Collaboative Filtering Method (Anime Recommendation System)

Methods Implemented A. Memory Based (User Based and Item Based Collaborative Filtering)

Load Data and Preprocess Step

```
# Package imports
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
from sklearn.model_selection import train_test_split
```

```
# Mount the google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Imports
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
file_path = '/content/drive/My Drive/Anime Recommender System/Datasets/' # Generic file path
anime_rating_file_path = file_path + 'user-filtered.csv' # File path of the anime ratings along with user IDs
anime_data_file_path = file_path + 'anime-filtered.csv' # File path of the anime data
user_details_file_path = file_path + 'users-details-2023.csv' # File path of user details data
ratings_df = pd.read_csv(anime_rating_file_path) # Create dataframe of anime ratings

anime_df = pd.read_csv(anime_data_file_path)  # Create dataframe of anime

user_df = pd.read_csv(user_details_file_path) # Create dataframe for user details
```

```
# Show some rows in rating dataframe
print(ratings_df.head(5))
```

```
   user_id  anime_id  rating
0        0        67       9
1        0      6702       7
2        0       242      10
3        0      4898       0
4        0        21      10
```

```
print(user_df.head(5))
```

```
   Mal ID Username  Gender                   Birthday             Location  \
0       1    Xinil    Male  1985-03-04T00:00:00+00:00           California
1       3  Aokaado    Male                        NaN         Oslo, Norway
2       4  Crystal  Female                        NaN  Melbourne, Australia
3       9   Arcane     NaN                        NaN                  NaN
4      18      Mad     NaN                        NaN                  NaN

                      Joined  Days Watched  Mean Score  Watching  Completed  \
0  2004-11-05T00:00:00+00:00         142.3        7.37       1.0      233.0
1  2004-11-11T00:00:00+00:00          68.6        7.34      23.0      137.0
2  2004-11-13T00:00:00+00:00         212.8        6.68      16.0      636.0
3  2004-12-05T00:00:00+00:00          30.0        7.71       5.0       54.0
4  2005-01-03T00:00:00+00:00          52.0        6.27       1.0      114.0

   On Hold  Dropped  Plan to Watch  Total Entries  Rewatched  Episodes Watched
0      8.0     93.0           64.0          399.0       60.0            8458.0
1     99.0     44.0           40.0          343.0       15.0            4072.0
2    303.0      0.0           45.0         1000.0       10.0           12781.0
3      4.0      3.0            0.0           66.0        0.0            1817.0
4     10.0      5.0           23.0          153.0       42.0            3038.0
```

```
# Show some rows in anime dataframe
print(anime_df.head(3))
```

```
   anime_id                             Name  Score  \
0         1                     Cowboy Bebop   8.78
1         5  Cowboy Bebop: Tengoku no Tobira   8.39
2         6                           Trigun   8.24

                                          Genres            English name  \
0    Action, Adventure, Comedy, Drama, Sci-Fi, Space            Cowboy Bebop
1              Action, Drama, Mystery, Sci-Fi, Space  Cowboy Bebop:The Movie
2  Action, Sci-Fi, Adventure, Comedy, Drama, Shounen                  Trigun

    Japanese name                                          sypnopsis   Type  \
0       カウボーイビバップ  In the year 2071, humanity has colonized sever...     TV
1  カウボーイビバップ 天国の扉  other day, another bounty—such is the life of ...  Movie
2           トライガン  Vash the Stampede is the man with a $$60,000,0...     TV

  Episodes                         Aired  ...         Duration  \
0       26  Apr 3, 1998 to Apr 24, 1999  ...  24 min. per ep.
1        1                  Sep 1, 2001  ...    1 hr. 55 min.
2       26  Apr 1, 1998 to Sep 30, 1998  ...  24 min. per ep.

                           Rating Ranked Popularity  Members Favorites  \
0  R - 17+ (violence & profanity)   28.0         39  1251960     61971
1  R - 17+ (violence & profanity)  159.0        518   273145      1174
2       PG-13 - Teens 13 or older  266.0        201   558913     12944

  Watching  Completed  On-Hold  Dropped
0   105808     718161    71513    26678
1     4143     208333     1935      770
2    29113     343492    25465    13925

[3 rows x 25 columns]
```

```
'''
Filter the ratings dataframe by taking values of users having user ID less than 10000
We take the subset of the data. We could not take location because very few users have location values and some of them have inconsistencies.
'''
filtered_rating_df = ratings_df[ratings_df['user_id'] <= 10000]

# Counting total number of rows in filtered_rating_df
total_rows = filtered_rating_df.shape[0]

# Display the filtered DataFrame and the total number of rows
print(filtered_rating_df)
print("Total number of rows:", total_rows)
```

```
           user_id  anime_id  rating
0                0        67       9
1                0      6702       7
2                0       242      10
3                0      4898       0
4                0        21      10
...            ...       ...     ...
2966486      10000      1382       0
2966487      10000      4416       0
2966488      10000     23283       0
2966489      10000      4038       0
2966490      10000     16005       0

[2966491 rows x 3 columns]
Total number of rows: 2966491
```

```
# Preprocess the data
filtered_rating_df = filtered_rating_df.drop_duplicates(subset=['user_id', 'anime_id']) # Drop duplicate values
filtered_rating_df = filtered_rating_df.dropna() # Drop missing values
filtered_rating_df['rating'] = filtered_rating_df['rating'].astype(float) # Converting type of ratings to float
```

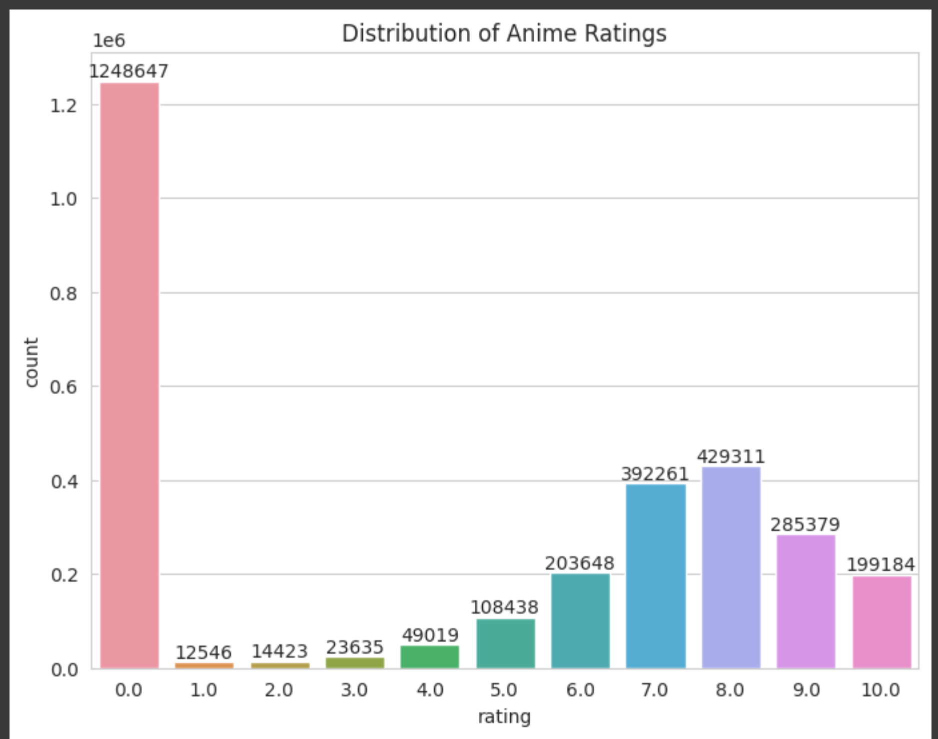
```
filtered_rating_df.head(4)
```

	user_id	anime_id	rating
0	0	67	9.0
1	0	6702	7.0
2	0	242	10.0
3	0	4898	0.0

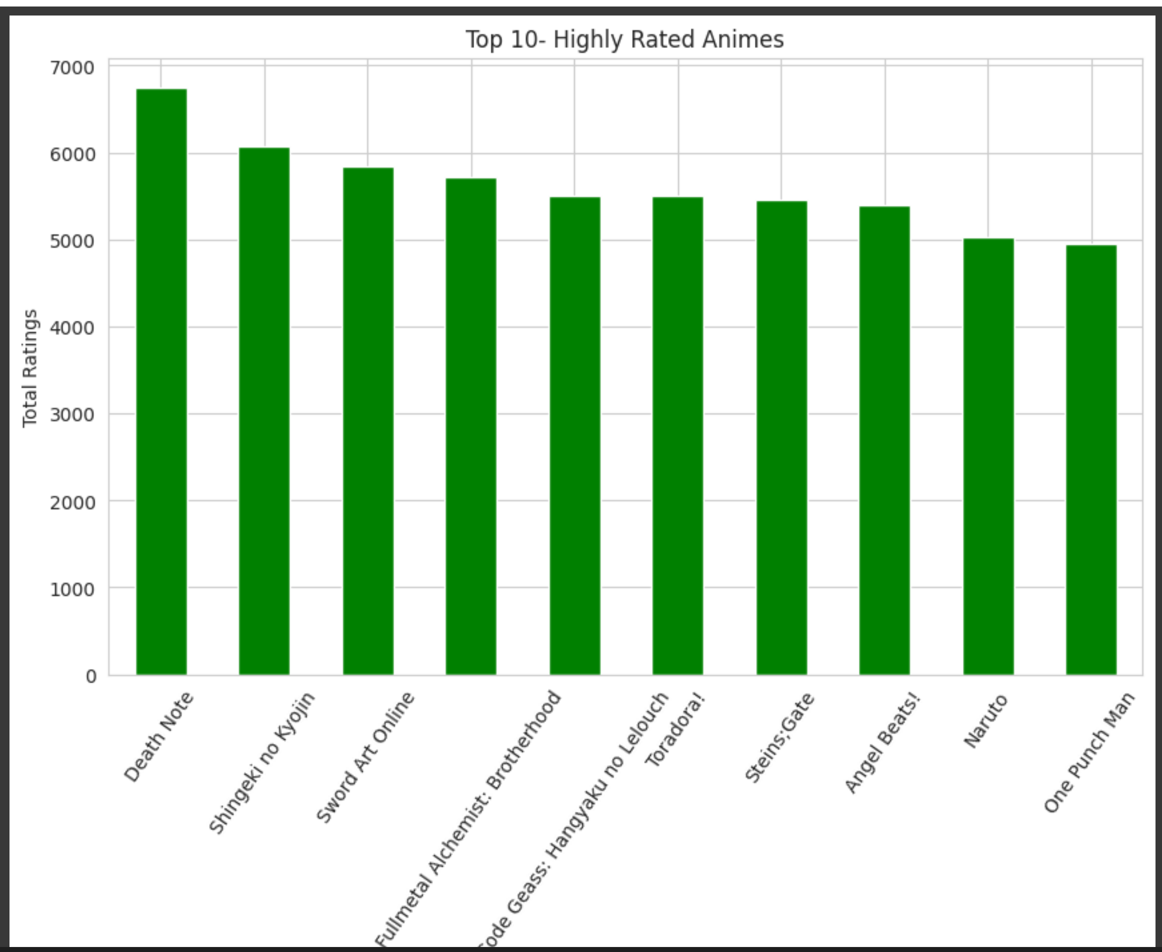
```
print(filtered_rating_df.isnull().sum()) # Checking missing values in dataframe
print(filtered_rating_df.duplicated().sum()) # Checking missing values in dataframe
print(filtered_rating_df.info()) # Print data frame summary
```

```
user_id     0
anime_id    0
rating      0
dtype: int64
0
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2966491 entries, 0 to 2966490
Data columns (total 3 columns):
 #   Column    Dtype  
---  ------    -----  
 0   user_id   int64  
 1   anime_id  int64  
 2   rating    float64
dtypes: float64(1), int64(2)
memory usage: 90.5 MB
None
```

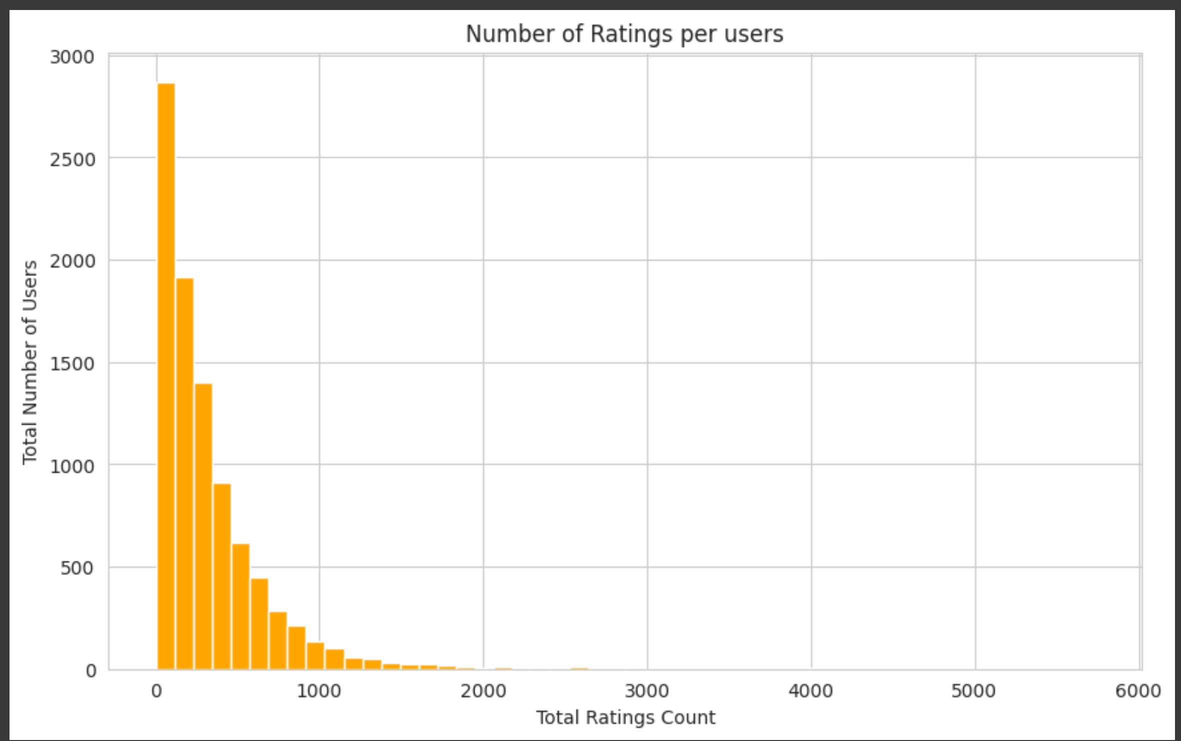
```
# Plot the distribution of ratings

sns.set_style("whitegrid")
plt.figure(figsize=(8,6))
# Calculating count of the rating
rating_counts = filtered_rating_df['rating'].value_counts().sort_index()

# Creating a countplot with frequency of the left
sns.countplot(x='rating', data=filtered_rating_df, order=rating_counts.index)

# Adding counts on top of bar
for i, count in enumerate(rating_counts):
    plt.text(i, count, str(count), ha='center', va='bottom')

plt.title('Distribution of Anime Ratings')
plt.show()
```




```
selected_anime_columns = anime_df[['anime_id', 'Name', 'Score', 'Genres', 'Type', 'Episodes']] # View selected columns of the anime dataframe
# Merge two dataframes based on 'anime_id' column value
merged_df = filtered_rating_df.merge(anime_df[['anime_id', 'Name', 'Genres']], how='left', on='anime_id')


# Highly rated animes (top 10 animes with more number ofa ratings)
highly_rated_movies = merged_df['Name'].value_counts().nlargest(10)
plt.figure(figsize=(10, 6))
highly_rated_movies.plot(kind='bar', color = 'green')
plt.xlabel('Anime Title')
plt.ylabel('Total Ratings')
plt.title('Top 10- Highly Rated Animes')
plt.xticks(rotation=55)
plt.show()
```




```
# Show number of ratings per given user
ratings_by_user = filtered_rating_df['user_id'].value_counts()
plt.figure(figsize=(10, 6))
plt.hist(ratings_by_user, bins=50, edgecolor='white', color='orange')
plt.xlabel('Total Ratings Count')
plt.ylabel('Total Number of Users')
plt.title('Number of Ratings per users')
plt.show()
```



A. Memory Based - User Based Collaborative Filtering

```
# Split the dataset into a train set and a test set
train_df, test_df = train_test_split(filtered_rating_df, test_size=0.3, random_state=42)

print("Train/test split of the dataset")
print()
print(f"Train Dataset: Showing 20 values from head \n\n {train_df.head(20)}")
print()
print("************************************************")
print()
print(f"Test Dataset: Showing 20 values from head \n\n {test_df.head(20)}")
print()
print("************************************************")
print()
print("Total Rows in train_df:", train_df.shape[0])
print("Total Rows test_df:", test_df.shape[0])
```

Train/test split of the dataset

Train Dataset: Showing 20 values from head

	user_id	anime_id	rating
2193031	7381	39587	0.0
2099470	7088	28221	0.0
181086	608	29758	6.0
2034105	6898	34577	9.0
829320	2800	10863	0.0
1115103	3784	5141	0.0
2697729	9113	38826	10.0
194165	653	37347	7.0
1157841	3941	26243	9.0
1124228	3824	12471	0.0
2065209	6986	31771	8.0
2078927	7024	48375	0.0
676915	2280	35889	9.0
1281850	4410	3791	0.0
2367594	7929	5525	10.0

```
340497      1059     23283     8.0
1885827     6416     42519     7.0
1782650     6083     14467     9.0
2195619     7387     38992     8.0
406711      1316      4793     8.0

***********************************************

Test Dataset: Showing 20 values from head

          user_id  anime_id  rating
2776983     9383     34281     7.0
2379902     7977      4282    10.0
2086774     7048      1727     8.0
379487      1221     39533     0.0
253332       822     40489     0.0
2735445     9222     31133     0.0
1287648     4432     35624     0.0
2322594     7798       949     9.0
209672       705     13759     9.0
1833788     6224      2012     0.0
1932274     6559     23233     0.0
1941382     6573      9751     0.0
1809107     6161     35203     0.0
2403573     8057       392     7.0
2586129     8707     34134     4.0
2092779     7062     10087     0.0
814029      2734     37799     7.0
528880      1733       585     6.0
2153879     7254      1818     0.0
2403031     8056     12487     8.0

***********************************************

Total Rows in train_df: 2076543
Total Rows test_df: 889948
```

Helper Functions Definition

All the functions needed in the code below for implementation are defined here

```
# Function to create user-anime matrix
def create_user_anime_matrix(train_df):
    """
    Using pandas-pivot function to create user-anime matrix
    Rows represent users, columns represent anime ids and values represent ratings
    """
    user_anime_matrix = train_df.pivot(index='user_id', columns='anime_id', values='rating')
    user_anime_matrix = user_anime_matrix.fillna(0) # Fill empty ratings with 0
    return user_anime_matrix

# Function to normalize user-anime matrix as ratings can have huge variation
def normalize_user_anime_matrix(user_anime_matrix):
   # Calculate average rating for each user considering mean of all the ratings
    user_rating_mean = user_anime_matrix.mean(axis = 1)

    # Subtracting value with row mean and normalizing the matrix
    user_anime_matrix_norm = user_anime_matrix.sub(user_rating_mean, axis = 0)

    return user_anime_matrix_norm

# Function to calculate similarity between users using cosine similarity
def calculate_user_similarity(user_anime_matrix_norm):

    # Calculate similarity using cosine similarity
    user_similarity = cosine_similarity(user_anime_matrix_norm)

    return user_similarity

"""
Function to create dataframe mapping user_ids and cosine similarity values between users.
The cosine similarity values are not normalized.
"""
def create_df_user_similarity(user_similarity, user_ids):

     # Create dataframe from numpy array of user similarity
    cosine_similarity_df =  pd.DataFrame(user_similarity)

    cosine_similarity_df.index = user_ids # Set row labels of dataframe
    cosine_similarity_df.columns = user_ids # Set column labels of dataframe

    return cosine_similarity_df

"""
Function to normalize cosine similarity between users
Those normalized similarities will be used as weight while calculating the anime weighted scores for recommendation
"""
def normalize_user_similarity(user_similarity):
    min_cosine_value = user_similarity.min() # Evaluate min value
    max_cosine_value = user_similarity.max() # Calculate max value

    # Normalize the similarity values between users
    normalized_user_similarity = (user_similarity - min_cosine_value) / (max_cosine_value - min_cosine_value)

    return normalized_user_similarity


"""
Function to create dataframe from normalized user similarity values.
The normalized user similarity that is numpy array will be converted to dataframe mapping user_ids.
The mappings will be used to find the similarity values of particular user (provided the user_id) with other users.
user_ids in argument is used to pass all the users such that we map respective similarity values.
"""
def create_df_norm_user_similarity(normalized_user_similarity, user_ids):

    # Create dataframe from numpy array of normalized user similarity
    cosine_weights_df =  pd.DataFrame(normalized_user_similarity)

    cosine_weights_df.index = user_ids # Set row labels of dataframe
    cosine_weights_df.columns = user_ids # Set column labels of dataframe

    return cosine_weights_df


"""
Function to calculate top N similar users for given user with user_id.
top_N_similary argument sets the number of similar users returned for the particular user.
similarity_threshold sets the value and we filter users having similarities more than threshold value.
"""
```

```
def find_similar_users(cosine_similarity_df, selected_user_id,
                       top_N_similar, similarity_threshold):

    # Find the user similarity from the cosine similarity dataframe
    selected_user_similarity = cosine_similarity_df.loc[selected_user_id]

    # Remove the selected_useer_id from calculation as it similarity with itself will be 1
    selected_user_similarity = selected_user_similarity.drop(selected_user_id)

    # Sort the similarity scores in descending order
    sorted_similarity = selected_user_similarity.sort_values(ascending=False)

    # Filter the similar users based on similarity_threshold value
    selected_similar_users = sorted_similarity[sorted_similarity >= similarity_threshold]

    # Select top N similar users
    top_N_similar_users = selected_similar_users.head(top_N_similar)


    return top_N_similar_users


# Function to calculate already watched animes by given user (user_id)
def already_watched_animes(train_df, selected_user_id):
    # Pick animes that the selected user has already watched
    watched_animes = train_df[train_df['user_id'] == selected_user_id]['anime_id'].tolist()

    watched_animes = list(set(watched_animes)) # Set unique anime ids

    return watched_animes

# Function to calculate animes watched by similar users
def similar_users_watched_animes(train_df, similar_user_ids):
    # Find animes watched by similar users
    similar_users_animes = train_df[train_df['user_id'].isin(similar_user_ids)]['anime_id'].tolist()

    similar_users_animes = list(set(similar_users_animes)) # Calculate unique anime ids

    return similar_users_animes

"""
Function to filter user_anime matrix by removing anime_ids such as anime_ids that selected user has
already watched is removed from the column. Likewise, the anime_ids that the similar users have watched are
kept, others are removed from columns for the calculation for particular selected user.
"""
def filter_user_anime_matrix(user_anime_matrix, user_watched_anime_ids,
                             similar_users_watched_anime_ids, similar_user_ids):

    # Removing the animes watched by the user
    user_anime_matrix_filtered = user_anime_matrix.drop(columns=user_watched_anime_ids, axis=1)

    # Keeping only similar users in the filtered matrix
    user_anime_matrix_filtered_sim = user_anime_matrix_filtered.loc[similar_user_ids]

    # Keeping only animes that the similar users have watched
    user_anime_matrix_filtered_sim = user_anime_matrix_filtered_sim[similar_users_watched_anime_ids]

    return user_anime_matrix_filtered_sim

"""
Function to calculate top-N anime recommendations ranked based on weighted anime scores.
The anime ratings provided by the similar users is multiplied by the cosine normalized weights
such that the user having high similarity will have high value of the score of the anime.
Likewise, the sum calculated for all the similar users is divided by the weights of all the similar users.
Thus, we get a weigthed score of the animes. The animes are sorted based on weighted score and top N
recommendations are returned.
"""
def calculate_top_N_recommendations(selected_user_id, user_anime_matrix_filtered_sim,  cosine_weights_df,
                                    similar_user_ids, top_N_value, printRecommendation=None):

    # Find the cosine normalized weight for particular user filtered by similar users of that user
    similarity_weights_user = cosine_weights_df.loc[selected_user_id][similar_user_ids]

    # Calculate the weighted ratings by multiplication with the weights
    weighted_scores = user_anime_matrix_filtered_sim.mul(similarity_weights_user, axis=0)

    sum_scores = weighted_scores.sum() # Calcuate sum of the weighted scores for animes

    similarity_weights_sum = similarity_weights_user.abs().sum() # Sum all the weights

    # Calculate weighted score for all the animes
    weighted_avg_scores = sum_scores / similarity_weights_sum

    # Sort animes on descending order based on scores
    sorted_anime_scores = weighted_avg_scores.sort_values(ascending=False)

    # Find top N anime recommended
    top_N_animes = sorted_anime_scores[:top_N_value]

    result = sorted_anime_scores.reset_index()  # Reset index to make merging with anime data easier

    # Merge the result with the anime_info DataFrame on "anime_id"
    result_with_anime_info = result.merge(anime_df, on="anime_id")
    result_with_anime_info = result_with_anime_info.rename(columns={0: "weighted_average_score"})

    if printRecommendation:
        # Print the recommended result
        print(f"Top-{top_N_value} Anime Recommendations for User ID = {selected_user_id}")
        print(f"{result_with_anime_info}")

    return result_with_anime_info['anime_id'].tolist() # Return list of recommended anime_ids
```

```
"""
Function to find Precision@N, Recall@N and F1-Score@N for particular user
The test dataframe is used to find out whether the recommendations in top N are relevant or not
rating_threshold value is passed such that we consider if the user has rated >= rating_threshold and the
anime is recommended, it falls into relevant recommended anime in the list.
"""
def evaluate_metrics_user(test_df, selected_user_id, rating_threshold, top_N_recommendations):

    # Finds all the animes above the rating_threshold from test dataframe considering the user liked those animes
    all_relevant_animes = test_df[(test_df['user_id'] == selected_user_id)
                                  & (test_df['rating'] >= rating_threshold)]['anime_id'].values

    total_recommendations = len(top_N_recommendations) # Total recommendations

    # Find relevant recommendations in top_N list
    relevant_recommendations = set(top_N_recommendations).intersection(all_relevant_animes)
```

```
    # Calculate precision@N
    if total_recommendations == 0:
        precision_at_N = 0
    else:
        precision_at_N = len(relevant_recommendations) / total_recommendations


    # Calculate Recall@N
    if total_recommendations == 0:
        recall_at_N = 0
    else:
        recall_at_N = len(all_relevant_animes) / len(relevant_recommendations)

    # Calculate F1-score@N
    if precision_at_N + recall_at_N == 0:
        f1_score_at_N = 0
    else:
        f1_score_at_N = 2 * (precision_at_N * recall_at_N) / (precision_at_N + recall_at_N)


    return precision_at_N, recall_at_N, f1_score_at_N


"""
Function to evaluate Average Precision@N
The Average Precision@K or AP@K is the sum of precision@K where the item at the kₜₕ rank is
relevant (rel(k)) divided by the total number of relevant items (r) in the top K recommendations
"""
def evaluate_average_precision_at_N(test_df, selected_user_id,
                                    rating_threshold, top_N_recommendations):
     # Finds all the animes above the rating_threshold from test dataframe considering the user liked those animes
    all_relevant_animes = test_df[(test_df['user_id'] == selected_user_id)
                                  & (test_df['rating'] >= rating_threshold)]['anime_id'].values

    total_recommendations = len(top_N_recommendations) # Total recommendations

    # Find relevant recommendations in top_N list
    relevant_recommendations = set(top_N_recommendations).intersection(all_relevant_animes)

    total_relevant_recommendations = len(relevant_recommendations) # Set the total number of relevant recommendations
    total_precision = 0 # Set the total precision value to 0
    relevant_anime_at_k = 0 # Set the number of relevant animes up to kth position

    for i in range(len(top_N_recommendations)):
        anime_rank = i + 1 # Set rank of anime as the loop starts from 0 index 1 is added

        if top_N_recommendations[i] in relevant_recommendations:
            relevant_anime_at_k += 1 # Relevant anime found, so increment count
            total_precision += relevant_anime_at_k / anime_rank

    if total_relevant_recommendations == 0:
        return 0 # If there are no relevant recommendations

    average_precision_at_N =  total_precision / total_relevant_recommendations

    return average_precision_at_N
```

```
# Build user anime matrix
user_anime_matrix = create_user_anime_matrix(train_df)
print(f"User-Anime Matrix: \n \n {user_anime_matrix}")
```

```
User-Anime Matrix:

 anime_id  1      5      6      7      8      15     16     17     18     \
user_id
0           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1           0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2           0.0    0.0    0.0    0.0    0.0    9.0    0.0    0.0    0.0
3           9.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    8.0
4           0.0    0.0    0.0    0.0    0.0    0.0    9.0    0.0    0.0
...         ...    ...    ...    ...    ...    ...    ...    ...    ...
9996        0.0    7.0    0.0    7.0    0.0    0.0   10.0    0.0    0.0
9997        0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9998        0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9999        0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
10000       0.0   10.0    8.0    0.0    0.0    0.0    0.0    0.0    0.0

anime_id  19     ...  48406  48409  48413  48414  48417  48418  48426  48438  \
user_id          ...
0           0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1           9.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2           0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3           0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
4           0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
...         ...  ...    ...    ...    ...    ...    ...    ...    ...    ...
9996        5.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9997        0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9998        0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9999        0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
10000       8.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

anime_id  48456  48488
user_id
0           0.0    0.0
1           0.0    0.0
2           0.0    0.0
3           0.0    0.0
4           0.0    0.0
...         ...    ...
9996        0.0    0.0
9997        0.0    0.0
9998        0.0    0.0
9999        0.0    0.0
10000       0.0    0.0

[9152 rows x 14904 columns]
```

```
# Normalize user anime matrix
user_anime_matrix_norm = normalize_user_anime_matrix(user_anime_matrix)
print(f"User-Anime Matrix (Normalized) : \n \n {user_anime_matrix_norm}")
```

```
 anime_id     1         5         6         7         8         15     \
user_id
0        -0.015700 -0.015700 -0.015700 -0.015700 -0.015700 -0.015700
1        -0.046028 -0.046028 -0.046028 -0.046028 -0.046028 -0.046028
2        -0.021538 -0.021538 -0.021538 -0.021538 -0.021538  8.978462
3         8.887547 -0.112453 -0.112453 -0.112453 -0.112453 -0.112453
4        -0.039721 -0.039721 -0.039721 -0.039721 -0.039721 -0.039721
```

```
    9996     -0.092391  9.907609 -0.092391  9.907609 -0.092391 -0.092391
    9997     -0.000604 -0.000604 -0.000604 -0.000604 -0.000604 -0.000604
    9998     -0.135333 -0.135333 -0.135333 -0.135333 -0.135333 -0.135333
    9999     -0.050322 -0.050322 -0.050322 -0.050322 -0.050322 -0.050322
    10000    -0.295894  9.704106  7.704106 -0.295894 -0.295894 -0.295894

    anime_id     16        17        18        19     ...     48406     48409  \
    user_id                                           ...
    0        -0.015700 -0.015700 -0.015700 -0.015700  ... -0.015700 -0.015700
    1        -0.046028 -0.046028 -0.046028  8.953972  ... -0.046028 -0.046028
    2        -0.021538 -0.021538 -0.021538 -0.021538  ... -0.021538 -0.021538
    3        -0.112453 -0.112453  7.887547 -0.112453  ... -0.112453 -0.112453
    4         8.960279 -0.039721 -0.039721 -0.039721  ... -0.039721 -0.039721
    ...            ...       ...       ...       ...  ...       ...       ...
    9996      9.907609 -0.092391 -0.092391  4.907609  ... -0.092391 -0.092391
    9997     -0.000604 -0.000604 -0.000604 -0.000604  ... -0.000604 -0.000604
    9998     -0.135333 -0.135333 -0.135333 -0.135333  ... -0.135333 -0.135333
    9999     -0.050322 -0.050322 -0.050322 -0.050322  ... -0.050322 -0.050322
    10000    -0.295894 -0.295894 -0.295894  7.704106  ... -0.295894 -0.295894

    anime_id     48413     48414     48417     48418     48426     48438  \
    user_id
    0        -0.015700 -0.015700 -0.015700 -0.015700 -0.015700 -0.015700
    1        -0.046028 -0.046028 -0.046028 -0.046028 -0.046028 -0.046028
    2        -0.021538 -0.021538 -0.021538 -0.021538 -0.021538 -0.021538
    3        -0.112453 -0.112453 -0.112453 -0.112453 -0.112453 -0.112453
    4        -0.039721 -0.039721 -0.039721 -0.039721 -0.039721 -0.039721
    ...            ...       ...       ...       ...       ...       ...
    9996     -0.092391 -0.092391 -0.092391 -0.092391 -0.092391 -0.092391
    9997     -0.000604 -0.000604 -0.000604 -0.000604 -0.000604 -0.000604
    9998     -0.135333 -0.135333 -0.135333 -0.135333 -0.135333 -0.135333
    9999     -0.050322 -0.050322 -0.050322 -0.050322 -0.050322 -0.050322
    10000    -0.295894 -0.295894 -0.295894 -0.295894 -0.295894 -0.295894

    anime_id     48456     48488
    user_id
    0        -0.015700 -0.015700
    1        -0.046028 -0.046028
    2        -0.021538 -0.021538
    3        -0.112453 -0.112453
    4        -0.039721 -0.039721
    ...            ...       ...
    9996     -0.092391 -0.092391
    9997     -0.000604 -0.000604
    9998     -0.135333 -0.135333
    9999     -0.050322 -0.050322
    10000    -0.295894 -0.295894

    [9152 rows x 14904 columns]
```

```
# Compute cosine similarity between users
user_similarity = calculate_user_similarity(user_anime_matrix_norm)
print(f"Cosine Similarity Between Users : \n \n {user_similarity}")
```

```
    Cosine Similarity Between Users :

     [[ 1.          0.09305576 -0.00223424 ...  0.01211851  0.01505642
       0.06560285]
     [ 0.09305576  1.          0.21828911 ...  0.20844695  0.09522955
       0.14238931]
     [-0.00223424  0.21828911  1.         ...  0.13680391  0.10536153
       0.07526903]
     ...
     [ 0.01211851  0.20844695  0.13680391 ...  1.          0.04023927
       0.08108417]
     [ 0.01505642  0.09522955  0.10536153 ...  0.04023927  1.
       0.07401341]
     [ 0.06560285  0.14238931  0.07526903 ...  0.08108417  0.07401341
       1.        ]]
```

```
# Extracting user_ids as lists from user-anime matrix
user_ids = user_anime_matrix_norm.index.tolist()

# Map the cosine similarity between users indicating user_ids creating a dataframe
cosine_similarity_df = create_df_user_similarity(user_similarity, user_ids)
print(f"Map Simlarity Values with User IDs \nCosine Similarity Dataframe (Not Normalized) : \n \n {cosine_similarity_df}")
```

```
    Map Simlarity Values with User IDs
    Cosine Similarity Dataframe (Not Normalized) :

              0         1         2         3         4         5         6      \
    0      1.000000  0.093056 -0.002234  0.009318  0.052230  0.030065  0.071011
    1      0.093056  1.000000  0.218289  0.140478 -0.005436  0.032044  0.037932
    2     -0.002234  0.218289  1.000000  0.062502  0.030805 -0.002537  0.030769
    3      0.009318  0.140478  0.062502  1.000000  0.027808  0.032518  0.045617
    4      0.052230 -0.005436  0.030805  0.027808  1.000000 -0.003610  0.063819
    ...         ...       ...       ...       ...       ...       ...       ...
    9996   0.101865  0.039352  0.026364  0.065612  0.151500  0.082243  0.200947
    9997  -0.000361  0.093193 -0.000416  0.070022 -0.000591 -0.000410 -0.000986
    9998   0.012119  0.208447  0.136804  0.040372 -0.000933 -0.006378  0.011432
    9999   0.015056  0.095230  0.105362  0.178714  0.100212  0.052713  0.084422
    10000  0.065603  0.142389  0.075269  0.087509  0.042346  0.011573  0.125302

             7         8      9      ...     9991      9992      9993      9994   \
    0      0.054155 -0.001423    0.0  ...  0.050050  0.054175 -0.004717 -0.000719
    1      0.124798  0.041753    0.0  ...  0.134206  0.072622  0.037030  0.065578
    2      0.032660  0.033289    0.0  ...  0.114930  0.034516  0.021280  0.085743
    3      0.177407  0.042551    0.0  ...  0.187048  0.065185  0.099097 -0.001991
    4      0.026512  0.027984    0.0  ...  0.074326  0.029989  0.008298 -0.001179
    ...         ...       ...    ...  ...       ...       ...       ...       ...
    9996   0.040692  0.047437    0.0  ...  0.141451  0.055363 -0.001840 -0.001794
    9997  -0.000516 -0.000265    0.0  ... -0.002062 -0.000555 -0.000877 -0.000134
    9998   0.018135  0.068240    0.0  ...  0.091429  0.043719  0.037474  0.036825
    9999   0.069914  0.045845    0.0  ...  0.146286  0.064617  0.021102 -0.001284
    10000  0.040113  0.038205    0.0  ...  0.115638  0.080777  0.002528  0.013879

             9995      9996      9997      9998      9999      10000
    0     -0.002845  0.101865 -0.000361  0.012119  0.015056  0.065603
    1      0.035319  0.039352  0.093193  0.208447  0.095230  0.142389
    2      0.056387  0.026364 -0.000416  0.136804  0.105362  0.075269
    3      0.073232  0.065612  0.070022  0.040372  0.178714  0.087509
    4      0.040207  0.151500 -0.000591 -0.000933  0.100212  0.042346
    ...         ...       ...       ...       ...       ...       ...
    9996   0.031527  1.000000 -0.000900  0.012448  0.132112  0.108211
    9997  -0.000529 -0.000900  1.000000 -0.001045  0.114496  0.048095
    9998   0.011411  0.012448 -0.001045  1.000000  0.040239  0.081084
    9999  -0.005077  0.132112  0.114496  0.040239  1.000000  0.074013
    10000  0.051135  0.108211  0.048095  0.081084  0.074013  1.000000

    [9152 rows x 9152 columns]
```

```
# Normalize the cosine similarity values as they will be used to find weighted score for anime in recommedation
normalized_user_similarity = normalize_user_similarity(user_similarity)
print(f"Similarity Between Users (Normalized) : \n \n {normalized_user_similarity}")
```

```
    Similarity Between Users (Normalized) :

     [[1.         0.16852384 0.08116305 ... 0.09432149 0.09701493 0.14335533]
```

```
 [0.16852384 1.         0.28333635 ... 0.27431317 0.17051675 0.21375228]
 [0.08116305 0.28333635 1.         ... 0.20863165 0.17980563 0.15221717]
 ...
 [0.09432149 0.27431317 0.20863165 ... 1.         0.12010228 0.15754842]
 [0.09701493 0.17051675 0.17980563 ... 0.12010228 1.         0.15106603]
 [0.14335533 0.21375228 0.15221717 ... 0.15754842 0.15106603 1.        ]]
```

```
# Map the normalized similarity between users indicating user_ids creating a dataframe
cosine_weights_df = create_df_norm_user_similarity(normalized_user_similarity, user_ids)
print(f"Map (Normalized) Simlarity Values with User IDs \nCosine Similarity Dataframe (Normalized) : \n \n {cosine_weights_df}")
```

```
Map (Normalized) Simlarity Values with User IDs
Cosine Similarity Dataframe (Normalized) :

              0         1         2         3         4         5         6      \
0      1.000000  0.168524  0.081163  0.091754  0.131095  0.110774  0.148314
1      0.168524  1.000000  0.283336  0.212000  0.078228  0.112589  0.117987
2      0.081163  0.283336  1.000000  0.140513  0.111453  0.080885  0.111420
3      0.091754  0.212000  0.140513  1.000000  0.108706  0.113024  0.125033
4      0.131095  0.078228  0.111453  0.108706  1.000000  0.079902  0.141720
...         ...       ...       ...       ...       ...       ...       ...
9996   0.176600  0.119289  0.107381  0.143364  0.222105  0.158611  0.267437
9997   0.082881  0.168649  0.082830  0.147407  0.082669  0.082836  0.082307
9998   0.094321  0.274313  0.208632  0.120224  0.082356  0.077365  0.093692
9999   0.097015  0.170517  0.179806  0.247054  0.175085  0.131538  0.160609
10000  0.143355  0.213752  0.152217  0.163439  0.122034  0.093822  0.198086

             7         8         9      ...     9991      9992      9993   \
0      0.132860  0.081907  0.083211  ...  0.129097  0.132878  0.078887
1      0.197625  0.121490  0.083211  ...  0.206250  0.149791  0.117160
2      0.113153  0.113730  0.083211  ...  0.188578  0.114856  0.102720
3      0.245856  0.122221  0.083211  ...  0.254695  0.142972  0.174063
4      0.107518  0.108867  0.083211  ...  0.151353  0.110705  0.090819
...         ...       ...       ...  ...       ...       ...       ...
9996   0.120518  0.126701  0.083211  ...  0.212892  0.133968  0.081525
9997   0.082738  0.082969  0.083211  ...  0.081321  0.082702  0.082407
9998   0.099837  0.145773  0.083211  ...  0.167033  0.123292  0.117567
9999   0.147308  0.125242  0.083211  ...  0.217325  0.142452  0.102557
10000  0.119986  0.118237  0.083211  ...  0.189227  0.157267  0.085529

            9994      9995      9996      9997      9998      9999      10000
0      0.082552  0.080603  0.176600  0.082881  0.094321  0.097015  0.143355
1      0.143332  0.115591  0.119289  0.168649  0.274313  0.170517  0.213752
2      0.161819  0.134907  0.107381  0.082830  0.208632  0.179806  0.152217
3      0.081386  0.150349  0.143364  0.147407  0.120224  0.247054  0.163439
4      0.082130  0.120073  0.222105  0.082669  0.082356  0.175085  0.122034
...         ...       ...       ...       ...       ...       ...       ...
9996   0.081567  0.112115  1.000000  0.082387  0.094624  0.204331  0.182418
9997   0.083089  0.082726  0.082387  1.000000  0.082254  0.188180  0.127304
9998   0.116972  0.093673  0.094624  0.082254  1.000000  0.120102  0.157548
9999   0.082034  0.078557  0.204331  0.188180  0.120102  1.000000  0.151066
10000  0.095936  0.130091  0.182418  0.127304  0.157548  0.151066  1.000000

[9152 rows x 9152 columns]
```

For the purpose of demonstrating the working of functions, we select one user_id and show all the evaluation for better understanding of the process. Later we compute all these values for all the users in a loop.

```
selected_user_id = 1 # Select a random user id to demonstrate the process for all the users
similarity_threshold = 0.1 # Set the similarity threshold to filter the similar users
top_N_similar = 15 # Set the total number of similar users to find

# Find similar users for particular user_id
similar_users = find_similar_users(cosine_similarity_df, selected_user_id,
                      top_N_similar, similarity_threshold)

print(f"Similar users for User ID: {selected_user_id} (Based on sorted similarity scores) ")
print()
print(similar_users)
```

```
Similar users for User ID: 1 (Based on sorted similarity scores)

9511    0.454667
9096    0.432448
1137    0.431257
3704    0.425074
7326    0.421491
2139    0.414810
7065    0.414104
1899    0.413951
3326    0.407948
832     0.401282
217     0.396570
7100    0.394617
9885    0.391612
3747    0.387095
3210    0.385602
Name: 1, dtype: float64
```

```
# Find already watched animes by selected user
user_watched_anime_ids = already_watched_animes(train_df, selected_user_id)

print(f"Already Watched Animes By User ID: {selected_user_id} \n {user_watched_anime_ids}")

similar_user_ids = similar_users.index.tolist() # Get similar user IDs

# Find watched animes by similar users
similar_users_watched_anime_ids = similar_users_watched_animes(train_df,similar_user_ids )

print(f"Already Watched Animes By similar users of User ID: {selected_user_id}")
print(similar_users_watched_anime_ids)
```

```
Already Watched Animes By User ID: 1
 [3588, 39940, 16894, 22535, 31240, 38408, 35849, 28171, 40456, 41487, 41488, 19, 20, 21, 22, 32282, 20507, 9253, 1575, 37430, 8246, 1604, 33352, 37450, 37965, 28755, 38483, 2144, 36456, 11371, 30831, 37999, 16498, 41587, 40052, 2167, 37497, 4224, 5114, 26243, 37521, 154, 39587, 32935, 25777, 35507, 28851, 22199, 9919, 199, 1735, 33486, 28891, 42203, 42205, 18679, 24833, 3609
Already Watched Animes By similar users of User ID: 1
[1, 12293, 34822, 6, 22535, 14345, 8, 30727, 28677, 16, 36882, 19, 20, 21, 22, 38935, 22547, 20507, 28701, 30, 30749, 32, 33, 36896, 10271, 47, 8246, 47160, 4155, 47164, 32827, 41025, 34881, 12355, 67, 68, 18507, 32843, 12365, 24655, 38992, 38993, 36946, 28755, 36949, 4181, 34902, 2142, 2144, 32867, 101, 4197, 20583, 39017, 30831, 16498, 39026, 34933, 34934, 2167, 121, 41084,
```

```
# Filter the anime IDs watched by similar users that are not in particular user watch list
filtered_sim_users_watched_anime_ids = list(set(similar_users_watched_anime_ids) - set(user_watched_anime_ids))


# Filter the user anime matrix considering animes watched by user and similar users of that particular user
user_anime_matrix_filtered_sim = filter_user_anime_matrix(user_anime_matrix, user_watched_anime_ids,
                             filtered_sim_users_watched_anime_ids, similar_user_ids)

print(f"Filtered User-Anime Matrix: \n {user_anime_matrix_filtered_sim}")
```

```
Filtered User-Anime Matrix:
 anime_id  1      12293  34822  6      14345  8      30727  28677  16     \
user_id
9511        0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
```

```
9096        0.0    0.0    6.0    0.0    0.0    0.0    0.0    0.0    0.0
1137        0.0    0.0    0.0    0.0    0.0    0.0    0.0    7.0    0.0
3704        0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
7326        0.0    0.0    8.0    0.0    6.0    0.0    0.0    0.0    0.0
2139        0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
7065        0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1899        0.0    0.0    0.0    8.0    8.0    0.0    0.0    6.0    0.0
3326        0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
832         8.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
217         0.0    0.0    7.0    0.0    0.0    0.0    0.0    0.0    0.0
7100        0.0    7.0    0.0    9.0    0.0    0.0    0.0    0.0    0.0
9885        7.0    0.0    8.0    0.0    0.0    0.0    0.0    0.0    0.0
3747        7.0    0.0    0.0    5.0    0.0    0.0    0.0    0.0    0.0
3210        0.0    0.0    7.0    0.0    0.0    0.0    0.0    0.0    0.0

anime_id  36882  ...  36838  40936  38889  2025   10218  42984  34798  4081   \
user_id          ...
9511        0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9096        0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1137        0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3704        0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
7326        1.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2139        0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
7065        0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1899        0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3326        8.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
832         0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
217         0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
7100        7.0  ...    0.0   10.0    0.0    0.0    0.0    0.0    0.0    0.0
9885        0.0  ...    0.0    0.0    7.0    0.0    0.0    0.0    0.0    0.0
3747        0.0  ...    9.0    0.0    0.0    0.0    7.0    0.0    0.0    0.0
3210        0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

anime_id  30709  43007
user_id
9511        0.0    0.0
9096        0.0    0.0
1137        0.0    0.0
3704        0.0    0.0
7326        0.0    0.0
2139        0.0    0.0
7065        0.0    0.0
1899        0.0    0.0
3326        0.0    0.0
832         0.0    0.0
217         0.0    0.0
7100        0.0    0.0
9885        0.0    0.0
3747        5.0    0.0
3210        0.0    0.0

[15 rows x 736 columns]
```

```python
top_N_value = 15 # Set the number of recommended items to return

print(f"Top - N Recommendations")
print()
# Calculate top N recommendations for selected user
top_N_recommendations = calculate_top_N_recommendations(selected_user_id,user_anime_matrix_filtered_sim,
                                                         cosine_weights_df,
                                   similar_user_ids, top_N_value, printRecommendation=True)
```

```
Top - N Recommendations

Top-15 Anime Recommendations for User ID = 1
     anime_id  weighted_average_score  \
0       38524                5.679343
1       38000                5.542302
2       30276                5.099950
3       20583                4.805786
4       23273                4.489617
..        ...                     ...
731     29787                0.000000
732     16742                0.000000
733     31433                0.000000
734      5690                0.000000
735     43007                0.000000

                                                Name  Score  \
0                   Shingeki no Kyojin Season 3 Part 2   9.10
1                                   Kimetsu no Yaiba   8.62
2                                     One Punch Man   8.57
3                                          Haikyuu!!   8.53
4                             Shigatsu wa Kimi no Uso   8.74
..                                               ...    ...
731                      Gochuumon wa Usagi Desu ka??   7.82
732  Watashi ga Motenai no wa Dou Kangaetemo Omaera...   7.04
733      Ginga Eiyuu Densetsu: Die Neue These - Kaikou   7.70
734                           Nodame Cantabile: Finale   8.27
735       Osananajimi ga Zettai ni Makenai Love Comedy   6.51

                                              Genres  \
0    Action, Drama, Fantasy, Military, Mystery, Sho...
1    Action, Demons, Historical, Shounen, Supernatural
2    Action, Sci-Fi, Comedy, Parody, Super Power, S...
3               Comedy, Sports, Drama, School, Shounen
4               Drama, Music, Romance, School, Shounen
..                                               ...
731                          Slice of Life, Comedy
732                  Slice of Life, Comedy, School
733         Action, Drama, Military, Sci-Fi, Space
734                  Comedy, Josei, Music, Romance
735                 Harem, Comedy, Romance, School

                                        English name  \
0                    Attack on Titan Season 3 Part 2
1                      Demon Slayer:Kimetsu no Yaiba
2                                      One Punch Man
3                                           Haikyu!!
4                                   Your Lie in April
..                                               ...
731                          Is the Order a Rabbit??
732  WataMote:No Matter How I Look At It, It's You ...
733       Legend of the Galactic Heroes:Die Neue These
734                                          Unknown
735                                          Unknown

                Japanese name  \
0        進撃の巨人 Season3 Part.2
1                        鬼滅の刃
```

```python
# Set the rating threshold to consider that user liked the anime
rating_threshold = 0.5

# Evaluate the performance of recommendation for particular user
precision_at_N, recall_at_N, f1_score_at_N = evaluate_metrics_user(test_df,
```

```
                                                                   selected_user_id,
                                                                   rating_threshold, top_N_recommendations)

# Evaluate averate precision at N
average_precision_at_N = evaluate_average_precision_at_N(test_df, selected_user_id,
                                    rating_threshold, top_N_recommendations)

print(f"Evaluation Metrics for User ID: {selected_user_id}")
print(f"Precision@{top_N_value}: {precision_at_N}")
print(f"Recall@{top_N_value}: {recall_at_N}")
print(f"F1_Score@{top_N_value}: {f1_score_at_N}")
print(f"Average Precision@{top_N_value}: {average_precision_at_N}")
```

```
Evaluation Metrics for User ID: 1
Precision@15: 0.035326086956521736
Recall@15: 1.1923076923076923
F1_Score@15: 0.06861910437595777
Average Precision@15: 0.40030256468242426
```

```
# Set the total average precision at N value 0, it will be used to calculate Mean Average Precision@MAP
total_average_precision_at_N = 0

# Get unique user IDs in train data
train_df_user_ids = train_df['user_id'].unique()
total_users = len(train_df_user_ids) # Set the total number of users
print(f"Total Users in Train Dataframe: {total_users}")

similarity_threshold = 0.1 # Set the similarity threshold to filter the similar users
top_N_similar = 15 # Set the total number of similar users to find
top_N_value = 15 # Set the number of recommended items to return
rating_threshold = 0.5 # Set the rating threshold to consider that user liked the anime

for index, selected_user_id in enumerate(train_df_user_ids):
    print(f"Iteration Number {index + 1}: User ID - {selected_user_id}")
    # Find similar users for particular user_id
    similar_users = find_similar_users(cosine_similarity_df, selected_user_id,
                   top_N_similar, similarity_threshold)

    # Find already watched animes by selected user
    user_watched_anime_ids = already_watched_animes(train_df, selected_user_id)

    similar_user_ids = similar_users.index.tolist() # Get similar user IDs

    # Find watched animes by similar users
    similar_users_watched_anime_ids = similar_users_watched_animes(train_df,similar_user_ids )


    # Filter the anime IDs watched by similar users that are not in particular user watch list
    filtered_sim_users_watched_anime_ids = list(set(similar_users_watched_anime_ids) - set(user_watched_anime_ids))


    # Filter the user anime matrix considering animes watched by user and similar users of that particular user
    user_anime_matrix_filtered_sim = filter_user_anime_matrix(user_anime_matrix, user_watched_anime_ids,
                                 filtered_sim_users_watched_anime_ids, similar_user_ids)

    # Calculate top N recommendations for selected user
    top_N_recommendations = calculate_top_N_recommendations(selected_user_id,user_anime_matrix_filtered_sim,  cosine_weights_df,
                                       similar_user_ids, top_N_value)

    # Evaluate the average precision at N of recommendation for particular user
    average_precision_at_N = evaluate_average_precision_at_N(test_df, selected_user_id,
                                    rating_threshold, top_N_recommendations)
    # Add to total scores
    total_average_precision_at_N += average_precision_at_N
```

```
Iteration Number 5003: User ID - 6187
Iteration Number 5004: User ID - 6132
Iteration Number 5005: User ID - 5535
Iteration Number 5006: User ID - 1283
Iteration Number 5007: User ID - 721
Iteration Number 5008: User ID - 5190
Iteration Number 5009: User ID - 2255
Iteration Number 5010: User ID - 7236
Iteration Number 5011: User ID - 1416
Iteration Number 5012: User ID - 5441
Iteration Number 5013: User ID - 4958
Iteration Number 5014: User ID - 3419
Iteration Number 5015: User ID - 485
Iteration Number 5016: User ID - 7898
Iteration Number 5017: User ID - 2460
Iteration Number 5018: User ID - 3185
Iteration Number 5019: User ID - 7580
Iteration Number 5020: User ID - 8306
Iteration Number 5021: User ID - 4756
Iteration Number 5022: User ID - 6432
Iteration Number 5023: User ID - 7109
Iteration Number 5024: User ID - 2647
Iteration Number 5025: User ID - 7978
Iteration Number 5026: User ID - 288
Iteration Number 5027: User ID - 5539
Iteration Number 5028: User ID - 7911
Iteration Number 5029: User ID - 6054
Iteration Number 5030: User ID - 1708
Iteration Number 5031: User ID - 8277
Iteration Number 5032: User ID - 160
Iteration Number 5033: User ID - 4813
Iteration Number 5034: User ID - 9339
Iteration Number 5035: User ID - 6032
Iteration Number 5036: User ID - 5493
Iteration Number 5037: User ID - 8951
Iteration Number 5038: User ID - 9416
Iteration Number 5039: User ID - 2039
Iteration Number 5040: User ID - 4636
Iteration Number 5041: User ID - 9853
Iteration Number 5042: User ID - 187
Iteration Number 5043: User ID - 1531
Iteration Number 5044: User ID - 5006
Iteration Number 5045: User ID - 344
Iteration Number 5046: User ID - 5992
Iteration Number 5047: User ID - 3549
Iteration Number 5048: User ID - 5871
Iteration Number 5049: User ID - 6937
Iteration Number 5050: User ID - 8591
Iteration Number 5051: User ID - 164
Iteration Number 5052: User ID - 1924
Iteration Number 5053: User ID - 3289
Iteration Number 5054: User ID - 1054
Iteration Number 5055: User ID - 851
Iteration Number 5056: User ID - 4858
Iteration Number 5057: User ID - 6636
Iteration Number 5058: User ID - 8192
Iteration Number 5059: User ID - 8596
Iteration Number 5060: User ID - 8057
Iteration Number 5061: User ID - 4908
```

```
mean_average_precision_at_N = total_average_precision_at_N / total_users

print("Performance Evaluation of User Based Collaborative Filtering")
print(f"Evaluation Metrics")
print(f"Mean Average Precision@{top_N_value} Over {total_users} users: {mean_average_precision_at_N}")
```

```
Performance Evaluation of User Based Collaborative Filtering
Evaluation Metrics
Mean Average Precision@15 Over 9152 users: 0.2549402497078613
```

Item based Collaborative Filtering

```
def extract_weights(name, model):
    # Get the layer by name from the model
    weight_layer = model.get_layer(name)

    # Get the weights from the layer
    weights = weight_layer.get_weights()[0]

    # Normalize the weights
    weights = weights / np.linalg.norm(weights, axis=1).reshape((-1, 1))

    return weights

# Extract weights for anime embeddings
anime_weights = extract_weights('anime_embedding', model)
```

```
def find_similar_animes(name, n=10, return_dist=False, neg=False):
    try:
        anime_row = df_anime[df_anime['Name'] == name].iloc[0]
        index = anime_row['anime_id']
        encoded_index = anime_encoder.transform([index])[0]
        weights = anime_weights
        dists = np.dot(weights, weights[encoded_index])
        sorted_dists = np.argsort(dists)
        n = n + 1
        if neg:
            closest = sorted_dists[:n]
        else:
            closest = sorted_dists[-n:]
        print('Animes closest to {}'.format(name))
        if return_dist:
            return dists, closest

        SimilarityArr = []

        for close in closest:
            decoded_id = anime_encoder.inverse_transform([close])[0]
            anime_frame = df_anime[df_anime['anime_id'] == decoded_id]

            anime_name = anime_frame['Name'].values[0]
            english_name = anime_frame['English name'].values[0]
            name = english_name if english_name != "UNKNOWN" else anime_name
            genre = anime_frame['Genres'].values[0]
            Synopsis = anime_frame['Synopsis'].values[0]
            similarity = dists[close]
            similarity = "{:.2f}%".format(similarity * 100)
            SimilarityArr.append({"Name": name, "Similarity": similarity, "Genres": genre, "Synopsis":Synopsis})
        Frame = pd.DataFrame(SimilarityArr).sort_values(by="Similarity", ascending=False)
        return Frame[Frame.Name != name]
    except:
        print('{} not found in Anime list'.format(name))

pd.set_option('display.max_colwidth', None)
```