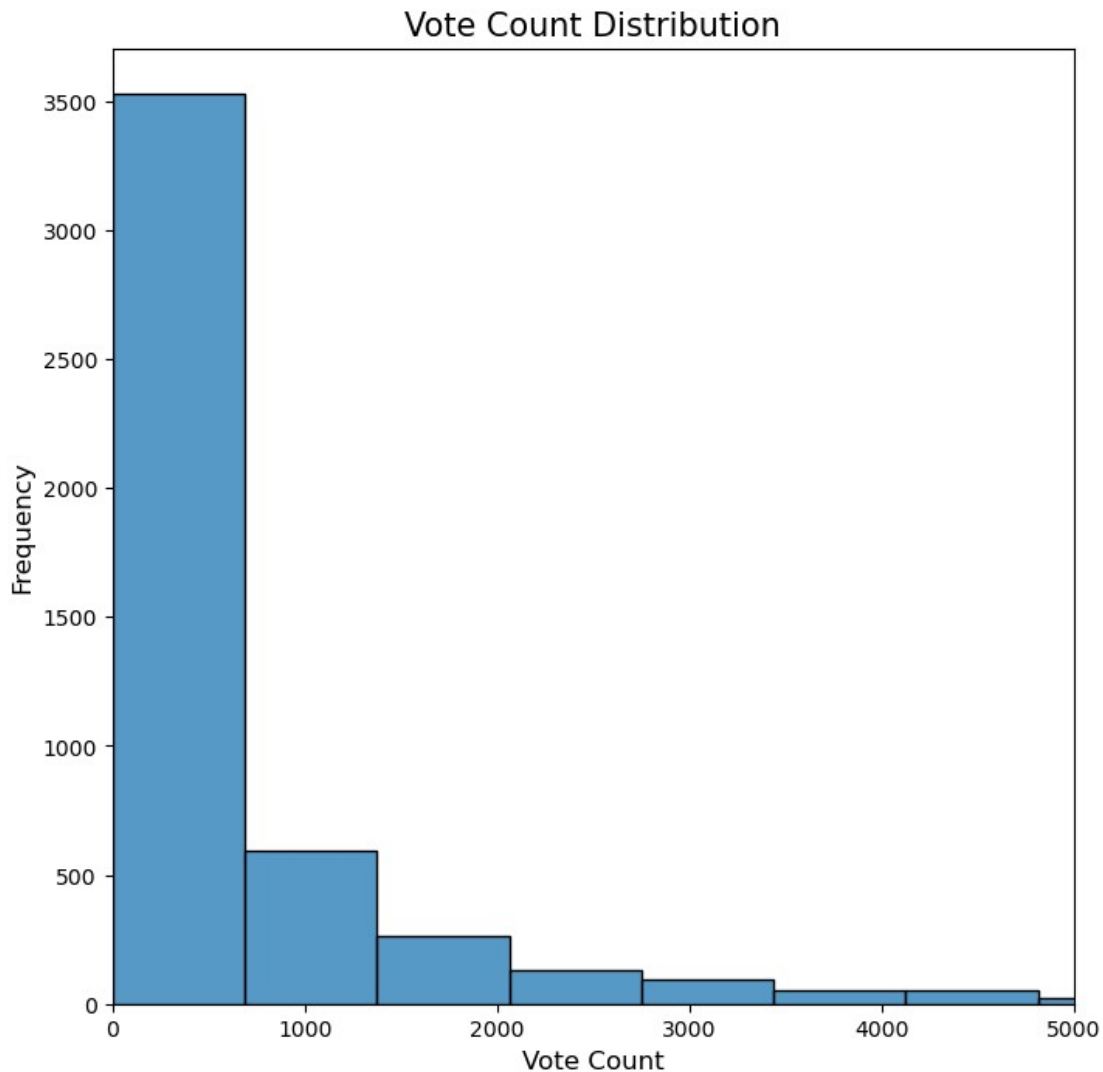
```
vote_count_univariate = movies_credits['vote_count'].describe()
print(vote_count_univariate)
```

```
# Plot vote count distribution
```

```
plt.figure(figsize=(8, 8))
sns.histplot(movies_credits['vote_count'], kde = False , bins = 20)
plt.xlabel("Vote Count", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
plt.xlim(0, 5000)
plt.title("Vote Count Distribution", fontsize=15)
plt.savefig(".data/images/vote_count_plot")
plt.show()
```

```
count      4803.000000
mean        690.217989
std         1234.585891
min          0.000000
25%          54.000000
50%         235.000000
75%         737.000000
```

```
max      13752.000000
Name: vote_count, dtype: float64
```



From the plot above we can determine that the vote count decreases hence a low concentration for the vote counts

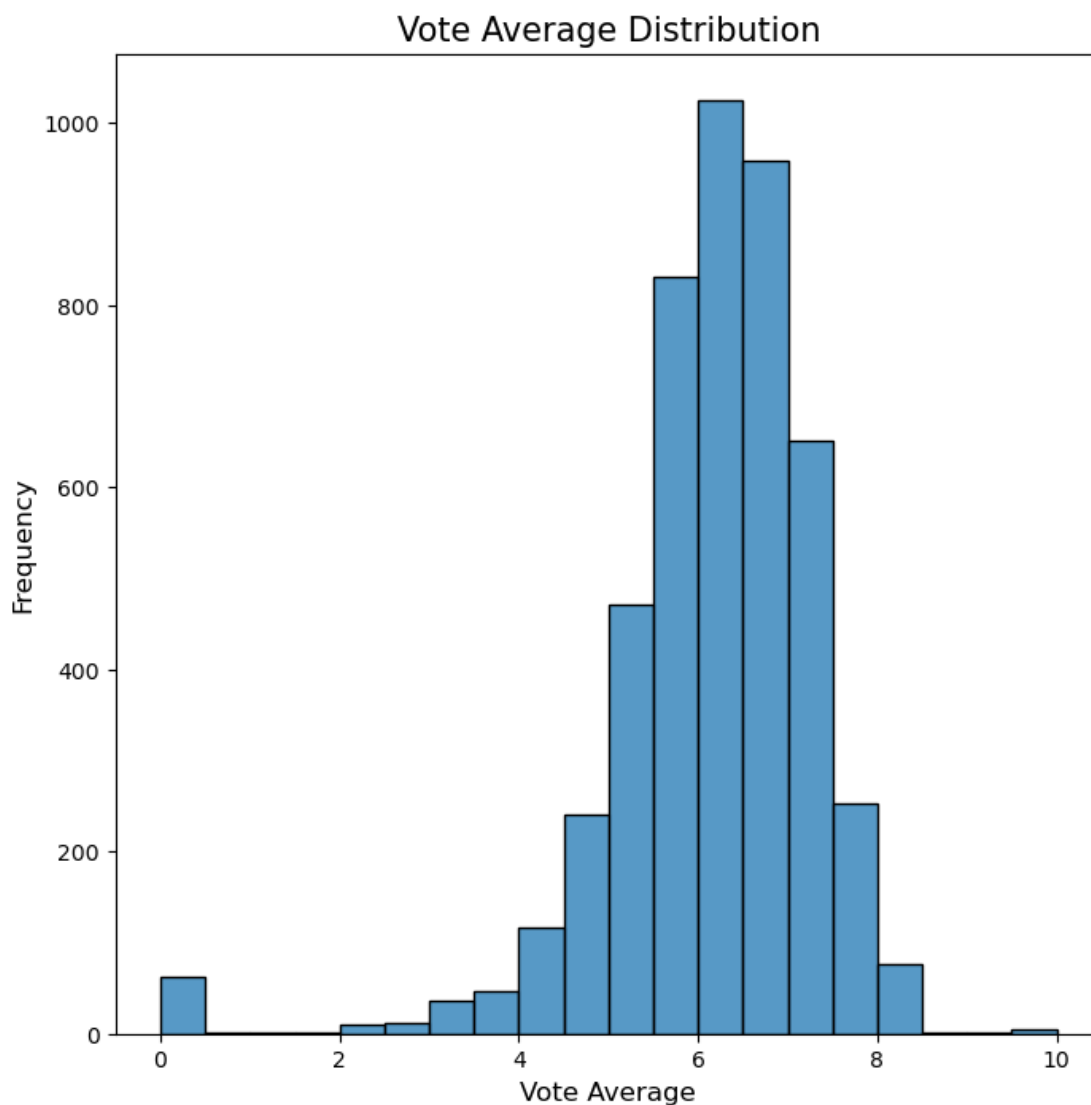
- Vote Average

```
vote_average_univariate = movies_credits['vote_average'].describe()
print(vote_average_univariate)
```

```
# Plot the vote average distribution
plt.figure(figsize=(8, 8))
sns.histplot(movies_credits['vote_average'], kde = False , bins = 20)
plt.xlabel("Vote Average", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
plt.title("Vote Average Distribution", fontsize=15)
# save the figure
```

```
plt.savefig(".data/images/vote_average_plot")  
# show the figure  
plt.show()
```

```
count    4803.000000  
mean      6.092172  
std       1.194612  
min       0.000000  
25%      5.600000  
50%      6.200000  
75%      6.800000  
max      10.000000  
Name: vote_average, dtype: float64
```



The vote average is normally distributed with the majority of it being 6-8.



### Check for existence of outliers

*# Select the variables you want to plot*

```
cols_to_plot = ['budget', 'popularity', 'revenue', 'vote_average',  
'vote_count']
```

*##### Create a subplot grid*

```
fig, axes = plt.subplots(nrows=1, ncols=len(cols_to_plot),  
figsize=(12, 12), sharey=True)
```

*##### Create a boxplot for each variable in a separate subplot*

```
for i, col in enumerate(cols_to_plot):  
    axes[i].boxplot(movies_credits[col])  
    axes[i].set_title(col)  
    axes[i].tick_params(axis='both', which='major')
```

*# Adjust spacing between subplots*

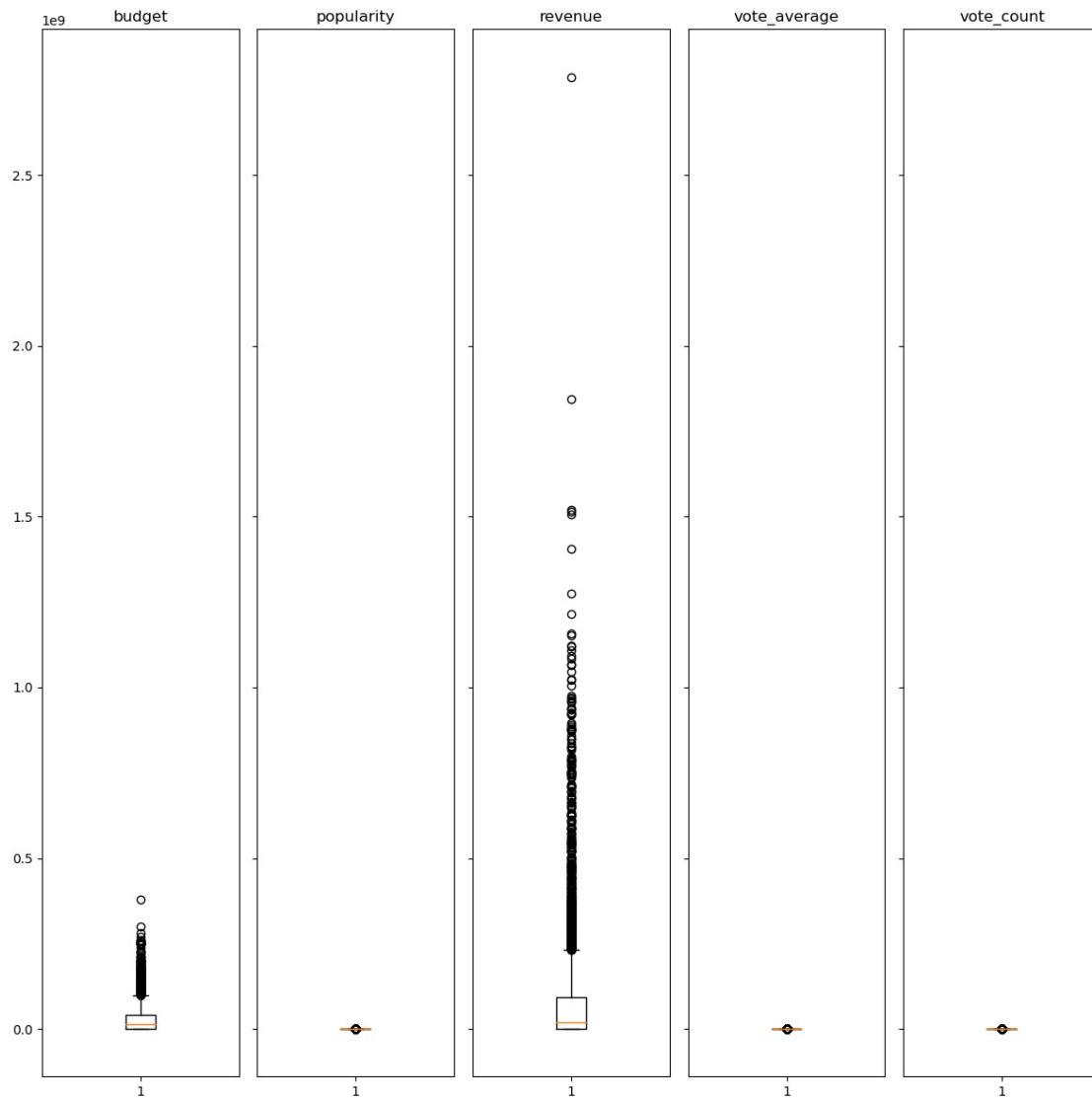
```
plt.tight_layout()
```

*# save te figure*

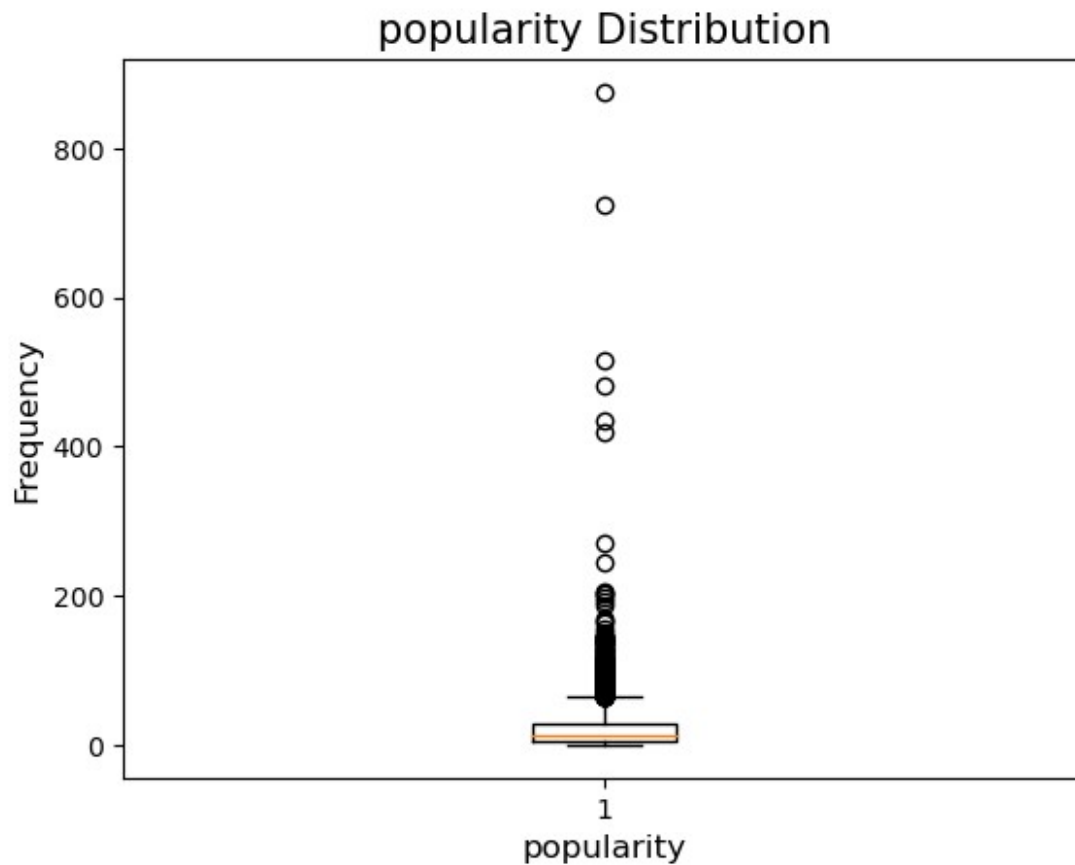
```
plt.savefig(".data/images/Outliers")
```

*# Show the figure*

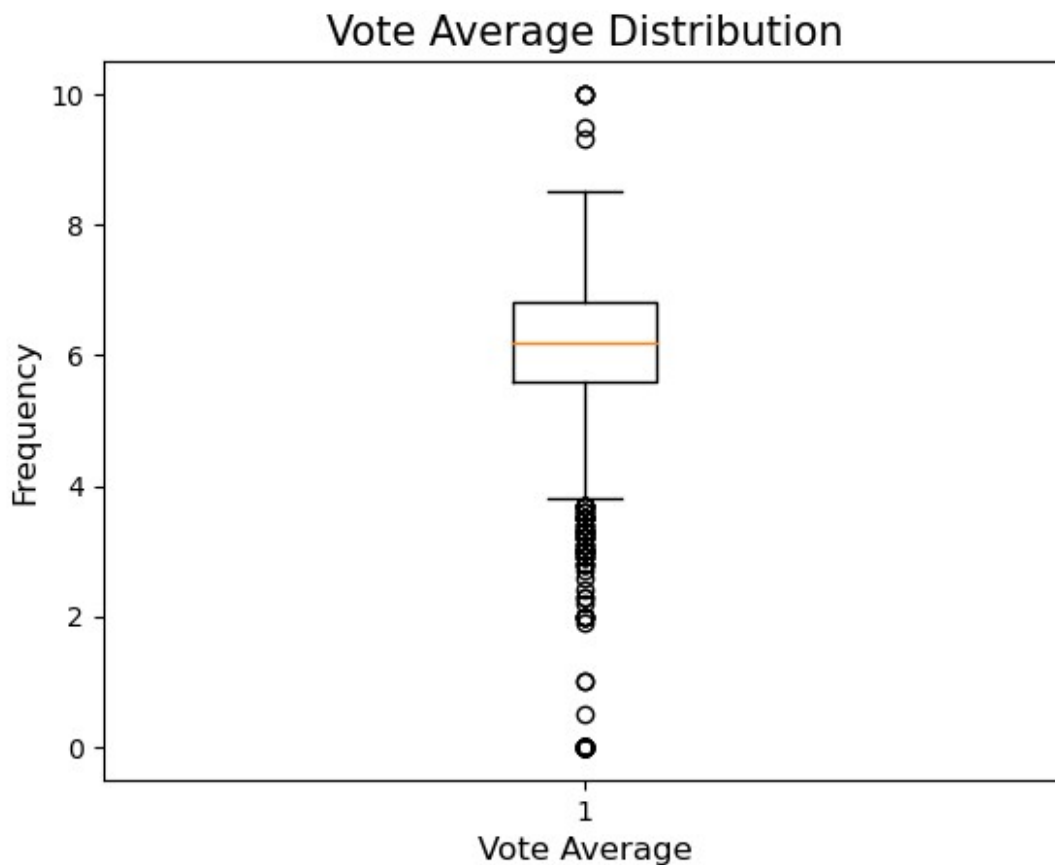
```
plt.show()
```



```
# Checking for outliers in the 'popularity' column
plt.boxplot(movies_credits['popularity'])
plt.xlabel("popularity", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
plt.title("popularity Distribution", fontsize=15)
# save the figure
plt.savefig(".data/images/popularity_outliers_plot")
plt.show()
```



```
## Checking for outliers in the 'vote average' column
plt.boxplot(movies_credits['vote_average'])
plt.xlabel("Vote Average", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
plt.title("Vote Average Distribution", fontsize=15)
# save the figure
plt.savefig(".data/images/vote_average_outliers_plot")
plt.show()
```



There are outliers from 0-4 and 8-10, hence we remove them

```
movies_credits['popularity'].nlargest(10)
```

```
546      875.581305
95       724.247784
788      514.569956
94       481.098624
127      434.278564
28       418.708552
199      271.972889
82       243.791743
200      206.227151
88       203.734590
```

Name: popularity, dtype: float64

```
result = (movies_credits['popularity'] >= 200 ).value_counts()
```

```
movies_credits['popularity'].nsmallest(10)
```

```
4553      0.000000
3361      0.000372
4727      0.001117
4118      0.001186
```

```

4625    0.001389
4509    0.001586
4305    0.002386
4721    0.002388
4487    0.003142
4589    0.003352
Name: popularity, dtype: float64

# Define the lower and upper bounds for the outliers
lower_bound = 0
upper_bound = 400

# Remove outliers from the 'vote_average' column
movies_credits_filtered = movies_credits[(movies_credits['popularity']
>= lower_bound) & (movies_credits['popularity'] <= upper_bound)]

# Display the shape filtered dataset without outliers
movies_credits_filtered.shape

(4797, 23)

# Define the lower and upper bounds for the outliers
lower_bound = 2
upper_bound = 8

# Remove outliers from the 'vote_average' column
movies_credits_filtered =
movies_credits[(movies_credits['vote_average'] >= lower_bound) &
(movies_credits['vote_average'] <= upper_bound)]

# Display the shape filtered dataset without outliers
movies_credits_filtered.shape

(4686, 23)

• Keywords
# Extract the 'keywords' column
keywords = movies_credits['keywords']

# Flatten the list of keywords
flat_keywords = [keyword for sublist in keywords for keyword in
sublist]

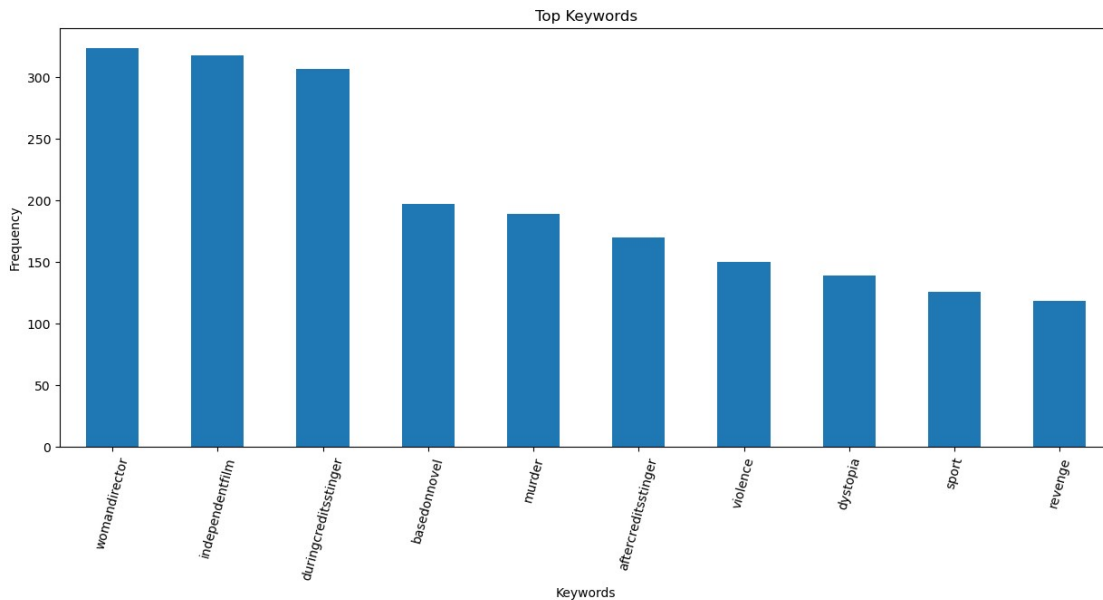
# Count the frequency of each keyword
keyword_counts = pd.Series(flat_keywords).value_counts().head(10)

# Select the top keywords
top_keywords = keyword_counts.head(20)

# Plot the top keywords

```

```
plt.figure(figsize=(15, 6))
top_keywords.plot(kind='bar')
plt.title('Top Keywords')
plt.xlabel('Keywords')
plt.ylabel('Frequency')
plt.xticks(rotation=75)
# save the figure
plt.savefig(".data/images/Keywords_plot")
plt.show()
```



- Genres

```
# Extract the 'genres' column
genres = movies_credits['genres']

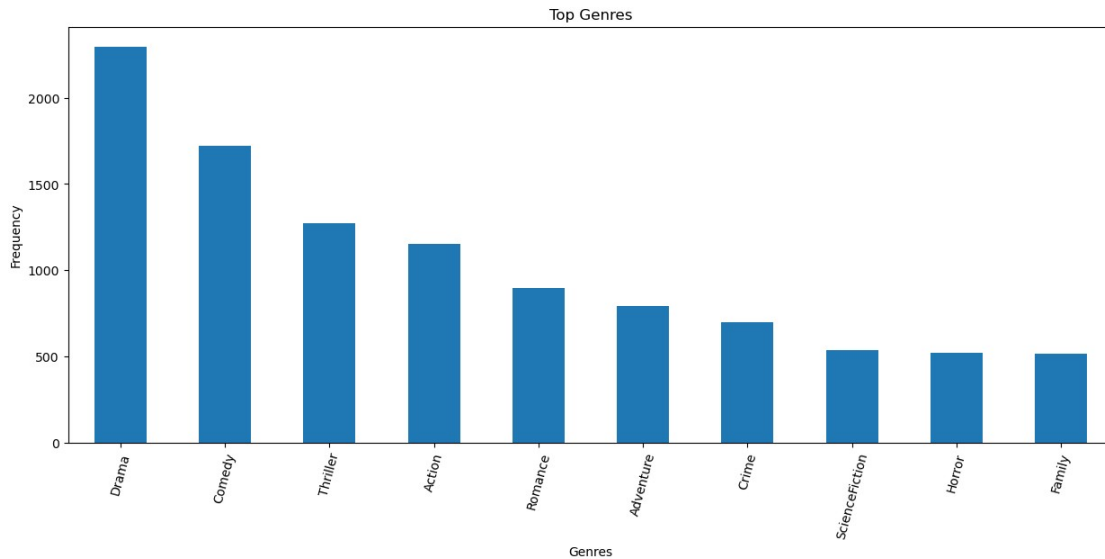
# Flatten the list of genres
flat_genres = [genre for sublist in genres for genre in sublist]

# Count the frequency of each genre
genre_counts = pd.Series(flat_genres).value_counts()

# Select the top genres
top_genres = genre_counts.head(10)

# Plot the top genres
plt.figure(figsize=(15, 6))
top_genres.plot(kind='bar')
plt.title('Top Genres')
plt.xlabel('Genres')
plt.ylabel('Frequency')
plt.xticks(rotation=75)
# save the figure
```

```
plt.savefig('.data/images/top_genres')
plt.show()
```



- Spoken Languages

```
# Extract the 'spoken_languages' column
```

```
spoken_languages = movies_credits['spoken_languages']
```

```
# Flatten the list of spoken languages
```

```
flat_languages = []
```

```
for sublist in spoken_languages:
```

```
    if isinstance(sublist, str):
```

```
        sublist = ast.literal_eval(sublist)
```

```
    for language in sublist:
```

```
        if isinstance(language, dict):
```

```
            flat_languages.append(language['name'])
```

```
# Count the frequency of each spoken language
```

```
language_counts = pd.Series(flat_languages).value_counts()
```

```
# Select the top spoken languages
```

```
top_languages = language_counts.head(5) # Change the number to select more or fewer top languages
```

```
# Plot the top spoken languages
```

```
plt.figure(figsize=(10, 6))
```

```
top_languages.plot(kind='bar')
```

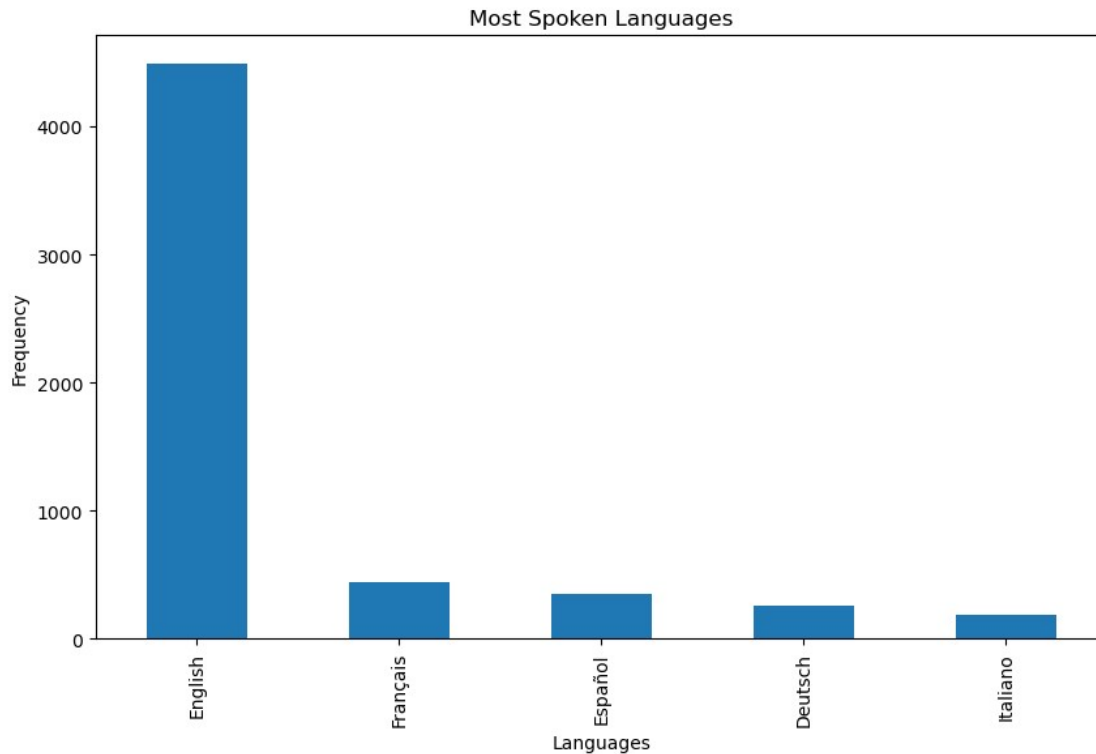
```
plt.title('Most Spoken Languages')
```

```
plt.xlabel('Languages')
```

```
plt.ylabel('Frequency')
```

```
plt.savefig('.data/images/most_spoken_languages')
```

```
plt.show()
```



- Movie Status

```
movies_credits['status'].value_counts()
```

```
status
Released          4795
Rumored            5
Post Production    3
Name: count, dtype: int64
```

- Production Companies

Top company collaborations

```
# Get value counts of production companies
```

```
production_company_counts =
movies_credits['production_companies'].value_counts()
```

```
# Select the top 5 production companies
```

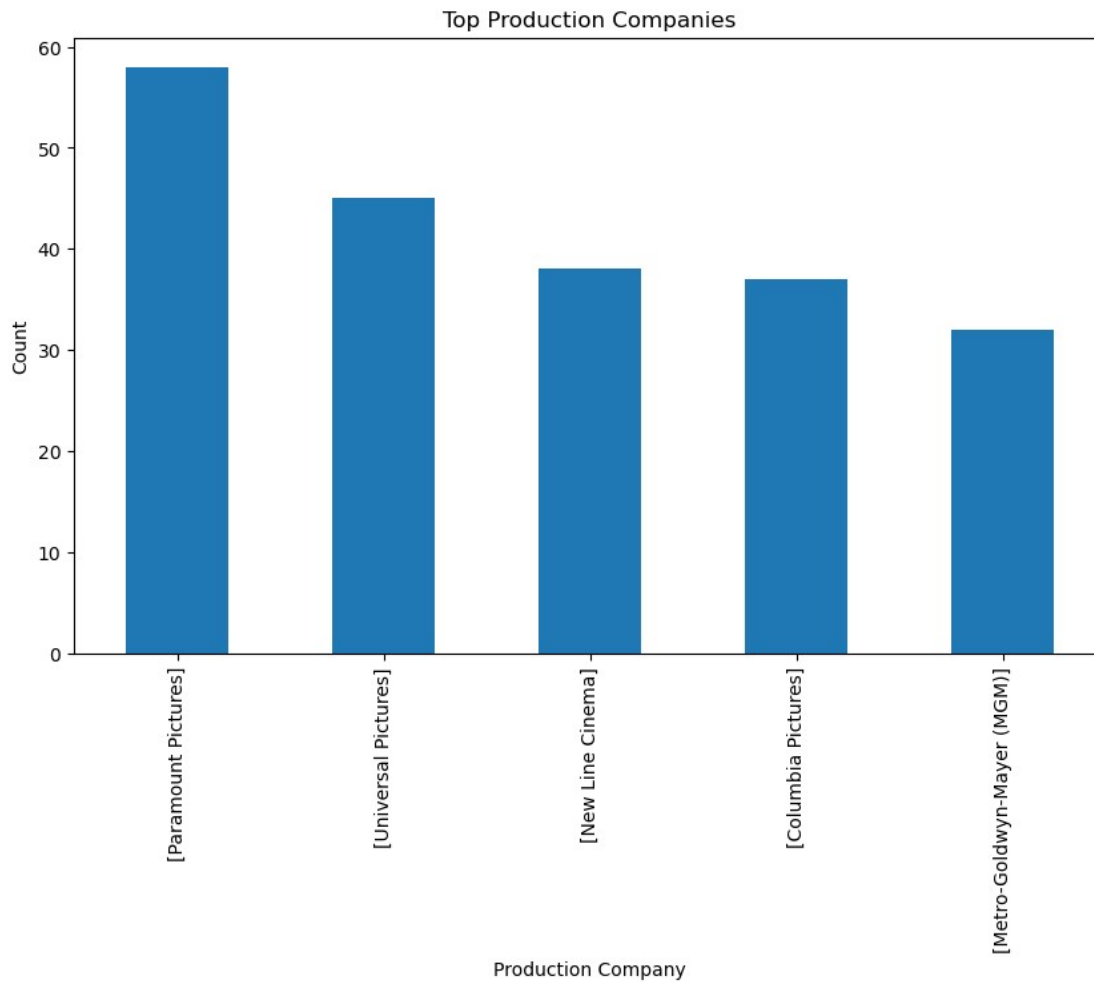
```
top_production_companies = production_company_counts[1:6]
```

```
# Plot the top production companies
```

```
plt.figure(figsize=(10, 6))
top_production_companies.plot(kind='bar')
plt.title('Top Production Companies')
plt.xlabel('Production Company')
plt.ylabel('Count')
```

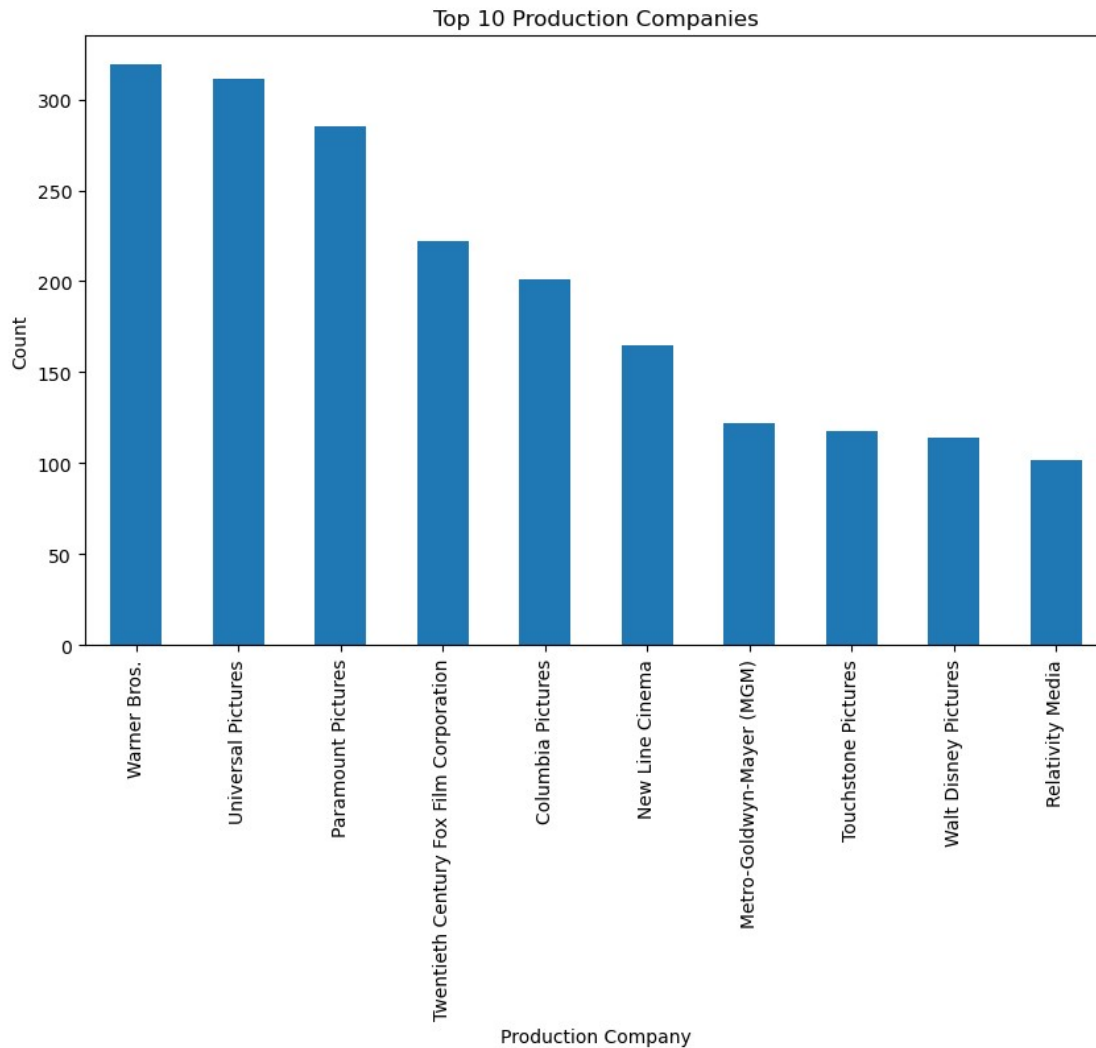


```
plt.savefig('.data/images/top_production_companies')
plt.show()
```



Top companies individually

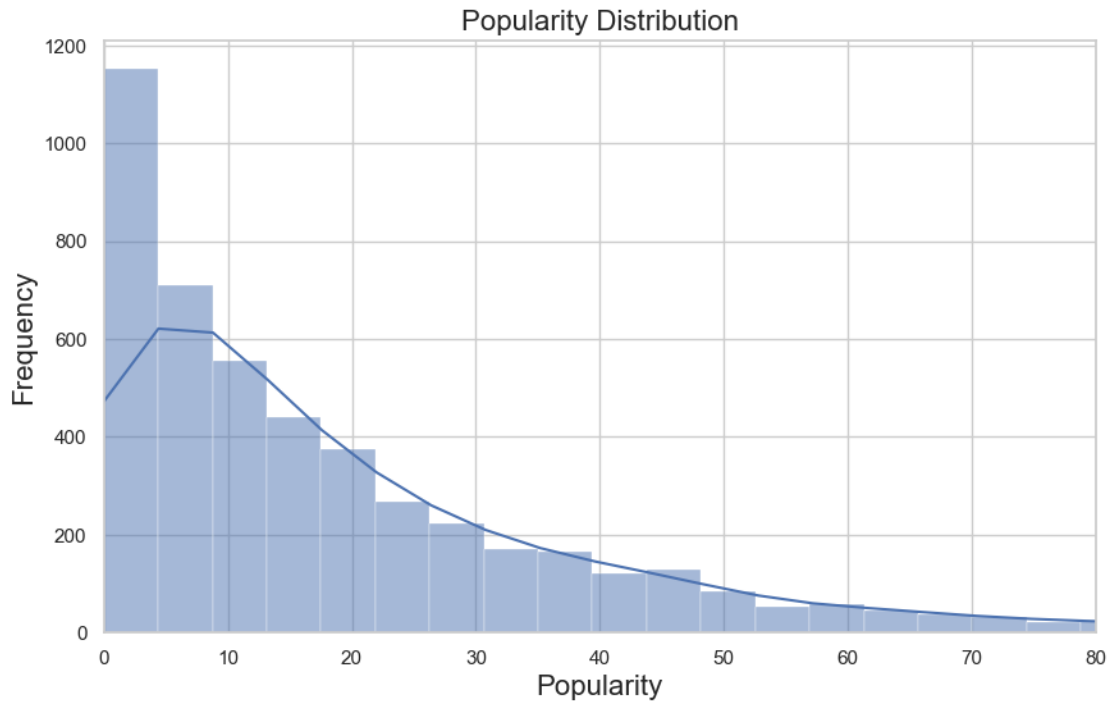
```
# Plotting 'production_companies' (top 10)
plt.figure(figsize=(10, 6))
top_10_production_companies =
movies_credits['production_companies'].explode().value_counts().head(10)
top_10_production_companies.plot(kind='bar')
plt.title('Top 10 Production Companies')
plt.xlabel('Production Company')
plt.ylabel('Count')
plt.savefig('.data/images/top_10_production_companies')
plt.show()
```



- Popularity

```
# Set the style and context  
sns.set(style='whitegrid')
```

```
# Histogram for popularity  
plt.figure(figsize=(10, 6))  
sns.histplot(data=movies_credits, x='popularity', bins=200, kde=True)  
plt.title('Popularity Distribution', fontsize=16)  
plt.xlabel('Popularity', fontsize=16)  
plt.ylabel('Frequency', fontsize=16)  
plt.xlim(0, 80) # Set the x-axis limits  
plt.show()
```



- Release date

*# Set the style and context for Seaborn*

```
sns.set(style='darkgrid')
```

*# Line plot for release\_date*

```
plt.figure(figsize=(10, 6))
```

```
movies_credits['release_date'] =
```

```
pd.to_datetime(movies_credits['release_date'], format='%Y-%m-%d',  
errors='coerce')
```

```
movies_counts =
```

```
movies_credits.groupby(movies_credits['release_date'].dt.year)
```

```
['release_date'].count().tail(20)
```

```
movies_counts.plot(kind='bar')
```

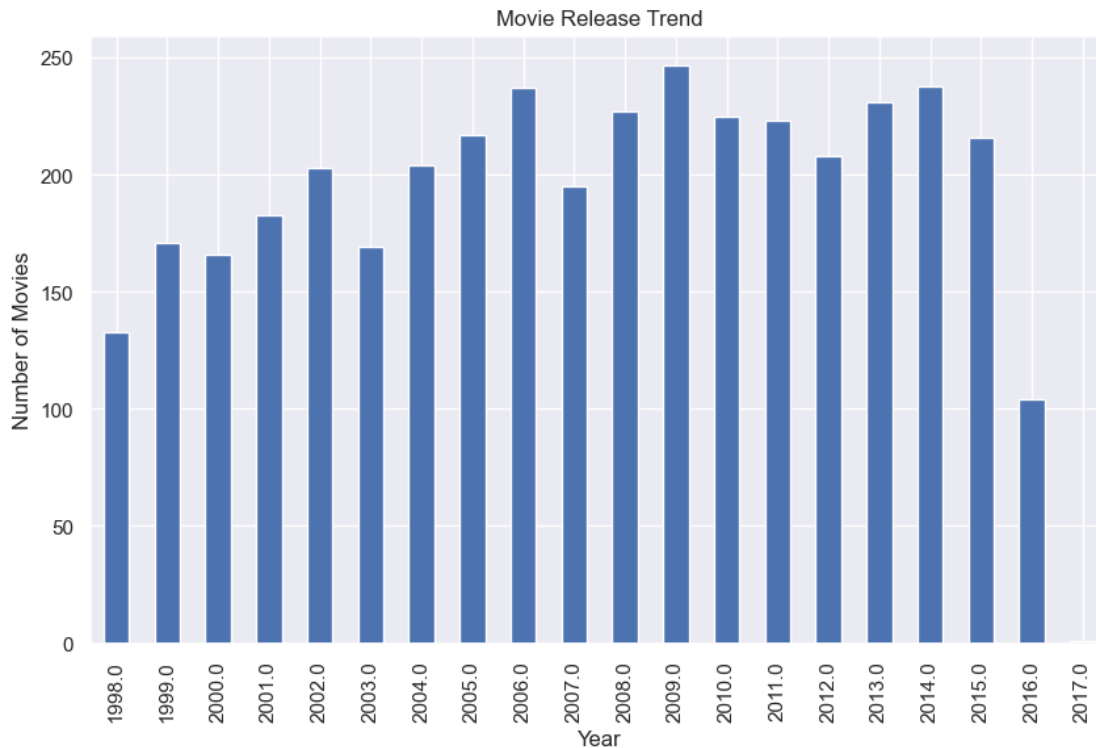
```
plt.title('Movie Release Trend')
```

```
plt.xlabel('Year')
```

```
plt.ylabel('Number of Movies')
```

```
plt.savefig(".data/images/movies_Reease_year")
```

```
plt.show()
```



- Word Cloud of Movie Overviews

*# Concatenate all overview strings into a single string and remove single quotes*

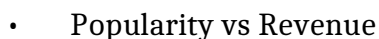
```
overview_text = ' '.join([str(overview).replace("'", "") for overview
in movies_credits['overview']])
```

*# Word cloud for overview*

```
plt.figure(figsize=(10, 6))
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(overview_text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud of Movie Overviews', fontsize=20)
plt.axis('off')
plt.savefig(".data/images/wordcloud")
plt.show()
```

- Original Title vs Popularity

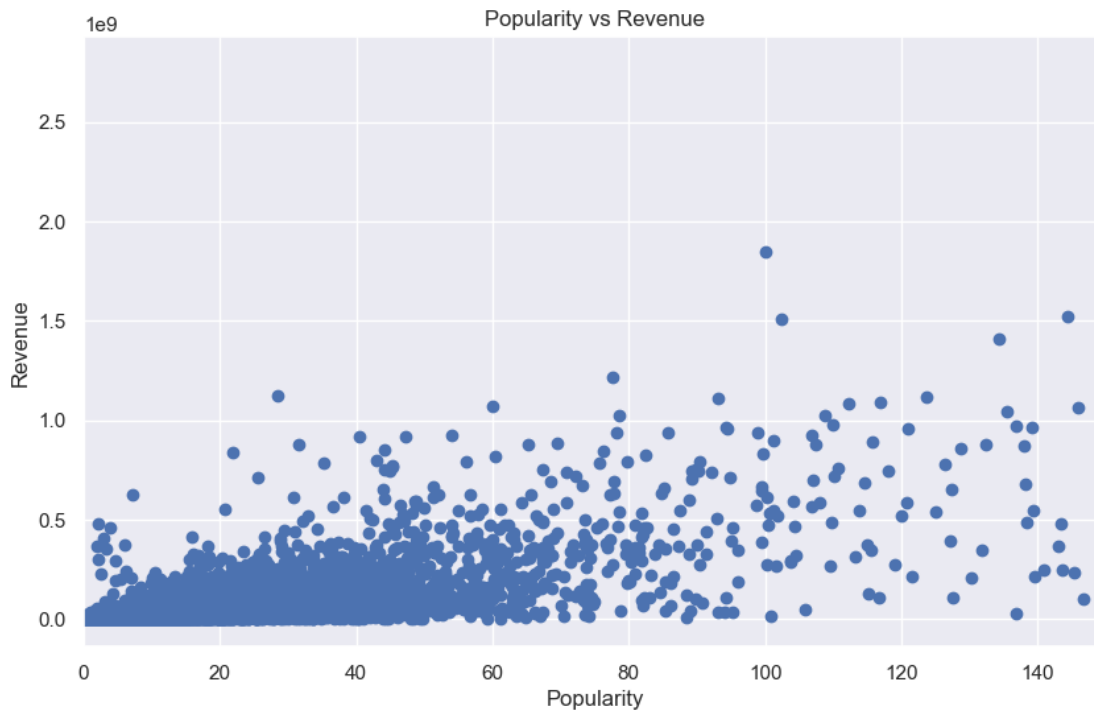
```
plt.figure(figsize=(10, 6))
top_10_movies = movies_credits.sort_values('popularity',
ascending=False).head(10)
plt.barh(top_10_movies['original_title'], top_10_movies['popularity'])
plt.title('Top 10 Movies by Popularity')
plt.xlabel('Popularity')
plt.ylabel('Original Title')
plt.gca().invert_yaxis()
plt.savefig('.data/images/top_10_movies_popularity')
plt.show()
```



```

# Plotting 'popularity' vs 'revenue'
plt.figure(figsize=(10, 6))
plt.scatter(movies_credits['popularity'], movies_credits['revenue'])
plt.title('Popularity vs Revenue')
plt.xlabel('Popularity')
plt.xlim(0, 150)
plt.ylabel('Revenue')
plt.savefig('.data/images/popularity vs revenue')
plt.show()

```



- Number of Movies by original language of production

```

#Get value counts of original languages
original_language_counts =
movies_credits['original_language'].value_counts()

# Get the top 5 languages
top_languages = original_language_counts[:5]

# Calculate the count for the "Others" category
others_count = original_language_counts[5:].sum()

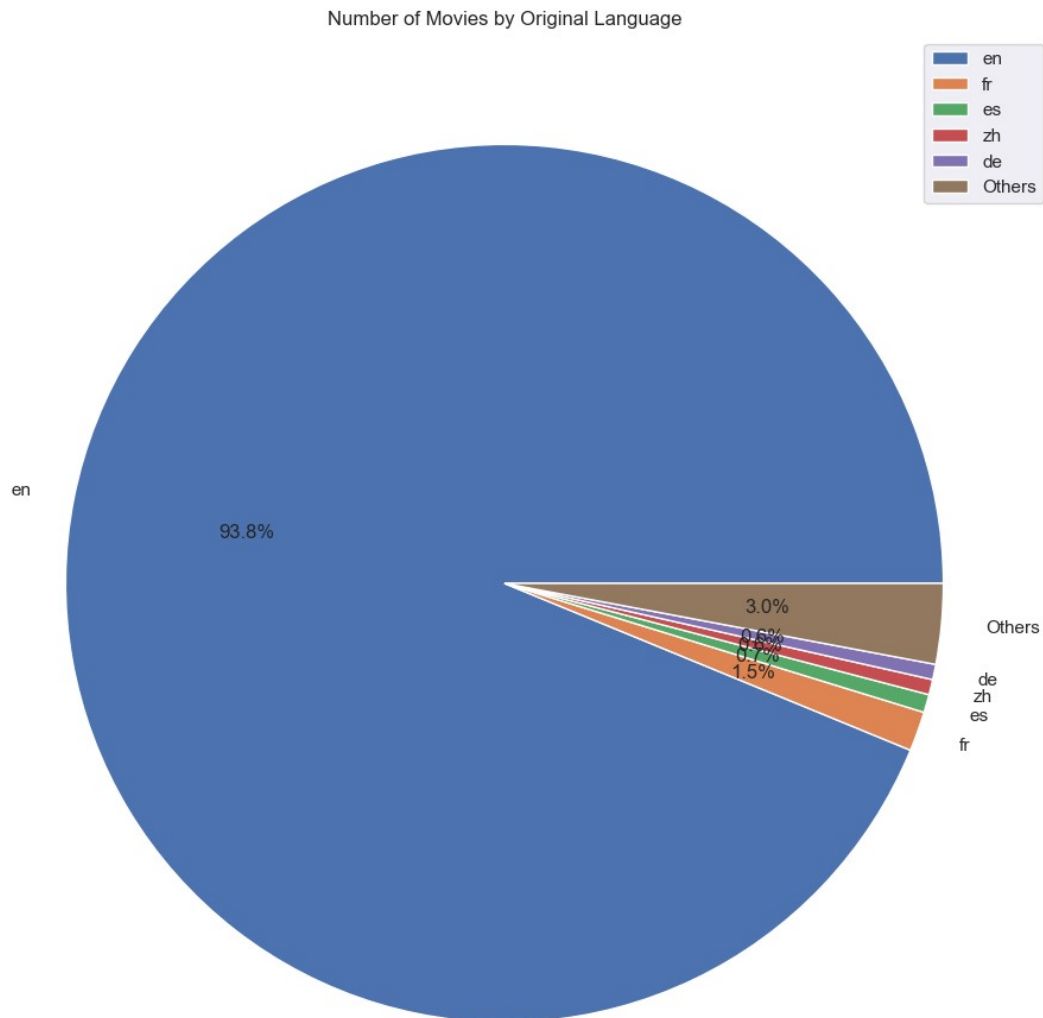
# Create a new series with the top 5 languages and "Others"
languages_data = pd.concat([top_languages, pd.Series(others_count,
index=['Others'])])

# Plotting 'original_language'
plt.figure(figsize=(12, 12))
languages_data.plot(kind='pie', autopct='%1.1f%%')

```

```
plt.title('Number of Movies by Original Language')
plt.ylabel('')
```

```
# Add labels to the pie chart
plt.legend(labels=languages_data.index)
plt.savefig('.data/images/no_movies by language')
plt.show()
```



Movies by production country

```
# Get value counts of production countries
production_countries_counts =
movies_credits['production_countries'].explode().value_counts()

# Get the top five production countries
top_countries = production_countries_counts[:5]
```

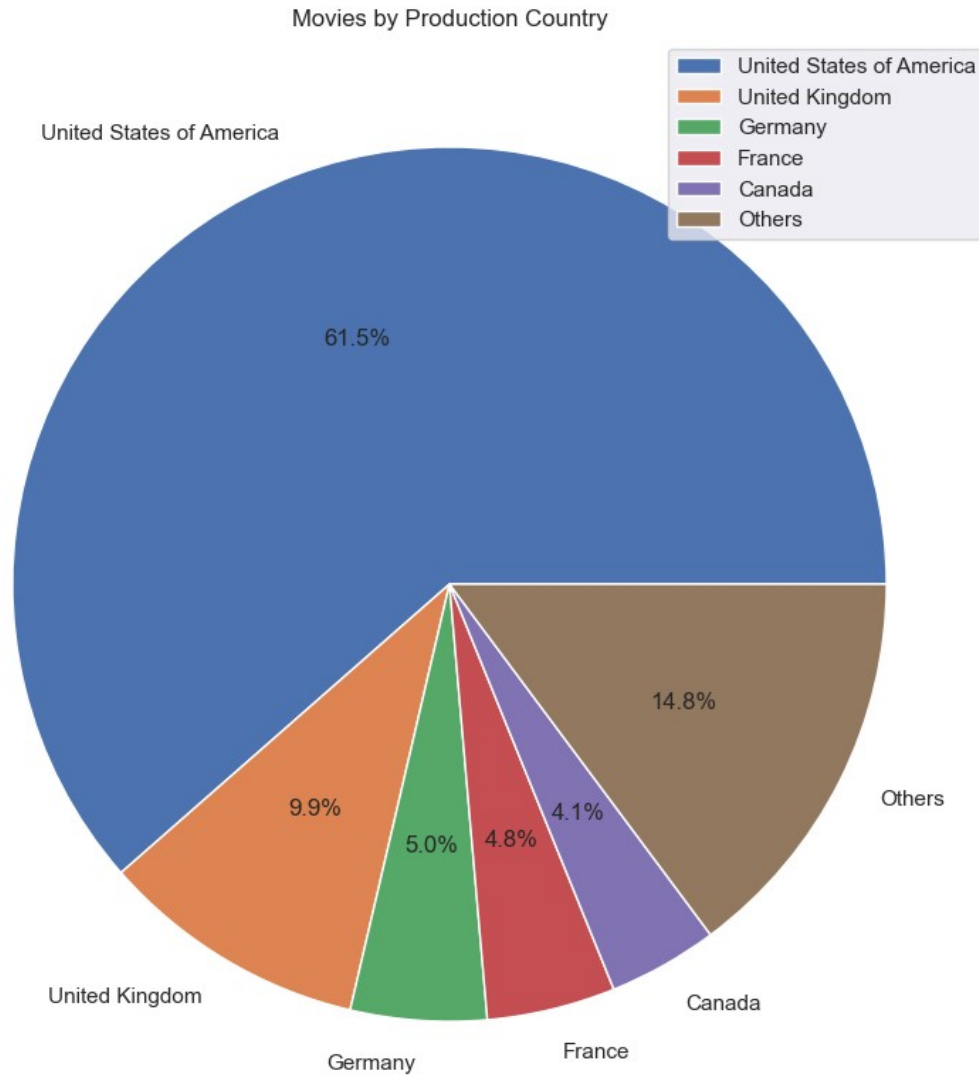
```
# Calculate the count for the sixth slice ("others")
others_count = production_countries_counts[5:].sum()

# Create a new series with the top five countries and "others"
countries_data = pd.concat([top_countries, pd.Series(others_count,
index=['Others'])])

# Plotting 'production_countries'
plt.figure(figsize=(20, 10))
countries_data.plot(kind='pie', autopct='%1.1f%%')
plt.title('Movies by Production Country')
plt.ylabel('')

# Labeling the countries
plt.legend(labels=countries_data.index, loc='best')
plt.savefig('.data/images/movies_countries')
plt.show()
```

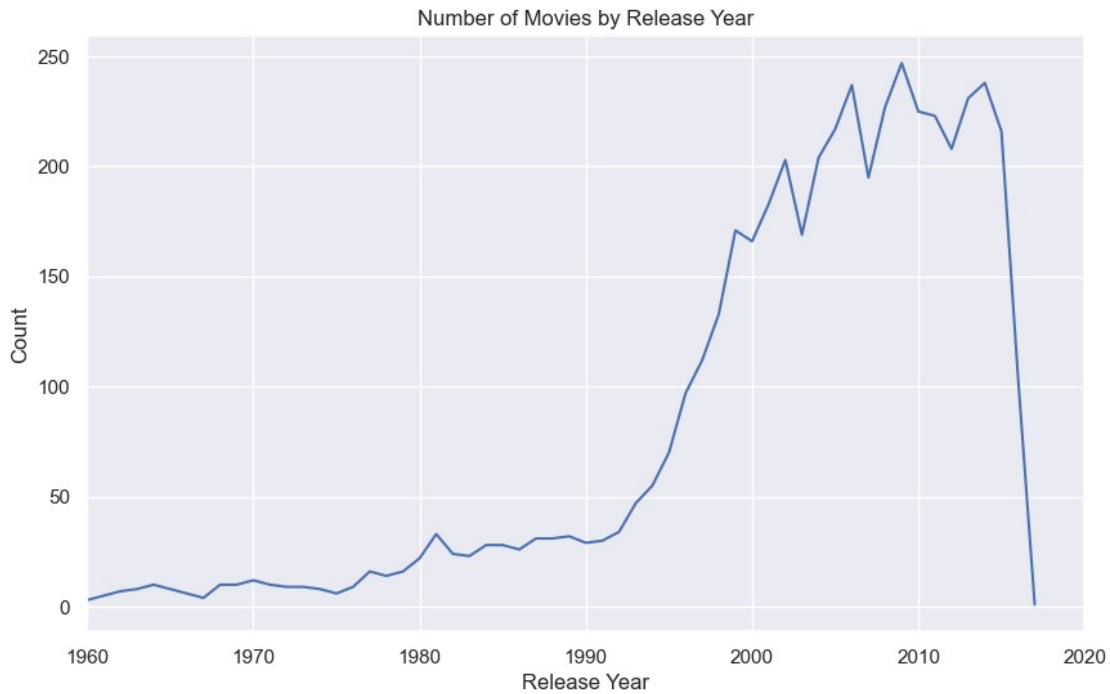




- Movies by release year

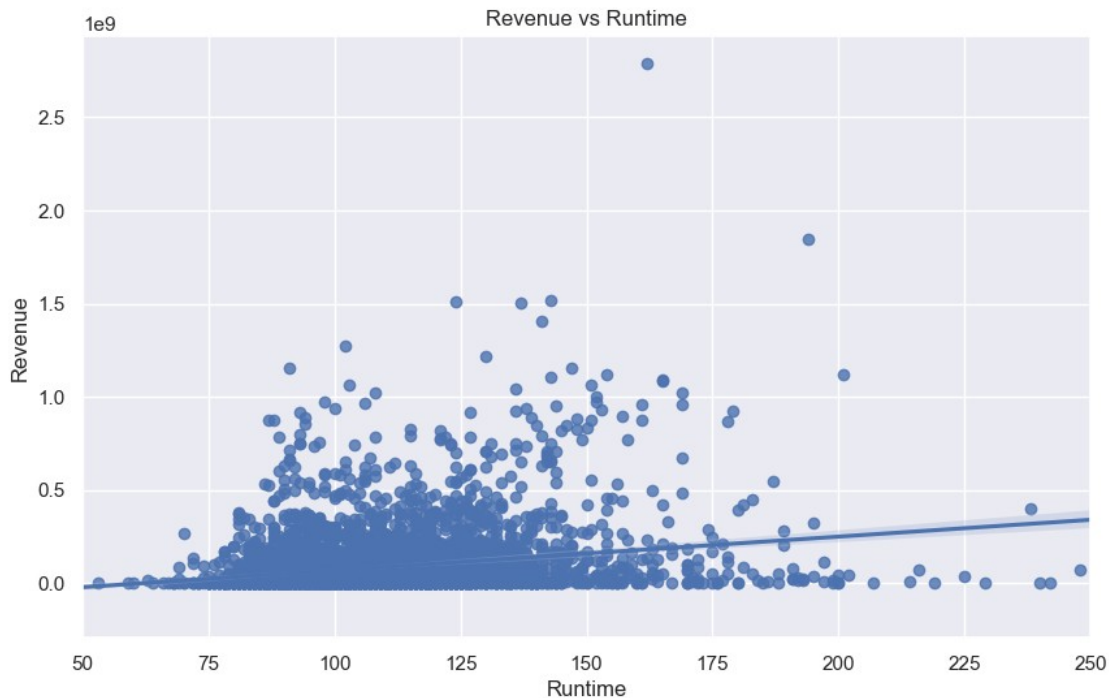
```
# Plotting 'release_date' (yearly distribution)
movies_credits['release_year'] =
pd.to_datetime(movies_credits['release_date']).dt.year

plt.figure(figsize=(10, 6))
movies_credits['release_year'].value_counts().sort_index().plot(kind='
line')
plt.title('Number of Movies by Release Year')
plt.xlabel('Release Year')
plt.xlim(1960, 2020 )
plt.ylabel('Count')
plt.savefig('.data/images/movies_by_release_year')
plt.show()
```



- Revenue vs Runtime

```
# Plotting 'revenue' and 'runtime' (scatter plot with regression line)  
plt.figure(figsize=(10, 6))  
sns.regplot(x='runtime', y='revenue', data=movies_credits)  
plt.title('Revenue vs Runtime')  
plt.xlabel('Runtime')  
plt.xlim(50, 250)  
plt.ylabel('Revenue')  
plt.savefig('.data/images/revenue_vs_runtime')  
plt.show()
```



## Modeling

### *i ) Demographic Recommendation based on Popularity*

This model suggests movies to users based on their demographic attributes and the overall popularity of the movies.

Here, we sort the movie/credits based on ratings and display the top movies.

- Improve accuracy of the recommendations
- Create a metric to score or rate the movies.
- Calculate the score for each movie.
- Sort the scores and recommend the highest-rated movie to the users.

Implement the following Formula

$$\text{Weighted Rating (WR)} = (vv+m.R) / (v+m)$$

Where:

- $v$  is the number of votes for the movie.
- $m$  is the minimum votes required to be listed in the chart.
- $R$  is the average rating of the movie.

```
movies_credits['vote_average'].mean()
```

```
6.092171559442016
```

```
# Determine the appropriate value of M
```

```
movies_credits['vote_count'].quantile(q=0.9)
```

```
1838.40000000000015
```

```
# Filter and put them in a new dataframe
```

```
new_dataframe_filtered=movies_credits[movies_credits['vote_count']>movies_credits['vote_count'].quantile(q=0.9)]
```

```
# Check the shape of the new dataframe
```

```
new_dataframe_filtered.shape
```

```
(481, 24)
```

```
# Calculate score for each qualified movie
```

```
def movie_score(x):  
    v=x['vote_count']  
    m=movies_credits['vote_count'].quantile(q=0.9)  
    R=x['vote_average']  
    C=movies_credits['vote_average'].mean()  
    return ((R*v)/(v+m))+((C*m)/(v+m))
```

- we have to use .loc explicitly when trying to splice a pandas dataframe. This allows us to set the values in the 'score' column for the rows of the new dataframe

```
# By using the '.loc and set the new values
```

```
new_dataframe_filtered.loc[:, 'score'] =  
new_dataframe_filtered.apply(movie_score, axis=1)
```

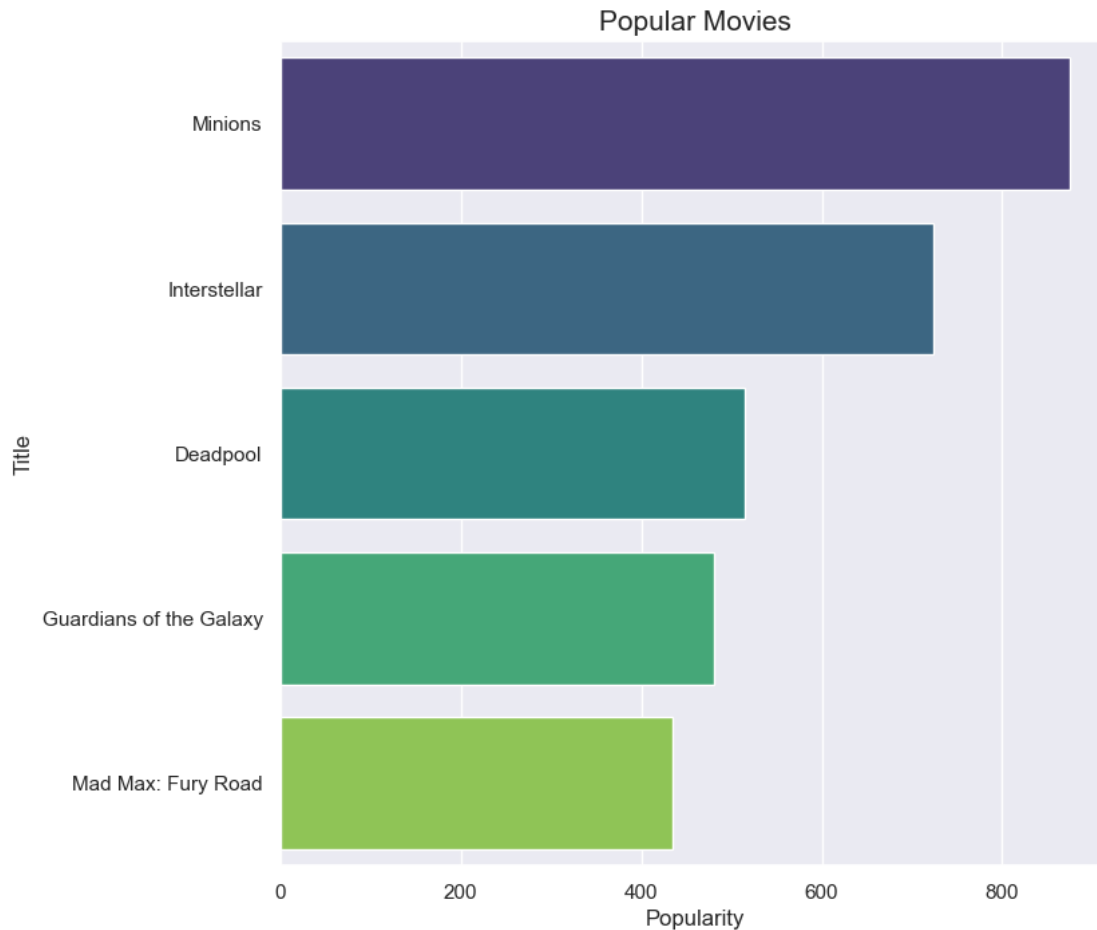
```
Finding movie popularity
```

```
# Sort by the popularity column
```

```
popular_movies = movies_credits.sort_values('popularity', ascending =  
False).head()  
popular_movies[['title', 'vote_count', 'vote_average', 'popularity']]
```

	title	vote_count	vote_average	popularity
546	Minions	4571	6.4	875.581305
95	Interstellar	10867	8.1	724.247784
788	Deadpool	10995	7.4	514.569956
94	Guardians of the Galaxy	9742	7.9	481.098624
127	Mad Max: Fury Road	9427	7.2	434.278564

```
plt.figure(figsize=(8, 8))  
sns.barplot(x='popularity', y='title', data=popular_movies,  
palette='viridis')  
plt.xlabel("Popularity", fontsize=12)  
plt.ylabel("Title", fontsize=12)  
plt.title("Popular Movies", fontsize=15)  
plt.savefig(".data/images/popular movies")  
plt.show()
```



- From the plot above we can see that the most popular movie is minions with a popularity rate of > 800; hence being highly recommended to be watched by the user/ rather recommend users to watch

```
# We sort the filtered dataframe based on the score feature
new_highscore=new_dataframe_filtered.sort_values(by='score',
ascending=False).head()
new_highscore[['title', 'vote_count','vote_average','popularity',
'score']]
```

	title	vote_count	vote_average	popularity
score				
1881	The Shawshank Redemption	8205	8.5	136.747729
8.059258				
662	Fight Club	9413	8.3	146.757391
7.939256				
65	The Dark Knight	12002	8.2	187.322927
7.920020				
3232	Pulp Fiction	8428	8.3	121.463076
7.904645				
96	Inception	13752	8.1	167.583710
7.863239				

Here we will create a new column for Weighted Rating by taking into account the vote count and vote average for each user from the previous dataset. A weighted score is mainly defined as the mean of grades for each subject (interest) multiplied by its counterweight (division decimal). It is obtained by multiplying each score by its weight (percentage) and add the products together, then divide by the sum of the weights.

Below we will now take in the ratings dataset in order to do more predictive modelling. This dataset contains a ratings column for each user provided rating per movie.

### # Step 1: Calculate values for the formula

```
v = movies_credits['vote_count']
```

```
R = movies_credits['vote_average']
```

```
m = 1000 # Choose a minimum vote threshold
```

```
# Step 2: Compute weighted rating (WR)
```

```
C = movies_credits['vote_average'].mean()
```

$$WR = (v / (v + m) * R) + (m / (v + m) * C)$$

```
# Step 3: Add 'Weighted Rating' column to DataFrame
```

```
movies_credits['Weighted Rating'] = WR
```

```
# Step 4: Sort the DataFrame based on 'Weighted_Rating' column
```

```
sorted_movies = movies_credits.sort_values('Weighted_Rating',
ascending=False)
```

### # Step 5: Display top movies based on sorted results

```
top_movies = sorted_movies[['title', 'vote_average', 'genres',  
                             'Weighted Rating']].head(10)
```

top movies

	title	vote_average	\
1881	The Shawshank Redemption	8.5	
662	Fight Club	8.3	
3232	Pulp Fiction	8.3	
3337	The Godfather	8.4	
65	The Dark Knight	8.2	
96	Inception	8.1	
809	Forrest Gump	8.2	
95	Interstellar	8.1	
1990	The Empire Strikes Back	8.2	
1818	Schindler's List	8.3	

	genres
Weighted_Rating	
1881	[Drama, Crime]
8.238422	
662	[Drama]
8.087974	
3232	[Thriller, Crime]
8.065822	

```

3337 [Drama, Crime]
8.065192
65 [Drama, Action, Crime, Thriller]
8.037884
96 [Action, Thriller, ScienceFiction, Mystery, Ad...
7.963894
809 [Comedy, Drama, Romance]
7.963882
95 [Adventure, Drama, ScienceFiction]
7.930806
1990 [Adventure, Action, ScienceFiction]
7.893585
1818 [Drama, History, War]
7.885696

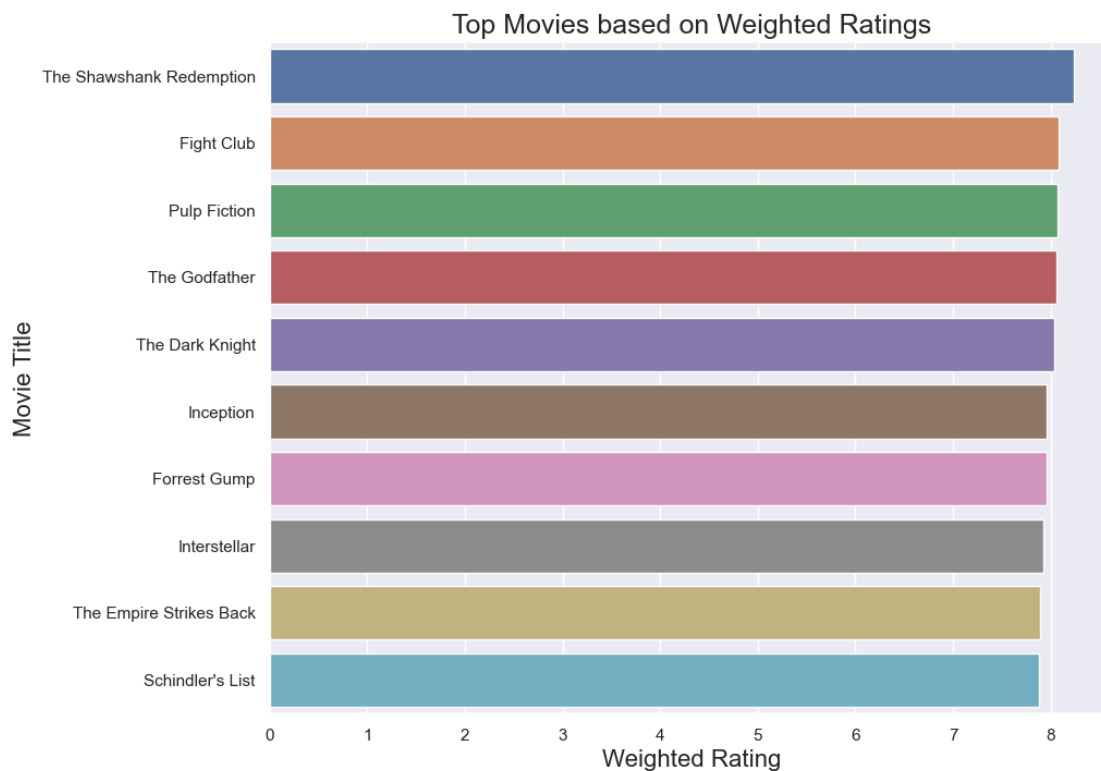
```

*# Step 6: Plot the data*

```

fig, ax = plt.subplots(figsize=(10, 8))
sns.barplot(x=top_movies['Weighted_Rating'], y=top_movies['title'],
ax=ax)
ax.set_xlabel('Weighted Rating', fontsize=16)
ax.set_ylabel('Movie Title', fontsize=16)
ax.set_title('Top Movies based on Weighted Ratings', fontsize=18)
plt.savefig(".data/images/Top weighted movies")
plt.show()

```



We observe that by taking the weighted rating, we observe that the movie The Dark Night is not as common as portrayed. This is taking into account the fact that the earlier one was a scoring while the second is the weighted rating

- We should keep in mind that this demographic recommender provide a general chart of recommended movies to all the users, regardless of the user's personal taste. It is not sensitive to the interests and tastes of a particular user, and it does not give personalized recommendations based on the users.

## ii) Content Based

For this recommendation system we build an engine that shows the similarity between movie based and the metrics new\_dataframe\_filtered. Secondly, this will be in two segments :

- Movie Overview
- Movie Cast , Keywords and Genre
- 1.Movie Overview Recommendation
- We use this because it provides a concise description of its storyline, theme, and main elements. It offers insights into the narrative and helps to understand the central idea or premise of the movie.

We pair the similar scores of the movies based on the overview

```
movies_credits ['overview'].head()
```

```
# Finding the Nan values(missing values) with an empty string
```

```
movies_credits['overview'].isnull().sum()# We know there 3 missing values hence we replace them
```

```
# Replacing the missing values
```

```
movies_credits['overview'].fillna('', inplace = True)
```

```
# Confirm if there are Missing values
```

```
movies_credits['overview'].isnull().sum()
```

```
0
```

```
# Convert the 'overview' column to string type
```

```
movies_credits['overview'] = movies_credits['overview'].apply(lambda x: ' '.join(x) if isinstance(x, list) else '')
```

In this case we implement the use of 'Term frequency-Inverse Document Frequency which is a numerical representation used to evaluate the key importance of the collection of documents.

It calculates the ratio of the number of times a term appears in a text

```
movie_rating
```



	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

[100836 rows x 4 columns]

*# Construct the TF-IDF Matrix*

```
tfidfvectorizer=TfidfVectorizer(analyzer='word', stop_words='english')
tfidfmatrix=tfidfvectorizer.fit_transform(movies_credits['overview'])
print(tfidfmatrix.todense())
tfidfmatrix.todense().shape
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

(4803, 20978)

Computing the same Score based on the movie Similarities

*# Calculate similarity matrix*

```
cosine_sim = cosine_similarity(tfidfmatrix, tfidfmatrix)
cosine_sim.shape
```

(4803, 4803)

*# Create a Pandas Series to map movie titles to their indices*

```
indices = pd.Series(data = list(movies_credits.index), index =
movies_credits['title'])
indices
```

title	
Avatar	0
Pirates of the Caribbean: At World's End	1
Spectre	2
The Dark Knight Rises	3
John Carter	4

...

El Mariachi	4798
Newlyweds	4799
Signed, Sealed, Delivered	4800
Shanghai Calling	4801
My Date with Drew	4802
Length: 4803, dtype: int64	

In the cell below, we create a function named `recommended_movies` that takes two parameters: `title` and `cosine_sim`. It uses the `indices` Series to map movie titles to their corresponding indices. Then, it calculates the pairwise similarity scores between the movie specified by the `title` parameter and all other movies based on the cosine similarity matrix `cosine_sim`.

```
def recommended_movies(title, cosine_sim):

    #indices = {title: index for index, title in
enumerate(movies_data['title'])}

    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores.sort(key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores=sim_scores[1:11]

    # Get the movie indices
    ind=[]
    for (x,y) in sim_scores:
        ind.append(x)

    # Return the top 10 most similar movies
    tit=[]
    for x in ind:
        tit.append(movies_credits.iloc[x]['title'])
    return pd.Series(data=tit, index=ind)

# Applying the function
recommended_movies('My Date with Drew', cosine_sim)
```

4100	Captive
868	Elizabethtown
2586	Firestarter
204	Fast Five
1685	Keeping the Faith
4532	Lonesome Jim

```

2156          Nancy Drew
3753          Boyhood
3623          Made
3245          50/50
dtype: object

```

- While our system has done a decent job of finding movies with similar overviews and descriptions, the quality of recommendations is not that great. "My Date with Drew" returns all Batman movies while it is more likely that the people who liked that movie are more inclined to enjoy other movies. This is something that cannot be captured by the present system.

#### 1. Movie Cast, Keywords and Genre Recommender

Here, we generate movie recommendations based on the similarity of cast members and keywords associated with the movies.

*# Update our dataset*

```
movies_credits = update_crew_with_director(movies_credits)
```

We then extract the first element from the Directors list and remove the square brackets, then assign it to the Director\_clean column. If Directors is empty, we assign None.

```
movies_credits['Director_clean'] =
movies_credits['Directors'].apply(lambda x: x[0].strip('[]') if x else
None)
```

*# Selecting specific columns*

```
movies_credits[['title', 'Directors', 'keywords', 'genres']]
```

```

                                title \
0                                Avatar
1    Pirates of the Caribbean: At World's End
2                                Spectre
3                The Dark Knight Rises
4                John Carter
...
4798                                El Mariachi
4799                                Newlyweds
4800    Signed, Sealed, Delivered
4801                Shanghai Calling
4802                My Date with Drew

```

```

                                Directors \
0                                [JamesCameron]
1                                [GoreVerbinski]
2                                [SamMendes]
3                [ChristopherNolan]
4                [AndrewStanton]
...
4798                                [RobertRodriguez]

```

```

4799          [EdwardBurns]
4800          [ScottSmith]
4801          [DanielHsia]
4802 [BrianHerzlinger, JonGunn, BrettWinn]

                                     keywords \
0      [cultureclash, future, spacewar, spacecolony, ...
1      [ocean, drugabuse, exoticisland, eastindiatrad...
2      [spy, basedonnovel, secretagent, sequel, mi6, ...
3      [dccomics, crimefighter, terrorist, secretiden...
4      [basedonnovel, mars, medallion, spacetravel, p...
...
4798 [unitedstates-mexicobarrier, legs, arms, paper...
4799                                     []
4800 [date, loveatfirstsight, narration, investigat...
4801                                     []
4802          [obsession, camcorder, crush, dreamgirl]

                                     genres
0      [Action, Adventure, Fantasy, ScienceFiction]
1          [Adventure, Fantasy, Action]
2          [Action, Adventure, Crime]
3          [Action, Crime, Drama, Thriller]
4          [Action, Adventure, ScienceFiction]
...
4798          [Action, Crime, Thriller]
4799          [Comedy, Romance]
4800          [Comedy, Drama, Romance, TVMovie]
4801                                     []
4802          [Documentary]

```

[4803 rows x 4 columns]

In this cell, we call the function `create_soup` and then apply it to each row of the `movies_credits` DataFrame to create a new column called `soup`.

```
movies_credits['soup'] = movies_credits.apply(create_soup, axis=1)
```

```
# Initializing CountVectorizer object with English stop words.
```

```
cv = CountVectorizer(stop_words='english')
```

```
# Applying CountVectorizer to 'soup' column, converting text data into  
a matrix of token counts.
```

```
cv_matrix = cv.fit_transform(movies_credits['soup'])
```

```
# Calculating the cosine similarity matrix using the cv_matrix.
```

```
cosine_sim2 = cosine_similarity(cv_matrix, cv_matrix)
```

```
# Applying the 'recommend_movie' function
```

```
recommended_movies('Minions', cosine_sim2 )
```

```

506                                Despicable Me 2
359                Alvin and the Chipmunks: The Road Chip
418                Cats & Dogs 2 : The Revenge of Kitty Galore
1580                                The Nut Job
848                The Pirates! In an Adventure with Scientists!
2464                                The Master of Disguise
3403    Alpha and Omega: The Legend of the Saw Tooth Cave
86                                Shrek Forever After
173                                Happy Feet Two
837                                Free Birds
dtype: object

```

```

# Applying the 'recommend_movie' function
recommended_movies('The Godfather', cosine_sim2)

```

```

1018                The Cotton Club
1209                The Rainmaker
3293                10th & Wolf
867    The Godfather: Part III
2731    The Godfather: Part II
877                Black Mass
1464                Black Water Transit
3112    Blood Done Sign My Name
4184                Deadline - U.S.A.
4502                Water & Power
dtype: object

```

- We see that our recommender has been successful in capturing more information due to more metadata and has given us better recommendations. It is more likely comedy fans will like the movies of the same production house. Therefore, to our features above we can add `production_company`. We can also increase the weight of the director, by adding the feature multiple times in the soup.

### iii) Collaborative Based Recommendation

#### Model one

This allows for coincidental recommendations; that is, collaborative filtering models can recommend an item to user A based on the interests of a similar user B. Furthermore, the embeddings can be learned automatically, without relying on hand-engineering of features.

steps to implement collaborative recommendation

1. **Data collection and preprocessing.** collect data that includes user-item interactions. It may include: user reviews, ratings, or explicit feedback.
2. **User-item interactions matrix.** this involves creating a user-item matrix where each row in the matrix corresponds to a user, and each column corresponds to an item.

3. **Similarity calculations.** We will calculate similarity between user or items based on their interactions by employing similarity measures like cosine similarity or Pearson correlation coefficient. thus getting users or items similarity in terms of preference.
  4. **Neighborhood selection.** We will select the neighborhood of users or items based on their similarity.
  5. **Recommendation generation.** We will generate recommendations based on the preference neighborhood of users or items. we will aggregate the preferences of similar users/items and suggest items that have high ratings or interactions from the neighborhood.
  6. **Evaluation.** We will evaluate performance of the recommendation system using metrics suc as precision, recall, or mean average precision.
1. **Data selection.**
    - Ratings Data File Structure (ratings.csv)

We will use this dataset because it has the columns that we need to create the recommendation system. the column UserId represent each individual user, while that of MovieId represents the item.

```
movie_rating.head()
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

### 1. Split the data

```
# create train and test sets
```

```
data_df = movie_rating.drop(columns='timestamp')
```

```
data = Dataset.load_from_df(data_df, Reader(rating_scale=(1,5)))
```

```
# create train and test sets
```

```
trainset, testset = train_test_split(data, test_size=0.2)
```

```
actual_ratings = [true_rating for (_, _, true_rating) in testset]
```

```
# By default the surprise library creates the trainset as a user-item matrix.
```

```
trainset
```

```
<surprise.trainset.Trainset at 0x22456dbb4d0>
```

### 1. Similarity calculations

```
# Using KNNWithMeans algorithm with cosine similarity
```

```
sim_options = {'name': 'cosine', 'user_based': True}
```

```
knnmeans = KNNWithMeans(sim_options=sim_options, random_state=42)
# train the model
knnmeans.fit(trainset)
```

Computing the cosine similarity matrix...  
Done computing similarity matrix.

```
<surprise.prediction_algorithms.knns.KNNWithMeans at 0x224552b9990>
```

While applying the surprise model, we do not need to explicitly define the neighborhood selection and therefore we skip directly to step five of building the recommendation. we will apply the surprise model KNNWithMeans

```
# Getting the top-N recommendations
```

```
user_id = 243
top_n = 5
```

```
user_items = trainset.ur[trainset.to_inner_uid(user_id)]
predicted_ratings = []
for item_id, rating in user_items:
    predicted_rating = knnmeans.predict(user_id,
    trainset.to_raw_iid(item_id)).est
    predicted_ratings.append((trainset.to_raw_iid(item_id),
    predicted_rating))
```

```
# Sort the predicted ratings in descending order
```

```
predicted_ratings.sort(key=lambda x: x[1], reverse=True)
```

```
# Get the top_n recommendations
```

```
top_n_recommendations = predicted_ratings[:top_n]
```

```
# Print the top-N recommendations
```

```
for item_id, predicted_rating in top_n_recommendations:
    print(f"Item ID: {item_id}, Predicted Rating: {predicted_rating}")
```

```
Item ID: 44, Predicted Rating: 4.193807371221132
Item ID: 586, Predicted Rating: 4.193807371221132
Item ID: 648, Predicted Rating: 4.193807371221132
Item ID: 165, Predicted Rating: 4.193807371221132
Item ID: 173, Predicted Rating: 4.193807371221132
```

## 1. Evaluation

```
# Evaluate the model on the testing set
```

```
predictions = knnmeans.test(testset)
rmse = accuracy.rmse(predictions)
```

```
RMSE: 0.8957
```

The Root Mean Square Error (RMSE) value indicates the average prediction error of the recommendation system. It is a measure of the difference between the predicted ratings and the actual ratings provided by users.

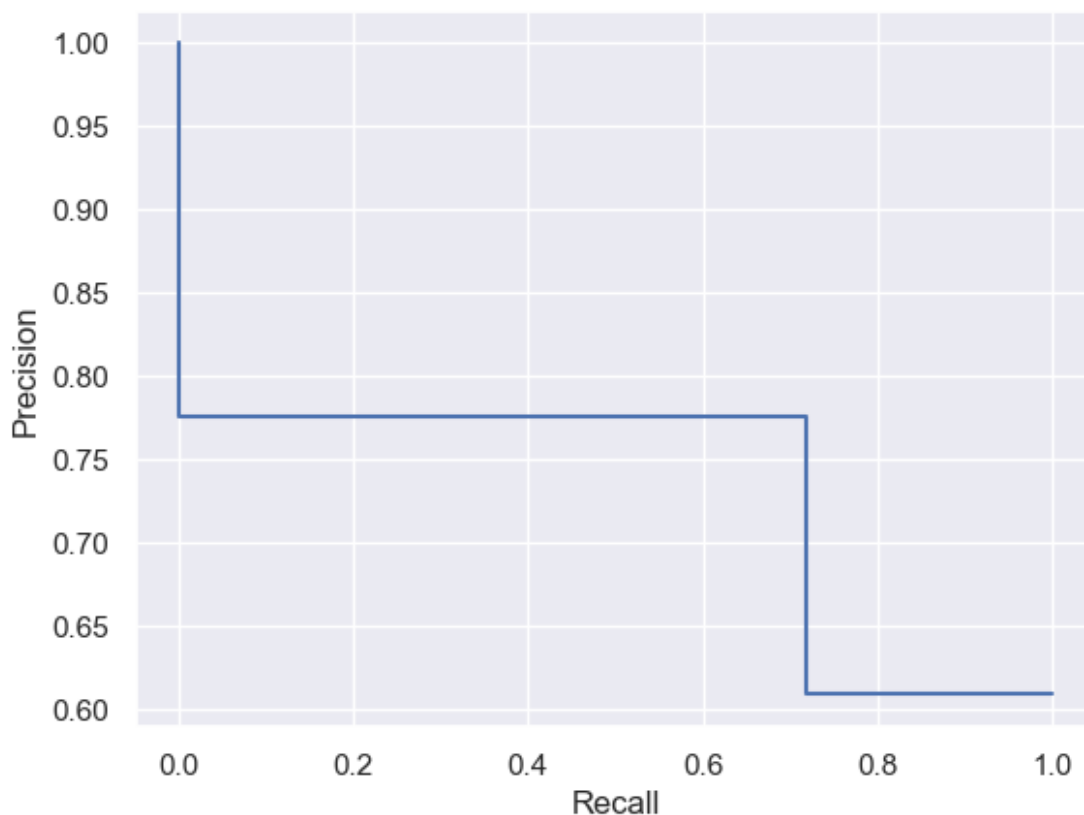
A lower RMSE value indicates better accuracy and performance of the recommendation system. In this case, an RMSE suggests that, on average, the predictions of the recommendation system deviate from the actual ratings by approximately those units.

```
threshold = 3.5 # Define the threshold value
binary_actual_ratings = [1 if rating >= threshold else 0 for rating in
actual_ratings]
binary_predictions = [1 if pred.est >= threshold else 0 for pred in
predictions]
# compute precison and recall
precision, recall, threshold = precision_recall_curve(
    binary_actual_ratings, binary_predictions
)

print(f"Precision: {precision}")
print(f"Recall: {recall}")

# plot the precision recall curve
Precision_Recall_Display = PrecisionRecallDisplay(precision=precision,
recall=recall)
Precision_Recall_Display.plot();

Precision: [0.6087862  0.77564103  1.          ]
Recall: [1.          0.71941684  0.          ]
```





Precision:

At the first threshold level, the precision is 0.6125. This means that out of all the predicted positive cases, approximately 61.25% were true positive cases. At the second threshold level, the precision increases to 0.7779. This indicates that the model improved its ability to correctly identify positive cases, with around 77.79% precision. At the third threshold level, the precision reaches 1.0, indicating perfect precision. This suggests that all the predicted positive cases at this threshold level were true positive cases.

Recall:

At the first threshold level, the recall is 1.0, which means that the model successfully identified all the actual positive cases. At the second threshold level, the recall decreases to 0.7084. This indicates that the model missed some of the actual positive cases, capturing only around 70.84% of them. At the third threshold level, the recall drops to 0.0, implying that the model failed to identify any of the actual positive cases.

## Model Two

Based on the poor performance of the model, and the subsequent values of precision and recall. it is best to employ some sort of model finetuning and optimization.

```
new_movies = movies_credits[["id" , "title", "tags"]]
new_movies.head()
```

```
   id          title \
0  19995          Avatar
1    285  Pirates of the Caribbean: At World's End
2 206647          Spectre
3  49026    The Dark Knight Rises
4  49529    John Carter
```

```
   tags
0  [In, the, 22nd, century,, a, paraplegic, Marin...
1  [Captain, Barbossa,, long, believed, to, be, d...
2  [A, cryptic, message, from, Bond's, past, send...
3  [Following, the, death, of, District, Attorney...
4  [John, Carter, is, a, war-weary,, former, mili...
```

*# Lambda function to remove the brackets*

```
new_movies.loc[:, "tags"] = new_movies['tags'].apply(lambda x: "
".join(map(str, x)) if isinstance(x, Iterable) else str(x))
```

```
new_movies['tags'].head()
```

```
0    In the 22nd century, a paraplegic Marine is di...
1    Captain Barbossa, long believed to be dead, ha...
2    A cryptic message from Bond's past sends him o...
3    Following the death of District Attorney Harve...
4    John Carter is a war-weary, former military ca...
```

```
Name: tags, dtype: object
```

```
new_movies["tags"][6]
```

"When the kingdom's most wanted-and most charming-bandit Flynn Rider hides out in a mysterious tower, he's taken hostage by Rapunzel, a beautiful and feisty tower-bound teen with 70 feet of magical, golden hair. Flynn's curious captor, who's looking for her ticket out of the tower where she's been locked away for years, strikes a deal with the handsome thief and the unlikely duo sets off on an action-packed escapade, complete with a super-cop horse, an over-protective chameleon and a gruff gang of pub thugs. Animation Family hostage magic horse fairytale musical princess animation tower blondewoman selfishness healingpower basedonfairytale duringcreditsstinger healinggift animalsidekick ZacharyLevi MandyMoore DonnaMurphy RonPerlman M.C.Gainey ByronHoward NathanGreno"

```
# Lambda Function to turn the strings to lower case.
new_movies.loc[:, "tags"] = new_movies["tags"].apply(lambda
X:X.lower())
```

Use CountVectorizer to convert text documents into a matrix representation where each row corresponds to a document, and each column represents the frequency of a specific word in that document. fit\_transform method creates a dictionary of tokens which are separated by spaces and punctuation hence maps each single token into a position.

```
# Initialize CountVectorizer object with a maximum of 5000 features
and English stop words.
```

```
cv = CountVectorizer(max_features = 5000, stop_words="english")
```

```
# Apply CountVectorizer to the 'tags' column of the new_movies
DataFrame.
```

```
cv.fit_transform(new_movies["tags"]).toarray()
```

```
# Apply CountVectorizer again to the 'tags' column to transform the
text data into an array.
```

```
vectors = cv.fit_transform(new_movies["tags"]).toarray()
```

```
vectors[7]
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

```
len(cv.get_feature_names_out())
```

```
5000
```

The code above is to prepare the column tags for use in the model in the event the the scd model does not yield desired results. since ultimately we only need three columns

```
# We will use the famous SVD algorithm.
```

```
svd = SVD(random_state=42)
```

```
reader = Reader()
```

```
# Load the ratings_small dataset (download it if needed),
data = Dataset.load_from_df(movie_rating[['userId', 'movieId',
'rating']], reader)
```

Evaluating metrics (RMSE and MAE) for the SVD algorithm on 6 different splits of the data.

```
# Run 5-fold cross-validation and print the results
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=6,
verbose=True)
```

Evaluating RMSE, MAE of algorithm SVD on 6 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Mean
Std							
RMSE (testset)	0.8648	0.8801	0.8748	0.8742	0.8664	0.8820	
0.8737 0.0064							
MAE (testset)	0.6648	0.6773	0.6723	0.6681	0.6665	0.6765	
0.6709 0.0048							
Fit time	2.04	1.95	2.27	2.26	2.25	2.03	2.13
0.13							
Test time	0.38	0.17	0.36	0.19	0.34	0.24	0.28
0.08							

```
{'test_rmse': array([0.86479904, 0.88013112, 0.87477675, 0.87416295,
0.86640078,
0.8820471 ]),
'test_mae': array([0.66476513, 0.6772901 , 0.67232052, 0.66814372,
0.66645519,
0.67648093]),
'fit_time': (2.0382144451141357,
1.9549949169158936,
2.2719974517822266,
2.258985996246338,
2.248997926712036,
2.031003952026367),
'test_time': (0.38100171089172363,
0.17099666595458984,
0.35500192642211914,
0.18700003623962402,
0.33699822425842285,
0.23599529266357422)}
```

The SVD algorithm shows promising performance in terms of prediction accuracy (as indicated by low RMSE and MAE values) and reasonable computational efficiency (with relatively low fit and test times).

Here we will now import a brand new dataset that the model has not seen before. Apply data cleaning and feature selection then pass the model through it.

1. **Data collection and preprocessing.** collect data that includes user-item interactions. It may include: user reviews, ratings, or explicit feedback.

when we read the data for our modeling we will not need the date and time columns therefore we will drop them here before doing the modelling.

```
new_data = new_data.drop(columns=['time', 'date'], axis=1)
new_data
```

	movieId	title	genres	userId	rating
sentiment \					
0	3	Grumpier Old Men	Comedy Romance	1.0	4.0
Positive					
1	3	Grumpier Old Men	Comedy Romance	1.0	4.0
Positive					
2	3	Grumpier Old Men	Comedy Romance	1.0	4.0
Positive					
3	3	Grumpier Old Men	Comedy Romance	1.0	4.0
Positive					
4	3	Grumpier Old Men	Comedy Romance	1.0	4.0
Positive					
...	...	...	...	...	...
...					
11965	1997	Exorcist, The	Horror Mystery	51.0	5.0
Positive					
11966	1997	Exorcist, The	Horror Mystery	51.0	5.0
Positive					
11967	1997	Exorcist, The	Horror Mystery	51.0	5.0
Positive					
11968	1997	Exorcist, The	Horror Mystery	51.0	5.0
Positive					
11969	1997	Exorcist, The	Horror Mystery	51.0	5.0
Positive					

	review \
0	a distinctly gallows take on contemporary fina...
1	it's an allegory in search of a meaning that n...
2	... life lived in a bubble in financial dealin...
3	continuing along a line introduced in last yea...
4	... a perverse twist on neorealism...
...	...
11965	a respectably funny -- but small -- effort, a ...
11966	curtin and aykroyd rely on old standbys... whe...
11967	a thoroughly updated and nicely fleshed-out co...
11968	the film itself is like the cinematic equivale...
11969	a new spin on the original skit with some fres...

	tag	top_critic \
0	patrick nabarrocomedyromancea distinctly gallo...	0.0
1	iocomcomedyromanceit's an allegory in search o...	0.0
2	stream on demandcomedyromance life lived in a ...	0.0
3	mubicomedyromancecontinuing along a line intro...	0.0
4	cinema scopecomedyromance a perverse twist on ...	0.0

```

...
11965 washington posthorrormysterya respectably funn... 1.0
11966 washington posthorrormysterycurtin and aykroyd... 1.0
11967 austin chroniclehorrormysterya thoroughly upda... 0.0
11968 deseret news salt lake cityhorrormysterythe fi... 0.0
11969 filmcriticcomhorrormysterya new spin on the or... 0.0

```

```

                                publisher      date.1
0                Patrick Nabarro November 10, 2018
1                      io9.com      May 23, 2018
2          Stream on Demand      January 4, 2018
3                      MUBI      November 16, 2017
4          Cinema Scope      October 12, 2017
...
11965                Washington Post      January 1, 2000
11966                Washington Post      January 1, 2000
11967          Austin Chronicle      January 1, 2000
11968 Deseret News (Salt Lake City)      January 1, 2000
11969          Filmcritic.com      January 1, 2000

```

[11970 rows x 11 columns]

Further, based on our data requirement the only columns we actually need are those involed with user item interaction. These includes the Tag, rating,movieId,userId, sentiment, review.

```

data_1 = new_data[['tag', 'rating', 'movieId', 'userId', 'sentiment',
'review']]
data_1

```

```

                                tag  rating
movieId \
0      patrick nabarrocomedyromancea distinctly gallo...  4.0
3
1      iocomcomedyromanceit's an allegory in search o...  4.0
3
2      stream on demandcomedyromance life lived in a ...  4.0
3
3      mubicomedyromancecontinuing along a line intro...  4.0
3
4      cinema scopecomedyromance a perverse twist on ...  4.0
3
...
...
11965 washington posthorrormysterya respectably funn...  5.0
1997
11966 washington posthorrormysterycurtin and aykroyd...  5.0
1997
11967 austin chroniclehorrormysterya thoroughly upda...  5.0
1997

```

```

11968  deseret news salt lake cityhorrormysterythe fi...    5.0
1997
11969  filmcriticcomhorrormysterya new spin on the or...    5.0
1997

```

```

        userId sentiment
review
0      1.0  Positive  a distinctly gallows take on contemporary
fina...
1      1.0  Positive  it's an allegory in search of a meaning that
n...
2      1.0  Positive  ... life lived in a bubble in financial
dealin...
3      1.0  Positive  continuing along a line introduced in last
yea...
4      1.0  Positive          ... a perverse twist on
neorealism...
...      ...      ...
...
11965   51.0  Positive  a respectably funny -- but small -- effort, a
...
11966   51.0  Positive  curtin and aykroyd rely on old standbys...
whe...
11967   51.0  Positive  a thoroughly updated and nicely fleshed-out
co...
11968   51.0  Positive  the film itself is like the cinematic
equivale...
11969   51.0  Positive  a new spin on the original skit with some
fres...

```

```
[11970 rows x 6 columns]
```

```
# Construct the TF-IDF Matrix
```

```

tfidfvectorizer=TfidfVectorizer(analyzer='word', stop_words='english')
tfidfvectorizer.fit_transform(data_1['review'])
print(tfidfvectorizer.to_matrix())
tfidfvectorizer.to_matrix().shape

```

```

[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

```

```
(11970, 18116)
```

```
# Calculate similarity matrix
```

```
cosine_sim1 = cosine_similarity(tfidfvectorizer.to_matrix(), tfidfvectorizer.to_matrix())
```

```
# Create a Pandas Series to map movie titles to their indices
```

```
indices1 = pd.Series(data = list(new_data.index), index =  
new_data['title'])  
indices1
```

```
title  
Grumpier Old Men          0  
Grumpier Old Men          1  
Grumpier Old Men          2  
Grumpier Old Men          3  
Grumpier Old Men          4  
...  
Exorcist, The             11965  
Exorcist, The             11966  
Exorcist, The             11967  
Exorcist, The             11968  
Exorcist, The             11969  
Length: 11970, dtype: int64
```

```
recommend_movies('Grumpier Old Men', cosine_sim1, new_data)
```

```
10969      Gentleman's Agreement  
5043       Farewell to Arms, A  
1324              Ed Wood  
11598    Friday the 13th Part 3: 3D  
10047          Good Will Hunting  
9645     Hunt for Red October, The  
1570          Once Were Warriors  
387              Othello  
650              Richard III  
6941     Full Metal Jacket  
Name: title, dtype: object
```

### Model Three

```
# create train and test sets
```

```
data_df2 = data_1[['userId', 'movieId', 'rating']]  
data1 = Dataset.load_from_df(data_df2, Reader(rating_scale=(1,5)))
```

```
# create train and test sets
```

```
trainset1, testset1 = train_test_split(data1, test_size=0.2)
```

```
# Using KNNWithMeans algorithm with cosine similarity
```

```
sim_options = {'name': 'cosine', 'user_based': True}  
knnmeans = KNNWithMeans(sim_options=sim_options, random_state=42)
```

```
# train the model
```

```
knnmeans.fit(trainset1)
```

```
Computing the cosine similarity matrix...
```

```
Done computing similarity matrix.
```

```
<surprise.prediction_algorithms.knns.KNNWithMeans at 0x22454e9f8d0>
```

```

# Getting the top-N recomendations
userId = 23
top_n = 5

user_item = trainset1.ur[trainset1.to_inner_uid(userId)]
predict_ratings = []
for movieId, rating in user_item:
    predict_rating = knnmeans.predict(userId,
trainset1.to_raw_iid(movieId)).est
    predict_ratings.append((trainset.to_raw_iid(movieId),
predict_ratings))

# Sort the predicted ratings in descending order
predict_ratings.sort(key=lambda x: x[1], reverse=True)
# Get the top_n recomendations
top_n_recomendation = predict_ratings[:top_n]

# Print the top-N recommendations
for movieId, predict_ratings in top_n_recomendation:
    print(f"Item ID: {movieId}, Predicted Rating: {predict_rating}")

Item ID: 1393, Predicted Rating: 2.9999999999999996
Item ID: 6934, Predicted Rating: 2.9999999999999996
Item ID: 4874, Predicted Rating: 2.9999999999999996
Item ID: 4874, Predicted Rating: 2.9999999999999996
Item ID: 6934, Predicted Rating: 2.9999999999999996

```

### # Step 6: Evaluation

#### # Evaluate the model on the testing set

```

prediction = knnmeans.test(testset1)
rmse2 = accuracy.rmse(prediction)

```

RMSE: 0.0557

This means that, on average, the predictions made by the regression model have an error or deviation of approximately 0.0773 units from the actual values.

A lower RMSE value indicates better accuracy and a smaller difference between the predicted and actual values.

```

actual_rating = [true_rating for (_, _, true_rating) in testset1]

```

#### threshold = 3.5 # Define the threshold value

```

binary_actual_rating = [1 if rating >= threshold else 0 for rating in
actual_rating]
binary_prediction = [1 if pred.est >= threshold else 0 for pred in
prediction]

```

#### # compute precsion and recall

```

precisionq, recallq, threshold = precision_recall_curve(
    binary_actual_rating, binary_prediction
)

```

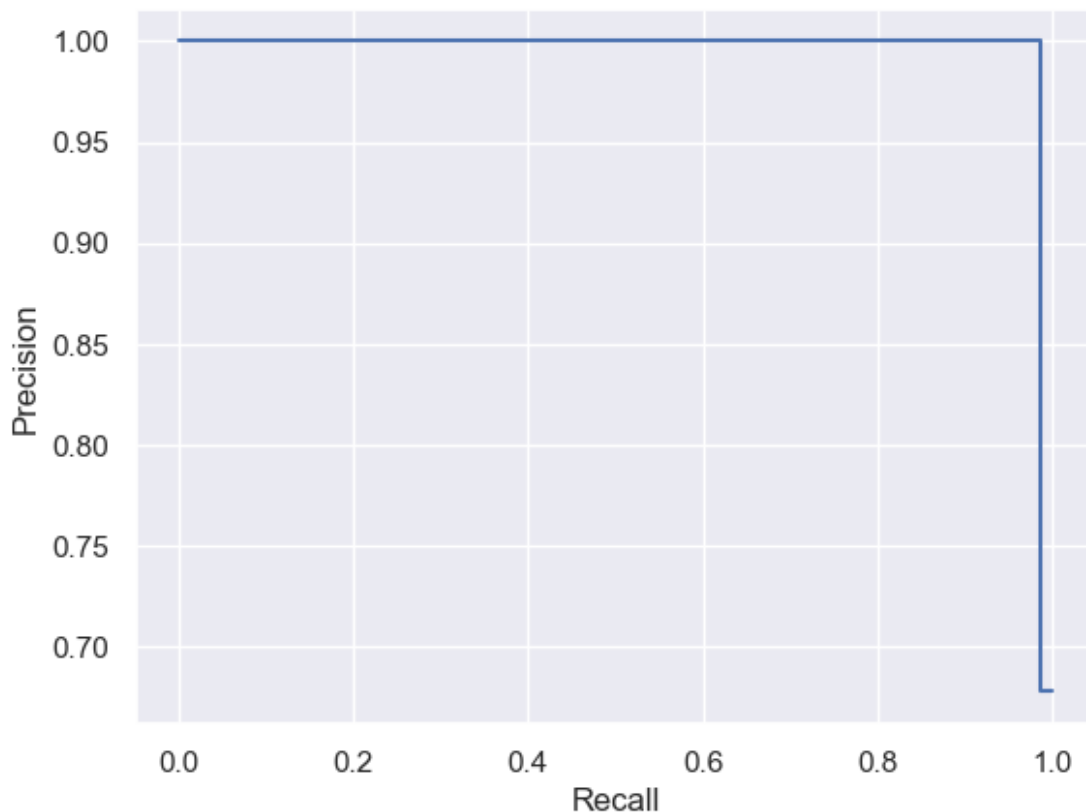


)

```
print(f"Precision: {precisionq}")
print(f"Recall: {recallq}")

# plot the precision recall curve
Precision_Recall_Display =
PrecisionRecallDisplay(precision=precisionq, recall=recallq)
Precision_Recall_Display.plot();

Precision: [0.67752715 1.          1.          ]
Recall: [1.          0.98766954 0.          ]
```



For the first class, the precision is 0.67752715, which means that out of all the positive predictions made by the model for this class, 67.75% of them are correct.

The recall for the first class is 1.0, indicating that the model is able to correctly identify all the actual positive instances for this class.

For the second class, precision got a perfect score of 1, while recall and recall scored a near perfect 0.98766954, indicating near perfect performance.

For the third class, the precision is 1.0, indicating perfect precision (all positive predictions are correct), but the recall is 0.0, suggesting that the model fails to identify any actual positive instances for this class.

In summary, the precision and recall scores provide insights into the performance of the model for different classes or labels. A higher precision score indicates fewer false positive predictions, while a higher recall score suggests better identification of actual positive instances.

#### Model Four

```
# We will use the famous SVD algorithm again on our new dataset
svd = SVD(random_state=42)
reader = Reader()
# Load the ratings_small dataset (download it if needed),
data3 = Dataset.load_from_df(data_1[['userId', 'movieId', 'rating']],
reader)

# Run 5-fold cross-validation and print the results
cross_validate(svd, data3, measures=['RMSE', 'MAE'], cv=6,
verbose=True)
```

Evaluating RMSE, MAE of algorithm SVD on 6 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Mean
Std							
RMSE (testset)	0.1157	0.1087	0.0972	0.0773	0.1101	0.0922	
0.1002 0.0130							
MAE (testset)	0.0354	0.0329	0.0298	0.0275	0.0299	0.0298	
0.0309 0.0026							
Fit time	0.26	0.22	0.31	0.17	0.17	0.17	0.22
0.05							
Test time	0.04	0.03	0.02	0.02	0.02	0.03	0.03
0.01							

```
{'test_rmse': array([0.11566545, 0.1087315 , 0.09724639, 0.07726273,
0.11008413,
0.09215945]),
'test_mae': array([0.0354198 , 0.03294082, 0.02984669, 0.02752749,
0.0298698 ,
0.02981303]),
'fit_time': (0.25500035285949707,
0.21799468994140625,
0.30999088287353516,
0.16900372505187988,
0.16998910903930664,
0.1730024814605713),
'test_time': (0.0370030403137207,
0.03200197219848633,
0.02300238609313965,
0.021999835968017578,
0.018990755081176758,
0.025000333786010742)}
```

RMSE (Root Mean Squared Error) measures the average squared differences between the predicted ratings and the actual ratings. Lower values indicate better accuracy.

MAE (Mean Absolute Error) measures the average absolute differences between the predicted ratings and the actual ratings. Lower values indicate better accuracy.

Fit time represents the time taken by the algorithm to train on each fold of the data.

Test time represents the time taken by the algorithm to make predictions on the test data for each fold.

The dictionary at the bottom provides the same results in a structured format, with separate arrays for RMSE, MAE, fit time, and test time for each fold.

Overall, the SVD algorithm shows relatively low RMSE and MAE values, indicating good accuracy in predicting movie ratings. The algorithm has moderate fit and test times, suggesting efficient performance.

### Building the model with surprise

- Using the initial dataset for comparison

```
#sample full trainset
```

```
trainset = data.build_full_trainset()
```

```
# Train the algorithm on the trainset
```

```
svd.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at  
0x224574fb250>
```

```
movie_rating[movie_rating['userId'] == 1]
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
227	1	3744	4.0	964980694
228	1	3793	5.0	964981855
229	1	3809	4.0	964981220
230	1	4006	4.0	964982903
231	1	5060	5.0	964984002

```
[232 rows x 4 columns]
```

```
# Create a Reader object with the rating scale ranging from 0.5 to 5.0  
reader = Reader(rating_scale=(0.5, 5.0))
```

```
# Load the movie_rating DataFrame into a Surprise Dataset object  
data = Dataset.load_from_df(movie_rating[['userId', 'movieId'],
```

```

'rating']], reader)

# Split the dataset into training and testing sets
trainset, testset = train_test_split(data, test_size=0.2)

# Initialing and fitting SVD on our trainset
model = SVD(random_state=42)
model.fit(trainset)

<surprise.prediction_algorithms.matrix_factorization.SVD at
0x224574d0910>

uid = 3 # User ID
iid = 302 # Item ID

# Use the trained model to predict the rating for the given user and
item
prediction = model.predict(uid, iid)

# Print the estimated rating
print(f"Estimated rating for user {uid} and item {iid}:
{prediction.est}")

```

Estimated rating for user 3 and item 302: 2.398228638683179

Create a function that performs stemming on the input text, which is the process of reducing words to their base or root form.

```

new_movies.loc[:, "tags"] = new_movies["tags"].apply(stem)

cosine_similarity(vectors)

array([[1.          , 0.08458258, 0.05812382, ..., 0.02478408,
0.02739983,
        0.          ],
       [0.08458258, 1.          , 0.06063391, ..., 0.02585438, 0.
,
        0.          ],
       [0.05812382, 0.06063391, 1.          , ..., 0.02665009, 0.
,
        0.          ],
       ...,
       [0.02478408, 0.02585438, 0.02665009, ..., 1.          ,
0.07537784,
        0.04828045],
       [0.02739983, 0.          , 0.          , ..., 0.07537784, 1.
,
        0.05337605],
       [0.          , 0.          , 0.          , ..., 0.04828045,
0.05337605,
        1.          ]])

```

```

cosine_similarity(vectors).shape
(4803, 4803)
similarity = cosine_similarity(vectors)
similarity[2]
array([0.05812382, 0.06063391, 1.          , ..., 0.02665009,
       0.          ,
       0.          ])
similarity[2].shape
(4803,)

```

In the cell below, we enumerate the similarity values for an index, then sort them in descending order based on the similarity value, and then we get the top 6 similar items excluding the first item.

```

sorted(list(enumerate(similarity[2])), reverse= True, key=lambda
x:x[1])[1:7]

[(11, 0.36336104634371585),
 (1343, 0.34521548171187133),
 (29, 0.3217979514674191),
 (4071, 0.28097574347450816),
 (3162, 0.27695585470349865),
 (1717, 0.23717082451262844)]

```

The function below takes a movie title as input and provides recommendations based on similarity

```

def recommend(movie):
    movie_index = new_movies[new_movies["title"]==movie].index[0]
    distances = similarity[movie_index]
    movies_list = sorted(list(enumerate(distances)), reverse = True,
key = lambda x:x[1])[1:7]

    for i in movies_list:
        print(new_movies.iloc[i[0]].title)

# Testing the function
recommend("Avatar")

```

```

Titan A.E.
Independence Day
Aliens vs Predator: Requiem
Small Soldiers
Battle: Los Angeles
Krull

```

#### iv) Hybrid Recommender¶

In this section, we try to build a simple hybrid recommender that brings together techniques we have implemented in the content-based and collaborative filter based engines. This is how it works:

Input: User ID and the Title of a Movie

Output: Similar movies sorted on the basis of expected ratings by that particular user.

```
movies_credits.columns=['movieId', 'title', 'cast', 'crew', 'budget',
'genres', 'homepage',
                        'keywords', 'original_language', 'original_title', 'overview',
                        'popularity', 'production_companies', 'production_countries',
                        'release_date', 'revenue', 'runtime', 'spoken_languages',
'status',
                        'tagline', 'vote_average', 'vote_count', 'director', 'actor',
'soup','userId','rating','timestamp']
```

The function below, named `hybrid_recommendations`, combines movie similarity and user ratings to provide personalized movie recommendations for a given user based on their preferences and the similarity of movies.

*# Function that takes in movie title as input and outputs most similar movies*

```
def hybrid_recommendations(userId, title):

    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim2[idx]))

    # Sort the movies based on the similarity scores
    sim_scores.sort(key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores=sim_scores[1:11]

    # Get the movie indices
    ind=[]
    for (x,y) in sim_scores:
        ind.append(x)

    # Grab the title,movieid,vote_average and vote_count of the top 10 most similar movies
    tit=[]
    movieid=[]
    vote_average=[]
    vote_count=[]
    for x in ind:
```

```

tit.append(movies_credits.iloc[x]['title'])
movieid.append(movies_credits.iloc[x]['movieId'])
vote_average.append(movies_credits.iloc[x]['vote_average'])
vote_count.append(movies_credits.iloc[x]['vote_count'])

# Predict the ratings a user might give to these top 10 most
similar movies
est_rating=[]
for a in movieid:
    est_rating.append(svd.predict(userId, a, r_ui=None).est)

return pd.DataFrame({'index': ind, 'title':tit, 'movieId':movieid,
'vote_average':vote_average,
'vote_count':vote_count,'estimated_rating':est_rating}).set_index('index').sort_values(by='estimated_rating', ascending=False)

# Applying the function
hybrid_recommendations(7,'Evil Dead')

```

	title	movieId	vote_average	\
index				
2477	Jennifer's Body	19994	5.3	
4644	Teeth and Blood	325123	3.0	
2146	The Stepfather	19904	5.4	
2715	Stan Helsing	23988	4.0	
4008	A Haunted House	139038	5.4	
3569	Paranormal Activity: The Marked Ones	227348	5.2	
3882	Feast	10070	6.1	
1627	Deliver Us from Evil	184346	5.9	
1648	Fright Night	58151	6.0	
2125	The Grudge 2	1975	5.2	

	vote_count	estimated_rating
index		
2477	837	3.100297
4644	1	3.100297
2146	167	3.100297
2715	97	3.100297
4008	516	3.100297
3569	449	3.100297
3882	160	3.100297
1627	690	3.100297
1648	603	3.100297
2125	283	2.734695

```

# Applying the function` is a comment in the code indicating that the
function
#`hybrid_recommendations` is being called with the User Id `7` and
`The Tooth Fairy`

```

*# Applying the function*

hybrid\_recommendations(57, 'The Tooth Fairy')

vote_count \ index	title	movieId	vote_average
3683	The Texas Chainsaw Massacre 2	16337	5.9
139			
4282	Friday the 13th Part 2	9725	6.0
315			
4644	Teeth and Blood	325123	3.0
1			
2137	Texas Chainsaw 3D	76617	5.3
465			
3159	The Texas Chain Saw Massacre	30497	7.2
590			
4753	Hayride	193603	5.1
6			
4765	Raymond Did It	228550	3.2
8			
4628	Graduation Day	27420	5.0
22			
2282	Scream	4232	7.0
1476			
1961	Scream 2	4233	6.1
840			

index	estimated_rating
3683	3.273786
4282	3.273786
4644	3.273786
2137	3.273786
3159	3.273786
4753	3.273786
4765	3.273786
4628	3.092864
2282	2.980563
1961	2.284500

*# A different genre of movie, some Science Fiction perhaps?*

hybrid\_recommendations(506, 'Avatar')

index	title	movieId	vote_average	\
466	The Time Machine	2135	5.8	
2327	Predator	106	7.3	
47	Star Trek Into Darkness	54138	7.4	
61	Jupiter Ascending	76757	5.2	
83	The Lovers	79698	4.8	



1201	Predators	34851	6.0
260	Ender's Game	80274	6.6
2372	Megaforce	27380	3.5
71	The Mummy: Tomb of the Dragon Emperor	1735	5.2
2403	Aliens	679	7.7

	vote_count	estimated_rating
index		
466	631	3.443435
2327	2093	3.184114
47	4418	3.123787
61	2768	3.123787
83	34	3.123787
1201	1206	3.123787
260	2303	3.123787
2372	15	3.123787
71	1387	3.122185
2403	3220	2.911398

## Exporting to Create GUI

Below we will save the models above to pickle files for reproducibility in future; as well as for the purposes of deployment

```
import pickle
```

```
#with open('Movie_dict.pkl', 'wb') as f:
#    pickle.dump(movies_credits, f)

# pickle.dump(movies_credits.to_dict(),open('movies.pkl','wb'))

# pickle.dump(similarity,open('.similarity.pkl','wb'))

# with open('hybrid_recommendations.pkl', 'wb') as f:
#    pickle.dump(hybrid_recommendations, f)

#with open('recommend.pkl', 'wb') as f:
#    pickle.dump(recommend, f)
```

## Conclusion

In conclusion, the recommendation system serves as a valuable tool in the movie industry to address the challenge of content navigation and provide personalized movie recommendations. By understanding user preferences, leveraging similarities between users, and utilizing movie features, the system aims to enhance the user experience, increase engagement, and ultimately contribute to user retention on the platform.

## Recommendations

- Real-time Updates: Incorporate a mechanism to continuously update the movie database with the latest releases, ratings, and reviews. This will ensure that the

recommendation system remains up-to-date and can provide users with the most relevant movie suggestions.

- **Contextual Factors:** Consider contextual factors such as time of day, location, mood, and social trends to provide personalized recommendations that align with the user's current situation and preferences.
- **User Feedback and Improvement Loop:** Implement a feedback mechanism that allows users to rate and provide feedback on recommended movies. Utilize this feedback to continuously improve the recommendation algorithms and enhance the accuracy and relevance of future recommendations.