

PYTHON CHO KHOA HỌC DỮ LIỆU

✓ FINAL PROJECT - MOVIE RECOMMENDATION SYSTEM

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

Mounted at /content/drive

Giới thiệu thành viên

MSSV	Họ Tên	Công Việc
22110170	Hồ Minh Quân	Data Preprocessing Data Visualization
22110123	Lê Nguyễn Đức Nam	2 Model: Weighted Rating & Content-based
22110124	Lê Thị Kim Nga	2 Model: Collaborative Filtering & Hybrid
22110155	Trần Nguyễn Thanh Phong	Data Exploration Data Preprocessing
22110158	Trần Châu Phú	Deploy model on website Write report

About dataset

The Movies Dataset

Metadata on over 45,000 movies. 26 million ratings from over 270,000 users.



Link: <https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset>

Bối cảnh

Những tập tin này chứa thông tin về metadata cho tất cả 45,000 bộ phim được liệt kê trong Bộ dữ liệu Full MovieLens. Bộ dữ liệu này bao gồm các bộ phim được phát hành vào hoặc trước tháng 7 năm 2017. Các điểm dữ liệu bao gồm diễn viên, đoàn làm phim, từ khóa cốt truyện, ngân sách, doanh thu, poster, ngày phát hành, ngôn ngữ, công ty sản xuất, quốc gia, số lượng phiếu bầu TMDB và điểm bình chọn trung bình TMDB.

Bộ dữ liệu này cũng chứa các tập tin chứa 26 triệu đánh giá từ 270,000 người dùng cho tất cả 45,000 bộ phim. Đánh giá được đưa ra trên một thang điểm từ 1-5 và đã được thu thập từ trang web chính thức của GroupLens.

Nội dung

Bộ dữ liệu này bao gồm các tập tin sau:

- movies_metadata.csv: Tập chính chứa thông tin về metadata của 45,000 bộ phim trong Bộ dữ liệu Full MovieLens. Các đặc điểm bao gồm poster, hình nền, ngân sách, doanh thu, ngày phát hành, ngôn ngữ,

quốc gia sản xuất và công ty.

- keywords.csv: Chứa các từ khóa cốt truyện cho các bộ phim MovieLens của chúng tôi. Có sẵn dưới dạng một Object JSON được chuỗi hóa.
- credits.csv: Bao gồm thông tin về Diễn viên và Đoàn làm phim cho tất cả các bộ phim của chúng tôi. Có sẵn dưới dạng một Object JSON được chuỗi hóa.
- ratings.csv: Đánh giá của người dùng.

Start coding or [generate](#) with AI.

Start coding or [generate](#) with AI.

✓ Exploratory Data Analysis

✓ Import module

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import seaborn as sns
#import missingno as msno
from collections import Counter
import warnings; warnings.simplefilter('ignore')
```

✓ Load dataset

The Full Dataset: Consists of 26,000,000 ratings and 750,000 tag applications applied to 45,000 movies by 270,000 users. Includes tag genome data with 12 million relevance scores across 1,100 tags.

```
df_movies = pd.read_csv('/content/drive/MyDrive/movies_metadata.csv')
df_credits = pd.read_csv('/content/drive/MyDrive/credits.csv')
df_keywords = pd.read_csv('/content/drive/MyDrive/keywords.csv')
df_ratings = pd.read_csv('/content/drive/MyDrive/ratings.csv')
```

✓ movie_metadata dataframe

```
df_movies.head().T
```



0

1

adult	False		False
belongs_to_collection	{'id': 10194, 'name': 'Toy Story Collection', ...}		{'id': 11905, 'name': 'Jumanji Collection', ...}
budget	30000000		65000000
genres	[{'id': 16, 'name': 'Animation'}, {'id': 35, 'name': 'Comedy'}, {'id': 10751, 'name': 'Family'}]	[{'id': 12, 'name': 'Adventure'}, {'id': 14, 'name': 'Fantasy'}, {'id': 10751, 'name': 'Family'}]	[{'id': 10751, 'name': 'Family'}]
homepage	http://toystory.disney.com/toy-story		NaN
id	862		8844
imdb_id	tt0114709		tt0113497
original_language	en		en
original_title	Toy Story		Jumanji
overview	Led by Woody, Andy's toys live happily in his room until Andy's birthday arrives. The gang of toys must逃离 to save the boy from a vengeful and dangerous collection of toys.	When siblings Judy and Peter discover an enchanted book in the attic, they find themselves transported to a magical world where they must team up with a brave monkey to escape the clutches of a wicked sorcerer.	A family vacation turns into a nightmare when a mysterious storm traps them in a remote location, and a deadly game of hide-and-seek begins.
popularity	21.946943		17.015539
poster_path	/rhIRbceoE9IR4veEXuwCC2wARtG.jpg	/vzmL6fP7aPKNKPRTFnZmiUfcyV.jpg	/6ksm1sjKM0B0UOYUwvU8B0u3B0.jpg
production_companies	[{'name': 'Pixar Animation Studios', 'id': 3}],	[{'name': 'TriStar Pictures', 'id': 559}, {'name': 'Walt Disney Pictures', 'id': 1}],	[{'name': 'Columbia Pictures', 'id': 1}],
production_countries	[{'iso_3166_1': 'US', 'name': 'United States of America'}]	[{'iso_3166_1': 'US', 'name': 'United States of America'}]	[{'iso_3166_1': 'US', 'name': 'United States of America'}]
release_date	1995-10-30		1995-12-15
revenue	373554033.0		262797249.0
runtime	81.0		104.0
spoken_languages	[{'iso_639_1': 'en', 'name': 'English'}]	[{'iso_639_1': 'en', 'name': 'English'}, {'iso_639_1': 'es', 'name': 'Spanish'}]	[{'iso_639_1': 'en', 'name': 'English'}]
status	Released		Released
tagline	NaN	Roll the dice and unleash the excitement!	Still Yelling
title	Toy Story		Jumanji
video	False		False
vote_average	7.7		6.9
vote_count	5415.0		2413.0

```
df_movies.columns
```



```
Index(['adult', 'belongs_to_collection', 'budget', 'genres', 'homepage', 'id',
      'imdb_id', 'original_language', 'original_title', 'overview',
      'popularity', 'poster_path', 'production_companies',
      'production_countries', 'release_date', 'revenue', 'runtime',
      'spoken_languages', 'status', 'tagline', 'title', 'video',
```

```
'vote_average', 'vote_count'],
dtype='object')
```

Feature

- **adult:** Cho biết nếu bộ phim có phân loại X-Rated hoặc dành cho người trưởng thành.
- **belongs_to_collection:** Một từ điển được chuyển thành chuỗi, cung cấp thông tin về loạt phim mà bộ phim cụ thể đó thuộc về.
- **budget:** Ngân sách của bộ phim tính bằng đô la.
- **genres:** Một danh sách được chuyển thành chuỗi, liệt kê tất cả các thể loại liên quan đến bộ phim.
- **homepage:** Trang chính thức của bộ phim.
- **id:** ID của bộ phim.
- **imdb_id:** ID của bộ phim trên trang web IMDb.
- **original_language:** Ngôn ngữ mà bộ phim được quay ban đầu.
- **original_title:** Tiêu đề gốc của bộ phim.
- **overview:** Một đoạn mô tả ngắn về bộ phim.
- **popularity:** Điểm Popularity được gán bởi TMDB.
- **poster_path:** Đường dẫn URL đến hình ảnh poster.
- **production_companies:** Một danh sách được chuyển thành chuỗi, liệt kê các công ty sản xuất tham gia vào việc làm phim.
- **production_countries:** Một danh sách được chuyển thành chuỗi, liệt kê các quốc gia nơi bộ phim được quay/sản xuất.
- **release_date:** Ngày phát hành trong các rạp chiếu phim.
- **revenue:** Tổng doanh thu của bộ phim tính bằng đô la.
- **runtime:** Thời lượng của bộ phim tính bằng phút.
- **spoken_languages:** Một danh sách được chuyển thành chuỗi, liệt kê các ngôn ngữ được sử dụng trong phim.
- **status:** Tình trạng của bộ phim (Released, To Be Released, Announced, v.v.).
- **tagline:** Câu khẩu hiệu của bộ phim.
- **title:** Tiêu đề chính thức của bộ phim.
- **video:** Cho biết liệu có video của bộ phim trên TMDB hay không.
- **vote_average:** Điểm đánh giá trung bình của bộ phim.
- **vote_count:** Số lượng phiếu bầu của người dùng, được đếm bởi TMDB.

```
df_movies.shape
```

```
⇒ (45466, 24)
```

Có tất cả 45466 dòng và 24 cột

```
df_movies.info()
```

```
>>> <class 'pandas.core.frame.DataFrame'>
RangeIndex: 45466 entries, 0 to 45465
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   adult                 45466 non-null  object
 1   belongs_to_collection 4494 non-null   object
 2   budget                45466 non-null  object
 3   genres                45466 non-null  object
 4   homepage              7782 non-null   object
 5   id                    45466 non-null  object
 6   imdb_id               45449 non-null  object
 7   original_language     45455 non-null  object
 8   original_title        45466 non-null  object
 9   overview              44512 non-null  object
10  popularity            45461 non-null  object
11  poster_path           45080 non-null  object
12  production_companies  45463 non-null  object
13  production_countries  45463 non-null  object
14  release_date          45379 non-null  object
15  revenue                45460 non-null  float64
16  runtime                45203 non-null  float64
17  spoken_languages      45460 non-null  object
18  status                 45379 non-null  object
19  tagline                20412 non-null  object
20  title                  45460 non-null  object
21  video                  45460 non-null  object
22  vote_average           45460 non-null  float64
23  vote_count             45460 non-null  float64
dtypes: float64(4), object(20)
memory usage: 8.3+ MB
```

```
def most_common_element_in_categorical_columns(df):
    # Finding the most common element in each categorical columns of DataFrame and return Data
    data = {'Column': [], 'Most Common Element': [], 'Count': []}

    for column in df.select_dtypes(include='object').columns:
        all_values = [value for value in df[column]]
        value_counter = Counter(all_values)
        most_common_value, count = value_counter.most_common(1)[0]
        data['Column'].append(column)
        data['Most Common Element'].append(most_common_value)
        data['Count'].append(count)

    result_df = pd.DataFrame(data).set_index('Column')
    print("Phần tử xuất hiện nhiều nhất trong mỗi cột categorical:")
    return result_df

def plot_missing_values(df):
    # visualizing missing values
    plt.figure(figsize=(30,20))
    plt.subplot(234)
    sns.heatmap(pd.DataFrame(df.isnull().mean() * 100), annot=True, cmap='viridis', linewidths=1,
    plt.title("Missing Value")
```

```
def plot_correlation_heatmap(df):
    # Tính ma trận tương quan
    correlation_matrix = df.corr()

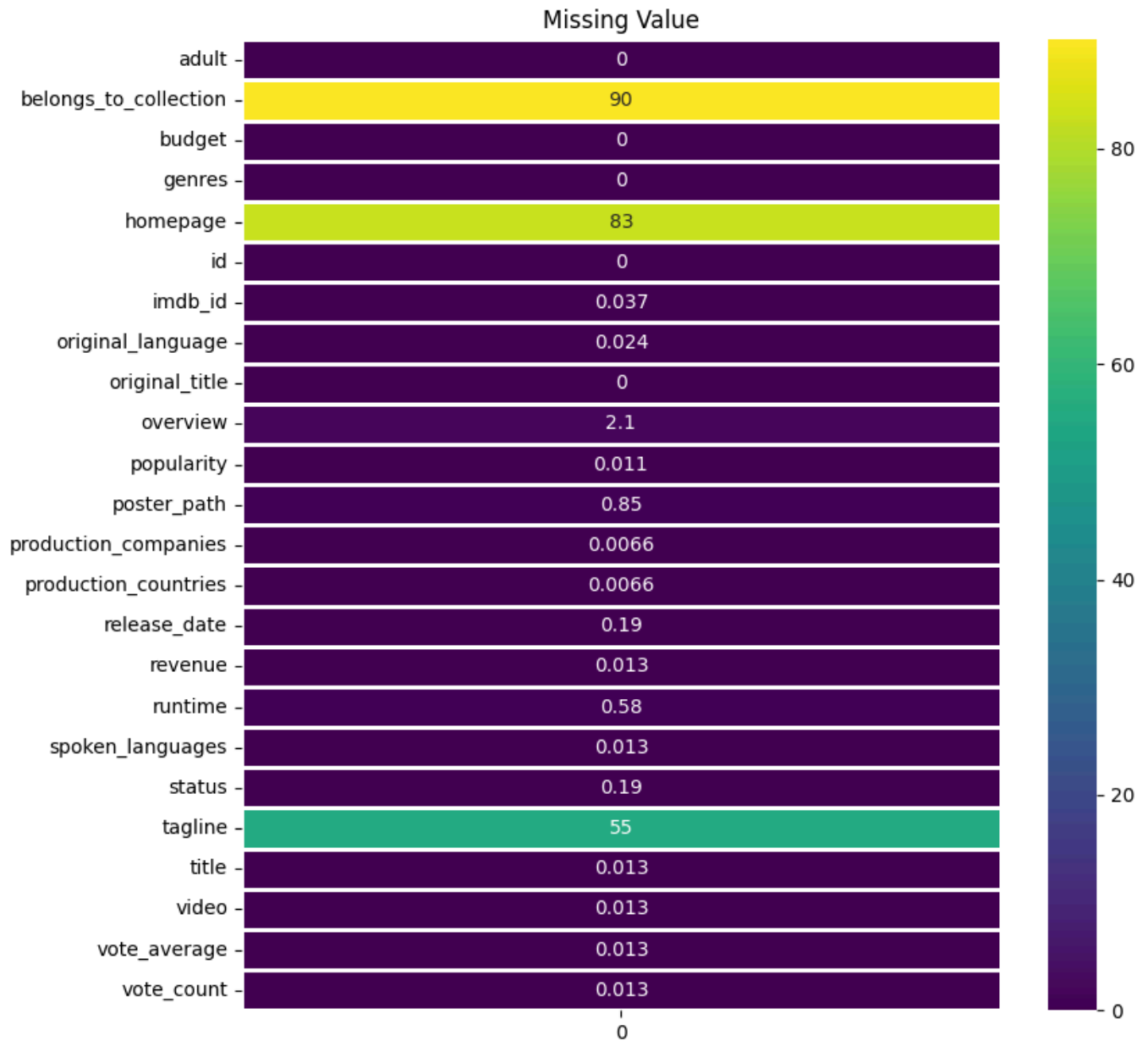
    # Vẽ biểu đồ nhiệt độ (heatmap) của ma trận tương quan
    plt.figure(figsize=(12, 5))
    sns.heatmap(correlation_matrix, annot=True, cmap='viridis', fmt=".2f", linewidths=0.5)
    plt.title('Biểu đồ tương quan')
    plt.show()
```

```
most_common_element_in_categorical_columns(df_movies)
```

⇒ Phần tử xuất hiện nhiều nhất trong mỗi cột categorical:

	Most Common Element	Count
Column		
adult	False	45454
belongs_to_collection	NaN	40972
budget	0	36573
genres	[{'id': 18, 'name': 'Drama'}]	5000
homepage	NaN	37684
id	141971	3
imdb_id	NaN	17
original_language	en	32269
original_title	Alice in Wonderland	8
overview	NaN	954
popularity	0.0	34
poster_path	NaN	386
production_companies	[]	11875
production_countries	[{'iso_3166_1': 'US', 'name': 'United States o...}	17851
release_date	2008-01-01	136
spoken_languages	[{'iso_639_1': 'en', 'name': 'English'}]	22395
status	Released	45014
tagline	NaN	25054
title	Cinderella	11
video	False	45367

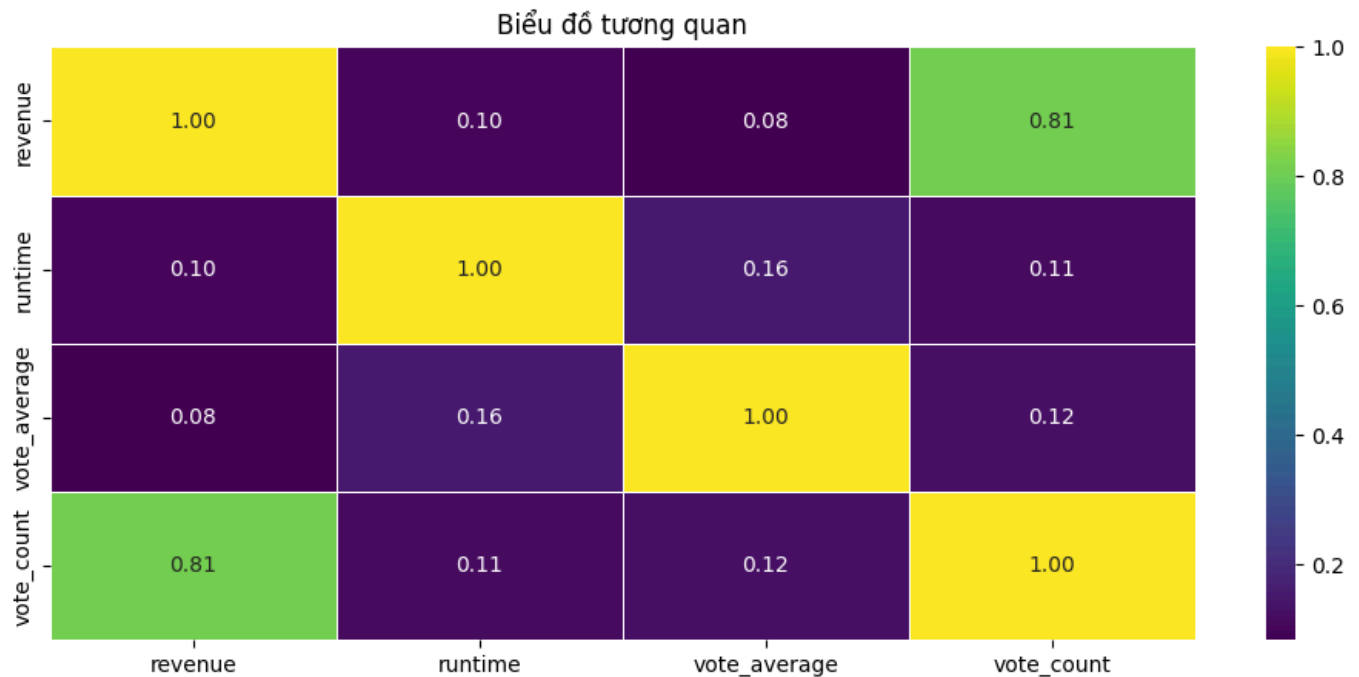
```
plot_missing_values(df_movies)
```



- Có hơn 50% bộ phim không có câu khẩu hiệu "tagline" ban đầu
- Hơn 80% bộ phim không có "homepage" và "belongs_to_collection"
- Một vài bộ phim không có thông tin công ty tham gia sản xuất phim "production_companies". Ở đây ký hiệu [] được dùng để thay thế cho giá trị trống NaN

```
df_columns = df_movies.select_dtypes(include=['float64'])
df_adj_movies = df_columns.copy()
```

```
plot_correlation_heatmap(df_adj_movies)
```



Tương quan giữa 'vote_count' và 'revenue':

- Độ tương quan giữa 'vote count' và 'revenue' là 0.81, điều này cho thấy có một mối tương quan tuyến tính mạnh giữa hai biến này. Giá trị 0.81 gần với 1, điều này chỉ ra rằng khi số lượng phiếu bầu ('vote count') tăng, doanh thu ('revenue') cũng có xu hướng tăng, và ngược lại.
- Mối tương quan mạnh này có thể được hiểu là:
 - Khi một bộ phim nhận được nhiều phiếu bầu, nó có thể thu hút sự chú ý và sự quan tâm từ khán giả, điều này có thể dẫn đến tăng cường doanh thu từ việc mọi người xem bộ phim.
 - Tuy nhiên, cần lưu ý rằng tương quan không có nghĩa là mối quan hệ nhân quả và không giải thích nguyên nhân của mối quan hệ này. Có thể có nhiều yếu tố khác nhau đồng thời ảnh hưởng đến cả 'vote count' và 'revenue'.

Tương quan giữa 'vote_count' và 'vote_average':

- Giá trị 0.12 có thể chỉ ra rằng số lượng phiếu bầu ('vote count') không có tương quan mạnh với điểm đánh giá trung bình ('vote average'). Điều này có thể xảy ra khi một bộ phim có thể thu hút một số lượng lớn phiếu bầu nhưng có thể không nhất thiết có điểm đánh giá cao.
- Có thể có nhiều yếu tố khác nhau đóng vai trò trong quyết định của người xem khi đánh giá một bộ phim, không chỉ là lượng phiếu bầu.

Tương quan giữa 'revenue' và 'vote_average':

- Giá trị 0.08 chỉ ra rằng doanh thu ('revenue') cũng không có tương quan mạnh với điểm đánh giá trung bình ('vote average'). Điều này có thể xảy ra khi một bộ phim có thể kiếm được nhiều doanh thu mà không nhất thiết phải có điểm đánh giá cao từ người xem.

- Một bộ phim có thể có doanh thu lớn do các yếu tố khác như chiến lược tiếp thị, dàn diễn viên nổi tiếng hoặc quảng cáo mạnh mẽ, mà không cần phải có sự đồng thuận cao từ người xem.

✓ credits dataframe

```
df_credits.head()
```

	cast	crew	id
0	[{'cast_id': 14, 'character': 'Woody (voice)',...	[{'credit_id': '52fe4284c3a36847f8024f49', 'de...	862
1	[{'cast_id': 1, 'character': 'Alan Parrish', '...	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'de...	8844
2	[{'cast_id': 2, 'character': 'Max Goldman', 'c...	[{'credit_id': '52fe466a9251416c75077a89', 'de...	15602
3	[{'cast_id': 1, 'character': "Savannah 'Vannah...	[{'credit_id': '52fe44779251416c91011acb', 'de...	31357
4	[{'cast_id': 1, 'character': 'George Banks', '...	[{'credit_id': '52fe44959251416c75039ed7', 'de...	11862

```
df_credits.columns
```

```
Index(['cast', 'crew', 'id'], dtype='object')
```

Feature

- cast: Thông tin về diễn viên. Tên diễn viên, giới tính và tên nhân vật mà họ đóng trong phim.
- crew: Thông tin về các thành viên đoàn làm phim. Chẳng hạn như người đạo diễn, biên tập viên và các vị trí khác.
- id: Đây là ID của bộ phim được cung cấp bởi TMDb (The Movie Database).

```
df_credits.shape
```

```
(45476, 3)
```

```
df_credits.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45476 entries, 0 to 45475
Data columns (total 3 columns):
#   Column  Non-Null Count  Dtype
---  -
0    cast    45476 non-null     object
1    crew    45476 non-null     object
2    id      45476 non-null     int64
dtypes: int64(1), object(2)
memory usage: 1.0+ MB
```

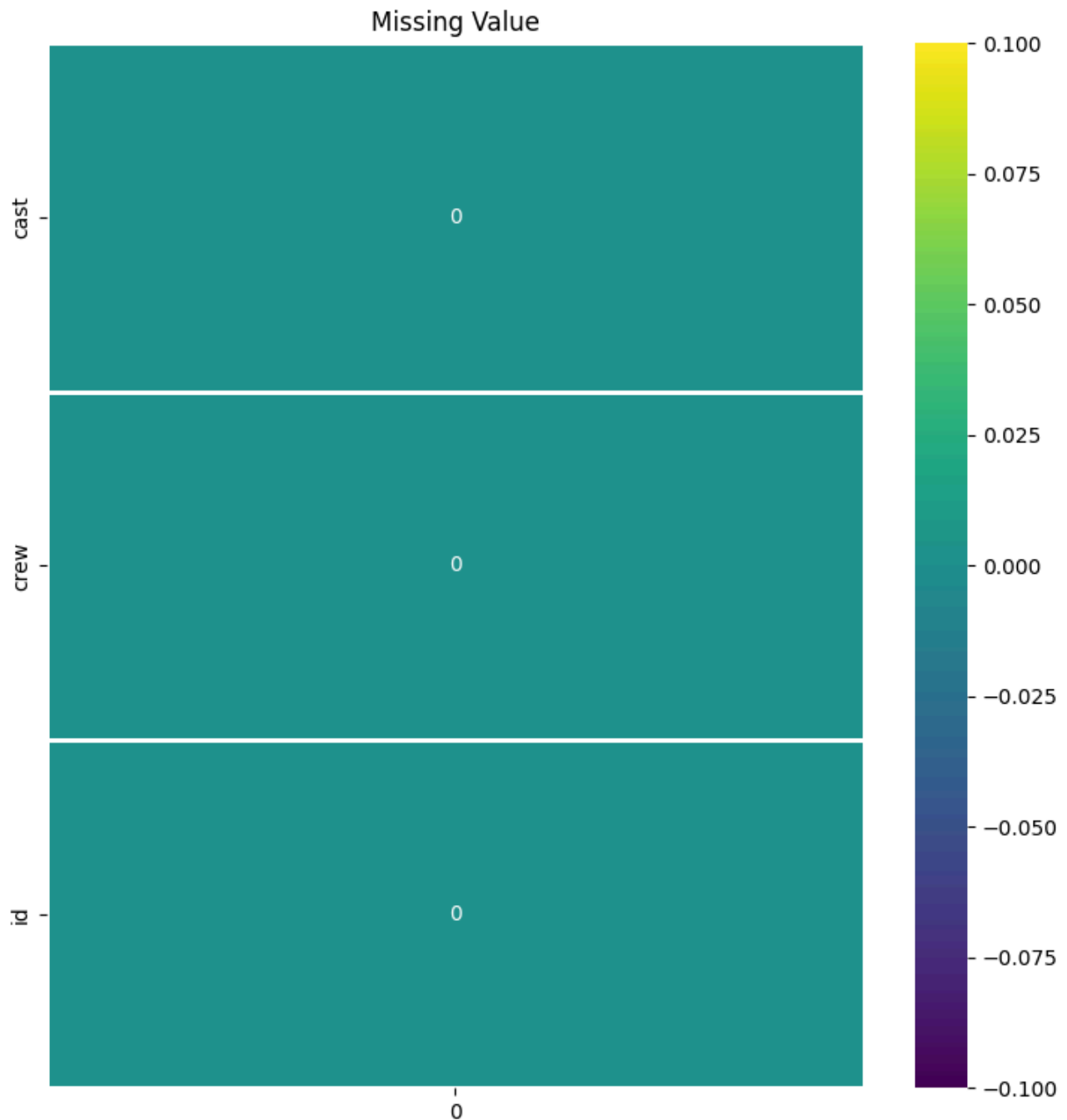
```
most_common_element_in_categorical_columns(df_credits)
```

➞ Phần tử xuất hiện nhiều nhất trong mỗi cột categorical:

Most Common Element Count

Column		
cast	0	2418
crew	0	771

```
plot_missing_values(df_credits)
```



Có nhiều dữ liệu 0 trong cả 2 cột "cast" và "crew". Điều này có thể xảy ra do trong quá trình thu thập dữ liệu, dữ liệu về diễn viên đoàn phim có thể không đầy đủ hoặc không chính xác

✓ keywords dataframe

```
df_keywords.head()
```

```
↗
```

	id	keywords
0	862	{{'id': 931, 'name': 'jealousy'}, {'id': 4290,...
1	8844	{{'id': 10090, 'name': 'board game'}, {'id': 1...
2	15602	{{'id': 1495, 'name': 'fishing'}, {'id': 12392...
3	31357	{{'id': 818, 'name': 'based on novel'}, {'id':...
4	11862	{{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n...

```
df_keywords.columns
```

```
↗ Index(['id', 'keywords'], dtype='object')
```

Feature

- id: Đây là ID của bộ phim được cung cấp bởi TMDb (The Movie Database).
- keywords: Tags/keywords cho bộ phim. Đây là danh sách các từ khóa hoặc thẻ mô tả nội dung hoặc chủ đề của bộ phim.

```
df_keywords.info()
```

```
↗ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 46419 entries, 0 to 46418
Data columns (total 2 columns):
#   Column      Non-Null Count  Dtype
---  -
0    id          46419 non-null  int64
1   keywords    46419 non-null  object
dtypes: int64(1), object(1)
memory usage: 725.4+ KB
```

```
df_keywords.shape
```

```
↗ (46419, 2)
```

```
most_common_element_in_categorical_columns(df_keywords)
```

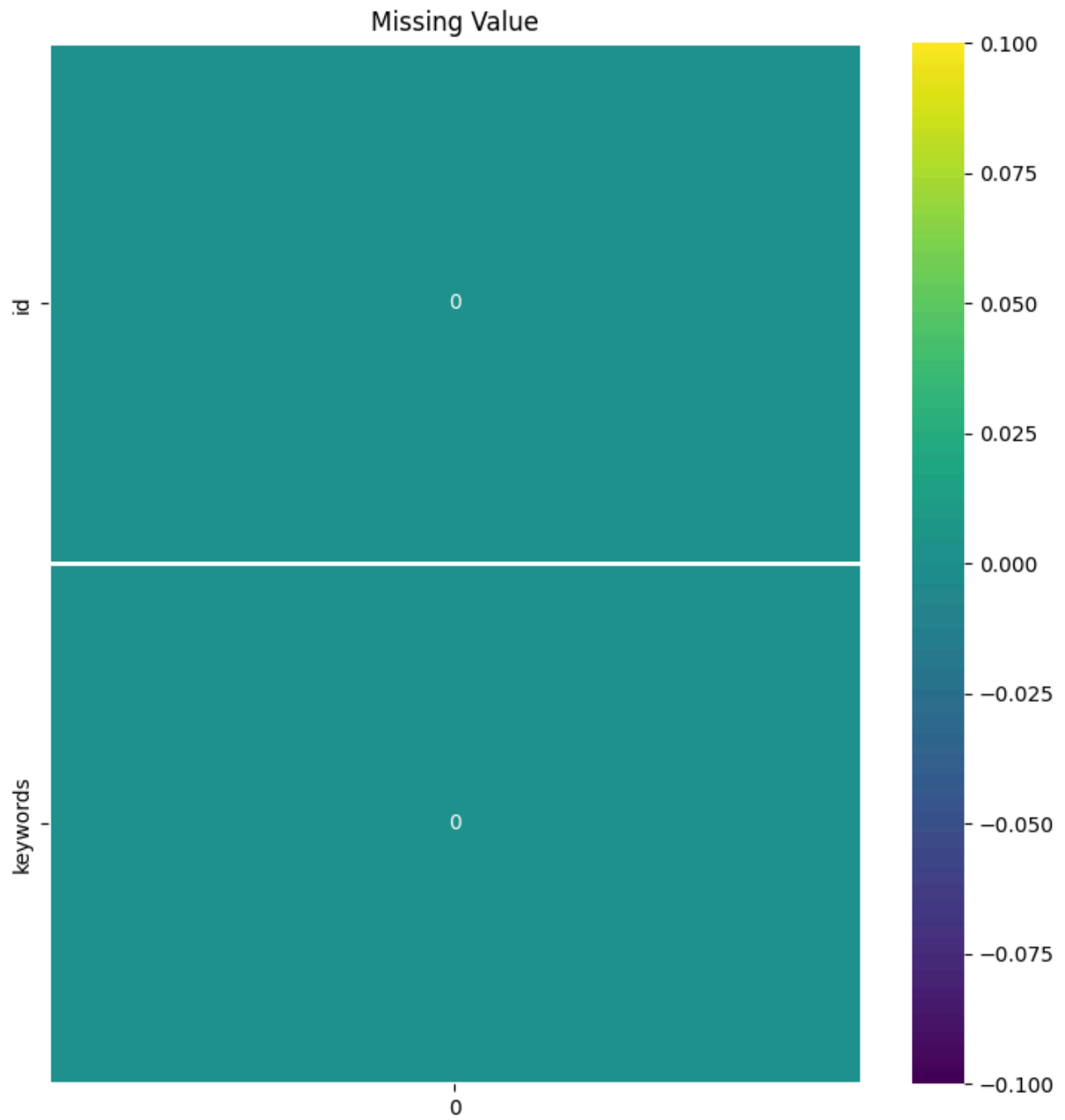
```
↗ Phần tử xuất hiện nhiều nhất trong mỗi cột categorical:
```

Most Common Element Count

Column

Column	Most Common Element	Count
keywords	[]	14795

```
plot_missing_values(df_keywords)
```



▼ ratings dataframe

```
df_ratings.head()
```



	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

```
df_ratings.columns
```



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

Feature

- userId: Đây là ID của người dùng.
- movieId: Đây là ID của bộ phim trên TMDb (The Movie Database).
- rating: Đánh giá được người dùng cụ thể đưa ra cho bộ phim tương ứng.
- timestamp: Thời điểm (timestamp) khi người dùng đưa ra đánh giá cho bộ phim.

```
df_ratings.shape
```



```
(100004, 4)
```

```
df_ratings.info()
```



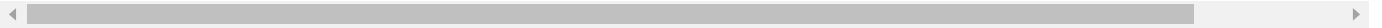
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100004 entries, 0 to 100003
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   userId      100004 non-null  int64
1   movieId     100004 non-null  int64
2   rating      100004 non-null  float64
3   timestamp   100004 non-null  int64
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

```
# In ra 100 'title' được xếp theo thứ tự 'id' tăng dần
sorted_df = df_movies.sort_values(by='id')
selected_columns = ['title', 'vote_average', 'vote_count', 'runtime', 'release_date']
result_df = sorted_df[selected_columns].head(100)
result_df
```



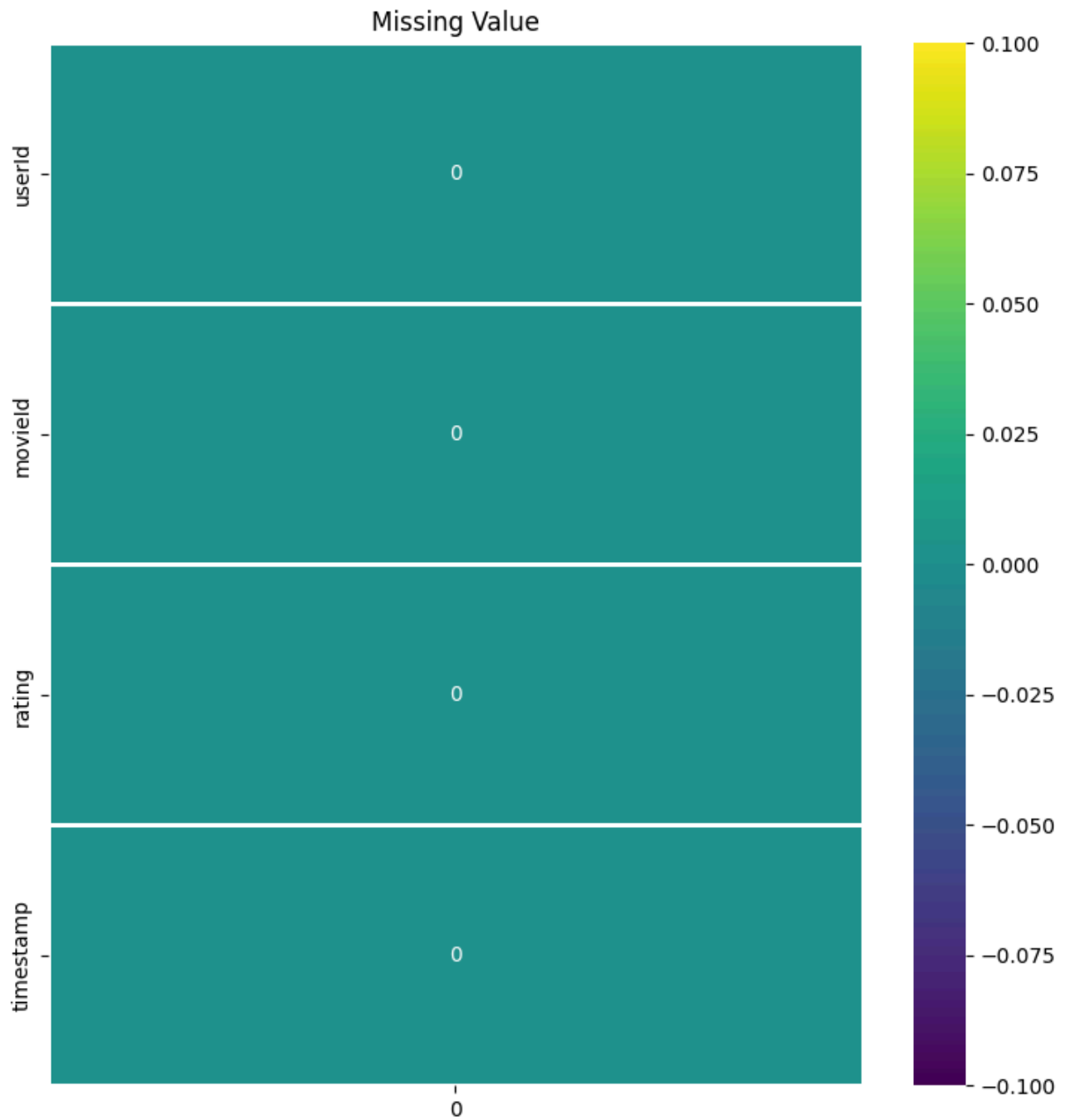
	title	vote_average	vote_count	runtime	release_date
2429	Lock, Stock and Two Smoking Barrels	7.5	1671.0	105.0	1998-03-05
13609	La estrategia del caracol	7.2	9.0	116.0	1993-12-25
4435	Young Einstein	4.5	46.0	91.0	1988-12-15
17451	Flight Command	6.0	1.0	116.0	1940-12-27
36946	Hounded	4.8	7.0	87.0	2006-08-06
...
4042	Get Over It	5.5	76.0	87.0	2001-03-08
11700	Avenue Montaigne	6.2	27.0	106.0	2006-02-15
4996	Dragonfly	6.2	209.0	104.0	2002-02-22
12373	Thunder Rock	7.3	3.0	112.0	1942-12-04
36520	The War Against Mrs. Hadley	0.0	0.0	86.0	1942-08-07

100 rows × 5 columns



Số lượng người xem cho những phim có 'movielfd' từ 0 đến 10000 là một con số lớn, đặc biệt nếu được so sánh với tổng số lượng người xem trong toàn bộ dữ liệu hoặc với những phim có 'movielfd' cao hơn. Điều này có thể chỉ ra rằng nhóm các phim này có sức hấp dẫn lớn đối với khán giả, hoặc có thể được quảng bá mạnh mẽ.

```
plot_missing_values(df_ratings)
```



```
df=df_ratings
```

```
# Chuyển đổi cột 'timestamp' sang kiểu dữ liệu ngày giờ
df['timestamp'] = pd.to_datetime(df['timestamp'], unit='s') # Giả sử 'timestamp' đang ở đơn vị giây
```

```
# Tách cột 'timestamp' thành 'ngày' và 'giờ'
df['day'] = df['timestamp'].dt.date
df['time'] = df['timestamp'].dt.time
df['hour'] = df['timestamp'].dt.hour
```

```
df.head()
```



	userId	movieId	rating	timestamp	day	time	hour
0	1	31	2.5	2009-12-14 02:52:24	2009-12-14	02:52:24	2
1	1	1029	3.0	2009-12-14 02:52:59	2009-12-14	02:52:59	2
2	1	1061	3.0	2009-12-14 02:53:02	2009-12-14	02:53:02	2
3	1	1129	2.0	2009-12-14 02:53:05	2009-12-14	02:53:05	2
4	1	1172	4.0	2009-12-14 02:53:25	2009-12-14	02:53:25	2

```
# Tạo subplot cho cột 'timestamp', 'day', và 'hour'
```

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))
```

```
# Biểu đồ histogram cho cột 'timestamp'
```

```
sns.histplot(df['timestamp'], bins=30, kde=True, color='blue', ax=axes[0])
```

```
axes[0].set_title("Biểu đồ Histogram của cột 'timestamp'")
```

```
axes[0].set_xlabel('Ngày giờ')
```

```
axes[0].set_ylabel('Số lượng')
```

```
# Biểu đồ histogram cho cột 'day'
```

```
sns.histplot(df['day'], bins=30, kde=True, color='green', ax=axes[1])
```

```
axes[1].set_title("Biểu đồ Histogram của cột 'day'")
```

```
axes[1].set_xlabel('Ngày')
```

```
axes[1].set_ylabel('Số lượng')
```

```
# Biểu đồ histogram cho cột 'hour'
```

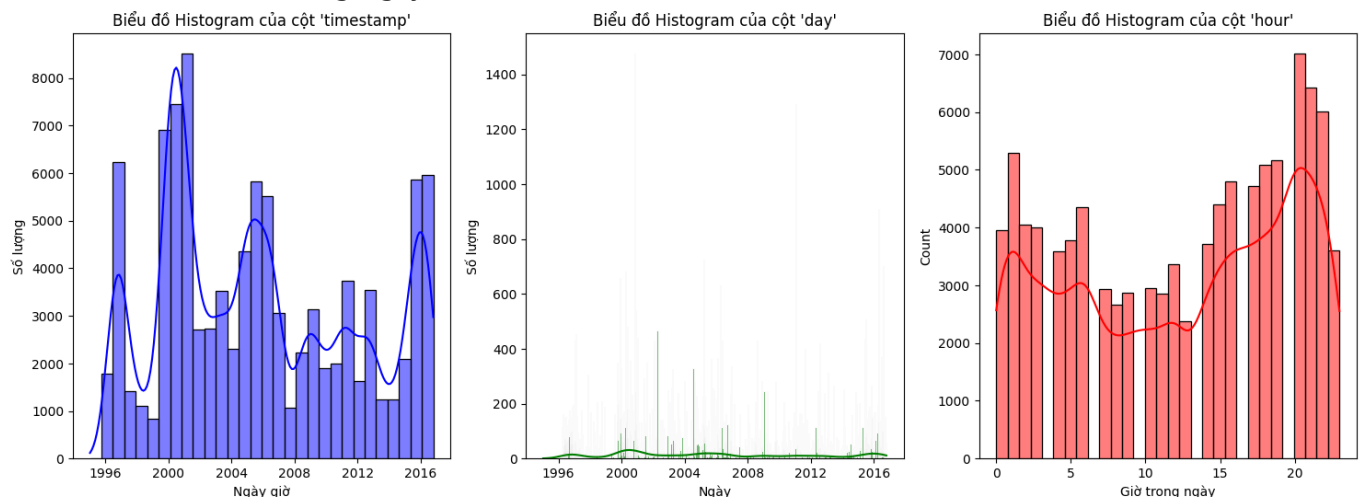
```
sns.histplot(df['hour'], bins=30, kde=True, color='red', ax=axes[2])
```

```
axes[2].set_title("Biểu đồ Histogram của cột 'hour'")
```

```
axes[2].set_xlabel('Giờ trong ngày')
```



```
Text(0.5, 0, 'Giờ trong ngày')
```



Dựa trên phân phối thời gian của ngày giờ xem phim, bạn có thể đưa ra những nhận xét hợp lý về thói quen xem phim của người dùng. Dưới đây là một số nhận xét có thể áp dụng:

Khung giờ xem phim: Nếu có một đỉnh lớn tại các giờ tối, có thể người xem thường xem phim vào buổi tối sau khi kết thúc công việc hoặc các hoạt động hàng ngày. Điều này có thể phản ánh thói quen giải trí sau giờ làm việc.

Mùa trong năm: Nếu có sự tăng đột ngột trong lượng xem phim ở cuối năm hoặc mùa hè, có thể đó là thời điểm mà người xem thường có nhiều thời gian rảnh hơn, chẳng hạn như trong kỳ nghỉ lễ hoặc kỳ nghỉ mùa hè.

Ngày trong tuần: Bạn cũng có thể xem xét thói quen xem phim trong các ngày cụ thể của tuần. Có thể có sự gia tăng vào cuối tuần khi mọi người có nhiều thời gian rảnh hơn so với các ngày trong tuần.

```
# Checking the feature "userID"
```

```
total_users = len(np.unique(df_ratings["userID"]))
print("The count of unique userID in the dataset is : ", total_users)
print("The top 5 userID in the dataset are : \n", df_ratings["userID"].value_counts()[:5])
```

```
➦ The count of unique userID in the dataset is : 671
The top 5 userID in the dataset are :
  userID
547    2391
564    1868
624    1735
15     1700
73     1610
Name: count, dtype: int64
```

Nhận xét:

1. "userID" là những Người dùng được chọn ngẫu nhiên để đưa vào và id của họ đã được ẩn danh.
2. Có hơn 270 nghìn người dùng duy nhất trong tập dữ liệu.
3. userID 45811 có hơn 18 nghìn bản ghi trong tập dữ liệu

```
# Checking the feature "movieID"
```

```
total_movies = len(np.unique(df_ratings["movieId"]))
print("The count of unique movieID in the dataset is : ", total_movies)
print("The top 5 movieID in the dataset are : \n", df_ratings["movieId"].value_counts()[:5])
```

```
➦ The count of unique movieID in the dataset is : 9066
The top 5 movieID in the dataset are :
  movieId
356     341
296     324
318     311
593     304
260     291
Name: count, dtype: int64
```

Nhận xét:

1. "movieId" đại diện cho các phim có ít nhất một loại hoặc thể trong dữ liệu.
2. Có hơn 45 nghìn bộ phim độc đáo trong tập dữ liệu.
3. movieId 356, 318 là một số bộ phim nổi tiếng đã được xếp hạng hơn 90 nghìn lần.

```
# Checking basic statistics for "rating"
```

```
print("The basic statistics for the feature is : \n", df_ratings["rating"].describe())
```

```
→ The basic statistics for the feature is :
count    100004.000000
mean      3.543608
std       1.058064
min       0.500000
25%       3.000000
50%       4.000000
75%       4.000000
max       5.000000
Name: rating, dtype: float64
```

```
# Helper function to Change the numeric label in terms of Millions
```

```
def changingLabels(number):
```

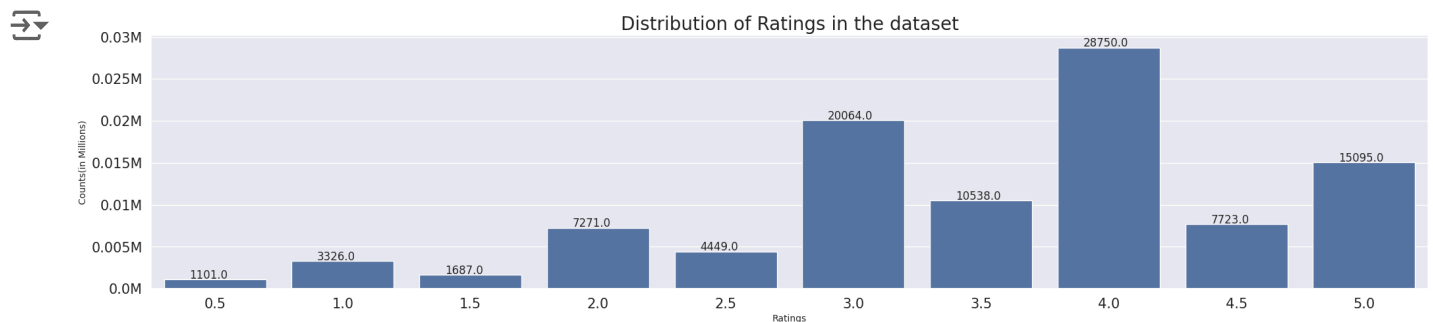
```
    return str(number/10**6) + "M"
```

```
# Visualizing the "rating" for the train set
```

```
sns.set(style="darkgrid")
fig, axes = plt.subplots(1, 1, figsize=(25, 5), sharey=True)

sns.countplot(x="rating", data=df_ratings, ax=axes)
axes.set_yticklabels([changingLabels(num) for num in axes.get_yticks()])
for p in axes.patches:
    axes.annotate('{}' .format(p.get_height()), (p.get_x()+0.2, p.get_height()+100))

plt.tick_params(labelsize = 15)
plt.title("Distribution of Ratings in the dataset", fontsize = 20)
plt.xlabel("Ratings", fontsize = 10)
plt.ylabel("Counts(in Millions)", fontsize = 10)
plt.show()
```



✓ Preprocessing Data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.express as px
import seaborn as sns
from ast import literal_eval
import warnings; warnings.simplefilter('ignore')
pd.set_option('display.max_columns', None)
```

✓ movie_metadata dataframe

```
movies = pd.read_csv('/content/drive/MyDrive/movies_metadata.csv', encoding='utf-8')
movies
```



	adult	belongs_to_collection	budget	genres	homepage	id
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, 'name': 'Family'}]	http://toystory.disney.com/toy-story	862
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, 'name': 'Fantasy'}]	NaN	8844
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, 'name': 'Family'}]	NaN	15602
3	False	NaN	16000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'name': 'Drama'}]	NaN	31357
4	False	{'id': 96871, 'name': 'Father of the Bride Col...	0	[{'id': 35, 'name': 'Comedy'}]	NaN	11862
...
45461	False	NaN	0	[{'id': 18, 'name': 'Drama'}, {'id': 10751, 'name': 'Family'}]	http://www.imdb.com/title/tt6209470/	439050
45462	False	NaN	0	[{'id': 18, 'name': 'Drama'}]	NaN	111109
45463	False	NaN	0	[{'id': 28, 'name': 'Action'}, {'id': 18, 'name': 'Drama'}]	NaN	67758
45464	False	NaN	0	[]	NaN	227506
45465	False	NaN	0	[]	NaN	461257

45466 rows × 24 columns

Year

```

movies['release_date'] = pd.to_datetime(movies['release_date'], errors='coerce')
movies['release_year'] = movies['release_date'].dt.year

movie_counts = movies['release_year'].value_counts().sort_index()

# Tạo biểu đồ
plt.figure(figsize=(12, 6))
sns.barplot(x=movie_counts.index, y=movie_counts.values, palette="viridis")

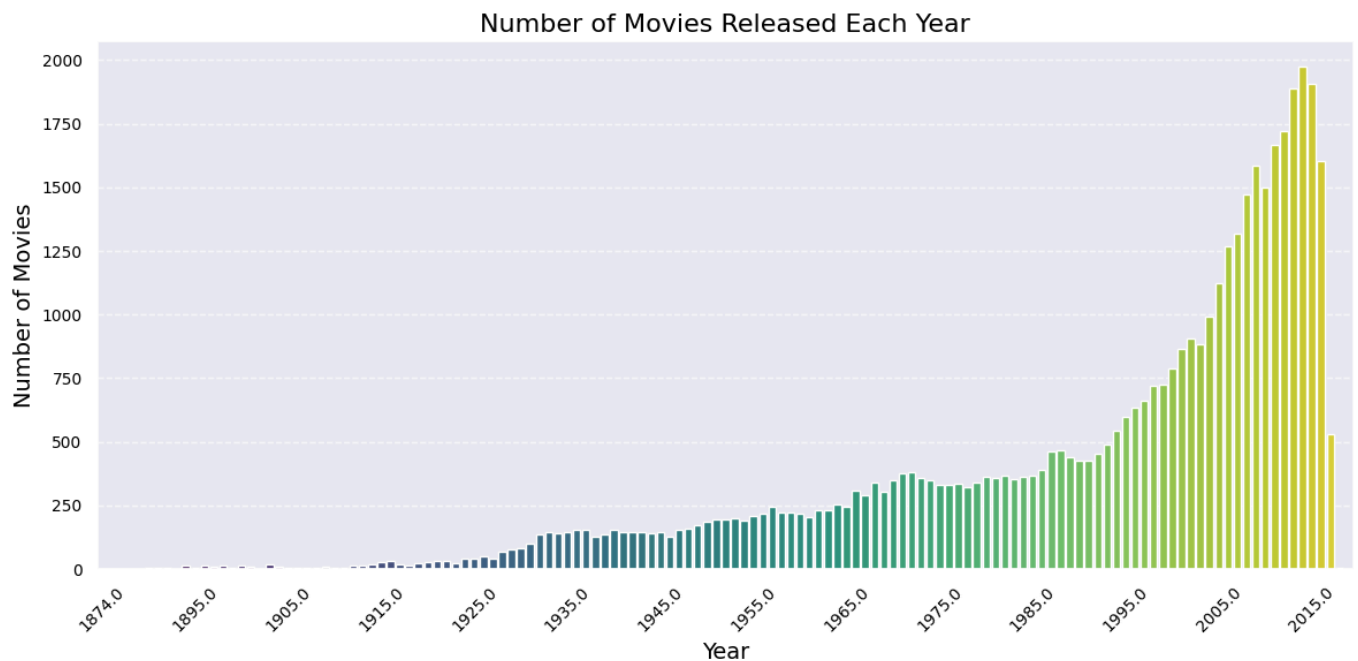
# Cài đặt các thông số biểu đồ
plt.title('Number of Movies Released Each Year', fontsize=16, color='black')
plt.xlabel('Year', fontsize=14, color='black')
plt.ylabel('Number of Movies', fontsize=14, color='black')

# Lấy chỉ số đại diện cho trục X (mỗi 10 năm)
xticks = movie_counts.index[::10] # Lấy mỗi 10 năm
plt.xticks(ticks=range(0, len(movie_counts), 10), labels=xticks, rotation=45, fontsize=10, color='black')

plt.yticks(fontsize=10, color='black')
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()

```



```

movies['release_year'].value_counts().sort_index()

```



release_year	count
1874.0	1
1878.0	1
1883.0	1
1887.0	1
1888.0	2
...	...
2015.0	1905
2016.0	1604
2017.0	532
2018.0	5
2020.0	1

135 rows × 1 columns

dtype: int64

Adult

```
print(*(i for i in movies['adult'].unique()), sep="\n")
```



False

True

- Written by Ørnås

Rune Balot goes to a casino connected to the October corporation to try to wrap up her cas

Avalanche Sharks tells the story of a bikini contest that turns into a horrifying affair v

Ánh xạ chuỗi 'True' hoặc 'False' trong meta['adult'] về thành kiểu boolean.

Nếu meta['adult'] không phải True hoặc False thì drop do dòng đó các giá trị trong dòng đó đảo lộn trật tự

```
movies['adult'] = movies['adult'].map({'True': True, 'False': False})
movies.drop(movies[~movies['adult'].isin([True, False])].index, inplace=True)
```

original_language

```
movies['original_language'].unique()
```



```
array(['en', 'fr', 'zh', 'it', 'fa', 'nl', 'de', 'cn', 'ar', 'es', 'ru',
       'sv', 'ja', 'ko', 'sr', 'bn', 'he', 'pt', 'wo', 'ro', 'hu', 'cy',
       'vi', 'cs', 'da', 'no', 'nb', 'pl', 'el', 'sh', 'xx', 'mk', 'bo',
       'ca', 'fi', 'th', 'sk', 'bs', 'hi', 'tr', 'is', 'ps', 'ab', 'eo',
       'ka', 'mn', 'bm', 'zu', 'uk', 'af', 'la', 'et', 'ku', 'fy', 'lv',
       'ta', 'sl', 'tl', 'ur', 'rw', 'id', 'bg', 'mr', 'lt', 'kk', 'ms',
```

```
'sq', nan, 'qu', 'te', 'am', 'jv', 'tg', 'ml', 'hr', 'lo', 'ay',
'kn', 'eu', 'ne', 'pa', 'ky', 'gl', 'uz', 'sm', 'mt', 'hy', 'iu',
'lb', 'si'], dtype=object)
```

```
(movies['original_language']== 'en').sum()
```

```
↔ 32269
```

Chỉ lựa chọn những bộ phim bằng tiếng Anh

Status & Video

```
movies['status'].unique()
```

```
↔ array(['Released', nan, 'Rumored', 'Post Production', 'In Production',
        'Planned', 'Canceled'], dtype=object)
```

```
print(*(i for i in movies['video'].unique()), sep="\n")
```

```
↔ False
    True
    nan
```

Budget, runtime, popularity, revenue, vote_count, vote_average

với budget <= 0 chuyển thành np.nan làm tương tự với runtime, popularity, revenue, vote_count, vote_average

```
movies['budget'] = movies['budget'].astype(str).apply(lambda x: int(x) if x.isdigit() else np.nan)
movies['popularity'] = pd.to_numeric(movies['popularity'], errors='coerce')
```

```
list_cols=['budget', 'runtime', 'popularity', 'revenue', 'vote_count', 'vote_average']
for i in list_cols:
    movies.loc[movies[i] <= 0, i] = np.nan
```

genres

```
movies['genres_list'] = movies['genres'].fillna('').apply(literal_eval).apply(lambda x: [i['name'] for i in x])
movies['genres_list'] = movies['genres_list'].apply(lambda x: x if x else ['other'])
```


production_companies & production_countries

```
movies['production_countries'] = movies['production_countries'].fillna('').apply(literal_eval)
movies['production_countries'] = movies['production_countries'].apply(lambda x: x if x else ['other'])
```

```
movies['production_companies'] = movies['production_companies'].fillna('').apply(literal_eval)
movies['production_companies'] = movies['production_companies'].apply(lambda x: x if x else ['other'])
```

New data

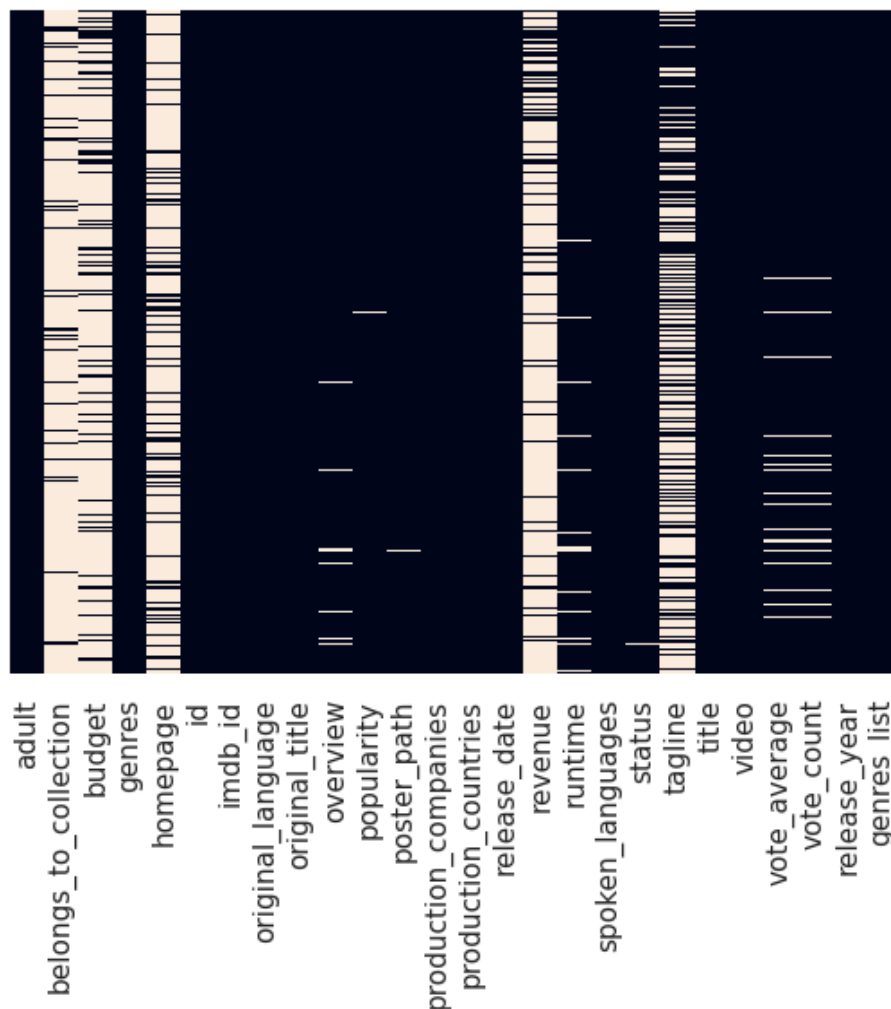
```
movies.isnull().sum()
```



	0
adult	0
belongs_to_collection	40972
budget	36573
genres	0
homepage	37684
id	0
imdb_id	17
original_language	11
original_title	0
overview	954
popularity	69
poster_path	386
production_companies	0
production_countries	0
release_date	87
revenue	38055
runtime	1818
spoken_languages	3
status	84
tagline	25051
title	3
video	3
vote_average	3001
vote_count	2902
release_year	87
genres_list	0

dtype: int64

```
_ = sns.heatmap(movies.isnull(), yticklabels=False, cbar=False)
```

```
new = movies.drop(columns = ['imdb_id', 'adult', 'video', 'status', 'belongs_to_collection', 'b
new.head()
```



	id	popularity	production_companies	production_countries	runtime	title	vote_ave
0	862	21.946943	[Pixar Animation Studios]	[United States of America]	81.0	Toy Story	
1	8844	17.015539	[TriStar Pictures, Teitler Film, Interscope Co...]	[United States of America]	104.0	Jumanji	
2	15602	11.712900	[Warner Bros., Lancaster Gate]	[United States of America]	101.0	Grumpier Old Men	
3	31357	3.859495	[Twentieth Century Fox Film Corporation]	[United States of America]	127.0	Waiting to Exhale	
4	11862	8.387519	[Sandollar Productions, Touchstone Pictures]	[United States of America]	106.0	Father of the Bride Part II	

✓ credits dataframe

```
credits = pd.read_csv('/content/drive/MyDrive/credits.csv', encoding='utf-8')
credits
```



	cast	crew	id
0	[{'cast_id': 14, 'character': 'Woody (voice)', ...	[{'credit_id': '52fe4284c3a36847f8024f49', 'de...	862
1	[{'cast_id': 1, 'character': 'Alan Parrish', '...	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'de...	8844
2	[{'cast_id': 2, 'character': 'Max Goldman', 'c...	[{'credit_id': '52fe466a9251416c75077a89', 'de...	15602
3	[{'cast_id': 1, 'character': 'Savannah Vannah...	[{'credit_id': '52fe44779251416c91011acb', 'de...	31357
4	[{'cast_id': 1, 'character': 'George Banks', '...	[{'credit_id': '52fe44959251416c75039ed7', 'de...	11862
...
45471	[{'cast_id': 0, 'character': '', 'credit_id': ...	[{'credit_id': '5894a97d925141426c00818c', 'de...	439050
45472	[{'cast_id': 1002, 'character': 'Sister Angela...	[{'credit_id': '52fe4af1c3a36847f81e9b15', 'de...	111109
45473	[{'cast_id': 6, 'character': 'Emily Shaw', 'cr...	[{'credit_id': '52fe4776c3a368484e0c8387', 'de...	67758
45474	[{'cast_id': 2, 'character': '', 'credit_id': ...	[{'credit_id': '533bccebc3a36844cf0011a7', 'de...	227506
45475	[]	[{'credit_id': '593e676c92514105b702e68e', 'de...	461257

45476 rows × 3 columns

Actors

```
from ast import literal_eval
credits['actor'] = credits['cast'].fillna('').apply(literal_eval).apply(lambda x: [i['name']
credits['actor'] = credits['actor'].apply(lambda x: x if x else ['unknown'])
```

Director

```
def get_director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
    return ['unknown']

credits['director_name'] = credits['crew'].apply(literal_eval).map(lambda x: get_director(x))
```

New dataframe

```
credits = credits.drop(columns = ['cast', 'crew'])
credits.head()
```



	id	actor	director_name
0	862	[Tom Hanks, Tim Allen, Don Rickles, Jim Varney...	John Lasseter
1	8844	[Robin Williams, Jonathan Hyde, Kirsten Dunst,...	Joe Johnston
2	15602	[Walter Matthau, Jack Lemmon, Ann-Margret, Sop...	Howard Deutch
3	31357	[Whitney Houston, Angela Bassett, Loretta Devi...	Forest Whitaker
4	11862	[Steve Martin, Diane Keaton, Martin Short, Kim...	Charles Shyer

✓ keywords dataframe

```
keywords=pd.read_csv('/content/drive/MyDrive/keywords.csv')
keywords=keywords.drop_duplicates(subset='id', keep='first')
keywords
```



	id	keywords
0	862	[{'id': 931, 'name': 'jealousy'}, {'id': 4290,...
1	8844	[{'id': 10090, 'name': 'board game'}, {'id': 1...
2	15602	[{'id': 1495, 'name': 'fishing'}, {'id': 12392...
3	31357	[{'id': 818, 'name': 'based on novel'}, {'id':...
4	11862	[{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n...
...
46414	439050	[{'id': 10703, 'name': 'tragic love'}]
46415	111109	[{'id': 2679, 'name': 'artist'}, {'id': 14531,...
46416	67758	[]
46417	227506	[]
46418	461257	[]

45432 rows × 2 columns

```
keywords['keywords'] = keywords['keywords'].fillna('').apply(literal_eval).apply(lambda x: [i|
keywords['keywords'] = keywords['keywords'].apply(lambda x: x if x else ['other'])
keywords
```



	id	keywords
0	862	[jealousy, toy, boy, friendship, friends, riva...
1	8844	[board game, disappearance, based on children'...
2	15602	[fishing, best friend, duringcreditsstinger, o...
3	31357	[based on novel, interracial relationship, sin...
4	11862	[baby, midlife crisis, confidence, aging, daug...
...
46414	439050	[tragic love]
46415	111109	[artist, play, pinoy]
46416	67758	[other]
46417	227506	[other]
46418	461257	[other]

45432 rows × 2 columns

✓ Full data

```
new['id'] = movies['id'].astype(int)
merge_movies = pd.merge(new, credits, on='id', how='inner')
merge_movies = pd.merge(merge_movies, keywords, on='id', how='inner')
merge_movies.drop_duplicates(subset='id', keep='first', inplace=True)
merge_movies
```



	id	popularity	production_companies	production_countries	runtime	title
0	862	21.946943	[Pixar Animation Studios]	[United States of America]	81.0	Toy Story
1	8844	17.015539	[TriStar Pictures, Teitler Film, Interscope Co...]	[United States of America]	104.0	Jumanji
2	15602	11.712900	[Warner Bros., Lancaster Gate]	[United States of America]	101.0	Grumpier Old Men
3	31357	3.859495	[Twentieth Century Fox Film Corporation]	[United States of America]	127.0	Waiting to Exhale
4	11862	8.387519	[Sandollar Productions, Touchstone Pictures]	[United States of America]	106.0	Father of the Bride Part II
...
45533	439050	0.072051	[other]	[Iran]	90.0	Subdue
45534	111109	0.178241	[Sine Olivia]	[Philippines]	360.0	Century of Birthing
45535	67758	0.903007	[American World Pictures]	[United States of America]	90.0	Betrayal
45536	227506	0.003503	[Yermoliev]	[Russia]	87.0	Satan Triumphant
45537	461257	0.163015	[other]	[United Kingdom]	75.0	Queerama

45432 rows × 13 columns

```
merge_movies.isnull().sum()
```



	0
<hr/>	
id	0
popularity	69
production_companies	0
production_countries	0
runtime	1817
title	3
vote_average	2997
vote_count	2898
release_year	87
genres_list	0
actor	0
director_name	0
keywords	0

dtype: int64

```
merge_movies = merge_movies.dropna()  
merge_movies
```



	id	popularity	production_companies	production_countries	runtime	title
0	862	21.946943	[Pixar Animation Studios]	[United States of America]	81.0	Toy Story
1	8844	17.015539	[TriStar Pictures, Teitler Film, Interscope Co...]	[United States of America]	104.0	Jumanji
2	15602	11.712900	[Warner Bros., Lancaster Gate]	[United States of America]	101.0	Grumpier Old Men
3	31357	3.859495	[Twentieth Century Fox Film Corporation]	[United States of America]	127.0	Waiting to Exhale
4	11862	8.387519	[Sandollar Productions, Touchstone Pictures]	[United States of America]	106.0	Father of the Bride Part II
...
45530	289923	0.386450	[Neptune Salad Entertainment, Pirie Productions]	[United States of America]	30.0	The Burkittsville 7
45531	222848	0.661558	[Concorde-New Horizons]	[United States of America]	85.0	Caged Heat 3000
45532	30840	5.683753	[Westdeutscher Rundfunk (WDR), Working Title F...]	[Canada, Germany, United Kingdom, United State...]	104.0	Robin Hood
45534	111109	0.178241	[Sine Olivia]	[Philippines]	360.0	Century of Birthing

45535	67758	0.903007	[American World Pictures]	[United States of America]	90.0	Betrayal
-------	-------	----------	---------------------------	----------------------------	------	----------

41056 rows × 13 columns

```
merge_movies.to_csv('/content/drive/MyDrive/data.csv', index=False)
```

✓ Visualize

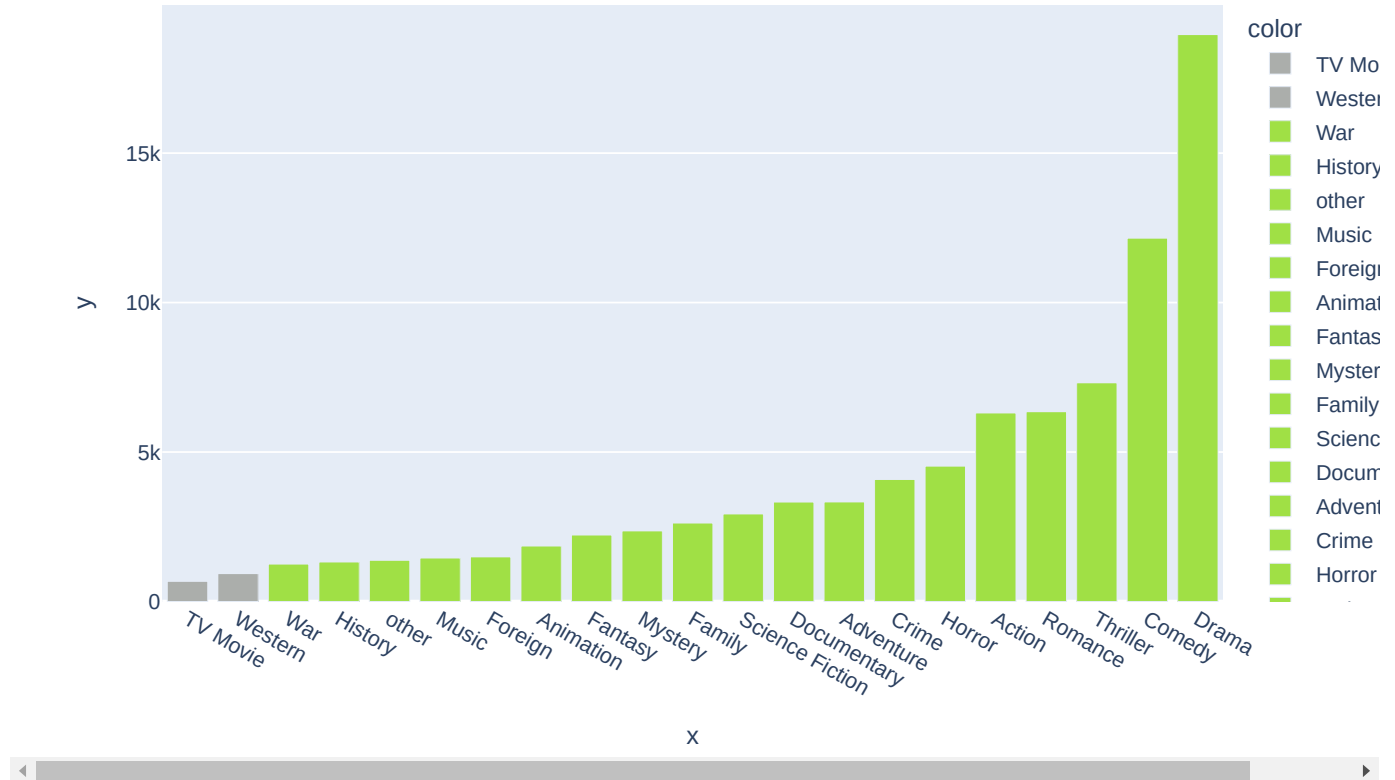
```
def get_counts(data, col):
    counts = {}
    for _, row in data.iterrows():
        if isinstance(row[col], list):
            genres = row[col]
            for genre in genres:
                if genre in counts:
                    counts[genre] += 1
                else:
                    counts[genre] = 1
    return counts

# Get the base counts of each category and sort them by counts
base_counts = get_counts(merge_movies, 'genres_list')
base_counts = pd.DataFrame(index=base_counts.keys(), data=base_counts.values(), columns=['Counts'])
base_counts.sort_values(by='Counts', inplace=True)

# Plot the chart which shows top genres and separate by color where genre<1000
colors = ['#abaeab' if count < 1000 else '#A0E045' for count in base_counts['Counts']]
fig = px.bar(x=base_counts.index, y=base_counts['Counts'], title='Most Popular Genre', color_discrete_map=colors)
fig.show()
```




Most Popular Genre



Nhận xét:

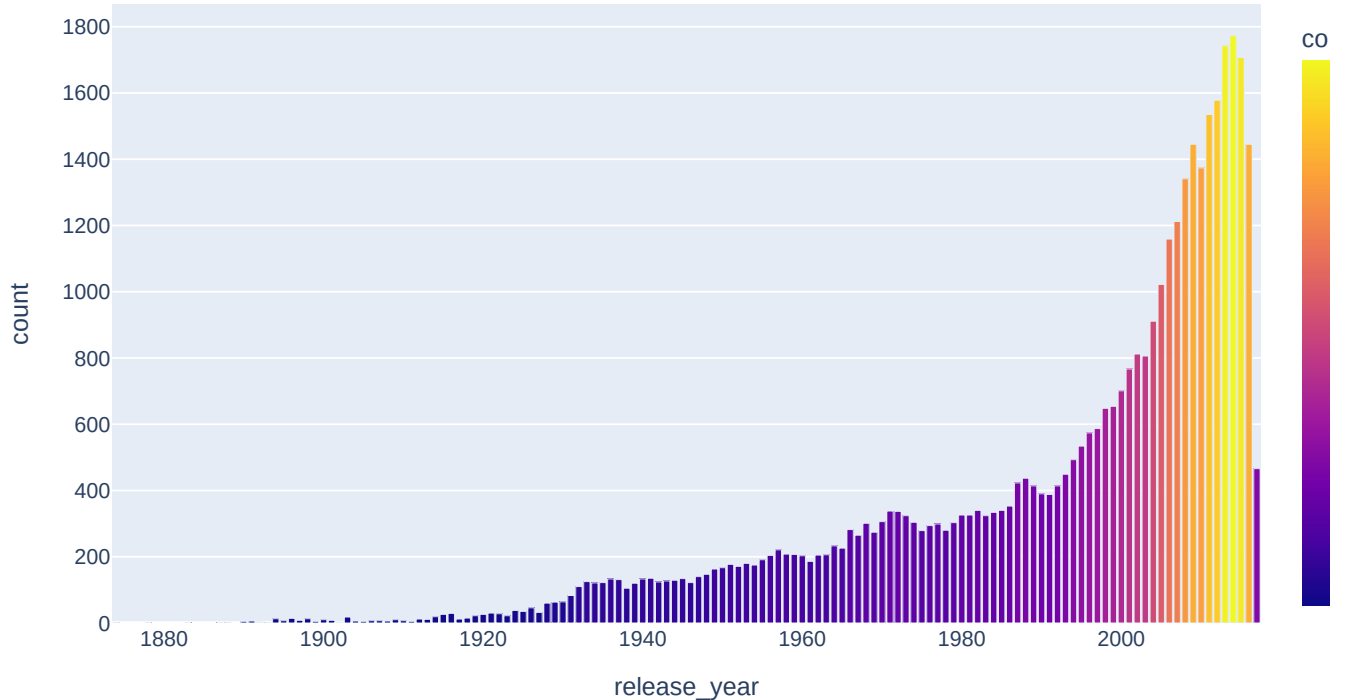
1. Có 21 thể loại phim khác nhau trong khi có một số ít thể loại chưa được đề cập đến.
2. Drama, Comedy, Thriller, Romance, Action là 5 thể loại phim được ưa chuộng nhất trong tập dữ liệu.

```
def plot_value_counts_bar(data, col):
    vc = data[col].value_counts().reset_index()
    fig = px.bar(vc, y='count', x=col, color='count', title=col)
    return fig
```

```
plot_value_counts_bar(merge_movies, 'release_year')
```



release_year



```
import math
def get_ratings(data, col, ratings_col):
    base_counts = get_counts(data, col)
    category_ratings = {}
    a = {}

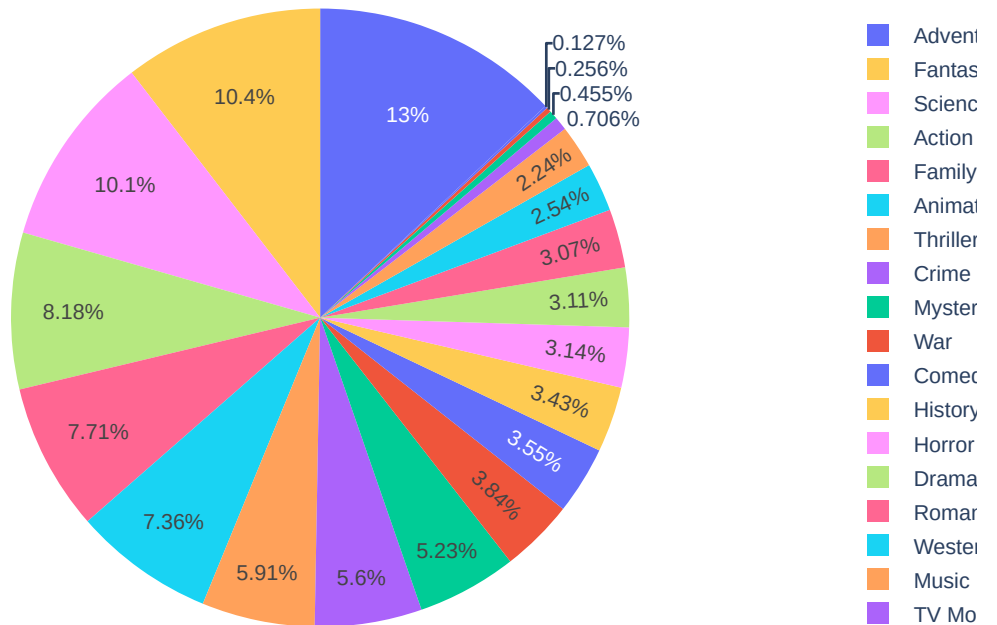
    for _, row in data.iterrows():
        if isinstance(row[col], list):
            genres = row[col]
            for genre in genres:
                if not math.isnan(row['vote_count']):
                    if genre in category_ratings:
                        category_ratings[genre] += row[ratings_col]
                    else:
                        category_ratings[genre] = row[ratings_col]
    for genre in base_counts:
        a[genre] = round(category_ratings[genre] / base_counts[genre], 2)
    return a
```

```
base_counts = get_ratings(merge_movies, 'genres_list', 'vote_count')
base_counts = pd.DataFrame(index=base_counts.keys(),
                            data=base_counts.values(),
                            columns=['Counts'])
base_counts.sort_values(by='Counts', inplace=True)
fig = px.pie(names=base_counts.index,
              values=base_counts['Counts'],
              title='Most Popular Genre by Votes',
```

```
color=base_counts.index)
fig.show()
```



Most Popular Genre by Votes



✓ Building Model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from ast import literal_eval
import builtins

import json
from itertools import islice
!pip install lightfm

from sklearn import preprocessing
from lightfm.evaluation import auc_score, precision_at_k
from lightfm import LightFM
from lightfm.data import Dataset
from lightfm import cross_validation
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
```



Collecting lightfm

Downloading lightfm-1.17.tar.gz (316 kB)

316.4/316.4 kB 5.7 MB/s eta 0:00:00

Preparing metadata (setup.py) ... done

Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from lightfm)

Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.10/dist-packages (from lightfm)

Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from lightfm)

Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from lightfm)

Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from lightfm)

Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from lightfm)

Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from lightfm)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from lightfm)

Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from lightfm)

Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from lightfm)

Building wheels for collected packages: lightfm

Building wheel for lightfm (setup.py) ... done

Created wheel for lightfm: filename=lightfm-1.17-cp310-cp310-linux_x86_64.whl size=806107

Stored in directory: /root/.cache/pip/wheels/4f/9b/7e/0b256f2168511d8fa4dae4fae0200fdbc7

Successfully built lightfm

Installing collected packages: lightfm

Successfully installed lightfm-1.17

✓ Model 1: Simple Recommender

```
df_movie_features = pd.read_csv('/content/drive/MyDrive/data.csv', encoding='utf_8')
df_movie_features.head()
```



	id	popularity	production_companies	production_countries	runtime	title	vote_ave
0	862	21.946943	['Pixar Animation Studios']	['United States of America']	81.0	Toy Story	
1	8844	17.015539	['TriStar Pictures', 'Teitler Film', 'Intersco...']	['United States of America']	104.0	Jumanji	
2	15602	11.712900	['Warner Bros.', 'Lancaster Gate']	['United States of America']	101.0	Grumpier Old Men	
3	31357	3.859495	['Twentieth Century Fox Film Corporation']	['United States of America']	127.0	Waiting to Exhale	
4	11862	8.387519	['Sandollar Productions', 'Touchstone Pictures']	['United States of America']	106.0	Father of the Bride Part II	

```
m = df_movie_features['vote_count'].quantile(0.95)
C = df_movie_features['vote_average'].mean()
```

```
def weighted_rating(x):
    v = x['vote_count']
    R = x['vote_average']
    return (v/(v+m) * R) + (m/(m+v) * C)
```

```
df_movie_features['genres_list'] = df_movie_features['genres_list'].apply(eval)
s = df_movie_features.apply(lambda x: pd.Series(x['genres_list']),axis=1).stack().reset_index()
s.name = 'genre'
gen_md = df_movie_features.drop('genres_list', axis=1).join(s)
```

```
def build_chart(genre, percentile=0.85):
    df = gen_md[gen_md['genre'] == genre]
    vote_counts = df[df['vote_count'].notnull()]['vote_count'].astype('int')
    vote_averages = df[df['vote_average'].notnull()]['vote_average'].astype('int')
    C = vote_averages.mean()
    m = vote_counts.quantile(percentile)
```

```

qualified = df[(df['vote_count'] >= m) & (df['vote_count'].notnull()) & (df['vote_average']
qualified['vote_count'] = qualified['vote_count'].astype('int')
qualified['vote_average'] = qualified['vote_average'].astype('int')

# qualified['wr'] = qualified.apply(lambda x: (x['vote_count']/(x['vote_count']+m) * x['vo
qualified['wr'] = qualified.apply(weighted_rating, axis=1)
qualified = qualified.sort_values('wr', ascending=False).head(250)

return qualified

```

```
build_chart('Animation')
```



	title	release_year	vote_count	vote_average	popularity	wr
350	The Lion King	1994.0	5520	8	21.605761	7.835769
5295	Spirited Away	2001.0	3968	8	41.048867	7.778722
9426	Howl's Moving Castle	2004.0	2049	8	16.136048	7.612135
2760	Princess Mononoke	1997.0	2041	8	17.166725	7.610914
5634	My Neighbor Totoro	1988.0	1730	8	13.507299	7.556651
...
30268	Arthur 3: The War of the Two Worlds	2010.0	371	5	11.371222	5.587063
10850	The Ant Bully	2006.0	375	5	7.000272	5.584380
17608	Happy Feet Two	2011.0	381	5	9.141045	5.580400
18680	Arthur and the Revenge of Maltazard	2009.0	392	5	9.603484	5.573242
6994	Home on the Range	2004.0	405	5	8.555544	5.565008

Start coding or [generate](#) with AI.

✓ Model 2: Content Based Recommender

```

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def create_soup(x):
    # Check if columns exist before accessing them
    keywords = x['keywords'] if 'keywords' in x else ''
    actor = x['actor'] if 'actor' in x else ''
    director_name = x['director_name'] if 'director_name' in x else ''
    genres_list = x['genres_list'] if 'genres_list' in x else ''
    title_x = x['title'] if 'title' in x else ''
    # Return the combined string
    return ' '.join(keywords) + ' ' + ' '.join(actor) + ' ' + director_name + ' ' + ' '.join(g

```

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



	id	popularity	production_companies	production_countries	runtime	title	vote_average
0	862	21.946943	['Pixar Animation Studios']	['United States of America']	81.0	Toy Story	
1	8844	17.015539	['TriStar Pictures', 'Teitler Film', 'Intersco...']	['United States of America']	104.0	Jumanji	
2	15602	11.712900	['Warner Bros.', 'Lancaster Gate']	['United States of America']	101.0	Grumpier Old Men	
3	31357	3.859495	['Twentieth Century Fox Film Corporation']	['United States of America']	127.0	Waiting to Exhale	
4	11862	8.387519	['Sandollar Productions', 'Touchstone Pictures']	['United States of America']	106.0	Father of the Bride Part II	

```
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df_movie_features['soup'])

cosine_sim = cosine_similarity(count_matrix, count_matrix)

df_movie_features = df_movie_features.reset_index()
indices = pd.Series(df_movie_features.index, index=df_movie_features['title'])
```

```
def get_recommendations(titles, cosine_sim=cosine_sim):
    id = indices.get(titles, None)
    if (id is None):
        a=df_movie_features.query('title.str.contains(@titles)').sort_values(by=['vote_average'])
        id = indices[a['title'][0]]
    if (id.shape != ()):
        id = id[0]

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

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, 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
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return df_movie_features[['id', 'title', 'director_name', 'actor', 'genres_list', 'keyword
```

```
get_recommendations('Harry Potter', cosine_sim)
```



	id	title	director_name	actor	genres_list	keywords
15652	12444	Harry Potter and the Deathly Hallows: Part 1	David Yates	['Daniel Radcliffe', 'Emma Watson', 'Rupert Gr...	[Adventure, Fantasy, Family]	['corruption', 'isolation', 'radio', 'magic', ...
16875	12445	Harry Potter and the Deathly Hallows: Part 2	David Yates	['Daniel Radcliffe', 'Rupert Grint', 'Emma Wat...	[Family, Fantasy, Adventure]	['self sacrifice', 'magic', 'frog', 'sorcerer'...
28413	259316	Fantastic Beasts and Where to Find Them	David Yates	['Eddie Redmayne', 'Colin Farrell', 'Katherine...	[Adventure, Family, Fantasy]	['robbery', 'magic', 'teleportation', 'suitcas...
11629	675	Harry Potter and the Order of the Phoenix	David Yates	['Daniel Radcliffe', 'Rupert Grint', 'Emma Wat...	[Adventure, Fantasy, Family, Mystery]	['prophecy', 'witch', 'loss of lover', 'magic'...
35950	294272	Pete's Dragon	David Lowery	['Bryce Dallas Howard', 'Oakes Fegley', 'Wes B...	[Adventure, Family, Fantasy]	['feral child', 'remake', 'dragon', 'orphan', ...
11788	2274	The Seeker: The Dark Is Rising	David L. Cunningham	['Ian McShane', 'Christopher Eccleston', 'Greg...	[Adventure, Drama, Fantasy, Family, Thriller]	['fight', 'dynasty', 'chosen one', 'earth', 'i...
21813	18224	Bionicle 3: Web of Shadows	David Molina	['Kathleen Barr', 'Trevor Devall', 'Brian Drum...	[Action, Adventure, Animation, Family, Fantasy]	['return', 'hero', 'enemy']

```
get_recommendations('Avatar', cosine_sim)
```




	id	title	director_name	actor	genres_list	keywords
1037	2756	The Abyss	James Cameron	['Ed Harris', 'Mary Elizabeth Mastrantonio', '...	[Adventure, Action, Thriller, Science Fiction]	['ocean', 'sea', 'diving suit', 'flying saucer...]
20374	76170	The Wolverine	James Mangold	['Hugh Jackman', 'Hiroyuki Sanada', 'Famke Jan...]	[Action, Science Fiction, Adventure, Fantasy]	['japan', 'samurai', 'mutant', 'world war i', ...]
13225	14164	Dragonball Evolution	James Wong	['Chow Yun-fat', 'Justin Chatwin', 'Joon Park'...]	[Action, Adventure, Fantasy, Science Fiction, ...]	['karate', 'superhero', 'revenge', 'dragon', '...]
565	280	Terminator 2: Judgment Day	James Cameron	['Arnold Schwarzenegger', 'Linda Hamilton', 'R...]	[Action, Thriller, Science Fiction]	['cyborg', 'shotgun', 'post-apocalyptic', 'dys...]
1133	218	The Terminator	James Cameron	['Arnold Schwarzenegger', 'Michael Biehn', 'Li...]	[Action, Thriller, Science Fiction]	['saving the world', 'artificial intelligence'...]
9383	22559	Aliens of the Deep	James Cameron	['Anatoly M. Sagalevitch', 'Pamela Conrad', 'J...]	[Action, Documentary, Science Fiction]	['deep sea']

✓ Model 3 & 4: LightFm model

```

df_rating = pd.read_csv('/content/drive/MyDrive/ratings.csv')
df_rating.head()

```



	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

```

df_rating.drop(columns = 'timestamp', inplace=True)
df_rating.columns = ['user_id', 'movie_id', 'rating']
df_rating.head()

```



	user_id	movie_id	rating
0	1	31	2.5
1	1	1029	3.0
2	1	1061	3.0
3	1	1129	2.0
4	1	1172	4.0



```
df_rating.user_id.unique(), df_rating.movie_id.unique()
```



```
(671, 9066)
```

✓ Prepare movie features

1. Apply the same encoder that we used to split train/test data
2. Columns refer to the column names of the item features (product_id excluded)
3. To prepare the item_features, need to use the Dataset class in LightFM API.
4. First fit the dataset instance and then call function build_item_features to generate the item features for modeling.

```
df_movie_features = pd.read_csv('/content/drive/MyDrive/data.csv', encoding='utf_8')
df_movie_features.rename(columns = {'id':'movie_id'}, inplace=True)
df_movie_features
```



	movie_id	popularity	production_companies	production_countries	runtime	title
0	862	21.946943	['Pixar Animation Studios']	['United States of America']	81.0	Toy Story
1	8844	17.015539	['TriStar Pictures', 'Teitler Film', 'Intersco...']	['United States of America']	104.0	Jumanj
2	15602	11.712900	['Warner Bros.', 'Lancaster Gate']	['United States of America']	101.0	Grumpier Old Mer
3	31357	3.859495	['Twentieth Century Fox Film Corporation']	['United States of America']	127.0	Waiting to Exhale
4	11862	8.387519	['Sandollar Productions', 'Touchstone Pictures']	['United States of America']	106.0	Father of the Bride Part I
...
41051	289923	0.386450	['Neptune Salad Entertainment', 'Pirie Product...']	['United States of America']	30.0	The Burkittsville 7
41052	222848	0.661558	['Concorde-New Horizons']	['United States of America']	85.0	Caged Heat 300C
41053	30840	5.683753	['Westdeutscher Rundfunk (WDR)', 'Working Titl...']	['Canada', 'Germany', 'United Kingdom', 'Unite...']	104.0	Robin Hood
41054	111109	0.178241	['Sine Olivia']	['Philippines']	360.0	Century of Birthinc

```

all_movie_ids = __builtins__.list(set(df_rating['movie_id']))

df_movie_features['movie_id'] = df_movie_features['movie_id'].apply(lambda x: 'other' if x not
df_movie_features = df_movie_features[df_movie_features['movie_id'] != 'other']
#1000 rows x 10 columns

len(__builtins__.list(set(df_movie_features['movie_id']))) == len(__builtins__.list(set(df_rat

↪ False

all_movie_ids_features = __builtins__.list(set(df_movie_features['movie_id']))

df_rating['movie_id'] = df_rating['movie_id'].apply(lambda x: 'other' if x not in all_movie_id
df_rating = df_rating[df_rating['movie_id'] != 'other']

len(__builtins__.list(set(df_movie_features['movie_id']))) == len(__builtins__.list(set(df_rat

↪ True

ratings = df_rating.to_dict('records')

for line in islice(ratings, 2):
    print(json.dumps(line, indent=4))

↪ {
    "user_id": 1,
    "movie_id": 1371,
    "rating": 2.5
  }
  {
    "user_id": 1,
    "movie_id": 1405,
    "rating": 1.0
  }
}

```

✓ Building the ID mapping

✓ CF model

```

dataset = Dataset()
dataset.fit((x['user_id'] for x in ratings), (x['movie_id'] for x in ratings))

# quick check to determine the number of unique users and items in the data
num_users, num_movies = dataset.interactions_shape()
print(f'Num users: {num_users}, num_movies: {num_movies}.')

↪ Num users: 671, num_movies: 2765.

```

✓ Hybrid model

```

def generate_feature_list(df, columns):
    '''
    Generate the list of features of corresponding columns to list
    In order to fit the lightfm Dataset
    '''
    features = df[columns].apply(lambda x: ','.join(x.map(str)), axis = 1)
    features = features.str.split(',')
    features = features.apply(pd.Series).stack().reset_index(drop = True)
    return features

def prepare_item_features(df, columns, id_col_name):
    '''
    Prepare the corresponding feature formats for
    the lightdm.dataset's build_item_features function
    '''
    features = df[columns].apply(lambda x: ','.join(x.map(str)), axis = 1)
    features = features.str.split(',')
    features = __builtins__.list(zip(df[id_col_name], features))
    return features

columns = df_movie_features.columns.to_list()
columns.remove('movie_id')

dataset2 = Dataset()

fitting_item_features = generate_feature_list(df_movie_features, columns)
lightfm_features = prepare_item_features(df_movie_features, columns, 'movie_id')

# dataset2.fit((x['user_id'] for x in ratings), (x['movie_id'] for x in ratings), item_features)
dataset2.fit((x['user_id'] for x in ratings), (x['movie_id'] for x in ratings), item_features)
# item_features = dataset2.build_item_features(((x['movie_id'], x['director_name']) for x in d
item_features = dataset2.build_item_features(lightfm_features, normalize = True)

```

✓ Building the Interaction matrix

✓ CF model

The build_interactions method returns 2 COO sparse matrices, namely the interactions and weights matrices.

```

(interactions, weights) = dataset.build_interactions((x['user_id'], x['movie_id'], x['rating']
print(repr(interactions))

```

```

↗ <671x2765 sparse matrix of type '<class 'numpy.int32'>'
   with 44648 stored elements in COOrdinate format>

```

Split train - test set

```

train_interactions, test_interactions = cross_validation.random_train_test_split
weights, test_percentage=0.2,

```

```
random_state=np.random.RandomState(42))
```

```
print(f"Shape of train interactions: {train_interactions.shape}")
print(f"Shape of test interactions: {test_interactions.shape}")
```

```
↳ Shape of train interactions: (671, 2765)
   Shape of test interactions: (671, 2765)
```

✓ Hybrid model

```
(interactions2, weights2) = dataset2.build_interactions(((x['user_id'], x['movie_id'], x['rating']),
                                                         (x['user_id'], x['movie_id'], x['rating'], x['rating'])))
print(repr(interactions2))
```

```
↳ <671x2765 sparse matrix of type '<class 'numpy.int32'>'
   with 44648 stored elements in COOrdinate format>
```

```
train_interactions2, test_interactions2 = cross_validation.random_train_test_split(
    interactions2, test_percentage=0.2,
    random_state=np.random.RandomState(42))
```

```
print(f"Shape of train interactions: {train_interactions2.shape}")
```