SUMMER TRAINING REPORT

On

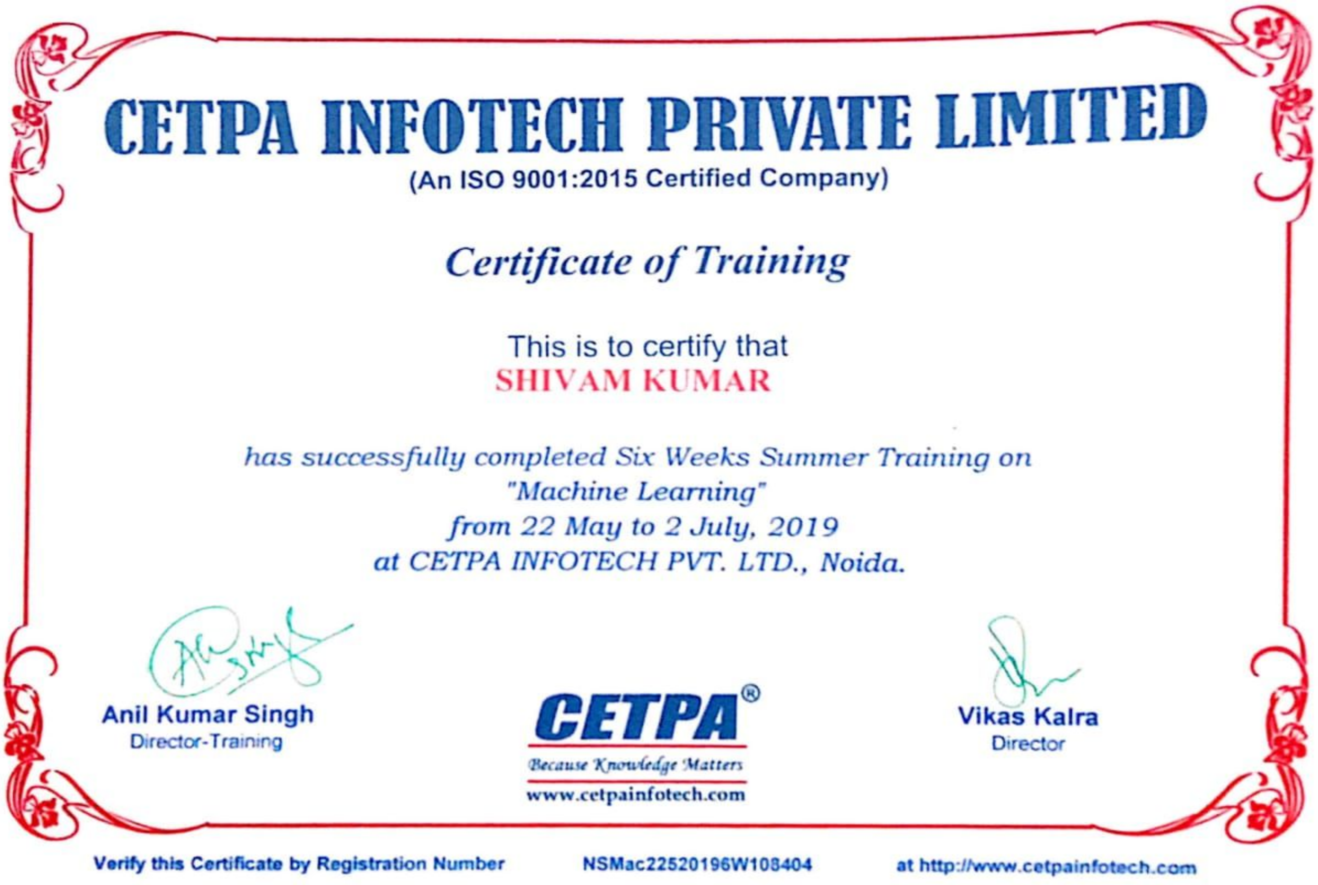
Machine Learning With Python

Submitted by

Shivam Kumar
Roll No: 171500320

Department of Computer Engineering & Applications
Institute of Engineering & Technology

GLA University
Mathura- 281406, INDIA
2019



CETPA INFOTECH PRIVATE LIMITED

(An ISO 9001:2015 Certified Company)

Certificate of Training

This is to certify that

SHIVAM KUMAR

has successfully completed Six Weeks Summer Training on
"Machine Learning"
from 22 May to 2 July, 2019
at CETPA INFOTECH PVT. LTD., Noida.

[illegible]

Anil Kumar Singh
Director-Training

CETPA®
Because Knowledge Matters
www.cetpainfotech.com

[illegible]

Vikas Kalra
Director

Verify this Certificate by Registration Number NSMac22520196W108404 at http://www.cetpainfotech.com

SYNOPSIS

Student Information:

Name: Shivam Kumar	University Roll. No.: 171500320
Mobile:7302232921	Email: shivam.kumar_cs17@gla.ac.in

Information about Industry/Organization:

Industry/Organization Name with full Address	**Cetpa Infotech Pvt. Ltd.** D-58, Sector-2, Near Red FM. Noida -201301, Uttar Pradesh
Contact Person	Name & Designation: Mr. Kuldeep Dixit , BDM Mobile/email: +918800558871

Project Information:

Title Of Project/Training/T	Movie Recommender System
Role & Responsibility	Designing and deploying scalable, highly available, fault-tolerant systems on the ML platform
Technical Details	Minimum Hardware Requirements: 1. Internet Connection 2. A Basic PC i.e. : a. 1 GB RAM b. 1.4GHz 64-bit quad-core c. A Display d. Keyboard and Mouse Software Requirements: 1. Any OS 2. Web Browser 3. Jupyter Notebook
Training Implementat Details	Partial Implemented
Training Period	Start Date: 22 May 2019 End Date: 02 July 2019 Duration Of Training (In Weeks): 6

Summary of the Training Work:

This project is based on machine learning with python which works on predefined datasets and simply recommends users new movies based on ratings and its personal experience. In this project we implement python libraries and uses bootstrapped aggregation, and correlations to reach our goal.

CONTENTS

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my guide **Ms. Sakshi, Machine Learning Trainer, Cetpa Infotech Pvt. Ltd.** for providing and guiding this training. I deeply respect trainer for his vast knowledge, numerous suggestions, and strong passion to complete this training. Valuable discussions with her not only made my work smooth but also encouraged me to think more professionally in the field of Machine Learning.

I also thank all my teaching and non-teaching staff for their support and well wishes.

Finally, I would like to express my deepest gratitude to my parents and friends for their encouragement and support.

Department of computer Engineering and Applications
GLA University, Mathura
17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,
Mathura – 281406

DECLARATION

I hereby declare that the work which is being presented in the Summer Training "**Machine Learning with Python",** in partial fulfillment of the requirements for Summer Training viva voice, is an authentic record of my own work carried under the supervision of "Cetpa Infotech Pvt Ltd."

Signature of Candidate:

Name of Candidate: Shivam Kumar

Roll. No. : 171500320

Course: B.Tech (CSE)

Year: III

Semester: V

Department of CEA,
GLAU Mathura

ABOUT THE ORGANIZATION

CETPA INFOTECH PVT LTD is the leader in the “specialized training” brands of India certified by ISO 9001:2015 for its best quality. CETPA INFOTECH is the largest training service provider in various engineering domains for all engineering students as well as for the working professionals. It has an extensive experience of nurturing over 200000+ students in the past few years.

CETPA has been awarded as the “Best IT and Embedded Training Company” for 5 consecutive years for delivering high quality training and workshops at more than 500 colleges across India. CETPA is a trustworthy brand in Education and Training industry with its presence across several cities such as Noida, Roorkee, Lucknow, Dehradun and . The company was started 12 years back and it is continuously expanding its overseas branches in Germany and Ukraine.

CETPA has specialization in 3 important domains namely: TRAINING, DEVELOPMENT and CONSULTANCY. The company provides specialized training in 50+ leading technologies like .NET, Java, PHP, Ethical Hacking, ANDROID, CCNA, AUTOCAD, VHDL, MATLAB, EMBEDDED SYSTEM, HVAC and many more. CETPA has a very committed team consisting of technical trainers who are continuously guiding, mentoring, admonish and coaching the students by providing them with exclusive personalized attention, which helps them to develop solid industry oriented knowledge.

Getting a post is as difficult as defeat the crew because being in the associate world need a lot from the candidate because of which the candidates are putting their perfect effort, which results in process of hardship level. Students can see each business is connected but resolve this problem is either consuming years to reach a crave place or come to CETPA. CETPA gives the entire essential computer training which helps the beginner and the experienced employees so that they can have better identification in this competitive globe.

Like other academic and training company, at CETPA learner will be offer assortments programs but the trainers differs and make CETPA stand out from leftovers. CETPA has variety of knowledgeable and experienced trainers whose accession is different which learners can see everywhere. CETPA try its level best to boost their trainee’s capability so that they stand out from others and whatever they grant to the corporate world inevitably becomes fruitful. Not only the fresher but also the professionals who are not adequate to deal with the upcoming technology and software are also helped here. CETPA try its aligned best to convey its services to every edge of the globe with the help of tailored education.

ABSTRACT

Recommender systems have become ubiquitous in our lives. Yet, currently, they are far from optimal. In this project, we attempt to understand the different kinds of recommendation systems and compare their performance on the MovieLens dataset. We attempt to build a scalable model to perform this analysis. We start by preparing and comparing the various models on a smaller dataset of 100,000 ratings. Then, we try to scale the algorithm so that it is able to handle 20 million ratings by using Apache Spark. We find that for the smaller dataset, using user-based collaborative filtering results in the lowest Mean Squared Error on our dataset.

CHAPTER 1:

INTRODUCTION

A recommendation system is a type of information filtering system which attempts to predict the preferences of a user, and make suggests based on these preferences. There are a wide variety of applications for recommendation systems. These have become increasingly popular over the last few years and are now utilized in most online platforms that we use. The content of such platforms varies from movies, music, books and videos, to friends and stories on social media platforms, to products on e-commerce websites, to people on professional and dating websites, to search results returned on Google. Often, these systems are able to collect information about a users choices, and can use this information to improve their suggestions in the future. For example, Facebook can monitor your interaction with various stories on your feed in order to learn what types of stories appeal to you. Sometimes, the recommender systems can make improvements based on the activities of a large number of people. For example, if Amazon observes that a large number of customers who buy the latest Apple Macbook also buy a USB-C-toUSB Adapter, they can recommend the Adapter to a new user who has just added a Macbook to his cart. Due to the advances in recommender systems, users constantly expect good recommendations. They have a low threshold for services that are not able to make appropriate suggestions. If a music streaming app is not able to predict and play music that the user likes, then the user will simply stop using it. This has led to a high emphasis by tech companies on improving their recommendation systems. However, the problem is more complex than it seems. Every user has different preferences and likes. In addition, even the taste of a single user can vary depending on a large number of factors, such as mood, season, or type of activity the user is doing. For example, the type of music one would like to hear while exercising differs greatly from the type of music he'd listen to when cooking dinner. Another issue that recommendation systems have to solve is the exploration vs exploitation problem. They must explore new domains to discover more about the user, while still making the most of what is already known about of the user. Two main approaches are widely used for recommender systems. One is content-based filtering, where we try to profile the users interests using information collected, and recommend items based on that profile. The other is collaborative filtering, where we try to group similar users together and use information about the group to make recommendations to the user. Both approaches are discussed in greater detail in Section 3.

CHAPTER 2:

WORK FLOW DIAGRAM

Machine learning has given the computer systems the abilities to automatically learn without being explicitly programmed. But how does a machine learning system work? So, it can be described using the life cycle of machine learning. Machine learning life cycle is a cyclic process to build an efficient machine learning project. The main purpose of the life cycle is to find a solution to the problem or project. Machine learning life cycle involves seven major steps, which are given below:

1.Gathering Data

2.Data preparation

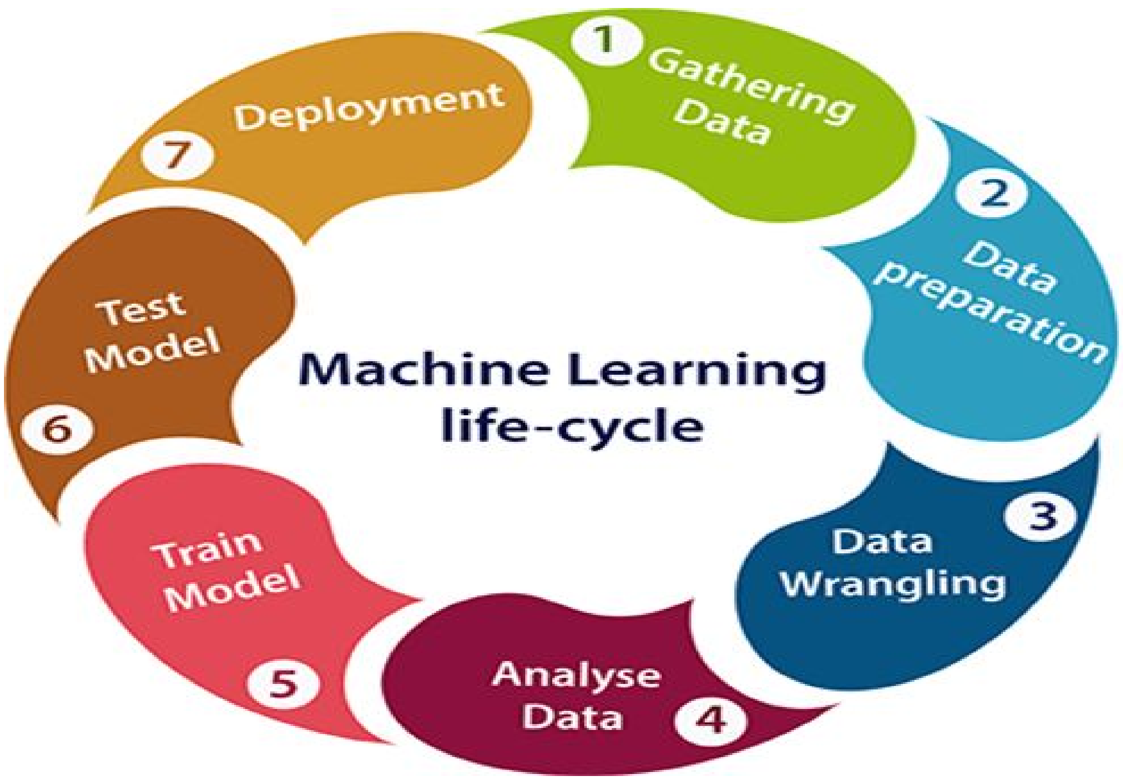
3.Data Wrangling

4.Analyse Data

5.Train the model

6.Test the model

7.Deployment



CHAPTER 3:

PROBLEM SETTING

The problem that we address in this project can be formulated as follows. Let R be the ratings matrix with dimensions (num_users × num_items). The entry rij in the ratings matrix R contains a nonzero rating value given by the user i for the item j. The matrix R is sparse in nature i.e. most of the entries rij are missing. We generally have a few ratings or some purchase history for each user and similarly each item will have been rated by a few users, but most of the entries in the ratings matrix are generally missing. The task at hand is to predict the missing entries in the ratings matrix R. The data that constitutes the ratings matrix R can be collected either explicitly by asking the users to rate the items or by implicitly deriving the ratings for items based on measures such as whether the user purchased the item, or whether the user clicked a certain page and like wise. We work with the MovieLens dataset, which contains explicit ratings given by users to movies. Evaluation We use the mean squared error metric to evaluate the predictions made by our system. The mean squared error is computed as MSE = 1 N Õ N i=1 (pij − rij) 2 where N is the number of ratings in the test partition, pij is the predicted rating for user i and movie j and rij is the actual rating.

CHAPTER 4:

ALGORITHMS

For our project, we focused on two main algorithms for recommendations: Collaborative filtering & Content-based filtering. 3.1 Collaborative Filtering Collaborative Filtering techniques make recommendations for a user based on ratings and preferences data of many users. The main underlying idea is that if two users have both liked certain common items, then the items that one user has liked that the other user

4.1 Collaborative Filtering

Collaborative Filtering techniques make recommendations for a user based on ratings and preferences data of many users. The main underlying idea is that if two users have both liked certain common items, then the items that one user has liked that the other user has not yet tried can be recommended to him. We see collaborative filtering techniques in action on various Internet platforms such as Amazon.com, Netflix, Facebook. We are recommended items based on the ratings and purchase data that these platforms collect from their user base. We explore two algorithms for Collaborative filtering, the Nearest Neighbors Algorithm and the Latent Factors Algorithm.

3.1.1 Nearest Neighbors Collaborative Filtering: This approach relies on the idea that users who have similar rating behaviors so far, share the same tastes and will likely exhibit similar rating behaviors going forward. The algorithm first computes the similarity between users by using the row vector in the ratings matrix corresponding to a user as a representation for that user. The similarity is computed by using either cosine similarity or Pearson Correlation. In order to predict the rating for a particular user for a given movie j, we find the top k similar users to this particular user and then take a weighted average of the ratings of the k similar users with the weights being the similarity values.

3.1.2 Latent Factor Methods: The latent factor algorithm looks to decompose the ratings matrix R into two tall and thin matrices Q and P, with matrix Q having dimensions num_users × k and P having the dimensions numitems × k where k is the number of latent factors. The decomposition of R into Q and P is such that R = Q.P T . Any rating rij in the ratings matrix can be computed by taking the dot product of row qi of matrix Q and pj of matrix P. The matrices Q and P are initialized randomly or by performing SVD on the ratings matrix. Then, the algorithm solves the problem of minimizing the error between the actual rating value rij and the value given by taking the dot product of rows qi and pj . The algorithm performs stochastic gradient descent to find the matrices Q and P with minimum error starting from the initial matrices.

4.2 Content Based Recommendations

Content Based Recommendation algorithm takes into account the likes and dislikes of the user and generates a User Profile. For generating a user profile, we take into account the item profiles(vector describing an item) and their corresponding user rating. The user profile is the weighted sum of the item profiles with weights being the ratings user rated. Once the user profile is generated, we calculate the similarity of the user profile with all the items in the dataset, which is calculated using cosine similarity between the user profile and item profile. Advantages of Content Based approach is that data of other users is not required and the recommender engine can recommend new items which are not rated currently, but the recommender algorithm doesn't recommend the items outside the category of items the user has rated.

CHAPTER 5:

DATASET

We used the MovieLens movie ratings dataset for our experiments1 . The experiments are conducted on two versions of the dataset. The first version consists of 100004 ratings by 671 users across 9125 movies. The ratings allowed at intervals of 0.5 on a 5-point scale, starting from 0.5 and going to 5. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided. The dataset has additional information about the movies in the form of genre and tags, however we use only the ratings given by the users to the movies and ignore the other information for the collaborative filtering techniques. For the content filtering portion, information was scrapped from the IMDB website for the corresponding movie. The links to the IMDB page for each movie is provided in a separate file. The second and bigger version of the dataset consists of 20000263 (20 million) ratings by 138493 users across 27278 movies. Apart from this, the structure of the two datasets also is identical. The movie ids for a particular movie is the same in both datasets, but the user id for the same user is different for the two datasets. We split each dataset into three partitions: training, validation and test by sampling randomly in the ratios 80%, 10%, 10% respectively. The validation partition is used to tune the hyper-parameters for the nearest neighbor and latent factor algorithms. The bigger version of the dataset presents significant scalability challenges.

CHAPTER 6:

IMPLEMENTATION

6.1 Baseline methods

We try out the following simple baseline methods to give us an idea of the performance to expect from the

6.1.1 Global Average. The global average technique serves as a simple baseline technique. The average rating for all users across all movies is computed. This global average serves as a prediction for all the missing entries in the ratings matrix.

6.1.2 User average. All users exhibit varying rating behaviors. Some users are lenient in their ratings, whereas some are very stringent giving lower ratings to almost all movies. This user bias needs to be incorporated into the rating predictions. We compute the average rating for each user. The average rating of the user is then used as the prediction for each missing rating entry for that particular user. This method can be expected to perform slightly better than the global average since it takes into account the rating behavior of the users into account.

6.1.3 Movie average. Some movies are rated highly by almost all users whereas some movies receive poor ratings from everyone. Another simple baseline which can be expected to perform slightly better than the global average is the movie average method. In this technique, each missing rating entry for a movie j is assigned the average rating for the movie j.

6.1.4 Adjusted Average. This simple method tries to incorporate some information about the user i and the movie j when making a prediction for the entry rij . We predict a missing entry for user i and movie j, by assigning it the global average value adjusted for the user bias and movie bias. The adjusted average rating is given by the formula below. rij = дlobal_авд + (u_авд(i) − дij) + (m_авд(j) − дij) The user bias is given by the difference between the average user rating and the global average rating. The movie bias is given by the difference between the average movie rating and the global average rating. Consider the following example which demonstrates the working of the adjusted average method. Let the global average rating be 3.7. The user A has an average rating of 4.1. Thus the bias of the user is (4.1 - 3.7) I.e the user rates 0.4 stars more than the global average. The movie Fast and Furious has an average rating of 3.1 stars. Thus the bias for the movie is -0.6. the adjusted average method will predict that user A will give the movie Fast and Furious a rating of 3.7 + 0.4 - 0.6 = 3.5 .

6.2 Collaborative Filtering

We implement the nearest neighbors and latent factor methods for Collaborative filtering. The smaller version of the dataset can be processed by using dense matrices and non-vectorized operations. However, if we try to use the simple implementations on the larger 20 million dataset, the code breaks.

If we store the ratings matrix for the 20 million dataset in a dense format, it would take up around 140000×27000×8 = 28GB of memory, assuming 8 bytes per matrix entry. Thus we need to use sparse matrix representations to be able to handle the bigger dataset effectively. Also, the matrix operations have to be as vectorized as possible to make efficient use of threads. We observed that even after our best attempts to optimize the code as much as possible, the algorithms still needed a lot of time to be able to process such huge amounts of data. We thus decided to make use of Apache Spark to parallelize the operations and improve the runtime performance of the algorithms by running them on Spark clusters. 5.3 Spark implementation We implemented the algorithms such that they could be run on an Apache Spark cluster. We then run the algorithms on a cluster of 20 AWS EC2 instances. we were able to achieve significant improvements in the running time by using Spark. The nearest neighbors algorithm took around 30 hrs to run on a single machine, whereas the Spark implementation on the 20 machine cluster was able to run in around 50 minutes.

6.3 Spark implementation

We implemented the algorithms such that they could be run on an Apache Spark cluster. We then run the algorithms on a cluster of 20 AWS EC2 instances. we were able to achieve significant improvements in the running time by using Spark. The nearest neighbors algorithm took around 30 hrs to run on a single machine, whereas the Spark implementation on the 20 machine cluster was able to run in around 50 minutes.

6.4 Content based implementation

6.4.1 Data Scrapping.

For this algorithm, we needed to obtain movie metadata. While some basic movie metadata, such as release year, tags and genre were provided in the set, we decided to collect further information about the movie, with hopes of incorporating them into our system. We decided to obtain the required information from the IMDB website. The pages content were accessed using the Beautiful Soup 2 python library. Beautiful Soup is a Python library designed for quick turnaround projects like screen-scraping. It allows us to easily search and navigate the pages content. Once the structure of the website was identified, we wrote a script to automatically extract the relevant information about a web-page and create a dump. In python3, Beautiful soup also handle the encoding of the incoming file, further assisting in the automation.In order to keep the dump size reasonable, we limited the number of fields we extracted. We only extracted the top 10 actors, 2 directors, and 1 writer for each movie. In addition, we extracted the plot summary instead of the synopsis, as extracting the later would have lead to create of a dump of around 1GB.

6.5 Movie Plot keyword search

We also implemented an additional feature that provides a distributed indexer on the movie synopsis as extracted from the IMDB website to allow searching for movies based on keyword searches. It is built on a MapReduce framework to build an inverted index. The framework for the indexer is the same as the one developed earlier in the semester for the assignments. It has been modified to allow searching on the movie information dataset. First, the dataset is generated by scrapping data from the IMDB pages of the respective movies. The IMDB movie id is provided along with the movie lens dataset. After accessing the webpage, the metadata from the movie is collected and then pickled and stored in a dump. This same dump can also be used to implement and extend the content based model. In the current model, the synopsis refers to the first of the provided plot summaries. The summaries tend to be much shorter than the synopsis and can be extracted much more easily. Also, the size of the dump increased by a factor of almost 100 when using synopses instead of plot summaries. One more reason to select plot summaries is that detailed synopsis are not available for a large portion of the movies, which would result in a bias in the search. The rest of the search is similar to the one provided in the assignments. The reformatter has been adjusted to match the format of the dump. The rest of the search can be run in the identical manner as the assignment. More detailed instructions are provided in a readMe file.

6.6 Person name search

In addition to searching the movie plot, we also developed a separate search for names of people associated with the movie. These could be the actors, the directors, or the writers. Both searches are independent, in the fact that using the person search, you will not get results for queries appearing in the movie plots, and visa-versa. In order to incorporate the person search for ambiguous queries that could appear in either plot of in a name, such as Hill, we expanded the number of documents that are returned by the index servers, and evaluated all the movies until either 10 appropriate results are obtained or all the movies returned by the index servers have been analysed.

CHAPTER 7:

PROJECT CANVASS

Recommendation systems play a vital role in today's web. Here are some of the benefits and drawbacks of recommendation systems.

Recommendation systems are based on actual user behaviour i.e. objective reality. This is the biggest advantage - watching people in their natural environment and making design decisions directly on the results. For example, the "Suggested Post" feature of Facebook suggests posts based on our activity and likes.
Recommendation systems are great for discovery. For example, the "Genius Recommendations" feature of iTunes, `"Frequently Bought Together" of Amazon.com makes surprising recommendations which are similar to what we already like. The "Now Touching the Void and Into Thin Air" example discussed in class is a best example.

Recommendation systems are effective tools for personalization. We often take recommendations from friends and family because we trust their opinion. They know what we like better than anyone else. This is the sole reason they are good at recommending things. This is what recommendation systems try to model.

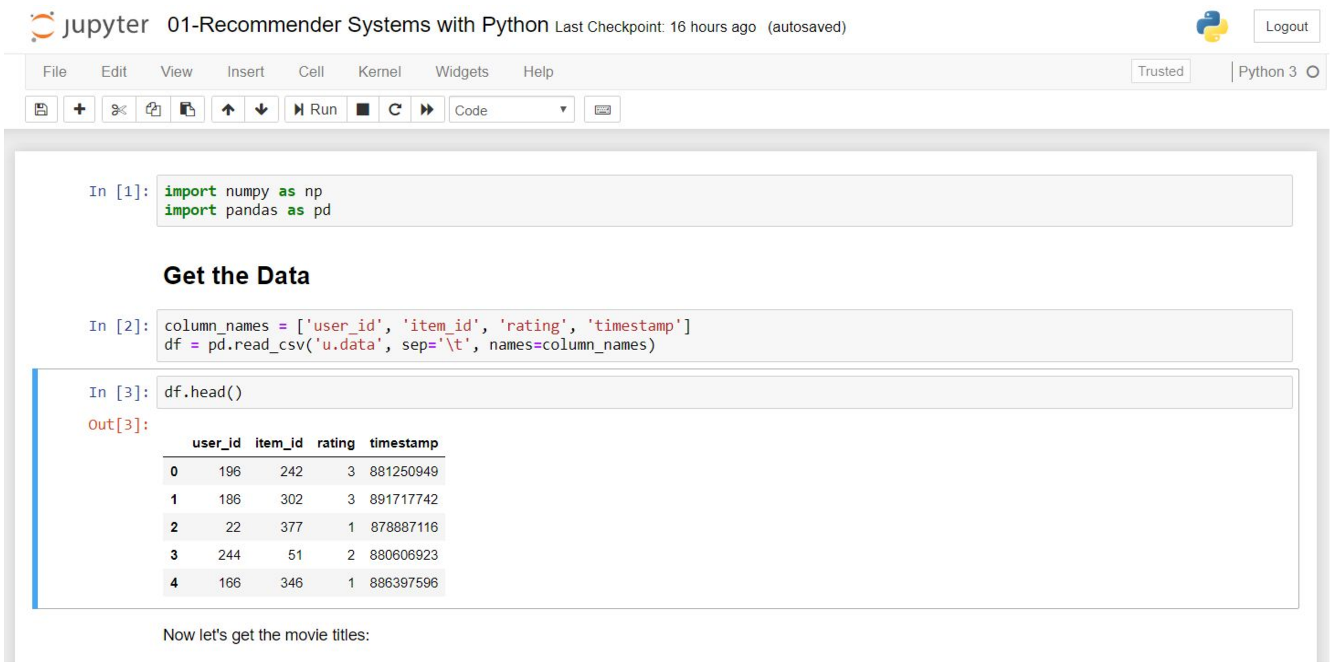
Recommendation systems are always up-to-date. A new product in Amazon gets recommended as long as people rate it highly. The ability for a recommendation system to bubble up activity in real time is a huge advantage because the system is always on.

Most of the organizational maintenance of a site is keeping the navigation system in line with the users' changing needs. With recommendation systems, organizational maintenance is reduced. Based on user activity, the system recommends navigation options to the user. It still takes a designer to decide what type of information should be displayed on what screen. This introduces a drawback too. Keeping the system up and running becomes a major task, so maintenance has to be shifted elsewhere.

CHAPTER 8:

CODE AND RESULTS

1. Importing Libraries and processing the data.


jupyter 01-Recommender Systems with Python Last Checkpoint: 16 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

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

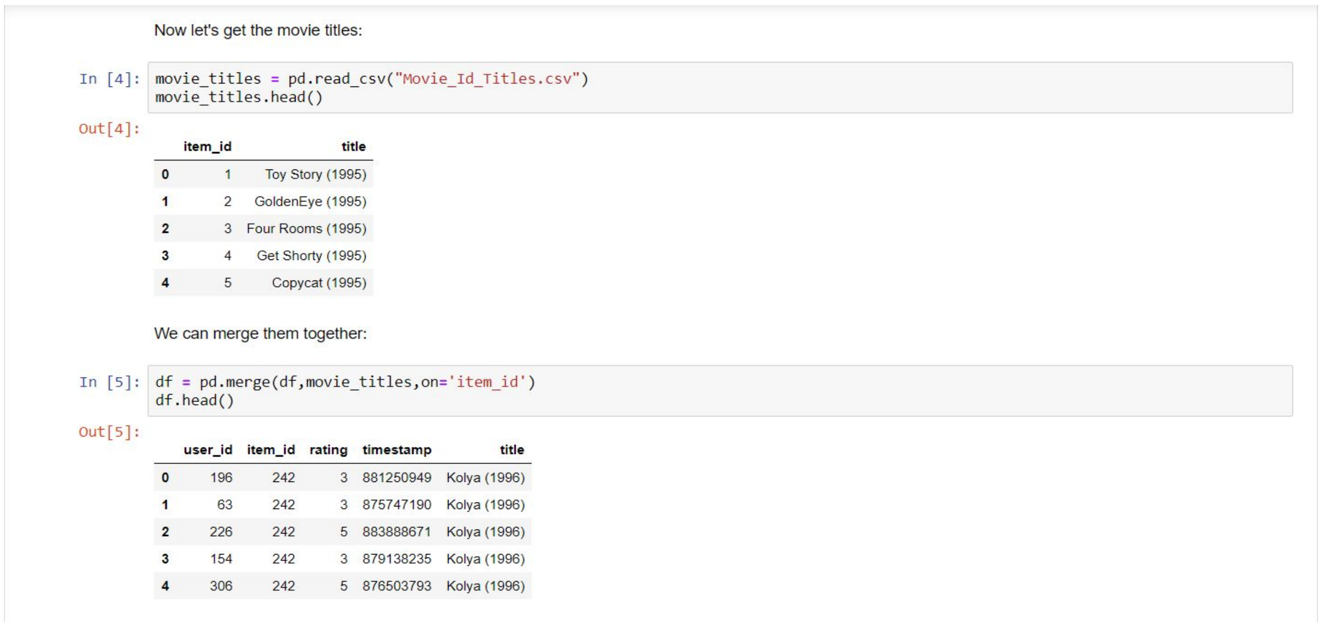
Get the Data

```
In [2]: column_names = ['user_id', 'item_id', 'rating', 'timestamp']
        df = pd.read_csv('u.data', sep='\t', names=column_names)
```

```
In [3]: df.head()
Out[3]:
     user_id  item_id  rating  timestamp
  0      196      242       3  881250949
  1      186      302       3  891717742
  2       22      377       1  878887116
  3      244       51       2  880606923
  4      166      346       1  886397596
```

Now let's get the movie titles:

2. Reading the CSV file using Pandas Library. And merging the titles of the movies with item_id.



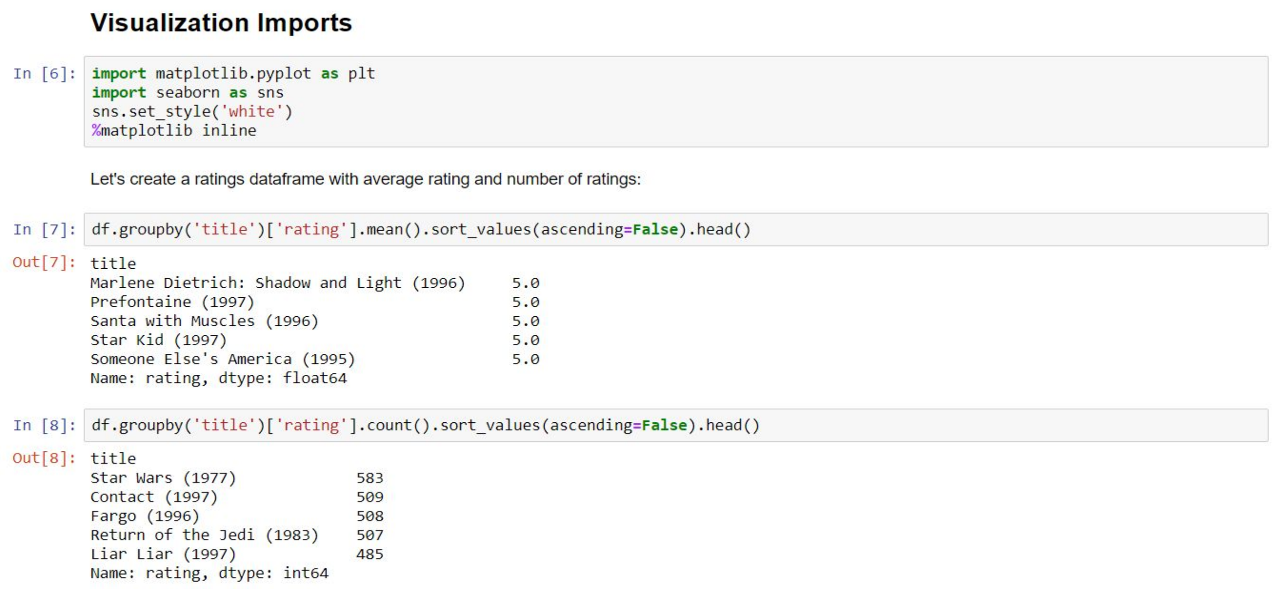
Now let's get the movie titles:

```
In [4]: movie_titles = pd.read_csv("Movie_Id_Titles.csv")
        movie_titles.head()
Out[4]:
     item_id              title
  0        1   Toy Story (1995)
  1        2   GoldenEye (1995)
  2        3  Four Rooms (1995)
  3        4  Get Shorty (1995)
  4        5     Copycat (1995)
```

We can merge them together:

```
In [5]: df = pd.merge(df,movie_titles,on='item_id')
        df.head()
Out[5]:
     user_id  item_id  rating  timestamp         title
  0      196      242       3  881250949  Kolya (1996)
  1       63      242       3  875747190  Kolya (1996)
  2      226      242       5  883888671  Kolya (1996)
  3      154      242       3  879138235  Kolya (1996)
  4      306      242       5  876503793  Kolya (1996)
```

3. Data Visualization using Matplotlib. And exploring the data a bit and get a look at some best rated movies by creating a ratings dataframe with average ratings and number of ratings.



Visualization Imports

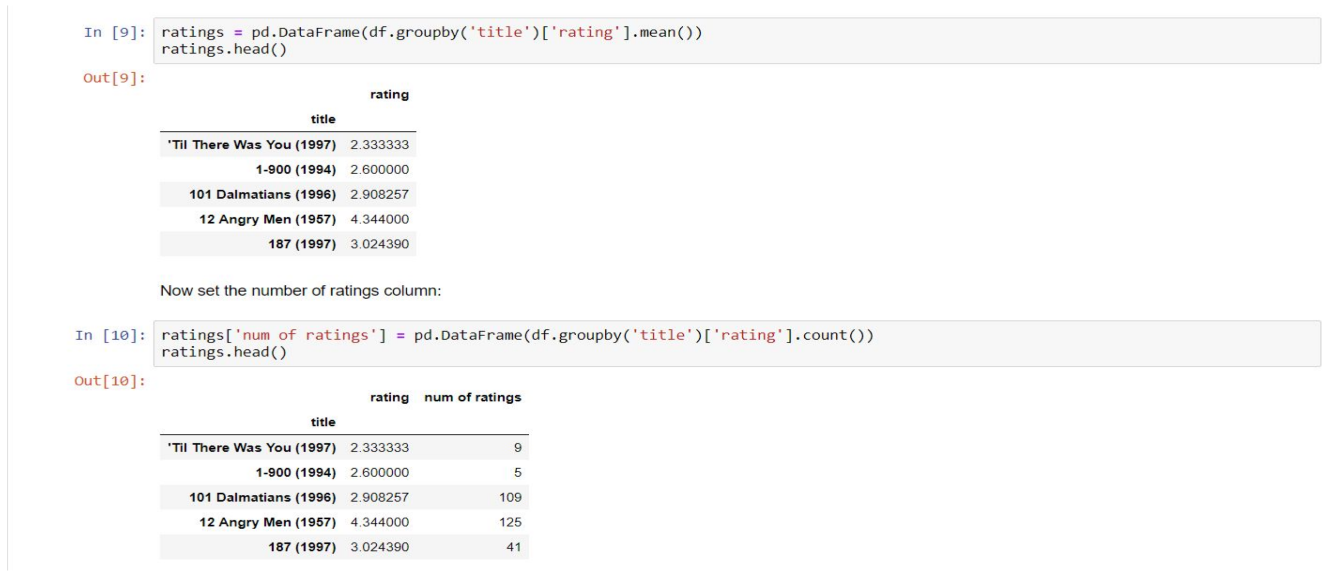
```
In [6]: import matplotlib.pyplot as plt
        import seaborn as sns
        sns.set_style('white')
        %matplotlib inline
```

Let's create a ratings dataframe with average rating and number of ratings:

```
In [7]: df.groupby('title')['rating'].mean().sort_values(ascending=False).head()

Out[7]: title
        Marlene Dietrich: Shadow and Light (1996)     5.0
        Prefontaine (1997)                            5.0
        Santa with Muscles (1996)                     5.0
        Star Kid (1997)                               5.0
        Someone Else's America (1995)                 5.0
        Name: rating, dtype: float64
```

```
In [8]: df.groupby('title')['rating'].count().sort_values(ascending=False).head()

Out[8]: title
        Star Wars (1977)              583
        Contact (1997)                509
        Fargo (1996)                  508
        Return of the Jedi (1983)     507
        Liar Liar (1997)              485
        Name: rating, dtype: int64
```

4. Getting the mean of the Data using mean() and counting using count().



```
In [9]: ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
        ratings.head()
```

Out[9]:

title	rating
'Til There Was You (1997)	2.333333
1-900 (1994)	2.600000
101 Dalmatians (1996)	2.908257
12 Angry Men (1957)	4.344000
187 (1997)	3.024390

Now set the number of ratings column:

```
In [10]: ratings['num of ratings'] = pd.DataFrame(df.groupby('title')['rating'].count())
         ratings.head()
```

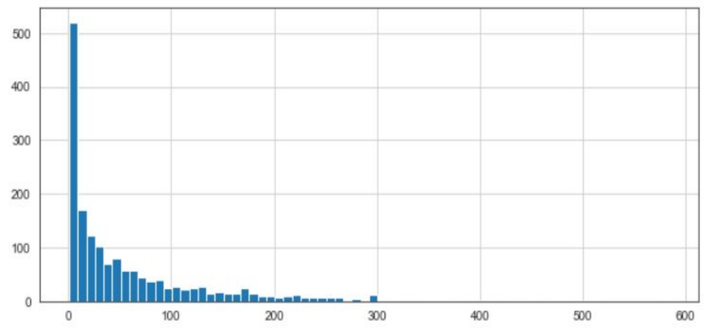
Out[10]:

title	rating	num of ratings
'Til There Was You (1997)	2.333333	9
1-900 (1994)	2.600000	5
101 Dalmatians (1996)	2.908257	109
12 Angry Men (1957)	4.344000	125
187 (1997)	3.024390	41

5. Making a Histogram of the number of ratings.

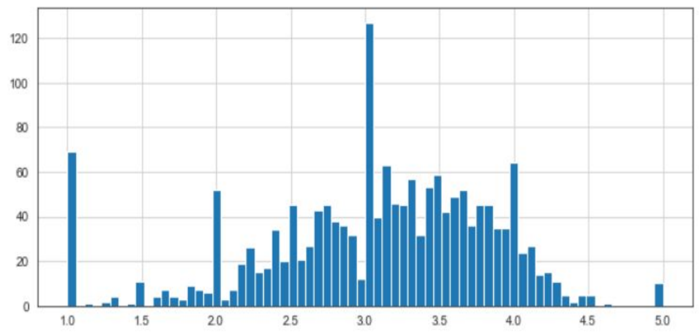
Now a few histograms:

```
In [11]: plt.figure(figsize=(10,4))
         ratings['num of ratings'].hist(bins=70)

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x294ea13b4e0>
```



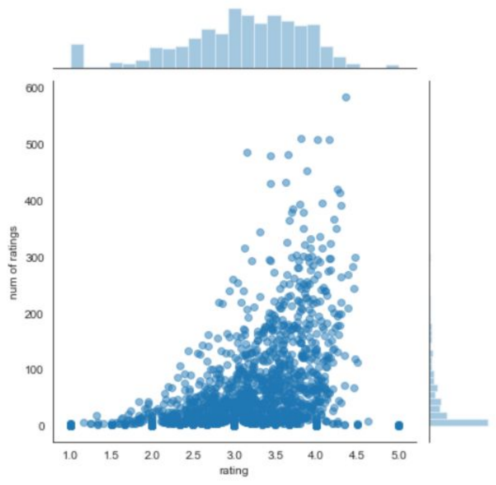
6.Making a histogram of ratings.

```
In [12]: plt.figure(figsize=(10,4))
         ratings['rating'].hist(bins=70)

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x294ea4d6e48>
```



7. Joining both the plots with ratings on x-axis and number of ratings on y-axis.

```
In [13]: sns.jointplot(x='rating',y='num of ratings',data=ratings,alpha=0.5)
Out[13]: <seaborn.axisgrid.JointGrid at 0x294ea5f05f8>
```



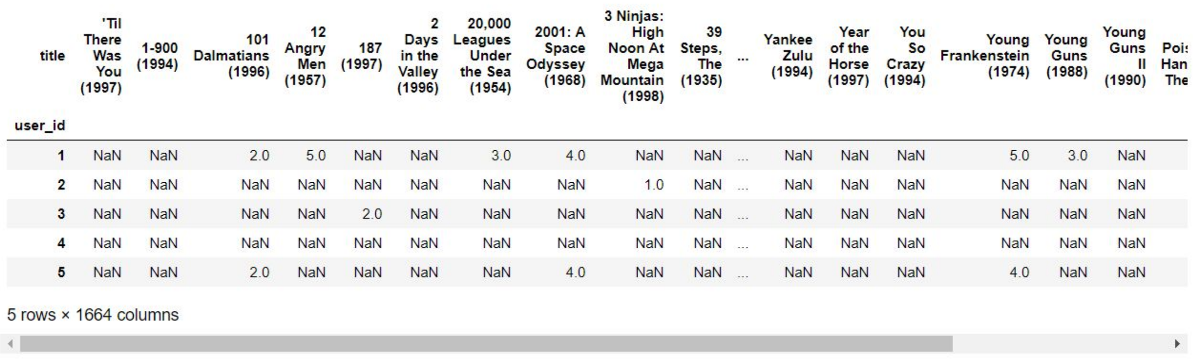
Okay! Now that we have a general idea of what the data looks like, let's move on to creating a simple recommendation system:

8. Revelation of Data.(Recommending similar movies by creating a matrix)

Recommending Similar Movies

Now let's create a matrix that has the user ids on one access and the movie title on another axis. Each cell will then consist of the rating the user gave to that movie. Note there will be a lot of NaN values, because most people have not seen most of the movies.

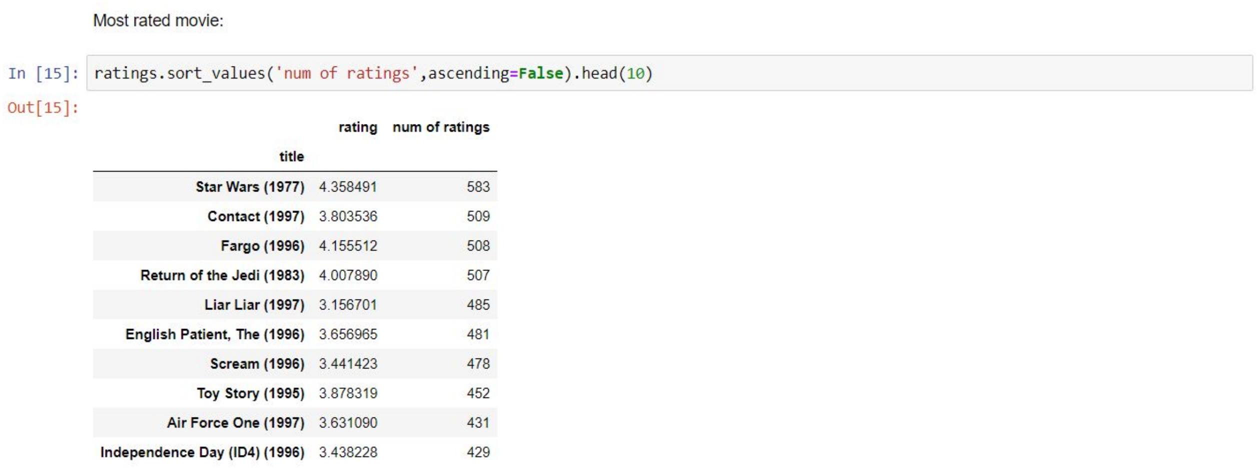
```
In [14]: moviemat = df.pivot_table(index='user_id',columns='title',values='rating')
         moviemat.head()
Out[14]:
```



title	'Til There Was You (1997)	1-900 (1994)	101 Dalmatians (1996)	12 Angry Men (1957)	187 (1997)	2 Days in the Valley (1996)	20,000 Leagues Under the Sea (1954)	2001: A Space Odyssey (1968)	3 Ninjas: High Noon At Mega Mountain (1998)	39 Steps, The (1935)	...	Yankee Zulu (1994)	Year of the Horse (1997)	You So Crazy (1994)	Young Frankenstein (1974)	Young Guns (1988)	Young Guns II (1990)	Poi Han The
user_id																		
1	NaN	NaN	2.0	5.0	NaN	NaN	3.0	4.0	NaN	NaN	...	NaN	NaN	NaN	5.0	3.0	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
5	NaN	NaN	2.0	NaN	NaN	NaN	NaN	4.0	NaN	NaN	...	NaN	NaN	NaN	4.0	NaN	NaN	

5 rows × 1664 columns

9. According to the number of ratings, Most rated movies arranged in descending order(most rated to least rated)



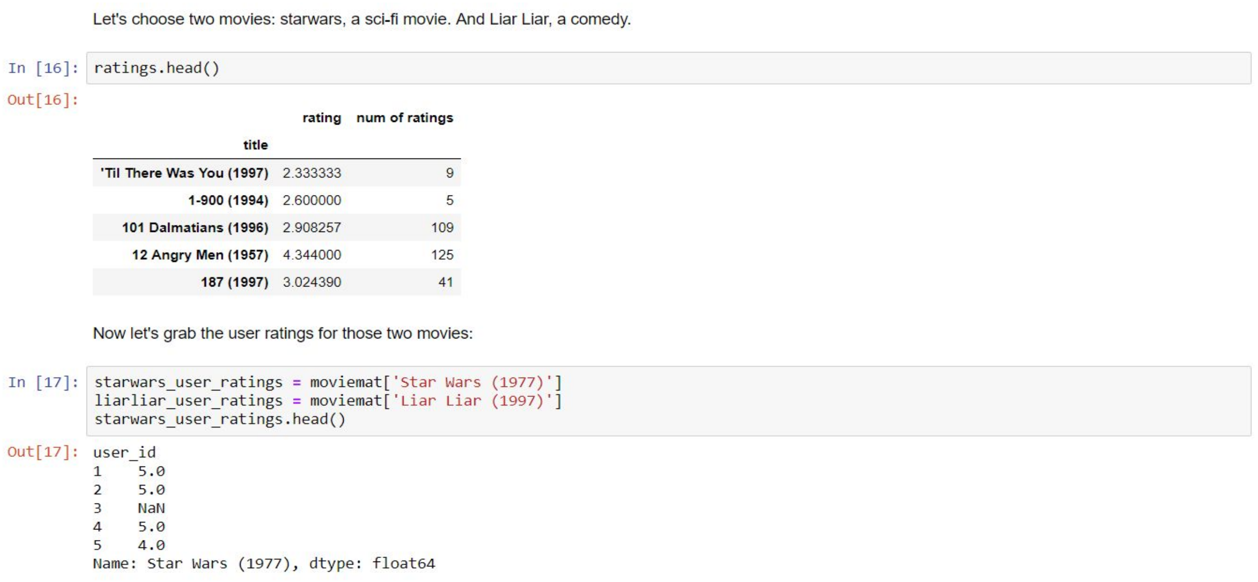
Most rated movie:

In [15]:
```
ratings.sort_values('num of ratings',ascending=False).head(10)
```

Out[15]:

title	rating	num of ratings
Star Wars (1977)	4.358491	583
Contact (1997)	3.803536	509
Fargo (1996)	4.155512	508
Return of the Jedi (1983)	4.007890	507
Liar Liar (1997)	3.156701	485
English Patient, The (1996)	3.656965	481
Scream (1996)	3.441423	478
Toy Story (1995)	3.878319	452
Air Force One (1997)	3.631090	431
Independence Day (ID4) (1996)	3.438228	429

10. Taking two Instances from the Dataset(having similar "ratings") for Analysis.



Let's choose two movies: starwars, a sci-fi movie. And Liar Liar, a comedy.

In [16]:
```
ratings.head()
```

Out[16]:

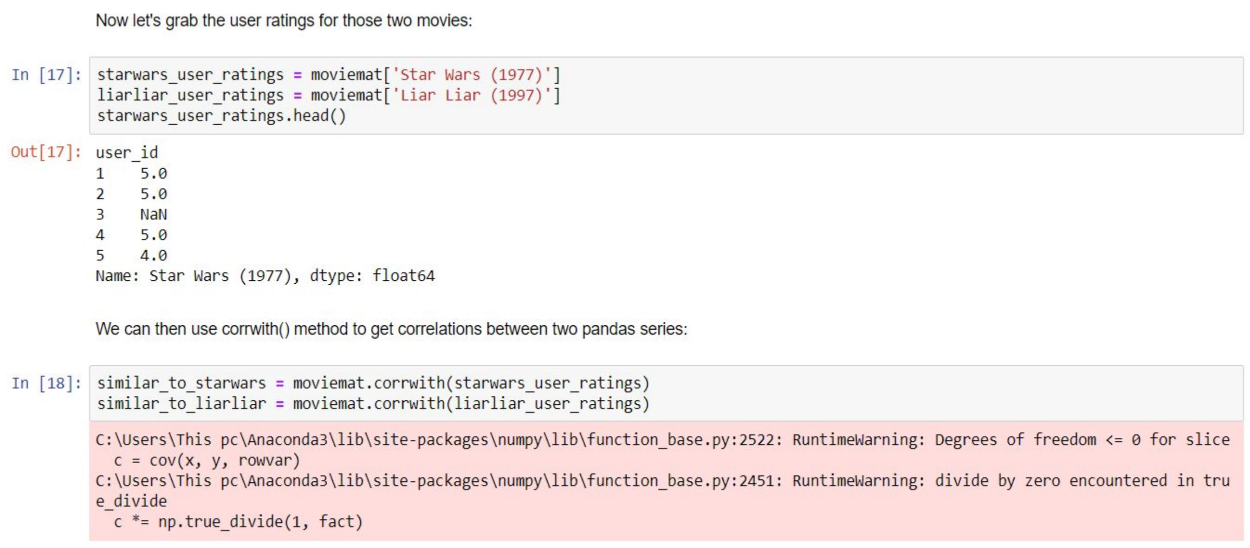
title	rating	num of ratings
'Til There Was You (1997)	2.333333	9
1-900 (1994)	2.600000	5
101 Dalmatians (1996)	2.908257	109
12 Angry Men (1957)	4.344000	125
187 (1997)	3.024390	41

Now let's grab the user ratings for those two movies:

In [17]:
```
starwars_user_ratings = moviemat['Star Wars (1977)']
liarliar_user_ratings = moviemat['Liar Liar (1997)']
starwars_user_ratings.head()
```

Out[17]:
```
user_id
1    5.0
2    5.0
3    NaN
4    5.0
5    4.0
Name: Star Wars (1977), dtype: float64
```

11. Calculating Correlation.(Movies having Similar Ratings).



Now let's grab the user ratings for those two movies:

```
In [17]: starwars_user_ratings = moviemat['Star Wars (1977)']
         liarliar_user_ratings = moviemat['Liar Liar (1997)']
         starwars_user_ratings.head()
```

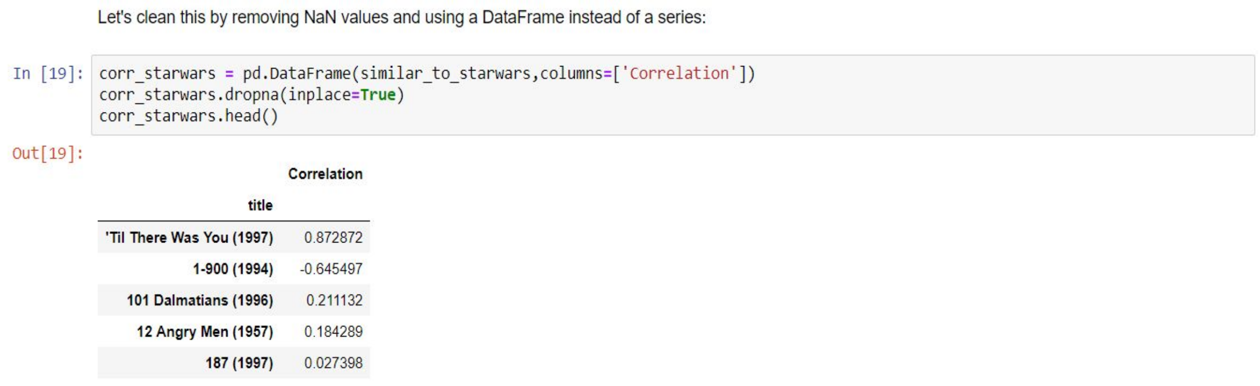
```
Out[17]: user_id
         1    5.0
         2    5.0
         3    NaN
         4    5.0
         5    4.0
         Name: Star Wars (1977), dtype: float64
```

We can then use corrwith() method to get correlations between two pandas series:

```
In [18]: similar_to_starwars = moviemat.corrwith(starwars_user_ratings)
         similar_to_liarliar = moviemat.corrwith(liarliar_user_ratings)
```

```
C:\Users\This pc\Anaconda3\lib\site-packages\numpy\lib\function_base.py:2522: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar)
C:\Users\This pc\Anaconda3\lib\site-packages\numpy\lib\function_base.py:2451: RuntimeWarning: divide by zero encountered in tru
e_divide
  c *= np.true_divide(1, fact)
```

12. Cleaning of values which have null i.e.NAN values and using a DataFrame instead of series.



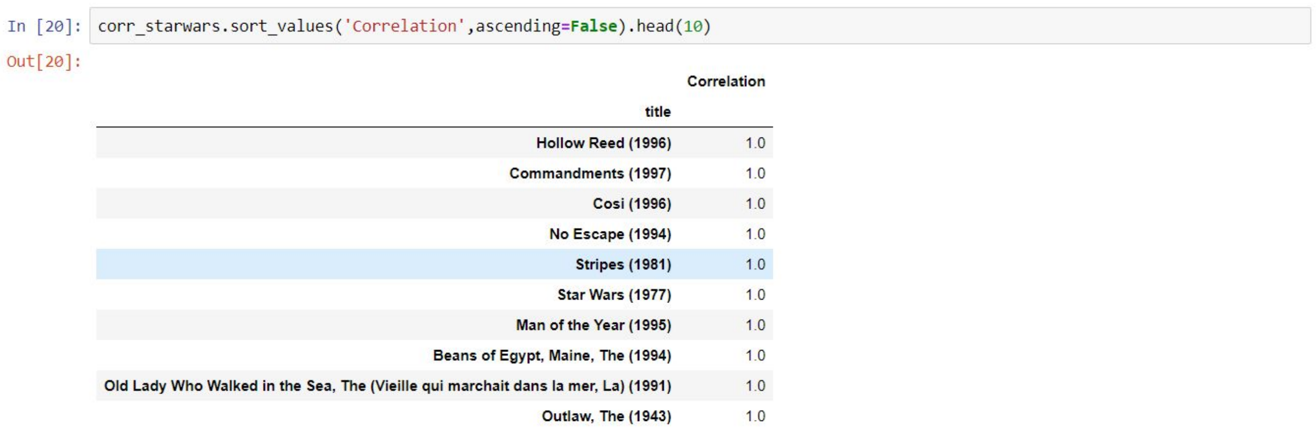
Let's clean this by removing NaN values and using a DataFrame instead of a series:

```
In [19]: corr_starwars = pd.DataFrame(similar_to_starwars,columns=['Correlation'])
         corr_starwars.dropna(inplace=True)
         corr_starwars.head()
```

Out[19]:

title	Correlation
'Til There Was You (1997)	0.872872
1-900 (1994)	-0.645497
101 Dalmatians (1996)	0.211132
12 Angry Men (1957)	0.184289
187 (1997)	0.027398

13. Now if we sort the data frame by correlation, we should get the most similar movies, however note that we get some results that don't really make sense. This is because there are a lot of movies only watched once by the users who also watched star wars (it was the most popular movie).

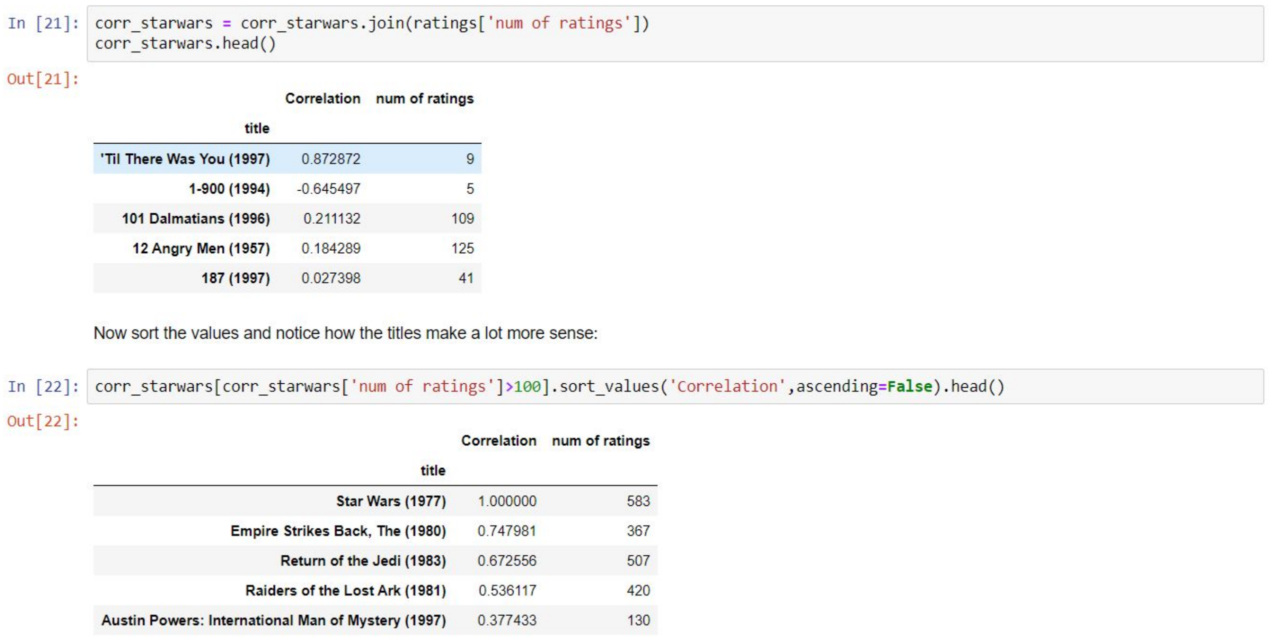


```
In [20]: corr_starwars.sort_values('Correlation',ascending=False).head(10)
Out[20]:
```

title	Correlation
Hollow Reed (1996)	1.0
Commandments (1997)	1.0
Cosi (1996)	1.0
No Escape (1994)	1.0
Stripes (1981)	1.0
Star Wars (1977)	1.0
Man of the Year (1995)	1.0
Beans of Egypt, Maine, The (1994)	1.0
Old Lady Who Walked in the Sea, The (Vieille qui marchait dans la mer, La) (1991)	1.0
Outlaw, The (1943)	1.0

14. Let's fix this by filtering out movies that have less than 100 reviews(this value was chosen based off the histogram from earlier).

And then sorting the values and notice how the titles make a lot more sense.



```
In [21]: corr_starwars = corr_starwars.join(ratings['num of ratings'])
         corr_starwars.head()
Out[21]:
```

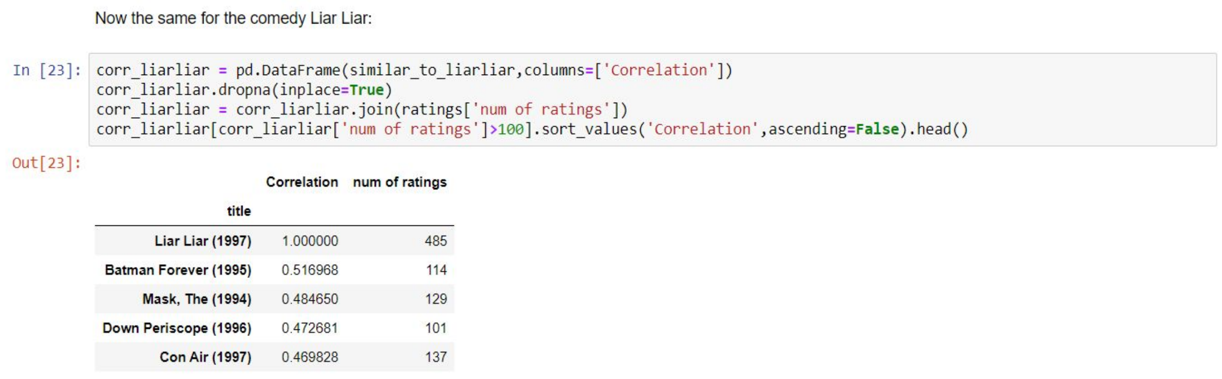
title	Correlation	num of ratings
'Til There Was You (1997)	0.872872	9
1-900 (1994)	-0.645497	5
101 Dalmatians (1996)	0.211132	109
12 Angry Men (1957)	0.184289	125
187 (1997)	0.027398	41

Now sort the values and notice how the titles make a lot more sense:

```
In [22]: corr_starwars[corr_starwars['num of ratings']>100].sort_values('Correlation',ascending=False).head()
Out[22]:
```

title	Correlation	num of ratings
Star Wars (1977)	1.000000	583
Empire Strikes Back, The (1980)	0.747981	367
Return of the Jedi (1983)	0.672556	507
Raiders of the Lost Ark (1981)	0.536117	420
Austin Powers: International Man of Mystery (1997)	0.377433	130

15. Now the same for comedy Liar Liar.


Now the same for the comedy Liar Liar:

```
In [23]: corr_liarliar = pd.DataFrame(similar_to_liarliar,columns=['Correlation'])
         corr_liarliar.dropna(inplace=True)
         corr_liarliar = corr_liarliar.join(ratings['num of ratings'])
         corr_liarliar[corr_liarliar['num of ratings']>100].sort_values('Correlation',ascending=False).head()
```

Out[23]:

	Correlation	num of ratings
title		
Liar Liar (1997)	1.000000	485
Batman Forever (1995)	0.516968	114
Mask, The (1994)	0.484650	129
Down Periscope (1996)	0.472681	101
Con Air (1997)	0.469828	137

CHAPTER 9:

ISSUES FACED

9.1 Scalability Issues

One of the major challenges in working with the 20 million dataset is memory constraints. The data cannot be stored as a dense matrix due to its huge size. We have to make use of sparse matrix representations in order for the program to work without memory issues. Further, intermediate results such as the user-user similarity matrix cannot be computed and stored due to the huge memory footprint. We had to think of ways to compute the similarity values as and when needed. Further, the 20 million dataset also needed a lot of time ti run. We were able to overcome the time requirements by writing parallelized implementations of the algorithms using Apache Spark.

9.2 Broken links

As mentioned earlier, meta-data about the movies was collected by scrapping details from the IMDB pages site. The smaller dataset provided auto-generated links to the movies url based on the movies title and release year. This caused a large portion of the links to broken. Some titles were ambiguity leading to a search page with recommendations rather the the movie page. For others, there was some sort of error in the reference to the link. Some example of these errors was usage of a secondary foreign title instead of the English one, usage of a former title, and incorrect year of the movie. As a result, almost a third of the links were broken, and had to be corrected before the data could be used. To fix this, we decided to use a different dataset where the IMDB movie id was provided instead, which was easier to use.

CHAPTER 10:

POSSIBLE FUTURE WORK

There are plenty of way to expand on the work done in this project. Firstly, the content based method can be expanded to include more criteria to help categorize the movies. The most obvious ideas is to add features to suggest movies with common actors, directors or writers. In addition, movies released within the same time period could also receive a boost in likelihood for recommendation. Similarly, the movies total gross could be used to identify a users taste in terms of whether he/she prefers large release blockbusters, or smaller indie films. However, the above ideas may lead to overfitting, given that a users taste can be highly varied, and we only have a guarantee that 20 movies (less than 0.2%) have been reviewed by the user. In addition, we could try to develop hybrid methods that try to combine the advantages of both content-based methods and collaborative filtering into one recommendation system.

REFERENCES

[1] https://grouplens.org/datasets/movielens/100k/ - MovieLens Dataset.

[2] https://en.wikipedia.org › wiki › Recommender_system

[3] https://www.geeksforgeeks.org/python-implementation-of-movie-recommender-system/

[4] https://www.kaggle.com/rounakbanik/movie-recommender-systems