Objective:

Create a recommendation system using a Content Based Model.

Get the data

In [1]:
```
!wget -O moviedataset.zip https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
```

In [3]:
```
import pandas as pd
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
#Storing the movie information into a pandas dataframe
movies_df = pd.read_csv('movies.csv')
#Storing the user information into a pandas dataframe
ratings_df = pd.read_csv('ratings.csv')
```

In [5]:
```
#Head is a function that gets the first N rows of a dataframe. N's default is 5.
movies_df.head()
```

Out[5]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Preprocessing

In [6]:
```
#Using regular expressions to find a year stored between parentheses
#We specify the parantheses so we don't conflict with movies that have years in their t
movies_df['year'] = movies_df.title.str.extract('(\(\d\d\d\d\))',expand=False)
#Removing the parentheses
movies_df['year'] = movies_df.year.str.extract('(\d\d\d\d)',expand=False)
#Removing the years from the 'title' column
movies_df['title'] = movies_df.title.str.replace('(\(\d\d\d\d\))', '')
#Applying the strip function to get rid of any ending whitespace characters that may ha
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
```

In [7]:
```
movies_df.head()
```

Out[7]:

	movieId	title	genres	year
0	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995
1	2	Jumanji	Adventure|Children|Fantasy	1995

	movieId	title	genres	year
2	3	Grumpier Old Men	Comedy|Romance	1995
3	4	Waiting to Exhale	Comedy|Drama|Romance	1995
4	5	Father of the Bride Part II	Comedy	1995

In [8]:
```
#Dropping the genres column
movies_df = movies_df.drop('genres', 1)
```

In [9]:
```
movies_df.head()
```

Out[9]:

	movieId	title	year
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

In [10]:
```
# Looking at ratings
ratings_df.head()
```

Out[10]:

	userId	movieId	rating	timestamp
0	1	169	2.5	1204927694
1	1	2471	3.0	1204927438
2	1	48516	5.0	1204927435
3	2	2571	3.5	1436165433
4	2	109487	4.0	1436165496

In [11]:
```
#Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', 1)

ratings_df.head()
```

Out[11]:

	userId	movieId	rating
0	1	169	2.5
1	1	2471	3.0
2	1	48516	5.0
3	2	2571	3.5
4	2	109487	4.0

Collaborative Filtering

The process for creating a User Based recommendation system is as follows:

- Select a user with the movies the user has watched
- Based on his rating to movies, find the top X neighbours
- Get the watched movie record of the user for each neighbour.
- Calculate a similarity score using some formula
- Recommend the items with the highest score

In [12]:

```
userInput = [
            {'title':'Breakfast Club, The', 'rating':5},
            {'title':'Toy Story', 'rating':3.5},
            {'title':'Jumanji', 'rating':2},
            {'title':"Pulp Fiction", 'rating':5},
            {'title':'Akira', 'rating':4.5}
         ]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

Out[12]:

	title	rating
0	Breakfast Club, The	5.0
1	Toy Story	3.5
2	Jumanji	2.0
3	Pulp Fiction	5.0
4	Akira	4.5

In [13]:

```
#Filtering out the movies by title
inputId = movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]\
#Then merging it so we can get the movieId. It's implicitly merging it by title.
inputMovies = pd.merge(inputId, inputMovies)
#Dropping information we won't use from the input dataframe
inputMovies = inputMovies.drop('year', 1)
#Final input dataframe
inputMovies
```

Out[13]:

	movieId	title	rating
0	1	Toy Story	3.5
1	2	Jumanji	2.0
2	296	Pulp Fiction	5.0
3	1274	Akira	4.5
4	1968	Breakfast Club, The	5.0

The users who has seen the same movies

Now with the movie ID's in our input, we can now get the subset of users that have watched and reviewed the movies in our input.

In [14]:

```
#Filtering out users that have watched movies that the input has watched and storing it
userSubset = ratings_df[ratings_df['movieId'].isin(inputMovies['movieId'].tolist())]
```

```
userSubset.head()
```

Out[14]:

	userId	movieId	rating
19	4	296	4.0
441	12	1968	3.0
479	13	2	2.0
531	13	1274	5.0
681	14	296	2.0

In [15]:

```
#Groupby creates several sub dataframes where they all have the same value in the colum
userSubsetGroup = userSubset.groupby(['userId'])
```

In [17]:

```
# the one with userID=1130
userSubsetGroup.get_group(1130)
```

Out[17]:

	userId	movieId	rating
104167	1130	1	0.5
104168	1130	2	4.0
104214	1130	296	4.0
104363	1130	1274	4.5
104443	1130	1968	4.5

In [18]:

```
#Sorting it so users with movie most in common with the input will have priority
userSubsetGroup = sorted(userSubsetGroup,  key=lambda x: len(x[1]), reverse=True)
```

In [19]:

```
userSubsetGroup[0:5]
```

Out[19]:

```
[(75,
        userId  movieId  rating
  7507      75        1     5.0
  7508      75        2     3.5
  7540      75      296     5.0
  7633      75     1274     4.5
  7673      75     1968     5.0),
 (106,
        userId  movieId  rating
  9083     106        1     2.5
  9084     106        2     3.0
  9115     106      296     3.5
  9198     106     1274     3.0
  9238     106     1968     3.5),
 (686,
         userId  movieId  rating
  61336     686        1     4.0
  61337     686        2     3.0
  61377     686      296     4.0
  61478     686     1274     4.0
  61569     686     1968     5.0),
 (815,
         userId  movieId  rating
  73747     815        1     4.5
```

```
 73748     815        2     3.0
 73922     815      296     5.0
 74362     815     1274     3.0
 74678     815     1968     4.5),
(1040,
        userId  movieId  rating
 96689    1040        1     3.0
 96690    1040        2     1.5
 96733    1040      296     3.5
 96859    1040     1274     3.0
 96922    1040     1968     4.0)]
```

Similarity of users to input user

Next, we're going to find out how similar each user is to the input through the **Pearson Correlation Coefficient**. It is used to measure the strength of a linear association between two variables. The formula for finding this coefficient between sets X and Y with N values can be seen in the image below.

Why Pearson Correlation?

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

The values given by the formula vary from r = -1 to r = 1, where 1 forms a direct correlation between the two entities (it means a perfect positive correlation) and -1 forms a perfect negative correlation.

In our case, a 1 means that the two users have similar tastes while a -1 means the opposite.

We will select a subset of users to iterate through. This limit is imposed because we don't want to waste too much time going through every single user.

In [20]:
```
userSubsetGroup = userSubsetGroup[0:100]
```

In [21]:
```
#Store the Pearson Correlation in a dictionary, where the key is the user Id and the va
pearsonCorrelationDict = {}

#For every user group in our subset
for name, group in userSubsetGroup:
    #Let's start by sorting the input and current user group so the values aren't mixed
    group = group.sort_values(by='movieId')
    inputMovies = inputMovies.sort_values(by='movieId')
    #Get the N for the formula
    nRatings = len(group)
    #Get the review scores for the movies that they both have in common
    temp_df = inputMovies[inputMovies['movieId'].isin(group['movieId'].tolist())]
    #And then store them in a temporary buffer variable in a list format to facilitate
    tempRatingList = temp_df['rating'].tolist()
    #Let's also put the current user group reviews in a list format
    tempGroupList = group['rating'].tolist()
    #Now let's calculate the pearson correlation between two users, so called, x and y
    Sxx = sum([i**2 for i in tempRatingList]) - pow(sum(tempRatingList),2)/float(nRatin
    Syy = sum([i**2 for i in tempGroupList]) - pow(sum(tempGroupList),2)/float(nRatings
    Sxy = sum( i*j for i, j in zip(tempRatingList, tempGroupList)) - sum(tempRatingList
```

```
    #If the denominator is different than zero, then divide, else, 0 correlation.
    if Sxx != 0 and Syy != 0:
        pearsonCorrelationDict[name] = Sxy/sqrt(Sxx*Syy)
    else:
        pearsonCorrelationDict[name] = 0
```

In [22]:
```
pearsonCorrelationDict.items()
```

Out[22]:
```
dict_items([(75, 0.8272781516947562), (106, 0.5860090386731182), (686, 0.832050294337843
7), (815, 0.5765566601970551), (1040, 0.9434563530497265), (1130, 0.2891574659831201),
(1502, 0.8770580193070299), (1599, 0.4385290096535153), (1625, 0.716114874039432), (195
0, 0.179028718509858), (2065, 0.4385290096535153), (2128, 0.5860090386731196), (2432, 0.
1386750490563073), (2791, 0.8770580193070299), (2839, 0.8204126541423674), (2948, -0.117
20180773462392), (3025, 0.45124262819713973), (3040, 0.89514359254929), (3186, 0.6784622
064861935), (3271, 0.26989594817970664), (3429, 0.0), (3734, -0.15041420939904673), (409
9, 0.05860090386731196), (4208, 0.29417420270727607), (4282, -0.4385290096535115), (429
2, 0.6564386345361464), (4415, -0.11183835382312353), (4586, -0.9024852563942795), (472
5, -0.08006407690254357), (4818, 0.4885967564883424), (5104, 0.7674257668936507), (5165,
-0.4385290096535153), (5547, 0.17200522903844556), (6082, -0.04728779924109591), (6207,
0.9615384615384616), (6366, 0.6577935144802716), (6482, 0.0), (6530, -0.351605423203870
9), (7235, 0.6981407669689391), (7403, 0.11720180773462363), (7641, 0.7161148740394331),
(7996, 0.626600514784504), (8008, -0.22562131409856986), (8086, 0.6933752452815365), (82
45, 0.0), (8572, 0.8600261451922278), (8675, 0.5370861555295773), (9101, -0.086002614519
22278), (9358, 0.692178738358485), (9663, 0.193972725041952), (9994, 0.503027272865958
7), (10248, -0.24806946917841693), (10315, 0.537086155529574), (10368, 0.468807230938494
5), (10607, 0.41602514716892186), (10707, 0.9615384615384616), (10863, 0.602018301634559
5), (11314, 0.8204126541423654), (11399, 0.517260600111872), (11769, 0.937614461876991
4), (11827, 0.4902903378454601), (12069, 0.0), (12120, 0.9292940047327363), (12211, 0.86
00261451922278), (12325, 0.9616783115081544), (12916, 0.5860090386731196), (12921, 0.661
1073566849309), (13053, 0.9607689228305227), (13142, 0.6016568375961863), (13260, 0.7844
645405527362), (13366, 0.8951435925492911), (13768, 0.8770580193070289), (13888, 0.25087
26030021272), (13923, 0.3516054232038718), (13934, 0.17200522903844556), (14529, 0.74179
01772340937), (14551, 0.537086155529574), (14588, 0.21926450482675766), (14984, 0.716114
874039432), (15137, 0.5860090386731196), (15157, 0.9035841064985974), (15466, 0.72057669
21228921), (15670, 0.516015687115336), (15834, 0.22562131409856986), (16292, 0.657793514
4802716), (16456, 0.7161148740394331), (16506, 0.5481612620668942), (17246, 0.4803844614
1526137), (17438, 0.7093169886164387), (17501, 0.8168748513121271), (17502, 0.8272781516
947562), (17666, 0.7689238340176859), (17735, 0.7042381820123422), (17742, 0.39223227027
63681), (17757, 0.64657575013984), (17854, 0.537086155529574), (17897, 0.877058019307028
9), (17944, 0.2713848825944774), (18301, 0.29838119751643016), (18509, 0.132221471336986
2)])
```

In [23]:
```
pearsonDF = pd.DataFrame.from_dict(pearsonCorrelationDict, orient='index')
pearsonDF.columns = ['similarityIndex']
pearsonDF['userId'] = pearsonDF.index
pearsonDF.index = range(len(pearsonDF))
pearsonDF.head()
```

Out[23]:

	similarityIndex	userId
0	0.827278	75
1	0.586009	106
2	0.832050	686
3	0.576557	815
4	0.943456	1040

The top x similar users to input user

Now let's get the top 50 users that are most similar to the input.

In [24]:
```
topUsers=pearsonDF.sort_values(by='similarityIndex', ascending=False)[0:50]
topUsers.head()
```

Out[24]:

	similarityIndex	userId
64	0.961678	12325
34	0.961538	6207
55	0.961538	10707
67	0.960769	13053
4	0.943456	1040

let's start recommending movies to the input user.

Rating of selected users to all movies

We're going to do this by taking the weighted average of the ratings of the movies using the Pearson Correlation as the weight. But to do this, we first need to get the movies watched by the users in our **pearsonDF** from the ratings dataframe and then store their correlation in a new column called _similarityIndex". This is achieved below by merging of these two tables.

In [25]:
```
topUsersRating=topUsers.merge(ratings_df, left_on='userId', right_on='userId', how='inn
topUsersRating.head()
```

Out[25]:

	similarityIndex	userId	movieId	rating
0	0.961678	12325	1	3.5
1	0.961678	12325	2	1.5
2	0.961678	12325	3	3.0
3	0.961678	12325	5	0.5
4	0.961678	12325	6	2.5

Now all we need to do is simply multiply the movie rating by its weight (The similarity index), then sum up the new ratings and divide it by the sum of the weights.

We can easily do this by simply multiplying two columns, then grouping up the dataframe by movieId and then dividing two columns:

It shows the idea of all similar users to candidate movies for the input user:

In [26]:
```
#Multiplies the similarity by the user's ratings
topUsersRating['weightedRating'] = topUsersRating['similarityIndex']*topUsersRating['ra
topUsersRating.head()
```

Out[26]:

	similarityIndex	userId	movieId	rating	weightedRating
0	0.961678	12325	1	3.5	3.365874
1	0.961678	12325	2	1.5	1.442517

	similarityIndex	userId	movieId	rating	weightedRating
2	0.961678	12325	3	3.0	2.885035
3	0.961678	12325	5	0.5	0.480839
4	0.961678	12325	6	2.5	2.404196

In [28]:
```
#Applies a sum to the topUsers after grouping it up by userId
tempTopUsersRating = topUsersRating.groupby('movieId').sum()[['similarityIndex','weight
tempTopUsersRating.columns = ['sum_similarityIndex','sum_weightedRating']
tempTopUsersRating.head()
```

Out[28]:

	sum_similarityIndex	sum_weightedRating
movieId		
1	38.376281	140.800834
2	38.376281	96.656745
3	10.253981	27.254477
4	0.929294	2.787882
5	11.723262	27.151751

In [30]:
```
#Creates an empty dataframe
recommendation_df = pd.DataFrame()
#Now we take the weighted average
recommendation_df['weighted average recommendation score'] = tempTopUsersRating['sum_we
recommendation_df['movieId'] = tempTopUsersRating.index
recommendation_df.head()
```

Out[30]:

	weighted average recommendation score	movieId
movieId		
1	3.668955	1
2	2.518658	2
3	2.657941	3
4	3.000000	4
5	2.316058	5

Now let's sort it and see the top 20 movies that the algorithm recommended!

In [31]:
```
recommendation_df = recommendation_df.sort_values(by='weighted average recommendation s
recommendation_df.head(10)
```

Out[31]:

	weighted average recommendation score	movieId
movieId		
5073	5.0	5073
3329	5.0	3329

	weighted average recommendation score	movieId
movieId		
2284	5.0	2284
26801	5.0	26801
6776	5.0	6776
6672	5.0	6672
3759	5.0	3759
3769	5.0	3769
3775	5.0	3775
90531	5.0	90531

In [32]:

```
movies_df.loc[movies_df['movieId'].isin(recommendation_df.head(10)['movieId'].tolist())]
```

Out[32]:

	movieId	title	year
2200	2284	Bandit Queen	1994
3243	3329	Year My Voice Broke, The	1987
3669	3759	Fun and Fancy Free	1947
3679	3769	Thunderbolt and Lightfoot	1974
3685	3775	Make Mine Music	1946
4978	5073	Son's Room, The (Stanza del figlio, La)	2001
6563	6672	War Photographer	2001
6667	6776	Lagaan: Once Upon a Time in India	2001
9064	26801	Dragon Inn (Sun lung moon hak chan)	1992
18106	90531	Shame	2011