k-means Clustering of Movie Ratings

May 14, 2019

1 k-means Clustering of Movie Ratings

Say you're a data analyst at Netflix and you want to explore the similarities and differences in people's tastes in movies based on how they rate different movies. Can understanding these ratings contribute to a movie recommendation system for users? Let's dig into the data and see.

The data we'll be using comes from the wonderful MovieLens user rating dataset. We'll be looking at individual movie ratings later in the notebook, but let us start with how ratings of genres compare to each other.

1.1 Dataset overview

The dataset has two files. We'll import them both into pandas dataframes:

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np
        from scipy.sparse import csr_matrix
        import helper

        # Import the Movies dataset
        movies = pd.read_csv('ml-latest-small/movies.csv')
        movies.head()
```

```
Out[1]:    movieId                                title  \
        0        1                    Toy Story (1995)
        1        2                      Jumanji (1995)
        2        3             Grumpier Old Men (1995)
        3        4            Waiting to Exhale (1995)
        4        5  Father of the Bride Part II (1995)

                                              genres
        0  Adventure|Animation|Children|Comedy|Fantasy
        1                   Adventure|Children|Fantasy
        2                               Comedy|Romance
        3                        Comedy|Drama|Romance
        4                                      Comedy
```

```
In [2]: # Import the ratings dataset
        ratings = pd.read_csv('ml-latest-small/ratings.csv')
        ratings.head()
```

Out[2]:

	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

Now that we know the structure of our dataset, how many records do we have in each of these tables?

```
In [3]: print('The dataset contains: ', len(ratings), ' ratings of ', len(movies), ' movies.')
```

```
The dataset contains:  100004  ratings of  9125  movies.
```

1.2 Romance vs. Scifi

Let's start by taking a subset of users, and seeing what their preferred genres are. We're hiding the most data preprocessing in helper functions so the focus is on the topic of clustering. It would be useful if you skim helper.py to see how these helper functions are implemented after finishing this notebook.

```
In [4]: # Calculate the average rating of romance and scifi movies

        genre_ratings = helper.get_genre_ratings(ratings, movies, ['Romance', 'Sci-Fi'], ['avg_
        genre_ratings.head()
```

Out[4]:

userId	avg_romance_rating	avg_scifi_rating
1	3.50	2.40
2	3.59	3.80
3	3.65	3.14
4	4.50	4.26
5	4.08	4.00

The function `get_genre_ratings` calculated each user's average rating of all romance movies and all scifi movies. Let's bias our dataset a little by removing people who like both scifi and romance, just so that our clusters tend to define them as liking one genre more than the other.

```
In [5]: biased_dataset = helper.bias_genre_rating_dataset(genre_ratings, 3.2, 2.5)

        print( "Number of records: ", len(biased_dataset))
        biased_dataset.head()
```

```
Number of records:  183
```

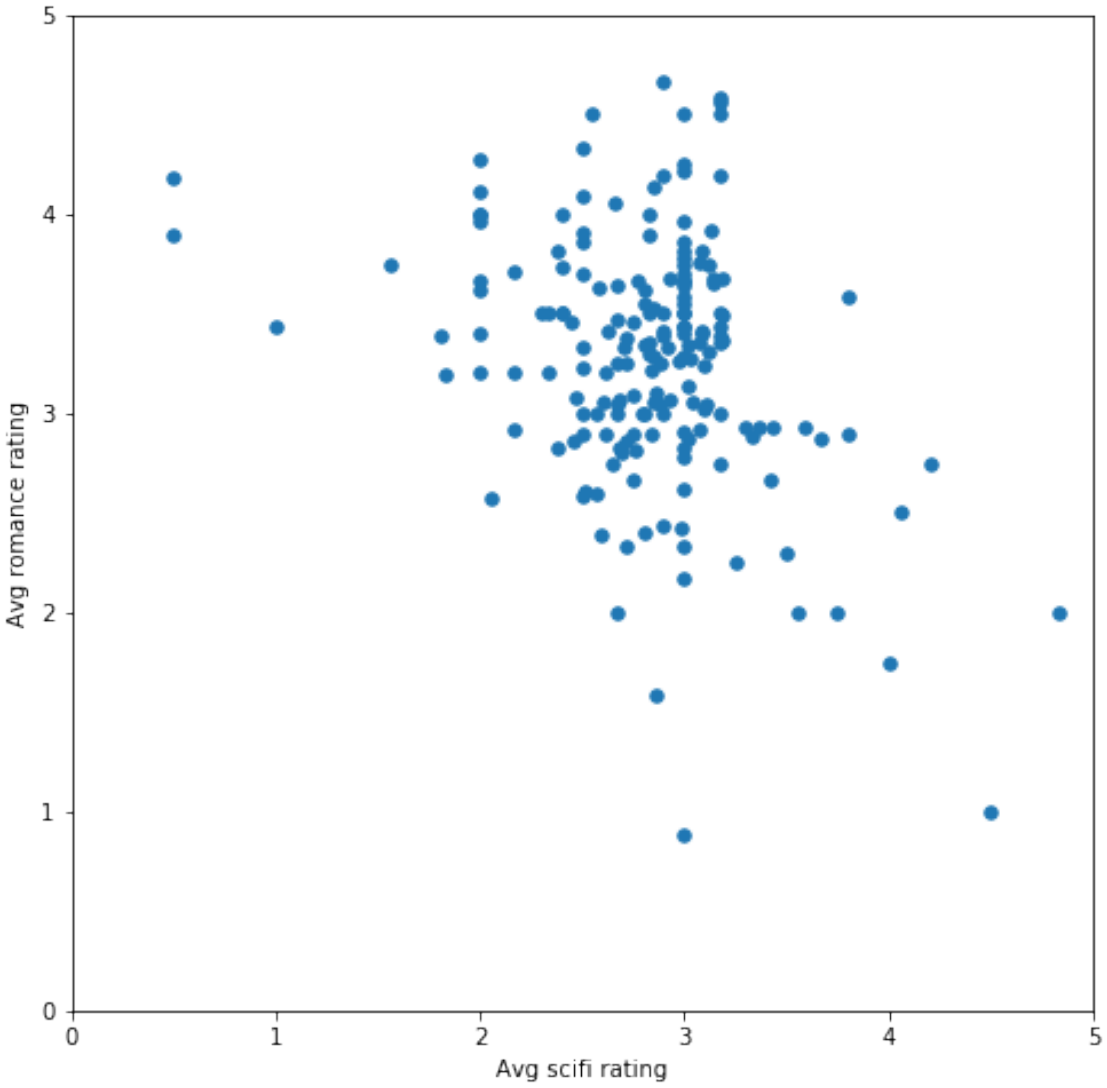
Out[5]:

	userId	avg_romance_rating	avg_scifi_rating
0	1	3.50	2.40
1	3	3.65	3.14
2	6	2.90	2.75
3	7	2.93	3.36
4	12	2.89	2.62

So we can see we have 183 users, and for each user we have their average rating of the romance and sci movies they've watched.

Let us plot this dataset:

```
In [6]: %matplotlib inline

        helper.draw_scatterplot(biased_dataset['avg_scifi_rating'],'Avg scifi rating', biased_
```

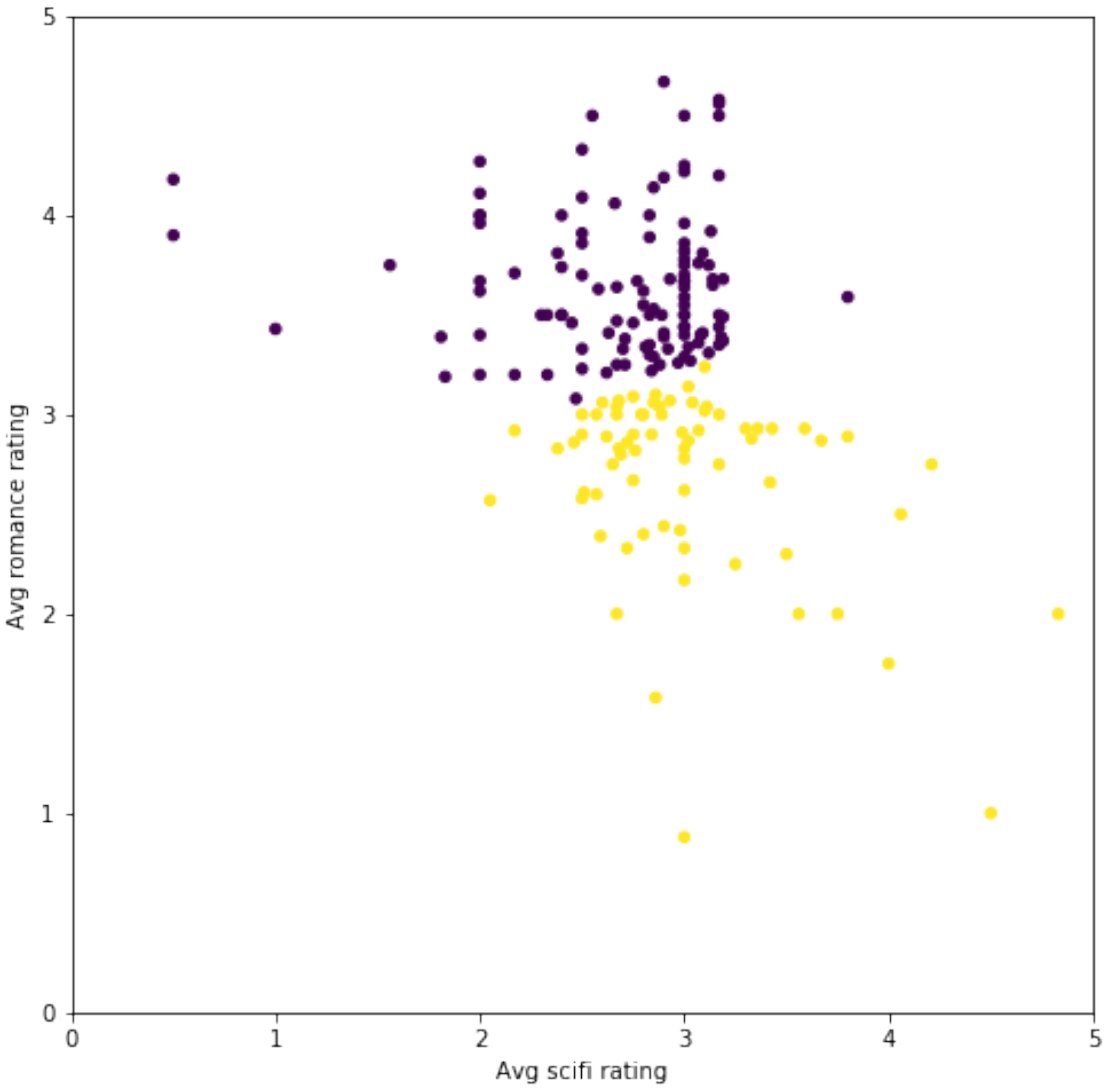


We can see come clear bias in this sample (that we created on purpose). How would it look if we break the sample down into two groups using k-means?

```
In [7]: # Let's turn our dataset into a list
        X = biased_dataset[['avg_scifi_rating','avg_romance_rating']].values
```

- Import KMeans
- Prepare KMeans with n_clusters = 2
- Pass the dataset **X** to KMeans' fit_predict method and retrieve the clustering labels into *predictions*

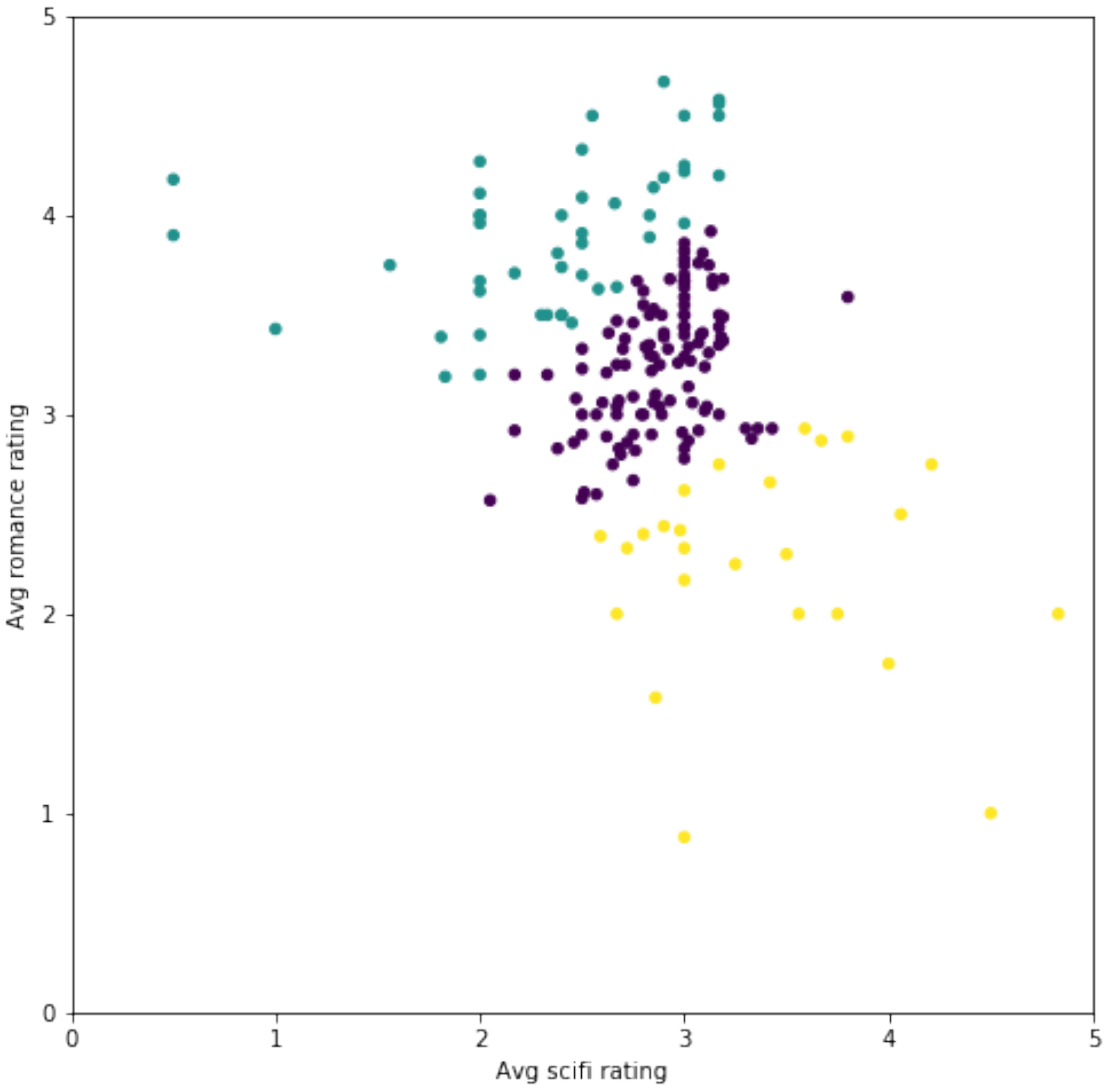
```
In [8]: # TODO: Import KMeans
        from  sklearn.cluster import KMeans

        # TODO: Create an instance of KMeans to find two clusters
        kmeans_1 = KMeans(n_clusters=2)

        # TODO: use fit_predict to cluster the dataset
        predictions = kmeans_1.fit_predict(X)

        # Plot
        helper.draw_clusters(biased_dataset, predictions)
```



We can see that the groups are mostly based on how each person rated romance movies. If their average rating of romance movies is over 3 stars, then they belong to one group. Otherwise, they belong to the other group.

What would happen if we break them down into three groups?

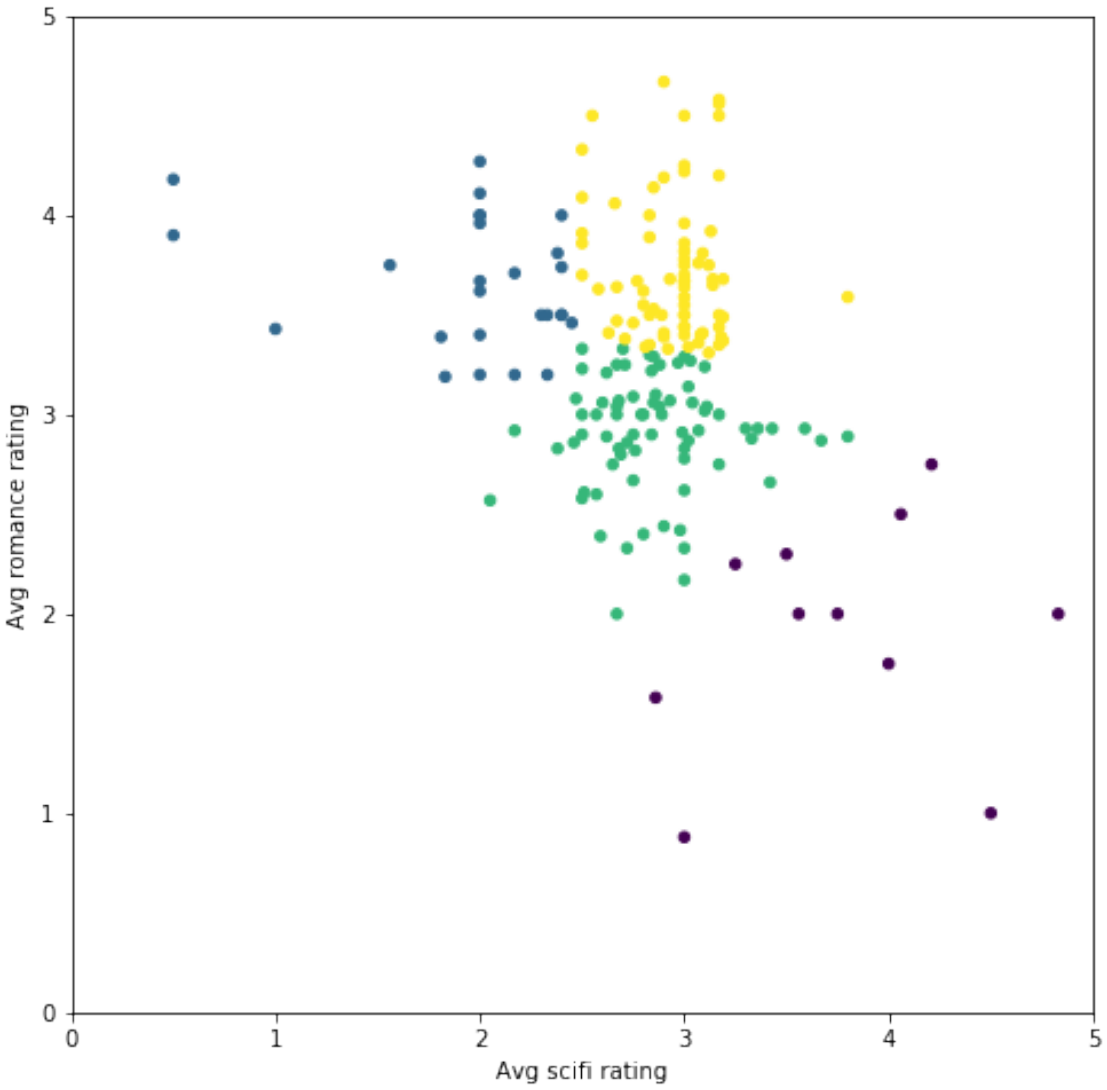
```
In [9]: # TODO: Create an instance of KMeans to find three clusters
        kmeans_2 = KMeans(n_clusters=3)

        # TODO: use fit_predict to cluster the dataset
        predictions_2 = kmeans_2.fit_predict(X)

        # Plot
        helper.draw_clusters(biased_dataset, predictions_2)
```



Now the average scifi rating is starting to come into play. The groups are: * people who like romance but not scifi * people who like scifi but not romance * people who like both scifi and romance

Let's add one more group

```
In [10]: # TODO: Create an instance of KMeans to find four clusters
         kmeans_3 = KMeans(n_clusters=4)

         # TODO: use fit_predict to cluster the dataset
         predictions_3 = kmeans_3.fit_predict(X)

         # Plot
         helper.draw_clusters(biased_dataset, predictions_3)
```



We can see that the more clusters we break our dataset down into, the more similar the tastes of the population of each cluster to each other.

1.3 Choosing K

Great, so we can cluster our points into any number of clusters. What's the right number of clusters for this dataset?

There are several ways of choosing the number of clusters, k. We'll look at a simple one called "the elbow method". The elbow method works by plotting the ascending values of k versus the total error calculated using that k.

How do we calculate total error? One way to calculate the error is squared error. Say we're calculating the error for k=2. We'd have two clusters each having one "centroid" point. For each point in our dataset, we'd subtract its coordinates from the centroid of the cluster it belongs to. We then square the result of that subtraction (to get rid of the negative values), and sum the values.

This would leave us with an error value for each point. If we sum these error values, we'd get the total error for all points when k=2.

Our mission now is to do the same for each k (between 1 and, say, the number of elements in our dataset)

```
In [11]: # Choose the range of k values to test.
         # We added a stride of 5 to improve performance. We don't need to calculate the error
         possible_k_values = range(2, len(X)+1, 5)

         # Calculate error values for all k values we're interested in
         errors_per_k = [helper.clustering_errors(k, X) for k in possible_k_values]
```

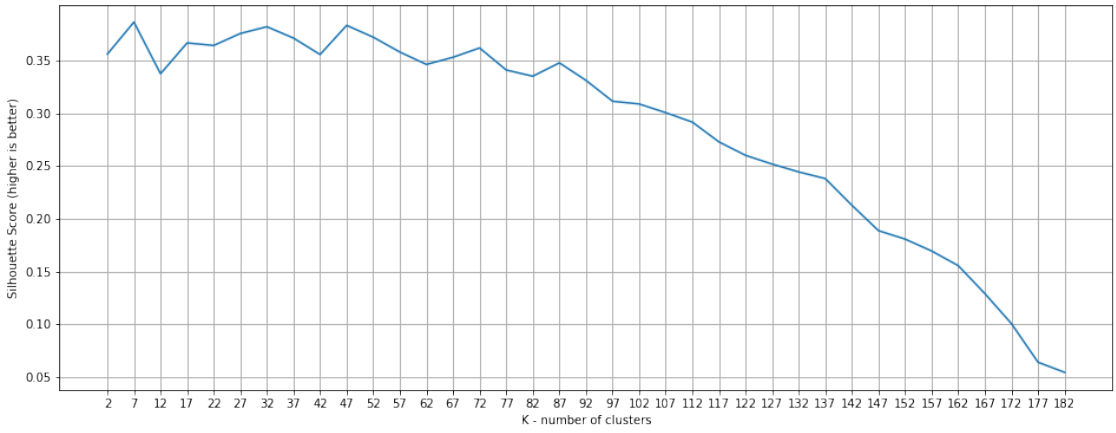
```
/home/deeplearning/anaconda3/lib/python3.7/site-packages/sklearn/cluster/k_means_.py:971: Conve
  return_n_iter=True)
```

```
In [12]: # Optional: Look at the values of K vs the silhouette score of running K-means with t
         list(zip(possible_k_values, errors_per_k))
```

```
Out[12]: [(2, 0.3558817876472827),
          (7, 0.3862169048216939),
          (12, 0.3373088862923391),
          (17, 0.3664881078131716),
          (22, 0.3640823909024716),
          (27, 0.37542430213984795),
          (32, 0.38182545623233116),
          (37, 0.3710683852863501),
          (42, 0.35550109189592194),
          (47, 0.38311174443209073),
          (52, 0.3718894128697994),
          (57, 0.3578481298204598),
          (62, 0.3461261022727476),
          (67, 0.35292617758853584),
          (72, 0.36173562028604184),
          (77, 0.3408419904324373),
          (82, 0.33496802030461065),
          (87, 0.34762196355892294),
          (92, 0.3310403939597036),
          (97, 0.31128445822459394),
          (102, 0.3087060193830088),
          (107, 0.3004577195648771),
          (112, 0.29154327938172053),
          (117, 0.2727934541931141),
          (122, 0.26007030421131266),
          (127, 0.251807088069259),
          (132, 0.24437182914070316),
          (137, 0.23805340070519967),
          (142, 0.21281541182333125),
          (147, 0.18878280868344108),
```

```
 (152, 0.18078982943342),
 (157, 0.16952046352864142),
 (162, 0.15564994602870558),
 (167, 0.12920960729960135),
 (172, 0.10075966098920461),
 (177, 0.0642301201631745),
 (182, 0.0546448087431694)]
```

In [13]:
```
# Plot the each value of K vs. the silhouette score at that value
fig, ax = plt.subplots(figsize=(16, 6))
ax.set_xlabel('K - number of clusters')
ax.set_ylabel('Silhouette Score (higher is better)')
ax.plot(possible_k_values, errors_per_k)

# Ticks and grid
xticks = np.arange(min(possible_k_values), max(possible_k_values)+1, 5.0)
ax.set_xticks(xticks, minor=False)
ax.set_xticks(xticks, minor=True)
ax.xaxis.grid(True, which='both')
yticks = np.arange(round(min(errors_per_k), 2), max(errors_per_k), .05)
ax.set_yticks(yticks, minor=False)
ax.set_yticks(yticks, minor=True)
ax.yaxis.grid(True, which='both')
```

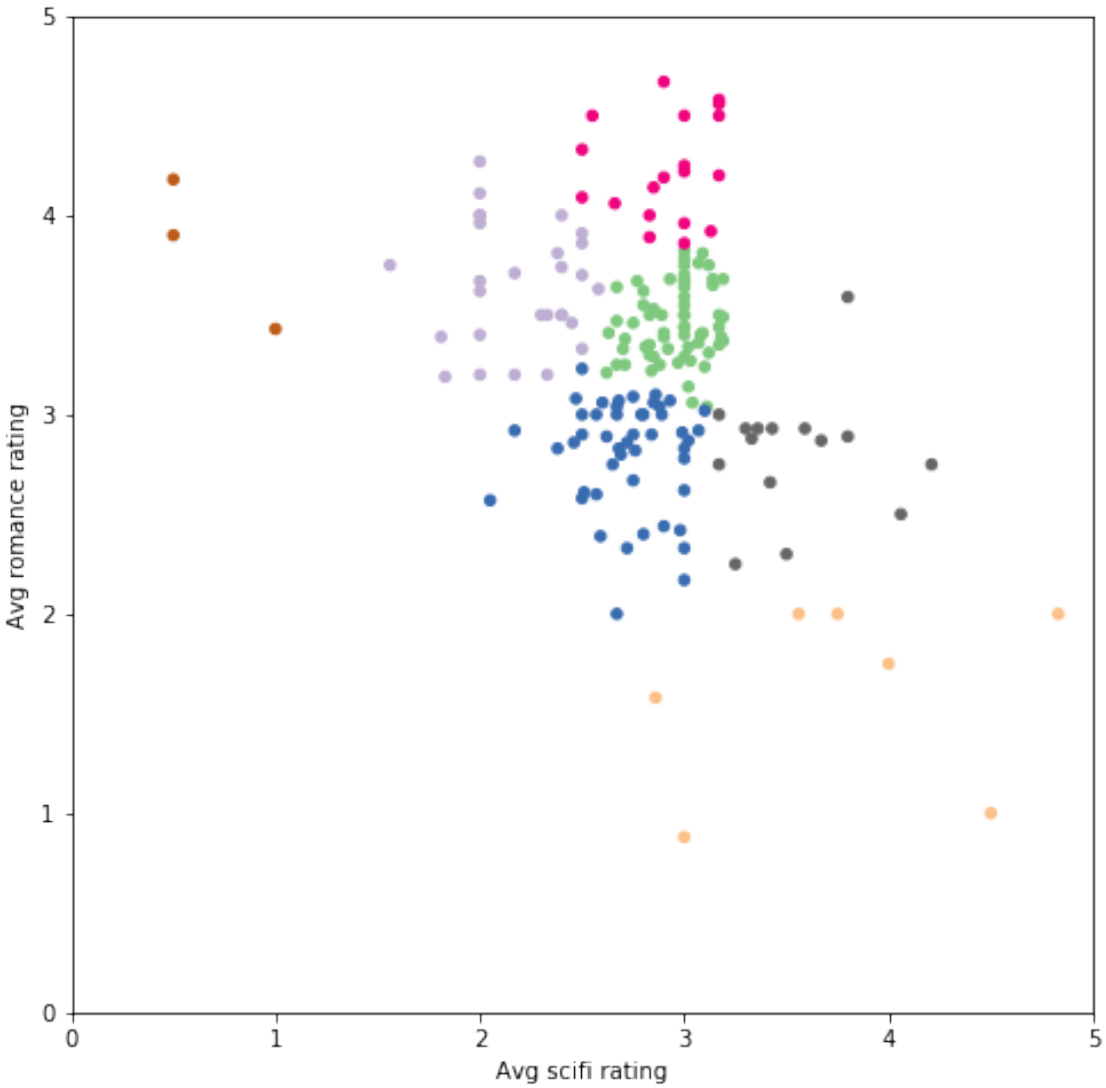


Looking at this graph, good choices for k include 7, 22, 27, 32, amongst other values (with a slight variation between different runs). Increasing the number of clusters (k) beyond that range starts to result in worse clusters (according to Silhouette score)

My pick would be k=7 because it's easier to visualize:

In [14]:
```
# TODO: Create an instance of KMeans to find seven clusters
kmeans_4 = KMeans(n_clusters=7)

# TODO: use fit_predict to cluster the dataset
```

```
predictions_4 = kmeans_4.fit_predict(X)

# plot
helper.draw_clusters(biased_dataset, predictions_4, cmap='Accent')
```



Note: As you try to plot larger values of k (more than 10), you'll have to make sure your plotting library is not reusing colors between clusters. For this plot, we had to use the matplotlib colormap 'Accent' because other colormaps either did not show enough contrast between colors, or were recycling colors past 8 or 10 clusters.

1.4 Throwing some Action into the mix

So far, we've only been looking at how users rated romance and scifi movies. Let's throw another genre into the mix. Let's add the Action genre.

Our dataset now looks like this:

```
In [15]: biased_dataset_3_genres = helper.get_genre_ratings(ratings, movies,
                                                            ['Romance', 'Sci-Fi', 'Action'],
                                                            ['avg_romance_rating', 'avg_scif
         biased_dataset_3_genres = helper.bias_genre_rating_dataset(biased_dataset_3_genres, 3

         print( "Number of records: ", len(biased_dataset_3_genres))
         biased_dataset_3_genres.head()
```

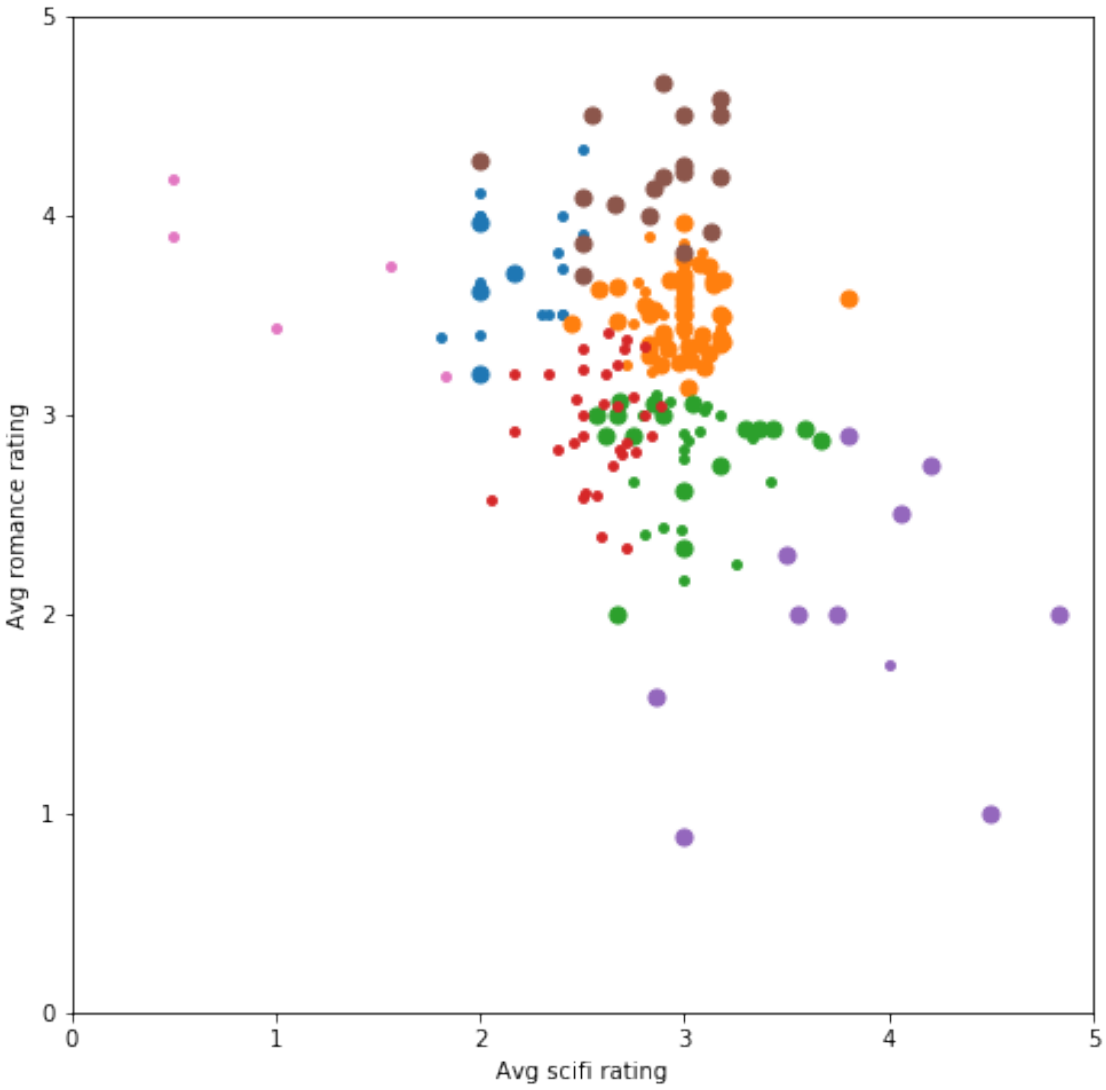
```
Number of records:  183
```

Out[15]:

	userId	avg_romance_rating	avg_scifi_rating	avg_action_rating
0	1	3.50	2.40	2.80
1	3	3.65	3.14	3.47
2	6	2.90	2.75	3.27
3	7	2.93	3.36	3.29
4	12	2.89	2.62	3.21

```
In [16]: X_with_action = biased_dataset_3_genres[['avg_scifi_rating',
                                                 'avg_romance_rating',
                                                 'avg_action_rating']].values
```

```
In [17]: # TODO: Create an instance of KMeans to find seven clusters
         kmeans_5 = KMeans(n_clusters=7)

         # TODO: use fit_predict to cluster the dataset
         predictions_5 = kmeans_5.fit_predict(X_with_action)

         # plot
         helper.draw_clusters_3d(biased_dataset_3_genres, predictions_5)
```



We're still using the x and y axes for scifi and romance respectively. We are using the size of the dot to roughly code the 'action' rating (large dot for avg ratings over than 3, small dot otherwise).

We can start seeing the added genre is changing how the users are clustered. The more data we give to k-means, the more similar the tastes of the people in each group would be. Unfortunately, though, we lose the ability to visualize what's going on past two or three dimensions if we continue to plot it this way. In the next section, we'll start using a different kind of plot to be able to see clusters with up to fifty dimensions.

1.5 Movie-level Clustering

Now that we've established some trust in how k-means clusters users based on their genre tastes, let's take a bigger bite and look at how users rated individual movies. To do that, we'll shape the dataset in the form of userId vs user rating for each movie. For example, let's look at a subset of the dataset:

In [18]:
```
# Merge the two tables then pivot so we have Users X Movies dataframe
ratings_title = pd.merge(ratings, movies[['movieId', 'title']], on='movieId' )
user_movie_ratings = pd.pivot_table(ratings_title, index='userId', columns= 'title', v

print('dataset dimensions: ', user_movie_ratings.shape, '\n\nSubset example:')
user_movie_ratings.iloc[:6, :10]
```

```
dataset dimensions:  (671, 9064)

Subset example:
```

Out[18]:

title	"Great Performances" Cats (1998)	$9.99 (2008)	\
userId			
1	NaN	NaN	
2	NaN	NaN	
3	NaN	NaN	
4	NaN	NaN	
5	NaN	NaN	
6	NaN	NaN	

title	'Hellboy': The Seeds of Creation (2004)	\
userId		
1	NaN	
2	NaN	
3	NaN	
4	NaN	
5	NaN	
6	NaN	

title	'Neath the Arizona Skies (1934)	'Round Midnight (1986)	\
userId			
1	NaN	NaN	
2	NaN	NaN	
3	NaN	NaN	
4	NaN	NaN	
5	NaN	NaN	
6	NaN	NaN	

title	'Salem's Lot (2004)	'Til There Was You (1997)	'burbs, The (1989)	\
userId				
1	NaN	NaN	NaN	
2	NaN	NaN	NaN	
3	NaN	NaN	NaN	
4	NaN	NaN	NaN	
5	NaN	NaN	NaN	
6	NaN	NaN	4.0	

```
title   'night Mother (1986)  (500) Days of Summer (2009)
userId
1                         NaN                          NaN
2                         NaN                          NaN
3                         NaN                          NaN
4                         NaN                          NaN
5                         NaN                          NaN
6                         NaN                          NaN
```

The dominance of NaN values presents the first issue. Most users have not rated and watched most movies. Datasets like this are called "sparse" because only a small number of cells have values.

To get around this, let's sort by the most rated movies, and the users who have rated the most number of movies. That will present a more 'dense' region when we peak at the top of the dataset.

If we're to choose the most-rated movies vs users with the most ratings, it would look like this:

```
In [19]: n_movies = 30
         n_users = 18
         most_rated_movies_users_selection = helper.sort_by_rating_density(user_movie_ratings,

         print('dataset dimensions: ', most_rated_movies_users_selection.shape)
         most_rated_movies_users_selection.head()

dataset dimensions:  (18, 30)


Out[19]: title  Forrest Gump (1994)  Pulp Fiction (1994)  \
         29                     5.0                 5.0
         508                    4.0                 5.0
         14                     1.0                 5.0
         72                     5.0                 5.0
         653                    4.0                 5.0

         title  Shawshank Redemption, The (1994)  Silence of the Lambs, The (1991)  \
         29                                  5.0                               4.0
         508                                 4.0                               4.0
         14                                  2.0                               5.0
         72                                  5.0                               4.5
         653                                 5.0                               4.5

         title  Star Wars: Episode IV - A New Hope (1977)  Jurassic Park (1993)  \
         29                                           4.0                  4.0
         508                                          5.0                  3.0
         14                                           5.0                  3.0
         72                                           4.5                  4.0
         653                                          5.0                  4.5

         title  Matrix, The (1999)  Toy Story (1995)  Schindler's List (1993)  \
         29                    3.0               4.0                      5.0
```

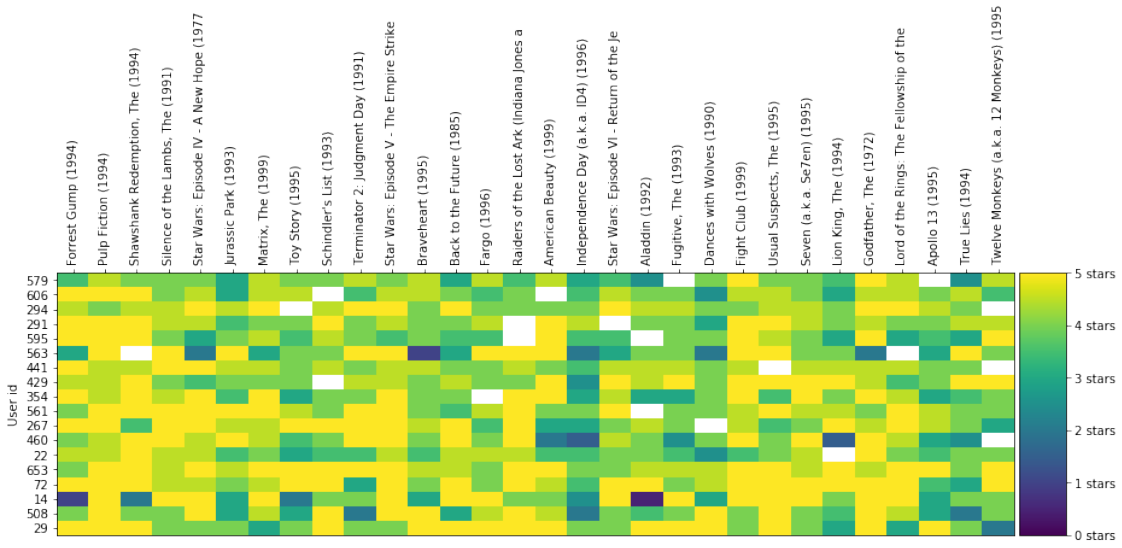
```
508                  4.5              3.0                    5.0
14                   5.0              2.0                    4.0
72                   4.5              5.0                    5.0
653                  5.0              5.0                    5.0

title  Terminator 2: Judgment Day (1991)  \
29                                   4.0
508                                  2.0
14                                   4.0
72                                   3.0
653                                  5.0

title                    ...                    Dances with Wolves (1990)  \
29                       ...                                          5.0
508                      ...                                          5.0
14                       ...                                          3.0
72                       ...                                          4.5
653                      ...                                          4.5

title  Fight Club (1999)  Usual Suspects, The (1995)  \
29                   4.0                         5.0
508                  4.0                         5.0
14                   5.0                         5.0
72                   5.0                         5.0
653                  5.0                         5.0

title  Seven (a.k.a. Se7en) (1995)  Lion King, The (1994)  \
29                             4.0                    3.0
508                            4.0                    3.5
14                             5.0                    4.0
72                             5.0                    5.0
653                            4.5                    5.0

title  Godfather, The (1972)  \
29                       5.0
508                      5.0
14                       5.0
72                       5.0
653                      4.5

title  Lord of the Rings: The Fellowship of the Ring, The (2001)  \
29                                                          3.0
508                                                         4.5
14                                                          5.0
72                                                          5.0
653                                                         5.0

title  Apollo 13 (1995)  True Lies (1994)  \
```

```
29                  5.0                 4.0
508                 3.0                 2.0
14                  3.0                 4.0
72                  3.5                 3.0
653                 5.0                 4.0

title  Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
29                                            2.0
508                                           4.0
14                                            4.0
72                                            5.0
653                                           5.0

[5 rows x 30 columns]
```

That's more like it. Let's also establish a good way for visualizing these ratings so we can attempt to visually recognize the ratings (and later, clusters) when we look at bigger subsets.

Let's use colors instead of the number ratings:

```
In [20]: helper.draw_movies_heatmap(most_rated_movies_users_selection)
```



Each column is a movie. Each row is a user. The color of the cell is how the user rated that movie based on the scale on the right of the graph.

Notice how some cells are white? This means the respective user did not rate that movie. This is an issue you'll come across when clustering in real life. Unlike the clean example we started with, real-world datasets can often be sparse and not have a value in each cell of the dataset. This makes it less straightforward to cluster users directly by their movie ratings as k-means generally does not like missing values.

For performance reasons, we'll only use ratings for 1000 movies (out of the 9000+ available in the dataset).

```
In [21]: user_movie_ratings =  pd.pivot_table(ratings_title, index='userId', columns= 'title',
         most_rated_movies_1k = helper.get_most_rated_movies(user_movie_ratings, 1000)
```

To have sklearn run k-means clustering to a dataset with missing values like this, we will first cast it to the sparse csr matrix type defined in the SciPi library.

To convert from a pandas dataframe to a sparse matrix, we'll have to convert to SparseDataFrame, then use pandas' `to_coo()` method for the conversion.

Note: `to_coo()` was only added in later versions of pandas. If you run into an error with the next cell, make sure pandas is up to date.

```
In [22]: sparse_ratings = csr_matrix(pd.SparseDataFrame(most_rated_movies_1k).to_coo())
```

1.6 Let's cluster!

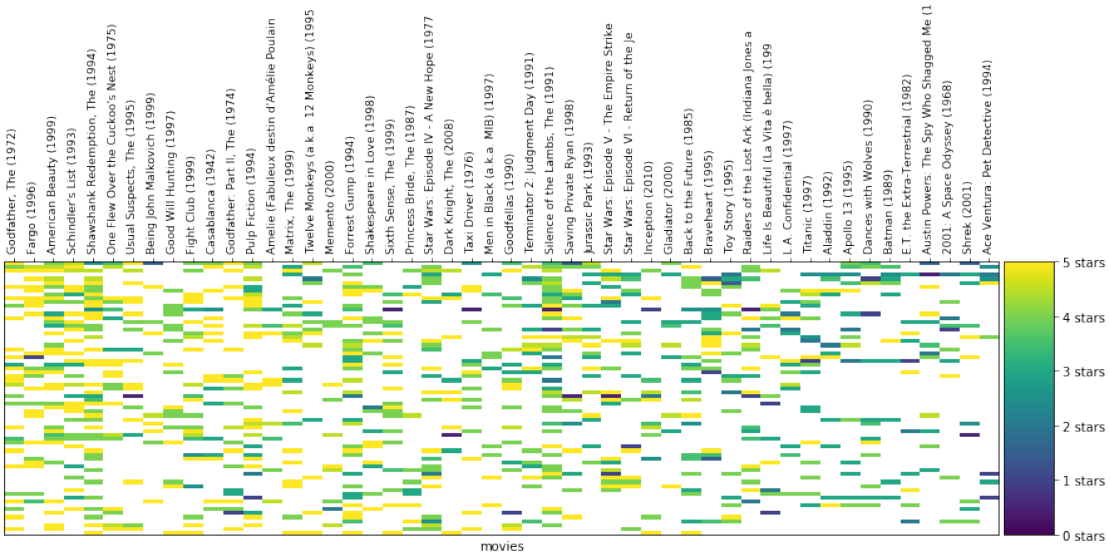
With k-means, we have to specify k, the number of clusters. Let's arbitrarily try k=20 (A better way to pick k is as illustrated above with the elbow method. That would take some processing time to run, however.):

```
In [23]: # 20 clusters
         predictions = KMeans(n_clusters=20, algorithm='full').fit_predict(sparse_ratings)
```

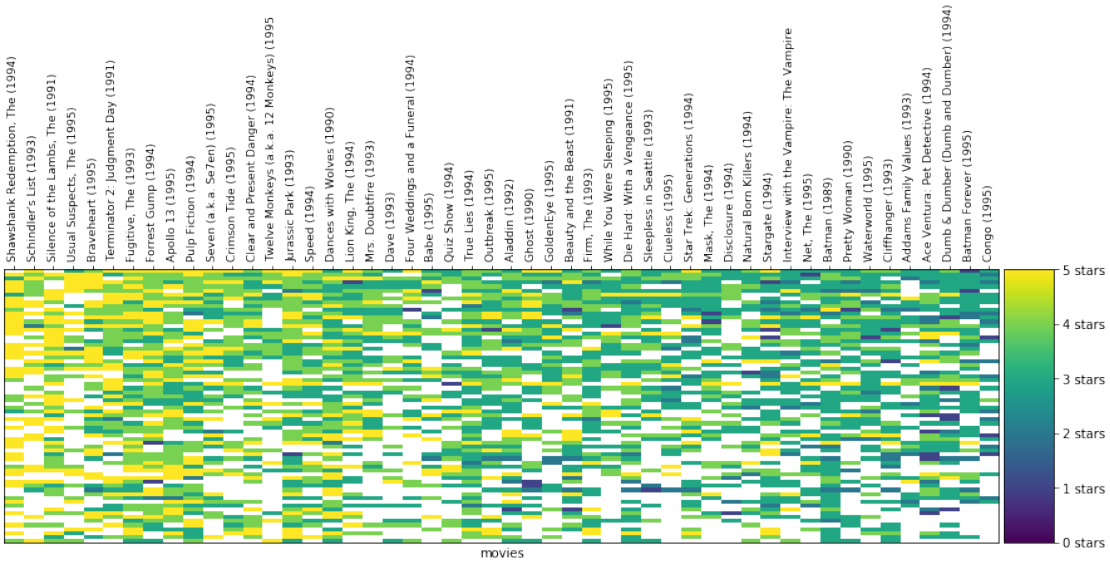
To visualize some of these clusters, we'll plot each cluster as a heat map:

```
In [24]: max_users = 70
         max_movies = 50

         clustered = pd.concat([most_rated_movies_1k.reset_index(), pd.DataFrame({'group':predi
         helper.draw_movie_clusters(clustered, max_users, max_movies)
```

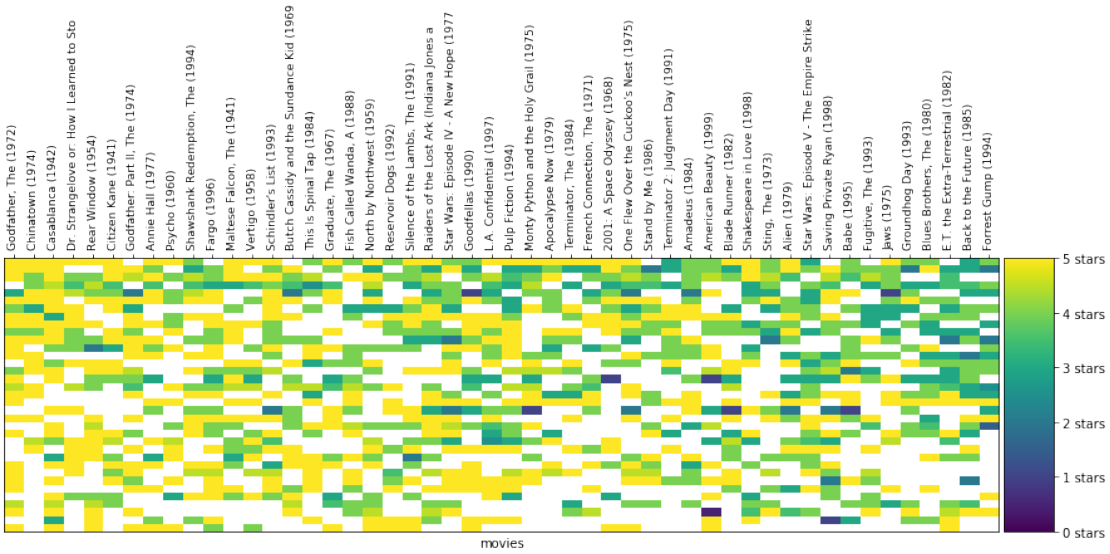
```
/home/deeplearning/Documents/UdacityMLND/K-means clustering of movie ratings/helper.py:115: Fu
  d = d.reindex_axis(d.mean().sort_values(ascending=False).index, axis=1)
/home/deeplearning/Documents/UdacityMLND/K-means clustering of movie ratings/helper.py:116: Fu
  d = d.reindex_axis(d.count(axis=1).sort_values(ascending=False).index)
/home/deeplearning/anaconda3/lib/python3.7/site-packages/matplotlib/cbook/__init__.py:424: Matp
Passing one of 'on', 'true', 'off', 'false' as a boolean is deprecated; use an actual boolean
  warn_deprecated("2.2", "Passing one of 'on', 'true', 'off', 'false' as a "


cluster # 18
# of users in cluster: 274. # of users in plot: 70
```



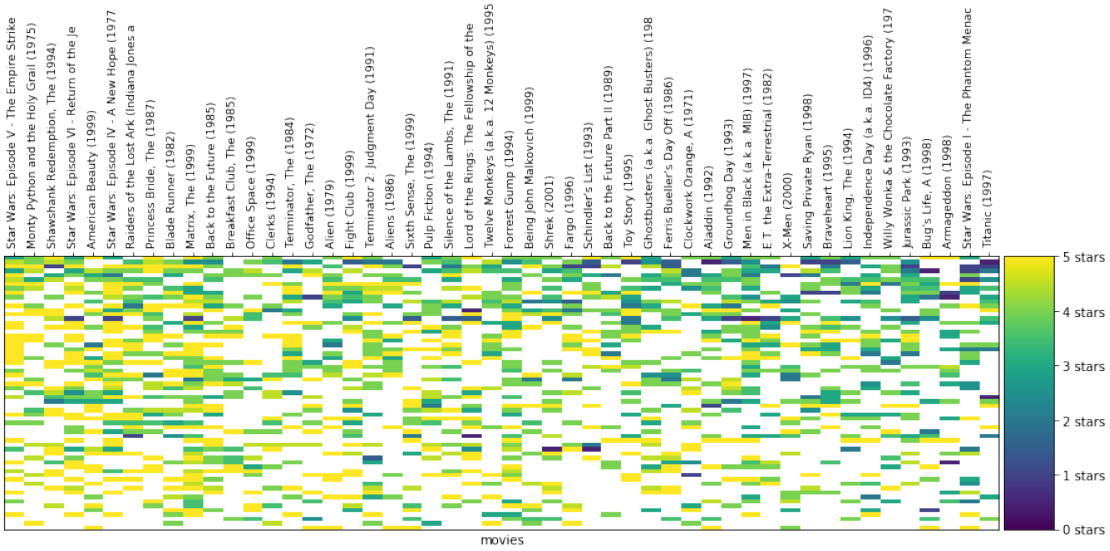
cluster # 1
# of users in cluster: 82. # of users in plot: 70



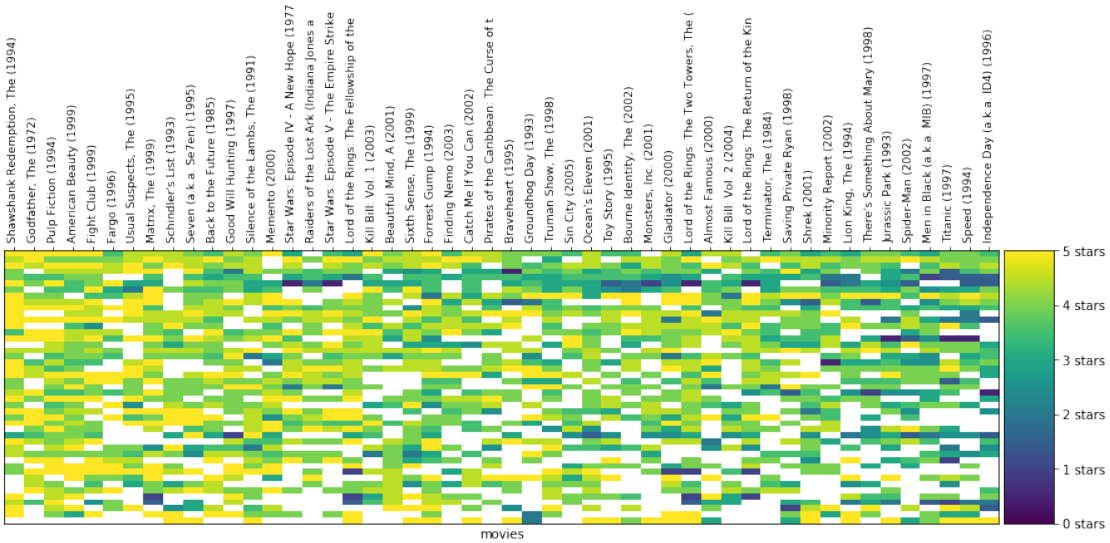
cluster # 4
# of users in cluster: 35. # of users in plot: 35



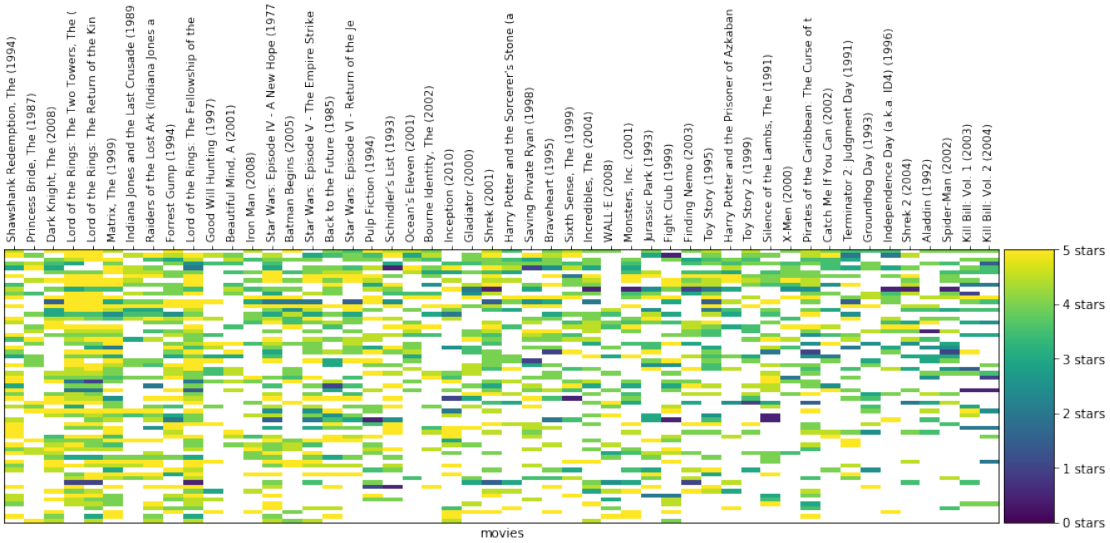
cluster # 13
# of users in cluster: 63. # of users in plot: 63



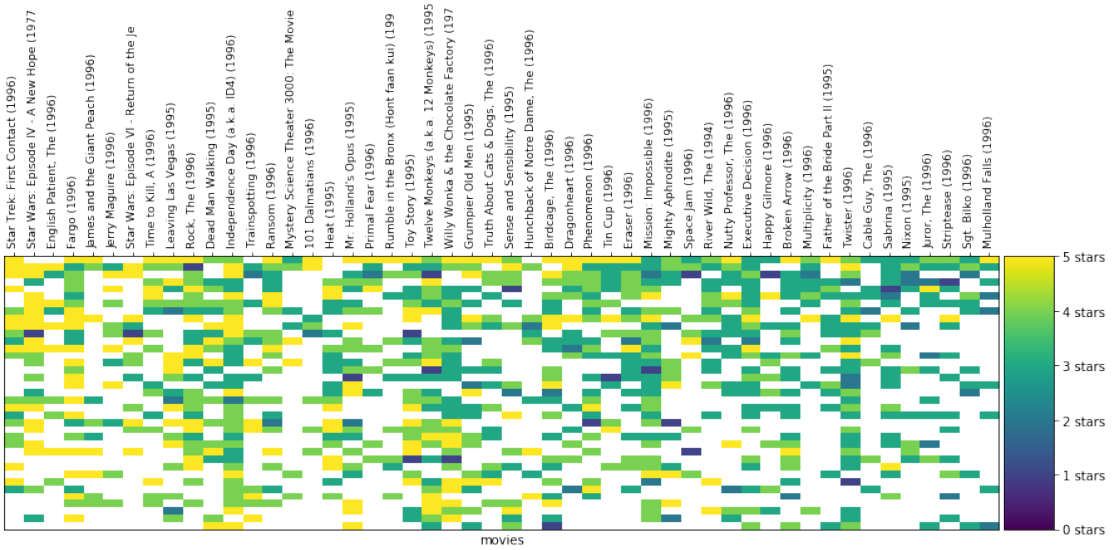
cluster # 6
# of users in cluster: 45. # of users in plot: 45



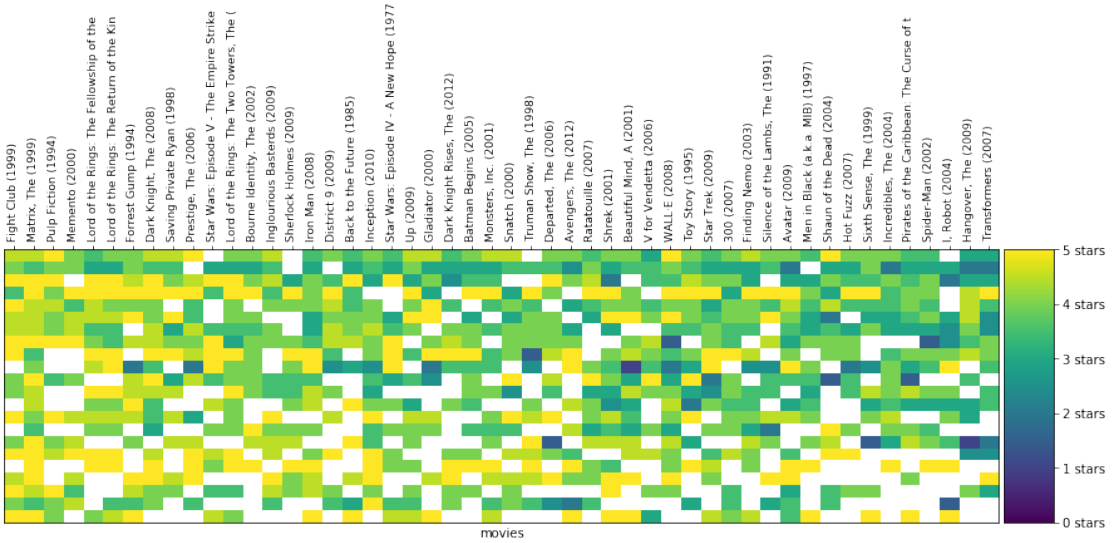
cluster # 11
# of users in cluster: 65. # of users in plot: 65



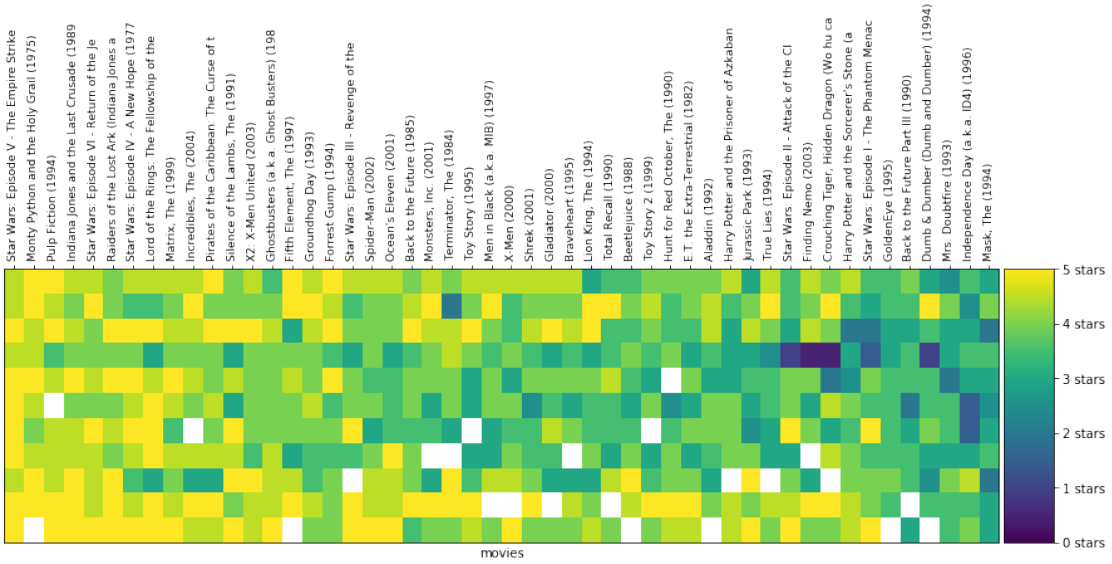
cluster # 7
# of users in cluster: 37. # of users in plot: 37



cluster # 9
# of users in cluster: 22. # of users in plot: 22



cluster # 0
# of users in cluster: 11. # of users in plot: 11



There are several things to note here: * The more similar the ratings in a cluster are, the more **vertical** lines in similar colors you'll be able to trace in that cluster. * It's super interesting to spot trends in clusters: * Some clusters are more sparse than others, containing people who probably watch and rate less movies than in other clusters. * Some clusters are mostly yellow and bring together people who really love a certain group of movies. Other clusters are mostly green or navy blue meaning they contain people who agree that a certain set of movoies deserves 2-3 stars. * Note how the movies change in every cluster. The graph filters the data to only show the most rated movies, and then sorts them by average rating. * Can you track where the Lord of the Rings movies appear in each cluster? What about Star Wars movies? * It's easy to spot **horizontal** lines with similar colors, these are users without a lot of variety in their ratings. This is likely one of the reasons for Netflix switching from a stars-based ratings to a thumbs-up/thumbs-down rating. A rating of four stars means different things to different people. * We did a few things to make the clusters visibile (filtering/sorting/slicing). This is because datasets like this are "sparse" and most cells do not have a value (because most people did not watch most movies).

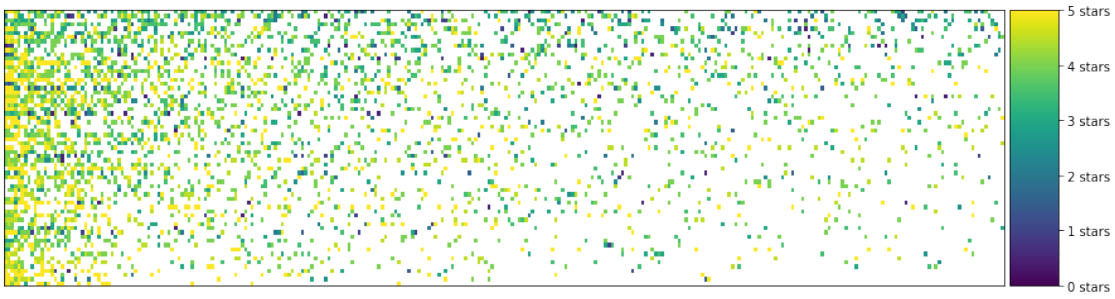
1.7 Prediction

Let's pick a cluster and a specific user and see what useful things this clustering will allow us to do.

Let's first pick a cluster:

```
In [25]: # TODO: Pick a cluster ID from the clusters above
         cluster_number = 11

         # Let's filter to only see the region of the dataset with the most number of values
         n_users = 75
         n_movies = 300
         cluster = clustered[clustered.group == cluster_number].drop(['index', 'group'], axis=
```

```
cluster = helper.sort_by_rating_density(cluster, n_movies, n_users)
helper.draw_movies_heatmap(cluster, axis_labels=False)
```



And the actual ratings in the cluster look like this:

```
In [26]: cluster.fillna('').head()
```

Out[26]:

	Lord of the Rings: The Fellowship of the Ring, The (2001)	Lord of the Rings: The Return of the King, The (2003)	Lord of the Rings: The Two Towers, The (2002)	Matrix, The (1999)	Star Wars: Episode IV - A New Hope (1977)	Shawshank Redemption, The (1994)	Forrest Gump (1994)	Shrek (2001)
22	4.5	5	5	4.5	4		3.5	
39	2			5	3	3		
38	4	4	4	3	3.5	3		
32	2.5		3	4	3.5	4		
6	5	5	5	4				

```
39                              5
38                  3           4
32                  4           4
6                             3.5

   Pirates of the Caribbean: The Curse of the Black Pearl (2003)  \
22                                                            4
39
38                                                          2.5
32                                                            4
6                                                           1.5

   Dark Knight, The (2008)         ...          \
22                       3         ...
39                       3         ...
38                                 ...
32                     3.5         ...
6                        5         ...

   Back to the Future Part II (1989)  \
22
39
38
32
6

   How the Grinch Stole Christmas (a.k.a. The Grinch) (2000)  \
22                                                            2
39
38
32                                                          4.5
6                                                             3

   Graduate, The (1967) Nightmare Before Christmas, The (1993)  \
22                                                                2.5
39
38                   2
32                                                                  4
6                                                                 1.5

   City of God (Cidade de Deus) (2002)  \
22                                    4
39
38
32
6

   Star Trek III: The Search for Spock (1984) Best in Show (2000)  \
```

```
22
39                                          3.5
38                                                                  3
32                                                                3.5
6

   Transformers (2007) GoldenEye (1995) Animal House (1978)
22
39
38
32                 2.5
6                    5

[5 rows x 300 columns]
```

Pick a blank cell from the table. It's blank because that user did not rate that movie. Can we predict whether she would like it or not? Since the user is in a cluster of users that seem to have similar taste, we can take the average of the votes for that movie in this cluster, and that would be a reasonable predicition for much she would enjoy the film.

```
In [27]: # TODO: Fill in the name of the column/movie. e.g. 'Forrest Gump (1994)'
         # Pick a movie from the table above since we're looking at a subset
         movie_name = "Forrest Gump (1994)"

         cluster[movie_name].mean()
```

```
Out[27]: 4.2317073170731705
```

And this would be our prediction for how she'd rate the movie.

1.8 Recommendation

Let's reiterate what we did in the previous step. We have used k-means to cluster users according to their ratings. This lead us to clusters of users with similar ratings and thus generally a similar taste in movies. Based on this, when one user did not have a rating for a certain movie we averaged the ratings of all the other users in the cluster, and that was our guess to how this one user would like the movie.

Using this logic, if we calculate the average score in this cluster for every movie, we'd have an understanding for how this 'taste cluster' feels about each movie in the dataset.

```
In [28]: # The average rating of 20 movies as rated by the users in the cluster
         cluster.mean().head(20)
```

```
Out[28]: Lord of the Rings: The Fellowship of the Ring, The (2001)                4.22
         Lord of the Rings: The Return of the King, The (2003)                   4.30
         Lord of the Rings: The Two Towers, The (2002)                           4.31
         Matrix, The (1999)                                                      4.29
         Star Wars: Episode IV - A New Hope (1977)                               4.16
         Shawshank Redemption, The (1994)                                        4.51
```

```
Forrest Gump (1994)                                                          4.23
Shrek (2001)                                                                 4.02
Pirates of the Caribbean: The Curse of the Black Pearl (2003)                3.85
Dark Knight, The (2008)                                                      4.34
Star Wars: Episode V - The Empire Strikes Back (1980)                        4.13
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)  4.24
Toy Story (1995)                                                             3.90
Back to the Future (1985)                                                    4.11
Incredibles, The (2004)                                                      3.95
Pulp Fiction (1994)                                                          4.10
Star Wars: Episode VI - Return of the Jedi (1983)                            4.10
Silence of the Lambs, The (1991)                                             3.87
Monsters, Inc. (2001)                                                        3.93
Schindler's List (1993)                                                      4.10
dtype: float64
```

This becomes really useful for us because we can now use it as a recommendation engine that enables our users to discover movies they're likely to enjoy.

When a user logs in to our app, we can now show them recommendations that are appropriate to their taste. The formula for these recommendations is to select the cluster's highest-rated movies that the user did not rate yet.

In [29]:
```
# TODO: Pick a user ID from the dataset
# Look at the table above outputted by the command "cluster.fillna('').head()"
# and pick one of the user ids (the first column in the table)
user_id = 2

# Get all this user's ratings
user_2_ratings  = cluster.loc[user_id, :]

# Which movies did they not rate? (We don't want to recommend movies they've already
user_2_unrated_movies =  user_2_ratings[user_2_ratings.isnull()]

# What are the ratings of these movies the user did not rate?
avg_ratings = pd.concat([user_2_unrated_movies, cluster.mean()], axis=1, join='inner'

# Let's sort by rating so the highest rated movies are presented first
avg_ratings.sort_values(ascending=False)[:20]
```

Out[29]:
```
My Neighbor Totoro (Tonari no Totoro) (1988)                 4.600000
Back to the Future Part II (1989)                            4.600000
Lock, Stock & Two Smoking Barrels (1998)                     4.571429
Harry Potter and the Deathly Hallows: Part 2 (2011)          4.500000
Monty Python and the Holy Grail (1975)                       4.500000
Prestige, The (2006)                                         4.450000
Intouchables (2011)                                          4.450000
Blood Diamond (2006)                                         4.416667
The Imitation Game (2014)                                    4.409091
```

```
One Flew Over the Cuckoo's Nest (1975)                          4.388889
Green Mile, The (1999)                                          4.375000
X-Men: First Class (2011)                                       4.333333
Howl's Moving Castle (Hauru no ugoku shiro) (2004)              4.333333
King's Speech, The (2010)                                       4.333333
Wolf of Wall Street, The (2013)                                 4.333333
Departed, The (2006)                                            4.323529
Equilibrium (2002)                                              4.300000
Harry Potter and the Half-Blood Prince (2009)                   4.300000
Snatch (2000)                                                   4.285714
Léon: The Professional (a.k.a. The Professional) (Léon) (1994)  4.277778
Name: 0, dtype: float64
```

And these are our top 20 recommendations to the user!

1.8.1 Quiz:

- If the cluster had a movie with only one rating. And that rating was 5 stars. What would the average rating of the cluster for that movie be? How does that effect our simple recommendation engine? How would you tweak the recommender to address this issue?

1.9 More on Collaborative Filtering

- This is a simplistic recommendation engine that shows the most basic idea of "collaborative filtering". There are many heuristics and methods to improve it. The Netflix Prize tried to push the envelope in this area by offering a prize of US$1,000,000 to the recommendation algorithm that shows the most improvement over Netflix's own recommendation algorithm.
- That prize was granted in 2009 to a team called "BellKor's Pragmatic Chaos". This paper shows their approach which employed an ensemble of a large number of methods.
- Netflix did not end up using this $1,000,000 algorithm because their switch to streaming gave them a dataset that's much larger than just movie ratings -- what searches did the user make? What other movies did the user sample in this session? Did they start watching a movie then stop and switch to a different movie? These new data points offered a lot more clues than the ratings alone.

1.10 Take it Further

- This notebook showed user-level recommendations. We can actually use the almost exact code to do item-level recommendations. These are recommendations like Amazon's "Customers who bought (or viewed or liked) this item also bought (or viewed or liked)". These would be recommendations we can show on each movie's page in our app. To do this, we simple transpose the dataset to be in the shape of Movies X Users, and then cluster the movies (rather than the users) based on the correlation of their ratings.
- We used the smallest of the datasets Movie Lens puts out. It has 100,000 ratings. If you want to dig deeper in movie rating exploration, you can look at their Full dataset containing 24 million ratings.