In [1]:
```
import pandas as pd
```

In [6]:
```
cd /Users/jyothi/Downloads/ml-100k/
```

```
/Users/jyothi/Downloads/ml-100k
```

In [7]:
```
!head u.data
!echo # line break
!wc -l u.data
```

```
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488
253	465	5	891628467
305	451	3	886324817
6	86	3	883603013

  100000 u.data
```

In [10]:
```
names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('u.data', sep='\t', names=names)
df.head()
```

Out[10]:

	user_id	item_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

In [8]:
```

```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call l
ast)
<ipython-input-8-501a26f430be> in <module>()
----> 1 df.columns = ['UserID','ItemId','Rating','Timestamp']

NameError: name 'df' is not defined
```

In [8]:
```
df.head()
```

Out[8]:

	UserID	ItemId	Rating	Timestamp
0	186	302	3	891717742
1	22	377	1	878887116
2	244	51	2	880606923
3	166	346	1	886397596
4	298	474	4	884182806

In [11]:
```
df.shape
```

Out[11]:
```
(100000, 4)
```

In [12]:
```
import matplotlib.pyplot as plt
```

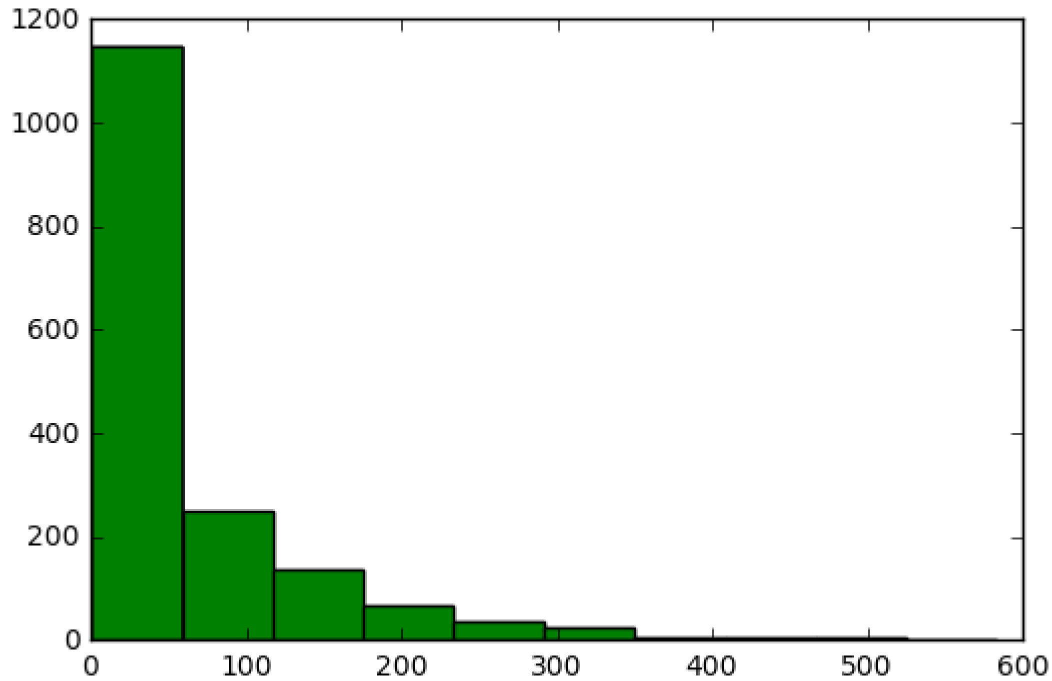
Data exploration

In this section, we will explore the MovieLens dataset and also prepare the data required for building collaborative filtering recommendation engines using python.

In [ ]:
```
plt.hist(df['Rating'])
plt.show()
```

In [ ]:
```

```

In [20]:
```
plt.hist( df.groupby(['ItemId'])['ItemId'].count())
plt.show()
```



In [15]:
```
n_users = df.user_id.unique().shape[0]
```

In [21]:
```
n_users = df.user_id.unique().shape[0]
n_items = df.item_id.unique().shape[0]
print (str(n_users) + ' users')
print (str(n_items) + ' items')
```

```
943 users
1682 items
```

In [22]:
```
print(str(n_users) + ' users')
```

```
943 users
```

In [23]:
```
print(str(n_items) + ' movies')
```

```
1682 movies
```

In [24]:
```
import numpy as np
ratings = np.zeros((n_users,n_items))
for row in df.itertuples():
    ratings[row[1]-1, row[ 2]-1] = row[3]
```

In [25]:
```
ratings.shape
```

Out[25]:
```
(943, 1682)
```

In [26]:
```
ratings[0][0]
```

Out[26]: 5.0

In [27]:
```
sparsity = float(len(ratings.nonzero()[0]))
```

In [28]:
```
sparsity /= (ratings.shape[0] * ratings.shape[1])
```

In [ ]:
```
We observe that the sparsity is 6.3% that is to say that we only have ra
ting information for 6.3% of the data and for the others it is just zero
s. Also please note that, the 0 value we see in the rating matrix
doesn't represent the rating given by the user, it just means that they
 are empty.
```

In [29]:
```
sparsity *= 100
print('Sparsity: {: 4.2f}%'.format( sparsity))
```

```
Sparsity:  6.30%
```

In [30]:
```
from sklearn.model_selection import train_test_split
```

In [32]:
```
def train_test_split(ratings):
    test = np.zeros(ratings.shape)
    train = ratings.copy()
    for user in range(ratings.shape[0]):
        test_ratings = np.random.choice(ratings[user, :].nonzero()[0],
                                        size=10,
                                        replace=False)
        train[user, test_ratings] = 0.
        test[user, test_ratings] = ratings[user, test_ratings]

    # Test and training are truly disjoint
    assert(np.all((train * test) == 0))
    return train, test
```

In [33]:
```
train, test = train_test_split(ratings)
```

In [34]:
```
def fast_similarity(ratings, kind='user', epsilon=1e-9):
    # epsilon -> small number for handling dived-by-zero errors
    if kind == 'user':
        sim = ratings.dot(ratings.T) + epsilon
    elif kind == 'item':
        sim = ratings.T.dot(ratings) + epsilon
    norms = np.array([np.sqrt(np.diagonal(sim))])
    return (sim / norms / norms.T)
```

In [36]:
```
user_similarity = fast_similarity(train, kind='user')
item_similarity = fast_similarity(train, kind='item')
print (item_similarity[:4, :4])
```

```
[[ 1.          0.40274949  0.32416998  0.43841137]
 [ 0.40274949  1.          0.25923761  0.49357849]
 [ 0.32416998  0.25923761  1.          0.33126502]
 [ 0.43841137  0.49357849  0.33126502  1.        ]]
```

In [38]:
```
def predict_fast_simple(ratings, similarity, kind='user'):
    if kind == 'user':
        return similarity.dot(ratings) / np.array([np.abs(similarity).su
m(axis=1)]).T
    elif kind == 'item':
        return ratings.dot(similarity) / np.array([np.abs(similarity).su
m(axis=1)])
```

In [39]:
```
from sklearn.metrics import mean_squared_error

def get_mse(pred, actual):
    # Ignore nonzero terms.
    pred = pred[actual.nonzero()].flatten()
    actual = actual[actual.nonzero()].flatten()
    return mean_squared_error(pred, actual)
```

In [41]:
```
item_prediction = predict_fast_simple(train, item_similarity, kind='ite
m')
user_prediction = predict_fast_simple(train, user_similarity, kind='use
r')

print( 'User-based CF MSE: ' + str(get_mse(user_prediction, test)))
print ('Item-based CF MSE: ' + str(get_mse(item_prediction, test)))
```

```
User-based CF MSE: 8.43892647874
Item-based CF MSE: 11.5001252374
```

In [42]:
```
def predict_topk(ratings, similarity, kind='user', k=40):
    pred = np.zeros(ratings.shape)
    if kind == 'user':
        for i in range(ratings.shape[0]):
            top_k_users = [np.argsort(similarity[:,i])[:-k-1:-1]]
            for j in range(ratings.shape[1]):
                pred[i, j] = similarity[i, :]
[top_k_users].dot(ratings[:, j][top_k_users])
                pred[i, j] /= np.sum(np.abs(similarity[i, :]
[top_k_users]))
    if kind == 'item':
        for j in range(ratings.shape[1]):
            top_k_items = [np.argsort(similarity[:,j])[:-k-1:-1]]
            for i in range(ratings.shape[0]):
                pred[i, j] = similarity[j, :]
[top_k_items].dot(ratings[i, :][top_k_items].T)
                pred[i, j] /= np.sum(np.abs(similarity[j, :]
[top_k_items]))

    return pred
```

In [44]:
```
pred = predict_topk(train, user_similarity, kind='user', k=40)
print ('Top-k User-based CF MSE: ' + str(get_mse(pred, test)))

pred = predict_topk(train, item_similarity, kind='item', k=40)
print ('Top-k Item-based CF MSE: ' + str(get_mse(pred, test)))
```

```
Top-k User-based CF MSE: 6.50944678543
Top-k Item-based CF MSE: 7.70361286276
```

In [58]:
```
user_pred
```

Out[58]:
```
array([[ 3.70695387,  1.77226928,  1.37134449, ...,  0.        ,
         0.05686528,  0.06737564],
       [ 2.11961776,  0.        ,  0.32432317, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.09570795,  0.        ,  0.04823979, ...,  0.05093449,
         0.        ,  0.        ],
       ...,
       [ 3.1734411 ,  0.        ,  0.23053873, ...,  0.        ,
         0.        ,  0.        ],
       [ 2.57122829,  0.86255222,  0.15276967, ...,  0.        ,
         0.        ,  0.        ],
       [ 3.17592083,  2.38051194,  1.40333091, ...,  0.        ,
         0.05770215,  0.05766682]])
```

In [57]:
```
item_pred
```

Out[57]:
```
array([[ 3.84926786,  2.64005515,  2.47447308, ...,  0.0404823 ,
         0.09380348,  0.22832037],
       [ 0.60617494,  0.        ,  0.1809131 , ...,  0.11566095,
         0.        ,  0.        ],
       [ 0.09208682,  0.        ,  0.        , ...,  0.38612203,
         0.        ,  0.        ],
       ...,
       [ 0.48244091,  0.        ,  0.36382112, ...,  0.        ,
         0.04533162,  0.        ],
       [ 0.92580901,  0.45569887,  0.25335882, ...,  0.10190914,
         0.        ,  0.        ],
       [ 2.15592136,  2.48589154,  2.00406752, ...,  0.        ,
         0.05774256,  0.22350171]])
```

As previously mentioned, the unknown values can be calculated for all the users by taking the dot product between the distance matrix and the rating matrix and then normalizing the data with the number of ratings as follows:

In [45]:
```
k_array = [5, 15, 30, 50]
user_train_mse = []
user_test_mse = []
item_test_mse = []
item_train_mse = []

def get_mse(pred, actual):
    pred = pred[actual.nonzero()].flatten()
    actual = actual[actual.nonzero()].flatten()
    return mean_squared_error(pred, actual)

for k in k_array:
    user_pred = predict_topk(train, user_similarity, kind='user', k=k)
    item_pred = predict_topk(train, item_similarity, kind='item', k=k)

    user_train_mse += [get_mse(user_pred, train)]
    user_test_mse += [get_mse(user_pred, test)]

    item_train_mse += [get_mse(item_pred, train)]
    item_test_mse += [get_mse(item_pred, test)]
```

In [46]:
```
item_train_mse
```

Out[46]:
```
[1.6274516838721487,
 2.6709604725951848,
 3.2756984602447701,
 3.7197318367662655]
```

In [47]:
```
item_test_mse
```

Out[47]:
```
[8.282234010015614, 7.4410129915028103, 7.5669281729059792, 7.839953122
3129566]
```

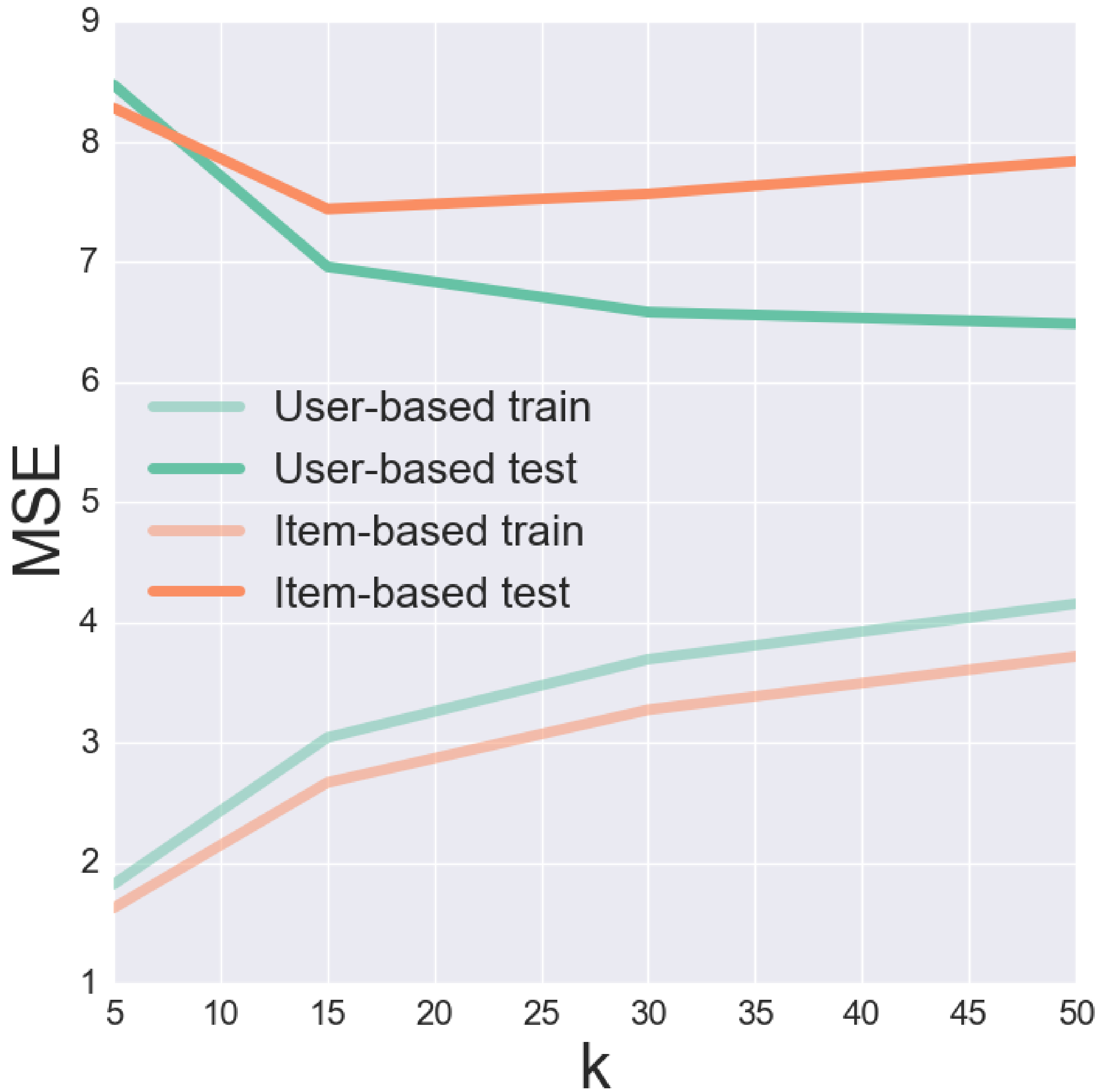
In [48]:
```
user_test_mse
```

Out[48]:
```
[8.4752074826747066,
 6.9606700502137331,
 6.5837024677703226,
 6.4869522928640588]
```

In [49]:
```
user_train_mse
```

Out[49]:
```
[1.8239131687245547,
 3.0448702482940631,
 3.6957804093257249,
 4.1577905850250874]
```

In [50]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

pal = sns.color_palette("Set2", 2)

plt.figure(figsize=(8, 8))
plt.plot(k_array, user_train_mse, c=pal[0], label='User-based train', al
pha=0.5, linewidth=5)
plt.plot(k_array, user_test_mse, c=pal[0], label='User-based test', line
width=5)
plt.plot(k_array, item_train_mse, c=pal[1], label='Item-based train', al
pha=0.5, linewidth=5)
plt.plot(k_array, item_test_mse, c=pal[1], label='Item-based test', line
width=5)
plt.legend(loc='best', fontsize=20)
plt.xticks(fontsize=16);
plt.yticks(fontsize=16);
plt.xlabel('k', fontsize=30);
plt.ylabel('MSE', fontsize=30);
```



In [59]:
```
!head -5 u.item
```

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20
Story%20(1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Golden
Eye%20(1995)|0|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Four%
20Rooms%20(1995)|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Get%2
0Shorty%20(1995)|0|1|0|0|0|1|0|0|1|0|0|0|0|0|0|0|0|0|0
5|Copycat (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Copycat%
20(1995)|0|0|0|0|0|0|1|0|1|0|0|0|0|0|0|0|1|0|0
```

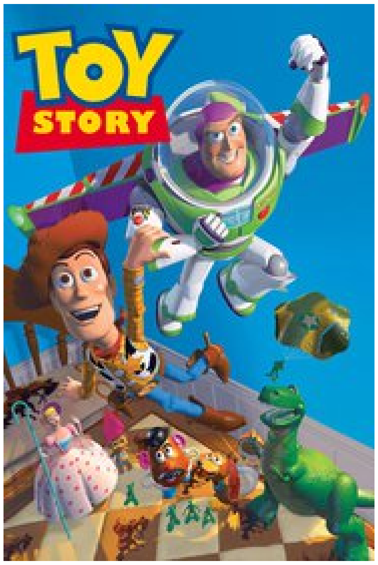
In [61]:
```
import requests
import json

response = requests.get('http://us.imdb.com/M/title-exact?Toy%20Story%20
(1995)')
print (response.url.split('/')[-2])
```

```
tt0114709
```

In [65]:
```
# Get base url filepath structure. w185 corresponds to size of movie pos
ter.
headers = {'Accept': 'application/json'}
payload = {'api_key': 'cf8e935c41dbc8f914661aa0a67a5fc1'}
response = requests.get("http://api.themoviedb.org/3/configuration", par
ams=payload, headers=headers)
response = json.loads(response.text)
base_url = response['images']['base_url'] + 'w185'

def get_poster(imdb_url, base_url):
    # Get IMDB movie ID
    response = requests.get(imdb_url)
    movie_id = response.url.split('/')[-2]
    # Query themoviedb.org API for movie poster path.
    movie_url = 'http://api.themoviedb.org/3/movie/{:}/images'.format(mo
vie_id)
    headers = {'Accept': 'application/json'}
    payload = {'api_key': 'cf8e935c41dbc8f914661aa0a67a5fc1'}
    response = requests.get(movie_url, params=payload, headers=headers)
    try:
        file_path = json.loads(response.text)['posters'][0]['file_path']
    except:
        # IMDB movie ID is sometimes no good. Need to get correct one.
        movie_title = imdb_url.split('?')[-1].split('(')[0]
        payload['query'] = movie_title
        response = requests.get('http://api.themoviedb.org/3/search/movi
e', params=payload, headers=headers)
        movie_id = json.loads(response.text)['results'][0]['id']
        payload.pop('query', None)
        movie_url = 'http://api.themoviedb.org/3/movie/{:}/images'.forma
t(movie_id)
        response = requests.get(movie_url, params=payload, headers=heade
rs)
        file_path = json.loads(response.text)['posters'][0]['file_path']

    return base_url + file_path
```

In [66]:
```
from IPython.display import Image
from IPython.display import display

toy_story = 'http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)'
Image(url=get_poster(toy_story, base_url))
```

Out[66]:



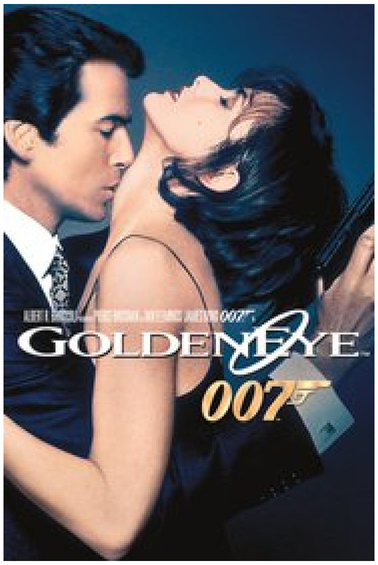
In [83]:
```
# Load in movie data
idx_to_movie = {}
with open('/Users/jyothi/Downloads/ml-100k/u.item', encoding = "ISO-8859
-1") as f:
    for line in f.readlines():
        info = line.split('|')
        idx_to_movie[int(info[0])-1] = info[4]
```

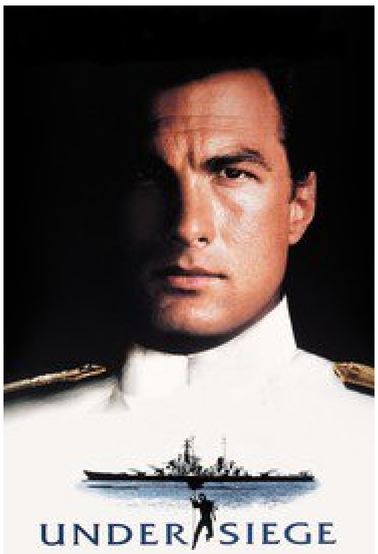
In [ ]:

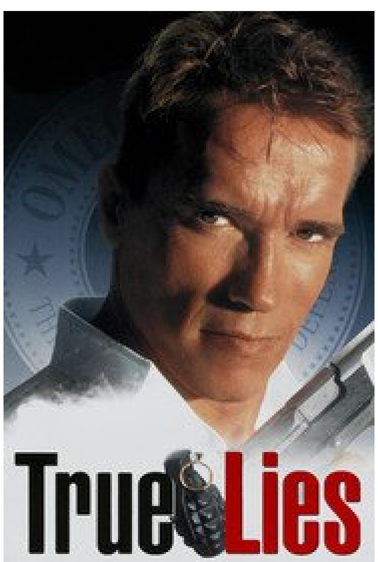
In [96]:
```
idx = 1 # Toy Story
movies = top_k_movies(item_similarity, idx_to_movie, idx)
posters = tuple(Image(url=get_poster(movie, base_url)) for movie in movi
es)
```

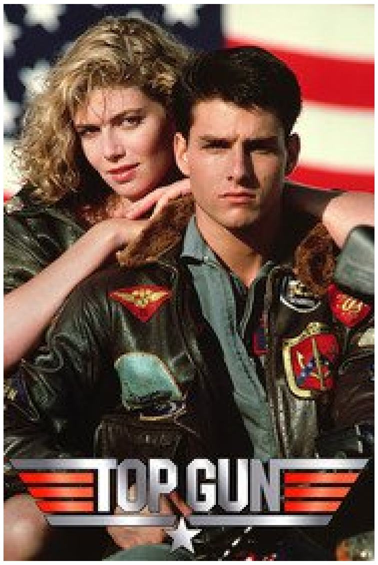
In [70]:
```
def top_k_movies(similarity, mapper, movie_idx, k=6):
    return [mapper[x] for x in np.argsort(similarity[movie_idx,:])[:-k-
1:-1]]
```

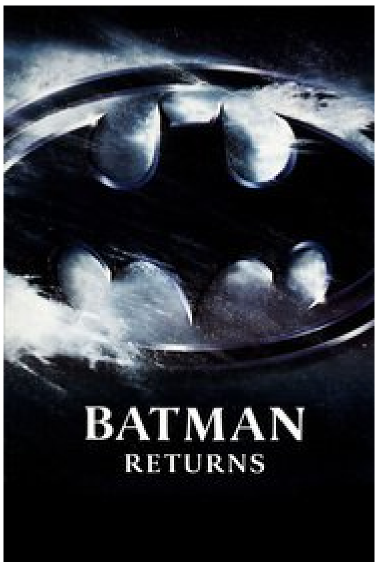
In [97]:
```
display(*posters)
```

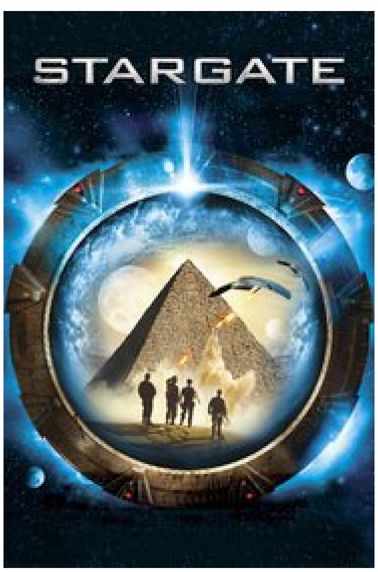

GOLDENEYE
007


UNDER SIEGE


True Lies


TOP GUN


BATMAN
RETURNS


STARGATE

In [87]:
```
movies
```

Out[87]:
```
['http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)',
 'http://us.imdb.com/M/title-exact?Star%20Wars%20(1977)',
 'http://us.imdb.com/M/title-exact?Independence%20Day%20(1996)',
 'http://us.imdb.com/M/title-exact?Return%20of%20the%20Jedi%20(1983)',
 'http://us.imdb.com/M/title-exact?Willy%20Wonka%20and%20the%20Chocolat
e%20Factory%20(1971)',
 'http://us.imdb.com/M/title-exact?Raiders%20of%20the%20Lost%20Ark%20(1
981)']
```

In [ ]: