Recommend For New Users

1. Simple Filter & Sort

```
-- Choose genre and recommend movies according to the rating.
```

```
# import pandas as pd
# import numpy as np

# # read movies data
# movies = pd.read_csv('ml-20m/movies.csv')
# movies.head()
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

```
# movies.shape
```

```
(27278, 3)
```

```
# len(set(movies.movieId))
```

```
27278
```

```
# from itertools import chain
# x = [s.split('|') for s in movies.genres]
# g_list = set(chain(*x))
# g_list
```

```
{'(no genres listed)',
 'Action',
 'Adventure',
 'Animation',
 'Children',
 'Comedy',
 'Crime',
 'Documentary',
 'Drama',
 'Fantasy',
 'Film-Noir',
 'Horror',
 'IMAX',
 'Musical',
 'Mystery',
 'Romance',
 'Sci-Fi',
 'Thriller',
 'War',
 'Western'}
```

1.1 Recommend according to ratings

```
# ratings = pd.read_csv('ml-20m/ratings.csv')
# ratings.head()
```

	userId	movieId	rating	timestamp
0	1	2	3.5	1112486027
1	1	29	3.5	1112484676
2	1	32	3.5	1112484819
3	1	47	3.5	1112484727
4	1	50	3.5	1112484580

```
# len(set(ratings.movieId))
```

```
26744
```

```
# ratio of rating > 4
len(ratings[ratings['rating']>4])/len(ratings)
```

```
0.22167128502260194
```

22% people rates above 4. So we can think it is a relatively good cutoff of recommending or not.

```
mean = pd.DataFrame(ratings[['movieId','rating']].groupby(['movieId'])['rating'].me
mean['movieId'] = mean.index
mean.index = range(len(mean))
```

```
mean.head()
```

	rating	movieId
0	3.921240	1
1	3.211977	2
2	3.151040	3
3	2.861393	4
4	3.064592	5

```
merged_table = pd.merge(movies, mean, on = 'movieId')
```

```
merged_table.head()
```

	movieId	title	genres	rat
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.921
1	2	Jumanji (1995)	Adventure|Children|Fantasy	3.211
2	3	Grumpier Old Men (1995)	Comedy|Romance	3.151
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	2.861
4	5	Father of the Bride Part II (1995)	Comedy	3.064

```
merged_table.shape
```

```
(26744, 4)
```

```
result = merged_table.sort_values('rating', ascending = False)
def genre_choose_rough(genre):
    final = result[result.genres.str.contains('|'.join(genre))].title[:10].tolist()
    return final
```

```
inp = ['Adventure']
genre_choose_rough(inp)
```

```
['Giorgino (1994)',
 'Life On A String (Bian chang Bian Zou) (1991)',
 'Victor and the Secret of Crocodile Mansion (2012)',
 'Into the Middle of Nowhere (2010)',
 'Stargate SG-1 Children of the Gods - Final Cut (2009)',
 'The Beautiful Story (1992)',
 'Curse of the Ring (Ring of the Nibelungs) (2004)',
 'The Magnificent Gladiator (1964)',
 'Symphony of the Soil (2012)',
 'Itinerary of a Spoiled Child (1988)']
```

1.2 Take the number of ratings for a single film into account

↳ *10 cells hidden*

2. Content-Based Recommendation

```
-- Recommend for a typical movie
```

```
# genome_scores = pd.read_csv('Project91/ml-20m/genome-scores.csv')
# genome_scores.head()
```

	movieId	tagId	relevance
0	1	1	0.02500
1	1	2	0.02500
2	1	3	0.05775
3	1	4	0.09675
4	1	5	0.14675

```
# genome_scores.shape
```

```
(11709768, 3)
```

```
# len(set(genome_scores.movieId))
```

```
10381
```

```
# max(genome_scores.tagId)
```

```
1128
```

```
# transform the genome_scores dataframe into relevance matrix
df = genome_scores.groupby(['movieId'])['relevance'].apply(list).apply(pd.Series)
score_matrix = np.matrix(df)
```

```
df.head()
```

	0	1	2	3	4	5	6	7	8
movieId									
1	0.02500	0.02500	0.05775	0.09675	0.14675	0.21700	0.06700	0.26275	0.26200
2	0.03975	0.04375	0.03775	0.04800	0.11025	0.07250	0.04775	0.10975	0.09925
3	0.04350	0.05475	0.02800	0.07700	0.05400	0.06850	0.05600	0.18500	0.04925
4	0.03725	0.03950	0.03675	0.03100	0.06825	0.04050	0.02325	0.08700	0.05125
5	0.04200	0.05275	0.05925	0.03675	0.07525	0.12525	0.02850	0.08500	0.02950

5 rows × 1128 columns

```
df.shape
```

```
(10381, 1128)
```

```
from sklearn.metrics.pairwise import cosine_similarity

# compute cosine similarity for every movie
cosine_sim = cosine_similarity(score_matrix, score_matrix)
```

```
# get the movieId value and reset the df index
df['movieId']=df.index
df.index = range(len(df))
```

```
df.head()
```

	0	1	2	3	4	5	6	7	8	9
0	0.02500	0.02500	0.05775	0.09675	0.14675	0.21700	0.06700	0.26275	0.26200	0.0320(
1	0.03975	0.04375	0.03775	0.04800	0.11025	0.07250	0.04775	0.10975	0.09925	0.0205(
2	0.04350	0.05475	0.02800	0.07700	0.05400	0.06850	0.05600	0.18500	0.04925	0.0267!
3	0.03725	0.03950	0.03675	0.03100	0.06825	0.04050	0.02325	0.08700	0.05125	0.0302!
4	0.04200	0.05275	0.05925	0.03675	0.07525	0.12525	0.02850	0.08500	0.02950	0.0287!

5 rows × 1129 columns

```
def recommendation(title):

    # link title to movieId and find the row number of the matrix
    ids = movies[movies['title'] == title]['movieId'].values[0]
    i = df[df.movieId==ids].index.values[0]

    # find the specific score for the movie and sort the scores
    sim_scores = list(enumerate(cosine_sim[i]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # store the row numbers and related movieId
    indices = [i[0] for i in sim_scores[1:]]
    movie_ids = df.iloc[indices,:].movieId.values

    result = []
    for i in movie_ids:
        result.append(movies[movies.movieId == i].title.values[0])

    return result
```

```
# recommend 10 movies
recommendation('Toy Story (1995)')[:10]
```

```
['Monsters, Inc. (2001)',
 'Toy Story 2 (1999)',
 "Bug's Life, A (1998)",
 'Finding Nemo (2003)',
 'Toy Story 3 (2010)',
 'Ratatouille (2007)',
 'Ice Age (2002)',
 'Shrek (2001)',
 'Up (2009)',
 'Antz (1998)']
```