# Instructions

We would like to implement and compare recommender systems for a video streaming company like Netflix based on the ratings of the viewers. We will assume a star rating system where users express their satisfaction by ticking the corresponding number of stars (between 0 and 5)

We would like to implement and compare the following recommender systems:

- Item to item collaborative filtering
- User to User collaborative filtering
- Global Baseline
- Principal component analysis (PCA using SVD)

The SVD computation is of no interest, consider using a library in your language of choice

# Overview

**Dataset:**

- Data file: movie_titles.csv (MovieID, CustID, Ratings, Date)
- Dataset Name: MovieLens 100K Dataset.
- Dataset source: https://grouplens.org/datasets/movielens/100k/

**Data summary:**

- Unique movies: 4,156.
- Unique users: 363,086.
- Total ratings: 2,000,000.
- Ratings are on a five star (integral) scale from 1 to 5.

## Prepare the data:

- Import the needed libraries (pandas, numpy, matplotlib, etc..)
- Read the CSV file and store it in a pandas dataframe
- Store the pandas dataframe as a picklefile for later use (Python pickle module is used for serializing and de-serializing a Python object structure.)
- Check the data for any NaN or duplicate value.
- Splitting data into Train and Test (80:20)

## Create the Sparse Matrix

- We create USER-ITEM sparse matrix from data frame (one for each, train and test dataframe)

  Shape of Train sparse matrix = (2649430, 13368)

  Shape of Test sparse matrix = (2649430, 13367)

  (shape of sparse matrix depends on highest value of User ID and highest value of Movie ID in each dataframe.)

## Finding Global averages based on the sparse matrix:

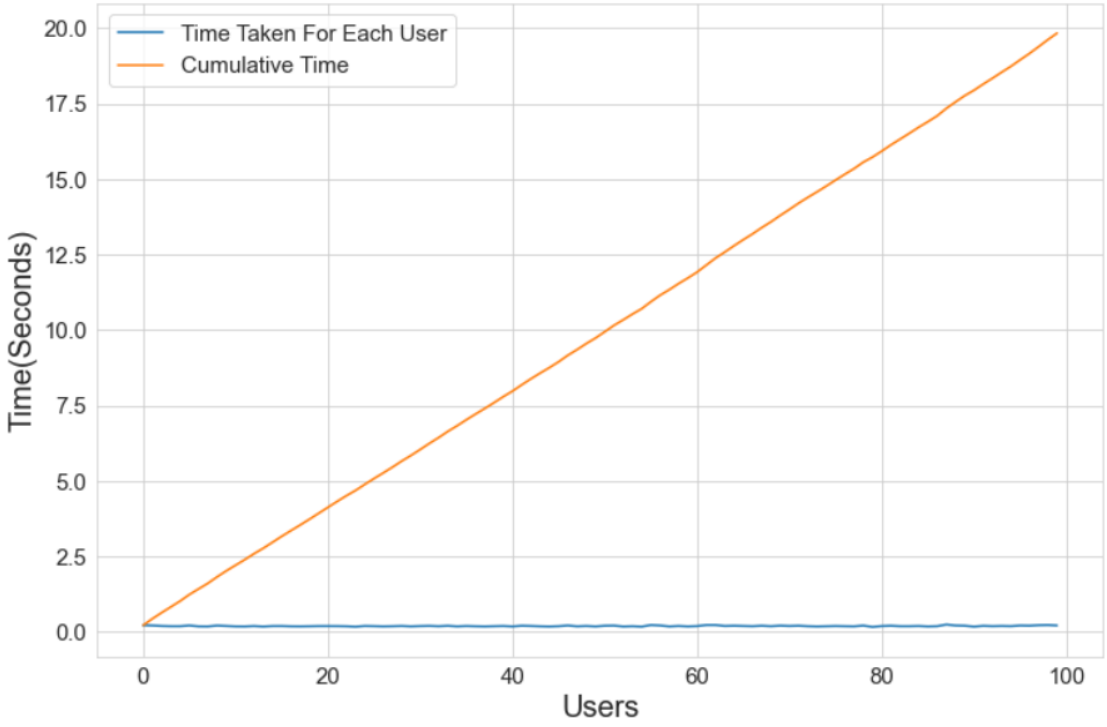
- All movie ratings: 3.5
- Average rating of user 1277134 = 3.2
- Average rating of movie 9211 (Blue Juice, 1995) = 2.3

# Computing Similarity Matrices

**Users-user similarity matrix:**

We first compute the matrix for the top 100 users in our sparse matrix and we got the following result:

```
Time elapsed for 20 users = 0:00:04.024741sec
Time elapsed for 40 users = 0:00:07.887276sec
Time elapsed for 60 users = 0:00:11.816932sec
Time elapsed for 80 users = 0:00:15.820362sec
Time elapsed for 100 users = 0:00:19.933427sec
Average Time taken to compute similarity matrix for 1 user = 0.1982944703102112 seconds
```




We have 295,834 Users in our training data.

Average time taken to compute similarity matrix for one user is 0.198 sec.

For 295834 users:

295834*0.198 == 58,575 sec == 16.27 hours

so for the sake of the time, We will 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.

**Movie-Movie Similarity Matrix**

We compute the similarity based on the cosine distance, then we store the matrix in a .npz file for better performance.

To verify the efficiency of our movie-movie similarity matrix, we will choose a random movie and check the top 10 movies similar to it:

**Top 10 Similar Movies to: South Park: Season 2**

```
In [59]: movie_titles_df.loc[similar_movies_dict[movieID_GR][:10]]
```

Out[59]:

| MovieID | Year_of_Release | Movie_Title |
|---|---|---|
| **10418** | 1997.0 | South Park: Season 1 |
| **11191** | 1999.0 | South Park: Season 3 |
| **13267** | 2002.0 | Gorillaz: Phase One Celebrity Take Down |
| **9785** | 1996.0 | Beavis and Butt-head Do America |
| **12834** | 2001.0 | Family Guy: Vol. 2: Season 3 |
| **11398** | 1980.0 | Airplane! |
| **9900** | 1994.0 | Airheads |
| **9628** | 1983.0 | Star Wars: Episode VI: Return of the Jedi |
| **10598** | 2004.0 | Chappelle's Show: Season 2 |
| **9298** | 1996.0 | From Dusk Till Dawn |

It seems that Movie-Movie similarity matrix is working perfectly.

# Global Average

- We create Sample Sparse Matrix for Train Data
- We create Sample Sparse Matrix for Test Data

From our sampled train data we will check the average of:

- All movies ratings in Train Sample Sparse is 3.48
- Average movie rating for movie 9426 (Platoon, 1986) is 3.93
- Average user rating for user 2447802 is 3.6

**Train data**:

Based on the Train Data, we run a regression model to analyze the ratings based on the models we prepared so far, and we got the top five rows of the result:

| | User_ID | Movie_ID | Global_Average | SUR1 | SUR2 | SUR3 | SUR4 | SUR5 | SMR1 | SMR2 | SMR3 | SMR4 | SMR5 | User_Average | Movie_Average | Rating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 65841 | 9266 | 3.481443 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 5.0 | 4.0 | 3.75 | 3.75 | 3.75 | 3.0 | 3 |
| **1** | 346450 | 9300 | 3.481443 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 4.0 | 2.5 | 2.5 | 2.50 | 2.50 | 2.50 | 1.0 | 1 |
| **2** | 718717 | 9305 | 3.481443 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 4.00 | 4.00 | 4.00 | 4.0 | 4 |
| **3** | 677888 | 9307 | 3.481443 | 5.0 | 5.0 | 5.0 | 5.0 | 5.0 | 2.0 | 3.5 | 3.5 | 3.50 | 3.50 | 3.50 | 5.0 | 5 |
| **4** | 388514 | 9309 | 3.481443 | 3.0 | 4.0 | 3.0 | 3.5 | 3.5 | 5.0 | 5.0 | 3.0 | 4.25 | 4.25 | 4.25 | 3.5 | 4 |

**SUR**: similar users with this User

**SMR**: similar movies with this Movie

**Rating**: Rating given by this User to this Movie

Shape of Train DataFrame = (1913, 16)

(we store the result on a CSV file on disk for better performance)

**Test Data**:

Based on the Test Data, we run a regression model to analyze the ratings based on the models we prepared so far, and we got the top five rows of the result:

| | User_ID | Movie_ID | Global_Average | SUR1 | SUR2 | SUR3 | SUR4 | SUR5 | SMR1 | SMR2 | SMR3 | SMR4 | SMR5 | User_Average | Movie_Average | Rating |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 694228 | 9279 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 4 |
| 1 | 1065039 | 9337 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 4 |
| 2 | 1866094 | 9337 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3 |
| 3 | 1295097 | 9399 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 4 |
| 4 | 2135554 | 9410 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3.481443 | 3 |

**SUR**: similar users with this User

**SMR**: similar movies with this Movie

**Rating**: Rating given by this User to this Movie

Shape of test DataFrame = (318, 16)

(we store the result on a CSV file on disk for better performance)

## Machine learning models to build global baseline and SVD matrices.

To evaluate our models, we will use **RMSE** (Root Mean Square Error)

**RMSE** is the error of each point which is squared. Then mean is calculated. Finally root of that mean is taken as final value.

where **At** is the actual value and **Ft** is the forecast value.

The difference between **At** and **Ft** is divided by the actual value **At** again. The absolute value in this calculation is summed for every forecasted point in time and divided by the number of fitted points n. Multiplying by 100% makes it a percentage error.

# Baseline Model

## Predicted Rating

$$\hat{r}_{ui} = \mu + b_u + b_i$$

- $\mu$: Average Global Ratings in training data
- $b_u$: User-Bias
- $b_i$: Item-Bias

## Optimization Function

$$\sum_{r_u i \in R_{Train}} (r_{ui} - (\mu + b_u + b_i))^2 + \lambda \left(b_u^2 + b_i^2\right) . [minimize\ b_u, b_i]$$

We ran our baseline model on the train and test data, and we got the following result:

```
-------------------------------------------------
TRAIN DATA
RMSE = 0.7388687105450215
-------------------------------------------------
TEST DATA
RMSE = 1.2011807573799085
-------------------------------------------------
Time Taken = 0:00:00.234866
```

## Matrix Factorization SVD

**Prediction $\hat{r}_{ui}$ is set as:**

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

- $q_i$ - Representation of item(movie) in latent factor space
- $p_u$ - Representation of user in new latent factor space

**If user u is unknown, then the bias $b_u$ and the factors $p_u$ are assumed to be zero. The same applies for item i with $b_i$ and $q_i$.**

**Optimization Problem**

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left( b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 \right) \left[ minimize\ b_u, b_i, q_i, p_u \right]$$

We ran our SVD model on the train and test data, and we got the following result:

```
------------------------------------------------
TRAIN DATA
RMSE = 0.8363026077600323
------------------------------------------------
TEST DATA
RMSE = 1.2006928307366789
------------------------------------------------
Time Taken = 0:00:00.822533
```

And here is the comparison table:

| | Model | Train RMSE | Test RMSE |
|---|---|---|---|
| **0** | BaselineOnly | 0.738869 | 1.201181 |
| **1** | SVD | 0.836303 | 1.200693 |