In [18]:
```
#To increase the accuracy of recommendations, we'll use more featur
es in addition to the movie plot.
```

In [19]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

In [20]:
```
movies=pd.read_csv("IMDBdata_MainData.csv")
```

In [21]:
```
#Selecting fatures like director, plot and genre
movies=movies[['Title','Plot','Director','Genre']]
```

In [22]:
```
#Function converts string to lower case and removes spaces from nam
es.
#For Eg the output for Dwight Schrute will be dwightschrute
def text_preprocessing(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        #Check if director exists. If not, return empty string
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''
```

In [23]:
```
movies['Director'] = movies['Director'].apply(text_preprocessing)
```

In [24]:
```
#Get all the features together
movies['features']=movies['Title']+" "+movies['Plot']+" "+movies['D
irector']+" "+movies['Genre']
```

In [26]:
```
movies['features'].fillna(" ",inplace=True)

'''
The CountVectorizer provides a simple way to both tokenize a collec
tion of text documents and build a vocabulary of
known words, but also to encode new documents using that vocabulary
.An encoded vector is returned with a length of
the entire vocabulary and an integer count for the number of times
each word appeared in the document.
Term Frequency – Inverse Document" Frequency which are the componen
ts of the resulting scores assigned to each word.

Term Frequency: This summarizes how often a given word appears with
in a document.
Inverse Document Frequency: This downscales words that appear a lot
across documents.
Without going into the math, TF-IDF are word frequency scores that
try to highlight words that are more interesting,
e.g. frequent in a document but not across documents.
source:https://machinelearningmastery.com/prepare-text-data-machine
-learning-scikit-learn/


The difference from the previous Content based recommender is using
the CountVectorizer() instead of TF-IDF.
This is not to down-weight the presence of an director if he or she
has directed in relatively more movies.
'''
cv = CountVectorizer(stop_words='english')
matrix = cv.fit_transform(movies.features)
```

In [27]:
```
cosineSimilarity = cosine_similarity(matrix,matrix)
```

In [28]:
```
movies = movies.reset_index()
AllIndex = pd.Series(movies.index, index=movies['Title'])
```

In [29]:
```
def recommendations(title,cs=cosineSimilarity):
    """
    :param: title - title of the movie to find similar movies
    :param: cs - Cosine similarty matrix
    """
    index=AllIndex[title]

    #Get all similarity scores for the given movie
    scores = list(enumerate(cs[index]))
    scores = sorted(scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    scores = scores[1:11]

    # Get the movie indices for top 10
    movie_indices = [i[0] for i in scores]

    return movies['Title'].iloc[movie_indices]
```

In [30]:
```
recommendations('The Dark Knight Rises')
```

Out[30]:
```
103                                 The Dark Knight
1960                                         Batman
1753                                A Knight's Tale
4350     Batman: The Dark Knight Returns, Part 2
1218                                      The Siege
42                               Alone in the Dark
315                                        Sin City
405                                          Baasha
2350                                      Dark City
2663                                  The Conjuring
Name: Title, dtype: object
```

In [ ]: