24 FRAMEZZ

Movie Recommender System

Kiruthika Sankaran,Abhinav Tummala 5/14/18 Data Mining

Contents

1.0 EXECUTIVE SUMMARY:

“The Cinema is truth at 24 frames per second,” says Jean-Luc Godard. Movies are ways to hear the story. It is the combination of so many arts. It is a source of entertainment and information. Many people in this world love movies and are looking for ways to select good movies to watch. Our system “24 FRAMEZZ” is the emerging online recommender system for movies which tries to fulfill the needs of such people.

Our objective is to develop a recommender system for movies which will be tailored to users’ specific preferences and needs based on user ratings and movie genres. Our personalized recommendation engine will help users narrow the universe of potential films to fit their unique tastes by breaking down movies into long lists of attributes and matches these elements to a viewer's preferences.

2.0 PROBLEM STATEMENT DESCRIPTION:

Our aim is to provide movie recommendation to two types of users.

1. Registered Users
2. Unregistered Users

1. Registered Users

The registered users have a registered account to the movie lens website. They have a unique user_id and have rated many movies they have watched. Our aim is to provide personalized recommendation to such users based on the movies they have watched previously.

2. Unregistered Users

There are another type of user who casually searches for good movies to watch. We do not have any information about them. We are going to provide them generalized recommendation of movies. These recommendations will be based on the input they are providing to the system. The recommendations could be based on their preference to the following:

- Their favorite genres
- Top rated movies of their favorite actors/cast member
- Most popular movies of their favorite actor/cast member
- Most popular movies of a particular year

3.0 DATASET:

We have combined two datasets for our problem.

1. Movie lens dataset (https://grouplens.org/datasets/movielens/)
2. IMDB dataset (https://www.imdb.com/interfaces/)

Brief Description of the dataset:

1. Movie lens dataset - contains 6 files of which we made use of 3 files.

File Name	Description	Columns name	No. of Records
movies.csv	Information about movies and genre	Movie_id, title, genres	9125
ratings.csv	Registered user's rating to movie	User_id, movie_id, rating	100004
links.csv	Movies link to imdb database	Movie_id, imdb_id	9125

2. IMDB dataset - contains 7 files of which we made use of 4 files.

File Name	Description	Columns name	No. of Records
name_basics.csv	Contains cast member details nconst - unique id for the cast member	Nconst, primaryName, birthyear,deathyear, primary profession	8604355
title_basics.csv	Contains movie details	Tconst, titletype, primarytitle, original title, isAdult, startyear, endyear, runtime	4997168
title_principle.csv	Contains the primary cast members for the movie	Tconst, nconst, category	28213889
title_ratings.csv	Contains imdb ratings details for a movie	Tconst, averageRatings, numberofvotes	831498

Note:

- Because imdb dataset is quite large we have taken the subset of movies from imdb which are present in movie lens dataset only.
- The screenshot of dataset is attached in appendix A.
- The linking of movie lens and imdb dataset is made using the merge of imdb_id from movie lens and tconst from imdb. (proof attached in appendix A).

4.0 ALGORITHMS FOR VARIOUS RECOMMENDATIONS

A. ITEM BASED COLLABORATIVE FILTERING FOR REGISTERED USERS:

For registered user as mentioned before we have planned to recommend movies based on their previous ratings.

Logical flow:

- Read movies.csv and ratings.csv into data frames

- Drop timeframe column from ratings, merge both data frames
- Create a pivot table name user rating with user_id as rows and movie_title as columns. The cell value will be the user's rating to each of the movie. It is a sparse matrix with lot of NaN values as the users may not have rated many movies.
- Create a correlation matrix for this table by finding the correlation between the movies.
- Obtain the user_id from the user.
- Fetch the row for the user from the user ratings. Remove NaN values (movies not rated by the user) and save into a dataframe called myRating. We are going to build recommendation by finding movies similar to these movies.
- For each of the movies list from myRating, fetch the entries from corr_matrix to get the similar movies. Drop the missing values from the corr_matrix.
- Scale the movies in corr_matrix based on the previous ratings by the user.
- Add all the scores to the particular movie. Filter the movies the user had already watched.
- Fetch top 5 movies and output the recommendations.

Input:

```
In [*]: user_id = input('enter your user ID')
        myRatings = userRatings.loc[int(user_id)].dropna()
        simCandidates = pd.Series()
        for i in range(0, len(myRatings.index)):
            sims = corrMatrix[myRatings.index[i]].dropna()
            sims = sims.map(lambda x: x * myRatings[i])
            simCandidates = simCandidates.append(sims)
        simCandidates = simCandidates.groupby(simCandidates.index).sum()
        simCandidates.sort_values(inplace = True, ascending = False)
        filteredSims = simCandidates.drop(myRatings.index)
        filteredSims.head()

        enter your user ID 2
```

Output:

```
         filteredSims.head()

         enter your user ID2

Out[26]: Of Mice and Men (1992)              132.643170
         Ice Age 2: The Meltdown (2006)      118.498342
         Wall Street (1987)                  116.851652
         Court Jester, The (1956)            114.863410
         Shanghai Knights (2003)             114.484460
         dtype: float64
```

B. RECOMMENDATION BASED ON THE GENRE

For unregistered users, we have planned to recommend top rated movies based on their genre preferences. Here we have taken the ratings from imdb dataset instead of movie_lens dataset as the imdb ratings are latest and rated by more people.

Logical flow:

- Read the movies.csv and links.csv from movie lens dataset into a dataframe.

- Read the ratings.csv from the imdb dataset.
- Imdb id contains ‘tt’ before the id. In order to make it more consistent, remove the ‘tt’ from the imdb ratings and convert the id to int. (for eg: in imdb dataset, toy story has id tt114709 while movie_lens has 114709. So remove ‘tt’ before imdb_id).
- Take the subset of movies from the imdb dataset which are contained in movie lens dataset which is 9112 rows.
- Take the genre from the movies and convert each genre to a column and fill the cell values as 1 if that movie belongs to particular genre.
- Merge the links and imdb_ratings dataframe on imdb_id as merged_data.
- Merge the movies dataframe to the merged_data.
- Multiply each of the genre column with the average rating of the movie. So, if the movie does not belong to a genre its value will be 0 otherwise, it contains the average rating for that movie.
- Get the genre preference from the user. Pick the highest 5 records from that column and give the output.

Input:

In [31]:
```
print("WELCOME TO 24 FRAMEZZ")
print("we are going to provide you the movies you should watch!!!")
gen_df
```

```
WELCOME TO 24 FRAMEZZ
we are going to provide you the movies you should watch!!!
```

Out[31]:

	Genre
0	action
1	adventure
2	animation
3	children
4	comedy
5	crime
6	documentary
7	drama
8	fantasy
9	horror
10	imax
11	musical
12	mystery
13	noir
14	romance
15	scifi
16	thriller
17	war
18	western

In [*]:
```
a = input('enter your genre of interest: (enter any number between 0-18)')
reco = merged_data.nlargest(5,gen_df.loc[int(a)] )
print(reco.to_csv(columns=['title', 'averageRating'], sep='\t', index=False))
```

```
enter your genre of interest: (enter any number between 0-18)
```

Output:

```
In [32]: a = input('enter your genre of interest: (enter any number between 0-18)')
         reco = merged_data.nlargest(5,gen_df.loc[int(a)] )
         print(reco.to_csv(columns=['title', 'averageRating'], sep='\t', index=False))

         enter your genre of interest: (enter any number between 0-18)9
         title   averageRating
         Michael Jackson's Thriller (1983)       8.7
         Silence of the Lambs, The (1991)        8.6
         Alien (1979)    8.5
         Psycho (1960)   8.5
         Aliens (1986)   8.4
```

C. TOP RATED OR POPULAR MOVIES OF THE USER'S FAVORITE CAST:

We have planned to provide recommendation to the user based on their favorite cast. Here there are two types of recommendations. This involves combining multiple data files into a single dataframe.

1. Give top rated movies - have high average_rating in imdb database
2. Give most popular movies - voted by more number of people.

Logical flow:

- As mentioned before, take the subset of movies by matching the imdb id from links.csv from movie lens and tconst from title.csv from imdb dataset into filter_titles dataframe.
- The principle cast_id details for each movie is obtained from principle.csv. Obtain and merge the principle cast against each movie.
- The details of each cast is obtained from name.csv and merged to the previous dataframe.
- The rating and number of votes for each movie is obtained from rating.csv and merged again. The final dataframe contains 89613 rows.
- The favorite cast is obtained from the user and user is also given an option of choosing the top rated or popular movie.
- Filter on the cast and obtain the top-rated or popular movie from the final dataframe.

The final dataframe contains the following information:

```
In [61]: merge_cast_title_name_ranks.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 89613 entries, 0 to 89612
         Data columns (total 19 columns):
         nconst               89613 non-null object
         primaryName          89613 non-null object
         birthYear            89613 non-null object
         deathYear            89613 non-null object
         primaryProfession    88896 non-null object
         tconst               89613 non-null int64
         ordering             89613 non-null int64
         category             89613 non-null object
         characters           89613 non-null object
         titleType            89613 non-null object
         primaryTitle         89613 non-null object
         originalTitle        89613 non-null object
         isAdult              89613 non-null int64
         startYear            89613 non-null object
         endYear              89613 non-null object
         runtimeMinutes       89613 non-null object
         genres               89613 non-null object
         averageRating        89613 non-null float64
         numVotes             89613 non-null int64
         dtypes: float64(1), int64(4), object(14)
         memory usage: 13.7+ MB
```

Input:

```
In [11]: a = input('enter your favourite cast name')
         b = input('enter 1 if toprated or 2 if popular:')
         a=a.lower()
         b=int(b)
         if b ==1:
             reco=merge_cast_title_name_ranks[merge_cast_title_name_ranks.primaryName.str.contains(a)]
             output=reco.nlargest(10,'averageRating')
             df=output[['primaryTitle','numVotes','averageRating','numVotes']]
             print (df.to_string(index=False))
         elif b ==2:
             reco=merge_cast_title_name_ranks[merge_cast_title_name_ranks.primaryName.str.contains(a)]
             output=reco.nlargest(10,'numVotes')
             df=output[['primaryTitle','numVotes','averageRating','numVotes']]
             print (df.to_string(index=False))
```

<u>Case 1 - popular movies of the interested cast</u>

```
enter your favourite cast nameTom Hanks
enter 1 if toprated or 2 if popular:2
primaryTitle  numVotes  averageRating  numVotes
       Forrest Gump   1480633            8.8   1480633
 Saving Private Ryan  1028336            8.6   1028336
      The Green Mile   929737            8.5    929737
           Toy Story   732770            8.3    732770
 Catch Me If You Can   654458            8.1    654458
         Toy Story 3   634927            8.3    634927
           Cast Away   446123            7.8    446123
         Toy Story 2   436599            7.9    436599
    Captain Phillips   367200            7.8    367200
   The Da Vinci Code   355093            6.6    355093
```

<u>Case 2 - top rated movies of the interested cast</u>

```
enter your favourite cast nameTom Hanks
enter 1 if toprated or 2 if popular:1
primaryTitle  numVotes  averageRating  numVotes
              Forrest Gump  1480633            8.8   1480633
From the Earth to the Moon     8987            8.7      8987
       Saving Private Ryan  1028336            8.6   1028336
            The Green Mile   929737            8.5    929737
                 Toy Story   732770            8.3    732770
               Toy Story 3   634927            8.3    634927
       Catch Me If You Can   654458            8.1    654458
               Toy Story 2   436599            7.9    436599
                 Cast Away   446123            7.8    446123
  Neil Young: Heart of Gold    2657            7.8      2657
```

D. POPULAR MOVIES OF THE PARTICULAR YEAR:

Here we want to recommend popular movies based on the year.

Logical flow:

- The same merged data frame of the previous section is used here.
- Obtain the year input from the user.
- Filter on the start year and pick top 10 highest values from the numVotes column and give the output.

```
In [16]: #Suggest top 10 popular movies based on the year
         a = input('enter the year in which you are interested to know about')
         reco=filter_titles_ratings[filter_titles_ratings.startYear.str.strip() == a]
         output=reco.nlargest(10,'numVotes')
         df=output[['primaryTitle','numVotes','averageRating','numVotes']]
         print (df.to_string(index=False))
```

```
enter the year in which you are interested to know about2016
primaryTitle  numVotes  averageRating  numVotes
                          Deadpool    722905           8.0    722905
Batman v Superman: Dawn of Justice    532017           6.6    532017
        Captain America: Civil War    497113           7.8    497113
                   Stranger Things    480355           8.9    480355
                     Suicide Squad    476732           6.1    476732
                          Zootopia    343853           8.0    343853
                X-Men: Apocalypse    318266           7.0    318266
               10 Cloverfield Lane    235852           7.2    235852
                   The Jungle Book    224498           7.4    224498
                     The Nice Guys    213845           7.4    213845
```

FUTURE ENHANCEMENTS:

The recommender system could further be enhanced by various aspects.

Split the cast to favorite director, actor, and actress.

Output the full data and details rather than just movies which can provide further insights to the user and urges them to watch the movie.

Recommend to the registered users based on their genre preferences.

Recommend the movies between particular periods of years.

Also an interactive frontend for these recommendations will enhance the project to a whole new level.

CONCLUSION:

Our objective was to develop a recommender system for movies which will be tailored to users' specific preferences and needs based on user ratings and movie genres. The personalization is only limited to the users who are registered. The recommendations are made based on the ratings the user has provided to different movies. Both registered and unregistered users will be able to access the genre based, cast (actor, director or any personnel) based classification depending on the popularity or highest ratings criterion. In addition to it, users can also get recommendations based on the year the movie is released. We also incorporated IMDB dataset because it already has vast user base and is also a legitimate data source. All these choices are provided because each individual has different preferences and different perspectives when it comes to making choices. We tried to provide few types of recommendation systems to match as many users as possible.

APPENDIX A: Screenshot of dataset

MOVIELENS:

Movies.csv

In [7]:
```
movies.head()
```

Out[7]:

	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

In [8]:
```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9125 entries, 0 to 9124
Data columns (total 3 columns):
movieId    9125 non-null int64
title      9125 non-null object
genres     9125 non-null object
dtypes: int64(1), object(2)
memory usage: 213.9+ KB
```

Ratings.csv

In [11]:
```
ratings = pd.read_csv('ratings.csv')
ratings.head()
```

Out[11]:

	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

In [9]:
```
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100004 entries, 0 to 100003
Data columns (total 6 columns):
movieId      100004 non-null int64
title        100004 non-null object
genres       100004 non-null object
userId       100004 non-null int64
rating       100004 non-null float64
timestamp    100004 non-null int64
dtypes: float64(1), int64(3), object(2)
memory usage: 5.3+ MB
```

Links.csv

In [4]:
```
links.head()
```

Out[4]:

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0

In [3]:
```
links.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9125 entries, 0 to 9124
Data columns (total 3 columns):
movieId    9125 non-null int64
imdbId     9125 non-null int64
tmdbId     9112 non-null float64
dtypes: float64(1), int64(2)
memory usage: 213.9 KB
```

IMDB Dataset:

Name.csv

```
In [3]: name.head()
Out[3]:
```

	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
0	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	tt0043044,tt0045537,tt0050419,tt0072308
1	nm0000002	Lauren Bacall	1924	2014	actress,soundtrack	tt0037382,tt0117057,tt0040506,tt0038355
2	nm0000003	Brigitte Bardot	1934	\N	actress,soundtrack,producer	tt0057345,tt0049189,tt0054452,tt0059956
3	nm0000004	John Belushi	1949	1982	actor,writer,soundtrack	tt0080455,tt0072562,tt0077975,tt0078723
4	nm0000005	Ingmar Bergman	1918	2007	writer,director,actor	tt0083922,tt0050976,tt0050986,tt0060827

```
In [4]: name.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8604355 entries, 0 to 8604354
Data columns (total 6 columns):
nconst               object
primaryName          object
birthYear            object
deathYear            object
primaryProfession    object
knownForTitles       object
dtypes: object(6)
memory usage: 393.9+ MB
```

Title_basics.csv

```
In [6]: title.head()
Out[6]:
```

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
0	tt0000001	short	Carmencita	Carmencita	0	1894	\N	1	Documentary,Short
1	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	0	1892	\N	5	Animation,Short
2	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	0	1892	\N	4	Animation,Comedy,Romance
3	tt0000004	short	Un bon bock	Un bon bock	0	1892	\N	\N	Animation,Short
4	tt0000005	short	Blacksmith Scene	Blacksmith Scene	0	1893	\N	1	Short

```
In [7]: title.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4997168 entries, 0 to 4997167
Data columns (total 9 columns):
tconst            object
titleType         object
primaryTitle      object
originalTitle     object
isAdult           int64
startYear         object
endYear           object
runtimeMinutes    object
genres            object
dtypes: int64(1), object(8)
memory usage: 343.1+ MB
```

Title_principle.csv

```
In [9]: principle.head()
```

Out[9]:

	tconst	ordering	nconst	category	job	characters
0	tt0000001	1	nm1588970	self	\N	["Herself"]
1	tt0000001	2	nm0005690	director	\N	\N
2	tt0000001	3	nm0374658	cinematographer	director of photography	\N
3	tt0000002	1	nm0721526	director	\N	\N
4	tt0000002	2	nm1335271	composer	\N	\N

```
In [8]: principle.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28213889 entries, 0 to 28213888
Data columns (total 6 columns):
tconst        object
ordering      int64
nconst        object
category      object
job           object
characters    object
dtypes: int64(1), object(5)
memory usage: 1.3+ GB
```

Ratings.csv

```
In [10]: ratings.head()
```

Out[10]:

	tconst	averageRating	numVotes
0	tt0000001	5.8	1373
1	tt0000002	6.5	160
2	tt0000003	6.6	954
3	tt0000004	6.4	96
4	tt0000005	6.2	1653

```
In [11]: ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 831498 entries, 0 to 831497
Data columns (total 3 columns):
tconst           831498 non-null object
averageRating    831498 non-null float64
numVotes         831498 non-null int64
dtypes: float64(1), int64(1), object(1)
memory usage: 19.0+ MB
```

Linking of imdb dataset and movie lens dataset

```
In [13]: title.tconst = title.tconst.str.replace('tt','')
```

```
In [14]: title.tconst = pd.to_numeric(title['tconst'], errors='coerce')
```

```
In [17]: #imdb dataset - title.csv - with imdb id = 114709
         title[title.tconst == 114709]
```

Out[17]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
112245	114709	movie	Toy Story	Toy Story	0	1995	\N	81	Adventure,Animation,Comedy

```
In [20]: #movie_lens dataset - links.csv - with imdb id = 114709 and movieId =1
         links[links.imdbId == 114709]
```

Out[20]:

	movieId	imdbId	tmdbId
0	1	114709	862.0

```
In [21]: movies = pd.read_csv('movies.csv')
```

```
In [22]: #movie_lens dataset with movieId = 1
         movies[movies.movieId == 1]
```

Out[22]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy

ITEM BASED COLLOBORATIVE FILTERING - REGISTERED USERS

In [ ]:
```
import pandas as pd
import numpy as np
movies = pd.read_csv('movies.csv')
ratings = pd.read_csv('ratings.csv')
ratings=pd.merge(movies,ratings)
```

In [26]:
```
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100004 entries, 0 to 100003
Data columns (total 6 columns):
movieId      100004 non-null int64
title        100004 non-null object
genres       100004 non-null object
userId       100004 non-null int64
rating       100004 non-null float64
timestamp    100004 non-null int64
dtypes: float64(1), int64(3), object(2)
memory usage: 5.3+ MB
```

In [27]:
```
ratings.head()
```

Out[27]:

	movieId	title	genres	userId	rating	timestam
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	7	3.0	851866703
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	9	4.0	938629179
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	13	5.0	133138005
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	15	2.0	997938310
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	19	3.0	855190091

In [28]:
```
userRatings = ratings.pivot_table(index=['userId'],columns=['title'],values
='rating')
corrMatrix = userRatings.corr(method='pearson')
```

In [34]:
```
user_id = input('enter your user ID')
myRatings = userRatings.loc[int(user_id)].dropna()
simCandidates = pd.Series()
for i in range(0, len(myRatings.index)):
    sims = corrMatrix[myRatings.index[i]].dropna()
    sims = sims.map(lambda x: x * myRatings[i])
    simCandidates = simCandidates.append(sims)
simCandidates = simCandidates.groupby(simCandidates.index).sum()
simCandidates.sort_values(inplace = True, ascending = False)
filteredSims = simCandidates.drop(myRatings.index)
filteredSims.head()
```

```
enter your user ID2
```

Out[34]:
```
Of Mice and Men (1992)            132.643170
Ice Age 2: The Meltdown (2006)    118.498342
Wall Street (1987)                116.851652
Court Jester, The (1956)          114.863410
Shanghai Knights (2003)           114.484460
dtype: float64
```

GENRE BASED RECOMMENDATION

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.feature_extraction.text import CountVectorizer
```

In [7]:
```
#read movies-links
movies = pd.read_csv('movies.csv')
links = pd.read_csv('links.csv')
```

In [11]:
```
#read imdb_ratings file
imdb_ratings = pd.read_csv('title_ratings.csv')
```

In [12]:
```
#remove tt to make consistent with imdb and movie_lens data
imdb_ratings.tconst = imdb_ratings.tconst.str.replace('tt','')
imdb_ratings.tconst = pd.to_numeric(imdb_ratings['tconst'], errors='coerce')
```

In [13]:
```
#fetch movie ratings from imdb for movie_lens
movie_rating = imdb_ratings.loc[imdb_ratings['tconst'].isin(links.imdbId)]
```

In [48]:
```
movie_rating.head()
```

Out[48]:

	imdbId	averageRating	numVotes
270	417	8.2	35532
1335	4972	6.7	19191
1542	6333	7.0	1303
1637	6864	8.0	12029
1809	8133	7.8	6088

In [14]:
```
#convert genres to columns
#from sklearn.feature_extraction.text import CountVectorizer
movies.genres=movies.genres.str.split("|")
movies.genres=movies.genres.str.join(' ')
movies.genres=movies.genres.str.replace('Sci-Fi', 'SciFi')
movies.genres=movies.genres.str.replace('Film-Noir', 'Noir')
movies.genres=movies.genres.str.replace('(no genres listed)', '')
cv = CountVectorizer ()
X = cv.fit_transform(movies.genres).toarray()
X = pd.DataFrame(X)
res = {v: k for k, v in cv.vocabulary_.items()}
X.columns = pd.Series(X.columns).map(res)
movies = pd.concat([movies, X], axis = 1)
```

In [15]:
```
movies.head()
```

Out[15]:

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

5 rows × 22 columns

In [16]:
```
movie_rating.rename(columns={'tconst': 'imdbId'}, inplace=True)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py:2844: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  **kwargs)
```

In [17]:
```
merged_data=pd.merge(links,movie_rating)
```

In [22]:
```
merged_data.head()
```

Out[22]:

	movieId	imdbId	averageRating	numVotes	title	genres	action	adventure	an
0	1	114709	8.3	732770	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	0	1	1
1	2	113497	6.9	252078	Jumanji (1995)	Adventure Children Fantasy	0	1	0
2	3	113228	6.6	21332	Grumpier Old Men (1995)	Comedy Romance	0	0	0
3	4	114885	5.7	8225	Waiting to Exhale (1995)	Comedy Drama Romance	0	0	0
4	5	113041	6.0	29595	Father of the Bride Part II (1995)	Comedy	0	0	0

5 rows × 25 columns

In [19]:
```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9112 entries, 0 to 9111
Data columns (total 5 columns):
movieId          9112 non-null int64
imdbId           9112 non-null int64
tmdbId           9101 non-null float64
averageRating    9112 non-null float64
numVotes         9112 non-null int64
dtypes: float64(2), int64(3)
memory usage: 427.1 KB
```

In [20]:
```
merged_data = merged_data.drop(['tmdbId'], axis=1)
```

In [23]:
```
merged_data=pd.merge(merged_data,movies)
```

In [24]:
```
merged_data.noir.value_counts()
```

Out[24]:
```
0    8979
1     133
Name: noir, dtype: int64
```

In [25]:
```
merged_data['action'] = merged_data['action']*merged_data['averageRating']
merged_data['adventure'] = merged_data['adventure']*merged_data['averageRatin
g']
merged_data['animation'] = merged_data['animation']*merged_data['averageRatin
g']
merged_data['children'] = merged_data['children']*merged_data['averageRating']
merged_data['comedy'] = merged_data['comedy']*merged_data['averageRating']
merged_data['crime'] = merged_data['crime']*merged_data['averageRating']
merged_data['documentary'] = merged_data['documentary']*merged_data['averageRa
ting']
merged_data['drama'] = merged_data['drama']*merged_data['averageRating']
merged_data['fantasy'] = merged_data['fantasy']*merged_data['averageRating']
merged_data['horror'] = merged_data['horror']*merged_data['averageRating']
merged_data['imax'] = merged_data['imax']*merged_data['averageRating']
merged_data['musical'] = merged_data['musical']*merged_data['averageRating']
merged_data['mystery'] = merged_data['mystery']*merged_data['averageRating']
merged_data['noir'] = merged_data['noir']*merged_data['averageRating']
merged_data['romance'] = merged_data['romance']*merged_data['averageRating']
merged_data['scifi'] = merged_data['scifi']*merged_data['averageRating']
merged_data['thriller'] = merged_data['thriller']*merged_data['averageRating']
merged_data['war'] = merged_data['war']*merged_data['averageRating']
merged_data['western'] = merged_data['western']*merged_data['averageRating']
```

In [44]:
```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9112 entries, 0 to 9111
Data columns (total 25 columns):
movieId          9112 non-null int64
imdbId           9112 non-null int64
averageRating    9112 non-null float64
numVotes         9112 non-null int64
title            9112 non-null object
genres           9112 non-null object
action           9112 non-null float64
adventure        9112 non-null float64
animation        9112 non-null float64
children         9112 non-null float64
comedy           9112 non-null float64
crime            9112 non-null float64
documentary      9112 non-null float64
drama            9112 non-null float64
fantasy          9112 non-null float64
horror           9112 non-null float64
imax             9112 non-null float64
musical          9112 non-null float64
mystery          9112 non-null float64
noir             9112 non-null float64
romance          9112 non-null float64
scifi            9112 non-null float64
thriller         9112 non-null float64
war              9112 non-null float64
western          9112 non-null float64
dtypes: float64(20), int64(3), object(2)
memory usage: 1.8+ MB
```

In [27]:
```
reco = merged_data.nlargest(5, 'crime')
print(reco['title'])
```

```
284     Shawshank Redemption, The (1994)
695                Godfather, The (1972)
5819     Decalogue, The (Dekalog) (1989)
977       Godfather: Part II, The (1974)
6909              Dark Knight, The (2008)
Name: title, dtype: object
```

In [42]:
```
genre = { 'Genre': ['action', 'adventure', 'animation','children','comedy','cr
ime','documentary','drama','fantasy',
                   'horror','imax','musical','mystery','noir','romance','scif
i','thriller','war','western']}
gen_df = pd.DataFrame(data=genre)
```

In [43]:
```
print("WELCOME TO 24 FRAMEZZ")
print("we are going to provide you the movies you should watch!!!")
gen_df
```

```
WELCOME TO 24 FRAMEZZ
we are going to provide you the movies you should watch!!!
```

Out[43]:

	Genre
0	action
1	adventure
2	animation
3	children
4	comedy
5	crime
6	documentary
7	drama
8	fantasy
9	horror
10	imax
11	musical
12	mystery
13	noir
14	romance
15	scifi
16	thriller
17	war
18	western

In [51]:
```
a = input('enter your genre of interest: (enter any number between 0-18)')
reco = merged_data.nlargest(5,gen_df.loc[int(a)] )
print(reco.to_csv(columns=['title', 'averageRating'], sep='\t', index=False))
```

```
enter your genre of interest: (enter any number between 0-18)4
title	averageRating
Pulp Fiction (1994)	8.9
Forrest Gump (1994)	8.8
Fawlty Towers (1975-1979)	8.8
Bill Hicks: Relentless (1992)	8.8
George Carlin: Jammin' in New York (1992)	8.8
```

RECOMMEND BASED ON CAST AND YEAR

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

name = pd.read_csv('name_basics.csv') akas = pd.read_csv('title_akas.csv') title = pd.read_csv('title_basics.csv') crew = pd.read_csv('title_crew.csv') episode = pd.read_csv('title_episode.csv') principle = pd.read_csv('title_principle.csv') ratings = pd.read_csv('title_ratings.csv')

In [2]:
```
principle = pd.read_csv('title_principle.csv')
title = pd.read_csv('title_basics.csv')
name = pd.read_csv('name_basics.csv')
links = pd.read_csv('links.csv')
ratings = pd.read_csv('title_ratings.csv')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:2
717: DtypeWarning: Columns (5) have mixed types. Specify dtype option on impo
rt or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [3]:
```
#Drop unnecessary columns
principle=principle.drop('job',axis=1)
name=name.drop('knownForTitles',axis=1)
```

In [4]:
```
#Convert tconst into numeric
title.tconst = title.tconst.str.replace('tt','')
title.tconst = pd.to_numeric(title['tconst'], errors='coerce')
```

In [5]:
```
#Convert tconst into numeric
ratings.tconst = ratings.tconst.str.replace('tt','')
ratings.tconst = pd.to_numeric(ratings['tconst'], errors='coerce')
```

In [6]:
```
#convert tconst into numeric
principle.tconst = principle.tconst.str.replace('tt','')
principle.tconst = pd.to_numeric(principle['tconst'], errors='coerce')
#principle[principle.tconst == 114709]
```

In [7]:
```
#Filter the data from movie lens dataset in IMDB
filter_titles=title.loc[title['tconst'].isin(links.imdbId)]
#filter_titles=pd.merge(filter_titles,title)
```

In [8]:
```
#Filter the cast names associated to each movie and the principle cast names f
rom IMDB and merge the filtered lists
filter_cast_titles=principle.loc[principle['tconst'].isin(filter_titles.tconst
)]
filter_cast_titles=pd.merge(filter_cast_titles,filter_titles)
```

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

Out[32]:

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMin
414	417	short	A Trip to the Moon	Le voyage dans la lune	0	1902	\N	13
4912	4972	movie	The Birth of a Nation	The Birth of a Nation	0	1915	\N	195
6258	6333	movie	20,000 Leagues Under the Sea	20,000 Leagues Under the Sea	0	1916	\N	105
6781	6864	movie	Intolerance: Love's Struggle Throughout the Ages	Intolerance: Love's Struggle Throughout the Ages	0	1916	\N	163
8026	8133	short	The Immigrant	The Immigrant	0	1917	\N	30

In [9]:
```
#Filter the cast from the IMDB dataset and merge the data to form an aggregate
d dataset
filter_cast_titles_name=name.loc[name['nconst'].isin(filter_cast_titles.nconst
)]
merge_cast_title_name=pd.merge(filter_cast_titles_name,filter_cast_titles)
```

In [10]:
```
#merge the ratings and aggregated list
merge_cast_title_name_ranks=pd.merge(merge_cast_title_name,ratings)
merge_cast_title_name_ranks.primaryName=merge_cast_title_name_ranks.primaryNam
e.str.lower()
```

In [55]:
```
merge_cast_title_name_ranks.head()
```

Out[55]:

	nconst	primaryName	birthYear	deathYear	primaryProfession	tconst
0	nm0000001	Fred Astaire	1899	1987	soundtrack,actor,miscellaneous	25164
1	nm0001677	Ginger Rogers	1911	1995	actress,soundtrack	25164
2	nm0002143	Edward Everett Horton	1886	1970	actor,soundtrack,director	25164
3	nm0103567	Alice Brady	1892	1939	actress,soundtrack	25164
4	nm0388755	Samuel Hoffenstein	1890	1947	writer	25164

In [12]:
```
#Suggest top rated movies or popular movies related each cast depending on use
r inputs
a = input('enter your favourite cast name')
b = input('enter 1 if toprated or 2 if popular:')
a=a.lower()
b=int(b)
if b ==1:
    reco=merge_cast_title_name_ranks[merge_cast_title_name_ranks.primaryName.s
tr.contains(a)]
    output=reco.nlargest(10,'averageRating')
    df=output[['primaryTitle','numVotes','averageRating','numVotes']]
    print (df.to_string(index=False))
elif b ==2:
    reco=merge_cast_title_name_ranks[merge_cast_title_name_ranks.primaryName.s
tr.contains(a)]
    output=reco.nlargest(10,'numVotes')
    df=output[['primaryTitle','numVotes','averageRating','numVotes']]
    print (df.to_string(index=False))
```

```
enter your favourite cast nameTom Hanks
enter 1 if toprated or 2 if popular:2
primaryTitle  numVotes  averageRating  numVotes
       Forrest Gump   1480633            8.8   1480633
Saving Private Ryan   1028336            8.6   1028336
     The Green Mile    929737            8.5    929737
          Toy Story    732770            8.3    732770
Catch Me If You Can    654458            8.1    654458
        Toy Story 3    634927            8.3    634927
          Cast Away    446123            7.8    446123
        Toy Story 2    436599            7.9    436599
   Captain Phillips    367200            7.8    367200
  The Da Vinci Code    355093            6.6    355093
```

In [13]:
```
#Merge the ratings and titles tables so that ratings and titles are present at
 one place
filter_titles_ratings=pd.merge(ratings,filter_titles)
#filter_titles_ratings.startYear=filter_titles_ratings.startYear.isnull().valu
e_counts()
```

In [16]:
```
#Suggest top 10 popular movies based on the year
a = input('enter the year in which you are interested to know about')
reco=filter_titles_ratings[filter_titles_ratings.startYear.str.strip() == a]
output=reco.nlargest(10,'numVotes')
df=output[['primaryTitle','numVotes','averageRating','numVotes']]
print (df.to_string(index=False))
```

```
enter the year in which you are interested to know about2016
primaryTitle  numVotes  averageRating  numVotes
                         Deadpool    722905            8.0    722905
Batman v Superman: Dawn of Justice    532017            6.6    532017
        Captain America: Civil War    497113            7.8    497113
                  Stranger Things    480355            8.9    480355
                    Suicide Squad    476732            6.1    476732
                         Zootopia    343853            8.0    343853
                X-Men: Apocalypse    318266            7.0    318266
              10 Cloverfield Lane    235852            7.2    235852
                  The Jungle Book    224498            7.4    224498
                    The Nice Guys    213845            7.4    213845
```