In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from surprise import SVD,Dataset,Reader,evaluate
```

In [2]:
```
df_credits = pd.read_csv('Data/tmdb_5000_credits.csv')
df_movies = pd.read_csv('Data/tmdb_5000_movies.csv')
```

In [3]:
```
# change the name of the 'movie_id'column to 'id' to merge with other df
df_credits.columns = ['id','title','cast','crew']
df = df_movies.merge(df_credits,on = 'id')
df.head()
```

Out[3]:

	budget	genres	homepage	id	keywords	original_
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	

5 rows × 23 columns

Ranking the Movies

Using IMDB's rating formula

```
Weighted Rating = [ (V / (V + M)) * R] +  [ (M / (V + M)) * R_mean]

V = no of votes
M = min votes required
R = average rating
R_mean = mean of average ratings of all movies
```

In [4]:
```
R_mean = df['vote_average'].mean()
M = df['vote_count'].mean() ## choosing this value to be the avg number of vot
es a movie received
print('R_mean :', R_mean)
print('M :', M)
```

```
R_mean : 6.092171559442011
M : 690.2179887570269
```

In [5]:
```
def imdb_rating(data_row,m = M, r_mean = R_mean ):
    v = data_row['vote_count']
    r = data_row['vote_average']
    return (v/(v+m) * r) + (m/(v+m) * r_mean)
```

In [6]:
```
# get movies that pass the minimum votes test i.e disregard the movies that re
ceived less than M votes because
# they received fewer votes

movies = df[df['vote_count'] >= M]
```

In [7]:
```
# calculate the weighted rating of each movie and append it to the data frame
movies['w_rating'] = movies.apply(imdb_rating,axis = 1)
```

```
C:\Users\Ammar\.conda\envs\neuralnets\lib\site-packages\ipykernel_launcher.p
y:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

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

In [8]:
```
movies.head(5)
```

Out[8]:

	budget	genres	homepage	id	keywords	original_
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	

5 rows × 24 columns

In [9]:
```
# Sort movies by attribute
def sort_by_attr(dataframe,attr):
    return dataframe.sort_values(attr,ascending=False)
```

Trending Now

In [10]:
```
sort_by_attr(movies,'popularity')[['id','original_title','w_rating','popularit
y']].head(10)
```

Out[10]:

	id	original_title	w_rating	popularity
546	211672	Minions	6.359616	875.581305
95	157336	Interstellar	7.980089	724.247784
788	293660	Deadpool	7.322750	514.569956
94	118340	Guardians of the Galaxy	7.780390	481.098624
127	76341	Mad Max: Fury Road	7.124422	434.278564
28	135397	Jurassic World	6.469901	418.708552
199	22	Pirates of the Caribbean: The Curse of the Bla...	7.373397	271.972889
82	119450	Dawn of the Planet of the Apes	7.136543	243.791743
200	131631	The Hunger Games: Mockingjay - Part 1	6.544135	206.227151
88	177572	Big Hero 6	7.627291	203.734590

Top Rated

In [11]:
```
sort_by_attr(movies,'w_rating')[['id','original_title','w_rating','popularity'
]].head(10)
```

Out[11]:

	id	original_title	w_rating	popularity
1881	278	The Shawshank Redemption	8.313166	136.747729
3337	238	The Godfather	8.158036	143.659698
662	550	Fight Club	8.149169	146.757391
3232	680	Pulp Fiction	8.132875	121.463076
65	155	The Dark Knight	8.085374	187.322927
809	13	Forrest Gump	8.031168	138.133331
96	27205	Inception	8.004042	167.583710
1818	424	Schindler's List	7.996390	104.469351
3865	244786	Whiplash	7.991785	192.528841
95	157336	Interstellar	7.980089	724.247784

Plot Description Based Recommendation

In [12]:
```
tfidf = TfidfVectorizer(stop_words='english')
df['overview'] = df['overview'].fillna('')
tfidf_matrix = tfidf.fit_transform(df['overview'])
tfidf_matrix.shape
```

Out[12]:
```
(4803, 20978)
```

In [13]:
```
cos_scores = cosine_similarity(tfidf_matrix) # no need to divide by magnitudes
because of normalized vectors
```

In [14]:
```
# create a mapping from movie titles to movie indices because scores are place
d with respect to indices
indices_movies = pd.Series(df.index,index = df['original_title'])
```

In [15]:
```
def get_recomm_plot(movie_title,top = 11,cos_scores=cos_scores,indices_movies
= indices_movies):

    y = cos_scores[indices_movies[movie_title]]
    indices = np.argsort(-y)
    return df['original_title'].iloc[indices[1:top]]
```

In [16]:
```
get_recomm_plot('The Dark Knight Rises')
```

Out[16]:
```
65                                   The Dark Knight
299                                   Batman Forever
428                                   Batman Returns
1359                                          Batman
3854    Batman: The Dark Knight Returns, Part 2
119                                    Batman Begins
2507                                       Slow Burn
9            Batman v Superman: Dawn of Justice
1181                                             JFK
210                                   Batman & Robin
Name: original_title, dtype: object
```

Metadata Based Recommendation

Metadata will be created from

- genres
- cast
- crew
- keywords

In [17]:
```
# convert genres, cast, crew, keywords to usable form from stringified lists
df['genres'] = df['genres'].apply(eval)
df['cast'] = df['cast'].apply(eval)
df['crew'] = df['crew'].apply(eval)
df['keywords'] = df['keywords'].apply(eval)
```

In [18]:
```
def get_list(x,num = 3):
    if isinstance(x,list):
        names = [i['name'] for i in x]
        if len(names) > num:
            names = names[:num]
        return names
    return []
```

In [19]:
```
def get_director(x):
    if isinstance(x,list):
        name = [i['name'] for i in x if i['job'] == 'Director']
        if len(name) == 0:
            return np.nan
        return name[0]
    return np.nan
```

In [20]:
```
df['director'] = df['crew'].apply(get_director)
```

In [21]:
```
df['genres'] = df['genres'].apply(get_list)
df['cast'] = df['cast'].apply(get_list)
df['keywords'] = df['keywords'].apply(get_list)
```

In [22]:
```
df[['cast','genres','keywords','director']].head(3)
```

Out[22]:

	cast	genres	keywords	director
0	[Sam Worthington, Zoe Saldana, Sigourney Weaver]	[Action, Adventure, Fantasy]	[culture clash, future, space war]	James Cameron
1	[Johnny Depp, Orlando Bloom, Keira Knightley]	[Adventure, Fantasy, Action]	[ocean, drug abuse, exotic island]	Gore Verbinski
2	[Daniel Craig, Christoph Waltz, Léa Seydoux]	[Action, Adventure, Crime]	[spy, based on novel, secret agent]	Sam Mendes

In [23]:
```
def clean_list(x):
    if isinstance(x,list):
        return [str.lower(i.replace(" ","")) for i in x]
```

In [24]:
```
def clean_director(x):
    if isinstance(x,str):
        return str.lower(x.replace(" ", ""))
    else :
        return ""
```

In [25]:
```
df['genres'] = df['genres'].apply(clean_list)
df['cast'] = df['cast'].apply(clean_list)
df['keywords'] = df['keywords'].apply(clean_list)
df['director'] = df['director'].apply(clean_director)
```

In [26]:
```
def create_metadata(x):
    return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['dire
ctor'] + ' ' + ' '.join(x['genres'])
```

In [27]:
```
df['metadata'] = df.apply(create_metadata,axis=1)
```

In [28]:
```
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df['metadata'])
count_matrix.shape
```

Out[28]:
```
(4803, 11520)
```

In [29]:
```
cos_scores_count = cosine_similarity(count_matrix)
indices_movies = pd.Series(df.index,index = df['original_title'])
```

In [30]:
```
get_recomm_plot('The Dark Knight Rises',cos_scores = cos_scores_count)
```

Out[30]:
```
119                Batman Begins
65               The Dark Knight
4638    Amidst the Devil's Wings
1196                The Prestige
3073           Romeo Is Bleeding
3326              Black November
1503                      Takers
1986                      Faster
2154                Street Kings
303                     Catwoman
Name: original_title, dtype: object
```

In [31]:
```
get_recomm_plot('The Godfather',cos_scores = cos_scores_count)
```

Out[31]:
```
867      The Godfather: Part III
2731      The Godfather: Part II
4638    Amidst the Devil's Wings
2649           The Son of No One
1525              Apocalypse Now
1209               The Rainmaker
2280                 Sea of Love
1394               Donnie Brasco
4209            The Conversation
4432           On the Waterfront
Name: original_title, dtype: object
```

Item based Collaborative Filtering

- We will use Singular Value Decomposition (SVD) to predict how a user will rate a movie that the user has not rated and can make recommendation based on that information.

In [32]:
```
df_ratings = pd.read_csv('Data/ratings_small.csv')
reader = Reader()
```

In [33]:
```
data = Dataset.load_from_df(df_ratings[['userId', 'movieId', 'rating']], reade
r)
data.split(n_folds=5)
```

In [34]:
```python
svd = SVD()
evaluate(svd, data, measures=['RMSE', 'MAE'])
```

```
C:\Users\Ammar\.conda\envs\neuralnets\lib\site-packages\surprise\evaluate.py:
66: UserWarning: The evaluate() method is deprecated. Please use model_select
ion.cross_validate() instead.
  'model_selection.cross_validate() instead.', UserWarning)
C:\Users\Ammar\.conda\envs\neuralnets\lib\site-packages\surprise\dataset.py:1
93: UserWarning: Using data.split() or using load_from_folds() without using
a CV iterator is now deprecated.
  UserWarning)

Evaluating RMSE, MAE of algorithm SVD.

------------
Fold 1
RMSE: 0.8866
MAE:  0.6839
------------
Fold 2
RMSE: 0.8907
MAE:  0.6854
------------
Fold 3
RMSE: 0.8972
MAE:  0.6917
------------
Fold 4
RMSE: 0.8998
MAE:  0.6923
------------
Fold 5
RMSE: 0.9027
MAE:  0.6930
------------
------------
Mean RMSE: 0.8954
Mean MAE : 0.6892
------------
------------
```

Out[34]:
```
CaseInsensitiveDefaultDict(list,
                           {'rmse': [0.8866470812766913,
                             0.8907292604210697,
                             0.8972371835860599,
                             0.8997685853684747,
                             0.902704894074013],
                            'mae': [0.6838521771541389,
                             0.6853723245869425,
                             0.6917116743210734,
                             0.6922775439280969,
                             0.6929555449688402]})
```

In [35]:
```python
trainset = data.build_full_trainset()
histpry = svd.fit(trainset)
```

In [36]:
```
df_ratings[df_ratings['userId'] == 1]
```

Out[36]:

	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
5	1	1263	2.0	1260759151
6	1	1287	2.0	1260759187
7	1	1293	2.0	1260759148
8	1	1339	3.5	1260759125
9	1	1343	2.0	1260759131
10	1	1371	2.5	1260759135
11	1	1405	1.0	1260759203
12	1	1953	4.0	1260759191
13	1	2105	4.0	1260759139
14	1	2150	3.0	1260759194
15	1	2193	2.0	1260759198
16	1	2294	2.0	1260759108
17	1	2455	2.5	1260759113
18	1	2968	1.0	1260759200
19	1	3671	3.0	1260759117

In [37]:
```
# in this way, we can predict how much a user will rate a movie based on how o
ther users have rated this movie
svd.predict(1,32)
```

Out[37]:
```
Prediction(uid=1, iid=32, r_ui=None, est=2.9338225551541943, details={'was_im
possible': False})
```