Indian Institute of Technology Bombay

Department of Electrical Engineering

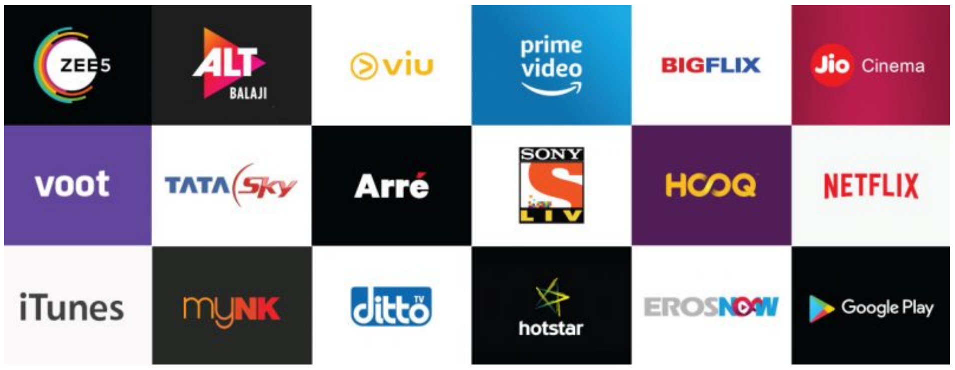
EE769 : Project Report

Movie Recommender System

Team Members:
17D070052 E. Harshvardhan
170070035 Farhan Ali

Course Instructor:
Amit Sethi

June 9, 2020



Abstract

In the recent years, "Netflix and Chill" have become one of the major hobbies for a huge set of people, particularly youngsters. With better broadband spreading around the world, various major OTT platforms in last few years have grown tremendously and have made a lot of shifting in a person's lifestyle. These platforms allow people easiness and present them with different choices at any time and have always been a major source of entertainment to them. One of the major techniques used by these streaming sites to keep one involved for more and more time is by recommending them the media that would particularly interest them. This hadn't been an easy task because it requires a lot of learning and data from all different sort of people around. But Machine Learning obviously made it possible as it is consequent from the fact that many people have been happily spending their major amount of time there today

1 Overview of the project

In this project we have implemented a method for recommending further movies to a user based on the movie they select to watch at a particular time. This is based on the idea of several movie/web-series streaming sites as when a person clicks a media to play, what suggestions should be shown to the user for their next destination. We have kept our method independent of users' history or preferences and show recommendations based only on the features for selected movie. For this purpose, we have used the cosine similarity between the input and all available movies to us and give recommendations according to the matching points between any two movies.

2 Methodology

We have used a huge data-set of 45,000 movies from MovieLens which we downloaded from kaggle. Data points are used from the set which include 'cast', 'director', 'keywords', 'genres', 'production companies', 'belongs to collection', 'popularity', 'vote_count' and 'vote_average'. These features constitute details of a movie that play a crucial role for content based recommendation of further movies to any user. First we did some preprocessing on the above mentioned data and then we seperated the content dependent and user dependent. Content dependencies consist of 'cast', 'director', 'keywords', 'genres', 'production companies' and 'belongs to collection' whereas user dependencies consist of 'popularity', 'vote_count' and 'vote_average'.
We considered 1 director,3 casts, all keywords, all genres ,all production companies and all belongs to collection and made a sentence of all the metadata consisting these. Then all these sentences of all movies are count vectorized to obtain a count vectorizer matrix(count_matrix) of all these words. This vector matrix has dimension of (no_of_movies , no_of_features_obtained) Then for which ever movie we want to make recommendation we will make a product of the sparse matrix of (1,no_of_features_obtained) which is the row of particular movie with the count_matrix transpose to obtain no of words matched between two movies. The more number of similar features, the more the value is for each column. From this 1D row array each index of array represents a movie index(Its a mapping from array index to movie index) and normalise it after which we enumerate the list and sort it based on the order of value inside it. After that we get the highest value as the movie we searched as we will be having a perfect match of features which we ignore and take the top 50 list.

After that we have user weighted features on our arsenal we will be taking a weighted score of the product of the movie vector with all the values and normalise it. Now we will have a weighted score of the content score and user score to get a final score. Then we sort it again to get the better recommendation based on both scores. The function is customised for content based and user based scores as well as for each of user dependencies(i.e. popularity, vote average, vote count). And to overcome movies with multiple movies with same name we will make a list of pandas series of recommended movies with an overview written so that we can have the movie which we want.

3 Implementation Details

- We import required libraries in our code such as **numpy** for using linear algebra methods, **pandas** for data processing and reading/writing CSV files, **time** to calculate execution times of the code and **preprocessing** from sklearn module for the purpose of min-max normalizations.
- We then import data such as movies' names, keywords and credits from our movies dataset. Next, we use a function to check if there are unusual ids of the movies in our dataset and then we remove them keeping clean ids only in our dataframe.
- Then we visualise the obtained data to realise that data type was string although it didn't appear so in the csv file every time. So, we use inbuilt python function **eval()** for string parsing and convert the string values to **dict** data-types (which we have verified later).
- We normalize float data type columns that includes popularity, vote average and vote count, transform these features after fitting their data type using **preprocessing.MinMaxScaler.fit-transform** function which scale each of these to [0, 1] range and then we visualise them.
- Next we have defined several functions such as:
  - *get director* to obtain director's name from the crew list,
  - *get cast* to return the top three cast names from the cast list,
  - *get list* to get all the keywords, genres and production companies together in **list** data type,

  - *get collection* to get the collection, if any to which movie belongs,
  - *strip* to merge string type data (or all string items in a list-type data) into one string (or string items) by stripping the spaces between the words, and
  - *metadata* to join all the data from dictionary into one single sentence.

- So, overall we obtain director, three top cast, genres, keywords, production companies and "belongs to collection" for the input, strip any spaces if there in all the strings and join individual strings to one big sentence using metadata with spaces in between(all the metadata into a sentence)

- We then create indices for data four frames so that we get the pandas indexes from the index of the array in count vectorizer.(pandas index is different from the index after the values are changed to numpy array)

- Then we create count vector matrix from the metadata sentence to get a large sparse matrix (approx dimension of (45000,100000))

- After that we implemented metadata_recommender function which has arguments in order as follows:
  - title = title of the movie
  - count_matrix = count vector matrix and the default argument will be count vector matrix obtained already.
  - df = the dataframe
  - indices = the index getting function of pandas which takes stripped title of movie and returns pandas index of movie having default argument as the indices runned in previous code.
  - indexing = the index getting function of pandas which takes index of numpy array made from movies data and returns pandas index of movie having default argument as the indices runned in previous code.
  - content_weight = the weight for content based recommendation from stringed feature data with default value as 1.
  - user_weight = the weight for user giver ratings dependent recommendation with default value as 0.01.

  - popularity_weight = if user wants popular movies based on the content this can be customised for that purposes with default value as 1
  - vote_average_weight = if user wants good vote average movies based on the content this can be customised for that purposes with default value as 1
  - vote_count_weight = if user wants a more voted movies based on the content this can be customised for that purposes with default value as 1

- metadata_recommender does find the similarity scores of content and user rated and have a weighted sum and sort in that order. The top 20 contents and the **overview of the movie which it took for recommendation** is displayed (This is enabled because if we have multiple movies based on same name so users can get what movie type they want)

- Finally we have implemented our recommender function which will take movie title in string format and strip it to make into one string with no spaces. We then get index of title in *numpy.int64* if there is only one movie with this name and *panda.series* of *numpy.int64* if there are more than one and proceed as:

  - Since if there is only one *numpy.int64* index, it won't iterate in a loop but converting it into a list can make it possible so we do it using an *if condition*.
  - Then we implement following *for loops* in order to consider more than one indices for the input movie name.
  - In the first loop at the beginning, we make a list of non-string features viz. popularity, vote_average, and vote_count and then store these features for our title in a numpy array **x**, and their corresponding weights in another array **w**. Apart from them, we create another array **y** to keep all the features for matching purposes i.e. to calculate the score. Then, **x** is multiplied with weights' array **w** and we further store dot product of **x** with transpose of features' matrix **y** to record the scores of non-string features in *math_scores* and normalize it. To get the cosine similarity, we calculate the dot product between the *count_matrix* that stores the string features of all the movies with the array of this sparse matrix corresponding to our input title, store it in

sim_scores and normalize it. Now we enumerate the *sim_scores* list to make it as tuple list and sort it into descending order. By the end of this loop, we take top 50 data from *sim_scores* leaving the first one which would correspond to the input movie itself and move further to a nested loop where we calculate weighted sum of two kinds of features viz. Content dependent and user dependent. *user* is kept tune-able here and again all the data are sorted and only top 20 data are considered from the combined list.

  – After getting corresponding indices for best 20 matches in the previous for loop, we append the title for these indices in **movies_list** and finally for all indices corresponding to the input movie name, we print dictionary of movies with it's overview and the list of 20 suggestions for each.

.

4 Challenges and Overcoming

- The data is string data multiple times parsed so it can't be obtained as python datatypes rather it is obtained as strings to over come that we used eval() function of python.

- Using eval() function has a problem if it takes some strings like the output when parsed is os.system('rm -rf /') then it will delete everything inside the computer or system. To overcome these disasters we will have a global dictionary which is passed as second argument for reducing disasters '"__builtins__":None'.

- Title might be differently given as input like small letters etc. To overcome that we created another column of panda series which has stripped titles.

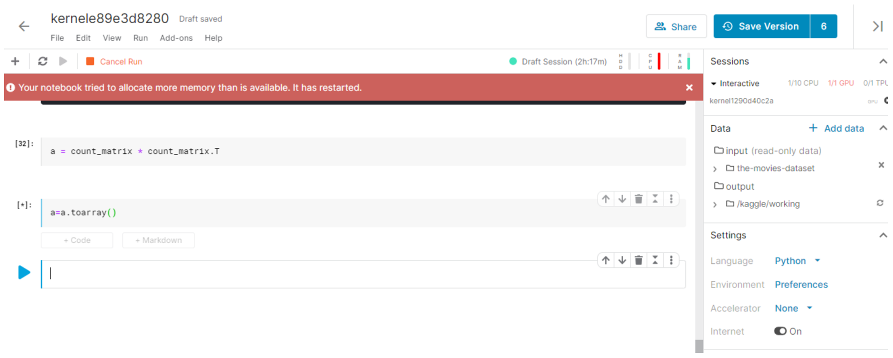
- At first we thought to make a big numpy array obtained by dot product of the count_matrix with itself. The numpy to numpy dot product will not happen at all because its takes lot of time for computing and ram will be more than 16GB(only the matrix costs 15.09GB(approximately) of memory) and computation time takes upto 36hrs for the matrix. Second idea is to match strings with each others strings instead of count matrix to reduce the parameters to check although it reduced the ram problem still it takes 26-36 hrs to make the count matrix. So third idea which overcame all problems is that since count vectorizer gives sparse matrix, we will take one dimension which contains the movie vector and take the similarity scores of that movie with all the other movies in a function which reduces the computational complexity and also space constraint as we made a (1,45000) out of it so we don't use up ram much too. This worked for us and made our recommendation to less than **0.5 second** (results in ipynb file.)

- We might have multiple movies with same name. If it happens we will be getting a panda series of indices of format numpy.int64 for which looping only once doesn't help. So we made a loop for all the values in panda series obtained. Also we made a check in the start whether idxs(a variable in ipynb file) is numpy.int64 or panda.series, for which panda.series can be looped whereas numpy.int64 cant be so we made a list of the same value which is looped only once to avoid problem with loop.
- For the two movies having same name, we distinguished between them by providing overview data.

5 Results and Inferences

The code for above described methodology and implementation is successfully working and here we have listed some results and inferences drawn from them. We have also attached the screenshots of some example inputs and some errors that occurred during the implementation:

- When we were trying to make a big numpy array by taking dot product of the count_matrix with itself, the RAM then getting allocated to our kaggle notebook exceeded it's limit of 16 GB. This caused us to move to further possible methods. The error printed in that case is shown here:



Your notebook tried to allocate more memory than is available. It has restarted.

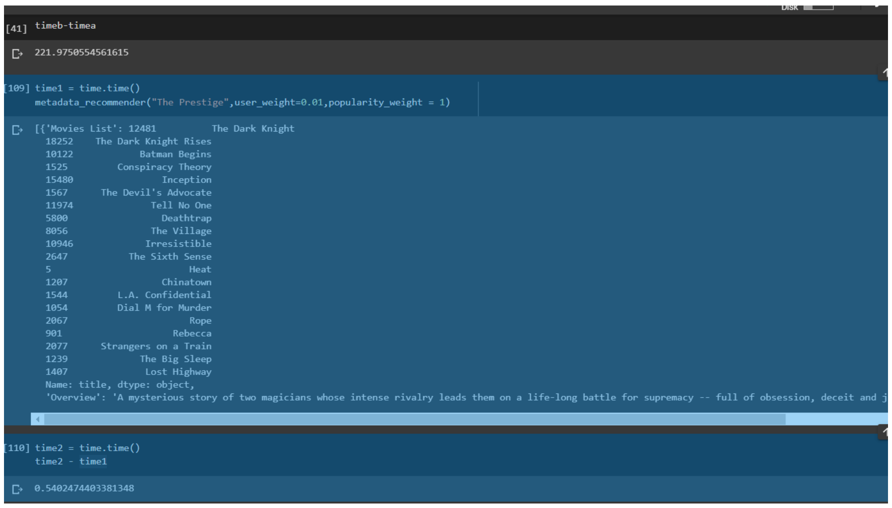
```
[32]: a = count_matrix * count_matrix.T

[*]: a=a.toarray()
```

Settings

Language: Python
Environment: Preferences
Accelerator: None
Internet: On

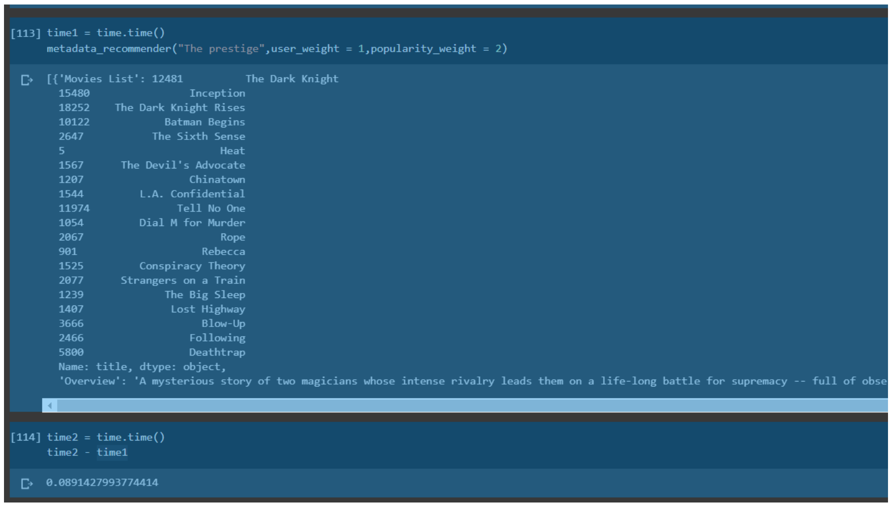
Figure 1: Memory Error

- We checked our results for input "The Prestige" by tuning user_weight and popularity_weight which we have given control to the user in our work. The suggestions obtained by our work are promising. As we left the default popularity weight and tuned user weight, we observed the high rated movies to be moving up in our list as it was supposed to. This observation is depicted by figure 2 and 3. Also, changing popularity weight by single digit range doesn't reflect any significant changes although when we set it to 100 as shown in Figure 4, few changes in the suggestions occur.



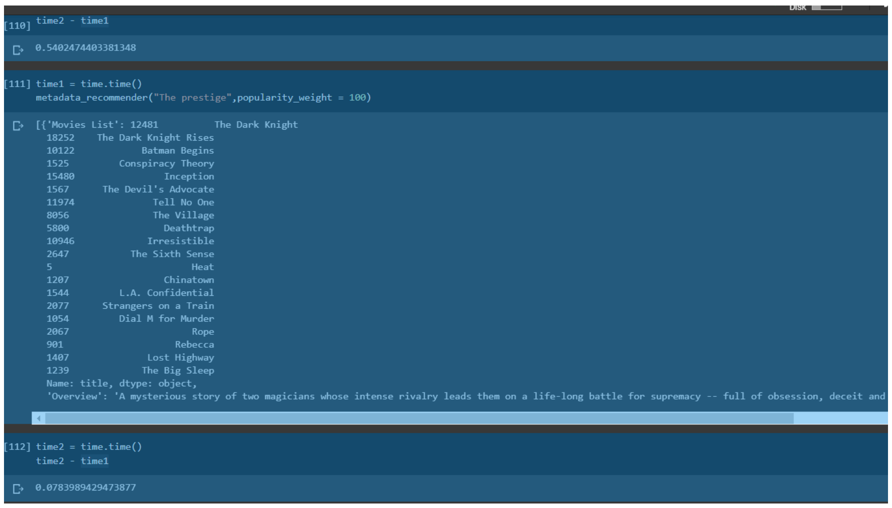
```
[41] timeb-timea

221.9750554561615

[109] time1 = time.time()
      metadata_recommender("The Prestige",user_weight=0.01,popularity_weight = 1)

[{'Movies List': 12481          The Dark Knight
  18252    The Dark Knight Rises
  10122           Batman Begins
  1525        Conspiracy Theory
  15480               Inception
  1567      The Devil's Advocate
  11974             Tell No One
  5800                Deathtrap
  8056              The Village
  10946            Irresistible
  2647           The Sixth Sense
  5                        Heat
  1207                Chinatown
  1544        L.A. Confidential
  1054        Dial M for Murder
  2067                     Rope
  901                   Rebecca
  2077      Strangers on a Train
  1239            The Big Sleep
  1407             Lost Highway
  Name: title, dtype: object,
  'Overview': 'A mysterious story of two magicians whose intense rivalry leads them on a life-long battle for supremacy -- full of obsession, deceit and j

[110] time2 = time.time()
      time2 - time1

0.5402474403381348
```

Figure 2: The Prestige: default



```
[113] time1 = time.time()
      metadata_recommender("The prestige",user_weight = 1,popularity_weight = 2)

[{'Movies List': 12481          The Dark Knight
  15480               Inception
  18252    The Dark Knight Rises
  10122           Batman Begins
  2647           The Sixth Sense
  5                        Heat
  1567      The Devil's Advocate
  1207                Chinatown
  1544        L.A. Confidential
  11974             Tell No One
  1054        Dial M for Murder
  2067                     Rope
  901                   Rebecca
  1525        Conspiracy Theory
  2077      Strangers on a Train
  1239            The Big Sleep
  1407             Lost Highway
  3666                  Blow-Up
  2466                Following
  5800                Deathtrap
  Name: title, dtype: object,
  'Overview': 'A mysterious story of two magicians whose intense rivalry leads them on a life-long battle for supremacy -- full of obse

[114] time2 = time.time()
      time2 - time1

0.0891427993774414
```

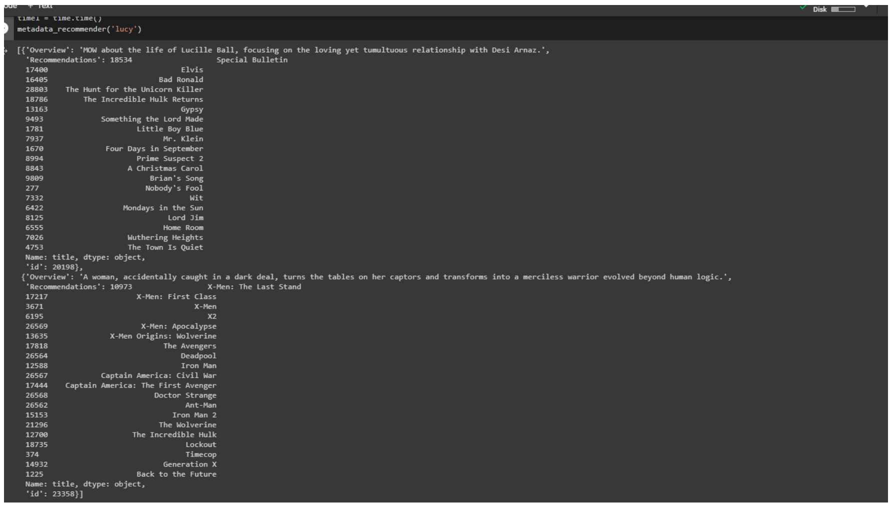
Figure 3: The Prestige: user_weight = 1, popularity_weight = 2



```
[110] time2 - time1

0.5402474403381348

[111] time1 = time.time()
      metadata_recommender("The prestige",popularity_weight = 100)

[{'Movies List': 12481         The Dark Knight
  18252    The Dark Knight Rises
  10122            Batman Begins
  1525         Conspiracy Theory
  15480                Inception
  1567       The Devil's Advocate
  11974              Tell No One
  8056               The Village
  5800                Deathtrap
  10946             Irresistible
  2647           The Sixth Sense
  5                        Heat
  1207                Chinatown
  1544          L.A. Confidential
  2077       Strangers on a Train
  1054          Dial M for Murder
  2067                     Rope
  901                   Rebecca
  1407             Lost Highway
  1239            The Big Sleep
  Name: title, dtype: object,
  'Overview': 'A mysterious story of two magicians whose intense rivalry leads them on a life-long battle for supremacy -- full of obsession, deceit and

[112] time2 = time.time()
      time2 - time1

0.0783989429473877
```

Figure 4: The Prestige: user_weight = 0.01, popularity_weight = 100

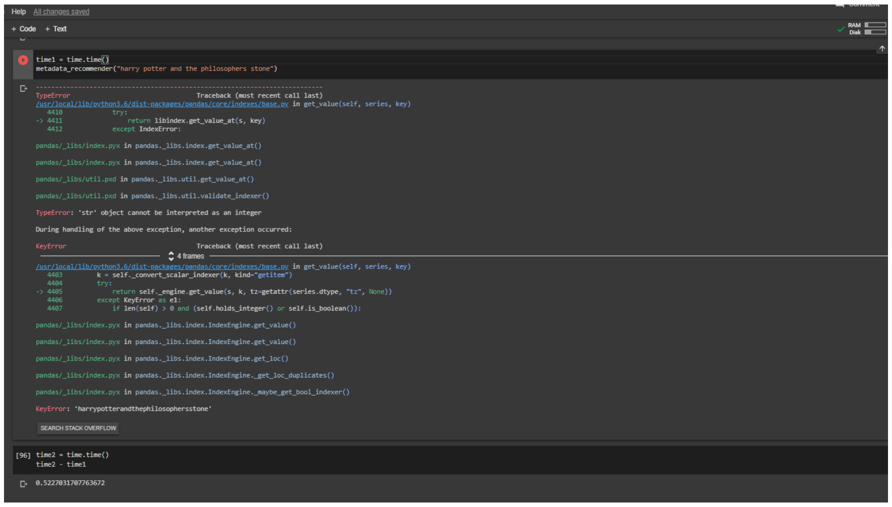
- Also, for an input which matches more than one indices in our dataset, we were able to obtain promising result for all of the indices and we print their different ids and overviews along with the corresponding recommendations for 20 movies as we had described above. A demonstration of this is shown in Figure 5 for user input ”Lucy”:



```
time1 = time.time()
metadata_recommender('lucy')

[{'Overview': 'MOW about the life of Lucille Ball, focusing on the loving yet tumultuous relationship with Desi Arnaz.',
  'Recommendations': 18534                   Special Bulletin
  17400                       Elvis
  16405                  Bad Ronald
  28803    The Hunt for the Unicorn Killer
  18786        The Incredible Hulk Returns
  13163                       Gypsy
  9493           Something the Lord Made
  1781                Little Boy Blue
  7937                     Mr. Klein
  1670           Four Days in September
  8994                Prime Suspect 2
  8843              A Christmas Carol
  9809                   Brian's Song
  277                   Nobody's Fool
  7332                          Wit
  6422             Mondays in the Sun
  8125                     Lord Jim
  6555                    Home Room
  7026              Wuthering Heights
  4753              The Town Is Quiet
  Name: title, dtype: object,
  'id': 20198},
 {'Overview': 'A woman, accidentally caught in a dark deal, turns the tables on her captors and transforms into a merciless warrior evolved beyond human logic.',
  'Recommendations': 10973                X-Men: The Last Stand
  17217                X-Men: First Class
  3671                             X-Men
  6195                                X2
  26569                X-Men: Apocalypse
  13635         X-Men Origins: Wolverine
  17818                     The Avengers
  26564                         Deadpool
  12588                         Iron Man
  26567          Captain America: Civil War
  17444    Captain America: The First Avenger
  26568                   Doctor Strange
  26562                          Ant-Man
  15153                       Iron Man 2
  21296                    The Wolverine
  12700              The Incredible Hulk
  18735                          Lockout
  374                            Timecop
  14932                     Generation X
  1225                 Back to the Future
  Name: title, dtype: object,
  'id': 23358}]
```

Figure 5: Top-Lucy(Desi Arnaz); Bottom-Lucy(Scarlett Johansson)

- Sometimes if user makes spelling mistakes in giving the input movie's title then it may not find any match for that input in the dataset even though intended input was there. Also, it may mistake it with some other title, if the mistake in case spelling gets that close. We have shown an example through figure 6 and 7 where if we try to search for "Harry Potter and the Philosophers Stone" instead of "Harry Potter and the Philosopher's Stone", we couldn't get recommendations. Instead error is printed on the screen. Although, there is scope for improvement for such cases. We can possibly allow some spelling errors instead of getting exact names as long as it doesn't match more closely to any other title.



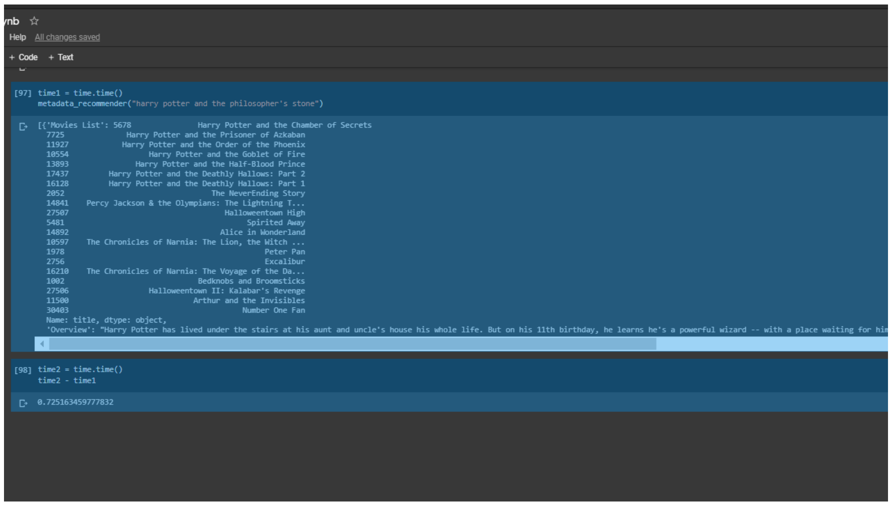
```
time1 = time.time()
metadata_recommender("harry potter and the philosophers stone")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/pandas/core/indexes/base.py in get_value(self, series, key)
   4410             try:
-> 4411                 return libindex.get_value_at(s, key)
   4412             except IndexError:

pandas/_libs/index.pyx in pandas._libs.index.get_value_at()

pandas/_libs/index.pyx in pandas._libs.index.get_value_at()

pandas/_libs/util.pxd in pandas._libs.util.get_value_at()

pandas/_libs/util.pxd in pandas._libs.util.validate_indexer()

TypeError: 'str' object cannot be interpreted as an integer

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/pandas/core/indexes/base.py in get_value(self, series, key)
   4403         k = self._convert_scalar_indexer(k, kind="getitem")
   4404         try:
-> 4405             return self._engine.get_value(s, k, tz=getattr(series.dtype, "tz", None))
   4406         except KeyError as e1:
   4407             if len(self) > 0 and (self.holds_integer() or self.is_boolean()):

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_value()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_value()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine._get_loc_duplicates()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine._maybe_get_bool_indexer()

KeyError: 'harrypotterandthephilosophersstone'
```

```
[96] time2 = time.time()
     time2 - time1
```

```
0.5227031707763672
```

Figure 6: Title not found due to spelling error



```
[97] time1 = time.time()
     metadata_recommender("harry potter and the philosopher's stone")

     [{'Movies List': 5678              Harry Potter and the Chamber of Secrets
       7725             Harry Potter and the Prisoner of Azkaban
       11927           Harry Potter and the Order of the Phoenix
       10554                 Harry Potter and the Goblet of Fire
       13893              Harry Potter and the Half-Blood Prince
       17437        Harry Potter and the Deathly Hallows: Part 2
       16128        Harry Potter and the Deathly Hallows: Part 1
       2052                              The NeverEnding Story
       14841    Percy Jackson & the Olympians: The Lightning T...
       27507                                Halloweentown High
       5481                                      Spirited Away
       14892                                 Alice in Wonderland
       10597    The Chronicles of Narnia: The Lion, the Witch ...
       1978                                          Peter Pan
       2756                                          Excalibur
       16210    The Chronicles of Narnia: The Voyage of the Da...
       1002                            Bedknobs and Broomsticks
       27506                 Halloweentown II: Kalabar's Revenge
       11500                           Arthur and the Invisibles
       30403                                     Number One Fan
       Name: title, dtype: object,
       'Overview': "Harry Potter has lived under the stairs at his aunt and uncle's house his whole life. But on his 11th birthday, he learns he's a powerful wizard -- with a place waiting for him

[98] time2 = time.time()
     time2 - time1

     0.725163459777832
```

Figure 7: Correct spelling shows recommendations

- Since, we have a limited dataset consisting of only 45000 data points which also consists of movies only before 2017, we would be unable to find some existing movies even if user gives correct input. As in the case of movies "Captain Marvel" and "District C-11" we couldn't find the title in our dataset since first one has come after 2017 and the other is not available in our limited dataset. Obviously, with a bigger dataset we would be able to get desired results for such user inputs too.