The dataset at hand contains records of movies watched by users and their ratings. Your job is to extract relations of the movies watched by a user and recommend movies to a user based on the previously watched movies. This is same as youtube recommending videos to you saying people who watched this video also watched this, or maybe like Netflix or Amazon prime recommending you other movies or series based on your watch history and of others who have watched the same movies as you.

Import the Libraries

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

Load the data

```
ratings_df = pd.read_csv('/content/ratings_small.csv')
movies_df = pd.read_csv('/content/movies_metadata.csv')
```

```
ratings_df.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

```
ratings_df.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
```

```
movies_df.head()
```

	adult	belongs_to_collection	budget	genres	homepage	i
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	86
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	884
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	NaN	1560
3	False	NaN	16000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	3135
4	False	{'id': 96871, 'name': 'Father of the Bride Col...	0	[{'id': 35, 'name': 'Comedy'}]	NaN	1186

5 rows × 24 columns

```
movies_df.info()
```

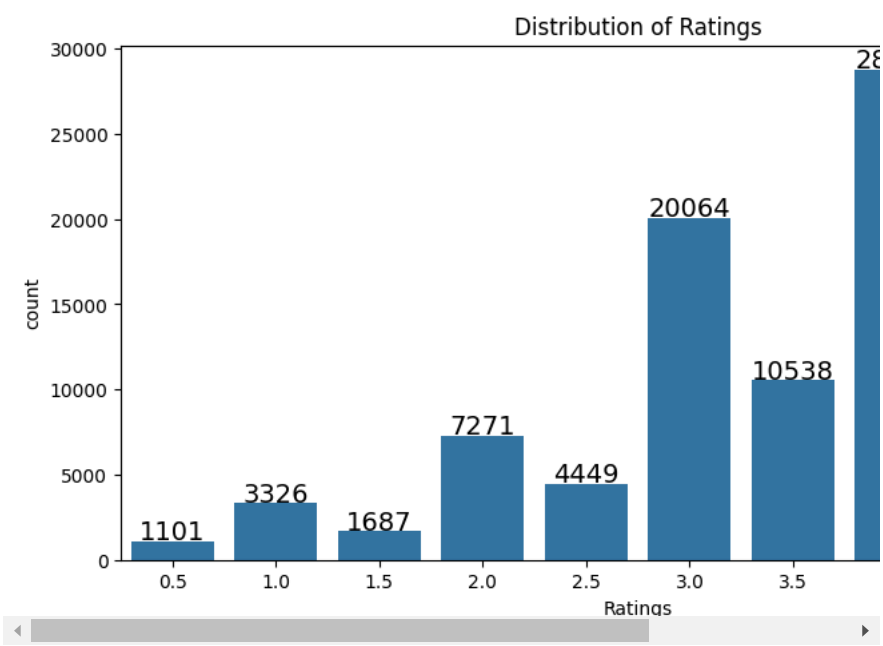
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8965 entries, 0 to 8964
Data columns (total 24 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   adult                  8965 non-null   bool
 1   belongs_to_collection  1273 non-null   object
 2   budget                 8965 non-null   int64
 3   genres                 8965 non-null   object
 4   homepage               565 non-null    object
 5   id                     8965 non-null   int64
 6   imdb_id                8965 non-null   object
 7   original_language      8965 non-null   object
 8   original_title         8965 non-null   object
 9   overview               8940 non-null   object
 10  popularity             8965 non-null   float64
 11  poster_path            8934 non-null   object
 12  production_companies   8965 non-null   object
 13  production_countries   8965 non-null   object
 14  release_date           8960 non-null   object
 15  revenue                8965 non-null   int64
 16  runtime                8959 non-null   float64
 17  spoken_languages       8965 non-null   object
 18  status                 8958 non-null   object
 19  tagline                6378 non-null   object
 20  title                  8964 non-null   object
 21  video                  8964 non-null   object
 22  vote_average           8964 non-null   float64
 23  vote_count             8964 non-null   float64
dtypes: bool(1), float64(4), int64(3), object(16)
memory usage: 1.6+ MB
```

The ratings dataframe contains information of userId, the movieId of the movie watched by that user, the rating given by the user and timestamp.

The movies dataframe contains the information of the movies like movieId, title, genre and so on.

```
plt.figure(figsize=(10,5))
ax = sns.countplot(data=ratings_df, x='rating')
labels = (ratings_df['rating'].value_counts().sort_index())
plt.title('Distribution of Ratings')
plt.xlabel('Ratings')

for i,v in enumerate(labels):
    ax.text(i, v+100, str(v), horizontalalignment='center', size=14, color='black')
plt.show()
```



The ratings distribution shows that there are relatively fewer movies that are lower rated. This can be because most of the users who didn't like the movie, didn't care enough to rate the movie. You should note this, it can be useful later. As you wouldn't want to recommend movies with relatively low number of ratings as users probably didn't like them.

Clean the Data

You can see that in the movies dataframe, there are few records with Nan title. This doesn't serve your purpose as you cannot recommend movies without title. You can remove these records

```
title_mask = movies_df['title'].isna()
```

```
movies_df = movies_df.loc[title_mask == False]
```

Before merging you need to convert the string datatype of id column of movies dataframe to int as that in the ratings dataframe.

```
movies_df = movies_df.astype({'id': 'int64'})
```

```
df = pd.merge(ratings_df, movies_df[['id', 'title']], left_on='movieId', right_on='id')
df.head()
```

	userId	movieId	rating	timestamp	id	title
0	1	1371	2.5	1260759135	1371	Rocky III
1	4	1371	4.0	949810302	1371	Rocky III
2	7	1371	3.0	851869160	1371	Rocky III
3	19	1371	4.0	855193404	1371	Rocky III
4	21	1371	3.0	853852263	1371	Rocky III

Next steps: Generate code with df View recommended plots

The apriori model needs data in a format such that the userId forms the index, the columns are the movie titles and the values can be 1 or 0 depending on whether that user has watched the movie of the corresponding column. The resulting data is like a user's watchlist, for each userId, having 1 in columns of the movies that the user has watched and 0 otherwise.

You can achieve this by using pivot on the dataframe. To do so you need to first make sure there are no duplicate records for the combination of userId and title.

```
df.drop(['timestamp', 'id'], axis=1, inplace=True)
```

```
df = df.drop_duplicates(['userId','title'])
```

```
df_pivot = df.pivot(index='userId', columns='title', values='rating').fillna(0)
```

You need to convert the ratings to 0 or 1 and also convert all float values to int.

```
df_pivot = df_pivot.astype('int64')
```

```
def encode_ratings(x):
    if x<=0:
        return 0
    if x>=1:
        return 1

df_pivot = df_pivot.applymap(encode_ratings)
```

```
df_pivot.head()
```

title	...And God Created Woman	10 Things I Hate About You	11'09''01 - September 11	12 Angry Men	15 Minutes	1900	20,000 Leagues Under the Sea	2001: A Space Odyssey	2010	2 Gram
userId										
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	1	0	
3	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	1	0	0	
5	0	0	0	0	0	0	0	0	0	

Train the Model

The apriori model calculates the probability to determine how likely a user will watch movie M2 if he has already watched a movie M1. It does so by computing support, confidence and lift for different combinations of movies.

```
from mlxtend.frequent_patterns import apriori
```

```
frequent_itemset = apriori(df_pivot, min_support=0.07, use_colnames=True)
```

```
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool type
  warnings.warn(
```

```
frequent_itemset.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning
  and should_run_async(code)
```

	support	itemsets
0	0.102832	(20,000 Leagues Under the Sea)
1	0.129657	(2001: A Space Odyssey)
2	0.298063	(48 Hrs.)
3	0.292101	(5 Card Stud)
4	0.093890	(A Brief History of Time)

Next steps: Generate code with `frequent_itemset` View recommended plots

The apriori algorithm has given you the support, using association_rules you can compute the other paramters like confidence and lift.

```
from mlxtend.frequent_patterns import association_rules

rules = association_rules(frequent_itemset, metric="lift", min_threshold=1)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_c
  and should_run_async(code)
```

```
rules.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning
  and should_run_async(code)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	l
0	(Back to the Future Part II)	(20,000 Leagues Under the Sea)	0.210134	0.102832	0.070045	0.333333	3.241546	0
1	(20,000 Leagues Under the Sea)	(Back to the Future Part II)	0.102832	0.210134	0.070045	0.681159	3.241546	0

Interpret the Results

```
df_res = rules.sort_values(by=['lift'], ascending=False)
df_res.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning
  and should_run_async(code)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	li
792691	(Terminator 3: Rise of the Machines, Solaris, ...	(Young and Innocent, Titanic, Monsoon Wedding)	0.099851	0.093890	0.070045	0.701493	7.4714
792698	(Young and Innocent, Titanic, Monsoon Wedding)	(Terminator 3: Rise of the Machines, Solaris, ...	0.093890	0.099851	0.070045	0.746032	7.4714

Let's see what your model recommends to someone who has watched the Men in Black II

```
df_MIB = df_res[df_res['antecedents'].apply(lambda x: len(x) ==1 and next(iter(x)) == 'Men in Black II')]
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_c
  and should_run_async(code)
```

```
df_MIB = df_MIB[df_MIB['lift'] > 2]
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_c
  and should_run_async(code)
```

```
df_MIB.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning
  and should_run_async(code)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	li
29087	(Men in Black II)	(Terminator 3: Rise of the Machines, Nostalgia)	0.33383	0.080477	0.077496	0.232143	2.8845
439543	(Men in Black II)	(Young and Innocent, Point Break, Terminator	0.33383	0.080477	0.073025	0.218750	2.7181

Next steps: Generate code with df_MIB View recommended plots

```
movies = df_MIB['consequents'].values

movie_list = []
for movie in movies:
    for title in movie:
        if title not in movie_list:
            movie_list.append(title)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_c
  and should_run_async(code)
```

```
movie_list[0:10]
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_c
  and should_run_async(code)
```