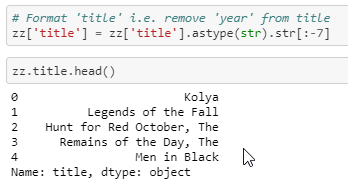
**1 - Data Pre-Processing**

**movie[‘title’]:**

****

**Interpretation:** The movie title also has the year included.

Following code-snippet demonstrates the updated column names.

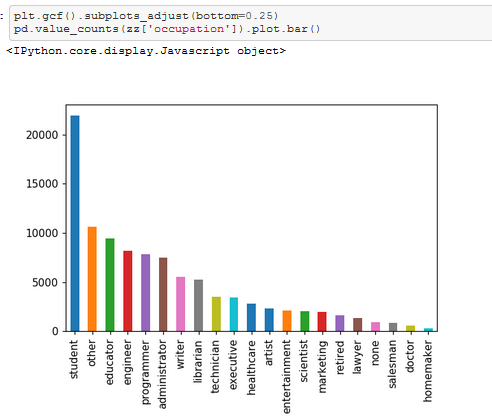


**Interpretation:** The attributes (column names) by default are self-explanatory. However, some of these are renamed to make it less confusing.

**2 – Exploratory Data Analysis**

**2.1 Univariate Analysis**

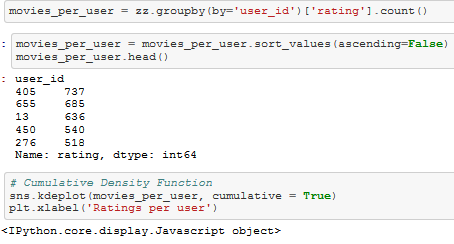
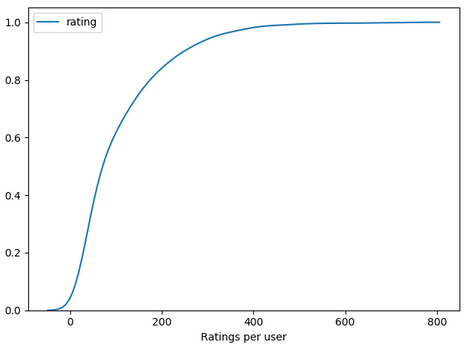
**‘occupation’:**



**Interpretation:** Highest number of users are students.

**2.2 Bivariate Analysis**

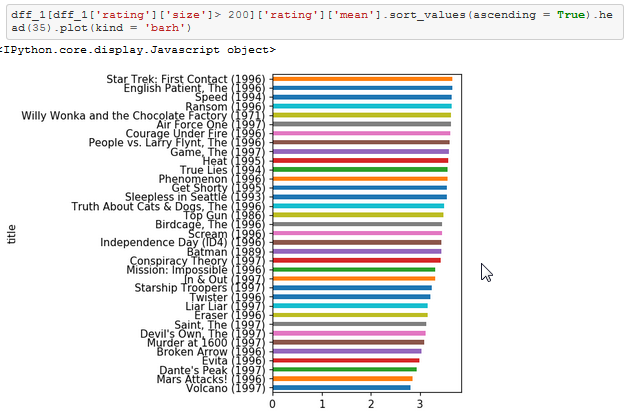
**Ratings vs User - Cumulative Density Function**

**** 

**Interpretation:** 82% of the users have made less than 200 ratings while 18% of the users have rated more than 200 of them.

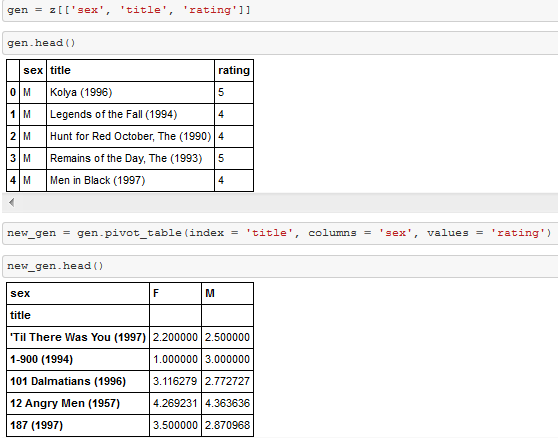
**High rated movies (by rating)**

Visual representation of highly rated movies.

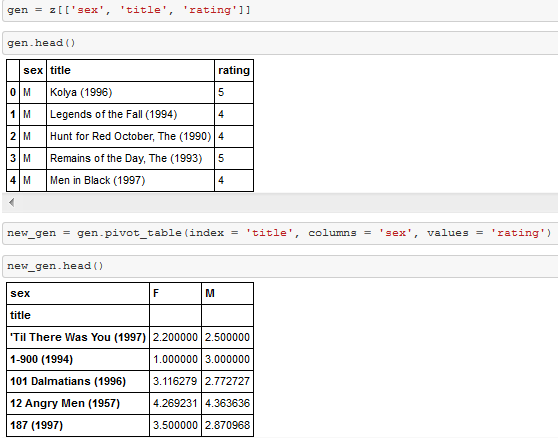


**Gender vs Rating vs Title**

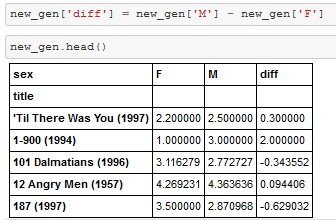
Gen dataframe has ‘sex’, ‘title’ and ‘rating’



We pivot the dataframe with title as index, sex as columns and fill values with rating.

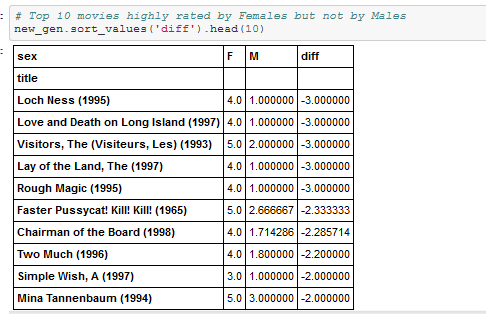
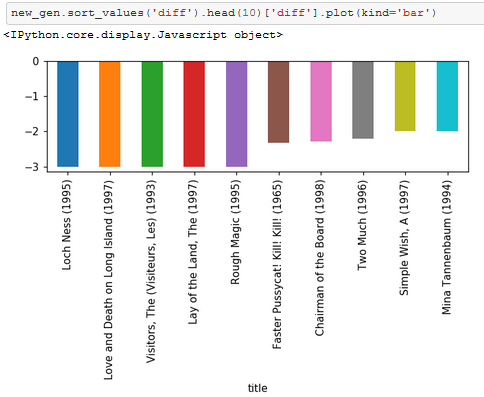


Now that we have a pivot table with average male and female ratings for each movie, we can go ahead and calculate their difference to find any interesting patterns in movie selection.



**# Top 10 movies highly rated by Females but not by Males**

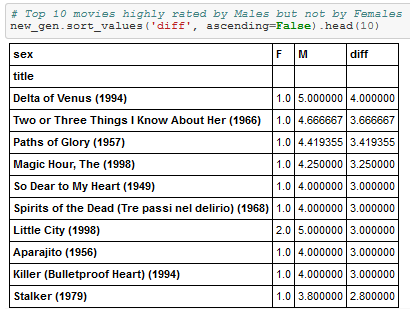
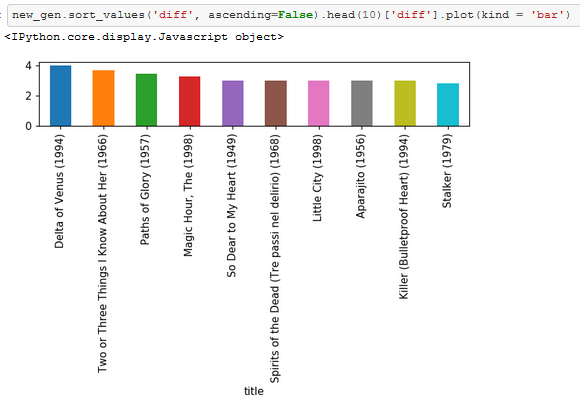
Negative values represent that females rated the movies higher than males on an average.

**Interpretation:** We see that ‘Loch Ness’, ‘Love Death and Long Island’ are among the movies that have been rated highly by females than that of males.

**# Top 10 movies highly rated by Males but not by Females**

Positive values represent that females rated the movies higher than males on an average.

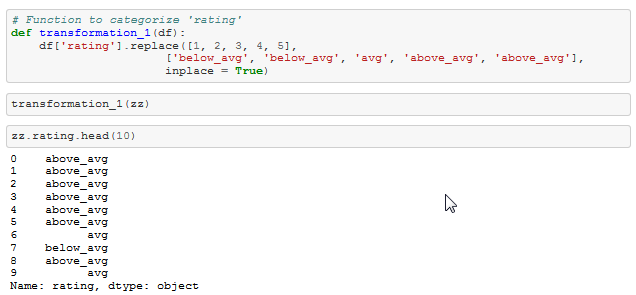
 

**Interpretation:** We see that ‘Loch Ness’, ‘Love Death and Long Island’ are among the movies that have been rated highly by females than that of males.

**3 – Transformations**

**Replacing ratings with below\_avg, avg and above\_avg:**

Ratings 1, 2 are replaced by ‘below\_average’, while 3 is replaced as ‘average’ and 4, 5 are categorized as ‘above\_average’.



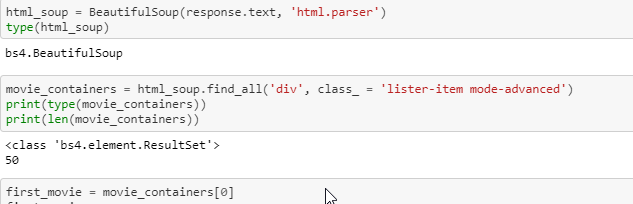
**4 - Web Scraping**

**4.1 Beautiful Soup:**

Using Python's Beautiful Soup to get data from IMDB's Top 150 movies

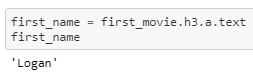


I use html parser to convert html text into beautiful soup object.

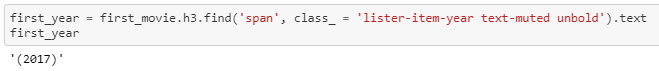


This returns a prettified version of html text.

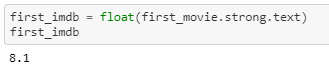
**movie title:**



**movie year:**

****

**imdb rating:**

****

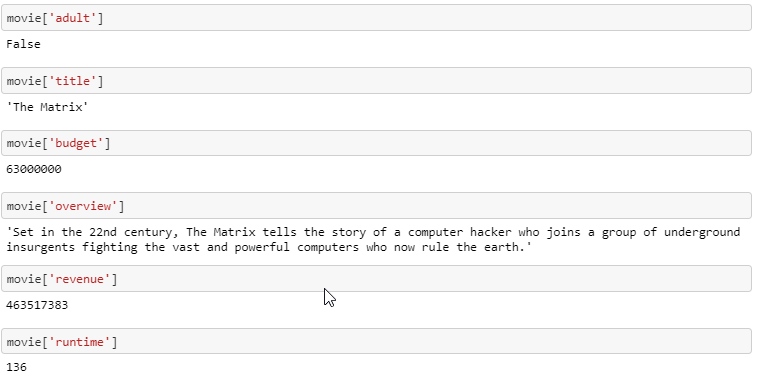
**Cons:** This approach seems tedious and computationally expensive. Also, this requires revisiting the IMDB website once for every request.

**4.2 Tmdbsimple:**

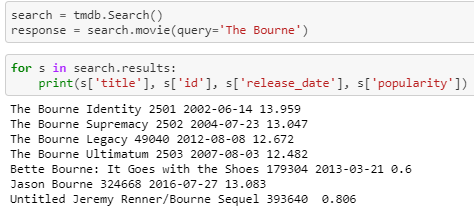
Importing ‘tmdbsimple' and key in the credentials

****

**Extracting Movie Attributes:**

****

Another way of accessing movie data is by passing the movie name to the argument ‘query’.



**Note:**When we use tmdb.search() we do get the tmdb\_id as well as the title. But using tmdb.Movies() yields much more information about the movie.

**New Approach:**

We can query TMDB API only using movie\_ids and not by movie titles. (When queried, API throws a 404 Cleint Error) and also takes longer time to that of movie\_id. However, Movie Lens dataset has its own movie\_id which are quite different from that of TMDBs (tmdb\_id)

Hence, we use the following approach:

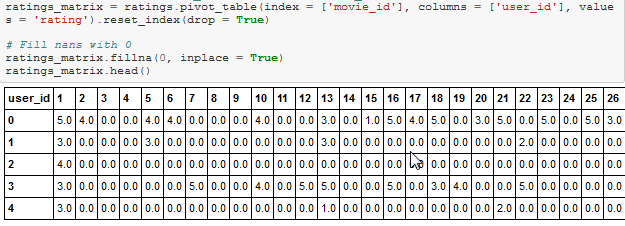
* Get the movielens\_id and title from movielens dataset
* Query TMDB API using movie title to get TMDB\_IDs
* Use queried tmdb\_id to get additional info about the movie

Based on this approach I web scrape using TMDB simple and get the metadata of the movie titles matching from movielens data.

**5. Popularity Based Recommendation**

**Simple Recommendation System (Popularity based - Ratings)**

Ratings matrix with movie\_id as columns and user\_id as rows and ratings as values



The above matrix has:

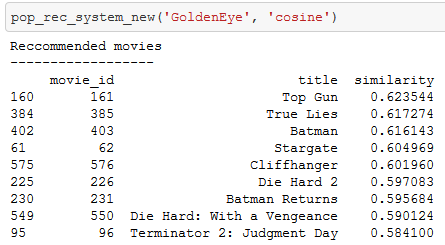
Rows – Users  
Columns – Movies  
Values – Ratings



The above code calculates pairwise distances using various metrics to return movies.

**Cosine Similarity:**

Results for the movie ‘Golden Eye’ using cosine similarity as a metric.



**Note:** This recommendation system is solely based on popularity. The movies returned with cosine, euclidean and manhattan distance are quite similar to each other. However, they are not so much when the recommendation system uses pearson correlation.

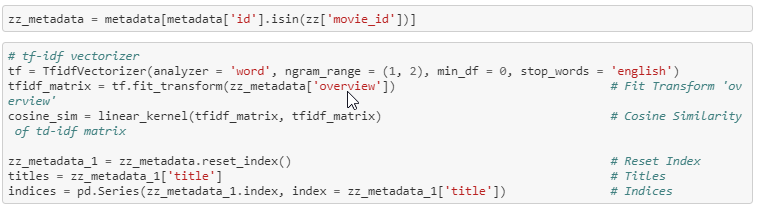
**Limitation:** This recommendation system suggests movies IRRSPECTIVE OF USER PREFERENCES.

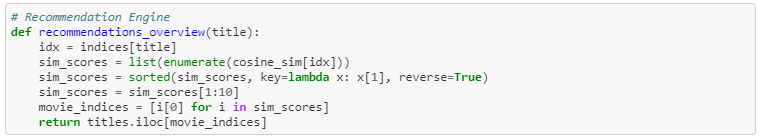
**6 – Content Based Recommendation**

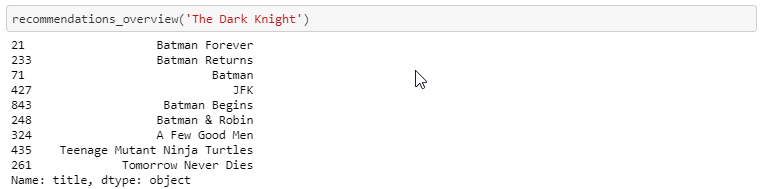
**6.1 Description Based Recommendation:**

**6.1.1 Recommendation Engine using ‘overview’:**

First recommendation engine considers only the ‘overview’ of the movie. ‘Overview’ stands for the descriptive text that is outlined for a movie in ‘IMDB’ official site.







**Interpretation:** This model provides robust recommendations using metadata ['overview'].

**Limitation:** But there are few not-so meaningful recommendations. Example: (Teenage Mutant Ninja Turtles, Tomorrow Never Dies)

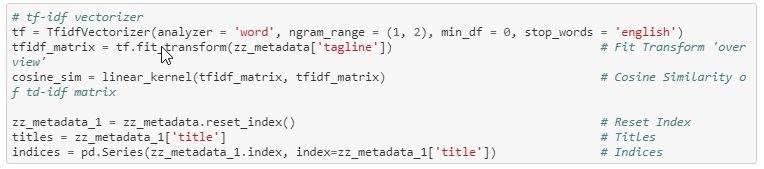
**6.1.2 Recommendation Engine using ‘tagline’:**

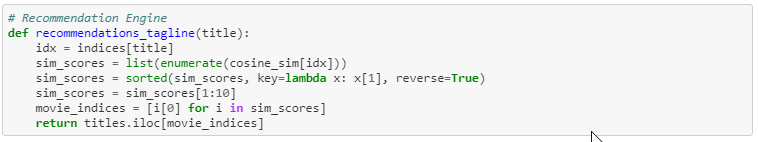
Second recommendation engine considers only the ‘tagline’ of a movie. ‘Tagline’ stands for the extended movie title which certain movies have.

Example: ‘Die Hard 3: With a Vengeance’

Title of the movie is ‘Die Hard 3’ while the tagline is ‘With a Vengeance’.







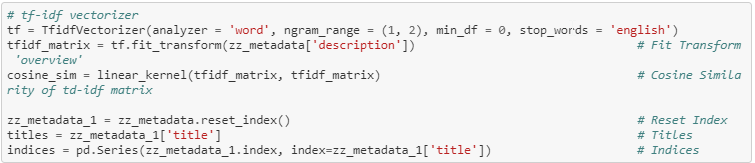


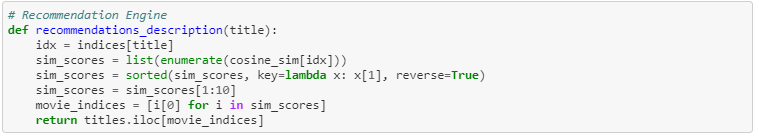
**Interpretation:** The model built with respect to 'tagline' is not as robust as the previous model. It is apparent that the first model (using metadata ['overview']) provides highly similar movies than the model using 'taglines'.

**6.1.3 Recommendation Engine using metadata ['overview'] + metadata ['tagline']:**

Final recommendation engine using description considers both the ‘overview’ and the ‘tagline’ of a movie. These two columns are concatenated to form a new column ‘description’.









**Interpretation:** This model provides similar recommendations to that of the initial model (using metadata ['overview']). We can infer that 'tagline' is not the best feature to consider building a recommendation system.

**6.2 Metadata Based Recommendation System**

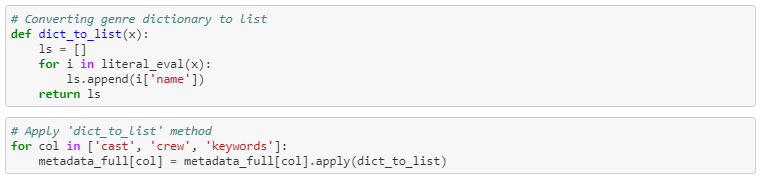
After scraping data from the web for the movie ids in the merged dataframe (‘movielens’), we can now use the metadata to build the recommendation system.



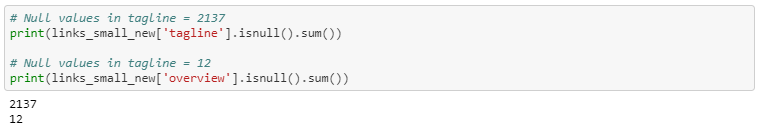
Movie lens data has a file ‘links’ that consists of ‘movie id’, ‘imdb id’ and ‘tmdb id’ using which the data was scraped from the web.

**Missing Values:**

The metadata has column values in dictionary. This can be trickier to handle. Instead of using the dictionary to operate on, I convert the dictionary to a list.

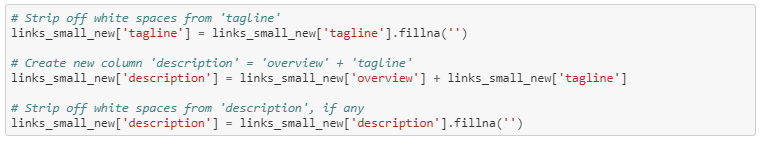


Checking for null values in ‘tagline’ and ‘overview’:



**Note:** Since there are null values in 'tagline' and 'overview', we cannot simply join them together to create a new column ('description').

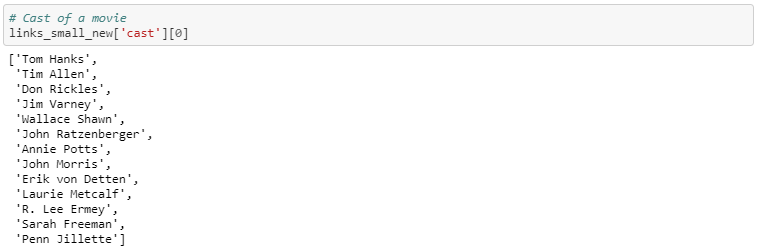
**Solution:** Strip off the white spaces.



**Note:** So far, links\_small\_new has cast, crew, credits and genres. But we do not need all the data in them. To efficiently use them, I clean each column further.

Creating new columns ‘cast\_size’ and ‘crew\_size’:





**Note:** Cast can include actors and actress that are both famous and infamous. However, famous artists are most likely to play a significant role in affecting the user’s opinion than others.

**Solution:** Select 4 artists [lead actor 1, lead actor 2, supporting actor 1, supporting actor 2] rather than considering all.

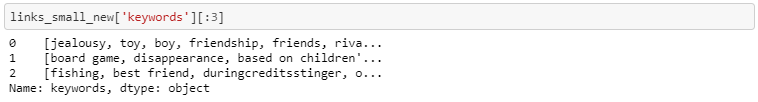
These are steps I follow in the preparation of genres and credits data:

1. **Strip Spaces and Convert to Lowercase** from all our features. This way, engine will not confuse between **Johnny Depp** and **Johnny Galecki.**
2. **Mention Director 2 times** to give it more weight relative to the entire cast.

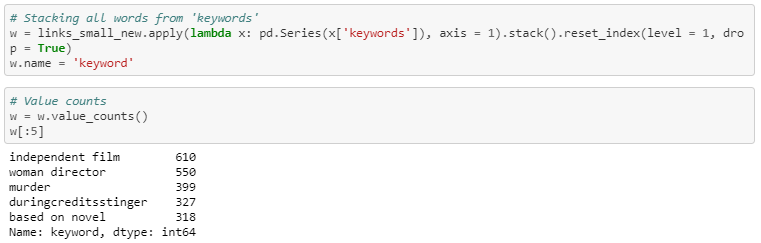


**Keywords:**

We will do a small amount of pre-processing of our keywords before putting them to any use. As a first step, we calculate the frequency counts of every keyword that appears in the dataset.



Not all words could prove significant.



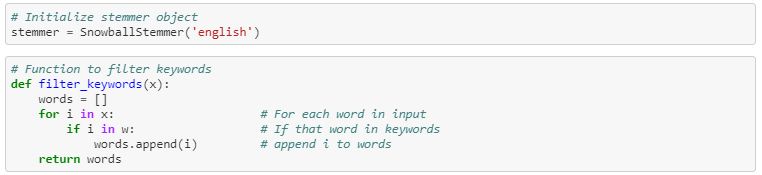
**Note:** Keywords occur in frequencies ranging from 1 to 610. We do not have any use for keywords that occur only once.   
**Interpretation:** Keywords that occur just once.



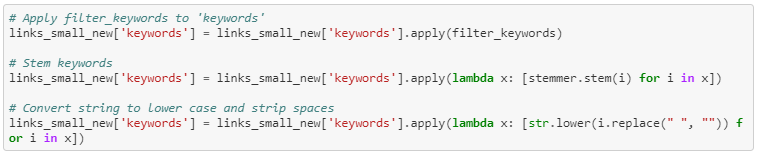
**Stemming:**

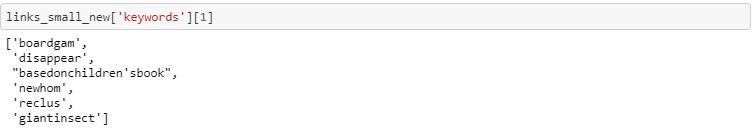
Words like ‘play’, ‘played’ and ‘playing’ can be stemmed to the word ‘play’. This process is called stemming.

Code to perform stemming.



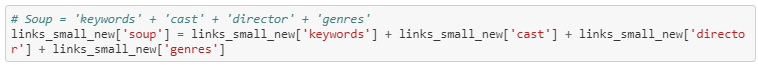
**Preprocess ‘keywords’ column:**



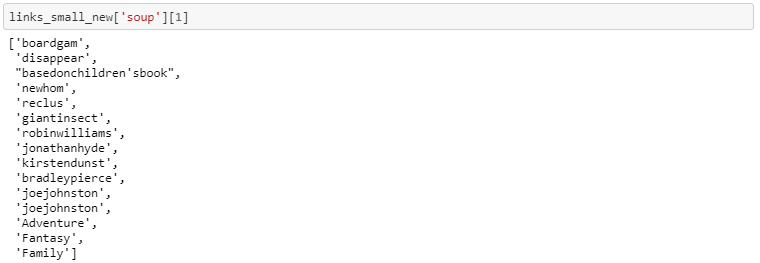


**Soup:**

Soup is the metadata of genres, director, cast and keywords.

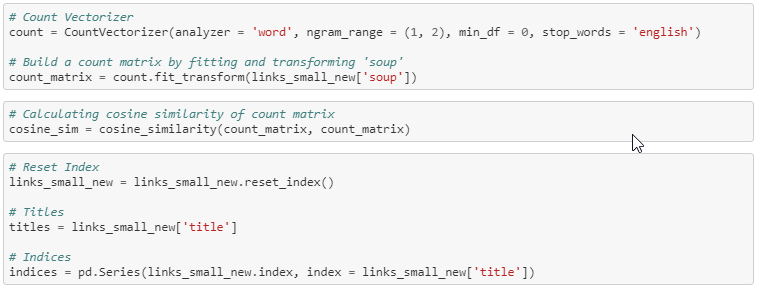


Soup contains genres, director, cast and keywords.

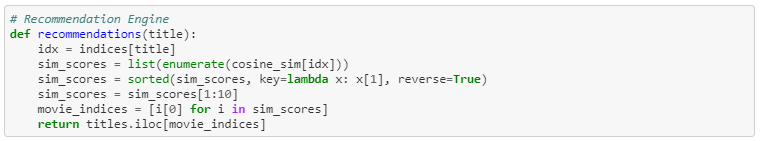


**Count Vectorizer:**

Create a count matrix and calculate the cosine similarities to find movies that are most similar.



Python code for recommendation engine



Recommendations:



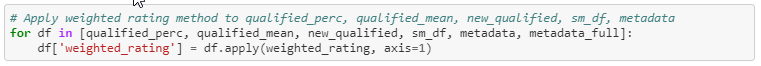
**Limitation:** This recommendation system returns only the movies based on soup. It does not consider popularity.

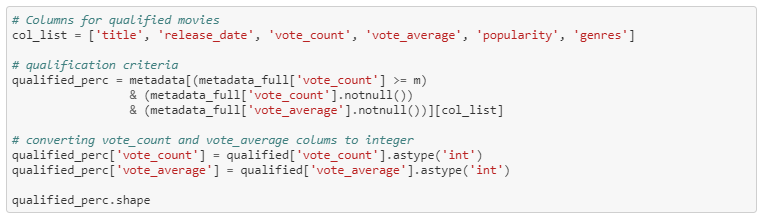
**Solution:** We use the results returned from our Count Vectorizer (indices) and return the movies that are popular based on the IMDB's weighted average. Additionally, I use three different criteria to cut-off the movies (75% percentile, Mean and No Cut-Off criteria)

**Weighted Average:**

Function to calculate weighted average:

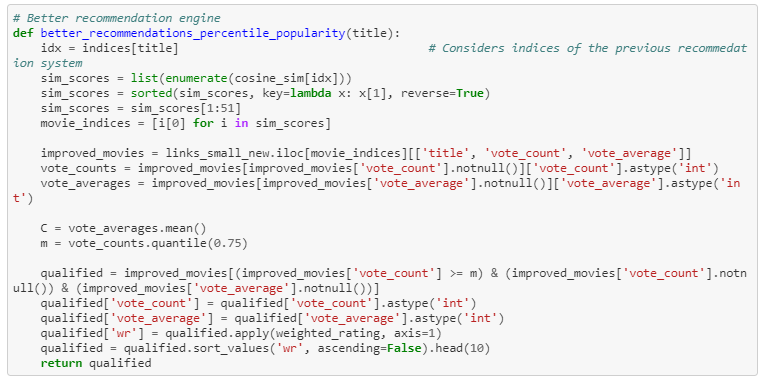




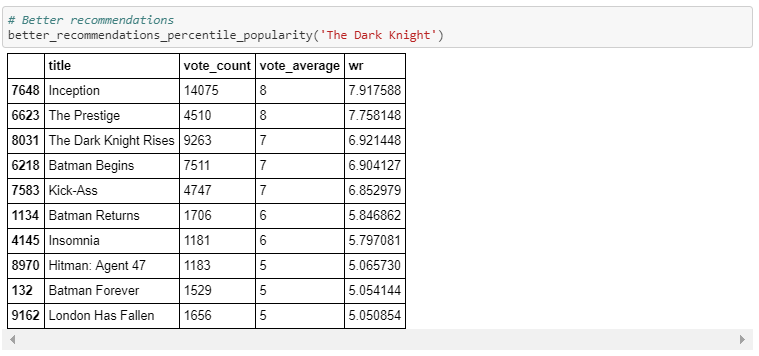


**Getting qualified movies (cutoff: 95%)**

Code for recommendation system with movies cutoff 95%



We see that the movies recommended by the engine highly emphasized on the crew (director).



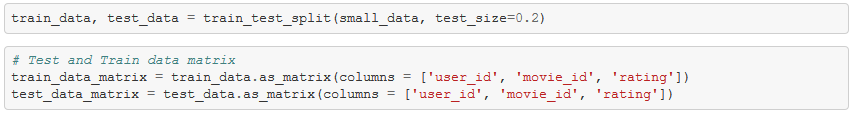
**7 – Collaborative Filtering**

**7.1 Collaborative Filtering:**

I pick only 25% of the data.



Dividing the data into train and test set:



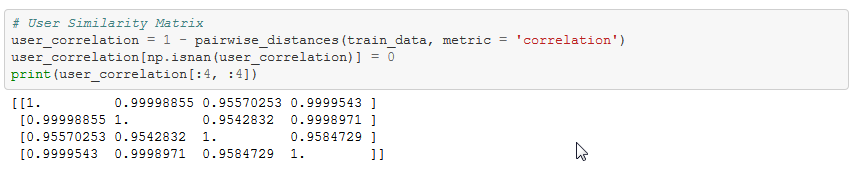
The train and test dataframes are converted to arrays using .as\_matrix()

**Idea behind user and item similarity:**

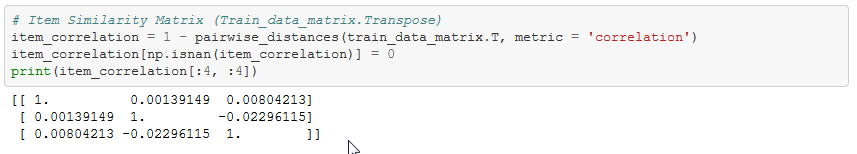
User similarity can be calculated by measuring 'pairwise distances' between ratings datset.

However, if you have to calculate the 'item similarity', we have to transpose the 'ratings' data and then calculate the pairwise distances.

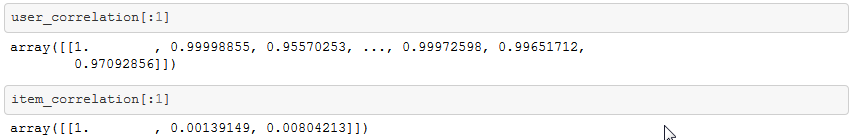
**User Similarity Matrix:**



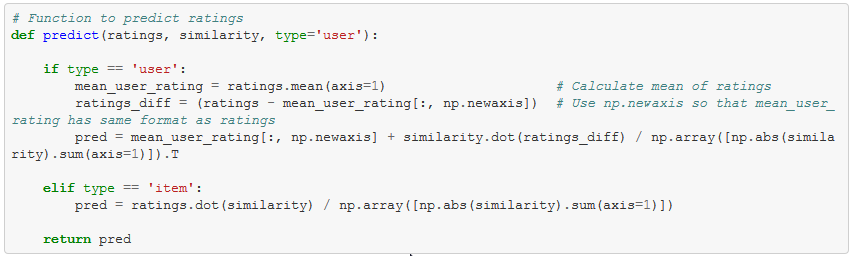
**Item Similarity Matrix:**



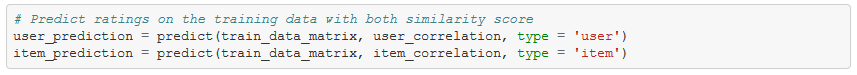
**User Correlation and Item Correlation:**



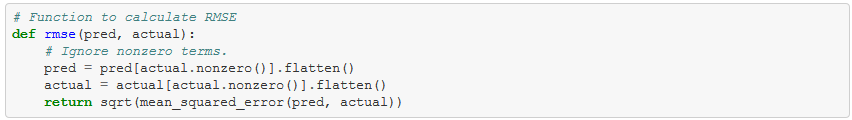
**Function to predict ratings**



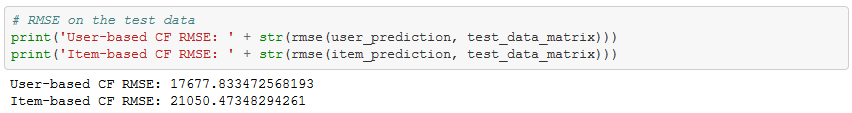
**Calling predict function:**



**Calculate Root Mean Squared Error:**



**Calling RMSE() to calculate error on user based and item based predictions:**



**8. Potential Next Steps:**

**Suggestions for Content-Based filtering from other data scientists I met during the meet-up:**

1. Use weighted average on each movie:
   * How about multiplying rating count and average rating.
   * For a linear column, there can be huge variance. [Try normalize and standardize]
2. Use metadata td-idf matrix (cosine similarity) rather than just the movies.
   * Use 'word2vec'
3. For collaborative filtering - try 'movie-movie' similarity and 'user-user' similarity (Computationally Expensive)
4. Try to build a Hybrid Recommender