**Term Project Final Submission**

**곽지우, 김민걸, 김원종, 이서빈**

**-Overview**

Many communities recommend content such as movies and books, but in the field of animation, you can see that there is no recommendation system. Therefore, we decided to create an animation recommendation system.

We use two filtering techniques and three machine learning models.

1. **Collaborative Recommender**

* **Using KNN(Cosine similarity)**
* **Using Aporior**

1. **Content Based Recommender**

* **Using TF-IDF weighting and Simgoid\_kernel**

**-Source code, output screenshot**

# Import Libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

# Import Dataset

anime = pd.read\_csv("anime.csv")

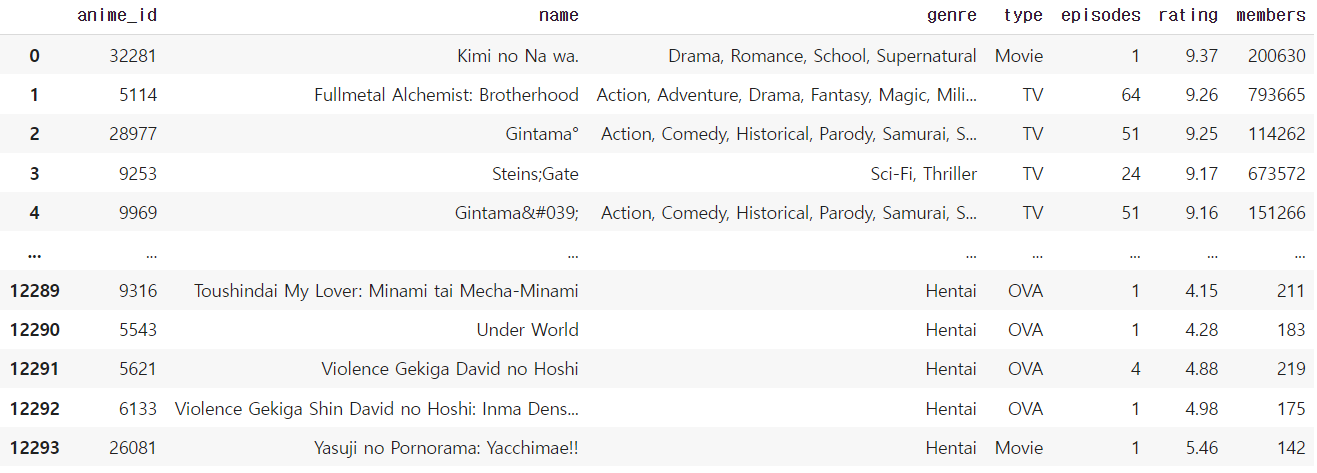
anime\_rating = pd.read\_csv("rating.csv")

anime2 = pd.read\_csv("animes.csv")

anime\_rating2 = pd.read\_csv("reviews.csv")

# Anime dataset

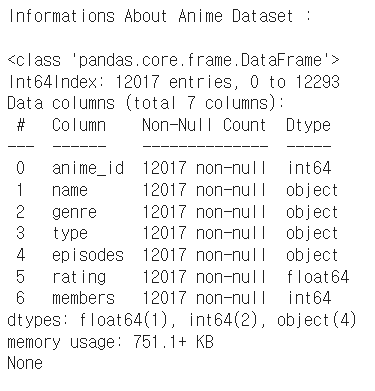
anime



# Informations About Anime Dataset

print(f"Informations About Anime Dataset :\n")

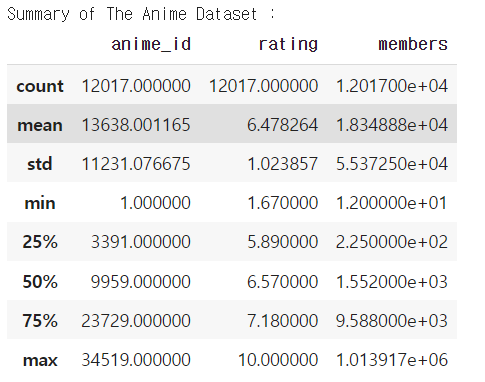
print(anime.info())



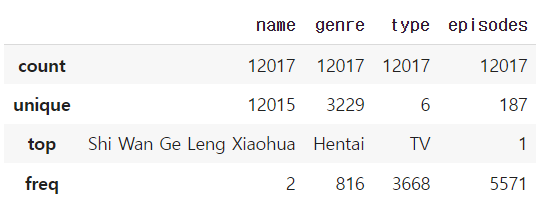
# Summary of the Anime Dataset

print(f"Summary of The Anime Dataset :")

anime.describe()



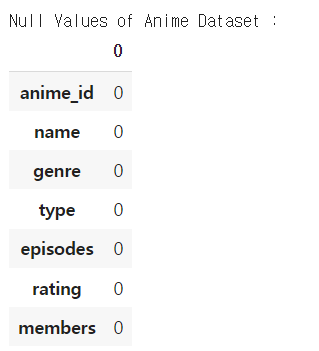
anime.describe(include=object)



# Null Values of Anime Dataset

print("Null Values of Anime Dataset :")

anime.isna().sum().to\_frame()

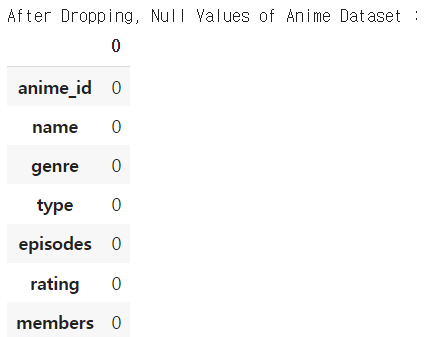


# Delete all rows with missing values

print("After Dropping, Null Values of Anime Dataset :")

anime.dropna(axis = 0, inplace = True)

anime.isna().sum().to\_frame()

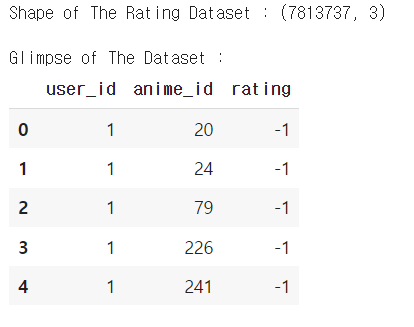


# Anime Rating dataset

print(f"Shape of The Rating Dataset : {anime\_rating.shape}")

print(f"\nGlimpse of The Dataset :")

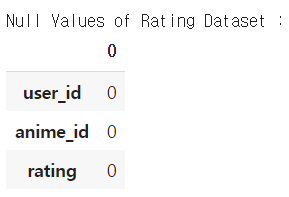
anime\_rating.head()



# Null Values of Rating Dataset

print("Null Values of Rating Dataset :")

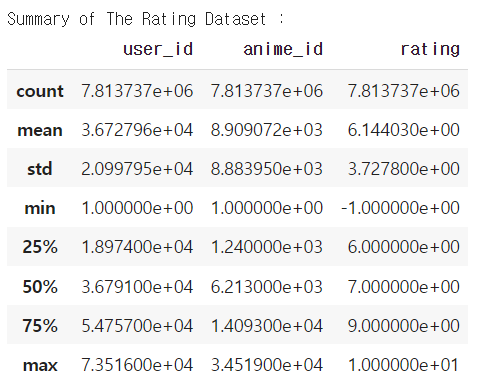
anime\_rating.isna().sum().to\_frame()



# Summary of The Rating Dataset

print(f"Summary of The Rating Dataset :")

anime\_rating.describe()



# Anime Full dataset

# Merge the anime and rating dataset

anime\_fulldata = pd.merge(anime,anime\_rating,on="anime\_id",suffixes= [None, "\_user"])

anime\_fulldata = anime\_fulldata.rename(columns={"rating\_user": "user\_rating"})

anime\_fulldata["user\_rating"].replace(to\_replace = -1 , value = np.nan ,inplace=True)

anime\_fulldata= anime\_fulldata.dropna(axis = 0)

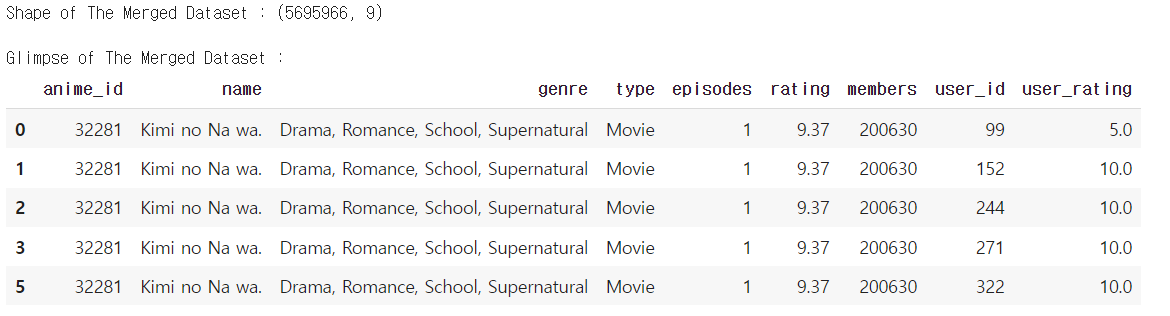
selected\_users = anime\_fulldata["user\_id"].value\_counts()

anime\_fulldata = anime\_fulldata[anime\_fulldata["user\_id"].isin(selected\_users[selected\_users >= 50].index)]

print(f"Shape of The Merged Dataset : {anime\_fulldata.shape}")

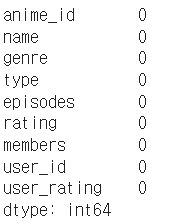
print(f"\nGlimpse of The Merged Dataset :")

anime\_fulldata.head()



# Fulldata Null Values

anime\_fulldata.isna().sum()



# sns setting

sns.set\_style("white")

sns.set\_context("poster",font\_scale = .7)

palette = ["#1d7874","#679289","#f4c095","#ee2e31","#ffb563","#918450","#f85e00","#a41623","#9a031e","#d6d6d6","#ffee32","#ffd100","#333533","#202020"]

# List animation types in the order in which they appear most frequently

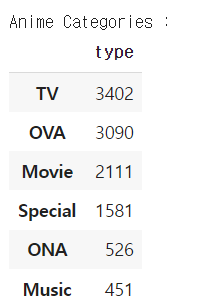
top\_anime = anime\_fulldata.copy()

top\_anime.drop\_duplicates(subset ="name", keep = "first", inplace = True)

top\_anime\_temp1 = top\_anime.sort\_values(["members"],ascending=False)

print("Anime Categories :")

top\_anime\_temp1['type'].value\_counts().to\_frame()



# Show the animation type distribution as a percentage in pie plot

plt.subplots(figsize=(12, 12))

labels = "TV","OVA","Movie","Special","ONA","Music"

size = 0.5

wedges, texts, autotexts = plt.pie([len(top\_anime\_temp1[top\_anime\_temp1["type"]=="TV"]["type"]),

                                    len(top\_anime\_temp1[top\_anime\_temp1["type"]=="OVA"]["type"]),

                                    len(top\_anime\_temp1[top\_anime\_temp1["type"]=="Movie"]["type"]),

                                    len(top\_anime\_temp1[top\_anime\_temp1["type"]=="Special"]["type"]),

                                    len(top\_anime\_temp1[top\_anime\_temp1["type"]=="ONA"]["type"]),

                                    len(top\_anime\_temp1[top\_anime\_temp1["type"]=="Music"]["type"])],

                                    explode = (0,0,0,0,0,0),

                                    textprops=dict(size= 20, color= "white"),

                                    autopct="%.2f%%",

                                    pctdistance = 0.7,

                                    radius=.9,

                                    colors = palette,

                                    shadow = True,

                                    wedgeprops=dict(width = size, edgecolor = "#1c1c1c",

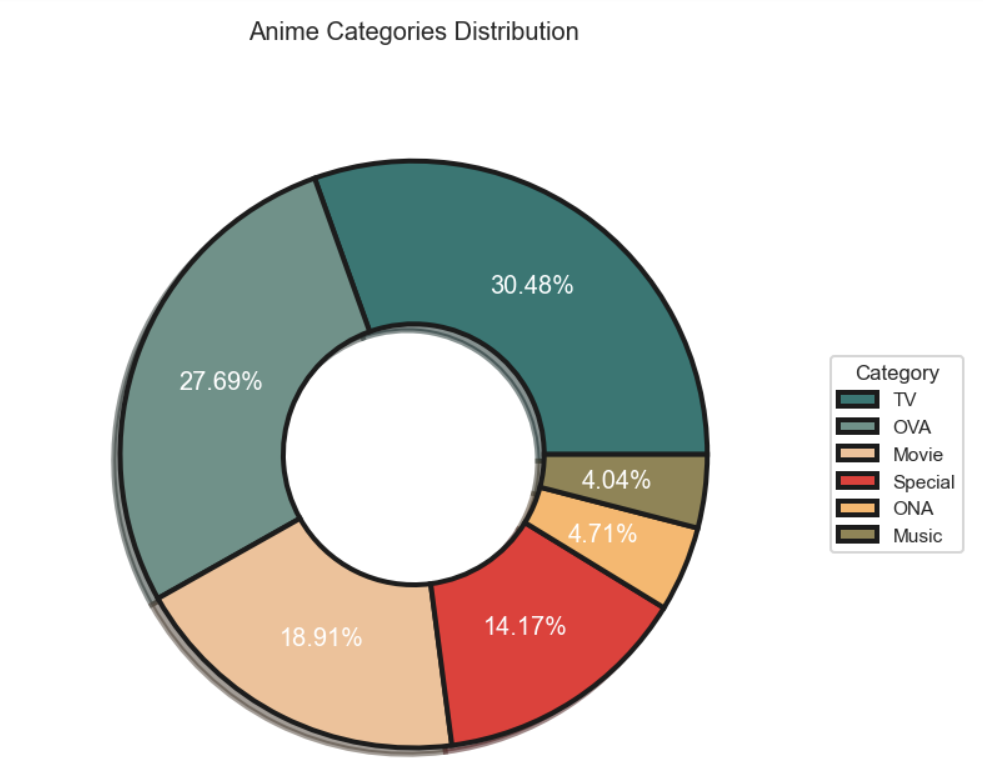
                                    linewidth = 4),

                                    startangle = 0)

plt.legend(wedges, labels, title="Category",loc="center left",bbox\_to\_anchor=(1, 0, 0.5, 1))

plt.title("\nAnime Categories Distribution",fontsize=20)

plt.show()



"""

Show the animation's average ratings distribution as a bar graph

- Most of the Anime ratings are spread between 5.5 - 8.0

- Most of the users ratings are spread between 6.0 - 10.0

- The mode of the users ratings distribution is around 7.0 - 8.0

- Both the distribution are left skewed

- Users rating(-1) is an outlier in ratings of users which can be discarded

"""

top\_anime\_temp2 = top\_anime.sort\_values(["rating"],ascending=False)

\_, axs = plt.subplots(2,1,figsize=(20,16),sharex=False,sharey=False)

plt.tight\_layout(pad=6.0)

sns.histplot(top\_anime\_temp2["rating"],color=palette[11],kde=True,ax=axs[0],bins=20,alpha=1,fill=True,edgecolor=palette[12])

axs[0].lines[0].set\_color(palette[12])

axs[0].set\_title("\nAnime's Average Ratings Distribution\n",fontsize = 25)

axs[0].set\_xlabel("Rating\n", fontsize = 20)

axs[0].set\_ylabel("Total", fontsize = 20)

sns.histplot(anime\_fulldata["user\_rating"],color=palette[12],kde=True,ax=axs[1],bins="auto",alpha=1,fill=True)

axs[1].lines[0].set\_color(palette[11])

# axs[1].set\_yscale("log")

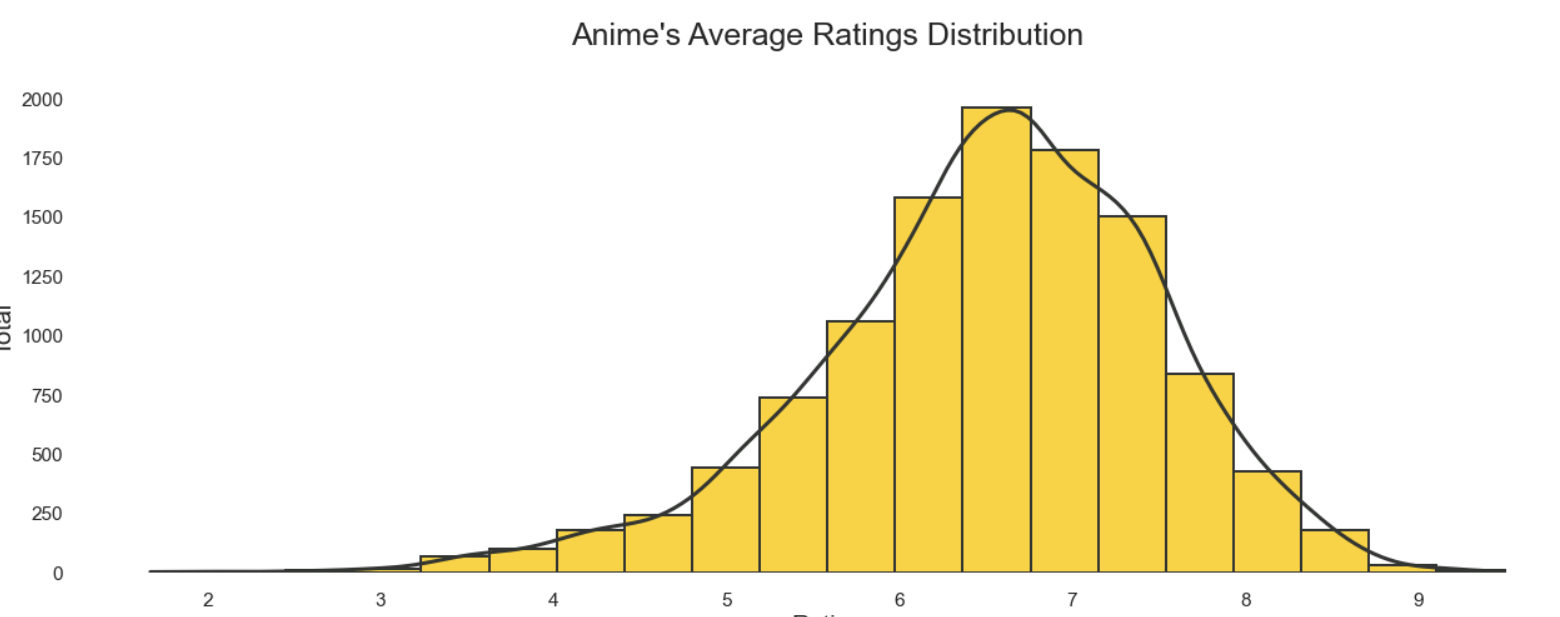
axs[1].set\_title("\n\n\nUsers Anime Ratings Distribution\n",fontsize = 25)

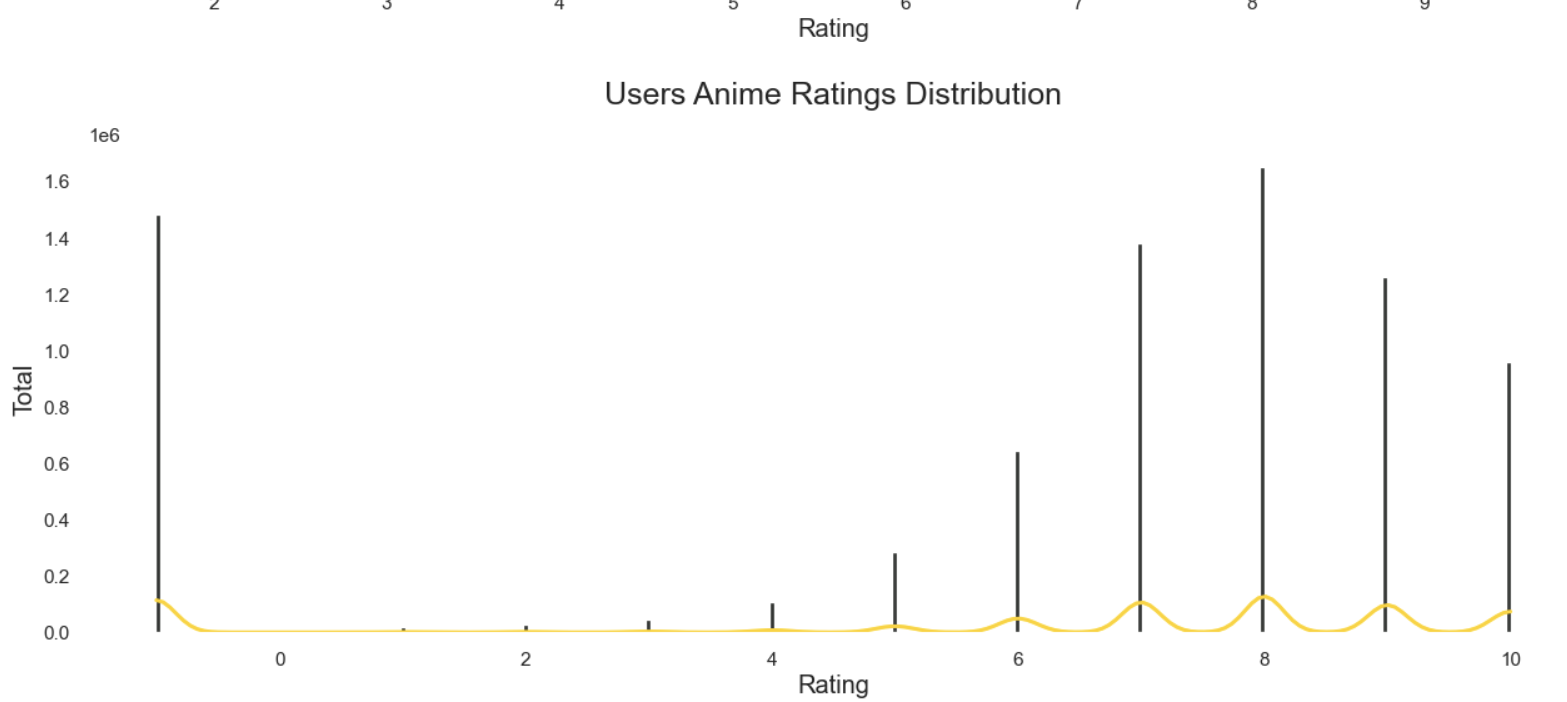
axs[1].set\_xlabel("Rating", fontsize = 20)

axs[1].set\_ylabel("Total", fontsize = 20)

sns.despine(left=True, bottom=True)

plt.show()





# List animation types in the order in which they appear most frequently

top\_anime\_temp3 = top\_anime[["genre"]]

top\_anime\_temp3["genre"] = top\_anime\_temp3["genre"].str.split(", | , | ,")

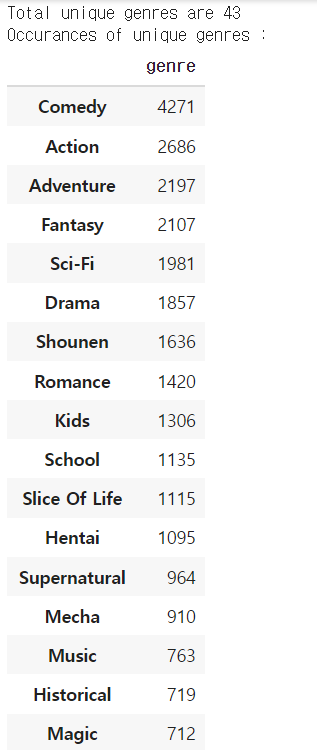
top\_anime\_temp3 = top\_anime\_temp3.explode("genre")

top\_anime\_temp3["genre"] = top\_anime\_temp3["genre"].str.title()

print(f'Total unique genres are {len(top\_anime\_temp3["genre"].unique())}')

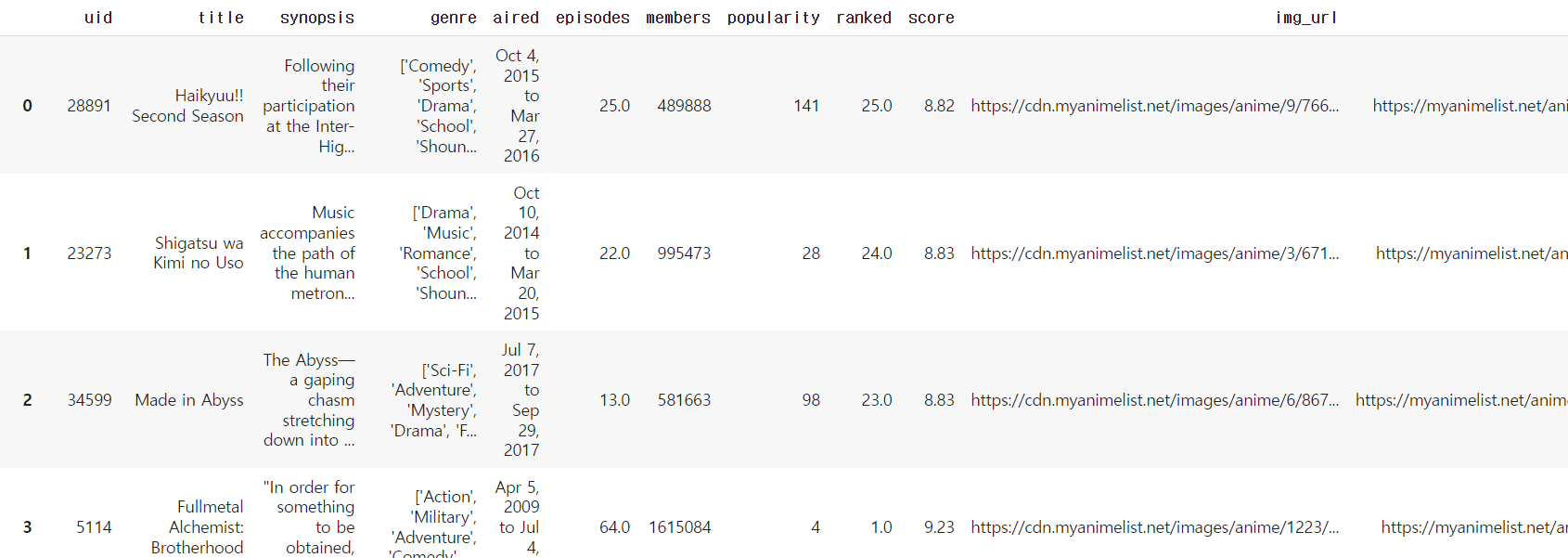
print(f'Occurances of unique genres :')

top\_anime\_temp3["genre"].value\_counts().to\_frame()



# Test anime data and rating

anime2



# remove not using features

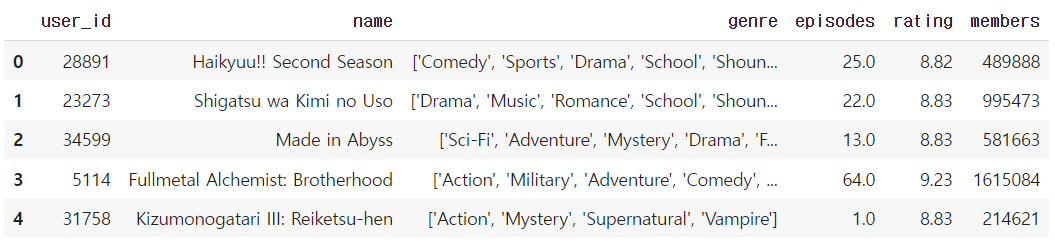
anime2 = anime2[['uid','title','genre','episodes','score','members']]

anime2.rename(columns = {"uid" : "user\_id"},inplace=True)

anime2.rename(columns = {"title" : "name"},inplace=True)

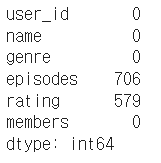
anime2.rename(columns = {"score" : "rating"},inplace=True)

anime2.head()



# anime2 null values

anime2.isna().sum()



# drop null values

anime2.dropna(axis = 0, inplace = True)

# genre

genre = anime2['genre']

for i,each\_anime\_genre in enumerate(genre):

    each\_anime\_genre = each\_anime\_genre.replace('[','')

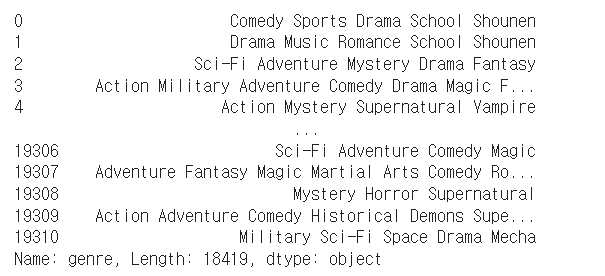
    each\_anime\_genre = each\_anime\_genre.replace(']','')

    each\_anime\_genre = each\_anime\_genre.replace("'","")

    each\_anime\_genre = each\_anime\_genre.replace(",","")

    genre.iloc[i] = each\_anime\_genre

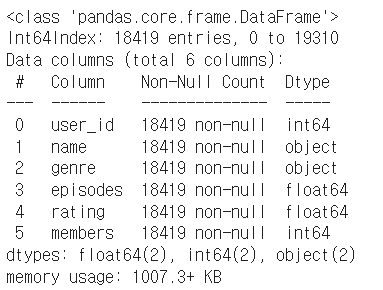
genre



anime2.head()



anime2.info()



# anime2 rating

anime\_rating2



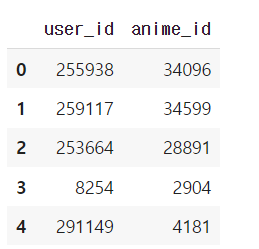
# rename rating2

anime\_rating2.rename(columns = {"uid" : "user\_id"},inplace=True)

anime\_rating2.rename(columns = {"anime\_uid" : "anime\_id"},inplace=True)

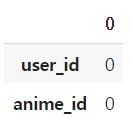
anime\_rating2 = anime\_rating2[["user\_id","anime\_id"]]

anime\_rating2.head()



# anime2 rating null values

anime\_rating2.isna().sum().to\_frame()



# make anime2 fulldata

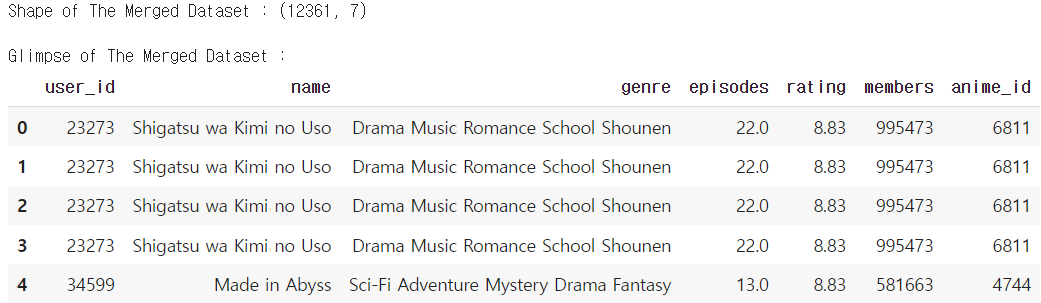
anime\_fulldata2 = pd.merge(anime2,anime\_rating2,on="user\_id",suffixes= [None, "\_user"])

anime\_fulldata2 = anime\_fulldata2.rename(columns={"rating\_user": "user\_rating"})

print(f"Shape of The Merged Dataset : {anime\_fulldata2.shape}")

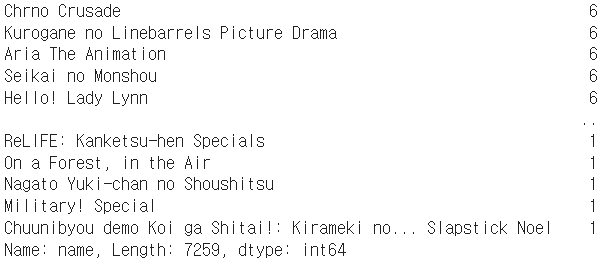
print(f"\nGlimpse of The Merged Dataset :")

anime\_fulldata2.head()

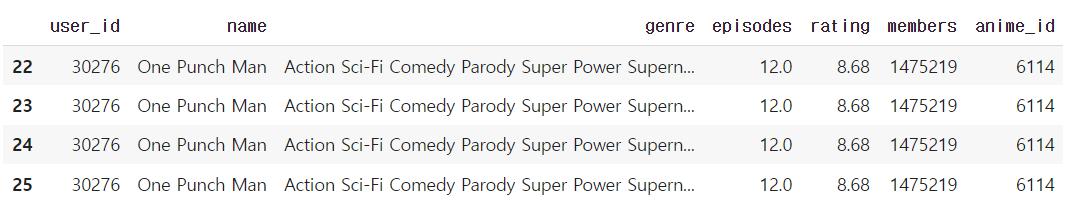


# anime2 fulldata name data

anime\_fulldata2['name'].value\_counts()



anime\_fulldata2[anime\_fulldata2['name'] == 'One Punch Man']



# Data Preporcessing

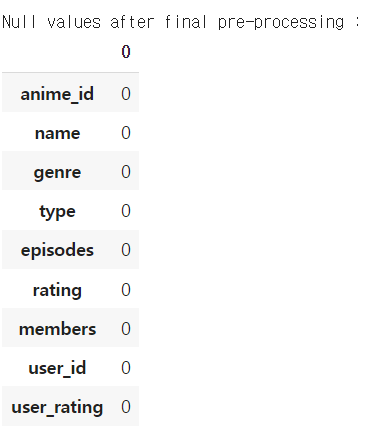
data = anime\_fulldata.copy()

data["user\_rating"].replace(to\_replace = -1 , value = np.nan ,inplace=True)

data = data.dropna(axis = 0)

print("Null values after final pre-processing :")

data.isna().sum().to\_frame()



"""

Consider minimum 50 ratings by the user as a threshold value

There are a lot of users who have rated only once, even if they have rated 5 animes,

so it can't be considered as a valuable record for recommendation.

"""

selected\_users = data["user\_id"].value\_counts()

data = data[data["user\_id"].isin(selected\_users[selected\_users >= 50].index)]

"""

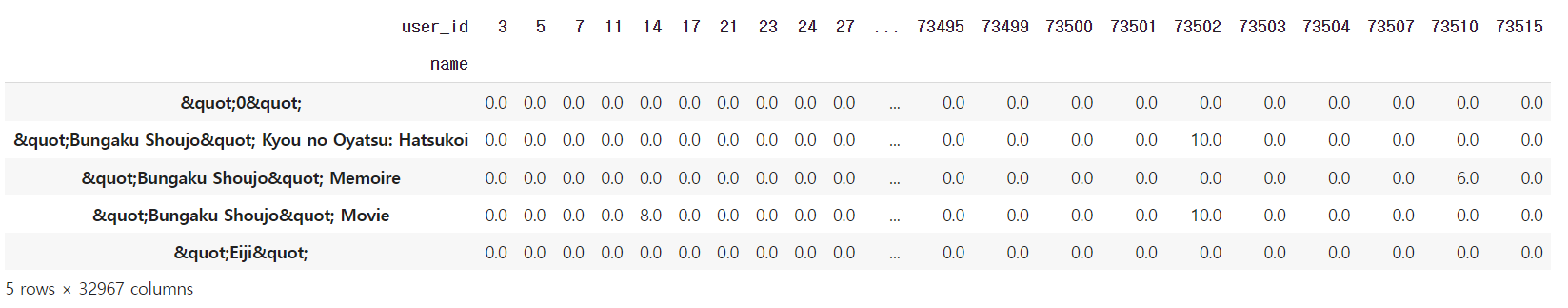
Create a pivot table consists of rows as title and columns as user id

It will help us to create sparse matrix which can be very helpful in finding the cosine similarity

"""

data\_pivot\_temp = data.pivot\_table(index="name",columns="user\_id",values="user\_rating").fillna(0)

data\_pivot\_temp.head()



# Remove japanese/special character symbols in anime name

import re

def text\_cleaning(text):

    text = re.sub(r'"', '', text)

    text = re.sub(r'.hack//', '', text)

    text = re.sub(r"'", '', text)

    text = re.sub(r"A's", '', text)

    text = re.sub(r"I'", 'I\'', text)

    text = re.sub(r'&', 'and', text)

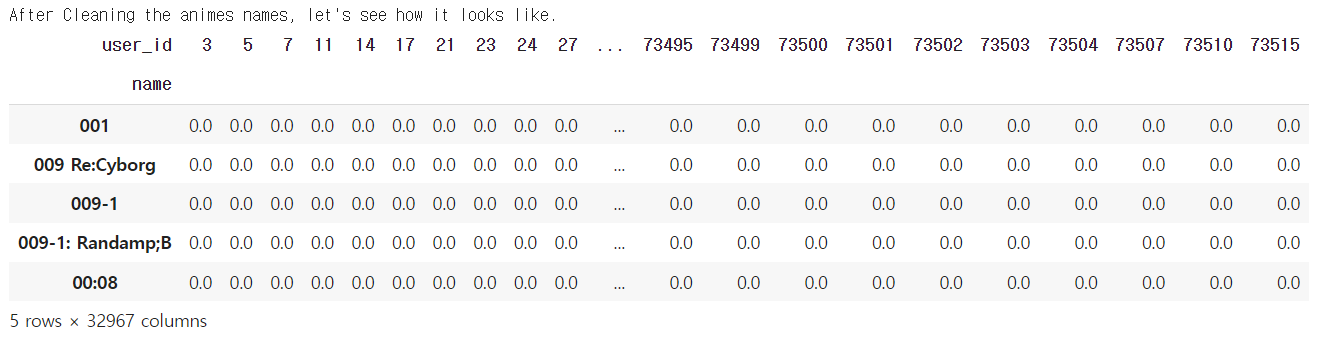
    return text

data["name"] = data["name"].apply(text\_cleaning)

data\_pivot = data.pivot\_table(index="name",columns="user\_id",values="user\_rating").fillna(0)

print("After Cleaning the animes names, let's see how it looks like.")

data\_pivot.head()



# Collaborative Recommender

"""

Using KNN(cosine similarity)

- KNN is a machine learning algorithm to find clusters of similar users based on common anime ratings, and make predictions using the average rating of top-k nearest neighbors

- Cosine similarity refers to the similarity of two vectors obtained using the cosine angle between two vectors. If the two vectors have exactly the same direction, they have a value of 1, and if they have an angle of 90°, they have a value of 0, 180° and vice versa, they have a value of -1

- The cosine distance is calculated using the metric method of the KNN algorithm. When an animation title is entered using cosine distance, an animation that users will like is provided.

"""

def search\_KNN(domain\_data,rating\_data,fulldata,domain\_id,title):

  data = fulldata.copy()

  data\_pivot = data.pivot\_table(index="name",columns="user\_id",values="rating").fillna(0)

  # Comput Cosin simmiliarity

  from scipy.sparse import csr\_matrix

  from sklearn.neighbors import NearestNeighbors

  data\_matrix = csr\_matrix(data\_pivot.values)

  model\_knn = NearestNeighbors(metric = "cosine", algorithm = "brute")

  model\_knn.fit(data\_matrix)

  distances, indices = model\_knn.kneighbors(data\_pivot.loc[title].values.reshape(1, -1), n\_neighbors = 6)

  # recommendation

  no = []

  name = []

  distance = []

  for i in range(0, len(distances.flatten())):

      if i == 0:

          print('Recommendations for {} viewers :'.format(title))

      else:

          no.append(i)

          name.append(data\_pivot.index[indices.flatten()[i]])

          distance.append(distances.flatten()[i])

  dic = {"No" : no, "Name" : name}

  recommendation = pd.DataFrame(data = dic)

  recommendation.set\_index("No", inplace = True)

  return recommendation

"""

Using Apriori

- The Apriori algorithm is an association rule, which refers to another event rule that occurs (often) together when a specific event occurs.

- When a single animation title is input using the apriori algorithm, and animations title that the user may like is suggested.

"""

from mlxtend.frequent\_patterns import apriori, association\_rules

def make\_data(domain\_data,rating\_data,fulldata,criterion,domain\_id):

  #make data frame

  df = fulldata.copy()

  df1 = df[df["rating"]>criterion].drop\_duplicates()

  return df1,df

def make\_rules(df1,df,min\_sup):

  crosstab = pd.crosstab(df1["user\_id"], df["name"]).astype('bool')

  freq\_domain = apriori(crosstab,min\_support=min\_sup,use\_colnames=True)

  rules = association\_rules(freq\_domain,metric='confidence',min\_threshold=0.1)

  rules['antecedents'] = rules['antecedents'].apply(lambda x: list(x)[0])

  rules['consequents'] = rules['consequents'].apply(lambda x: list(x)[0])

  return rules

def serach\_aprior(rules,string):

  search\_df = rules[rules['antecedents'].str.lower()== string.lower()]

  search\_df.sort\_values(by='lift', ascending=False)

  return search\_df[:10]['consequents']

# Content Based Recommender

"""

Using TF-IDF weighting and Sigmoid\_kernel

- TF-IDF, or term frequency-inverse document frequency, is a figure that expresses the statistical importance of any given word to the document collection as a whole. TF-IDF is calculated by multiplying term frequency and inverse document frequency.

- Sigmoid function returns two values, 0 and 1, therefore it is more suitable for binary classification problems.

- It recommends movies using SVD using animation information that the user has seen in the past.

"""

def search\_Tfidf(title,domain\_data,rating\_data,full\_data,split):

    from sklearn.feature\_extraction.text import TfidfVectorizer

    from sklearn.metrics.pairwise import sigmoid\_kernel

    # Convert vector

    tfv = TfidfVectorizer(min\_df=3, max\_features=None, strip\_accents="unicode", analyzer="word",

                      token\_pattern=r"\w{1,}", ngram\_range=(1, 3), stop\_words = "english")

    rec\_data = full\_data.copy()

    rec\_data.drop\_duplicates(subset = "name", keep = "first", inplace = True)

    rec\_data.reset\_index(drop = True, inplace = True)

    genres = rec\_data["genre"].str.split(split).astype(str)

    tfv\_matrix = tfv.fit\_transform(genres)

    sig = sigmoid\_kernel(tfv\_matrix, tfv\_matrix)      # Computing sigmoid kernel

    rec\_indices = pd.Series(rec\_data.index, index = rec\_data["name"]).drop\_duplicates()

    idx = rec\_indices[title] # Getting index corresponding to original\_title

    sig\_score = list(enumerate(sig[idx]))  # Getting pairwsie similarity scores

    sig\_score = sorted(sig\_score, key=lambda x: x[1], reverse=True)

    sig\_score = sig\_score[1:11]

    anime\_indices = [i[0] for i in sig\_score]

    # Top 10 most similar movies

    rec\_dic = {"No" : range(1,11),

               "Name" : domain\_data["name"].iloc[anime\_indices].values}

    dataframe = pd.DataFrame(data = rec\_dic)

    dataframe.set\_index("No", inplace = True)

    print(f"Recommendations for {title} viewers :\n")

    return dataframe

# recommend function

def recommend(title, main\_data, rating\_data, fulldata, data\_id, criterion, split,min\_sup):

    print('-- Collaborative Recommender / KNN --')

    print(search\_KNN(main\_data, rating\_data, fulldata, data\_id, title))

    print()

    print('-- Collaborative Recommender / Apriori --')

    df1,df = make\_data(main\_data, rating\_data, fulldata, criterion, data\_id)

    rules = make\_rules(df1,df,min\_sup)

    print(serach\_aprior(rules,title))

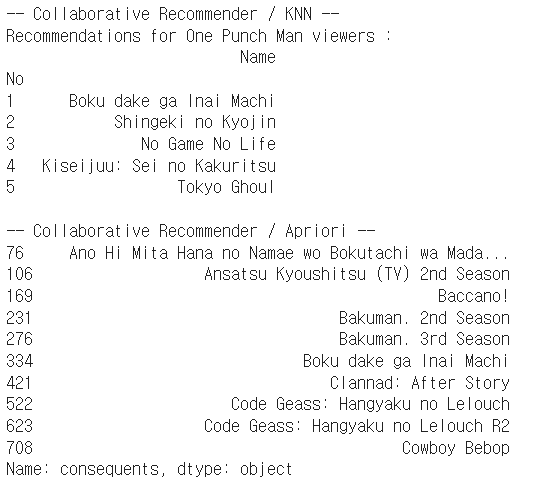
    print()

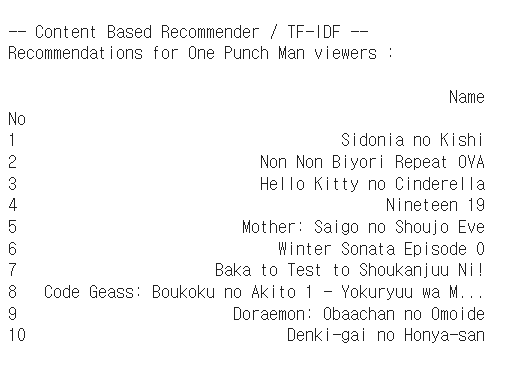
    print('-- Content Based Recommender / TF-IDF --')

    print(search\_Tfidf(title, main\_data, rating\_data, fulldata, split))

    print()

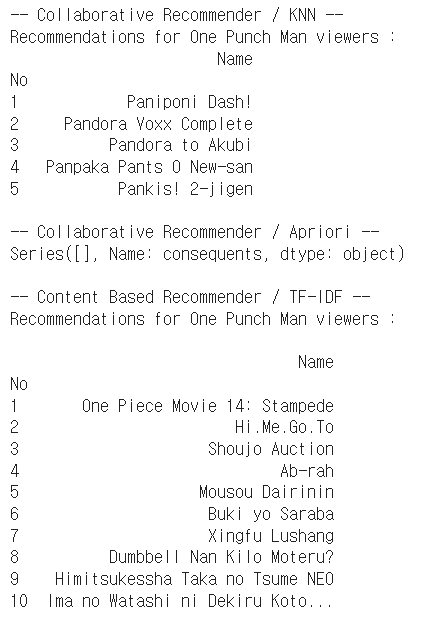
recommend('One Punch Man', anime, anime\_rating, anime\_fulldata, 'anime\_id', 8.5, ', | , | ,',0.05)





# Applied to the model

recommend('One Punch Man', anime2, anime\_rating2, anime\_fulldata2, 'anime\_id', 8.5, ' ',0.01)#min\_sup = 0.01



**-Open source software architecture description**

**First we make 3 function for recommendation system**

**텍스트, 테이블이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

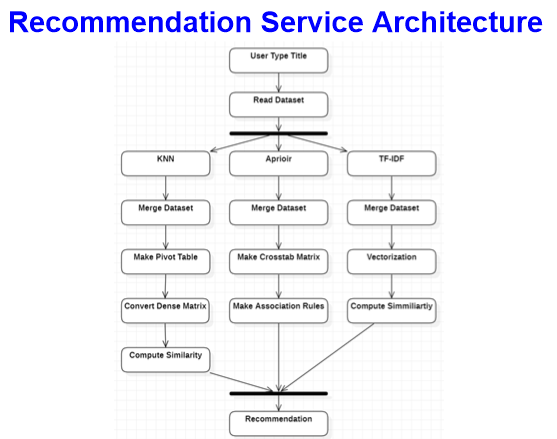
자동 생성된 설명**

**We combined the above three models and created a function that allows users to receive recommendations for all three models at once.**

테이블이(가) 표시된 사진

자동 생성된 설명

**Return : KNN, Aprior, TF-IDF recommendation list**

****

When a user leaves and inputs data (domain data, rating data) in the animation title and model, the animation is recommended to the user through three recommendation systems. After integrating the data on the ceiling and measuring the similarity through a machine learning model, we recommend an animation based on the title entered by the user based on the similarity.

**-Second dataset used to apply model we’ve madepasted-image.tiff**

This dataset contains information about Anime (16k), Reviews (130k) and Profiles (47k) crawled from <https://myanimelist.net/> at 05/01/20.

The dataset contains 2 files:

* animes.csv contains list of anime, with title, title synonyms, genre, duration, rank, populatiry, score, airing date, episodes and many other important data about individual anime providing sufficient information about trends in time about important aspects of anime. Rank is in float format in csv, but it contains only integer value. This is due to NaN values and their representation in pandas.
* reviews.csv contains information about reviews users x animes, with text review and scores.

**-Compare with actual recommendation system**

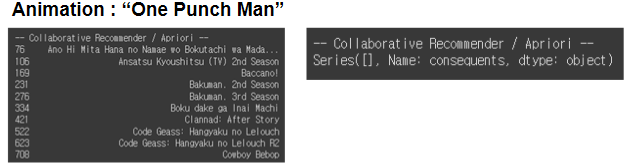
**KNN**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**It can be seen that even one animation has no overlapping parts.**

**Aporior**

****

**If other data is applied to the model, it may not be possible to recommend a movie.**

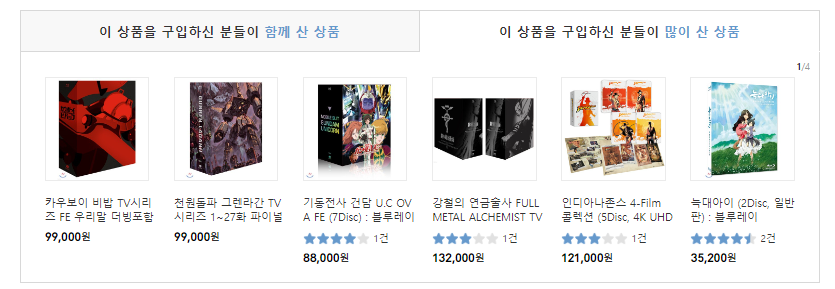
**TF-IDF**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**It can be seen that even one animation has no overlapping parts.**

At first, we tried to compare two data(animation dataset, movie dataset) for one model, but it was meaningless. So we compared it with the current recommendation system.We predicted that the model would yield the same result for the same animation, even with different data, if the model is correct. However, as I tried it myself, I found out that this is not the case. We've been discussing these results. It was found that it is not appropriate to check the results with the other two data because users of the two data have different evaluations of the content. In order to evaluate the model, we need to know how users rated the recommended data. But this is very difficult!

Following is the animations recommended by yes24 to users who purchased one punch man

When applying the apriori model to the anime1 dataset, <Cowboy Bebob> was recommended in the same way as in yes24. And there was no same animation on the list of recommendations using anime2 dataset. There are many differences between the animation list that yes24 has and the animation list of the dataset we used, so the similarity of the results seems to be poor.

**-Github URL**

<https://github.com/leeseobin00/AnimationRecommendation>

**-Google Colab URL**

**We collaborated in Google’s colab environment. You can check the code in the following link.**

https://colab.research.google.com/drive/1TVSxOjAcFROo3wUjmlmAaz5I9iAZgDoO?usp=sharing