Bridging Ratings and Recommendations: Analyzing the Role of User Behavior and Animation Features in Anime Ratings

Jingwen Wang
BIS Major
72405305@cityu-dg.edu.cn

Peishan Jing
BIS Major
72406166@cityu-dg.edu.cn

Sifan An
BIS Major
72404401@cityu-dg.edu.cn

ABSTRACT

In recent years, anime industry has developed rapidly, gaining popularity worldwide. However, research on user behavior and recommendation mechanisms within anime platforms remains limited. To bridge this gap, this study aims to explore how user behavior and anime features influence anime ratings and provide a foundation for personalized recommendation systems. By analyzing more than 200,000 user rating records on the MyAnimeList platform, the study identifies relationships between ratings and factors such as user viewing habits, anime types, and premiere years. Additionally, it examines the impact of user gender, age, and other behavioral traits on rating behavior. Ultimately, based on these findings, a recommendation system was developed to integrate user and animation features, helping users discover high-quality anime that suits their tastes and enhance their viewing experience.

1 INTRODUCTION

Animation, a combination of painting and storytelling, has become a prominent form of entertainment these years. In English, anime refers specifically to animation produced in Japan. Originating in the early 20th century with the works of cartoonist Osamu Tezuka, it has transformed from simple, hand-drawn sequences to complex, computer-generated masterpieces. The animation industry has been rapidly developing and becoming increasingly diverse, attracting audiences worldwide with its unique storytelling and artistic expressions.

As the popularity of anime continues to soar, understanding the factors that influence viewer ratings has become increasingly important for content creators, platforms, and marketers. MyAnimeList (MAL) [4], one of the most prominent online anime communities, provides a rich repository of user-generated data, including ratings and reviews, which offers valuable insights into user preferences and behaviors.

This study aims to bridge the gap between anime ratings and personalized recommendations by analyzing how user behavior and anime features influence viewer preferences. Utilizing the extensive data available on MyAnimeList, we aim to identify the key determinants of anime ratings and explore how these insights can enhance recommendation systems.

2 DATASET DESCRIPTION

This study utilizes the data from MyAnimeList, a globally leading online anime community and database platform, established in 2004. Users can create personal profiles on MAL, track the anime and manga they've watched, rate them, write reviews, and interact with other users. So far, MyAnimeList has cataloged hundreds of thousands of anime and manga titles, and has accumulated data of ratings and reviews from millions of users worldwide. This data not only reflects individual user preferences and behaviors but also provides valuable insights into the popularity and audience reception of various anime works.

The dataset used in this study, updated in November 2023, contains data from the year 1961, with 6 CSV files, over 20 million records and more than 40 variables, named *Anime Data 2023* [3]. In anime-data-2023.csv, most variables reflect animation features, such as Name, Score, and Genres. In users-details-2023.csv, the variables reflect user characteristics and behavior, such as Gender, Birthday, and Days Watched. The users-score-2023.csv records users' rating information for various anime. In this study, we focused on finding the key determinants of anime ratings by analyzing the anime, user, and rating data in the above three CSV files, exploring how these findings can be applied to recommend animations to users. Descriptions of the variables in the dataset can be found in Table 1 to Table 3.

3 PROBLEM DEFINITION

Platforms like MyAnimeList playing a crucial role in bringing together anime enthusiasts from various demographics. Since animation has become a significant cultural and entertainment phenomenon worldwide, understanding how various factors influence animation ratings can provide valuable insights for content creators, platforms, and marketers. This study aims to address 5 key questions regarding these influences by analyzing data from MyAnimeList.

(1) ***What is the overall distribution of anime ratings on MyAnimeList, and what key features does it exhibit? (RQ1)***
(2) ***How do fundamental features of anime, such as genre, premiere year, and number of episodes, correlate with ratings? (RQ2)***
(3) ***How are user ratings distributed on the platform, and are there observable biases or tendencies in user rating behaviors? (RQ3)***
(4) ***How do user characteristics and behaviors, including age, gender and others, influence rating behavior? (RQ4)***
(5) ***How can we effectively recommend high-quality, highly-rated animations to users by integrating both anime and user features? (RQ5)***

The primary objective of this study is to explore the influence of user behavior and anime characteristics on anime ratings, and it also aims to apply these insights to improve recommendation systems. The specific objectives are as follows:

- Analyzing the overall distribution of anime ratings and identifying key characteristics.
- Investigating how various features of anime correlate with user ratings.

Table 1: Variables of User Ratings

Name	Description	Data type
Mal ID	Unique ID for each user.	int64
Username	The username of the user.	object
Gender	The gender of the user.	object
Birthday	The birthday of the user (in ISO format).	object
Location	The location or country of the user.	object
Joined	The date when the user joined the platform (in ISO format).	object
Days Watched	The total number of days the user has spent watching anime.	float64
Mean Score	The average score given by the user to the anime they have watched.	float64
Watching	The number of anime currently being watched by the user.	float64
Completed	The number of anime completed by the user.	float64
On Hold	The number of anime on hold by the user.	float64
Dropped	The number of anime dropped by the user.	float64
Plan to Watch	The number of anime the user plans to watch in the future.	float64
Total Entries	The total number of anime entries in the user's list.	float64
Rewatched	The number of anime rewatched by the user.	float64
Episodes Watched	The total number of episodes watched by the user.	float64

Table 2: Variables of Users

Name	Description	Data type
user id	Unique ID for each user.	int64
Username	The username of the user.	object
anime id	Unique ID for each anime.	int64
Anime Title	The title of the anime.	object
rating	The rating given by the user to the anime.	int64

- Examining the distribution of user ratings to uncover potential biases or tendencies.
- Assessing the impact of user demographics on rating behaviors.
- Developing a framework for personalized recommendation systems that combine anime and user features to enhance viewer satisfaction.

Table 3: Variables of Animations

Name	Description	Data type
anime id	Unique ID for each anime.	int64
Name	The name of the anime in its original language.	object
English name	The English name of the anime.	object
Other name	Native name or title of the anime (can be in Japanese, Chinese or Korean).	object
Score	The score or rating given to the anime.	object
Genres	The genres of the anime, separated by commas.	object
Synopsis	A brief description or summary of the anime's plot.	object
Type	The type of the anime (e.g., TV series, movie, OVA, etc.).	object
Episodes	The number of episodes in the anime.	object
Aired	The dates when the anime was aired.	object
Premiered	The season and year when the anime premiered.	object
Status	The status of the anime (e.g., Finished Airing, Currently Airing, etc.).	object
Producers	The producers of the anime.	object
Licensors	The licensors of the anime (e.g., streaming platforms).	object
Studios	The animation studios that worked on the anime.	object
Source	The source material of the anime (e.g., manga, light novel, original).	object
Duration	The duration of each episode.	object
Rating	The age rating of the anime.	object
Rank	The rank of the anime based on popularity or other criteria.	object
Popularity	The popularity rank of the anime.	int64
Favorites	The number of times the anime was marked as a favorite by users.	int64
Scored By	The number of users who scored the anime.	object
Members	The number of members who have added the anime to their list on the platform.	int64
Image URL	The URL of the anime's image or poster.	object

By addressing these questions, this study aims to provide a comprehensive understanding of the dynamics influencing anime ratings and offer practical recommendations for personalizing user experience on platforms like MyAnimeList.

4 METHODOLOGY

Our goal is to find out the key determinants of anime ratings and explore how these findings can be applied to enhance anime recommendation systems. Research questions in part 3 can be integrated into **two stages** in our study:

(1) *Analysis of the correlation between anime and user features and ratings. (RQ1-RQ4)*
(2) *Construction of an anime recommendation system based on the above analysis. (RQ5)*

In the first stage, we will use multiple Python libraries to preprocess, analyze, and visualize data from the dataset. For preprocessing and data analysis, we used **numpy** and **pandas**; for data visualization, we used **plotly** to create necessary charts, including histograms, scatter plots, KDE plots, etc. The entire process will independently analyze the influence of multiple variables on ratings and evaluated their roles in the anime recommendation process.

In the second stage, we will adopt deep learning methods to build an anime recommendation system. By using **PyTorch** [2], **transformers** (a Python library), and Google's pre-trained language model **BERT** [1], we will be able to calculate the similarity between each user's favorite anime (i.e., those rated highest by the user) and the synopses of 500 high-rated anime. Based on the findings from the first stage, we will filter the results to ultimately select high-rated, high-quality anime that match the users' preference and recommend them to users.

4.1 Data Pre-processing, Variables Analysis, and Visualization

In order to analyze the Anime Data 2023 dataset and explore RQ1 to RQ4, this section requires three steps: data preprocessing, data analysis, and data visualization. Specific implementation methods are introduced below.

Correcting the Data Type.

After checking the data types of the variables in Table 1 to Table 3, it is obvious that some data type are incorrect. For instance, in Table 3, *Score*, *Score By*, *Rank*, and *Episodes* are object variables, leading to trouble in conducting data analysis. Type of these variables should be converted into numerical.

```
df_anime['Score'] = df_anime['Score'].astype(np.float64
    )
df_anime['ScoredBy'] = df_anime['ScoredBy'].astype(np.
    float64)
df_anime['Rank'] = df_anime['Rank'].astype(np.float64)
df_anime['Episodes'] = df_anime['Episodes'].astype(np.
    float64)
```

Handling the Missing Values, Useless Values, and Outliers.

Apart from data types, this dataset also contains a large number of missing values, useless values, and outliers that need to be addressed. For example, in the *anime-dataset-2023.csv*, the **Score**, **Rank**, and **Episodes** columns contain a large number of UNKNOWN values. We dropped rows where the **Score** column is UNKNOWN. For the other two csv files, we changed them to null values. This is because Score is the target variable, and if it is null, it cannot be included in the analysis at all. However, directly dropping Rank and Episodes may result in the loss of many excellent anime entries.

```
df_anime = df_anime[df_anime['Score'] != 'UNKNOWN']
df_anime['Rank'] = df_anime['Rank'].replace('UNKNOWN',
    np.nan)
df_anime['Episodes'] = df_anime['Episodes'].replace('
    UNKNOWN', np.nan)
```

In the *users-details-2023.csv*, more than 50% of users only registered one account and have never finished watching any anime, nor have they given any ratings. This part of data is useless and needs to be dropped.

```
drop_id = ((df_user['DaysWatched']==0) & (df_user['Mean
    Score']==0) & (df_user['Watching']==0) & (df_user[
    'Completed']==0) & (df_user['OnHold']==0) & (
    df_user['Dropped']==0) & (df_user['PlantoWatch'
    ]==0) & (df_user['TotalEntries']==0) & (df_user['
    Rewatched']==0))
df_user = df_user[~drop_id]
```

For other variables in the dataset that need adjustment, we also adopted a similar approach.

Calculating the Variables That are Not in the Dataset.

This dataset does not contains all the variables we need. For instance, in the *users-details-2023.csv*, there is only a **Birthday** column that contains users' birthday in the form of **ISO 8601**. We have to to calculate the age of the users:

```
df_user_age = df_user.dropna(subset=['Birthday', 'Mean
    Score'])
df_user_age['Birthday'] = pd.to_datetime(df_user_age['
    Birthday'], errors='coerce')
current_year = 2023
df_user_age['Age'] = current_year - df_user_age['
    Birthday'].dt.year
```

After that, we will set multiple age groups to conduct the analysis of rating tendencies among users of different age groups:

```
age_slice = [0, 18, 25, 35, 45, 55, 65, 120]
labels = ['0-18', '18-25', '25-35', '35-45', '45-55', '
    55-65', '65+']
df_user_age['AgeGroup'] = pd.cut(df_user_age['Age'],
    age_slice, labels=labels, right=False)
```

Data Analysis and Visualization.

We used **plotly** to create charts that illustrates the relationship between anime ratings and other variables, including histograms, scatter plots, KDE plots, violin plots, etc. These charts provide a clear visual insight into the association between anime ratings and both anime features and user characteristics.

4.2 Construction of Anime Recommendation System

After analyzing multiple variables related to the characteristics of anime and users, we found that some variables are strongly correlated with anime rating. We will use these variables to select animations by using BERT model's text similarity comparison of the **Synopsis** column. This approach make sure that the recommended animations have a high similarity to the user's favorite ones while significantly enhancing the quality and popularity of the recommended animations.

Extra and Necessary Data Pre-processing.

In order to ensure the quality of recommendations, the system must pre-process the animation data to filter out eligible animations.

```
df_filtered = df_anime[df_anime['Type'].isin(['TV', '
     Movie', 'OVA'])]
df_filtered = df_filtered[df_filtered['Duration'].apply
     (lambda x: 'hr' not in x)]
def extract_year(aired_date):
    try:
        return int(aired_date.split('')[-1])
    except ValueError:
        return np.nan
df_filtered['Year'] = df_filtered['Aired'].apply(
     extract_year)
df_filtered = df_filtered.dropna(subset=['Year'])
df_filtered = df_filtered[df_filtered['Year'] >= 2000]
df_sorted = df_filtered.sort_values(by='Score',
     ascending=False)
df_anime_500 = df_sorted.head(500)
```

For those data in *users-details-2023.csv* and *users-score-2023.csv*, we will conduct similar steps to ensure it usable for our system design.

Analyzing Synopsis Text Similarity Using BERT.

we will use the BERT pre-trained language model to analyze the synopsis text of the top 3 highest-rated anime from the target user's history. Using this method, 15 animations with the highest similarity are selected and ranked according to their ratings. The top 5 animations will be recommended to the user.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-
     uncased')
model = BertModel.from_pretrained('bert-base-uncased')

device = torch.device('cuda' if torch.cuda.is_available
     () else 'cpu')
model.to(device)

def get_embedding(text):
    inputs = tokenizer(text, return_tensors='pt',
         truncation=True, padding=True, max_length=512)
    inputs = {key: value.to(device) for key, value in
         inputs.items()}
    with torch.no_grad():
        outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).
         squeeze().cpu()

df_anime_500 = df_anime_500.copy()
df_anime_500['embedding'] = df_anime_500['Synopsis'].
     apply(get_embedding)
```

Filter the Results Using Other Variables.

Using the analysis results from RQ1-RQ4, we will add corresponding parameters in the process of filtering the results from BERT to make it more representative and better meet user needs. The details will be described in Section 5 of this paper.

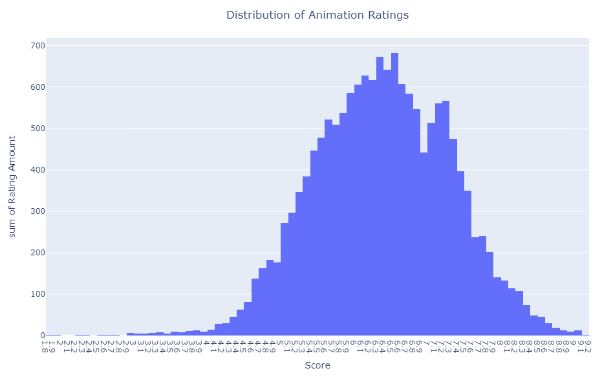


Figure 1: The distribution of animation ratings. The horizontal axis represents the ratings, and the vertical axis represents the number of animations that received each rating.

5 RESULTS AND ANALYSIS

This section presents an analysis of the distribution of user ratings, anime types, genres, and more, followed by the construction of an enhanced anime recommendation system designed to provide users with high-quality, highly-rated animations that they may enjoy.

5.1 Results and Analysis: Distribution of Anime Ratings and Its Features (RQ1)

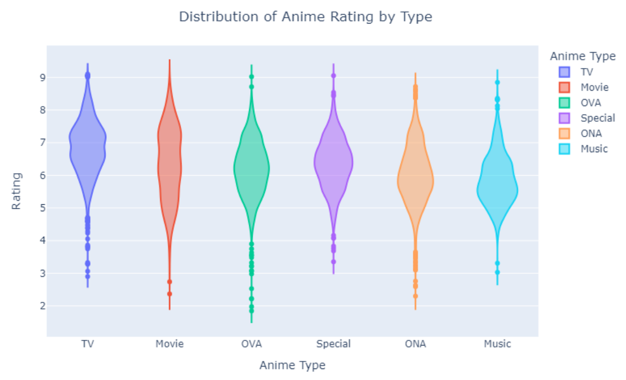
As demonstrated in Fig. 1, this histogram shows the distribution of anime ratings, displaying an approximately normal distribution shape. Most ratings are concentrated in range 5 to 8 , forming a distinct peak, while ratings on both sides gradually decrease and approach zero. It demonstrates the symmetry and width of the distribution, reflecting that the anime ratings are generally concentrated, most of which are mid-to-high ratings.

The result also infers that scoring above 8 can be considered high-rated. Also, the number of these animations are relatively small. Animations scoring above 9 are even rarer and are typically considered exceptionally high-rated.

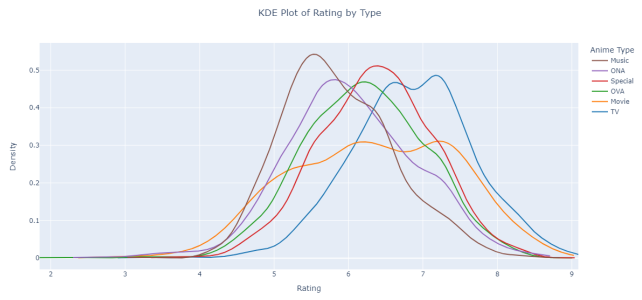
Based on this result, we set the following evaluation standard in RQ5: if the score of recommended animations are all above 8, this system is considered successful.

5.2 Results and Analysis: The Relationship between Features of Anime and Its Ratings (RQ2)

Types and Ratings. Fig. 2a and Fig. 2b reveals the rating distribution for different types of anime (TV, Movie, OVA, Special, ONA,



(a) The distribution of animation rating by type.



(b) The KDE plot of anime rating by type.

Figure 2: The Distribution of animation ratings by type. The Fig. 2a and Fig. 2b reflect the relationship between animation ratings and their different types.

Music). It is evident that TV and Movie types generally have higher ratings, with significantly more animations scoring above 8. Based on this result, our recommendation system should focus on recommending these two types of anime.

Duration and Ratings. Fig. 3a illustrates the relationship characteristics between anime ratings and duration. Although there is no significant positive or negative correlation trend between the two, it can be observed that most high-rated animations are relatively short. Overall, most animations have ratings concentrated between 5 and 8, with durations concentrated between 0 and 100 minutes. The highest density is found in the region where ratings are between 7 and 8 and the duration is shorter. Based on this, our recommendation system will be more willing to suggest shorter animations to users.

Episodes and Ratings. Fig. 3b shows the correlation between anime ratings and the number of episodes. Most ratings range from 4 to 8, with episode counts between 0 and 300. The number of high-rated anime (above 8) is small, and they generally do not have episodes exceeding 300. A few animations have more than 1000 episodes, but not all of their ratings are high. Therefore, there is no clear linear correlation between the number of episodes and ratings.

Favorites and Ratings. Fig. 3c indicates the significant relationship between anime ratings and the number of user favorites: animations with higher ratings are more likely to receive a large number of user favorites, especially those with ratings above 8. Animations with lower ratings generally have fewer favorites, demonstrating a positive correlation. Based on this result, our recommendation system will prioritize anime with large number of favorites.

Popularity and Ratings. Fig. 3d clearly illustrates the strong association between anime ratings and popularity. Highly-rated ones (especially those with scores above 7) are generally more popular (with lower popularity values), while lower-rated animations tend to be less popular (with higher popularity values). Animations with ratings between 6 and 8 are the most numerous and generally very popular. This indicates that audiences have a higher level of recognition and appreciation for highly-rated anime. Based on this, it is necessary for our recommendation system to take popularity into consideration when suggesting anime to users.

Premiered and Ratings. Fig. 4 indicates that since 1960, the average rating of animations has generally increased gradually, with a notable improvement in quality from the late 1980s to the mid-1990s. However, the ratings have also fluctuated year by year. In recent years, the average ratings have tended to be higher, especially in 2023, which reached a new high on the chart, demonstrating that the quality of anime works tend to be better in the new century. Based on this, our recommendation system will suggest more animations premiered in 21st century to users.

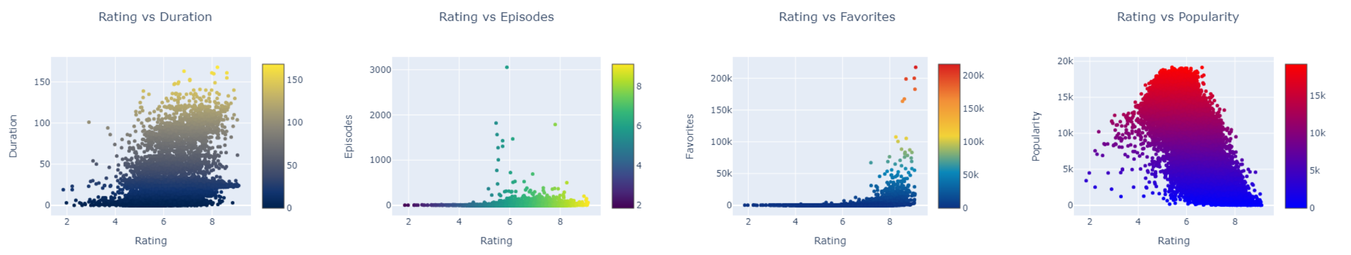
5.3 Results and Analysis: Distribution of Users' Mean Ratings and Biases or Tendencies (RQ3)

As is demonstrated in the Fig. 5, the shape of the histogram resembles a right-skewed bell-shaped distribution, indicating that the mean of user ratings is relatively concentrated in the medium to high score range. The mean of user ratings is mainly concentrated between 5.0 and 10.0, with the highest concentration around 8.0 and 9.0, where the bars are particularly tall, indicating that most user ratings are approximately within this range.

This result further confirms the conclusion in RQ1 that animations with ratings above 8 can be considered high-scoring, and the evaluation standard proposed in RQ1 is also reasonable.

5.4 Results and Analysis: The Relationship between User Characteristics and Behaviors and Their Mean Ratings (RQ4)

Gender and Ratings. Fig. 6a reflects the distribution of anime ratings among users of different genders. Although overall ratings are concentrated between 6 and 9, female users are more likely to give high scores, with a significant number of 9 10. Male users' ratings are mostly around 8, generally lower than females'. Non-binary users have the lowest overall ratings, peaking between 6 and 8. This indicates some differences in rating behavior and preferences among different genders, but they are not significant enough to be considered in recommendation systems.



(a) The relationship between ratings and durations. **(b) The relationship between ratings and episodes.** **(c) The relationship between ratings and favorites.** **(d) The relationship between ratings and popularity.**

Figure 3: Analysis of the relationship between anime ratings and numerical variables. The variables involved are: Duration, Eposides, Favorites, and Popularity.

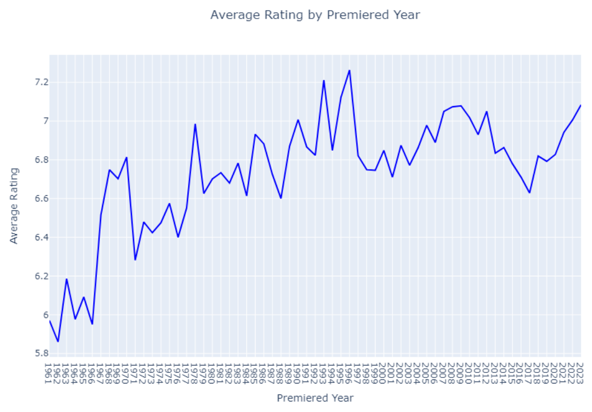


Figure 4: The average rating of animes per year.

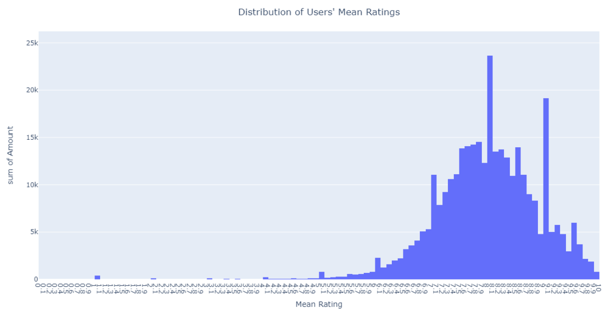
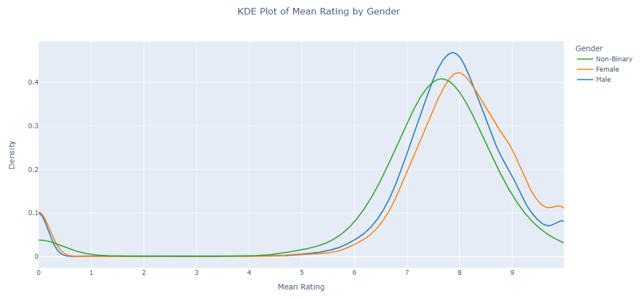
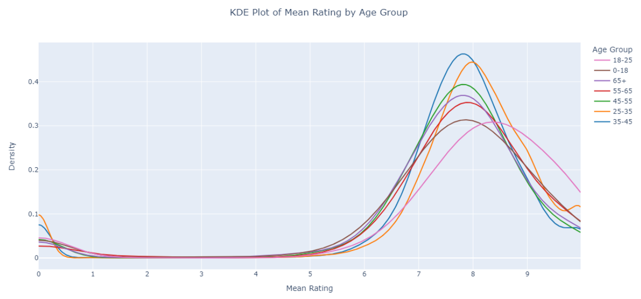


Figure 5: The distribution of Users' mean ratings.



(a) The KDE plot of users' mean ratings by gender.

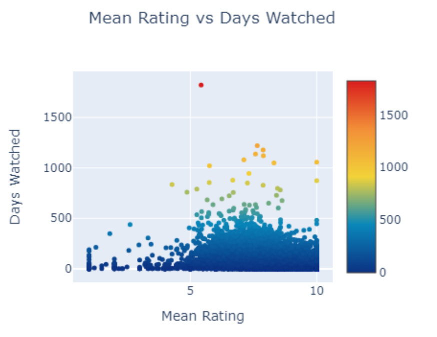


(b) The KDE plot of users' mean ratings by age group.

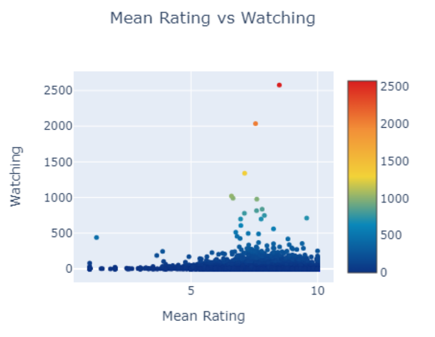
Figure 6: The KDE plot of users' mean ratings by gender and age group. The Fig. 6a and Fig. 6b reflect the relationship between users' mean ratings and their characteristics, including gender and age.

Age and Ratings. Fig. 6b reflects the distribution characteristics of animation ratings among different age groups. The peak for most groups is around 7.8, while the 18-25 age group peaks at 8.4, the highest among all age groups. This group also gives most of the 9-10 scores. It indicates that anime generally receives high ratings from users of all ages, particularly appealing to younger audiences aged 18-25.

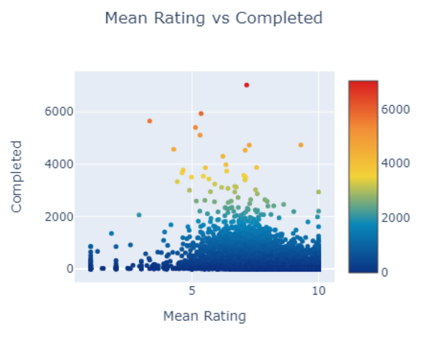
Days Watched and Ratings. Fig. 7a indicates a positive correlation between users' mean ratings of anime and the number of days they have watched: users who watched for more days tend to have higher or more stable average ratings. This is because after a long time of watching, they may have a better understanding of the whole content and quality, leading to more fair ratings. Those who didn't spend much time watching might give a lower score according to the not very attractive beginning.



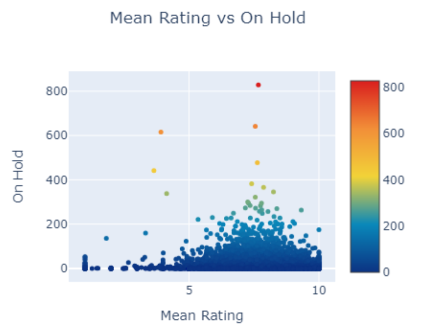
(a) The relationship between mean rating and days watched.



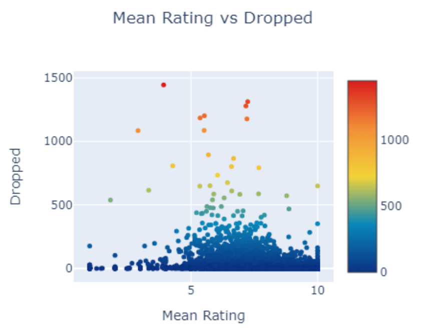
(b) The relationship between mean rating and watching.



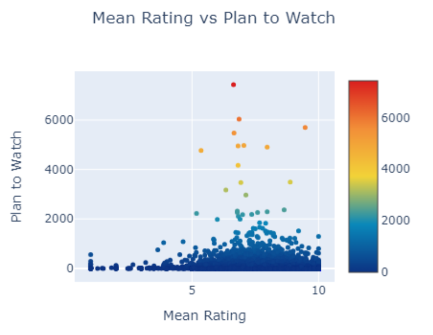
(c) The relationship between mean rating and completed.



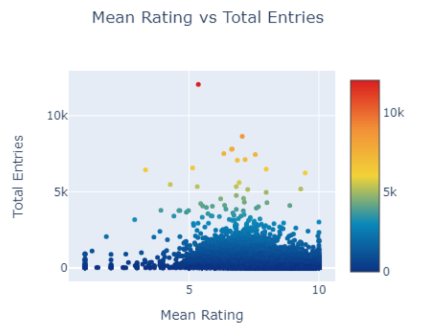
(d) The relationship between mean rating and on hold.



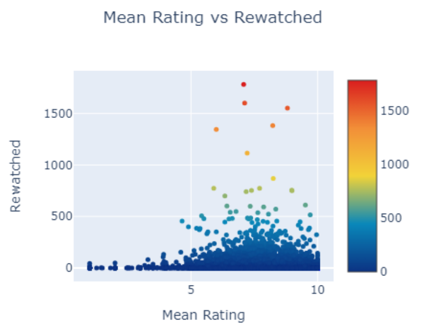
(e) The relationship between mean rating and dropped.



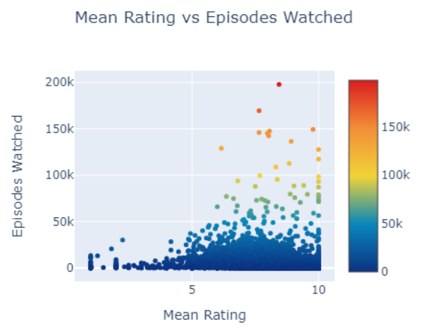
(f) The relationship between mean rating and plan to watch.



(g) The relationship between mean rating and total entries.



(h) The relationship between mean rating and rewatched.



(i) The relationship between mean rating and episodes watched.

Figure 7: Analysis of the relationship between anime ratings and numerical variables of user behaviors. The variables involved are: Days Watched, Watching, Completed, On Hold, Dropped, Plan to Watch, Total Entries, Rewatched and Episodes Watched.

Watching and Ratings. Fig. 7b illustrates a positive correlation between the number of animations currently being watched and the mean ratings given by users. Users giving high-rating (above 7 points) are often more willing to try and continue watching more anime, while low-rating users might be less satisfied and thus avoid watching more. User satisfaction (high ratings) directly influences their interest and watching behavior. This indicates that for high-rated animations, users are more likely to continue watching and then expand their watching list.

Completed and Ratings. Fig. 7c also shows the positive relationship between mean ratings and the number of completed views. When users have higher mean ratings (greater than 7), they tend to finish watching more, reflecting higher satisfaction and interest in the anime. On the other hand, users with lower average ratings (less than 5) generally watched fewer animations in full, inferring dissatisfaction with the quality or a lack of interest.

On Hold and Ratings. Fig. 7d shows a negative correlation between mean ratings and the number of shelved animations. When users have higher mean ratings (greater than 8), the number of shelved anime is relatively low, reflecting higher satisfaction and a continuous viewing. For users who gave lower mean ratings (less than 8), there are more animations on hold, indicating they are less satisfied with or interested in the anime they watched.

Dropped and Ratings. Fig. 7e primarily shows the negative correlation between users' mean ratings and the number of animations they abandoned: users with lower mean ratings (below 7) tend to drop more, while those with higher average ratings dropped fewer, indicating higher satisfaction with the anime they watch.

Plan to Watch and Ratings. Fig. 7f shows that when users have a higher average rating, they also plan to watch more, reflecting their interest and confidence in anime. But for users with lower mean ratings who are not very satisfied with the anime they have watched, they still plan to watch a lot. Overall, there is no significant correlation between ratings and the number of anime planned to be watched.

Total Entries and Ratings. Fig. 7g demonstrates the relationship between mean ratings and the total number of followed series. Users who followed a large amount of anime series tend to give higher mean ratings, reflecting that they continued exploration based on high satisfaction. For users in the mid-range (5-7 points), the total number of followed series is relatively higher, showing they are trying to find more preferred works. Users with low ratings follow fewer series, showing their overall lower satisfaction with the anime they watched.

Re-watched and Ratings. Fig. 7h shows a certain degree of positive correlation between the number of times an anime is re-watched by a user and his mean rating. Users with higher mean ratings (above 7) are more likely to re-watch animations, while those with lower ratings tend to re-watch less. The higher the user's satisfaction with the anime (higher rating), the more likely they are to re-watch these works, indicating that high-rated animations usually have greater re-watch value.

Episodes Watched and Ratings. In Fig. 7i, the scatter plots in the image are generally concentrated on the right side, indicating a clear positive correlation between episodes watched and the mean rating given by users. Users who give higher mean ratings typically watch more episodes, while those with lower ratings tend to watch fewer episodes. This suggests that high-rated anime have a higher watching value, and users are willing to invest more time in them.

5.5 Results and Analysis: Construction of A Personalized Recommendation System Combining Anime and User Features (RQ5)

According to the results regarding certain variables in sections 5.2 and 5.4, we decided to take the user's age, gender, as well as the anime's rating, popularity, and number of favorites into account . If the users has not provided their age and gender information, these two variables will not be considered. Based on rating, popularity, and the number of favorites of the anime, we designed the following formula to calculate the weighted score:

$$\begin{aligned}\textbf{Weighted Score} = \textbf{Rating} \times \textbf{0.5} \\ + \frac{\textbf{Favorites}}{\textbf{1000}} \times \textbf{0.3} \\ + \frac{\textbf{50}}{\textbf{Popularity}} \times \textbf{0.2}\end{aligned} \tag{1}$$

The weight of Rating is 50%, because it is considered the most important factor. The weight of Favorites is 30%. For Popularity, we plan to focus on recommending the top 50 most popular animations, thus subtracting 50 from it and assigning a 20% weight. A sample output is as follows:

```
The 3 highest-rated anime by users:
    anime_id           Anime Title rating
9      11061  Hunter x Hunter (2011)   10
10      9253           Steins;Gate    10
11       121    Fullmetal Alchemist   10

Recommended anime:
                                 Name  Score Popularity
3961  Fullmetal Alchemist: Brotherhood 9.10        3
5667                      Steins;Gate  9.07       13
1822        Tengen Toppa Gurren Lagann 8.63       65
833                           Gintama  8.94      138
100               Fullmetal Alchemist 8.11        76

      Favorites      similarity    score_weighted
3961    217606        0.829451      73.165133
5667    182964        0.904289      60.193431
1822    75870         0.825382      27.229846
833     57678         0.838205      21.845864
100     25502         0.838971      11.837179
```

The system will display the top 3 highest-rated animations first . Next, it calculates the similarity between the synopses of these animations and those in the database, while also considering user factors such as age, gender, ratings, number of favorites, and popularity. After that, the system automatically selects the most suitable recommended anime. Analysis of the output reveals that the recommended animations exhibit a similarity score greater than 0.80 with the input selections, generally have ratings above 8, and demonstrate high levels of favorites and popularity. These results indicate that the system is effective.

The code of recommended function of this system is attached in the end of this paper.

6 CONCLUSION

This study examined the relationship between user characteristics and behavior and anime features with anime ratings, including factors such as the user's gender, age, and the anime's type and popularity. We found that user behaviors and variables, including the anime's popularity and number of favorites, are significantly correlated with ratings. These insights are valuable for industry professionals. Furthermore, we developed an anime recommendation system that integrates both user and anime characteristics, helping users discover high-quality content that aligns with their preferences and enhancing their overall viewing experience. Future research could incorporate additional user behavior variables or employ advanced deep learning techniques to further optimize the system.

REFERENCES

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (Minneapolis, Minnesota) *(NAACL-HLT '19)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[2] Adam Paszke, Sam Gross, Francisco Massa, and et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv preprint arXiv:1912.01703* (2019). https://arxiv.org/abs/1912.01703

[3] SAJID. 2023. Anime Dataset 2023. https://www.kaggle.com/datasets/dbdmobile/myanimelist-dataset Accessed: 2024-10-29.

[4] Staff of MyAnimeList. 2024. MyAnimeList. https://myanimelist.net Accessed: 2024-10-29.

data_exploration_and_RQ1-3

October 30, 2024

0.1 Import libraries

```
[ ]: import numpy as np
     import pandas as pd
     import plotly.express as px
     import plotly.graph_objects as go
     import plotly.figure_factory as ff
```

0.2 Read the dataset

```
[ ]: df_anime = pd.read_csv('data/anime-dataset-2023.csv')
     print(df_anime.shape)
     df_anime.head()
```

```
[ ]: df_user = pd.read_csv('data/users-details-2023.csv')
     print(df_user.shape)
     df_user.head()
```

```
[ ]: df_score = pd.read_csv('data/users-score-2023.csv')
     print(df_score.shape)
     df_score.head()
```

0.3 Explore the data

```
[ ]: # check the information of df_anime
     df_anime.info()
```

```
[ ]: # check the amount of values in Score column
     df_anime['Score'].value_counts()
```

0.3.1 Many rows in the Score column have the value UNKNOWN. These rows are useless and need to be dropped.

```
[ ]: df_anime = df_anime[df_anime['Score'] != 'UNKNOWN'] # drop the UNKNOWN rows
     df_anime['Score'].value_counts()
```

```
[ ]: df_anime.info()
[ ]: # check the amount of values in Rank column
     df_anime['Rank'].value_counts()
```

0.3.2 The Rank column has 2991 rows marked as UNKNOWN. These can be changed to null values instead of being dropped, as some important animation data might be lost.

```
[ ]: df_anime['Rank'] = df_anime['Rank'].replace('UNKNOWN', np.nan) # change the␣
      ↪value to nan
     df_anime['Rank'].value_counts()
```

```
[ ]: # check the amount of values in Episodes column
     df_anime['Episodes'].value_counts()
```

0.3.3 The Episodes column has 88 rows marked as UNKNOWN. These should be changed to null values as well.

```
[ ]: df_anime['Episodes'] = df_anime['Episodes'].replace('UNKNOWN', np.nan) # #␣
      ↪change the value to nan
```

```
[ ]: df_anime.info()
```

0.3.4 Score,Scored By,Rank,Episodes are object type.They need to be converted to numerical data type.

```
[ ]: # convert object to float64
     df_anime['Score'] = df_anime['Score'].astype(np.float64)
     df_anime['Scored By'] = df_anime['Scored By'].astype(np.float64)
     df_anime['Rank'] = df_anime['Rank'].astype(np.float64)
     df_anime['Episodes'] = df_anime['Episodes'].astype(np.float64)
```

```
[ ]: df_anime.info()
```

```
[ ]: # check the amount of values in Duration column
     df_anime['Duration'].value_counts()
```

0.3.5 The Duration column is not purely numerical and needs to be converted entirely to numerical values, with the unit being minutes.

```
[ ]: import re # it is the support for regular expressions

     # convert duration to minutes
     def convert_duration_to_minutes(duration):
         # use regular expressions to match hours and minutes
         hr_pattern = re.compile(r'(\d+)\s*hr')
         min_pattern = re.compile(r'(\d+)\s*min')
         # find hours and minutes in the Duration column
```

```
    hours = hr_pattern.findall(duration)
    minutes = min_pattern.findall(duration)
    # convert hours and minutes into integers, and convert hours into minutes
    hours = int(hours[0]) * 60 if hours else 0
    minutes = int(minutes[0]) if minutes else 0

    return hours + minutes # return the total minutes
```

[ ]:
```
df_anime['Duration'] = df_anime['Duration'].apply(convert_duration_to_minutes) #␣
↪apply the above function to Duration column
df_anime.head()
```

[ ]:
```
df_anime.info()
```

0.3.6 What is the overall distribution of anime ratings on MyAnimeList, and what key features does it exhibit? (RQ1)

[ ]:
```
# extract the Score column and count it, then merge it into a new dataframe
df_score_counts = df_anime['Score'].value_counts().reset_index()
df_score_counts.columns = ['Score', 'Count']
df_score_counts.head()
```

[ ]:
```
# check the distribution of Score (Animation Ratings)
fig = px.histogram(
    df_score_counts,
    x='Score',
    y='Count',
    labels={'Ratings': 'Ratings', 'Count': 'Rating Amount'}
)
fig.update_layout(
    title={'text': 'Distribution of Animation Ratings', 'x': 0.5}, # set the␣
↪title and make it in the center
    width=1000,
    height=600,
    showlegend=False # hide the legend
)
fig.update_xaxes(dtick=0.1) # set the x-axis interval to 0.1
fig.update_traces(xbins=dict(size=0.1)) # set the width of bars to 0.1
fig.show()
```

(1) explore the relationship between Type and Ratings

[ ]:
```
# check the amount of values in Type column
df_anime['Type'].value_counts()
```

[ ]:
```
df_anime = df_anime[df_anime['Type'] != 'UNKNOWN'] # drop the UNKNOWN column
```

```
[ ]: # draw the violin picture for Type and Score (Rating)
     fig = px.violin(
         df_anime,
         x='Type',
         y='Score',
         labels={'Type':'Anime Type', 'Score':'Rating'},
         color='Type'
     )
     fig.update_layout(
         title={'text': 'Distribution of Anime Rating by Type', 'x': 0.5}, # set the␣
      ↪title and make it in the center
         width=800,
         height=500,
         showlegend=True # show the legend (in the right part of the picture)
     )
     fig.show()
```

```
[ ]: # handle NaNs and infinite values
     df_anime = df_anime.replace([np.inf, -np.inf], np.nan)
     df_anime_num_cat1 = df_anime.dropna(subset=['Score', 'Type'])
     # filter out 'Type' categories with insufficient data
     min_elements = 2  # Set the minimum number of elements required in each group
     filtered_types = df_anime_num_cat1.groupby('Type').filter(lambda x: len(x) >=␣
      ↪min_elements)
     # prepare data for KDE plot
     types = filtered_types['Type'].unique()
     hist_data = [filtered_types[filtered_types['Type'] == t]['Score'].values for t␣
      ↪in types]
     # check if there is sufficient data to plot
     if all(len(data) > 1 for data in hist_data):
         # create KDE plot
         fig = ff.create_distplot(hist_data, types, show_hist=False, show_rug=False)

         fig.update_layout(
             title={'text': "KDE Plot of Rating by Type", "x": 0.5}, # set the title␣
      ↪and make it in the center
             xaxis_title="Rating",
             yaxis_title="Density",
             legend_title="Anime Type",
             width=1200,
             height=600
         )

         fig.show()
     else:
         print("No enough data to create a KDE plot for each Type.")
```

(2) exploring the relationship between the number of episodes, episode duration, popularity, number of favorites, and ratings of an animation

[ ]:
```
# Score vs Episodes
fig1 = go.Figure()
fig1.add_trace(
    go.Scatter(
        x=df_anime['Score'],
        y=df_anime['Episodes'],
        mode='markers',
        name='Rating vs Episodes',
        marker=dict(
            size=5,  # set the size of the point
            color=df_anime['Score'],  # set the color of the point using value␣
 ↪of Score column
            colorscale='Viridis',  # set the color mapping
            showscale=True  # show the color bar
        )
    )
)
fig1.update_layout(
    title={'text': 'Rating vs Episodes', 'x': 0.5},
    xaxis_title='Rating',
    yaxis_title='Episodes',
    width=450,
    height=400,
    showlegend=False
)
fig1.show()
```

[ ]:
```
# Score vs Duration
fig2 = go.Figure()
fig2.add_trace(
    go.Scatter(
        x=df_anime['Score'],
        y=df_anime['Duration'],
        mode='markers',
        name='Rating vs Duration',
        marker=dict(
            size=5,
            color=df_anime['Duration'],
            colorscale='Cividis',
            showscale=True
        )
    )
)
fig2.update_layout(
    title={'text': 'Rating vs Duration', 'x': 0.5},
    xaxis_title='Rating',
    yaxis_title='Duration',
    width=450,
    height=400,
    showlegend=False
)
fig2.show()
```

[ ]:
```
# Score vs Popularity
fig3 = go.Figure()
fig3.add_trace(
    go.Scatter(
        x=df_anime['Score'],
        y=df_anime['Popularity'],
        mode='markers',
        name='Rating vs Popularity',
        marker=dict(
            size=5,
            color=df_anime['Popularity'],
            colorscale='Bluered',
            showscale=True
        )
    )
)
fig3.update_layout(
    title={'text': 'Rating vs Popularity', 'x': 0.5},
    xaxis_title='Rating',
    yaxis_title='Popularity',
    width=450,
    height=400,
    showlegend=False
)
fig3.show()
```

[ ]:
```
# Score vs Favorites
fig4 = go.Figure()
fig4.add_trace(
    go.Scatter(
        x=df_anime['Score'],
        y=df_anime['Favorites'],
        mode='markers',
        name='Rating vs Favorites',
        marker=dict(
            size=5,
            color=df_anime['Favorites'],
            colorscale='Portland',
            showscale=True
        )
    )
)
fig4.update_layout(
    title={'text': 'Rating vs Favorites', 'x': 0.5},
    xaxis_title='Rating',
    yaxis_title='Favorites',
    width=450,
    height=400,
    showlegend=False
)
fig4.show()
```

(3) explore the relationship between the premiere year and ratings

[ ]:
```
# crate a new dataframe to process the data about premiered year
df_premiered = pd.DataFrame(df_anime)
# extract the year and drop UNKNOWN rows
df_premiered['Year'] = df_premiered['Premiered'].apply(lambda x: x.split()[-1]␣
 ↪if x != 'UNKNOWN' else None)
# filter the null values
df_premiered = df_premiered.dropna(subset=['Year'])
# convert the data type to int64
df_premiered['Year'] = df_premiered['Year'].astype(np.int64)

# create the picture
fig = px.scatter(
    df_premiered,
    x='Year',
    y='Score',
    labels={'Year': 'Premiered Year', 'Score': 'Score'},
    title='Premiered Year vs Score'
)

fig.update_layout(
    title={'text': 'Premiered Year vs Score', 'x': 0.5}
)

fig.update_xaxes(dtick=1) # set the interval of x-axis to 1 year

fig.show()
```

[ ]:
```
# calculate the average score for each year
average_scores = df_premiered.groupby('Year')['Score'].mean().reset_index()

# create line chart
fig = px.line(
    average_scores,
    x='Year',
    y='Score',
    labels={'Year': 'Premiered Year', 'Score': 'Average Rating'},
    title='Average Score by Premiered Year'
)

fig.update_layout(
    title={'text': 'Average Rating by Premiered Year', 'x': 0.5},
    width=900,
    height=600,
)

fig.update_xaxes(dtick=1)

fig.update_traces(line=dict(color='blue')) # set the color of the line

fig.show()
```

(4) explore the differences in ratings for animations of various genres

```
[ ]: # process NaN and outliers
df_anime = df_anime.replace([np.inf, -np.inf], np.nan)
df_anime_num_cat2 = df_anime.dropna(subset=['Score', 'Genres'])
# split the Genres column
df_anime_num_cat2 = df_anime_num_cat1.assign(Genres=df_anime_num_cat2['Genres'].
 ↪str.split(', ')).explode('Genres')
# filter out categories with insufficient samples
min_elements = 2  # eet the minimum number of samples for each category
filtered_genres = df_anime_num_cat2.groupby('Genres').filter(lambda x: len(x) >=␣
 ↪min_elements)
# prepare data for KDE chart
genres = filtered_genres['Genres'].unique()
hist_data = [filtered_genres[filtered_genres['Genres'] == genre]['Score'].values␣
 ↪for genre in genres]
# check if there is enough data to create a chart
if all(len(data) > 1 for data in hist_data):
    # create KDE chart
    fig = ff.create_distplot(hist_data, genres, show_hist=False, show_rug=False)

    fig.update_layout(
        title={'text': "KDE Plot of 'Score' by 'Genres'", "x": 0.5},
        xaxis_title="Score",
        yaxis_title="Density",
        legend_title="Genres",
        width=1200,
        height=700,
    )
    fig.show()
else:
    print("Not enough data to create a KDE plot for each 'Genres'.")
```

0.4 Clean up user-details-2023

```
[ ]: df_user.info()
```

```
[ ]: df_user_num_list = df_user.select_dtypes(include=['number']).columns.tolist() #␣
     ↪numerical data
     df_user_cat_list = df_user.select_dtypes(include=['object', 'category']).columns.
     ↪tolist() # object data
     print(df_user_num_list)
     print(df_user_cat_list)
```

```
[ ]: df_user.isnull().sum() / df_user.shape[0] * 100 # check the amount of null values
```

0.4.1 Over 69% of the fields for Gender, Birthday, and Location are empty.

```
[ ]: df_user.describe() # check the data distribution of df_user
```

0.4.2 It is shocking that over 50% of the accounts are useless; they are merely registered and have not fully watched any anime. These entries must be cleared.

```
[ ]: # remove useless accounts
     drop_id = ((df_user['Days Watched']==0) & (df_user['Mean Score']==0) &␣
     ↪(df_user['Watching']==0) & (df_user['Completed']==0)
                & (df_user['On Hold']==0) & (df_user['Dropped']==0) & (df_user['Plan␣
     ↪to Watch']==0) & (df_user['Total Entries']==0)
                & (df_user['Rewatched']==0))
     df_user = df_user[~drop_id]
     df_user.shape
```

```
[ ]: df_user.describe()
```

```
[ ]: df_user.info()
```

0.5 How do fundamental features of anime, such as genre, premiere year, and number of episodes, correlate with ratings? (RQ2)

```
[ ]: # extract the Mean Score of users and count them
     df_user_mean_score_counts = df_user['Mean Score'].value_counts().reset_index()
     df_user_mean_score_counts.columns = ['Mean Score', 'Count']
     df_user_mean_score_counts =␣
     ↪df_user_mean_score_counts[df_user_mean_score_counts['Mean Score'] != 0] # drop␣
     ↪0 rows
```

```
df_user_mean_score_counts.head()
```

[ ]:
```
# check the distribution of Mean Score
fig = px.histogram(
    df_user_mean_score_counts,
    x='Mean Score',
    y='Count',
    labels={'Mean Score': 'Mean Rating', 'Count': 'Amount'}
)
fig.update_layout(
    title={'text': 'Distribution of Users\' Mean Ratings', 'x': 0.5},
    width=1200,
    height=600,
    showlegend=False # hide the legend
)
fig.update_xaxes(range=[0, 10], dtick=0.1) # set the range of x-axis as 0 to 10,␣
↪and the interval to 0.1
fig.update_traces(xbins=dict(size=0.1)) # set the width of bars to 0.1
fig.show()
```

0.6 How are user ratings distributed on the platform, and are there observable biases or tendencies in user rating behaviors? (RQ3)

[ ]:
```
# Mean Score and gender
df_user_gender_score = df_user.dropna(subset=['Mean Score', 'Gender'])
# filter out 'Gender' categories with insufficient data
min_elements = 2  # set the minimum number of elements required in each group
filtered_genders = df_user_gender_score.groupby('Gender').filter(lambda x:␣
↪len(x) >= min_elements)
# prepare data for KDE plot
genders = filtered_genders['Gender'].unique()
hist_data = [filtered_genders[filtered_genders['Gender'] == t]['Mean Score'].
↪values for t in genders]
# check if there is sufficient data to plot
if all(len(data) > 1 for data in hist_data):
    # create KDE plot
    fig = ff.create_distplot(hist_data, genders, show_hist=False,␣
↪show_rug=False) # hide the bar and rug plot

    fig.update_layout(
        title={'text': "KDE Plot of Mean Rating by Gender", "x": 0.5},
        xaxis_title="Mean Rating",
        yaxis_title="Density",
        legend_title="Gender",
        width=1200,
        height=600
    )
    fig.show()
else:
    print("Not enough data to create a KDE plot for each Gender.")
```

[ ]:

```
# Mean Score and age
df_user_age = df_user.dropna(subset=['Birthday', 'Mean Score'])
df_user_age['Birthday'] = pd.to_datetime(df_user_age['Birthday'],␣
↪errors='coerce') # ensure Birthday is the type of datatime

current_year = 2023 # set current year
df_user_age['Age'] = current_year - df_user_age['Birthday'].dt.year # calculate␣
↪the age

age_slice = [0, 18, 25, 35, 45, 55, 65, 120] # split the age
labels = ['0-18', '18-25', '25-35', '35-45', '45-55', '55-65', '65+'] # set the␣
↪age group
df_user_age['Age Group'] = pd.cut(df_user_age['Age'], age_slice, labels=labels,␣
↪right=False)
# filter out 'Age Group' categories with insufficient data
min_elements = 2  # set the minimum number of elements required in each group
filtered_age_groups = df_user_age.groupby('Age Group').filter(lambda x: len(x)␣
↪>= min_elements)
# prepare data for KDE plot
age_groups = filtered_age_groups['Age Group'].unique()
hist_data = [filtered_age_groups[filtered_age_groups['Age Group'] == t]['Mean␣
↪Score'].values for t in age_groups]
# check if there is sufficient data to plot
if all(len(data) > 1 for data in hist_data):
    # create KDE plot
    fig = ff.create_distplot(hist_data, age_groups, show_hist=False,␣
↪show_rug=False)

    fig.update_layout(
        title={'text': "KDE Plot of Mean Rating by Age Group", "x": 0.5},
        xaxis_title="Mean Rating",
        yaxis_title="Density",
        legend_title="Age Group",
        width=1200,
        height=600
    )

    fig.show()
else:
    print("Not enough data to create a KDE plot for each Age Group.")
```

RQ4

October 30, 2024

0.0.1 How do user characteristics and behaviors, including age, gender and others, influence rating behavior? (RQ4)

```
[ ]: import pandas as pd
```

```
[ ]: df_user = pd.read_csv('data/users-details-2023.csv')
     print(df_user.shape)
     df_user.head()
```

```
[ ]: df_user_mean_score = df_user.dropna(subset=['Mean Score'])
     df_user_mean_score = df_user_mean_score[(df_user_mean_score['Mean Score'] != 0)␣
      ↪& (df_user_mean_score['Mean Score'] <= 10) & (df_user_mean_score['Days␣
      ↪Watched'] < 4000) & (df_user_mean_score['Mean Score'] > 5)]
     df_user = df_user_mean_score
```

```
[ ]: # Select the variables to be plotted.
     variables = [
         'Days Watched',
         'Watching',
         'Completed',
         'On Hold',
         'Dropped',
         'Plan to Watch',
         'Total Entries',
         'Rewatched',
         'Episodes Watched'
     ]
```

```
[ ]: import plotly.graph_objects as go  # for 3D plot visualization
     from plotly.offline import init_notebook_mode, iplot
     init_notebook_mode(connected=True)

     sample_fraction = 0.1 # set sample fraction to 0.1

     # crate scatter chart for each variable
     for var in variables:
         sampled_df = df_user.sample(frac=sample_fraction, random_state=1) # randomly␣
      ↪select sample
```

```
fig = go.Figure()
fig.add_trace(
    go.Scatter(
        x=sampled_df['Mean Score'],
        y=sampled_df[var],
        mode='markers',
        name=f'Mean Rating vs {var}',
        marker=dict(
            size=5,
            color=sampled_df[var],
            colorscale='Portland',
            showscale=True
        )
    )
)

fig.update_layout(
    title={'text': f'Mean Rating vs {var}', 'x': 0.5},
    xaxis_title='Mean Rating',
    yaxis_title=var,
    width=450,
    height=400,
    showlegend=False
)

fig.show()
```

anime_recommendation

October 30, 2024

0.0.1 How can we effectively recommend high-quality, highly-rated animations to users by integrating both anime and user features? (RQ5)

[ ]:
```python
import pandas as pd
import numpy as np
import torch
from transformers import BertTokenizer, BertModel
from sklearn.metrics.pairwise import cosine_similarity
```

[ ]:
```python
# road the dataset
df_anime = pd.read_csv('data/anime-dataset-2023.csv')
df_score = pd.read_csv('data/users-score-2023.csv')
df_user = pd.read_csv('data/users-details-2023.csv')
```

[ ]:
```python
# preprocess for anime dataset
df_anime = df_anime[df_anime['Score'] != 'UNKNOWN']
df_anime = df_anime[df_anime['Scored By'] != 'UNKNOWN']
# convert the floating-point numbers in the Scored By column to strings for␣
 ↪processing
df_anime['Scored By'] = df_anime['Scored By'].astype(str).str.split('.').str[0]
df_anime['Score'] = df_anime['Score'].astype(np.float64)
df_anime['Scored By'] = df_anime['Scored By'].astype(np.int64)
df_anime = df_anime[df_anime['Scored By'] >= 1000]
df_filtered = df_anime[df_anime['Type'].isin(['TV', 'Movie', 'OVA'])] # only␣
 ↪reserve TV Movie OVA
# filter animations with a duration of less than 1 hour
df_filtered = df_filtered[df_filtered['Duration'].apply(lambda x: 'hr' not in x)]
# filter out convertible years and retain only 21st-century animations
def extract_year(aired_date):
    try:
        return int(aired_date.split(' ')[-1])
    except ValueError:
        return np.nan
df_filtered['Year'] = df_filtered['Aired'].apply(extract_year)
df_filtered = df_filtered.dropna(subset=['Year'])
df_filtered = df_filtered[df_filtered['Year'] >= 2000]
df_sorted = df_filtered.sort_values(by='Score', ascending=False)
df_anime_500 = df_sorted.head(500)
```

```
[ ]: # preprocess for users' scores dataset
     df_score = df_score.dropna()
     user_counts = df_score['user_id'].value_counts()
     valid_users = user_counts[user_counts >= 3].index # drop users who rated less␣
      ↪than 3 anime
     df_score_filtered = df_score[df_score['user_id'].isin(valid_users)]
     df_top3_per_user = df_score_filtered.sort_values(by=['user_id', 'rating',␣
      ↪'anime_id'], ascending=[True, False, True])
     df_top3_per_user = df_top3_per_user.groupby('user_id').apply(lambda x: x.
      ↪sort_values(by='rating', ascending=False)).reset_index(drop=True)
```

```
[ ]: df_user_watched = df_top3_per_user # save a copy for filtering the anime watched␣
      ↪by users
```

```
[ ]: # merge two data frames to obtain popularity
     df_top3_per_user = df_top3_per_user.merge(df_anime_500[['anime_id',␣
      ↪'Popularity']], on='anime_id', how='left')
     df_top3_per_user = df_top3_per_user.dropna(subset=['Popularity'])
     df_top3_per_user = df_top3_per_user.sort_values(by=['user_id', 'rating',␣
      ↪'Popularity'], ascending=[True, False, True])
     df_top3_per_user = df_top3_per_user.groupby('user_id').head(3).
      ↪reset_index(drop=True)
```

```
[ ]: df_top3_per_user
```

```
[ ]: # preprocess for user dataset
     df_user = df_user.dropna(subset=['Birthday'])

     df_user['Birthday'] = pd.to_datetime(df_user['Birthday'], errors='coerce') #␣
      ↪ensure the Birthday column is of datetime type
     current_year = 2023
     df_user['age'] = current_year - df_user['Birthday'].dt.year # calculate age of␣
      ↪users
```

```
[ ]: # handle the missing gender, such as filling it with an empty string
     df_user['Gender'] = df_user['Gender'].fillna('')
     # check the distribution of gender
     print(df_user['Gender'].value_counts())
     # analyze the preferences of users of different genders in rating different␣
      ↪types of animations
     gender_preferences = df_score_filtered.merge(df_user[['Mal ID', 'Gender']],␣
      ↪left_on='user_id', right_on='Mal ID')
     # Calculate the mean and median of scores by gender.
     gender_anime_ratings = gender_preferences.merge(df_anime_500[['anime_id',␣
      ↪'Genres']], on='anime_id')
     gender_rating_stats = gender_anime_ratings.groupby(['Gender',␣
      ↪'Genres'])['rating'].agg(['mean', 'median']).reset_index()
```

```
# check the result
print(gender_rating_stats.sort_values(by='mean', ascending=False))
```

[ ]:
```
# initialize the BERT model and tokenizer locally (we cannot initialize the␣
↪model online for internet issue)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
```

[ ]:
```
# move the model to the GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# convert the text into a fixed-size vector
def get_embedding(text):
    inputs = tokenizer(text, return_tensors='pt', truncation=True, padding=True,␣
↪max_length=512)
    inputs = {key: value.to(device) for key, value in inputs.items()}
    with torch.no_grad():
        outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).squeeze().cpu()
```

[ ]:
```
# calculate embeddings for each anime in df_anime_500
df_anime_500 = df_anime_500.copy()
df_anime_500['embedding'] = df_anime_500['Synopsis'].apply(get_embedding)
```

[ ]:
```
df_top3_per_user
```

[ ]:
```
# the recommendation function
def recommend_anime(user_id, num_recommendations=5):
    user = df_user[df_user['Mal ID'] == user_id]
    # check if the user exists
    if user.empty:
        print(f"User ID {user_id} doesn't exist")
        return None

    user_age = user['age'].values[0]
    user_gender = user['Gender'].values[0]

    # Get the top three highest-rated animations by users.
    user_top_anime_ids = df_top3_per_user[df_top3_per_user['user_id'] ==␣
↪user_id]['anime_id'].unique()
    if len(user_top_anime_ids) == 0:
        print(f"User ID {user_id} doesn't have enough anime")
        return None

    user_top_anime = df_anime_500[df_anime_500['anime_id'].
↪isin(user_top_anime_ids)]
```

```
    # initialize an empty dataframe to store similarity results
    similarity_results = pd.DataFrame()

    # print the 3 highest-rated anime by users
    print("The 3 highest-rated anime by users:")
    print(df_top3_per_user[df_top3_per_user['user_id'] == user_id][['anime_id',␣
↪'Anime Title', 'rating']])

    # recommend based on three animations one by one
    for _, user_anime in user_top_anime.iterrows():
        user_embedding = user_anime['embedding']

        df_anime_500_copy = df_anime_500.copy()
        df_anime_500_copy['similarity'] = df_anime_500_copy['embedding'].
↪apply(lambda x: cosine_similarity([user_embedding], [x])[0][0])

        # filter the age of users
        if user_age < 17:
            df_anime_500_copy = df_anime_500_copy[df_anime_500_copy['Rating'] !=␣
↪'R - 17+ (violence & profanity)']

        # combine gender preference (weighted values scored by gender preference)
        if user_gender:
            for genre in df_anime_500_copy['Genres'].unique():
                gender_genre_mean =␣
↪gender_rating_stats[(gender_rating_stats['Gender'] == user_gender) &␣
↪(gender_rating_stats['Genres'] == genre)]['mean']
                if not gender_genre_mean.empty:
                    mean_rating = gender_genre_mean.values[0]
                    df_anime_500_copy.loc[df_anime_500_copy['Genres'] == genre,␣
↪'similarity'] *= (mean_rating / 10)

        # calculated the weighted score
        df_anime_500_copy['score_weighted'] = (df_anime_500_copy['Score'] * 0.5 +
                                               df_anime_500_copy['Favorites'] /␣
↪1000 * 0.3 +
                                               df_anime_500_copy['Popularity'].
↪apply(lambda y: (50 / y) * 0.2))

        # Remove the animations already included in the user ratings.
        df_anime_500_filtered = df_anime_500_copy[~df_anime_500_copy['anime_id'].
↪isin(df_user_watched)]

        # select the top 5 most similar animations
        top_recommendations = df_anime_500_filtered.sort_values(by='similarity',␣
↪ascending=False).head(5)
        similarity_results = pd.concat([similarity_results, top_recommendations])

    # calculate the total weighted score
    similarity_results['score_weighted'] = (similarity_results['Score'] * 0.5 +
                                            similarity_results['Favorites'] /␣
↪1000 * 0.3 +
                                            similarity_results['Popularity'].
↪apply(lambda y: (50 / y) * 0.2))

    # remove duplicates and select the top 'num_recommendations' animations with␣
↪the highest weighted scores
    final_recommendations = similarity_results.
↪drop_duplicates(subset='anime_id').sort_values(by='score_weighted',␣
↪ascending=False).head(num_recommendations)

    # print the results
    print("\nRecommended anime:")
    print(final_recommendations[['Name', 'Score', 'Popularity', 'Favorites',␣
↪'similarity', 'score_weighted']])

    return final_recommendations
```

[ ]:
```
df_user # check valid user ids
```

[ ]:
```
recommend_anime(20, 5) # test
```