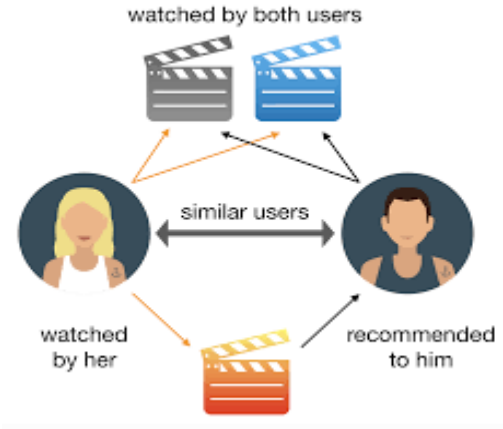
Movie Recommender Systems

A movie recommender system predicts the likelihood that a user might prefer to watch a movie not seen before based on past behaviour of same user or other similar users and user's preferences.

There are two types of recommender systems:

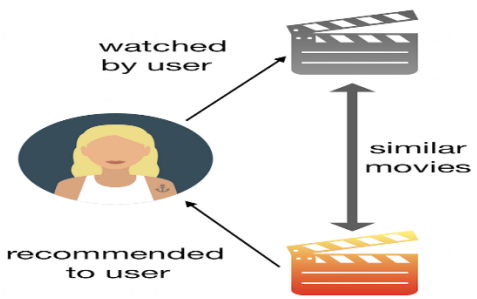
User based: It suggests new movies based on both users' previously watched movies and experiences of other similar users. For example, if two users have watched few of the same set of movies, the one can be suggested unseen movie based on watchlist of other user. Basically, they can be considered neighbours in such case.



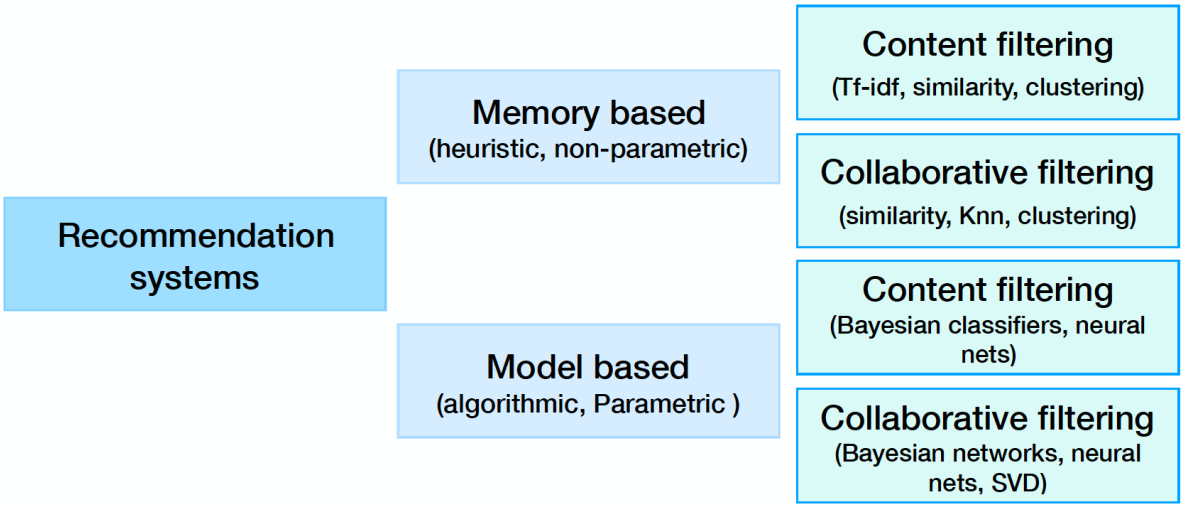
Limitations:

1. As we are comparing one use with all other users, it is computationally burdensome.
2. Also, individual user can change his/her preferences and start behaving differently than its neighbour which is not considered in user-based recommender system.

Item based: In this system, instead of finding relationship between users, used items like movies are grouped together or compared with each other. In such cases, even if the habit of user's changes, this doesn't impact recommendations. This is also computationally less challenging as the comparison is made between movies than users.



Different types of recommendation Models



As mentioned in above diagram, the recommendation systems can be broadly classified into parametric and non-parametric models. Each of these can have content based filtering or Collaborative filtering methods to provide recommendation.

Parametric models: The advantage is that they are simpler to explain and understand.

Non-parametric Models: These are based on matrix factorization to predict user's rating for all unseen movies and then filter top movies. They are better in handling data sparsity issues.

ABOUT MOVIELENS DATASET

Source:

Kaggle: https://www.kaggle.com/grouplens/movielens-20m-dataset

Context

The datasets describe ratings and free-text tagging activities from MovieLens, a movie recommendation service. It contains 20000263 ratings and 465564 tag applications across 27278 movies. These data were created by 138493 users between January 09, 1995 and March 31, 2015. This dataset was generated on October 17, 2016 by GroupLens.

Data Gathering process

Users were selected at random for inclusion. All selected users had rated at least 20 movies.

Content of data

The data are contained in six files:

1. tag.csv that contains tags applied to movies by users:
2. rating.csv that contains ratings of movies by users:
3. movie.csv that contains movie information:
4. link.csv that contains identifiers(IMDB,TMDB) that can be used to link to other sources:
5. genome_scores.csv that contains movie-tag(provided by users) relevance data:
6. genome_tags.csv that contains tag descriptions

Acknowledgements

To acknowledge use of the dataset in publications, please cite the following paper:

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

Limitations in data:

- No user demographic information is provided which can help in suggesting age specific movies.

Approach taken for recommender system

Step 01: Importing relevant packages

```
#import required packages
import pandas as pd
import statistics as st
import scipy
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import seaborn as sns
import datetime as dt
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
```

Step 02: Loading Datasets and check for basic summaries and missing values

```
#####import scores dataset from Movielens data
data_scores = pd.read_csv('C:\\Users\\surbhi36152\\Downloads\\archive\\genome_scores.csv')

###get summaries
data_scores.head(10) ###what is relevance?
scores_summary=data_scores.describe()
len(data_scores['movieId'].unique().tolist())

###search for missing values
data_scores.isnull().any()

#####import rating dataset from Movielens data
data_rating = pd.read_csv('C:\\Users\\surbhi36152\\Downloads\\archive\\rating.csv')
data_rating.head(10)
###get summaries
rating_summary=data_rating.describe()#not much poorly rated movie, movies is almost same as no.of users
len(data_rating['movieId'].unique().tolist())

###search for missing values
data_rating.isnull().any()
```

Step 03: Merge datasets on similar columns to bring information together about movies. Also recommend common tags like imdb top 250 movies and Oscar nominated movies

```
##combine tagged datasets and find relevant tags like top 250 imdb and oscar
data_tag_name=data_scores.merge(data_genome_tags,on=['tagId'],how='inner') ##not useful (only imdbID)
data_tag_name[(data_tag_name['tag']=='imdb top 500') & (data_tag_name['relevance']>0.9)].sort_values(by='relevance',ascending=False)

data_tag_name[(data_tag_name['tag']=='oscar') & (data_tag_name['relevance']>0.9)].sort_values(by='relevance',ascending=False)

most_tagged= pd.DataFrame(data_tag_name.groupby('tag').size().sort_values(ascending=False)).reset_index()

####merge imdbid with movie title
data_movie_link_new=data_movie.merge(data_link,on=['movieId']) ##not useful (only imdbID)

###merge movie with user ratings
data_rating_genre = data_rating.merge(data_movie,on='movieId', how='left')
data_rating_genre.columns

###check for nulls
data_rating_genre.isnull().any()
```

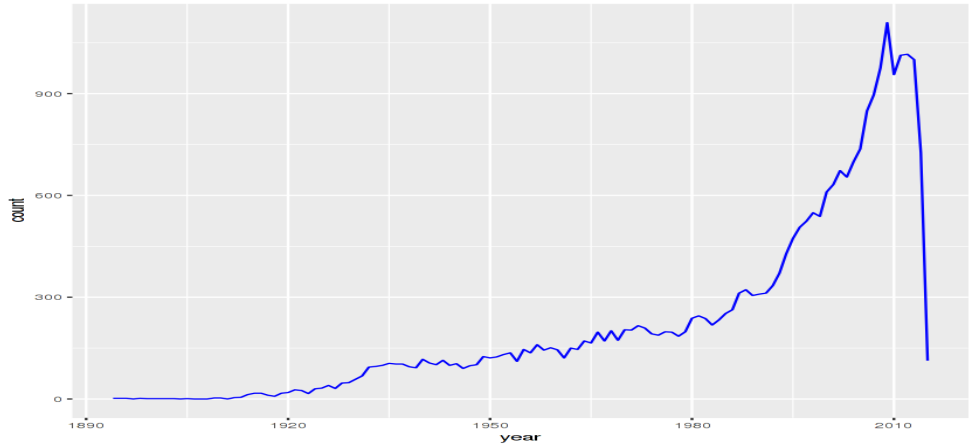
Output:

store - DataFrame

Index	gId	relevance	tag	title	genres
908575		0.995	imdb top 250	It Happened One Night (1…	Comedy|Romance
2713375		0.9945	imdb top 250	Killing, The (1956)	Crime|Film-Noir
1897831		0.99375	imdb top 250	On the Waterfront (…	Crime|Drama
1192831		0.99175	imdb top 250	One Flew Over the Cuckoo's…	Drama
3448831		0.99175	imdb top 250	Hustler, The (1961)	Drama
928879		0.99175	imdb top 250	Citizen Kane (1941)	Drama|Mystery
1257127		0.99125	imdb top 250	Treasure of the Sierra M…	Action|Adventure|Dr…
932263		0.99025	imdb top 250	All About Eve (1950)	Drama
916471		0.9895	imdb top 250	Casablanca (1942)	Drama|Romance
4394095		0.9895	imdb top 250	Man Who Shot Liberty Vala…	Crime|Drama|Western
1230055		0.98925	imdb top 250	Raging Bull (1980)	Drama
3414991		0.989	imdb top 250	Double Indemnity (1…	Crime|Drama|Film-Noir
1177039		0.98875	imdb top 250	Paths of Glory (1957)	Drama|War
55897		0.98825	imdb top 250	Usual	Crime|

Step 04: Data feature engineering to extract new value from existing columns.For example, extract year of movie from title to recommend recent released movies

```
##get year out of title  ###very slow, can make faster
data_rating_genre['year'] =data_rating_genre['title'].str.extract('.*\((.*)\).*',expand = False)
data_rating_genre.iloc[:,3:].head(5)

########recent released movies
```

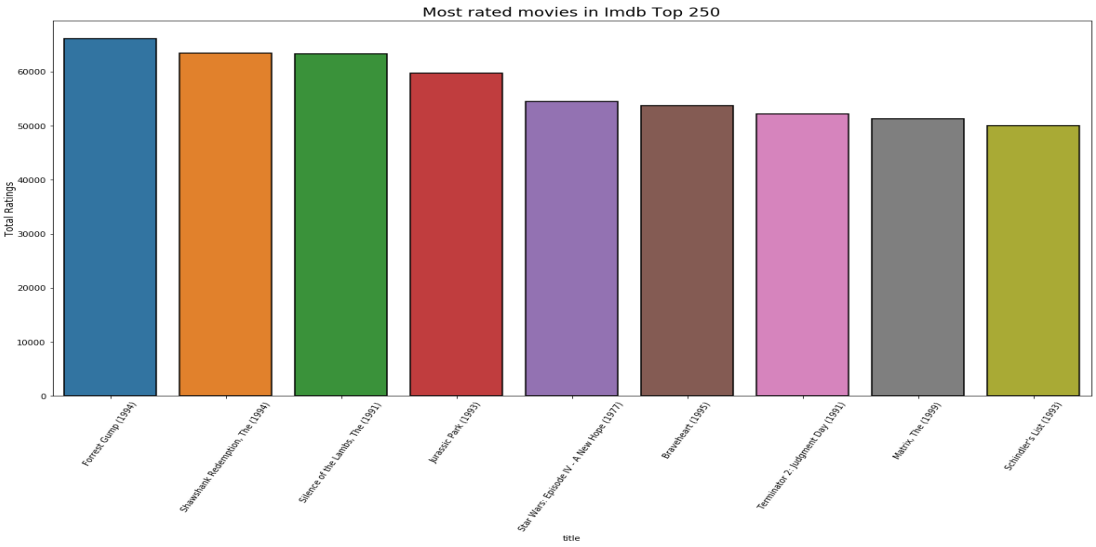


Step 05: Basic EDA to get most viewed movies and highly rated movies(after removing movies with less than 500 ratings)

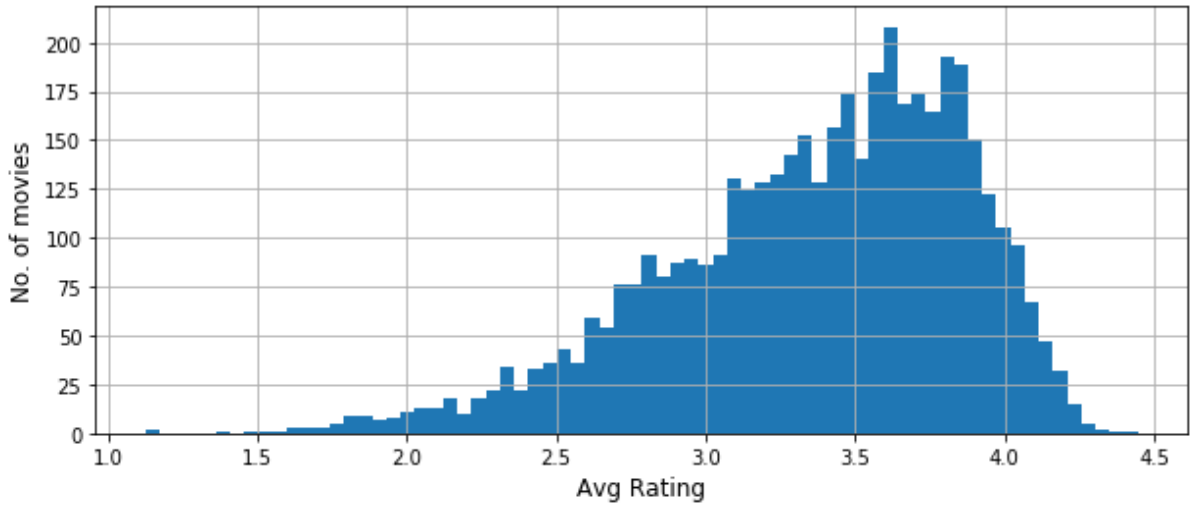
```
##########most viewed movies#############s
most_rated = pd.DataFrame(data_rating_genre.groupby('title').size().sort_values(ascending=False)).reset_index()
most_rated.columns=['title','Total Ratings']
most_rated.sum()


 ###get top rated movies with minimum of 500 ratings
relevant_movies=most_rated[most_rated['Total Ratings']>500]['title']
Average_ratings= pd.DataFrame(data_rating_genre[data_rating_genre['title'].isin(list(relevant_movies))].groupby('title')['rating'].mean().sort_values(ascending=
Average_ratings.head(10)
Average_ratings.columns=['title','Average_ratings']

title_avg_rating_no_users = most_rated.merge(Average_ratings,on='title', how='inner')
weight_avg=pd.DataFrame(title_avg_rating_no_users.sort_values(by=['Total Ratings'],ascending=[False])).reset_index()
```

Output:

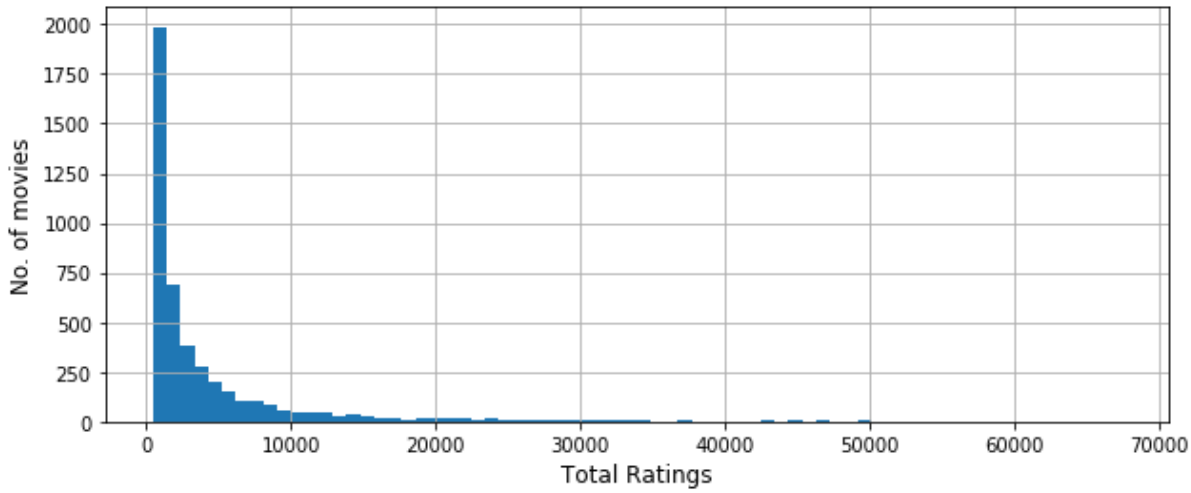
IMDB top 250- This gives list of top 10 most rated movies



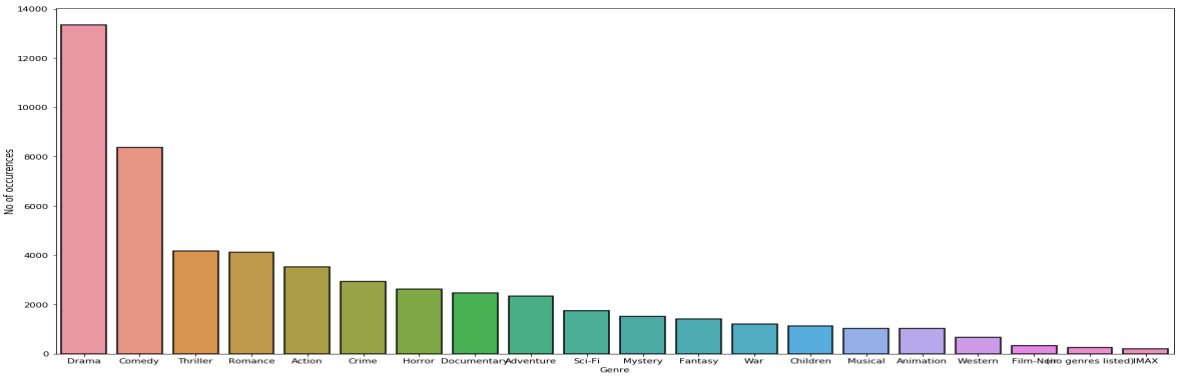
Distribution of Average Ratings – Giving idea about average rating as ~3.5 for most of the movies



Distribution of total ratings for movies- We can see most of the movies are rated by less than 10,000 users**.**



Most popular genres:




```
########Popularity of genre ######################

movies['(no genres listed)'].sum()
movies.columns
movies[movies['(no genres listed)']==1].iloc[:,1:4].head(1)

genre_freq=movies.iloc[:,3:].sum(axis=0).sort_values(ascending=False)
genre_freq=pd.DataFrame(genre_freq).reset_index()
genre_freq.columns=['Genre','No of occurences']


plt.figure(figsize = (20,10))
plt.ylabel("No. of occurences", fontsize = 12, labelpad = 0)
ax = sns.barplot(x = 'Genre', y = 'No of occurences', data = genre_freq,  linewidth = 1.5, edgecolor = 'black')
plt.show()
```

Output:

Step 06:

1) First Recommendation is simple and based on ratings(item) correlation

This method creates a matrix for all movies with ratings as values and finds correlation of all movies with movie already seen by user e.g. Toy Story (1995)

```
##transform to get movies in columns
movie_rater = data_rating_genre_filtered.pivot_table(index='userId',columns='title',values='rating')
movie_rater.head()
movie_rater=movie_rater.fillna(0)

###get correlation from corrwith(pearson) takes times
correlations = movie_rater.corrwith(movie_rater['Toy Story (1995)'])
correlations.head()

recommendation = pd.DataFrame(correlations,columns=['Correlation']).reset_index()
recommendation.dropna(inplace=True)
recommendation = recommendation.join(most_rated['Total Ratings'])

recommendation.head()

recc = recommendation[recommendation['Total Ratings']>500].sort_values('Correlation',ascending=False).
recc = recc.merge(data_movie,on='title', how='left').sort_values(by=['Correlation', 'Total Ratings'],a
#recc.sort_values(by=['Correlation', 'Total Ratings'],ascending=[False, False]).head(10)
```

The Output comes out as similar rated other movies:

	index	title	Correlation	Total Ratings	mo
	1012	Back to the Future (1985)	0.326903	4950	1270
	422	Aladdin (1992)	0.324874	10697	588
	422	Aladdin (1992)	0.324874	10697	114240
	2030	Bug's Life, A (1998)	0.321581	2006	2355
	4374	Finding Nemo (2003)	0.299268	531	6377

Drawbacks: This method doesn't integrate other items and similar user behaviour.

2) Another recommendation is again item based(genres) based on cosine similarity

This method first does one hot encoding of genres in data and then calculates cosine distance between each pair of genres and recommends movies like genre seen previously by user. Here example is taken of k=6 (7^{th} movie) i.e. Sabrina (genre comedy/romance)

```
####genre based on genre movie neighbourhood based on cosine distance##########

movies = data_movie.join(data_movie.genres.str.get_dummies("|"))

# compute the cosine similarity
cos_sim = cosine_similarity(movies.iloc[:,3:])


sabrina_top5 = np.argsort(cos_sim [6])[-5:][::-1]
movies[movies.index.isin(sabrina_top5)]['title']

##limitations : doesn't cater to user tailored suggestions by incorporating ratings
```

The Output recommends similar genre movies

```
[52]: movies[movies.index.isin(sabrina_top5)]['title
t[52]:
396                      Kate & Leopold (2001)
183                           Wimbledon (2004)
186                      First Daughter (2004)
5607                  My Dear Secretary (1948)
9945    Four more years (Fyra år till) (2010)
ame: title, dtype: object
```

3) Another recommendation is similar user based collaborative filtering based on ratings.

```
avg_rating = data_rating['rating'].mean() # calculate mean rating

data_rating_filtered= data_rating.iloc[:1000000,:]
preferance_matrix = data_rating_filtered[['userId', 'movieId', 'rating']].pivot(index='userId', columr

preferance_matrix = preferance_matrix - avg_rating # subtract avg rating i.e. adjust ratings

item_avg_rating = preferance_matrix.mean(axis=0)  ###get column avg
preferance_matrix = preferance_matrix - item_avg_rating # item avg made adjustment

user_avg_rating = preferance_matrix.mean(axis=1)  ##get row avg
preferance_matrix = preferance_matrix - user_avg_rating# item avg made adjustment

mat_avg=preferance_matrix.fillna(0) + user_avg_rating + item_avg_rating + avg_rating

mat_avg = mat_avg.values

##for user 1
np.nansum((mat_avg - mat_avg[150,:])**2,axis=1)[1:].argmin() # returns 99
# check it:
np.nansum(mat_avg[149] - mat_avg[150]) # returns 0.0

np.where(~np.isnan(mat_avg[149]) & np.isnan(mat_avg[150]) == True)
```

The Output recommends 11th movie which is yet not seen by user and liked by another user.

```
[71]: np.where(~np.isnan(mat_avg[4]) & np.isnan(mat_avg[700]) == True)
t[71]:
rray([   1,   10,   16,   59,   61,  103,  139,  140,  223,  234,  259,
        281,  315,  317,  349,  363,  366,  367,  369,  375,  376,  379,
        439,  453,  456,  474,  479,  490,  499,  507,  514,  528,  530,
        586,  587,  588,  589,  592,  593,  594,  607,  630,  647,  670,
        707,  719,  735,  779,  787,  831, 1034, 1035, 1041, 1072, 1078,
       1079, 1096, 1135, 1195, 1197, 1209, 1290, 1392], dtype=int64),)

[72]: mat_avg[4][[1,   10,   16]]
t[72]: array([-0.86882537,  0.627803  , -1.41592753])
```

Drawbacks: It doesn't consider sudden change in user behaviour.

Future Scope

1. Try parametric models by dividing data in train and test as data is more than 20Mn, which is more robust for training model e.g. SVM, other neural nets, matrix factorization
2. We can further explore usage of timestamps in data by exploring more on temporal patterns on user ratings and genres views.