In [1]:
```
#Importing Libraries
```

In [2]:
```
%matplotlib inline
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.corpus import stopwords
from scipy import stats
from wordcloud import WordCloud,STOPWORDS
from ast import literal_eval
from sklearn.feature_extraction.text import TfidfVectorizer, CountVector
from sklearn.metrics.pairwise import linear_kernel, cosine_similarity
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
import warnings; warnings.simplefilter('ignore')
```

In [3]:
```
#Loading Data
```

In [4]:
```
input_data = pd. read_csv('movies_metadata.csv')
input_data.head()
```

Out[4]:

	adult	belongs_to_collection	budget	genres	homepage	id	imd
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	862	tt011
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt011
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35, ...	NaN	15602	tt011
3	False	NaN	16000000	[{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...	NaN	31357	tt011
4	False	{'id': 96871, 'name': 'Father of the Bride Col...	0	[{'id': 35, 'name': 'Comedy'}]	NaN	11862	tt011

5 rows × 24 columns

In [5]:
```
input_data.head().transpose()
```

Out[5]:

	0	1	
adult	False	False	
belongs_to_collection	{'id': 10194, 'name': 'Toy Story Collection', ...	NaN	
budget	30000000	65000000	
genres	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	[{'ic
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 ...	When siblings Judy and Peter discover an encha...	A f
popularity	21.9469	17.0155	
poster_path	/rhIRbceoE9lR4veEXuwCC2wARtG.jpg	/vzmL6fP7aPKNKPRTFnZmiUfciyV.jpg	/6ksı
production_companies	[{'name': 'Pixar Animation Studios', 'id': 3}]	[{'name': 'TriStar Pictures', 'id': 559}, {'na...	[
production_countries	[{'iso_3166_1': 'US', 'name': 'United States o...	[{'iso_3166_1': 'US', 'name': 'United States o...	[{
release_date	1995-10-30	1995-12-15	
revenue	3.73554e+08	2.62797e+08	
runtime	81	104	
spoken_languages	[{'iso_639_1': 'en', 'name': 'English'}]	[{'iso_639_1': 'en', 'name': 'English'}, {'iso...	[{'
status	Released	Released	
tagline	NaN	Roll the dice and unleash the excitement!	S
title	Toy Story	Jumanji	
video	False	False	
vote_average	7.7	6.9	
vote_count	5415	2413	

In [6]:
```
#Testing and understanding the dataset
```

In [7]:
```
input_data.columns
```

Out[7]:
```
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')
```

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

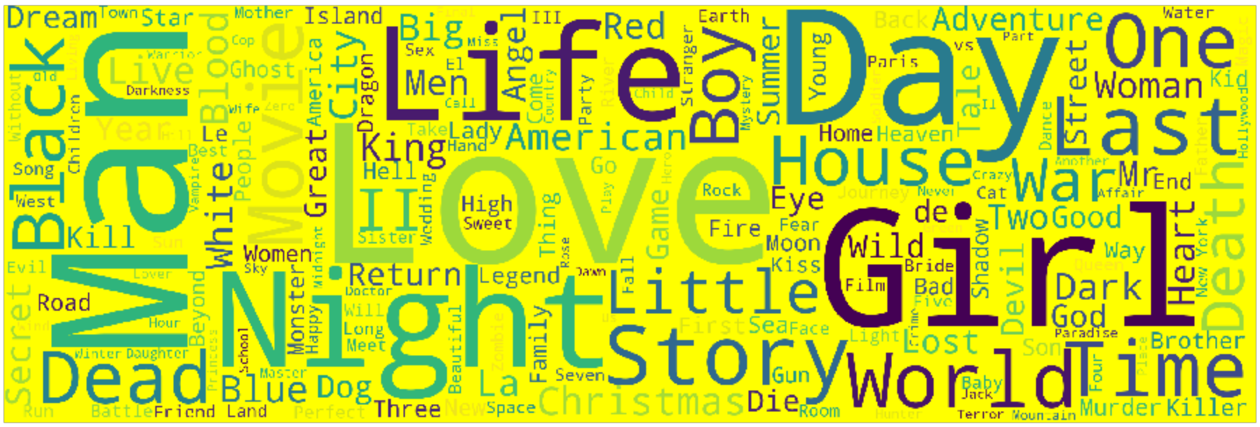
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45466 entries, 0 to 45465
Data columns (total 24 columns):
adult                    45466 non-null object
belongs_to_collection    4494 non-null object
budget                   45466 non-null object
genres                   45466 non-null object
homepage                 7782 non-null object
id                       45466 non-null object
imdb_id                  45449 non-null object
original_language        45455 non-null object
original_title           45466 non-null object
overview                 44512 non-null object
popularity               45461 non-null object
poster_path              45080 non-null object
production_companies     45463 non-null object
production_countries     45463 non-null object
release_date             45379 non-null object
revenue                  45460 non-null float64
runtime                  45203 non-null float64
spoken_languages         45460 non-null object
status                   45379 non-null object
tagline                  20412 non-null object
title                    45460 non-null object
video                    45460 non-null object
vote_average             45460 non-null float64
vote_count               45460 non-null float64
dtypes: float64(4), object(20)
memory usage: 8.3+ MB
```

In [9]:
```
#Analysing data and building a wordCloud
```

In [10]:
```
input_data['title'] = input_data['title'].astype('str')
input_data['overview'] = input_data['overview'].astype('str')
```

In [11]:
```
movie_title = ' '.join(input_data['title'])
```

In [12]:
```
movieTitleWC = WordCloud(stopwords=STOPWORDS, background_color='Yellow',
plt.figure(figsize=(16,8))
plt.imshow(movieTitleWC)
plt.axis('off')
plt.show()
```



In [13]:
```
#Implementation of Simple Recommendation System
```

In [14]:
```
_data['genres'].fillna('[]').apply(literal_eval).apply(lambda x: [i['name
```

In [15]:
```
no_of_votes = input_data[input_data['vote_count'].notnull()]['vote_count
avgVotes = input_data[input_data['vote_average'].notnull()]['vote_averag
meanVotes = avgVotes.mean()
meanVotes
```

Out[15]: 5.244896612406511

In [16]:
```
minVotes = no_of_votes.quantile(0.95)
minVotes
```

Out[16]: 434.0

In [17]:
```
_datetime(input_data['release_date'], errors='coerce').apply(lambda x: st
```

In [18]:
```
movies_qualified = input_data[(input_data['vote_count'] >= minVotes) & (
movies_qualified['vote_count'] = movies_qualified['vote_count'].astype('
movies_qualified['vote_average'] = movies_qualified['vote_average'].asty
movies_qualified.shape
```

Out[18]: (2274, 6)

In [19]:
```
def weighted_rating(a):
    noOfVotes = a['vote_count']
    avgRating = a['vote_average']
    return (noOfVotes/(noOfVotes+minVotes) * avgRating) + (minVotes/(min
```

In [20]:
```
movies_qualified['wr'] = movies_qualified.apply(weighted_rating, axis=1)
```

In [21]:
```
movies_qualified = movies_qualified.sort_values('wr', ascending=False).h
```

In [22]:
```
movies_qualified.head(20)
```

Out[22]:

	title	year	vote_count	vote_average	popularity	genres	wr
15480	Inception	2010	14075	8	29.1081	[Action, Thriller, Science Fiction, Mystery, A...	7.917588
12481	The Dark Knight	2008	12269	8	123.167	[Drama, Action, Crime, Thriller]	7.905871
22879	Interstellar	2014	11187	8	32.2135	[Adventure, Drama, Science Fiction]	7.897107
2843	Fight Club	1999	9678	8	63.8696	[Drama]	7.881753
4863	The Lord of the Rings: The Fellowship of the Ring	2001	8892	8	32.0707	[Adventure, Fantasy, Action]	7.871787
292	Pulp Fiction	1994	8670	8	140.95	[Thriller, Crime]	7.868660
314	The Shawshank Redemption	1994	8358	8	51.6454	[Drama, Crime]	7.864000
7000	The Lord of the Rings: The Return of the King	2003	8226	8	29.3244	[Adventure, Fantasy, Action]	7.861927
351	Forrest Gump	1994	8147	8	48.3072	[Comedy, Drama,	7.860656

						Romance]	
5814	The Lord of the Rings: The Two Towers	2002	7641	8	29.4235	[Adventure, Fantasy, Action]	7.851924
256	Star Wars	1977	6778	8	42.1497	[Adventure, Action, Science Fiction]	7.834205
1225	Back to the Future	1985	6239	8	25.7785	[Adventure, Comedy, Science Fiction, Family]	7.820813
834	The Godfather	1972	6024	8	41.1093	[Drama, Crime]	7.814847
1154	The Empire Strikes Back	1980	5998	8	19.471	[Adventure, Action, Science Fiction]	7.814099
46	Se7en	1995	5915	8	18.4574	[Crime, Mystery, Thriller]	7.811669
24860	The Imitation Game	2014	5895	8	31.5959	[History, Drama, Thriller, War]	7.811074
359	The Lion King	1994	5520	8	21.6058	[Family, Animation, Drama]	7.799175
18465	The Intouchables	2011	5410	8	16.0869	[Drama, Comedy]	7.795394
22841	The Grand Budapest Hotel	2014	4644	8	14.442	[Comedy, Drama]	7.764530
586	The Silence of the Lambs	1991	4549	8	4.30722	[Crime, Drama, Thriller]	7.760041

In [24]:
```
a = input_data.apply(lambda z: pd.Series(z['genres']),axis=1).stack().re
a.name = 'genre'
gen_data = input_data.drop('genres', axis=1).join(a)
```

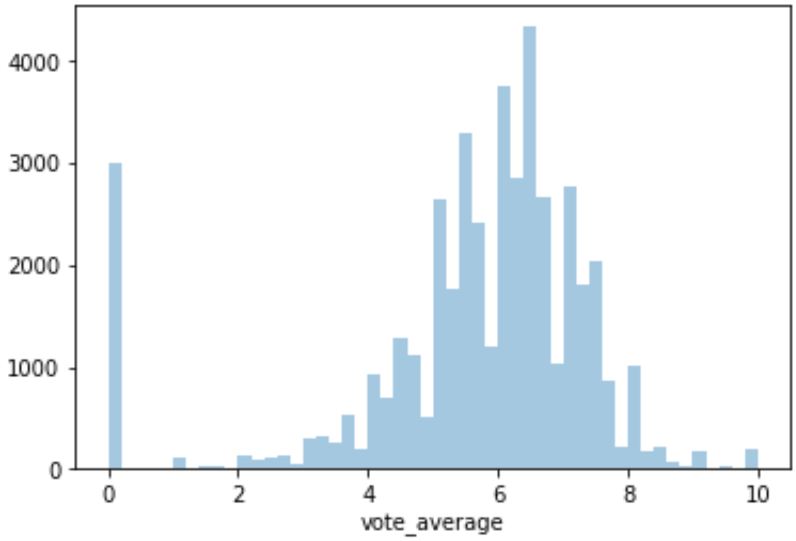
In [26]:
```
def buildChart(genre, percentile=0.86):
    dataframe1 = gen_data[gen_data['genre'] == genre]
    no_of_votes = input_data[input_data['vote_count'].notnull()]['vote_c
    avgVotes = input_data[input_data['vote_average'].notnull()]['vote_av
    meanVotes = avgVotes.mean()
    minVotes = no_of_votes.quantile(percentile)

    movies_qualified = input_data[(input_data['vote_count'] >= minVotes)
    movies_qualified['vote_count'] = movies_qualified['vote_count'].asty
    movies_qualified['vote_average'] = movies_qualified['vote_average'].

    movies_qualified['wr'] = movies_qualified.apply(lambda z: (z['vote_c
    movies_qualified = movies_qualified.sort_values('wr', ascending=Fals
    return movies_qualified
```

In [27]:
```
buildChart('Romance').head(20)
```

Out[27]:

	title	year	vote_count	vote_average	popularity	wr
10309	Dilwale Dulhania Le Jayenge	1995	661	9	34.457	8.545593
15480	Inception	2010	14075	8	29.1081	7.982302
12481	The Dark Knight	2008	12269	8	123.167	7.979716
22879	Interstellar	2014	11187	8	32.2135	7.977770
2843	Fight Club	1999	9678	8	63.8696	7.974336
4863	The Lord of the Rings: The Fellowship of the Ring	2001	8892	8	32.0707	7.972090
292	Pulp Fiction	1994	8670	8	140.95	7.971383
314	The Shawshank Redemption	1994	8358	8	51.6454	7.970326
7000	The Lord of the Rings: The Return of the King	2003	8226	8	29.3244	7.969855
351	Forrest Gump	1994	8147	8	48.3072	7.969566
5814	The Lord of the Rings: The Two Towers	2002	7641	8	29.4235	7.967574
256	Star Wars	1977	6778	8	42.1497	7.963501
1225	Back to the Future	1985	6239	8	25.7785	7.960393
834	The Godfather	1972	6024	8	41.1093	7.959000
1154	The Empire Strikes Back	1980	5998	8	19.471	7.958825
46	Se7en	1995	5915	8	18.4574	7.958256
24860	The Imitation Game	2014	5895	8	31.5959	7.958117
359	The Lion King	1994	5520	8	21.6058	7.955317
18465	The Intouchables	2011	5410	8	16.0869	7.954424
22841	The Grand Budapest Hotel	2014	4644	8	14.442	7.947051

In [31]:
```
sns.distplot(input_data['vote_average'],kde=False,bins=50)
```
Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1a25d100d0>



In [32]:
```
#Content Based Recommender
```

In [33]:
```
linksSmallDataSet = pd.read_csv('links_small.csv')
linksSmallDataSet = linksSmallDataSet[linksSmallDataSet['tmdbId'].notnul
```

In [34]:
```
input_data = input_data.drop([19730, 29503, 35587])
```

In [35]:
```
input_data['id'] = input_data['id'].astype('int')
```

In [36]:
```
smallMovieDataset = input_data[input_data['id'].isin(linksSmallDataSet)]
smallMovieDataset.shape
```
Out[36]: (9099, 25)

In [37]:
```
#Description based recommender
```

In [38]:
```
smallMovieDataset['tagline'] = smallMovieDataset['tagline'].fillna('')
smallMovieDataset['description'] = smallMovieDataset['overview'] + small
smallMovieDataset['description'] = smallMovieDataset['description'].fill
```

In [39]:
```
termfrequency = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_d
tfIdfMatrix = termfrequency.fit_transform(smallMovieDataset['description
```

In [40]:
```
tfIdfMatrix.shape
```

Out[40]:
```
(9099, 268124)
```

In [41]:
```
cosineSimilarity = linear_kernel(tfIdfMatrix, tfIdfMatrix)
```

In [42]:
```
cosineSimilarity[0]
```

Out[42]:
```
array([1.        , 0.00680476, 0.        , ..., 0.        , 0.00344913
,
       0.        ])
```

In [43]:
```
smallMovieDataset = smallMovieDataset.reset_index()
movieTitles = smallMovieDataset['title']
movieIndices = pd.Series(smallMovieDataset.index, index=smallMovieDatase
```

In [44]:
```
def getRecommendation(name):
    indx = movieIndices[name]
    similarityScores = list(enumerate(cosineSimilarity[indx]))
    similarityScores = sorted(similarityScores, key=lambda z: z[1], reve
    similarityScores = similarityScores[1:31]
    movie_Inds = [o[0] for o in similarityScores]
    return movieTitles.iloc[movie_Inds]
```

In [45]:
```
getRecommendation('Shanghai Triad').head(20)
```

Out[45]:
```
6773                                  Lust, Caution
8209                                      Stowaway
692                                   The Godfather
3274                              Empire of the Sun
8387                                     The Family
7416                                   Soul Kitchen
7571                                       Ip Man 2
3669                                    Billy Liar
2072                                           Life
4196                             Johnny Dangerously
985                                       The Sting
1681                              The Addams Family
8527                                Survival Island
973                         The Godfather: Part II
6528                                   Stormbreaker
8753                                   Wicked Blood
2761                 National Lampoon's Last Resort
2644    Black Tar Heroin: The Dark End of the Street
3101                                Steal This Movie
9065                                 The Handmaiden
Name: title, dtype: object
```

In [46]:
```
getRecommendation('Batman Forever').head(20)
```

Out[46]:
```
7931                       The Dark Knight Rises
2579                Batman: Mask of the Phantasm
6900                             The Dark Knight
6144                               Batman Begins
8165    Batman: The Dark Knight Returns, Part 1
524                                       Batman
1240                               Batman & Robin
1113                              Batman Returns
7565                  Batman: Under the Red Hood
7901                           Batman: Year One
8227    Batman: The Dark Knight Returns, Part 2
681                          Eyes Without a Face
6206                                    Cry_Wolf
1135                     Night Falls on Manhattan
2075                               Open Your Eyes
149                                      Hackers
8917         Batman v Superman: Dawn of Justice
2696                                          JFK
8680                          The Young Savages
7242                   The File on Thelma Jordon
Name: title, dtype: object
```

In [47]:
```
#Metadata Based recommender
```

In [48]:
```
creditsData = pd.read_csv('credits.csv')
keywordsData = pd.read_csv('keywords.csv')
```

In [49]:
```
keywordsData['id'] = keywordsData['id'].astype('int')
creditsData['id'] = creditsData['id'].astype('int')
input_data['id'] = input_data['id'].astype('int')
```

In [50]:
```
input_data.shape
```

Out[50]:
```
(45463, 25)
```

In [51]:
```
input_data = input_data.merge(creditsData, on='id')
input_data = input_data.merge(keywordsData, on='id')
```

In [52]:
```
smallMovieDataset = input_data[input_data['id'].isin(linksSmallDataSet)]
smallMovieDataset.shape
```

Out[52]:
```
(9219, 28)
```

In [53]:
```
smallMovieDataset['cast'] = smallMovieDataset['cast'].apply(literal_eval
smallMovieDataset['crew'] = smallMovieDataset['crew'].apply(literal_eval
smallMovieDataset['keywords'] = smallMovieDataset['keywords'].apply(lite
smallMovieDataset['cast_size'] = smallMovieDataset['cast'].apply(lambda
smallMovieDataset['crew_size'] = smallMovieDataset['crew'].apply(lambda
```

In [54]:
```
def getDirector(z):
    for a in z:
        if a['job'] == 'Director':
            return a['name']
    return np.nan
```

In [55]:
```
smallMovieDataset['director'] = smallMovieDataset['crew'].apply(getDirec
```

In [56]:
```
smallMovieDataset['cast'] = smallMovieDataset['cast'].apply(lambda z: [a
smallMovieDataset['cast'] = smallMovieDataset['cast'].apply(lambda z: z[
```

In [57]:
```
aset['keywords'] = smallMovieDataset['keywords'].apply(lambda z: [a['name
```

In [58]:
```
smallMovieDataset['cast'] = smallMovieDataset['cast'].apply(lambda z: [s
```

In [59]:
```
smallMovieDataset['director'] = smallMovieDataset['director'].astype('st
smallMovieDataset['director'] = smallMovieDataset['director'].apply(lamb
```

In [60]:
```
abc = smallMovieDataset.apply(lambda z: pd.Series(z['keywords']),axis=1)
abc.name = 'keyword'
```

In [61]:
```
abc = abc.value_counts()
abc[:5]
```

Out[61]:
```
independent film        610
woman director          550
murder                  399
duringcreditsstinger    327
based on novel          318
Name: keyword, dtype: int64
```

In [62]:
```
abc = abc[abc > 1]
```

In [66]:
```
stemming = SnowballStemmer('english')
stemming.stem('Big Datas')
```

Out[66]:
```
'big data'
```

In [67]:
```
def filterKeywords(a):
    words = []
    for p in a:
        if p in abc:
            words.append(p)
    return words
```

In [68]:
```
allMovieDataset['keywords'] = smallMovieDataset['keywords'].apply(filterK
allMovieDataset['keywords'] = smallMovieDataset['keywords'].apply(lambda
allMovieDataset['keywords'] = smallMovieDataset['keywords'].apply(lambda
```

In [69]:
```
smallMovieDataset['soup'] = smallMovieDataset['keywords'] + smallMovieDa
smallMovieDataset['soup'] = smallMovieDataset['soup'].apply(lambda z: '
```

In [70]:
```
num = CountVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_
numMatrix = num.fit_transform(smallMovieDataset['soup'])
```

In [71]:
```
cosineSimilarity = cosine_similarity(numMatrix, numMatrix)
```

In [72]:
```
smallMovieDataset = smallMovieDataset.reset_index()
movieTitles = smallMovieDataset['title']
movieIndices = pd.Series(smallMovieDataset.index, index=smallMovieDataset
```

In [73]:
```
getRecommendation('Inception').head(5)
```

Out[73]:
```
6623             The Prestige
3381                  Memento
4145                 Insomnia
2085                Following
8031    The Dark Knight Rises
Name: title, dtype: object
```

In [74]:
```
f impRecommendations(name):
    indx = movieIndices[name]
    similarityScores = list(enumerate(cosineSimilarity[indx]))
    similarityScores = sorted(similarityScores, key=lambda z: z[1], reverse
    similarityScores = similarityScores[1:26]
    movie_Inds = [p[0] for p in similarityScores]

    movieNames = input_data.iloc[movie_Inds][['title', 'vote_count', 'vote_
    no_of_votes = movieNames[movieNames['vote_count'].notnull()]['vote_coun
    avgVotes = movieNames[movieNames['vote_average'].notnull()]['vote_avera
    meanVotes = avgVotes.mean()
    minVotes = no_of_votes.quantile(0.60)
    movies_qualified = movieNames[(movieNames['vote_count'] >= minVotes) &
    movies_qualified['vote_count'] = movies_qualified['vote_count'].astype(
    movies_qualified['vote_average'] = movies_qualified['vote_average'].ast
    movies_qualified['wr'] = movies_qualified.apply(weighted_rating, axis=1
    movies_qualified = movies_qualified.sort_values('wr', ascending=False).
    return movies_qualified
```

In [75]:
```
impRecommendations('Batman Forever')
```

Out[75]:

	title	vote_count	vote_average	year	wr
2523	The Mummy	2796	6	1999	5.898540
315	Shallow Grave	247	7	1994	5.881476
1260	The Big Sleep	244	7	1946	5.876527
3174	Hard Boiled	169	7	1992	5.736791
6981	Beauty and the Beast	133	7	1946	5.656588
6703	The Rules of the Game	109	7	1939	5.597210
524	Searching for Bobby Fischer	100	7	1993	5.573568
1721	Suicide Kings	86	6	1997	5.369779
2442	Earthquake	76	5	1974	5.208402
6218	Anger Management	937	5	2003	5.077524

In [76]:
```
#Collaborative Filtering
```

In [77]:
```
read = Reader()
```

In [78]:
```
movieRatings = pd.read_csv('ratings_small.csv')
movieRatings.head()
```

Out[78]:

	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

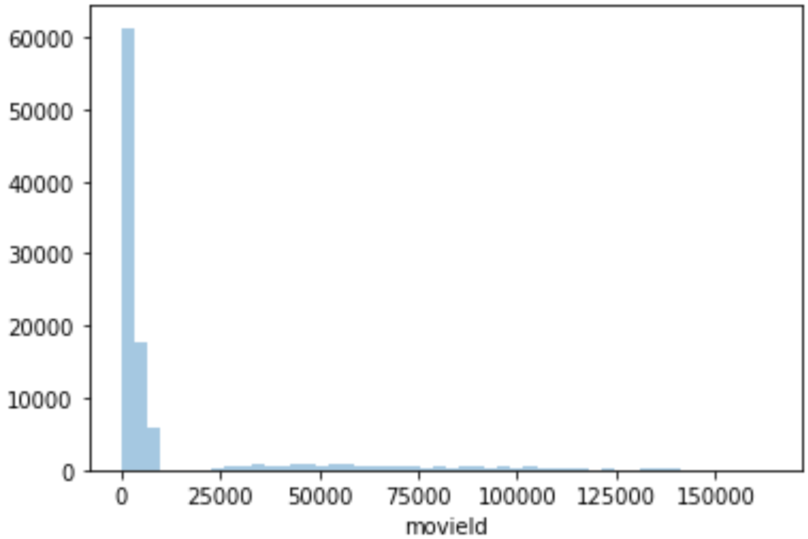
In [79]:
```
inputData = Dataset.load_from_df(movieRatings[['userId', 'movieId', 'rat
```

In [80]:
```
singleVD = SVD()
cross_validate(singleVD, inputData, measures=['RMSE', 'MAE'])
```

Out[80]:
```
{'test_rmse': array([0.89707477, 0.88324853, 0.90921328, 0.89043196, 0
.90354828]),
 'test_mae': array([0.69212425, 0.68049989, 0.69980372, 0.68921543, 0.
69242386]),
 'fit_time': (5.8468077182769775,
  6.20166802406311,
  5.724970817565918,
  5.793152093887329,
  5.855086088180542),
 'test_time': (0.18041586875915527,
  0.37017107009887695,
  0.1652078628540039,
  0.18491697311401367,
  0.20931696891784668)}
```

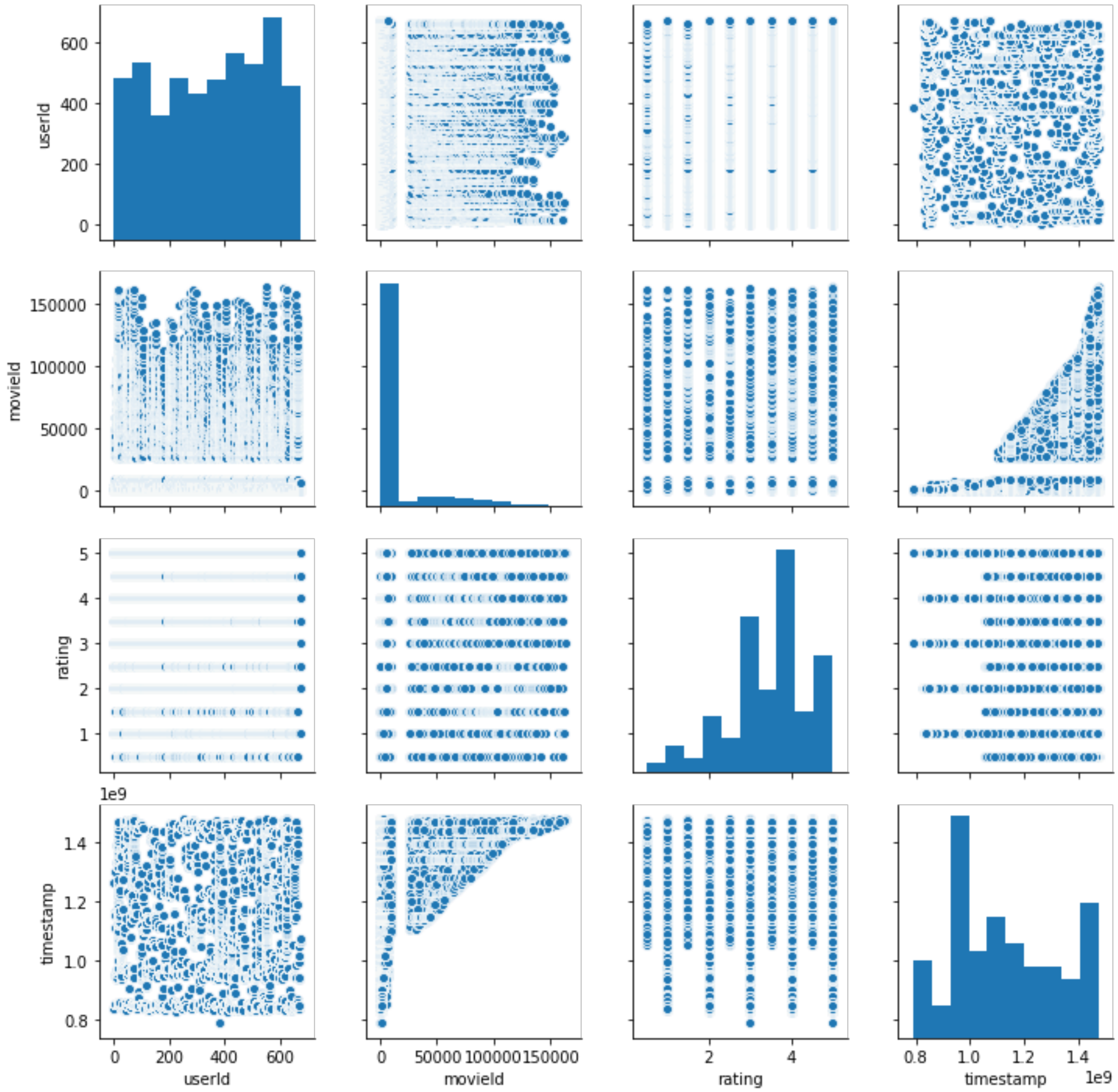
In [81]:
```
sns.distplot(movieRatings['movieId'],kde=False,bins=50)
```

Out[81]: <matplotlib.axes._subplots.AxesSubplot at 0x1a6d247410>



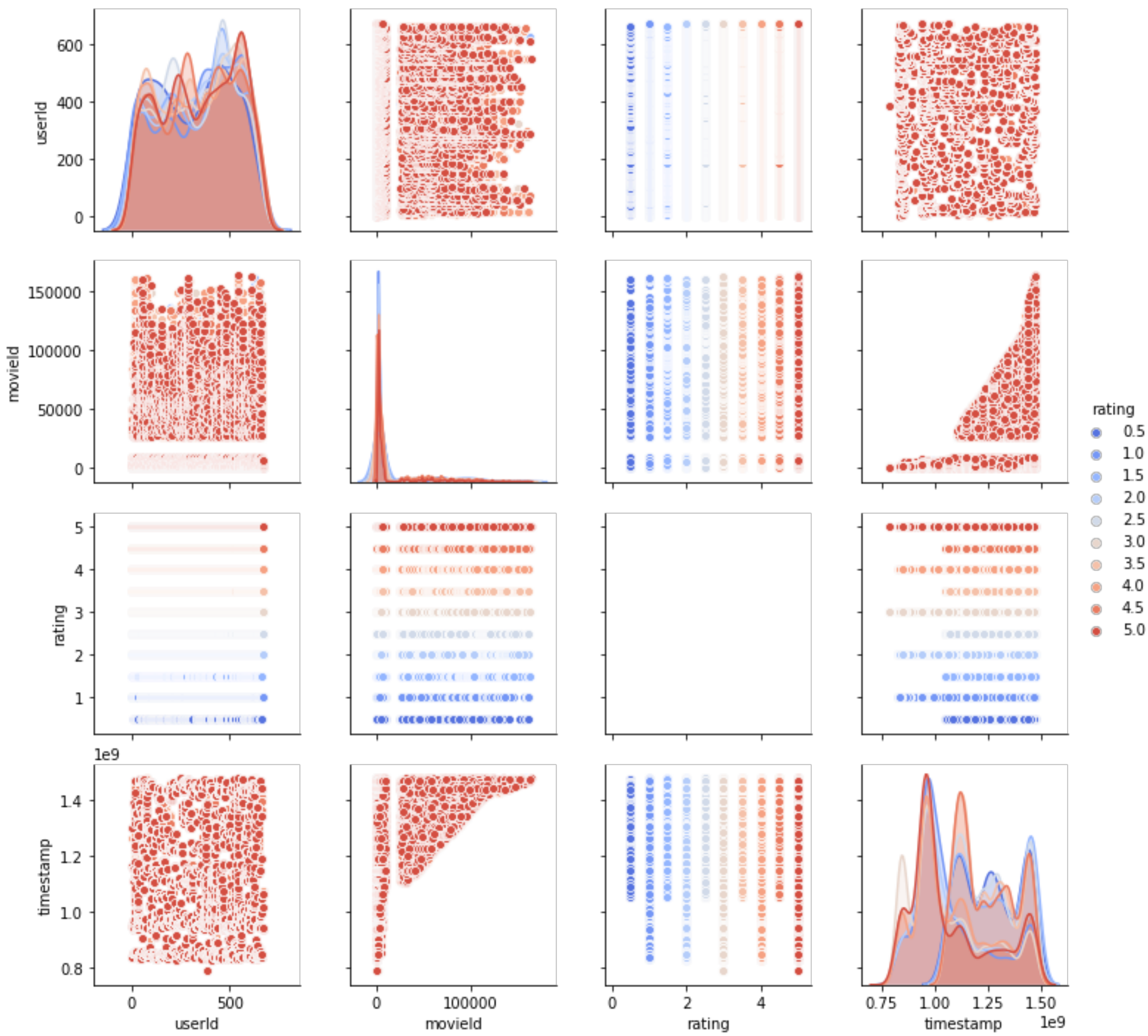
In [82]:
```
sns.pairplot(movieRatings)
```

Out[82]: <seaborn.axisgrid.PairGrid at 0x1a6d29f790>



In [83]:
```
sns.pairplot(movieRatings,hue='rating',palette='coolwarm')
```

Out[83]: <seaborn.axisgrid.PairGrid at 0x1a6d881550>



In [84]:
```
trainsetData = inputData.build_full_trainset()
singleVD.fit(trainsetData)
```

Out[84]: <surprise.prediction_algorithms.matrix_factorization.SVD at 0x1a6da67b10>

In [85]:
```
movieRatings[movieRatings['userId'] == 1]
```

Out[85]:

	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 [86]:
```
singleVD.predict(1, 300, 5)
```

Out[86]:
```
Prediction(uid=1, iid=300, r_ui=5, est=3.0034711864442394, details={'w
as_impossible': False})
```

In [87]:
```
#Hybrid
```

In [88]:
```
def convertInt(a):
    try:
        return int(a)
    except:
        return np.nan
```

In [89]:
```
idMap = pd.read_csv('links_small.csv')[['movieId', 'tmdbId']]
idMap['tmdbId'] = idMap['tmdbId'].apply(convertInt)
idMap.columns = ['movieId', 'id']
idMap = idMap.merge(smallMovieDataset[['title', 'id']], on='id').set_ind
```

In [90]:
```
indicesMap = idMap.set_index('id')
```

In [91]:
```
def hybridRecommendation(usernum, name):
    indx = movieIndices[name]
    tdbId = idMap.loc[name]['id']
    movieId = idMap.loc[name]['movieId']

    similarityScores = list(enumerate(cosineSimilarity[int(indx)]))
    similarityScores = sorted(similarityScores, key=lambda d: d[1], reve
    similarityScores = similarityScores[1:26]
    movie_Inds = [h[0] for h in similarityScores]

    movieNames = smallMovieDataset.iloc[movie_Inds][['title', 'vote_coun
    movieNames['est'] = movieNames['id'].apply(lambda w: singleVD.predic
    movieNames = movieNames.sort_values('est', ascending=False)
    return movieNames.head(10)
```

In [92]:
```
hybridRecommendation(1, 'The Terminator')
```

Out[92]:

	title	vote_count	vote_average	year	id	est
7488	Avatar	12114.0	7.2	2009	19995	3.018819
974	Aliens	3282.0	7.7	1986	679	2.981203
522	Terminator 2: Judgment Day	4274.0	7.7	1991	280	2.954099
6967	Doomsday	374.0	5.8	2008	13460	2.927908
6394	District B13	572.0	6.5	2004	10045	2.819424
1376	Titanic	7770.0	7.5	1997	597	2.747625
7403	Gamer	778.0	5.6	2009	18501	2.706491
7502	The Book of Eli	2207.0	6.6	2010	20504	2.694188
5296	Zardoz	106.0	5.8	1974	4923	2.683079
7991	In Time	3512.0	6.7	2011	49530	2.658243

In [93]:
```
hybridRecommendation(100, 'The Terminator')
```

Out[93]:

	title	vote_count	vote_average	year	id	est
974	Aliens	3282.0	7.7	1986	679	3.899735
7991	In Time	3512.0	6.7	2011	49530	3.727733
7502	The Book of Eli	2207.0	6.6	2010	20504	3.707291
522	Terminator 2: Judgment Day	4274.0	7.7	1991	280	3.657294
922	The Abyss	822.0	7.1	1989	2756	3.618038
6622	Children of Men	2120.0	7.4	2006	9693	3.599097
6394	District B13	572.0	6.5	2004	10045	3.511455
7296	Terminator Salvation	2496.0	5.9	2009	534	3.448091
344	True Lies	1138.0	6.8	1994	36955	3.436736
2412	RoboCop	1494.0	7.1	1987	5548	3.411205

In [94]:
```
#End
```