

# Case Study 9:Netflix Movie Recommendation System (Collaborative based recommendation)

## 7.1 Business/Real world problem : Problem definition

This case study is based on the competition hosted by Netflix.

This competition began on October 2,2008 and ended on October 2,2011.

The prize amount was 1 million US dollars. It's a huge cash prize.

What is Netflix ?

Netflix is an online streaming platform where the users can stream movies,web series and much more. Netflix would recommend movies or web series or any other video content based on your watch history. For example, if you use Netflix more to watch horror movies, then when you visit the Netflix next time, it will recommend you top horror movies. This is one such simple example. But, we tend to watch multiple genres of movies based on our current mood,references made by our friends,bias towards certain actors, etc. So, Netflix needs to encompass all of these and need to give you good recommendations.

Let's make it formal

- 1.)We have a set of users where a ith user is denoted as  $u_i$ .
- 2.)We have a set of movies where a ith movie is denoted as  $m_i$ .
- 3.)We have a set of ratings where a  $i,j$  rating is defined as the rating given by user  $u_i$  on movie  $m_j$ . The possible values for ratings are [1,2,3,4,5].
- 4.)Netflix collects all those ratings data which are provided by the users.
- 5.)Netflix internally uses a system called **Cinematch** for recommendations. Cinematch's job is to predict whether someone will enjoy a movie based on how much they liked or disliked other movies.

6.) The aim of this competition is to do better than the existing system called Cinematch which Netflix uses. If any participant surpassed 10% of Cinematch performance, then the participant would be declared as the winner. The performance metric used here is Root Mean Squared Error.

7.) For more details on this competition, please refer to [this](#)

## 7.2 Objectives and constraints

### Objectives :

- 1.) Predict the rating that a user would give to a movie that he has not yet rated.
- 2.) Minimize the difference between predicted and actual rating (RMSE and MAPE). We will use RMSE here.

### Constraints :

- 1.) Some form of interpretability. Our model should explain why a particular user  $u_i$  given a rating  $r_{ij}$  to movie  $m_j$  i.e, if a user gives a higher rating to a particular movie, then our algorithm or model should explain the reason behind it.

Let's look at the goal of our solution.

**1.4 Real world/Business Objectives and constraints**

Objectives:

1. Predict the rating that a user would give to a movie that he has not yet rated.
2. Minimize the difference between predicted and actual rating (RMSE and MAPE)

Constraints:

1. Some form of interpretability.

**2. Machine Learning Problem**

**2.1 Data**

Timestamp : 02:17

- 1.)  $r_{ij}$  represents the actual rating given by user  $u_i$  on movie  $m_j$ .
- 2.)  $\hat{r}_{ij}$  (with a cap) represents the predicted rating by our model or system or algorithm given by user  $u_i$  on movie  $m_j$ .
- 3.) The goal is to minimize the difference across all the ratings given by user  $u_i$  on movie  $m_j$  i.e, for all possible combinations of  $i$  and  $j$ .
- 4.) If we minimize this difference squared term, then we are doing better. This is because the difference squared term is minimized only if the two terms are close enough to each other or  $r_{ij}$  (with a cap) is close to  $\hat{r}_{ij}$ .

### 7.3 Mapping it to the ML problem: Data overview.

#### Data :

Data files : Get the data from [here](#)

- 1.)combined\_data\_1.txt
- 2.)combined\_data\_2.txt
- 3.)combined\_data\_3.txt
- 4.)combined\_data\_4.txt
- 5.)movie\_titles.csv : It contains movie\_id and the movie name

## Example data point

1: -> movie\_id  
148844,3,2005-09-06

The above says that user id of 148844 watched the movie with movie id 1 and gave a rating 3. This was done on 2005-09-06 ( YYYY-MM-DD ).

A screenshot of a Jupyter Notebook interface. The title of the notebook is "Netflix\_Movie.ipynb". The code cell contains the following text:

```
2.1.2 Example Data point
1: Movie Id
148844,3,2005-09-06
YYYY-MM-DD
User Id
822109,5,2005-05-13
885013,4,2005-10-19
30878,4,2005-12-26
823519,3,2004-05-03
893988,3,2005-11-17
124105,4,2004-08-05
1248029,3,2004-04-22
1842128,4,2004-05-09
2238063,3,2005-05-11
1503895,4,2005-05-19
2207774,5,2005-06-06
2590061,3,2004-08-12
2442,3,2004-04-14
543865,4,2004-05-29
```

Timestamp : 01:50

Note : Ratings are from 1 to 5 i.e, {1,2,3,4,5}

MovieIDs range from 1 to 17,770 in a sequence.

CustomerIDs range from 1 to 2649429 with gaps. There are 480189 users.

## 7.4 Mapping to an ML problem:ML problem formulation

The screenshot shows a Jupyter Notebook interface with the following content:

- Section Title:** 2.2 Mapping the real world problem to a Machine Learning Problem
- Section Subtitle:** 2.2.1 Type of Machine Learning Problem
- Text:** For a given movie and user we need to predict the rating would be given by him/her to the movie.  
The given problem is a Recommendation problem  
It can also seen as a Regression problem
- Section Subtitle:** 2.2.2 Performance metric
- List:**
  - Mean Absolute Percentage Error: [https://en.wikipedia.org/wiki/Mean\\_absolute\\_percentage\\_error](https://en.wikipedia.org/wiki/Mean_absolute_percentage_error)
  - Root Mean Square Error: [https://en.wikipedia.org/wiki/Root-mean-square\\_deviation](https://en.wikipedia.org/wiki/Root-mean-square_deviation)

Timestamp : 0:29

The above is a matrix where a cell  $u_{imj}$  contains the rating  $r_{ij}$  given by user  $u_i$  on movie  $m_j$ .

We have some training data, where some ratings are given. Our task is to fill the rest of the matrix.

It can be seen as a regression problem. Here the  $y_i$ 's are the ratings. But  $x_i$ 's are not explicitly given as features.

We will use that matrix and somehow invent the features  $x_i$ 's.

So, now it can be treated as a regression problem as we have  $x_i$ 's and  $y_i$ 's.

**Performance Metric :**

- 1.) Mean Absolute Percentage Error
- 2.) Root Mean Square Error

We will minimize the RMSE.

We will try to provide some interpretability.

## 7.5 Exploratory Data Analysis:Data preprocessing

In the data.csv file , you will see the data in the format which is given below.

Movie\_id,User\_id,Rating,Date.

For example, 1,1488844,3,2005-09-06.

Some statistics about the ratings.

The screenshot shows a Jupyter Notebook interface with several tabs at the top: 'Contents - Google Docs', 'localhost:8888/notebooks/Netflix\_Movie.ipynb', 'Surprise - A Python scikit for...', and 'Factor in the neighbors: Scaler...'. The main area shows a preview of the data and the output of a code cell.

Preview of the data:

	Movie_id	User_id	Rating	Date	
5	56431994	10341	510180	4	1999-11-11
6	9056171	1798	510180	5	1999-11-11
7	58698779	10774	510180	3	1999-11-11
8	48101611	8651	510180	2	1999-11-11
9	81893208	14660	510180	2	1999-11-11

Code cell output:

```
In [7]: df.describe()['rating']
Out[7]: count    1.004805e+08
mean     3.603290e+00
std      1.085219e+00
min      0.000000e+00
25%     3.000000e+00
50%     4.000000e+00
75%     4.000000e+00
max     5.000000e+00
Name: rating, dtype: float64
```

Timestamp : 04:42

The average rating is 3.6. This means on average the ratings are above and below 3.6.

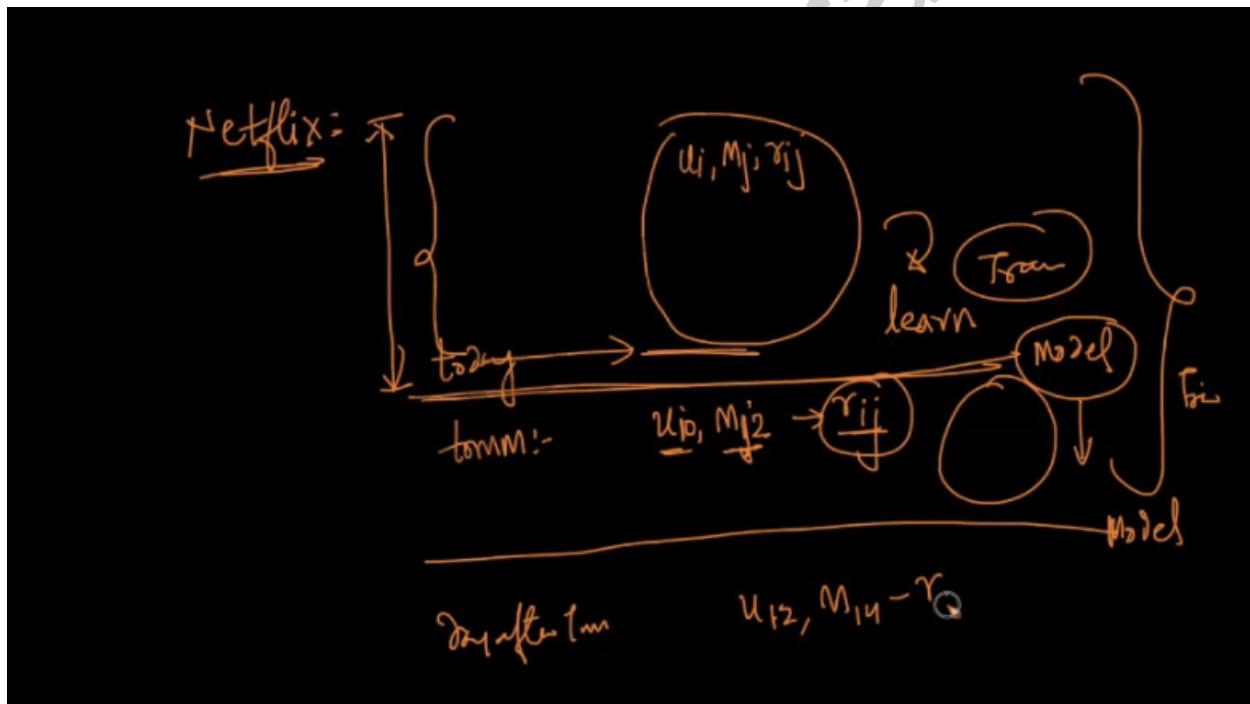
Then , we remove the NaN and duplicate values.

Total number of ratings : 100480507

Total number of users : 480189

Total number of movies : 17770

## 7.6 Exploratory Data Analysis:Temporal Train-Test split.

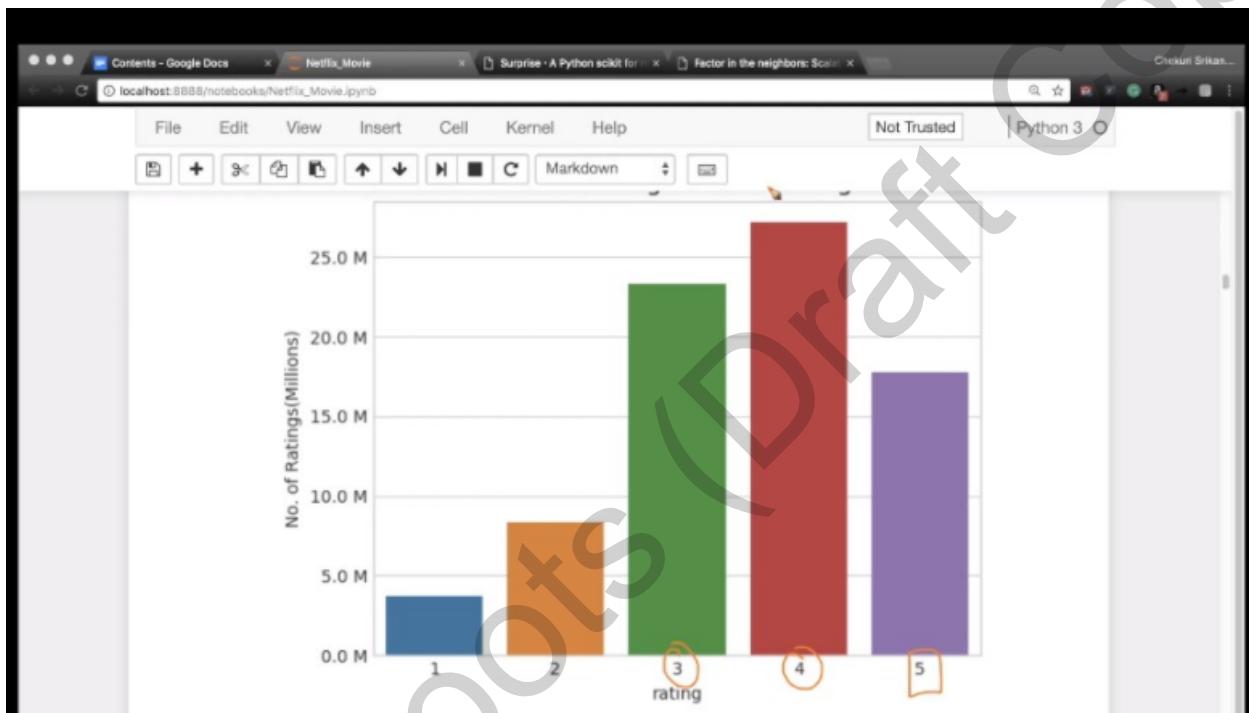


Timestamp : 02:23

As the ratings provided by the user  $u_i$  on movie  $m_j$  also depends on the time. In simple terms, the data has temporal nature. So, while splitting the data, we need to incorporate this.

So, basically sort the data in ascending order of dates i.e, the oldest date first. Then simply split in the ratio of 80:20.

## 7.7 Exploratory Data Analysis:Preliminary data analysis.

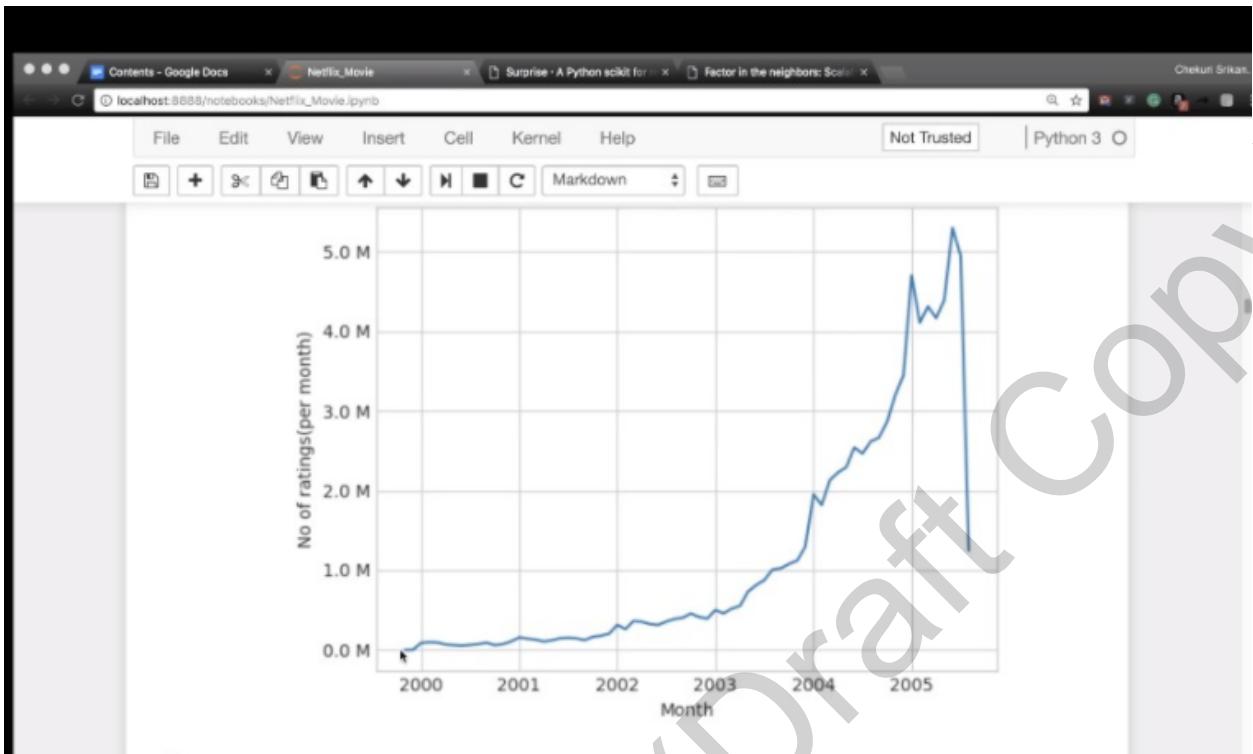


Timestamp : 00:52

Most of the users gave a rating of 4. This is clearly evident as the average rating is 3.6 which we saw before.

Only very few of the users gave a rating of 1.

So, typically users tend to give higher ratings.



Timestamp : 01:54

The ratings tend to increase as we progress through the years and then flattens out. This means, netflix as a company gave better content over the years as users liked it much.

Contents - Google Docs   Netflix\_Movie   Surprise - A Python toolkit for...   Factor in the neighbors: Sc...   Checkin Srikan...

localhost:8888/notebooks/Netflix\_Movie.ipynb

File Edit View Insert Cell Kernel Help Not Trusted Python 3

In [20]: `no_of_rated_movies_per_user = train_df.groupby(by='user')['rating'].count()`

`no_of_rated_movies_per_user.head()`

Out[20]: user  
305344 → 17112  
2439493 15896  
387418 15402  
1639792 9767  
1461435 → 9447  
Name: rating, dtype: int64

# movies rated by a user

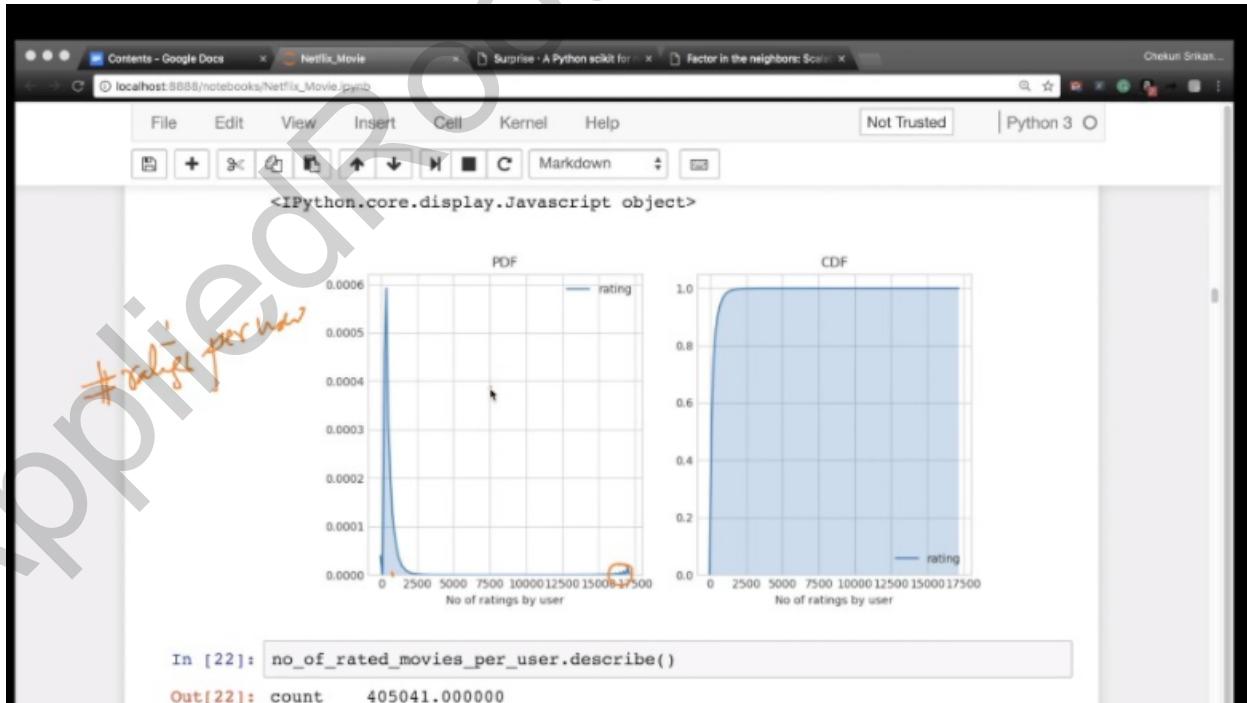
In [21]: `fig = plt.figure(figsize=plt.figaspect(.5))`

`ax1 = plt.subplot(121)`  
`sns.kdeplot(no_of_rated_movies_per_user, shade=True, ax=ax1)`  
`plt.xlabel('No of ratings by user')`  
`plt.title("PDF")`

`ax2 = plt.subplot(122)`  
`sns.kdeplot(no_of_rated_movies_per_user, shade=True, cumulative=True, ax=ax2)`  
`plt.xlabel('No of ratings by user')`  
`plt.title('CDF')`

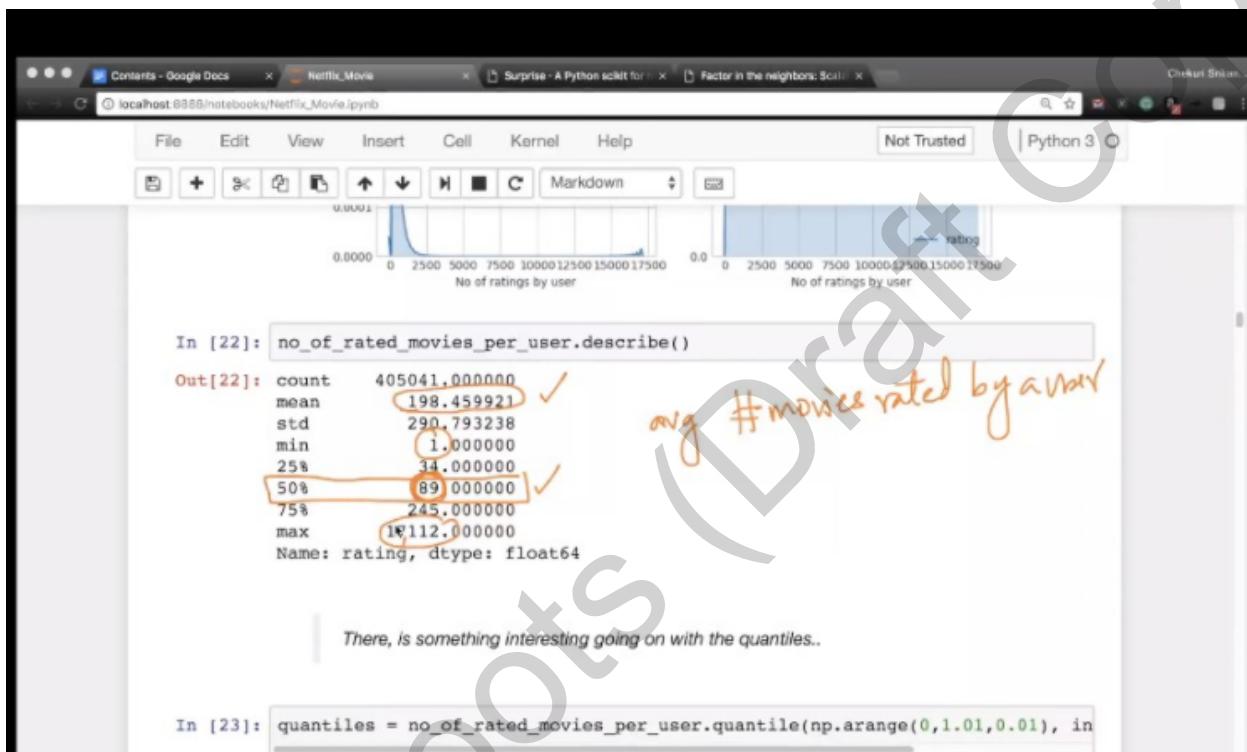
Timestamp : 04:30

The above data shows how many movies are rated by a user. The first record seems to be abnormal as the user rated for 17,112 movies.



Timestamp : 05:02

The above is the pdf where the x axis represents the number of ratings by user and the y-axis represents the frequency of it. We can observe that only very few gave higher ratings.



Timestamp : 06:41

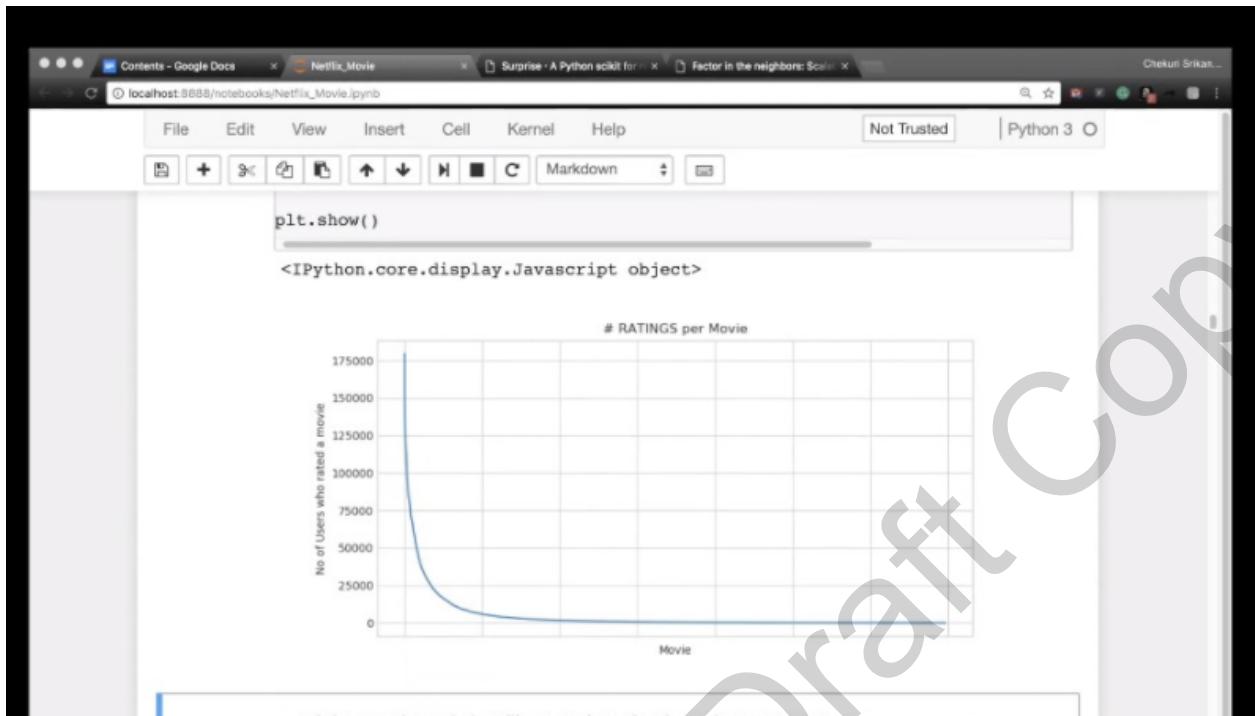
Some of the statistics are shown above.

```
In [25]: quantiles[::-5]
Out[25]: 0.00      1
0.05      7
0.10     15
0.15     21
0.20     27
0.25     34
0.30     41
0.35     50
0.40     60
0.45     73
0.50     89
0.55    109
0.60    133
0.65    163
0.70    199
0.75    245
0.80    307
0.85    392
0.90    520
0.95    749
1.00   17112
Name: rating, dtype: int64
```

Timestamp : 08:29

We can see only 5% of the users gave the number of ratings more than 749 and the maximum is 17112 which is abnormal.

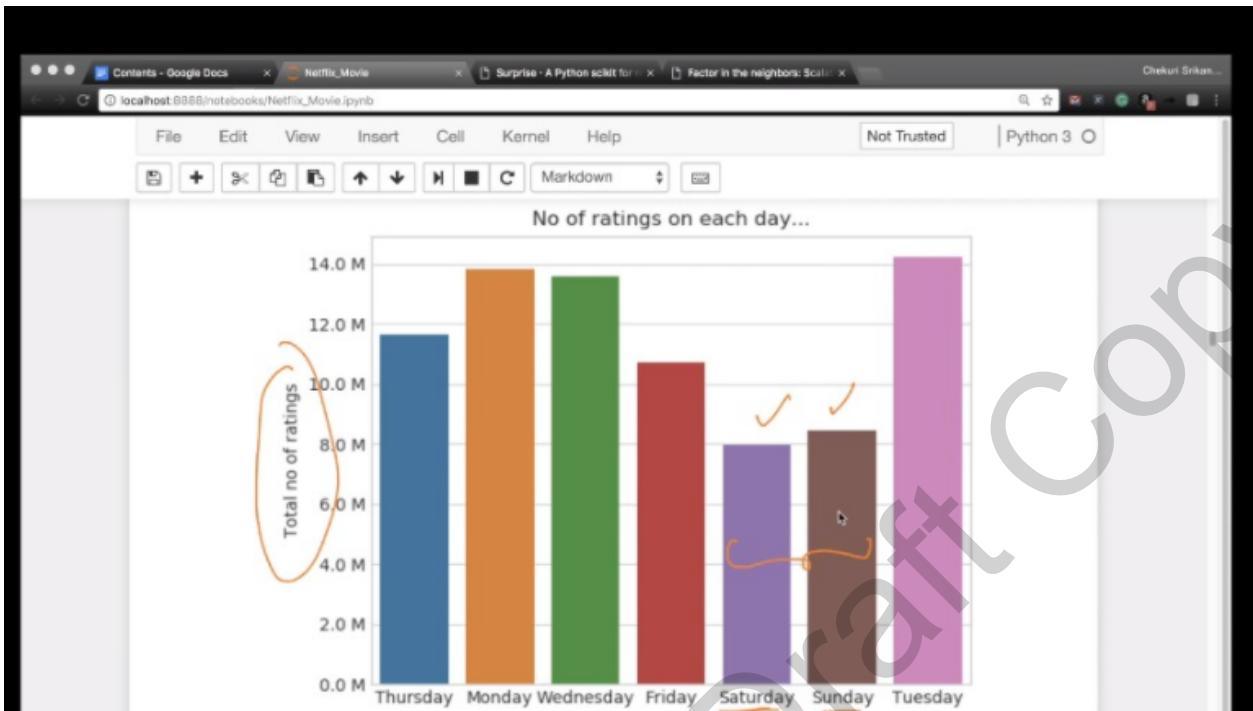
### Analysis of ratings of a movie given by a user.



Timestamp : 09:47

X-axis : Movie\_id and y-axis : Number of users who rated that movie.

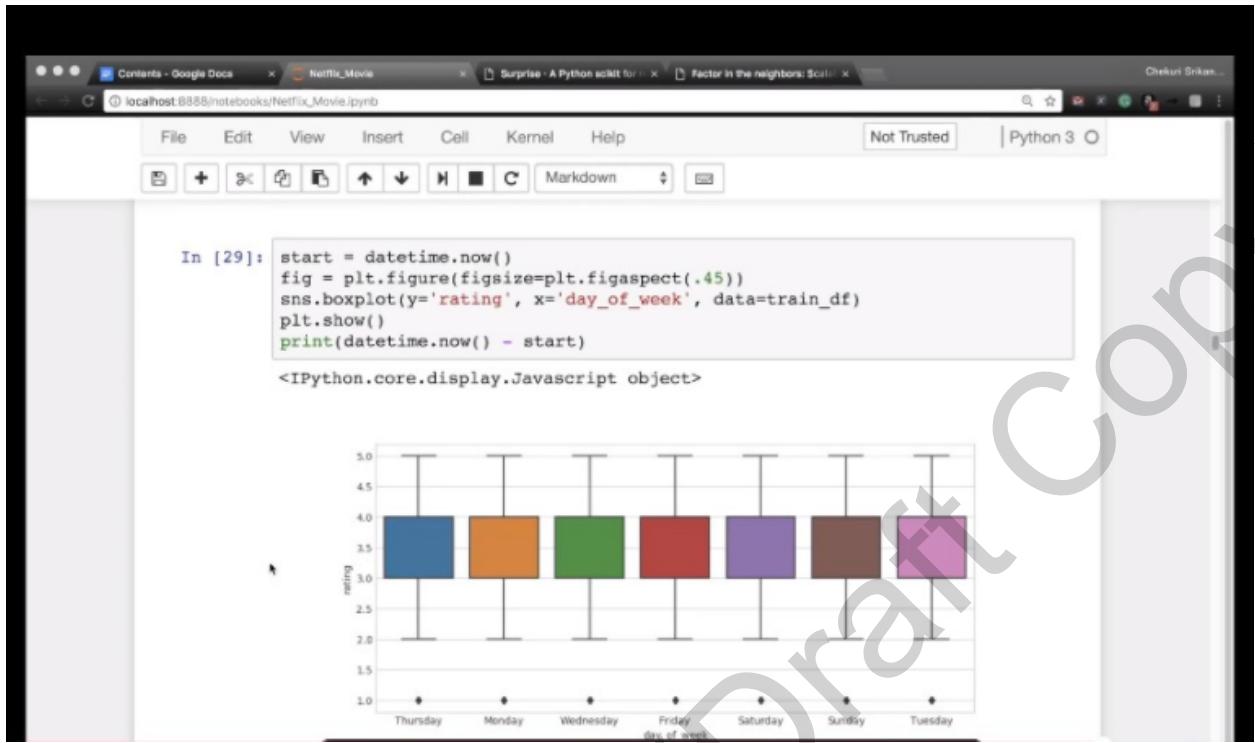
It's a skewed distribution.



Timestamp : 11:24

Number of ratings on each day. On Tuesday, the users tend to rate more movies.

Can we use day of week for predicting the rating ?



Timestamp :12:45

As we can see from the box plot above, it's not a good feature to work on.

## 7.8 Exploratory Data Analysis:Sparse matrix representation

The screenshot shows a Jupyter Notebook interface with several tabs at the top: 'Contents - Google Docs', 'Netflix\_Movie', 'scipy.sparse.csr\_matrix - Sc', 'Surprise - A Python scikit for...', 'Factor in the neighbors: Scala...', and 'Chokud Srikan...'. The main area has a toolbar with File, Edit, View, Insert, Cell, Kernel, Help, Not Trusted, and Python 3. Below the toolbar is a menu bar with icons for file operations like Open, Save, and Print, along with a 'Markdown' button. The notebook content starts with a section titled '3.3.6 Creating sparse matrix from data frame'. Handwritten notes in orange highlight the word 'date' and show arrows pointing to 'T' and 'test'. Below this is a table:

MOVIE_ID	USER_ID	RATING
1	1	3
2	1	4
3	1	2
3	2	1
4	2	4
5	2	2
1	3	3
7	3	1
10	3	5

An arrow points from the table to a sparse matrix representation:

```
1 2 3 4 5 6 7 8 9 10 (movie)
1 3 4 2 - - - - -
2 - - 1 4 - - - -
3 3 - - - - 1 - - 5
(user)
```

Below this is a sub-section titled '3.3.6.1 Creating sparse matrix from train data frame' with the following code in the 'In [32]:' cell:

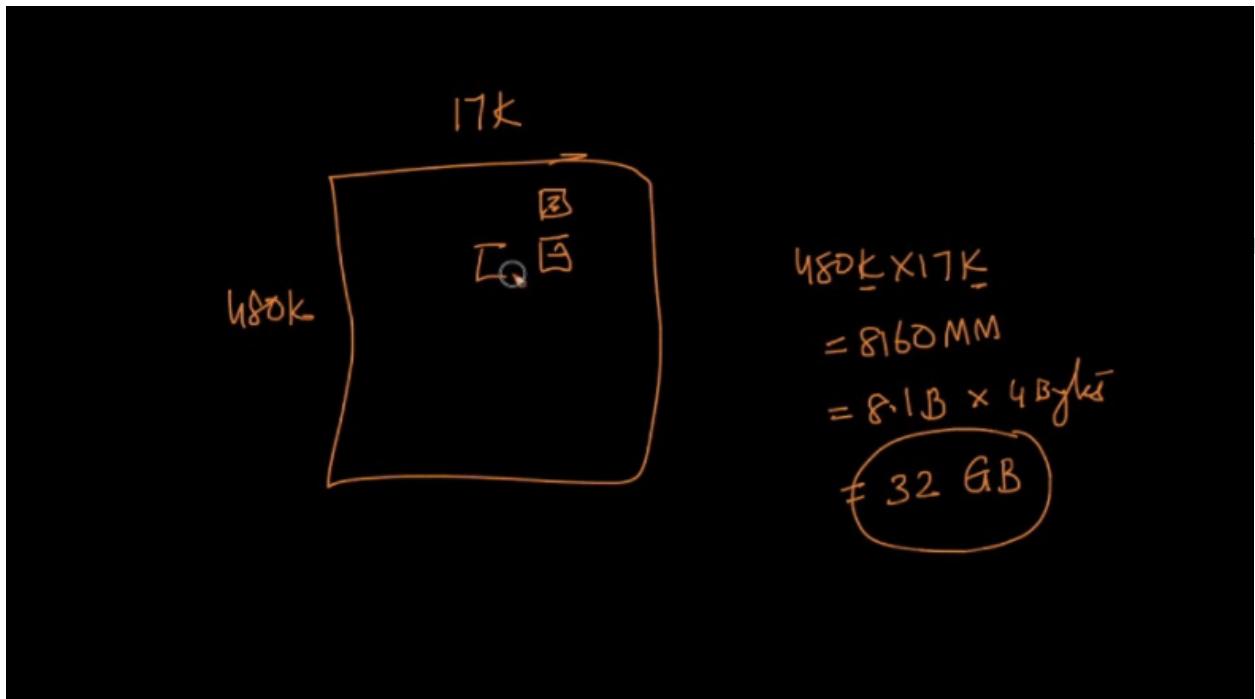
```
In [32]: start = datetime.now()
if os.path.isfile('train_sparse_matrix.npz'):
    print("It is present in your pwd, getting it from disk....")
    # just get it from the disk instead of computing it
    train_sparse_matrix = sparse.load_npz('train_sparse_matrix.npz')
    print("DONE..")
else:
```

Timestamp : 0:18

We are simply converting it to a matrix where each cell represents the rating given by a user ui on movie mj. This is a sparse matrix since some of the entries are missing. This is because a particular user can't give ratings to all movies. The user can only give ratings to a subset of movies.

Using the `csr_matrix()` function, we can convert it to a compressed sparse matrix.

The data is extremely sparse.

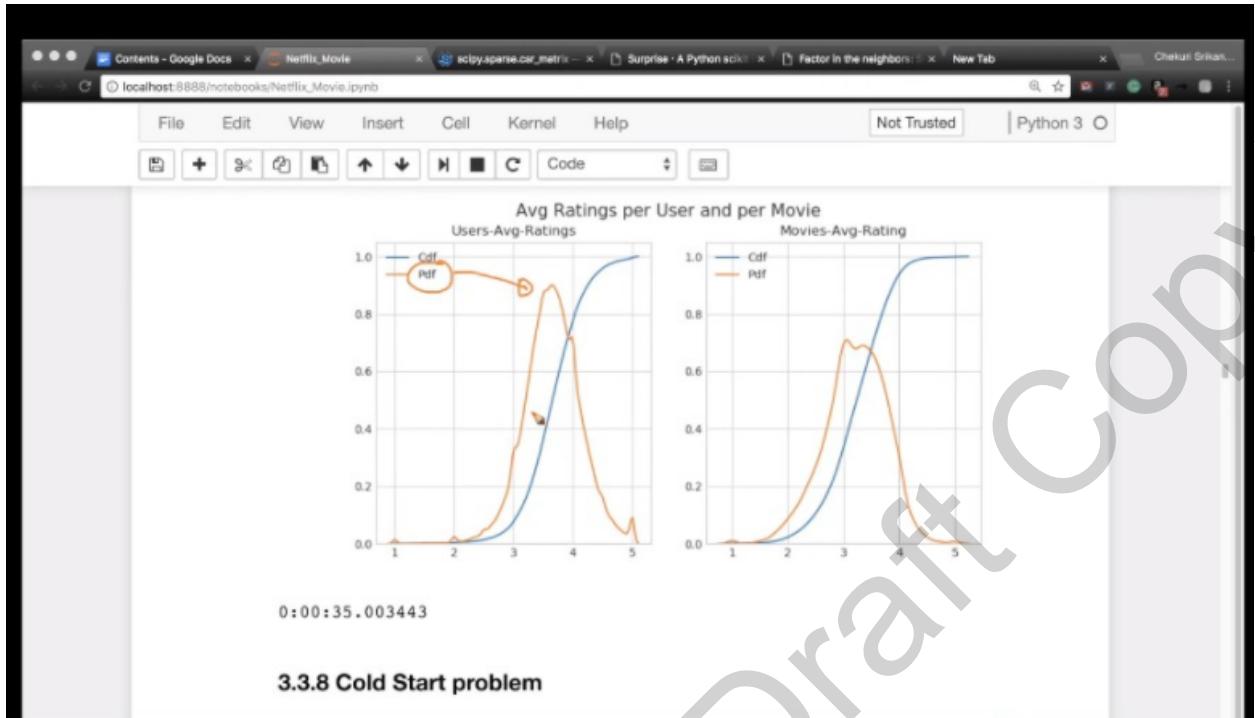


Timestamp : 03:36

Loading the entire matrix requires 32 GB of memory. But, we know that some of the entries are not filled. So, we can use this property and store and retrieve it much more efficiently.

## 7.9 Exploratory Data Analysis: Average ratings for various slices

- 1.) Global mean of all ratings.
- 2.) User averages : For each user, compute the average of the ratings provided by that user.
- 3.) Movie averages : For each movie, compute the average of the ratings.



Timestamp : 05:13

The above resembles an approximation of gaussian distribution.

## 7.10 Exploratory Data Analysis:Cold start problem

Since the data is temporal in nature, we splitted accordingly. But there is a catch. There may be some users who are present in the training data but not in the test data. For these users, we don't have any information. Note : New movies are going to be released and new users also sign up on the website.

There are a total of 480189 users.

Number of users in train data : 405041

Number of users in test but not in train : 75,148 ( Roughly 15% of the data ).

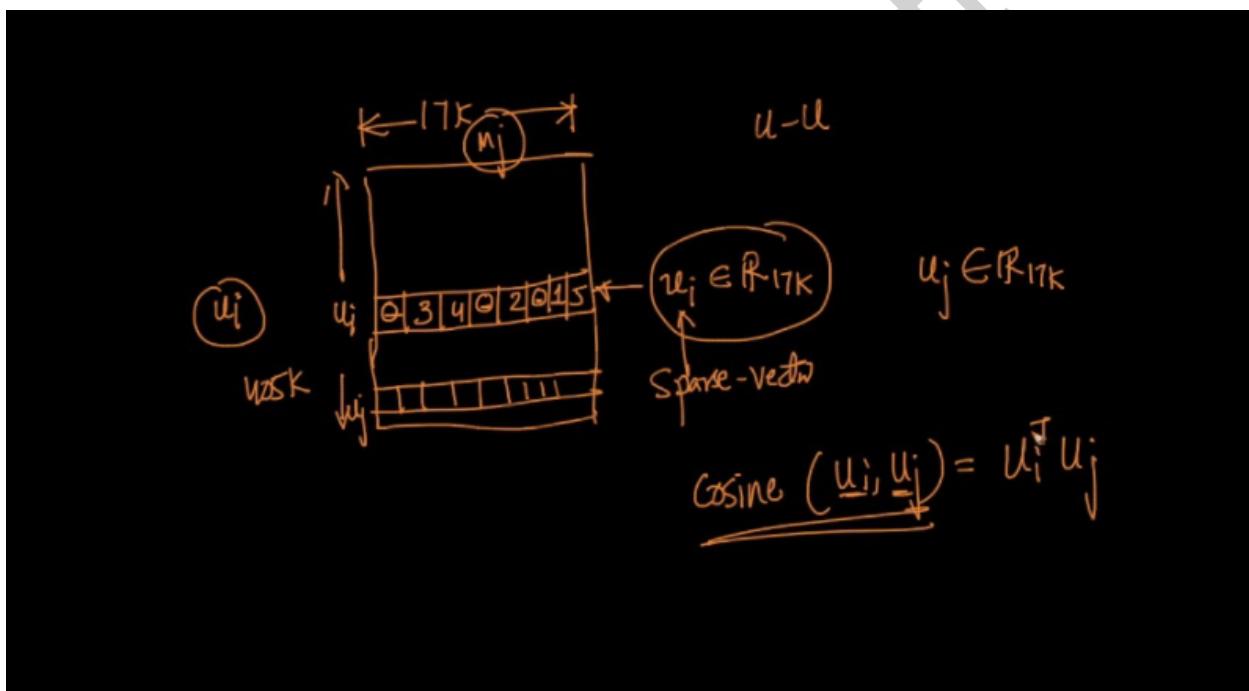
There are a total of 17,770 movies.

Number of users in train data : 17424

Number of users in test but not in train : 346 ( Roughly 1.95 % of the data ).

## 7.11 Computing Similarity matrices:User-User similarity matrix

Computing the user-user similarity matrix is hard and computationally infeasible as we have more number of users.



Timestamp : 02:26

$U_i$  belongs to a 17k dimensional space.  $U_j$  belongs to a 17k dimensional space.

The similarity between  $U_i$  and  $U_j$  is defined as  $\text{cosine}(U_i, U_j) = U_i^T U_j$  (dot product).

Both  $U_i$  and  $U_j$  are sparse vectors.

User-User similarity matrix. There are 405k users. So, the similarity matrix would be of shape 405k\*405k since for every pair of users, we need the similarity value.

$$A = u_i \begin{bmatrix} u_j \\ \vdots \end{bmatrix}_{405K \times 17K}$$

$$S_u = u_i \begin{bmatrix} u_j \\ \vdots \end{bmatrix}_{405K \times 405K}$$

$$(S_u)_{ij} = u_i^T u_j$$

Timestamp : 04:06

The above S matrix is the user-user similarity matrix. Each cell  $u_{ij}$  contains the similarity value between  $u_i$  and  $u_j$ . This matrix is not sparse.

The above matrix is symmetric i.e,  $S_{uij} = S_{uj}$  i.e, the value in the cell  $u_{ij}$  is equal to the value in the cell  $u_{ji}$ . This is because  $\text{cosine}(u_{ij}) = \text{cosine}(u_{ji})$ . So, computing  $S_{uij}$  is enough and we don't need to compute  $S_{uj}$ . So, there are  $405k * 405k / 2$  values we need to compute.

We can parallelize the operation of computing the similarity matrix. But still, it's computationally infeasible.

Idea : The user vector belongs to 17k dimensional space. Can we use any dimensionality reduction techniques to reduce the dimensionality ? But still, it doesn't work.

## Why ?

Suppose if we reduce the dimension of a user vector from 17k to 500. Then, the values in the vector would be dense rather than sparse. So, we still need to do 500 multiplication and addition in the computation of dot product. Earlier, the user vector with 17k was sparse, so some of the entries were zero. This reduced the computation.

An alternative approach is as follows :

The screenshot shows a Google Docs notebook with the following content:

**Is there any other way to compute user user similarity..??**

-An alternative is to compute similar users for a particular user, whenever required (ie., Run time)

- We maintain a binary Vector for users, which tells us whether we already computed or not..
- \*\*\*If not\*\*\* :
  - Compute top (let's just say, 1000) most similar users for this given user, and add this to our datastructure, so that we can just access it(similar users) without recomputing it again.
  - 
  - \*\*\*If It is already Computed\*\*\*:
    - Just get it directly from our datastructure, which has that information.
    - In production time, We might have to recompute similarities, if it is computed a long time ago. Because user preferences change over time. If we could maintain some kind of Timer, which when expires, we have to update it ( recompute it ).
    -
  - \*\*\*Which datastructure to use:\*\*\*

Handwritten annotations include:

  - A circled "Su" on the left.
  - A circled "Ui" with an arrow pointing to "Sim" and the value "8.88e-1".
  - A circled "isComputed" with arrows pointing to "t" and "u1" and "u2".

Timestamp : 16:50

Which data structure to use ?

```
- ***Which datastructure to use:***
- It is purely implementation dependant.
- One simple method is to maintain a **Dictionary Of Dictionaries**.
- 
- **key :** _userid_
- _value_ : _Again a dictionary_
  - __key__ : _Similar User_
  - __value__ : _Similarity Value_
```

Timestamp : 17:14

## 7.12 Computing Similarity matrices:Movie-Movie similarity

Each mi (movie vector) is of 405k dimensions and it's sparse too. We have a total of 17k movies. So, the shape of the similarity matrix is 17k \* 17k.

It's a symmetric matrix. So, we need to perform 17k \*17k/2 values.

How to find top similar movies to a particular movie ?

It's simple to find. Let's say we want to find top similar movies to movie id 3. We have the similarity matrix right ? Take the 3rd row from the similarity matrix. This row contains the similarity values between movie id 3 and all other movies. Now, sort it in descending order. Now, choose the top k values. So, these movies are most similar to movie id 3.

## 7.13 Computing Similarity matrices:Does movie-movie similarity work?

Yes, it works.

Top 10 similar movies

```
In [68]: movie_titles.loc[sim_indices[:10]]
```

```
Out[68]:
```

movie_id	year_of_release	title
323	1999.0	Modern Vampires
4044	1998.0	Subspecies 4: Bloodstorm
1688	1993.0	To Sleep With a Vampire
13962	2001.0	Dracula: The Dark Prince
12053	1993.0	Dracula Rising
16279	2002.0	Vampires: Los Muertos
4667	1996.0	Vampirella
1900	1997.0	Club Vampire
13873	2001.0	The Breed
15867	2003.0	Dracula II: Ascension

Timestamp : 03:59

We are printing the top similar movies to Vampire Journals. We can see that those sets of movies are very similar.

## 7.14 ML Models: Surprise library

Surprise is a Python scikit building and analyzing recommender systems.

The screenshot shows a web browser window with multiple tabs open. The active tab is for the Surprise library, which is described as a Python scikit for recommender systems. The page features a teal sidebar on the left with links to Home, Documentation, GitHub page, and social media icons. The main content area has a white background with a hand-drawn style illustration of a document labeled  $[u_i, M_j, Y_j]$ . The 'Overview' section includes a brief description of the library's purpose and a list of design purposes. The list highlights various features like control over experiments, built-in datasets, ready-to-use prediction algorithms (SVD, PMF, NMF), and cross-validation tools.

**Overview**

Surprise is a Python scikit building and analyzing recommender systems.

Surprise was designed with the following purposes in mind:

- Give users perfect control over their experiments. To this end, a strong emphasis is laid on documentation, which we have tried to make as clear and precise as possible by pointing out every detail of the algorithms.
- Alleviate the pain of Dataset handling. Users can use both *built-in* datasets (MovieLens, Jester), and their own *custom* datasets.
- Provide various ready-to-use prediction algorithms such as baseline algorithm, neighborhood methods, matrix factorization-based (SVD, PMF, SVD++, NMF), and many others. Also, various similarity measures (cosine, MSD, pearson...) are built-in.
- Make it easy to implement new algorithm ideas.
- Provide tools to evaluate, analyse and compare the algorithms performance. Cross-validation procedures can be run very easily using powerful CV iterators (inspired by scikit-learn excellent tools), as well as exhaustive search over a set of parameters.

The name *SurPRISE* (roughly :) stands for Simple Python Recommendation System Engine.

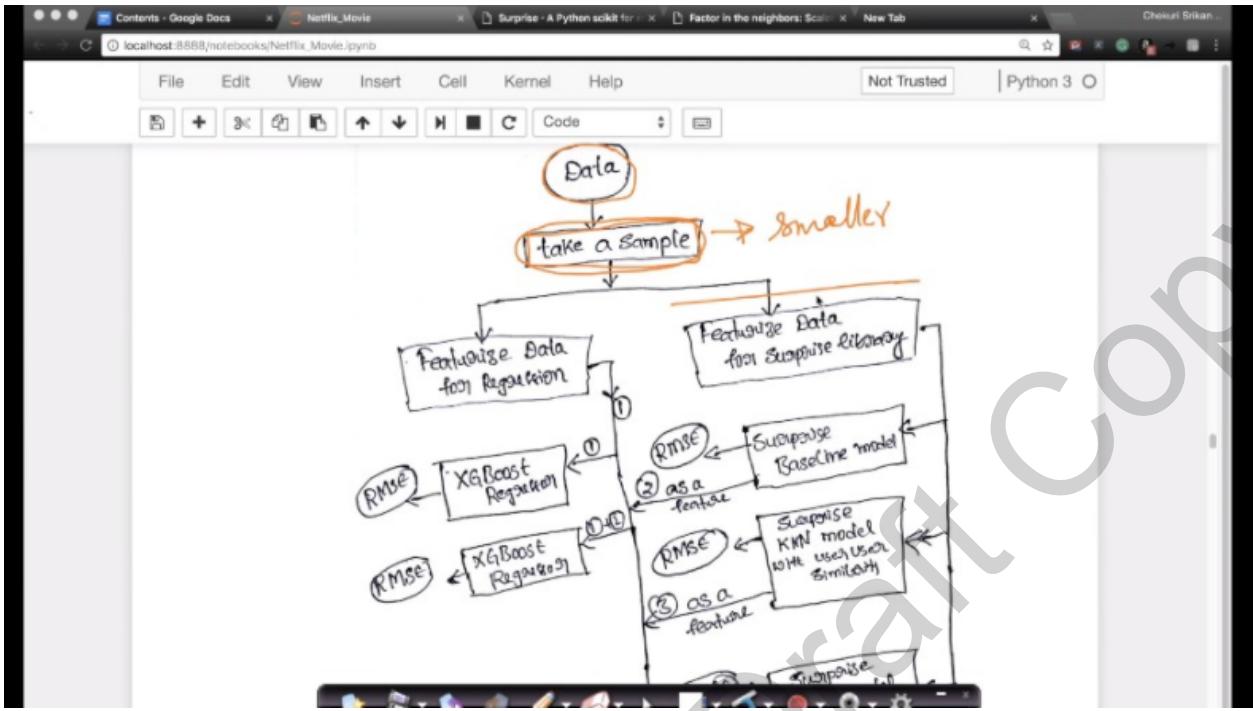
**Getting started, example**

Timestamp : 01:34

Surprise stands for Simple Python Recommendation System Engine.

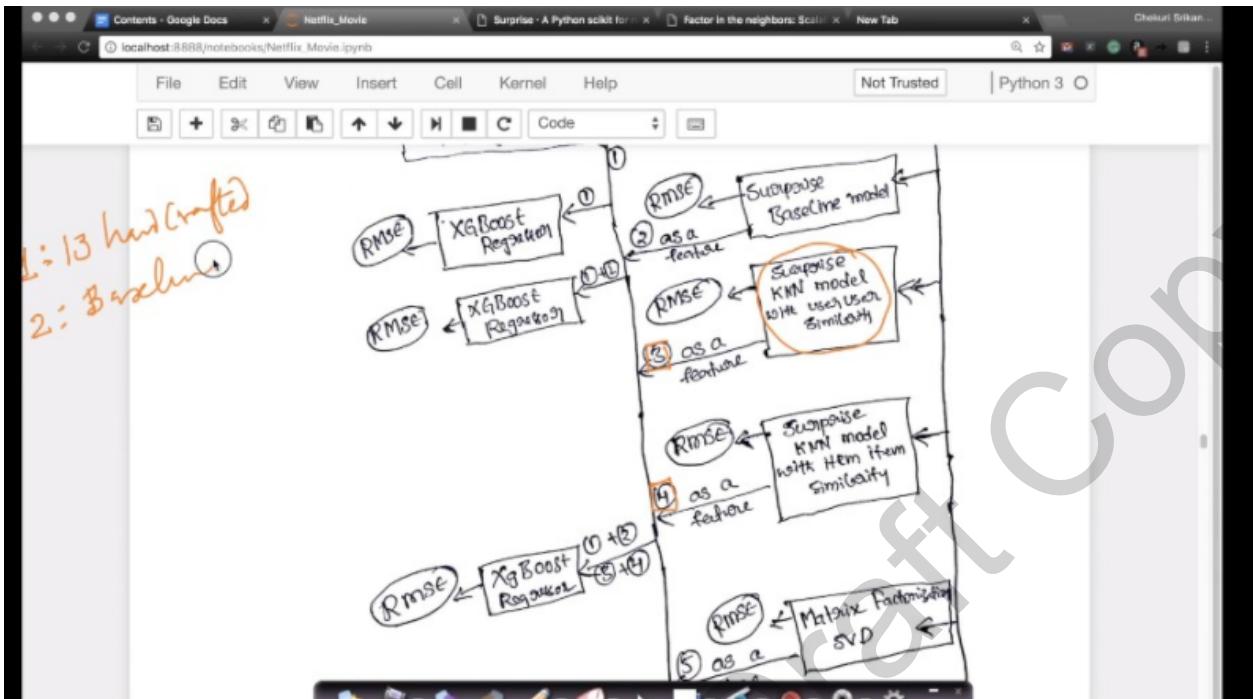
pip install numpy  
pip install scikit-surprise ( For installing the library )

## 7.15 Overview of the modelling strategy.



Timestamp : 00:46

- 1.) Take a subset of samples from the data as working with entire data is computationally infeasible.
- 2.) Come up with some features for regression algorithms.
- 3.) Also, in parallel, featurize the data for surprise baseline model,knn model with user user similarity and with item item similarity.
- 4.) The output predicted by the baseline model is also used for featurization in regression algorithms. This is similar to stacking of models.



Timestamp : 05:01

We also use SVD and SVD++.

## 7.16 Data Sampling.

Dataset is splitted into train and test data.

Train data : 405k \* 17k

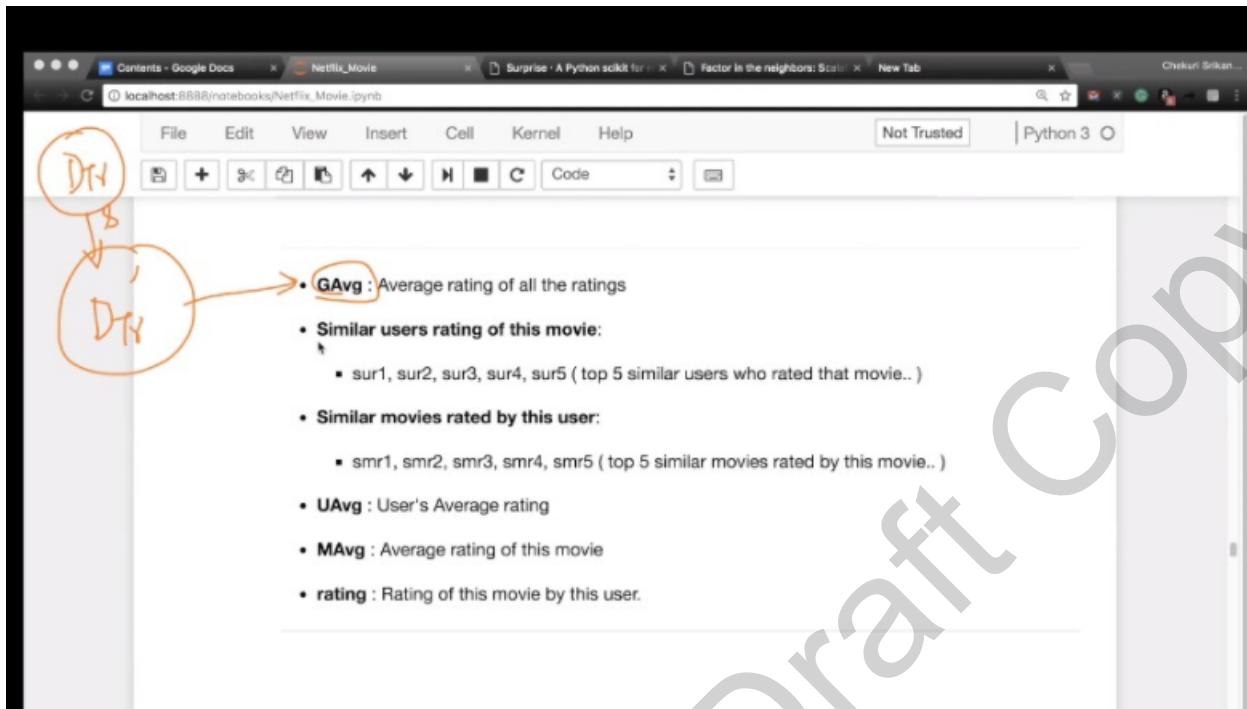
Test data : 348k \* 17k

We take a subset of data from both train and test.

In train, we sampled 10k \* 1k and from test we took 5k\*500.

Let's build the models on the sampled data. We will find the best model out of it. Then, train the best model on the whole data.

## 7.18 Featurizations for regression.



Timestamp : 01:29

The above are self explanatory.

Sur1 ?

Let's pick a user ui. Now , find top k similar users to ui ( Assume k = 5 ).  
Let those sets of users be u1,u2,u3,u4,u5.

Now, check whether u1 has watched mj or not. Similarly for others, perform the same.

Now, sur1 is the rating given by u1 on mj.

If any users have not watched movie mj, we don't consider them.

Similarly perform the same for movies too which is smr1,smr2,etc.

There are multiple ways people handle the Cold Start problem. Those are

1. You take the features of the movies based on its content and then evaluate the similar type of movies of the new user based on 2 to 3 movies he watched.
2. You recommend globally top movies initially to a new user who is a new user.
3. You try to show movies which are recently being popular from the region where your IP address is pointing to.

etc etc..

**As the rest of the video lectures are only about the code sample discussion, we are not providing any notes for it. For any queries regarding the code samples, please feel free to post them in the comments section below the video lecture (or) you can mail us at [mentors.diploma@appliedroots.com](mailto:mentors.diploma@appliedroots.com)**

AppliedRoots (Draft Copy)

AppliedRoots (Draft Copy)

AppliedRoots (Draft Copy)