MSCI 623: Final Project

SPRING 2020

Final Project Report

Anime Recommender System

(Collaborative Filtering & Content Based Filtering)

Akshita Singh (Student ID: 20813645)
Dhaval Thakur (Student ID: 20837892)

Abstract

Gone are those days when we had to wait for 24hours to watch the next episode of our favorite series. Platforms like Netflix, Hulu, Amazon prime etc. are nowadays releasing movies and entire TV shows at a go, which we can watch anytime, anywhere. Moreover, when we watch a particular movie, for example Star Wars, it suggests movies that we may like, say Star Trek or Pacific Rim. These movie suggestions are often similar to the one's we have watched with respect to either the genre or the ratings and also based on what other users, with similar interests, might have watched. Moreover, on large databases, users may not know what to search for and they could easily miss movies or series that they could have liked. Recommender systems aid users in exploring these movies based on movie-movie similarity as well as user to user preferences. It's like having a friend with amazing taste, which 90% of the times doesn't err while making suggestions because it gives priority to our viewing preferences, our likes and dislikes. As per Google developers' recommendation systems course, 40% of app installs on Google Play come from recommendations and 60% of watch time on YouTube comes from recommendations. This is the power of **recommender systems** and we seek to harness it for our project.

Since a lot of research and work exists on movies, songs etc., we would be tapping into an uncharted territory, Japanese anime, where there is a dire need for good recommender systems that would help the sellers and consumers alike. At the inception stage of our project, we will start by performing exploratory data analysis on our business problem to understand various features of the dataset which impacts the decision making and decipher the shortcomings of the data such as missing values, outdated entries etc. which have hampered the data analysis so far. Post data pre-processing we will apply two types of recommender systems, First one being unsupervised learning based method called collaborative filtering (both item-item and user-user based collaborative filtering) and is computed using cosine similarities and the second one is a supervised learning based method called content-based filtering, and is computed using K Nearest Neighbors. For content-based filtering, we will experiment with different subsets of features and observe the results. For collaborative filtering, we will be computing the Root Mean Square Error and a new, less explored metric called Coverage, to evaluate the system. Furthermore, we will compare which of these two methods are more effective for making the right recommendations and discuss the problematic areas we encountered during our research and suggest possible areas of improvement.

1. Introduction

1.1 Business problem: description, approach and beneficiaries

While the recommender system for movies and TV series are widely in use (thanks to Netflix! which has the best recommender system till date), there's a wide market, predominant in Asian sub-continent, that is **widely untapped**. This is the Japanese anime market. Anime is a style of Japanese film and television animation, typically aimed at adults as well as children. The term anime is derived from the English word animation, and in Japan it's used to refer to all forms animation styles characterized by colorful graphics, vibrant characters and fantastical themes. Netflix has a limited number of anime shows on its platform. The Japanese anime shows are more popular on local sites, which either do not have a good recommender system or have none at all. They suggest anime based on simple algorithms such as 'most popular' which is based on the highest number of views or algorithms like 'newly added or newly launched'. Through this project, we aim to implement a **recommender system for Japanese Anime** (which includes Anime movies, series and Original Video Animations), the data for which has been collected from entirely anime-based sites which have more options than our currently popular platforms such as Netflix and Amazon Prime. This will benefit the anime viewers in discovering the less popular hidden gems as per their preference and simultaneously help the anime creators and owners in expanding the viewership and fanbase.

1.2 Why can't we use simple SQL queries here?

The recommendations for anime can't be solved simply by retrieving data from the datasets using Structured Query Languages because within a single genre, the anime can vary greatly from each other and furthermore, may highly depend upon user's similarities or similarities amongst the anime (w.r.t storylines or characters) which requires a more complex algorithm. For example, it is easy to tell which is the highest rated anime, we can run a simple query and get the max rating. However, to retrieve anime that a user may like based on their past views is not a straight-forward problem. We would need to predict the ratings based on genres, length, characters, how other people with similar tastes have responded to that anime etc. We aim to solve this business problem with an unsupervised learning algorithm called '**Collaborative filtering'** and also with a supervised learning algorithm, '**Content based filtering via KNN**'.

1.3 Collaborative filtering

This method of unsupervised learning uses the prior interaction information between the user and the anime. It creates a user-anime interaction matrix where all the anime that are watched by the user are rated. It is a sparse matrix (matrix with mostly empty cells) as the user might have only watched a few anime from the entire database. **Important thing to note here is that no other information about the user and the anime is needed.** E.g. we don't need to know if the user was a child or an adult or if the anime was from a particular genre etc. All we need to know is what are the ratings a user has given to all the anime they have watched. The following table shows how the user-anime table is structured.

User/Anime	**Dragon Ball Z**	**Fairy tail**	**Bleach**	**Inuyasha**	....
Akshita		9		8	....
Dhaval	10			7	....
Chandler	9	8	5		....
Joey		9	4		....
. . .	. . .	. . .	. . .	. . .	. . .

From this matrix, it finds a smaller set of users with tastes similar to a particular user. This can be done using various methods, we will be using cosine similarity for our project.

Collaborative filtering is further bifurcated into two types viz user-item collaborative filtering and item-item collaborative filtering.

1.3.1 User-user collaborative filtering

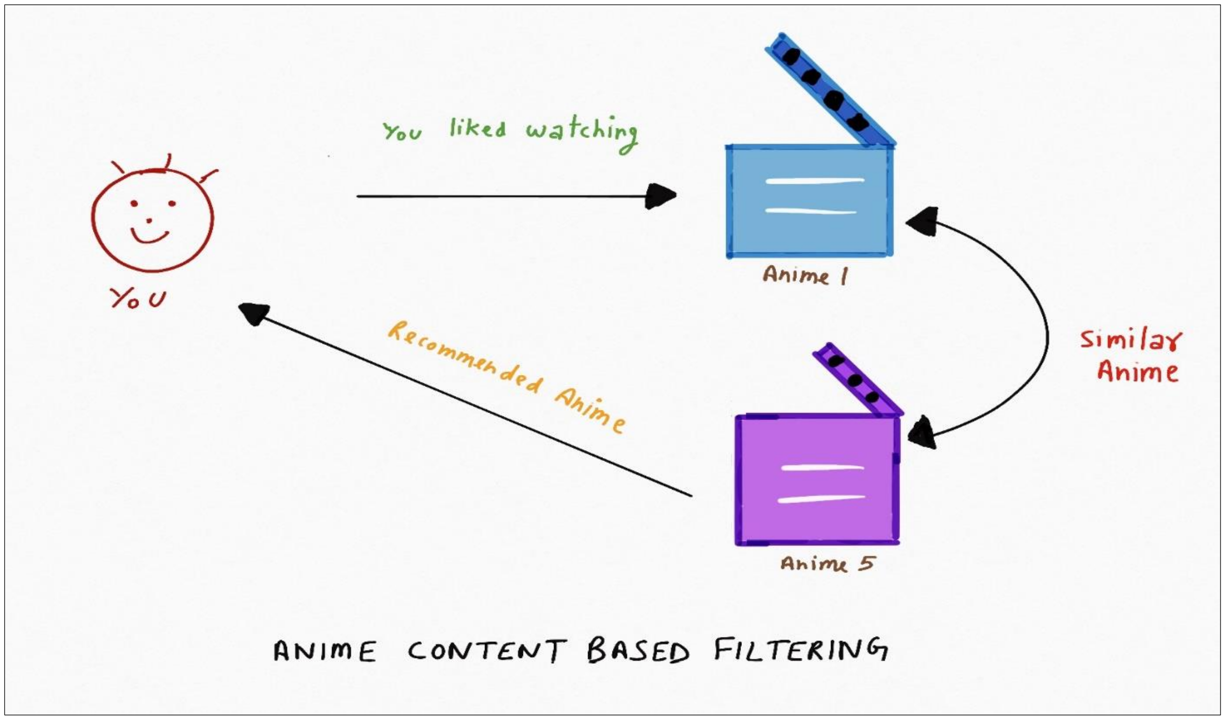
Suppose in the above table, we want to find out how Joey will react to Dragon Ball Z. We will take all the interactions that Joey has, the entire row against Joey, and compare it with the other users who have similar interactions, i.e., except for the unrated anime that Joey hasn't watched, we will look at all the other anime that Joey has seen and compare it with other users who have the most no. of similar viewed anime as Joey. For example, Chandler and Joey have both seen Fairy tail and Bleach and have more in common than Joey and Dhaval. There might be more such similar users to Joey and based on the most similar user we will predict the rating that Joey will give to Dragon Ball Z. If the predicted rating is high, say 7 and above, it will be a good idea to suggest Joey to watch this anime. **Important thing to note here is that we want to pick a user who has rated many anime which are similar to our targeted user** and not a user who only has 1 common anime and has rated it exactly the same as our targeted user. E.g. Joey and Akshita have rated Fairy Tail quite high and exactly the same but in reality, Chandler has more in common with Joey when it comes to preferences even if Chandler has given a rating of 1 less to Fairy Tail than what Akshita has.

1.3.2 Item-Item collaborative filtering

Here, we aim to find a list of anime similar to the one user has already given a positive or high rating to. The two anime will be considered similar if most of the viewers have given similar ratings to them. Similarity between two anime is calculated by taking the ratings of the users who have rated both the anime by calculating their cosine similarity. Once we have the similarity between the anime, we predict the rating by computing a weighted average of the target user's ratings on the similar anime.

Item-Item collaborative is usually preferred over user-user collaborative filtering. This is owing to the fact that a user will not be able to rate most of the anime when the dataset is too large. They would have only seen a few anime and hence would end up rating a very small chunk of the dataset. Therefore, on an average the number of ratings an item will receive in dataset will be more than the number of ratings a user will give.

1.4 Content based filtering



Content based filtering is a supervised learning technique where we make recommendations based on the features of user and/or anime. This method can be used in two ways, one is to use it as a classification problem where we will predict if the user will *like* an anime or *not* and the second one is a regression method where we will predict the rating that will be given to an anime by the user. Then based on the like or rating we can recommend the anime to the user.

Here, if we use the features of anime like genre, episodes, length etc., we call this approach as user-centered. If we use the features of user like age, gender etc., we call it item-centered approach. It is imperative to keep in mind that most of the time users might be reluctant in providing so much information. Whereas, the features of an item are readily and mostly available by the owners during releases. For this reason, *we will be using item-based features* such as the genre of the anime, number of episodes, anime type (TV, movie, OVA) etc. and make recommendations based on user preferences for item (anime) features.

For example, we may define the genre of an anime as:

x= (romance, action, comedy)

Then, we'll apply supervised learning, which is KNN (K Nearest Neighbors) to learn the genre of an anime. For example, the genre of a comedy anime can be calculated as:

g_c= (1,0,0)

Then we can learn how a person rates an anime based on the type of genre. For example, a person might give high rating to action anime low for comedy anime. The preference u_i for this person will be:

xi= (0,9,9)

Content-based filtering builds a model to predict rating or recommendation r given u_i of a person and g_c for an anime:

r=wTg

1.4.1 Why is KNN suitable for our use-case?

While performing exploratory data analysis, we observed that most of the variables in our dataset were numeric (rating, members count, episodes etc.) and hence, it was apt to perform the Nearest Neighbor Algorithm. **KNN** is a **non-parametric, lazy** (it does not learn and make a model, it just stores the data until needed) learning method which does not make any assumptions on the underlying data distribution and relies only on **items' feature similarity** (essence of content-based filtering). To make an inference about an anime, KNN will calculate the "distance" between the target anime and every other anime in its dataset and will rank this distance to return the top K nearest neighboring anime as the most similar anime recommendations. We have specified **K =5** due to the following reasons:

a) If K is small, then results might not be reliable because noise will have a higher influence on the result and if K is large, then there will be a lot of processing which may adversely impact the performance of the algorithm.
b) K should be an odd number so that there are no ties in the voting.

1.5 Explanatory and response variables

For **collaborative filtering** (both user-user and item-item) we will be using only 3 explanatory variables which are **user ID, anime name, anime ID and ratings**. User ID is a numeric ID unique to each user, anime name is a categorical variable, anime ID is a numeric ID unique to each anime and ratings are floating-point numbers ranging from 0-10, 0 being the lowest. 0 doesn't mean that the user has not rated the anime. For unrated anime we have -1 which we are later replacing with NaN or NULL to do our computations for collaborative filtering. This is because if we replace it by 0, the algorithm will take it that the user has seen the anime, which is False and if we keep it -1 our rating average will consider -1 too while calculating the average which will lead to a huge bias and hamper our analysis.

For **content-based filtering** we will be using the features of items, in our case anime, as discussed in the above sections. We will initially test our algorithm with all features and then remove subsets one by one (remove one feature and test, then remove a subset of two features and test and so on).
The features or explanatory variables in this case are **genre, type, rating, episodes and members**. Genre is a categorical variable (comedy, romance etc.), type is a categorical variable (TV, Movie, OVA etc.), rating is a floating-point number ranging from 0-10 as stated above, episodes are integers starting from 1 which tells us the number of episodes in the anime and finally members are the number of people who have joined a community or discussion forum for the particular anime, this integer variable can range anywhere from 0 to thousands or millions depending on the enrollment.

The dataset and variables are explained in detail later, in the section "Datasets and features".

2. Related work and Literature review

Recommendations can be personalized or non-personalized. In non-personalized type, recommendation of item to a user is based on the number of times that item has been visited in the past by other users. However, in personalized type, the main objective is to recommend the best item to a user based on their individual preferences. Although, in many domains, recommender systems have gained significant improvements and provided better services for users, they still require further research. This research is essential to improve accuracy in cases such where we want to incorporate new users and new items into the system, especially in Japanese animation sector. Currently, these scenarios are being handled more on hit-and-trail or random basis.

Based on our research, there hasn't been much work on anime and all the existing work is done on data which is at least 4 years old. Each year, more than 100 anime are released, which means the existing work is quite outmoded. This further leads to existing data having lots of missing values. E.g. for many anime it was mentioned 0 episodes whereas IMDb told us otherwise. This is because the list has not been updated. Many anime which were only 10 episodes long back then have gone up to 60+ episodes and the ratings have varied too. We also found out that pre-processing of the data was done on many assumptions, like taking the median for number of episodes to fill out missing episode lists and ratings. Few anime have 1000s of episodes, and if we take the median of all episodes and assign a much smaller value to the concerned anime, it will create bias, leading to bad recommendations.

Lastly, there have been a few ventures around content-based recommenders but collaborative filtering is still relatively newer in terms of exploration. Moreover, there is a lot of work done on the exploratory data analysis on the anime dataset, whereas very little work has been done on recommendations and evaluation. We aim to tackle the above problems and build a recommender system on updated and improved data with varied methods which incorporate both anime features and user preferences respectively.

3. Datasets and features

Our data is conceived mainly from the site Myanimelist.net, a huge database of anime with a vibrant community who rate and review the anime with precise ratings. Myanimelist.net launched a dataset (via their own API) on Kaggle 4 years ago. Each user is able to add anime to their completed list and give it a rating and this data set is a compilation of those ratings. There are two datasets, **anime.csv** (used for content-based filtering) with anime related data and **rating.csv** (used for collaborative filtering by merging it with anime.csv dataset) containing user preference related data.
We have furthermore updated the datasets using the latest information from this site.
The datasets contain 12199 unique anime (distinct anime_id) with 12324 rows and 7 columns and the rating.csv has 73,516 unique users.

For collaborative filtering, we merged the anime.csv and rating.csv data to get all the user_id, anime_id and rating combinations. We limited the users to 25,000 due to limited computing capacity as Google Colab was running out of memory when with higher number of users.

Below is the list of features along with their descriptions for both the datasets:

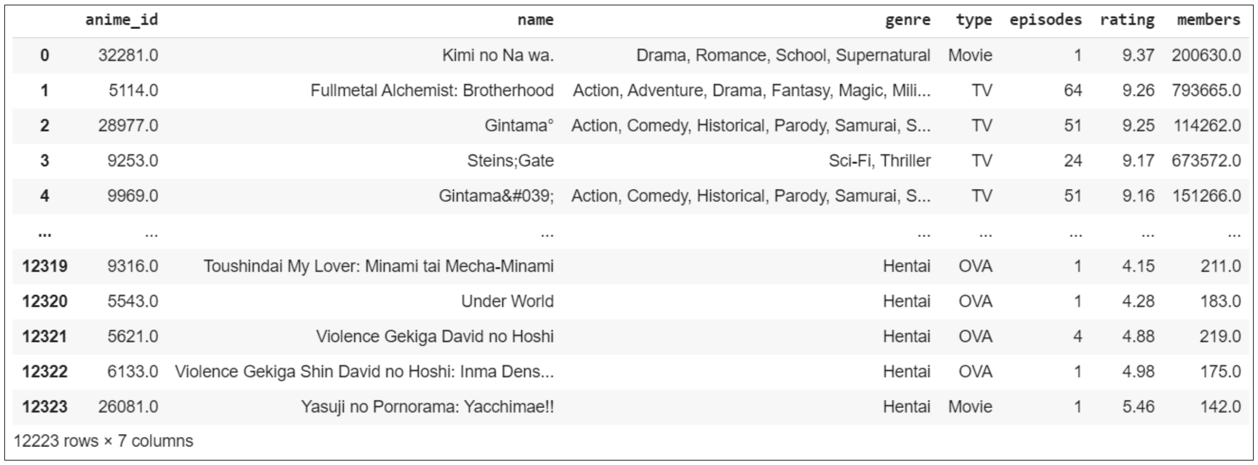
Feature name	**Description**
anime_id	myanimelist.net's unique id identifying an anime
Name	full name of anime
Genre	comma separated list of genres for this anime
Type	movie, music, TV, OVA, ONA etc.
Episodes	how many episodes in this show (1 if movie or OVA)
Rating	average rating out of 10 for this anime.
Members	number of community members that are in this anime's 'group'
User_id (rating.csv)	myanimelist.net's unique id identifying a user

3.1 Data-Preprocessing

1) Handling unknown values

- For unknown values (320 UNK values) such as episode numbers we manually searched the web (IMDb and latest myanimelist.net data) and entered the details for whichever anime's data was available
- For movies we know that they are single entities (sub parts are considered separate) and we have given rating = 1 for movies
- For Anime that are grouped are "OVA" (Original Video Animation) we have substituted 1 for the unknown values (OVAs are generally long 1 episode long)
- Finally, we removed all the anime which were left with unknown values after the above cleaning steps.

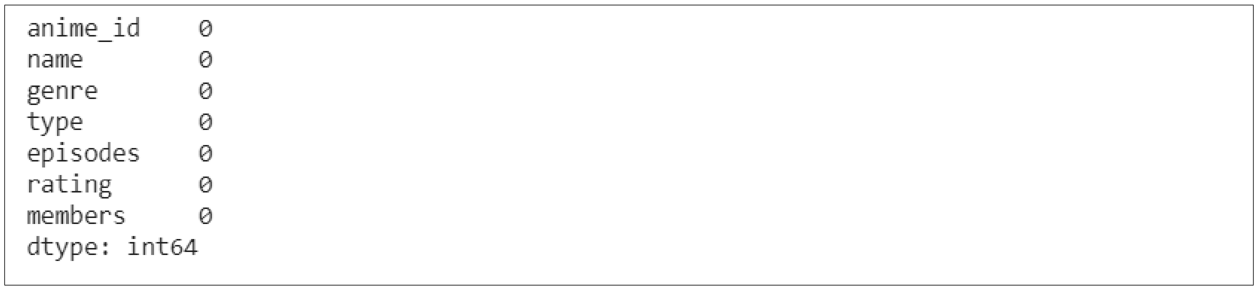
Pre-processed dataset with no-unknown values and updated anime list with ratings, episodes and latest anime:



	anime_id	name	genre	type	episodes	rating	members
0	32281.0	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630.0
1	5114.0	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26	793665.0
2	28977.0	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.25	114262.0
3	9253.0	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572.0
4	9969.0	Gintama&#039;	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.16	151266.0
...	...	...	...	...	...	...	...
12319	9316.0	Toushindai My Lover: Minami tai Mecha-Minami	Hentai	OVA	1	4.15	211.0
12320	5543.0	Under World	Hentai	OVA	1	4.28	183.0
12321	5621.0	Violence Gekiga David no Hoshi	Hentai	OVA	4	4.88	219.0
12322	6133.0	Violence Gekiga Shin David no Hoshi: Inma Dens...	Hentai	OVA	1	4.98	175.0
12323	26081.0	Yasuji no Pornorama: Yacchimae!!	Hentai	Movie	1	5.46	142.0

12223 rows × 7 columns

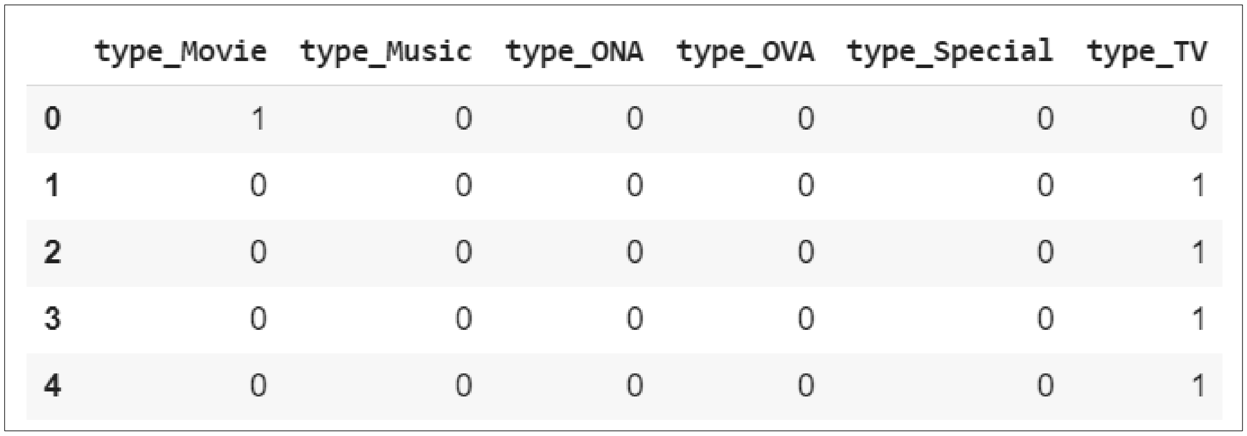
Checking the pre-processed dataset for unknown values:



```
anime_id    0
name        0
genre       0
type        0
episodes    0
rating      0
members     0
dtype: int64
```

2) Creation of dummy variables for the feature 'type'

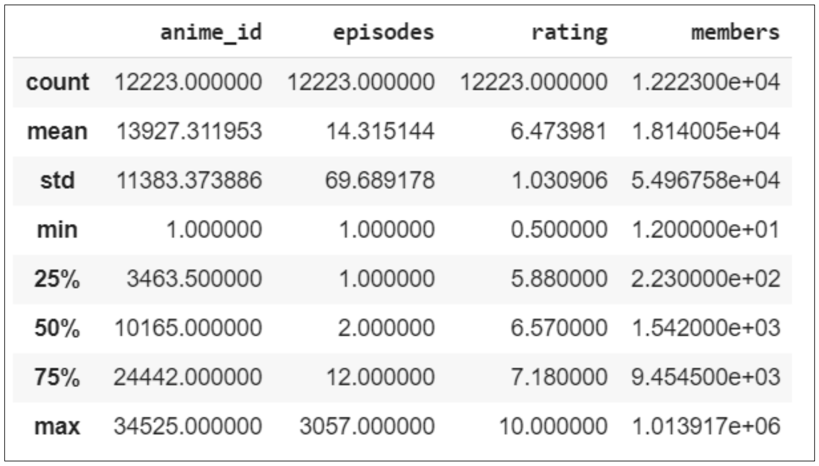
We created dummy variables where each row represents an anime and column represents the type of anime. If an anime corresponds to one type of anime, value of 1 would be assigned or else 0 (one hot encoding). This pre-processing is required for content-based filtering.



	type_Movie	type_Music	type_ONA	type_OVA	type_Special	type_TV
0	1	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	0	1
3	0	0	0	0	0	1
4	0	0	0	0	0	1

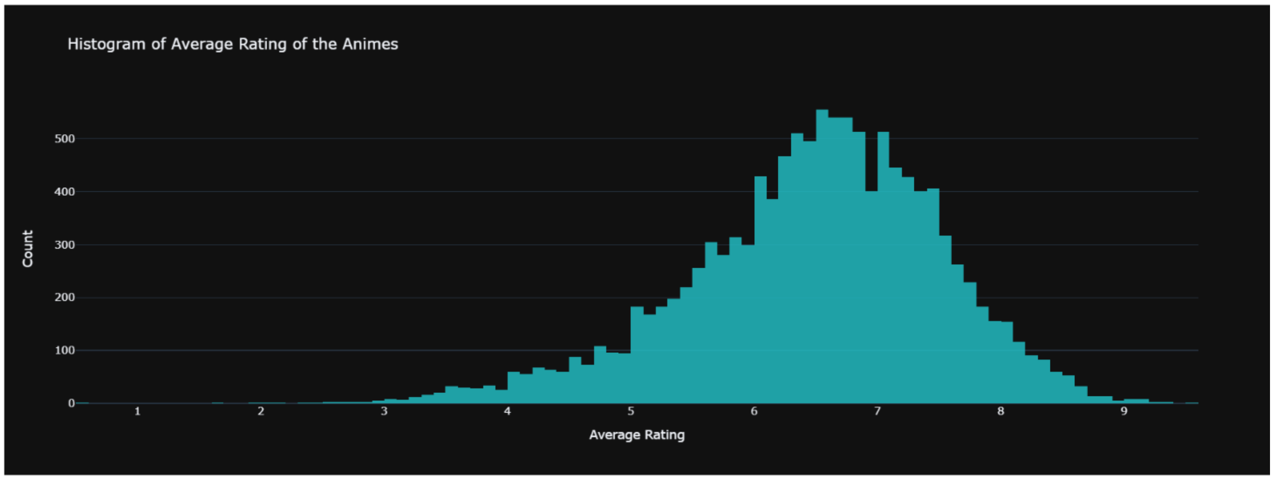
4. Exploratory Data Analysis

1) Counts and measures of central tendencies for anime.csv dataset:



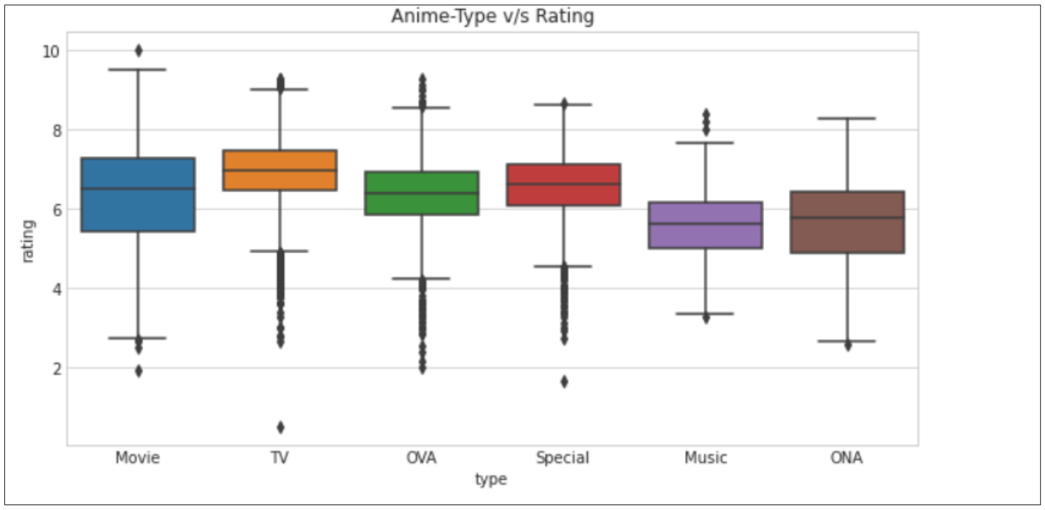
	anime_id	episodes	rating	members
count	12223.000000	12223.000000	12223.000000	1.222300e+04
mean	13927.311953	14.315144	6.473981	1.814005e+04
std	11383.373886	69.689178	1.030906	5.496758e+04
min	1.000000	1.000000	0.500000	1.200000e+01
25%	3463.500000	1.000000	5.880000	2.230000e+02
50%	10165.000000	2.000000	6.570000	1.542000e+03
75%	24442.000000	12.000000	7.180000	9.454500e+03
max	34525.000000	3057.000000	10.000000	1.013917e+06

We observed that on an average the rating is around 6.5 with lowest rating being 0.5 and highest being 10. Furthermore, the below histogram tells us that the distribution of ratings is negatively skewed (most of the anime are appreciated by the users!)



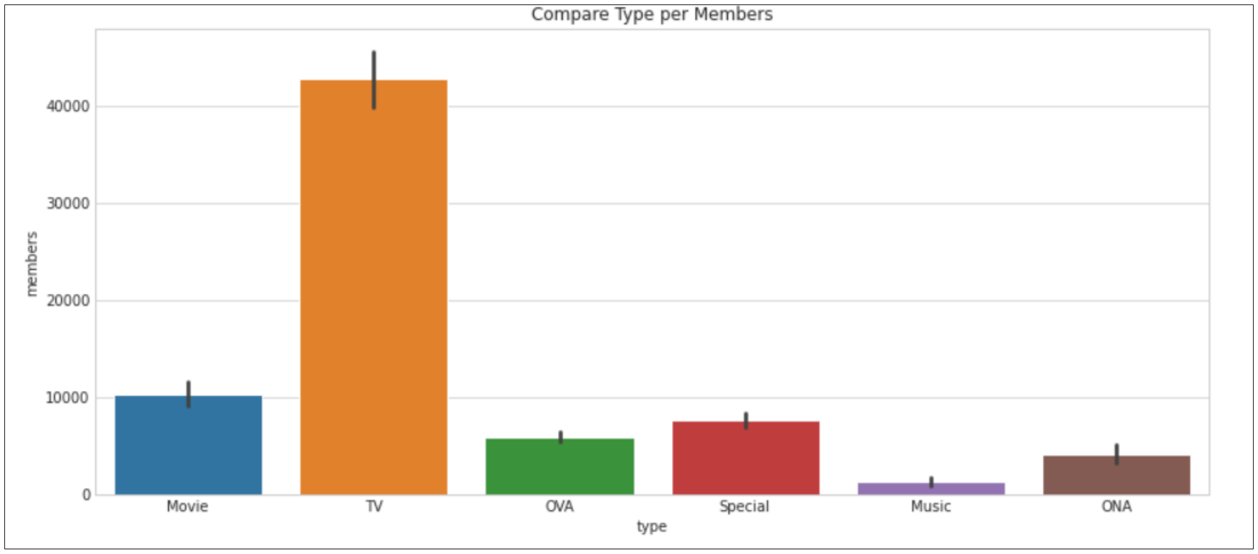
<u>From the above histogram:</u>
Anime less than rating **8.0**: 11581 - relative to **94.75%** of the data
Anime less than rating **7.0**: 8245 - relative to **67.45%** of the data
Anime less than rating **6.0**: 3409 - relative to **27.89%** of the data
Anime less than rating **5.0**: 1008 - relative to **8.25%** of the data
Anime less than rating **4.0**: 240 - relative to **1.96%** of the data
Anime less than rating **3.0**: 26 - relative to **0.21%** of the data
Anime less than rating **2.0**: 3 - relative to **0.02%** of the data

2) Average ratings based on 'Type' feature



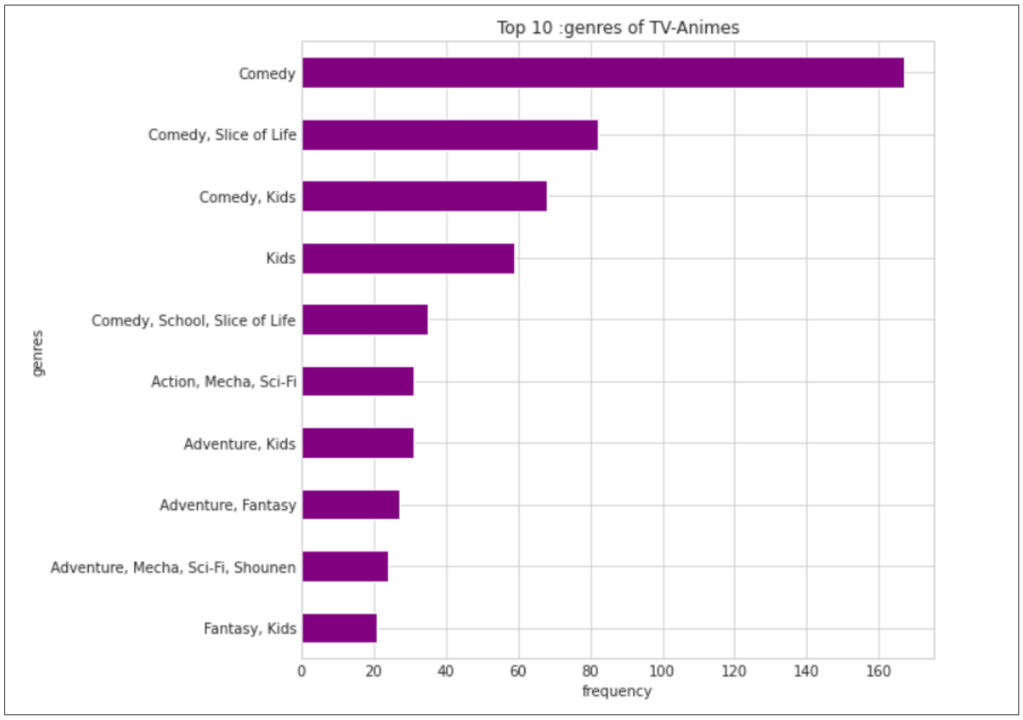
We observed that average ratings are highest for type TV and movie with many outliers. We don't wish to remove these outliers because these users may have contrasting preference than most users.

3) Frequency of members based on 'type'



TV has the highest frequency of members in its community forum. Therefore, we can say that the most popular type of anime is TV followed by movie. Rest of the types are less popular and at similar levels with music being the least popular (musicals can be hard to sit through!!)

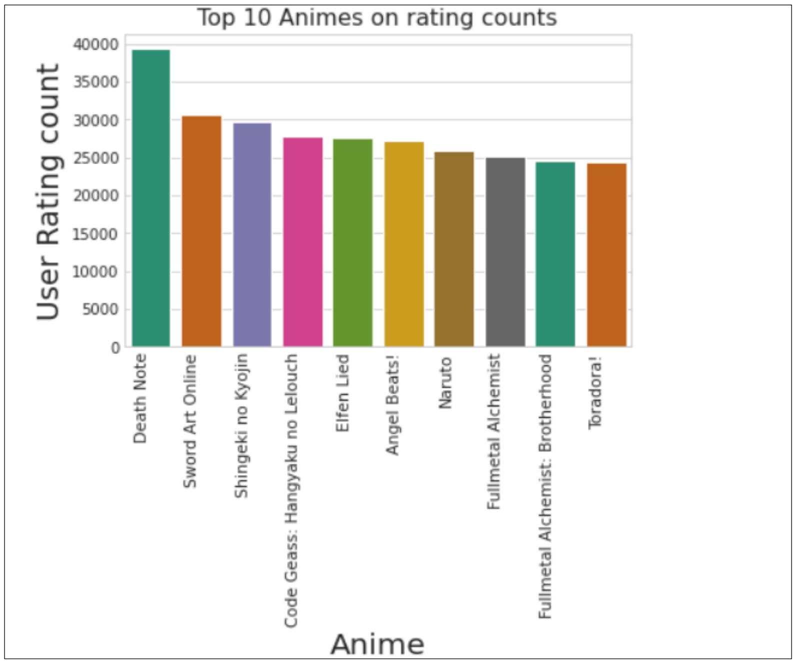
4) Frequency for ‘Genre’ feature for type TV (TV has the biggest user community)



Japanese people like humor! Most of the anime are of genre comedy when it comes to TV series. However, running the same plots for type Movie and OVA we get very different results where genre like ‘Dementia’ and ‘Hentai’ are the more in frequency (More EDA outputs are present in the python notebook).

5) Top 10 most rated anime

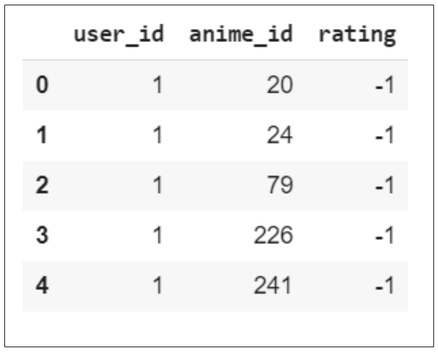
The above graph displays the anime which have received most number or ratings (hence views). We have Death note, followed Sword Art Online. It's surprising to note that few famous anime namely Naruto, Full metal Alchemist etc. have not been rated by lots of users. Our intuition for this observation is that they are very famous anime and users tend to not rate them as they have already seen it or feel that these shows are well known, hence, they don’t need rating and should be watched anyhow.



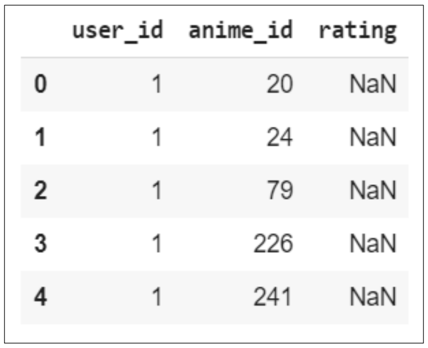
5. Results

5.1 Collaborative filtering

As discussed in the prior sections, we first **converted all the -1 into NaN**. This is because we need to compute average ratings in this section and -1 will distort our computations by introducing a bias since we have a sparse matrix where most of the users have not rated anime and hence there are majority of -1 values.



	user_id	anime_id	rating
0	1	20	-1
1	1	24	-1
2	1	79	-1
3	1	226	-1
4	1	241	-1

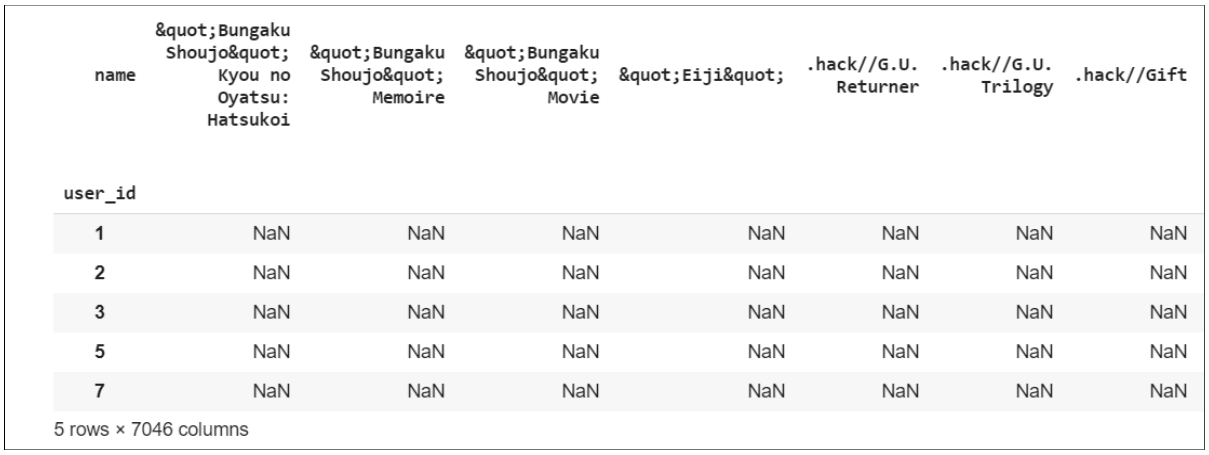


	user_id	anime_id	rating
0	1	20	NaN
1	1	24	NaN
2	1	79	NaN
3	1	226	NaN
4	1	241	NaN

Next, we considered a **subset of 25000 users and their ratings from the super set of 73516 users**. Reduction in dataset was done owing to less computation power (12GB Google Colab RAM crashes if we construct a sparse matrix for more than 25000 users). How did we select these 25000 users? During our exploratory data analysis we saw that TV has the highest number of members in the community as well has highest average ratings, followed by movies and OVA respectively. Hence, we only selected those users who have rated TV, movie or OVA type anime and selected 25000 users from the pool.

Finally, we created a pivot table selecting the three columns, user_id, anime name and ratings. **This pivot table is our sparse matrix** which we will use to apply collaborative filtering on our data. **On the x-axis of our pivot table we have all the anime names and, on the y-axis, we have all the user_id.** A combination of x-y gives us the rating the anime x has received from user y.

Below is the pivot table, many NaN are there as this is a sparse matrix:



name	&quot;Bungaku Shoujo&quot; Kyou no Oyatsu: Hatsukoi	&quot;Bungaku Shoujo&quot; Memoire	&quot;Bungaku Shoujo&quot; Movie	&quot;Eiji&quot;	.hack//G.U. Returner	.hack//G.U. Trilogy	.hack//Gift
user_id							
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 7046 columns

5.1.1 Item-Item collaborative filtering

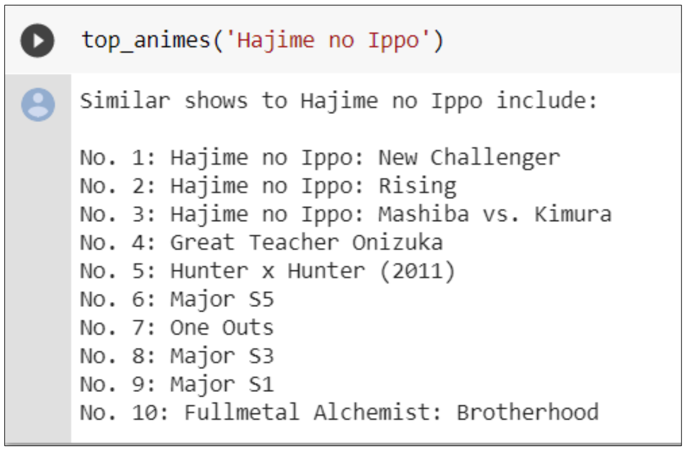
Hypothesis:
Ho (Null hypothesis): Recommended items are not similar
Ha (Alternate hypothesis): Recommended items are similar

Once, the sparse matrix is ready, we need to normalize our data before we can proceed with the computations. For normalization, we are subtracting the row average from each rating and any user who has only one rating or rated every anime the same, will be dropped. The ratings will be centered around 0 now and any rating which is above 0 will be considered above average and anything below zero will be below average rating.
Now, we will finally compute the cosine similarity between anime-anime (item-item) pair.

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined to equal the cosine of the angle between them, which is also the same as the inner product of the same vectors normalized to both have length 1.

$$\text{Cosine-Similarity (anime1, anime2)} = \frac{(\text{anime1} * \text{anime2})}{(|\text{anime1}||\text{anime2}|)}$$

After computing the cosine similarities, we will be inserting the similarity matrices into data frame objects so that we can perform operations and can further start the main task of recommendation.
Finally, we will ask the item-item based system to recommend us **top 10 anime similar to a particular anime of our choice**, say "**Hajime No Ippo**":



```
top_animes('Hajime no Ippo')

Similar shows to Hajime no Ippo include:

No. 1: Hajime no Ippo: New Challenger
No. 2: Hajime no Ippo: Rising
No. 3: Hajime no Ippo: Mashiba vs. Kimura
No. 4: Great Teacher Onizuka
No. 5: Hunter x Hunter (2011)
No. 6: Major S5
No. 7: One Outs
No. 8: Major S3
No. 9: Major S1
No. 10: Fullmetal Alchemist: Brotherhood
```

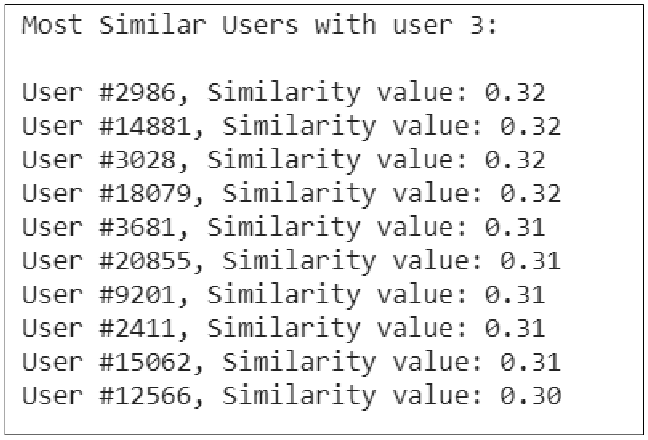
As we can see, the **top 3 suggestions are the sequels** to Hajime No Ippo **followed by similar action-based anime!** We know that we **did not feed any information to the system about prequels, sequels** or the genre. **All we provided was the ratings** along with user_id and anime name and the system itself suggested the sequels to be most similar to the input anime. Rest of the suggestion are similar action anime and we can see another series, Major S1, S3 and S5 in the recommendations. **Sequels are**

essentially similar in storylines and preference. The algorithm could capture the same with least amount of information provided to it. **Hence, we reject the Null hypothesis in favor of alternate hypothesis and conclude that the recommended items are similar and the suggested anime are indeed similar to the input anime.**

5.1.2 User-User collaborative filtering

Hypothesis:
Ho (Null hypothesis): Recommended users are not similar
Ha (Alternate hypothesis): Recommended users are similar

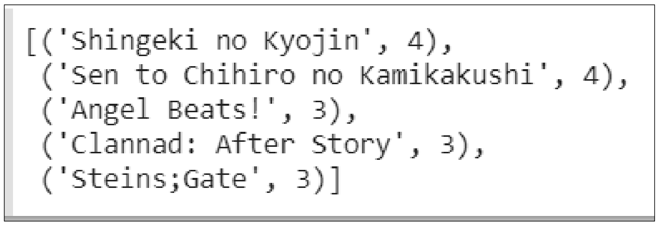
Using the normalized pivot table, we will compute cosine similarity and store the similarity matrix into data-frame object for ease of calculation. As we saw in section 1.3.1, we will calculate user-user similarity and see which are the top 10 most similar users to the input user.
Below are the top 10 users similar to user_id #3:



```
Most Similar Users with user 3:

User #2986, Similarity value: 0.32
User #14881, Similarity value: 0.32
User #3028, Similarity value: 0.32
User #18079, Similarity value: 0.32
User #3681, Similarity value: 0.31
User #20855, Similarity value: 0.31
User #9201, Similarity value: 0.31
User #2411, Similarity value: 0.31
User #15062, Similarity value: 0.31
User #12566, Similarity value: 0.30
```

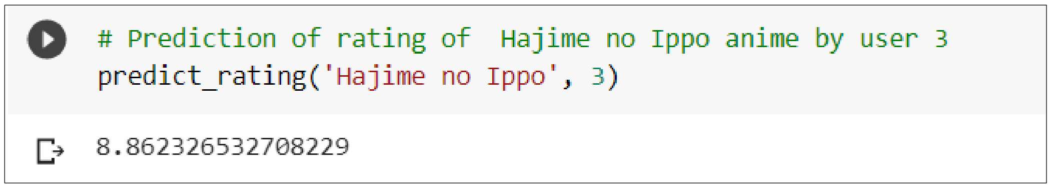
From the below image, we can see the **lists of lists containing the highest rated shows per similar user** and returns the **name of the show along with the frequency it appears in the list**. Each show appears **atleast 3-4 times.** This proves that the top users which are similar to the input user, have rated these shows in the **same manner or preference**. **Hence, we can reject the NULL hypothesis in favour of Alternate hypothesis and conclude that users are similar.** However, we should notice , that item-item gave us much reliable recommendations than user-user.



```
[('Shingeki no Kyojin', 4),
 ('Sen to Chihiro no Kamikakushi', 4),
 ('Angel Beats!', 3),
 ('Clannad: After Story', 3),
 ('Steins;Gate', 3)]
```

5.2 Evaluating the collaborative recommender system: RMSE

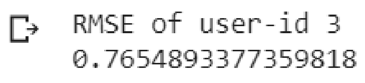
For a particular user, say user_id 3, we will first predict the rating it will give to a particular anime. In the below snippet, we see that user 3's predicted rating for anime 'Hajime no Ippo' is around 8.86. Which means that this user will like watching this show.



```
# Prediction of rating of  Hajime no Ippo anime by user 3
predict_rating('Hajime no Ippo', 3)
```

```
8.862326532708229
```

Now we will evaluate the accuracy of this predicted rating against the actual rating in the database and calculate RMSE (Root Mean Square Error). RMSE is used to measure the differences between values (sample or population values) predicted by a model or recommender and the values observed.



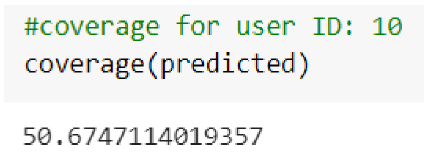
```
RMSE of user-id 3
0.7654893377359818
```

RMSE is just 0.7. That means the actual rating was close to 8. Hence, the recommender system made a close prediction and works well. We further calculated RMSE for few more anime-user combination and our average RMSE value was around 0.46. Below are some of the users and their individual RMSE scores. Therefore, the recommender system has shown good accuracy and low error rate for the recommendations.

User_id	3	7	8	43	27	14	21	27	24	25
RMSE	0.81	0.76	0.52	0.88	0.46	0.53	0.814	0.46	0.52	0.91

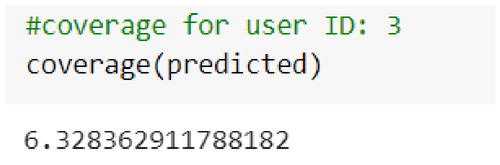
5.3 Non-traditional metrics of evaluation: Coverage

Coverage is the percent of items in the training data the model is able to recommend on a test set. If the recommender system is too accurate it can end up recommending the same type of products over and over again leading to a monotonous experience. This metric trades off accuracy for new discoveries in order two break this monotony. Higher the coverage, the better.



```
#coverage for user ID: 10
coverage(predicted)
```

```
50.6747114019357
```



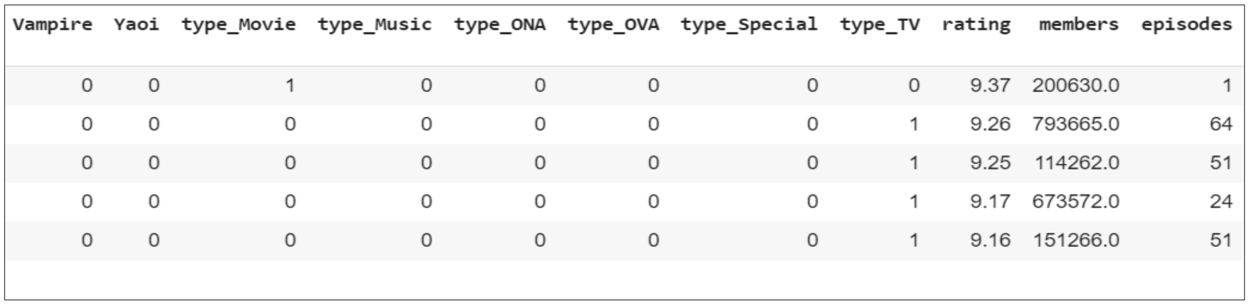
```
#coverage for user ID: 3
coverage(predicted)
```

```
6.328362911788182
```

As shown above, we calculated coverage for various ranges of users and not surprisingly for few of the users the coverage percent was around 50.5 % and as low as 6%. This is due to the fact that not all user_id rated a lot of anime so that the recommendation system can recommend or guess the anime that particular user_id would like.

5.4 Content based filtering

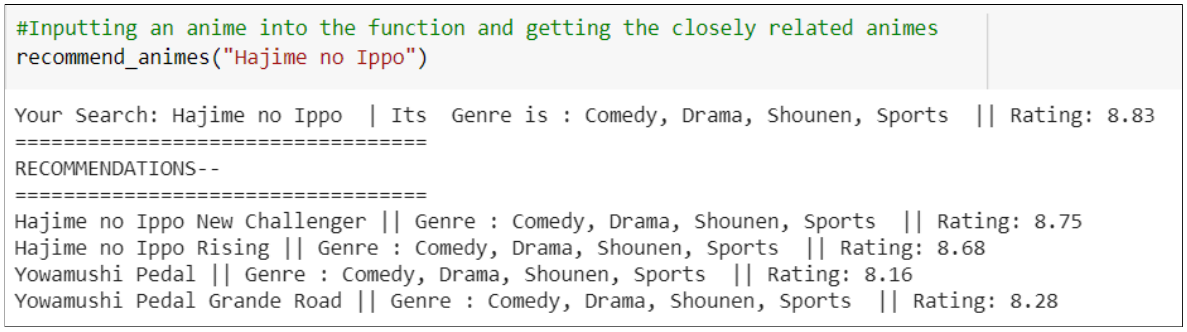
For content-based filtering, we will combine the item (anime) features along with the dummy variables that we created earlier as show below:



Vampire	Yaoi	type_Movie	type_Music	type_ONA	type_OVA	type_Special	type_TV	rating	members	episodes
0	0	1	0	0	0	0	0	9.37	200630.0	1
0	0	0	0	0	0	0	1	9.26	793665.0	64
0	0	0	0	0	0	0	1	9.25	114262.0	51
0	0	0	0	0	0	0	1	9.17	673572.0	24
0	0	0	0	0	0	0	1	9.16	151266.0	51

Next step, using a scaling function we translate each feature individually such that the maximal absolute value of each feature will be 1.0. It does not shift/center the data, and thus, does not destroy any sparsity. The data is now ready for us to apply KNN.

1) Recommendations with KNN=5 and all the features:

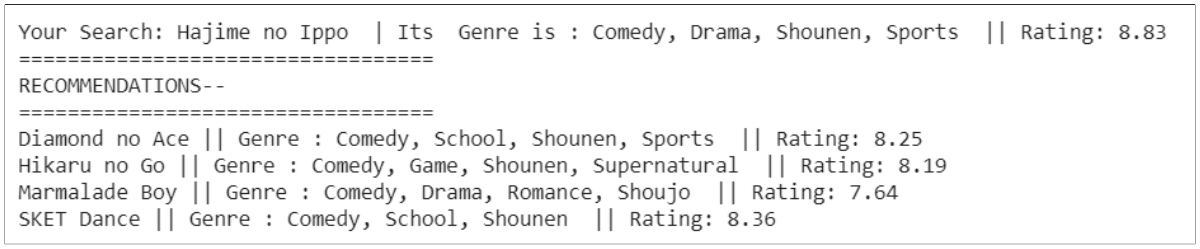


```
#Inputting an anime into the function and getting the closely related animes
recommend_animes("Hajime no Ippo")
```

```
Your Search: Hajime no Ippo  | Its  Genre is : Comedy, Drama, Shounen, Sports  || Rating: 8.83
==================================
RECOMMENDATIONS--
==================================
Hajime no Ippo New Challenger || Genre : Comedy, Drama, Shounen, Sports  || Rating: 8.75
Hajime no Ippo Rising || Genre : Comedy, Drama, Shounen, Sports  || Rating: 8.68
Yowamushi Pedal || Genre : Comedy, Drama, Shounen, Sports  || Rating: 8.16
Yowamushi Pedal Grande Road || Genre : Comedy, Drama, Shounen, Sports  || Rating: 8.28
```

From the above screenshot, we can see that all the anime recommended by the system, have **exactly similar genre and ratings**. **All the ratings are above 8**. Moreover, it two of the recommendations are the sequel of the anime "Hajime no Ippo". Hence, with all the features at KNN=5, the system has suggested highly accurate anime with respect to the input anime.

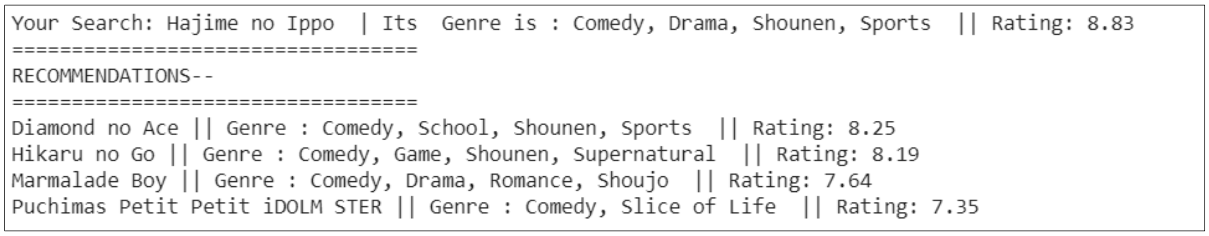
2) Recommendation with KNN=5 and removing one feature "members"



```
Your Search: Hajime no Ippo  | Its  Genre is : Comedy, Drama, Shounen, Sports  || Rating: 8.83
==================================
RECOMMENDATIONS--
==================================
Diamond no Ace || Genre : Comedy, School, Shounen, Sports  || Rating: 8.25
Hikaru no Go || Genre : Comedy, Game, Shounen, Supernatural  || Rating: 8.19
Marmalade Boy || Genre : Comedy, Drama, Romance, Shoujo  || Rating: 7.64
SKET Dance || Genre : Comedy, School, Shounen  || Rating: 8.36
```

With a single feature reduced, we see a different subset of recommendations. The genre and ratings are still similar but with lesser accuracy. We now have a suggestion with rating 7.6 and the system is no longer suggesting the sequels. Additionally, we have new genre like school and supernatural in the recommendations.

3) Recommendation with KNN=5 and removing a subset of two features “members and type”



```
Your Search: Hajime no Ippo  | Its  Genre is : Comedy, Drama, Shounen, Sports  || Rating: 8.83
==================================
RECOMMENDATIONS--
==================================
Diamond no Ace || Genre : Comedy, School, Shounen, Sports  || Rating: 8.25
Hikaru no Go || Genre : Comedy, Game, Shounen, Supernatural  || Rating: 8.19
Marmalade Boy || Genre : Comedy, Drama, Romance, Shoujo  || Rating: 7.64
Puchimas Petit Petit iDOLM STER || Genre : Comedy, Slice of Life  || Rating: 7.35
```

With more features removed, we see a further drop in the ratings of suggested anime. There is another genre, slice of life which is being recommended by the system. We have two anime with ratings below 8 and no sequels. Therefore, it has even lower accuracy then the previous model.

Therefore, the system with all the features gave the best recommendations to the input anime. The accuracy of recommendations kept getting lower with the removal of each feature. Moreover, we can see that item-item based collaborative filtering recommended us all the sequels (three) and the content-based approach recommended us only two sequels. Hence, item-item based collaborative filtering has performed better than content-based recommenders.

6. Implementation: difficulties and insights

In implementing the recommendation System, we observed that it's always good to have a cloud space to perform computations on large dataset. Initially, we worked with our datasets on local machines however, dealing with two large datasets, one having 12k+ rows and the second one with 1048k+ rows, was a slow process and required a lot of computing time, especially dealing with really large pivot tables and sparse matrices. Using the GPU’s and RAM provided by Google Colab rather than our personal systems, catalyzed our processing speed at every stage.
Initially we aimed to process all 76k+ user ids but due to computation limitation we were only able to incorporate 25000 user ids. Even Google Colab’s RAM was crashing with out of memory error. Owing to this, we picked the maximum value at which the Collaboratory did not crash, i.e., 25000.

From evaluation standpoint, we noticed that it is can be cumbersome to assess the quality of a recommendation system. How do we know if a new recommendation is relevant prior to actually recommending it to our user? Moreover, it is not always that our model outputs numeric values such as ratings predictions or matching probabilities, like we saw in the above sections, many of the recommendations were categorical where similar anime were suggested. Error metrics like RMSE do not always cover all the aspects of evaluation in such cases. One way to test the ability of our recommender system is to release it to a bunch of users and monitor the actions they take based on the recommendations and evaluate the performance based on user satisfaction. This will however be a post-production test and would require dedicated funds and resources which can yield loss if the system doesn’t work as per the expectations.

7. Learnings from the project

During the course of this project, a major portion of our time was spent on data pre-processing and cleaning. About 18% of the data from one of the datasets being used was missing, and it required tremendous manual effort to scrape the source websites for our missing data. Moreover, since the data was quite old, we had to cross check the valid values too, and ensure that the outdated values were updated. Therefore, applying machine learning algorithm is not the only pivotal task. With illegitimate data, even a strong algorithm can crumble and give biased or random results owing to a weak backbone. Hence, a lot of proactive work should be undertaken during data collection itself where such websites or companies can standardize the inputs and create business rules to regulate the data entry and ensure a simple process via extraction, transformation and loading tools to constantly refresh the data at regular intervals.

8. Conclusion & future work

Upon comparing the various methods, we observed that item-item based collaborative filtering gave best results. Moreover, collaborative filtering in general had more advantages over content-based filtering method. One of them being the generation on best output using least features. The collaborative method didn't require additional context such as anime genre or user age etc. Secondly, focusing only on the anime features results in forgetting the dynamic nature of human moods and preferences. I may have enjoyed comedy anime few years back but lately I have been watching action-based anime. Since the essence of collaborative filtering is user-item interaction, the user preferences are updated and recorded every time they watch and rate something new. However, collaborative filtering does have one disadvantage, i.e. the cold start problem. Since the recommendations are dependent upon user-item interaction and absolutely no interactions are present for a new anime, the model will not know whom to recommend this anime until someone views it. This is where the features in content-based system do come in handy. Suppose a new action anime is released and the system knows through my interactions that I rate action anime highly, it will recommend me this new anime based on the genre feature.

Hence, for future work, a hybrid model which incorporates the advantages of both the features can be constructed for anime recommendation. We can train the two models independently and then collate their recommendations. Or we could deep dive into deep learning and experiment on the lines of a neural network. This neural network model should be able to take all the features as inputs from content-based perspective and simultaneously a user-interaction matrix from collaborative filtering perspective.

9. References

The following research papers / web links were the motivation behind this project and were referred to as part of the literature review.

[1] http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

[2] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.736.6037&rep=rep1&type=pdf

[3] https://ieeexplore.ieee.org/abstract/document/7991602

[4] https://aja.gr.jp/english/japan-anime-data

[5] https://heartbeat.fritz.ai/recommender-systems-with-python-part-ii-collaborative-filtering-k-nearest-neighbors-algorithm-c8dcd5fd89b2

[6] http://users.ics.forth.gr/~potamias/mlnia/paper_6.pdf

[7] https://pdfs.semanticscholar.org/1e6b/0c387c62b3dfdde8b8226fc206e41e72e7d9.pdf

[8] https://www.sciencedirect.com/science/article/pii/S1110866515000341

[9] https://developers.google.com/machine-learning/recommendation

[10] https://www.hindawi.com/journals/aai/2009/421425/

Appendix

Following Python code was implemented.

1. ANIME DATASET EDA Python File

```
#Importing the colab drive library to get the dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import seaborn as sns
import scipy as sp
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
dataset = pd.read_csv("/content/drive/My Drive/Anime_Recommend_System/anim
e_new.csv")
dataset_old = pd.read_csv("/content/drive/My Drive/Anime_Recommend_System/
anime_new.csv")
new_ratings=pd.read_csv("/content/drive/My Drive/Anime_Recommend_System/an
ime_updated.csv")
rating =pd.read_csv("/content/drive/My Drive/Anime_Recommend_System/rating
.csv")
dataset.update(new_ratings)

import seaborn as sns
plt.style.use('seaborn-whitegrid')
sns.set_style('whitegrid')


# Getting the statistics of the original dataset
print('ORIGINAL :dataset shape of Anime Dataset before cleaning: \n')
print('number of rows: ',dataset_old.shape[0],' number of columns: ',datas
et_old.shape[1])
print('dataset column names: \n')
print(dataset_old.columns)
print('columns data-type')
print(dataset_old.dtypes)
print('---missing data---')
c=dataset_old.isnull().sum()
print(c[c>0])
```

```
#Statistics of the new dataset which we would carry on our future tasks
print('UPDATED: dataset shape of Anime Dataset after updated data points:
\n')
print('number of rows: ',dataset.shape[0],' number of columns: ',dataset.s
hape[1])
print('dataset column names: \n')
print(dataset.columns)
print('columns data-type')
print(dataset.dtypes)
print('---missing data---')
c=dataset.isnull().sum()
print(c[c>0])
```

Data Cleaning

```
#There are lots of values which are missing or unknown and thus needs to b
e addressed
#Hentai Anime are adult rated 1 episode anime only thus we can interpolate
 the values of the episodes
dataset.loc[(dataset["genre"]=="Hentai") & (dataset["episodes"]=="Unknown"
),"episodes"] = "1"

#OVAs are special episodes or animes with  1 episodes only
dataset.loc[(dataset["type"]=="OVA") & (dataset["episodes"]=="Unknown"),"e
pisodes"] = "1"

#Movies are  of 1 episode only (ie there are no episodes for the movies)
dataset.loc[(dataset["type"] == "Movie") & (dataset["episodes"] == "Unknow
n"),"episodes"] = "1"

#Dropping unknown episodes
dataset.drop(dataset.loc[dataset["episodes"]=="Unknown"].index, inplace=Tr
ue)
dataset.dropna(subset = ["rating"], inplace=True)

#Mean of ratings of various Genre
for i in dataset['type'].unique().tolist():
    print('mean of '+str(i)+' :\n')
    print(dataset[dataset['type']==i]['rating'].mean())
```

Data Visualization

```
#Visualizing and finding out top 10 genres of TV type animes
TV_anime=dataset[dataset['type']=='TV']
```

```
TV_anime['genre'].value_counts().sort_values(ascending=True).tail(10).plot
.barh(figsize=(8,8))
plt.title('Top 10 :genres of TV-Animes')
plt.xlabel('frequency')
plt.ylabel('genres')
plt.show()

#Visualizing Movie type Anime and finding out top Genres for the same
Movie_anime=dataset[dataset['type']=='Movie']
Movie_anime['genre'].value_counts().sort_values(ascending=True).tail(10).p
lot.barh(figsize=(8,8))
plt.title('Top 10 :genres of Movie - Anime')
plt.xlabel('frequency')
plt.ylabel('genres')
plt.show()

#Top 5 Genres for OVA type Anime
OVA_anime=dataset[dataset['type']=='OVA']
OVA_anime['genre'].value_counts().sort_values(ascending=True).tail(5).plot
.barh(figsize=(8,8))
plt.title('Top 5 OVA - Anime- Genre Distribution')
plt.xlabel('frequency')
plt.ylabel('genres')
plt.show()

ONA_anime=dataset[dataset['type']=='ONA']
ONA_anime['genre'].value_counts().sort_values(ascending=True).tail(10).plo
t.barh(figsize=(8,8))
plt.title('Top 10 : ONA - Anime- Genre Distribution')
plt.xlabel('frequency')
plt.ylabel('genres')
plt.show()

#Rating and members distribution
fig=plt.figure(figsize=(13,5))
for i,j in zip(TV_anime[['rating','members']].columns,range(3)):
    ax=fig.add_subplot(1,2,j+1)
    sns.distplot(TV_anime[i],ax=ax)
    plt.axvline(TV_anime[i].mean(),label='mean',color='blue')
    plt.axvline(TV_anime[i].median(),label='median',color='green')
    plt.axvline(TV_anime[i].std(),label='std',color='red')
    plt.title('{} distribtion'.format(i))
    plt.legend()
plt.show()
```

```
#Visualizing Ratings v/s type of Anime
plt.figure(figsize=(10,5))
sns.boxplot(x='type',y='rating',data=dataset)
plt.title('Anime-Type v/s Rating')
plt.show()

#Highest Rated Anime Type- TV
TV_anime[TV_anime['rating']==TV_anime['rating'].max()]

#Lowest Rated Anime- Type - TV
TV_anime[TV_anime['rating']==TV_anime['rating'].min()]

#MyAnimeList.net is a Anime community, visualizing which type of anime is
most popular among members
plt.figure(figsize=(14,6))
plt.title("Members per Type")
sns.barplot(x=dataset['type'], y=dataset['members'])

#Rating of Animes less then 8
for p in range(8, 1, -1):
    total=dataset.query("rating <"+str(p))['rating'].count()
    print("Anime less than rating %.1f: %d relative of the %.2f%% of the d
ata" % (p, total, ((total*100)/dataset.shape[0])))

import plotly.express as px
#Plotting the histogram for the ratings of all the concerned animes , we s
ee that majority of the animes are rated around 6.5 to 6.59 out on the sca
le of 10
fig = px.histogram(dataset,
                 x       = 'rating',
                 nbins   = 100,
                 opacity = 0.8,
                 title   = 'Histogram of Average Rating of the Animes',
                 template = 'plotly_dark',
                 color_discrete_sequence=['#23c6cc','#23c6cc']
                )


fig.update_xaxes(title='Average Rating')
fig.update_yaxes(title='Count')
fig.show()

#Anime with maximum number of episodes in the dataset
dataset[dataset['episodes']==dataset['episodes'].max()]
```

```
#From the rating dataset we can see that majority of the users rated any a
nime an score of 8 , and also here -
1 signifies that there are lots of animes that are not rated by the
#users
rating['rating'].value_counts().sort_values(ascending=True).tail(10).plot.
barh(figsize=(8,8))
plt.title('Rating Frequency by the users')
plt.xlabel('number of users rated')
plt.ylabel('Rating out of 10')
plt.show()

print("There are ",rating.shape[0]," rows in the rating dataset")
print("Out of which ",rating[rating.rating == -
1].shape[0],"are the rows/animes which are not rated (have rating =-1)")


#Combining both datasets to get a clear picture

combined_dataset=pd.merge(dataset,rating,on='anime_id',suffixes= ['', '_us
er'])
combined_dataset = combined_dataset.rename(columns={'name': 'anime_title',
 'rating_user': 'user_rating'})
# Creating a dataframe for rating counts
combine_anime_rating = combined_dataset.dropna(axis = 0, subset = ['anime_
title'])
anime_rating_count = (combine_anime_rating.
     groupby(by = ['anime_title'])['user_rating'].
     count().
     reset_index().rename(columns = {'rating': 'totalRatingCount'})
    [['anime_title', 'user_rating']]
    )


top10_animerated=anime_rating_count[['anime_title', 'user_rating']].sort_v
alues(by = 'user_rating',ascending = False).head(10)
ax=sns.barplot(x="anime_title", y="user_rating", data=top10_animerated, pa
lette="Dark2")
ax.set_xticklabels(ax.get_xticklabels(), fontsize=11, rotation=90, ha="rig
ht")
ax.set_title('Top 10 Animes on rating counts',fontsize = 15)
ax.set_xlabel('Anime',fontsize = 20)
ax.set_ylabel('User Rating count', fontsize = 20)

#Finding the top 10 communities sizes of Anime present in the dataset
anime_2=combined_dataset.copy()
```

```
anime_2.drop_duplicates(subset ="anime_title",
                     keep = 'first', inplace = True)
top10_animemembers=anime_2[['anime_title', 'members']].sort_values(by = 'm
embers',ascending = False).head(10)
ax=sns.barplot(x="anime_title", y="members", data=top10_animemembers, pale
tte="gnuplot2")
ax.set_xticklabels(ax.get_xticklabels(), fontsize=11, rotation=90, ha="rig
ht")
ax.set_title('Top 10 Anime based on members',fontsize = 15)
ax.set_xlabel('Anime',fontsize = 20)
ax.set_ylabel('Community Size', fontsize = 20)
```

2. CONTENT BASED FILTERING PYTHON CODE

```
#Importing required libraries for this project
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import seaborn as sns
from sklearn.preprocessing import MaxAbsScaler
from sklearn.neighbors import NearestNeighbors
%matplotlib inline

# Importing our Anime dataset, which includes 30 more animes, not present
in original dataset. Though this dataset is not preprocessed and contains
NaN and unknowns
dataset = pd.read_csv("/content/drive/My Drive/Anime_Recommend_System/anim
e_new.csv")

#Checking the dataset few top rows
dataset.head(3)

#Now we need to filter and see if there are any episodes which are not ava
ilable in our dataset
filtered_data = dataset[dataset["episodes"]=="Unknown"]

#Now importing our scrapped file which has updated ratings and episodes wh
ich we scraped from the original source
```

```
new_ratings=pd.read_csv("/content/drive/My Drive/Anime_Recommend_System/an
ime_updated.csv")

#Now viewing the newly imported table which is replica of the old dataset
but with new ratings and updated episodes number
new_ratings

#As we have now new updated ratings and episodes numbers from the scrapped
 source website, we update our main dataset with the updated values from n
ew_ratings file
dataset.update(new_ratings)

#Even after scrapping the data, there might be some Unknown episodes or ra
tings, thus creating another table to view them
new_filtered_data = dataset[dataset["episodes"]=="Unknown"]
new_filtered_data

#We see that there are view type of animes which we can interpolate values
 because these types of animes have fixed number of episodes
#The Hentai animes are adult rated animes and are of 1 episode
dataset.loc[(dataset["genre"]=="Hentai") & (dataset["episodes"]=="Unknown"
),"episodes"] = "1"

# OVAs are another special episodes or animes with  1 episodes only.
dataset.loc[(dataset["type"]=="OVA") & (dataset["episodes"]=="Unknown"),"e
pisodes"] = "1"

#Movies are of 1 episode only (ie there are no episodes for the movies)
dataset.loc[(dataset["type"] == "Movie") & (dataset["episodes"] == "Unknow
n"),"episodes"] = "1"

#After our above values are updated, checking again for the unknown values
dataset[dataset["episodes"]=="Unknown"]

# Now the missing values we have got, we cant predict or assume their epis
odes count or rating, thus, removing these datapoints
dataset.drop(dataset.loc[dataset["episodes"]=="Unknown"].index, inplace=Tr
ue)

#converting rating column to float data type
dataset["rating"] = dataset["rating"].astype(float)

#dropping all null values in rating column
dataset.dropna(subset = ["rating"], inplace=True)
dataset
```

```
#Checking again the whole dataset if there are still any unknown values le
ft or not
(dataset[dataset["episodes"]=="Unknown"]).count()
```

Implementation of Content Based

```
#Now we create dummy variable where each row represents an anime and colum
n represents the type of anime.If a anime corresponds to one type of anime
, value of 1 would be assigned
# or else 1  (one hot encoding)
pd.get_dummies(dataset[["type"]]).head()

#Converting members column as float datatype
dataset["members"] = dataset["members"].astype(float)

#Feature Selection
anime_features = pd.concat([dataset["genre"].str.get_dummies(sep=","),pd.g
et_dummies(dataset[["type"]]),dataset[["rating"]],dataset[["members"]],dat
aset["episodes"]],axis=1)

#Animes are Japanese, there might be many special characters present in th
e Animes names, thus using regex to keep all animes names in normal englis
h language
dataset["name"] = dataset["name"].map(lambda name:re.sub('[^A-Za-z0-
9]+', " ", name))

#Viewing the animes_features table we created one step above
anime_features.head()

#Viewing all the columns of the above dataframe, to get glimpse of various
 genre of animes
anime_features.columns

# translates each feature individually such that the maximal absolute valu
e of each feature in the training set will be 1.0. It does not shift/cente
r the data,
#and thus does not destroy any sparsity.
max_abs_scaler = MaxAbsScaler()
anime_features = max_abs_scaler.fit_transform(anime_features)

#Using KNN to find similar animes and then fitting it on our anime_feature
s
nearest_neighbours = NearestNeighbors(n_neighbors=5, algorithm='ball_tree'
).fit(anime_features)
distances, indices = nearest_neighbours.kneighbors(anime_features)
```

```
#Function to get index of anime by inputting anime name
def get_index_from_name(name):
    return dataset[dataset["name"]==name].index.tolist()[0]

#converting and putting all anime names in a list
all_anime_names = list(dataset.name.values)

# calling the function to get index of anime
get_index_from_name("Hajime no Ippo")

#Printing the array for the index of anime computed in previous cell
distances[20]

#Printing the indices of the above anime
indices[20]

#Function which takes anime name as an input and using the index gives the
 animes names list related to it
def recommend_animes(query=None):
    if query:
        found_id = get_index_from_name(query)
        print("Your Search:",query," | Its  Genre is :",dataset.loc[found_
id]["genre"]," || Rating:",dataset.loc[found_id]["rating"])
        print("==================================")
        print("RECOMMENDATIONS--")
        print("==================================")
        for id in indices[found_id][1:]:
            print(dataset.loc[id]["name"],"|| Genre :",dataset.loc[id]["ge
nre"]," || Rating:",dataset.loc[id]["rating"])
    else:
      print("Please enter an anime present in the dataset to get recommend
ation")


#Inputting an anime into the function and getting the closely related anim
es
recommend_animes("Hajime no Ippo")


# What if we change our feature selection? # Removing members feature from
 our model.
#Feature Selection
```

```
anime_features_2 = pd.concat([dataset["genre"].str.get_dummies(sep=","),pd
.get_dummies(dataset[["type"]]),dataset[["rating"]],dataset["episodes"]],a
xis=1)
anime_features_2.head(4)
max_abs_scaler_2 = MaxAbsScaler()
anime_features = max_abs_scaler_2.fit_transform(anime_features_2)

#Using KNN again
nearest_neighbours = NearestNeighbors(n_neighbors=5, algorithm='ball_tree'
).fit(anime_features_2)
distances, indices = nearest_neighbours.kneighbors(anime_features_2)
recommend_animes("Hajime no Ippo")

# What if we change our feature selection? # Removing members, type featur
e from our feature set
#Feature Selection
anime_features_2 = pd.concat([dataset["genre"].str.get_dummies(sep=","),pd
.get_dummies(dataset[["rating"]]),dataset["episodes"]],axis=1)
anime_features_2.head(4)
max_abs_scaler_2 = MaxAbsScaler()
anime_features = max_abs_scaler_2.fit_transform(anime_features_2)

#Using KNN again
nearest_neighbours = NearestNeighbors(n_neighbors=5, algorithm='ball_tree'
).fit(anime_features_2)
distances, indices = nearest_neighbours.kneighbors(anime_features_2)
recommend_animes("Hajime no Ippo")

#Here we see that there is discovery also (Genre : Slice of Life is recomm
ended rather then just Comedy,drama,shounen and sports), but here we see t
hat out of 4 recommendations
#2 recommendations are not rated near to the rating of the queried anime.
```

3. Collaborative Filtering Python Code

```
Importing the colab drive library to get the dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import seaborn as sns
import scipy as sp
from google.colab import drive
```

```
%matplotlib inline
from sklearn.metrics.pairwise import cosine_similarity
import operator
drive.mount('/content/drive')

#Importing the required  rating dataset as well as anime dataset
dataset=pd.read_csv("/content/drive/My Drive/Anime_Recommend_System/anime_
updated.csv")
rating =pd.read_csv("/content/drive/My Drive/Anime_Recommend_System/rating
.csv")
rating.head()

# Rating dataset has ample rows which have -
1 ratings , thus replacing all the -1 values with NaN
rating.rating.replace(-1,np.nan,inplace = True)
rating.head()

#Since most popular types of anime are TV, Movie and OVA , filtering for t
hose types of animes
genre = ['TV', 'Movie','OVA']
filtered_Dataset = dataset[dataset.type.isin(genre)]
filtered_Dataset.head()

# Now we have to merge both the ratings and the anime dataset using merge
function and we rename the column for easy name convention
merged_dataframe = rating.merge(filtered_Dataset, left_on = 'anime_id', ri
ght_on = 'anime_id', suffixes= ['_user', ''])
merged_dataframe.rename(columns = {'rating_user':'user_rating'}, inplace =
 True)

#We are limiting the dataframe length to 25,000 users as Google colab was
running out of memory if we try to process more users
merged_dataframe=merged_dataframe[['user_id', 'name', 'user_rating']]
merged_sub_dataframe= merged_dataframe[merged_dataframe.user_id <= 25000]
merged_sub_dataframe.head()

#Now we need to create pivot table of the above sub-
merged dataframe we acquired in the last frame, so that we get row for eve
ry user id and columns as Anime names
pivottable = merged_sub_dataframe.pivot_table(index=['user_id'], columns=[
'name'], values='user_rating')

#Checking the shape of the pivot table and viewing the top few rows of the
 pivot table, how it looks like
print(pivottable.shape)
```

```
pivottable.head()

#Now as different users would rate differently thus we need to normalise t
he scores present in the pivot table
pivot_norm = pivottable.apply(lambda x: (x-np.mean(x))/(np.max(x)-
np.min(x)), axis=1)

#Now as seen from the EDA, there are lots of unrated content in the databa
se thus we have to remove it
# Drop all columns containing only zeros representing users who did not ra
te
pivot_norm.fillna(0, inplace=True)
pivot_norm = pivot_norm.T
pivot_norm = pivot_norm.loc[:,(pivot_norm!= 0).any(axis=0)]

pivot_sparse = sp.sparse.csr_matrix(pivot_norm.values)  #there is lot of s
parcity in the matrix
```

IMPLEMENTING ITEM BASED

```
#We are working upon Collaborative Filtering and thus we would try making
the recommendation on item based similarity
item_similarity = cosine_similarity(pivot_sparse)

# Inserting the similarity matricies into dataframe objects So that we can
 perform operations and can further start the main task of recommendation
item_sim_df = pd.DataFrame(item_similarity, index = pivot_norm.index, colu
mns = pivot_norm.index)

#Function to display top 10 similar animes given an anime name
def top_animes(anime_name):
    count = 1
    print('Similar shows to {} include:\n'.format(anime_name))
    for item in item_sim_df.sort_values(by = anime_name, ascending = False
).index[1:11]:
        print('No. {}: {} '.format(count, item))
        count +=1

top_animes('Hajime no Ippo')
```

IMPLEMENTING USER BASED COLLABORATIVE SYSTEM

```
#Now we use cosine similarity function to get user similarity by inputting
 the transpose of our pivot sparse table
user_similarity = cosine_similarity(pivot_sparse.T)
```

```
#Inserting the similarity matricies into dataframe objects So that we can
perform operations and can further start the main task of recommendation
user_sim_df = pd.DataFrame(user_similarity, index = pivot_norm.columns, co
lumns = pivot_norm.columns)

#This function will return the top 5 users with the highest similarity val
ue
# Input : User id
# Output : list of users with highest similarity
def top_users(user):

    if user not in pivot_norm.columns:
        return('No data available on user {}'.format(user))

    print('Most Similar Users with user {}:\n'.format(user))
    sim_items = user_sim_df.sort_values(by=user, ascending=False).loc[:,us
er].tolist()[1:11]
    sim_users = user_sim_df.sort_values(by=user, ascending=False).index[1:
11]
    zipped = zip(sim_users, sim_items,)
    for user, sim in zipped:
        print('User #{0}, Similarity value: {1:.2f}'.format(user, sim))

# This function calculates the weighted average of similar users to determ
ine a potential rating for an input user and show
def predict_rating(anime_name, user):
    sim_users = user_sim_df.sort_values(by=user, ascending=False).index[1:
1000]
    user_values = user_sim_df.sort_values(by=user, ascending=False).loc[:,
user].tolist()[1:1000]
    rating_list = []
    weighted_list = []
    for j, i in enumerate(sim_users):
        rating = pivottable.loc[i, anime_name]
        similarity = user_values[j]
        if np.isnan(rating):
            continue
        elif not np.isnan(rating):
            rating_list.append(rating*similarity)
            weighted_list.append(similarity)
    return sum(rating_list)/sum(weighted_list)
# Function to make list of highest rated animes per simiar users , returns
 the anime and its frequency it appears in sorted list
def animes_per_sim_user_freq(user):
    if user not in pivot_norm.columns:
        return(' Sorry! There is no data available on user  id {}'.format(
user))

    sim_users = user_sim_df.sort_values(by=user, ascending=False).index[1:
11]
    best = []
    most_common = {}

    for i in sim_users:
        max_score = pivot_norm.loc[:, i].max()
        best.append(pivot_norm[pivot_norm.loc[:, i]==max_score].index.toli
st())
    for i in range(len(best)):
        for j in best[i]:
            if j in most_common:
                most_common[j] += 1
            else:
                most_common[j] = 1
    sorted_list = sorted(most_common.items(), key=operator.itemgetter(1),
reverse=True)
    return sorted_list[:5]

top_users(3)
animes_per_sim_user_freq(3)

predict_rating('Hajime no Ippo', 3)
```

Evaluating the System

```
# This function would calculate the coverage for any user present in the d
ataset of ratings, usually coverage is percentage of the animes that he ca
n be given recommendation.
def  coverage (pred) :
  pred = pred+ 0.001
  pred = pred/predicted_sum
  return -np.sum(pred * np.log2(pred))*100

#Now lets examine how the predict_rating function performs compared to the
 observed rated values for user 7

watched = pivottable.T[pivottable.loc[7,:]>0].index.tolist()

#Make a list of the squared errors between actual and predicted value for
user id 7
```

```
errors = []
predicted_sum=0
for i in watched:
    actual=pivottable.loc[7, i]
    predicted = predict_rating(i,7)
    predicted_sum=predicted_sum+predicted
    errors.append((actual-predicted)**2)

#This is the average squared error for user id 7
print("RMSE of user-id 7")
np.mean(errors)

#Calling our coverage function to know what percentage of our anime datase
t can be covered by a user through our recommendation system
coverage(predicted)

#Examples of other user_ids

# Calculating errors and finding RMSE for user id: 3
watched = pivottable.T[pivottable.loc[3,:]>0].index.tolist()
errors = []
predicted_sum=0
for i in watched:
    actual=pivottable.loc[3, i]
    predicted = predict_rating(i,3)
    predicted_sum=predicted_sum+predicted
    errors.append((actual-predicted)**2)
print("RMSE of user-id 3")
np.mean(errors)
#coverage for user ID: 3
coverage(predicted)

# Calculating errors and finding RMSE for user id: 10
watched = pivottable.T[pivottable.loc[10,:]>0].index.tolist()
errors = []
predicted_sum=0
for i in watched:
    actual=pivottable.loc[10, i]
    predicted = predict_rating(i,10)
    predicted_sum=predicted_sum+predicted
    errors.append((actual-predicted)**2)
print("RMSE of user-id 10")
np.mean(errors)
#coverage for user ID: 10
coverage(predicted)
```