HW#7 - Recommendation Systems

PRASHANT TOMAR

04/09/2023

Q1 - Find 3 and Movie Preference

Answer

Initially, we will identify three users from the u.user.txt dataset that have similar characteristics to my age, gender, and occupation. The identified users will be saved in a distinct list. To convert the data into a DataFrame, we will use the same code from Chapter 2 of Programming Collective Intelligence.

Subsequently, we will loop through each user and obtain their ratings data from the u.data.txt file. For this operation and the rest of the task, we will use pandas DataFrame to filter and index data.

During each iteration, we will extract movie ratings provided by the user and save them in a different DataFrame, which will then be sorted according to the rating. We will then use the iloc function of the DataFrame to select and save the top three and last three rows in separate lists.

```python
import pandas as pd

def load_item_dataset():
    # Define column names for u.item file
    col = ['movie_id','movie_title','release_date','video_release_date','IMDb_URL','unknown','Action','Adventure','Animation','Childrens','Comedy','Crime','Documentary','Drama','Fantasy','Film_Noir','Horror','Musical','Mystery','Romance','Sci_Fi','Thriller','War','Western']

    # Read u.item file into item_dataset DataFrame
    item_dataset = pd.read_csv('u.item.txt', delimiter='|', header=None, names=col)

    return item_dataset


def find_similar_users(age, gender, occupation):
    similar_user = []
    for line in open('u.user.txt'):
        (user_id, user_age, user_gender, user_occ, user_zip) = line.split('|')
        if user_age == age and user_gender == gender and user_occ ==
occupation:
            similar_user.append(user_id)

    return similar_user


def load_rating_dataset(item_dataset):
    # Create a list of dictionaries containing rating data for each item
    rating_item_list = []
    for line in open('u.data.txt'):
        (user_id, item_id, rating, timestamp) = line.split('\t')

        # Filter item_dataset to only include the current item
        item = item_dataset[item_dataset['movie_id'] == pd.to_numeric(item_id)]

        # Add current rating data to rating_item_list as a dictionary
        rating_item = {'user_id': user_id, 'item_id': item_id, 'item_name': item.iloc[0,1], 'rating': rating, 'timestamp': timestamp}
        rating_item_list.append(rating_item)

    # Convert rating_item_list to a DataFrame
    rating_dataset = pd.DataFrame(rating_item_list)

    return rating_dataset


def find_favorite_and_least_fav_films(similar_user, rating_dataset):
    favorite_films_list = []
    least_fav_films_list = []

    for user in similar_user[0:3]:
        user_rated_data = rating_dataset[rating_dataset['user_id'] == user]
        user_rated_data = user_rated_data.sort_values('rating', ascending=False)
        user_rated_data_fav_films = user_rated_data.iloc[0:3, :]
        user_rated_data_least_fav_films = user_rated_data.iloc[-3:]
        favorite_films_list.append(user_rated_data_fav_films)
        least_fav_films_list.append(user_rated_data_least_fav_films)

    return favorite_films_list, least_fav_films_list


if __name__ == '__main__':
    # Load item dataset
    item_dataset = load_item_dataset()

    # Find similar users
    similar_user = find_similar_users('30', 'M', 'student')

    # Load rating dataset
    rating_dataset = load_rating_dataset(item_dataset)

    # Find favorite and least favorite films for each user in
   similar_user
    favorite_films_list, least_fav_films_list =
   find_favorite_and_least_fav_films(similar_user, rating_dataset)

    # Print favorite and least favorite films for each user in
   similar_user
    for ds in favorite_films_list:
        print(ds.to_string(index=False))

    for lds in least_fav_films_list:
        print(lds.to_string(index=False))
```

Listing 1: Finding users with my preferences.

Below are the result of top favorite and least favorite movies of the three similar preferences users.

Table 1: 774 User Favorite Movies

User	Movie id	Movie Title	Rating
774	180	Apocalypse Now (1979)	5
774	202	Groundhog Day (1993)	5
774	193	Right Stuff, The (1983)	5

Table 2: 869 User Favorite Movies

User	Movie id	Movie Title	Rating
869	412	Very Brady Sequel, A (1996)	5
869	151	Willy Wonka and the Chocolate Factory (1971)	5
869	127	Godfather, The (1972)	5

Table 3: 774 User Least Favorite Movies

User	Movie id	Movie Title	Rating
774	411	Nutty Professor, The (1996)	1
774	168	Monty Python and the Holy Grail (1974)	1
774	393	Mrs. Doubtfire (1993)	1

Table 4: 869 User Least Favorite Movies

User	Movie id	Movie Title	Rating
869	1061	Evening Star, The (1996)	1
869	476	First Wives Club, The (1996)	1
869	284	Tin Cup (1996)	1

Analysis

Out of all the users, I will choose user 774 as my ”substitute” because I have watched Apocalypse Now and it happens to be one of my favorite movies. After searching for the other movies that were present in different users' favorite lists, I found out that they were mainly non-realistic and fictional movies, which I am not particularly fond of.

Q2 - Most Correlated User Least Correlated User with respect to Substitute You.

Answer

To accomplish this, we will utilize the code provided in Chapter 2 of Programming Collective Intelligence. The code consists of two key functions, one that computes the distance between two users and another that identifies the top match based on this distance function. We will use both Euclidean Distance and Pearson method to calculate the distance between users and find the most correlated user. However, we will only share the output generated using Pearson method since it is the most reliable and effective method.

```python
import numpy as np
import math

def sim_distance(prefs: dict[str, dict[str, float]], person1: str,
   person2: str) -> float:
    '''
    Returns a distance-based similarity score for person1 and person2.
    '''
    shared_items = {item for item in prefs[person1] if item in prefs[
   person2]}
    if len(shared_items) == 0:
        return 0
    sum_of_squares = sum((prefs[person1][item] - prefs[person2][item])
   ** 2 for item in shared_items)
    return 1 / (1 + math.sqrt(sum_of_squares))

def sim_pearson(prefs: dict[str, dict[str, float]], person1: str,
   person2: str) -> float:
    '''
    Returns the Pearson correlation coefficient for person1 and person2
   .
    '''
    shared_items = {item for item in prefs[person1] if item in prefs[
   person2]}
    if len(shared_items) == 0:
        return 0
    n = len(shared_items)
    sum1 = sum(prefs[person1][item] for item in shared_items)
    sum2 = sum(prefs[person2][item] for item in shared_items)
    sum1_sq = sum(prefs[person1][item] ** 2 for item in shared_items)
    sum2_sq = sum(prefs[person2][item] ** 2 for item in shared_items)
    p_sum = sum(prefs[person1][item] * prefs[person2][item] for item in
    shared_items)
    num = p_sum - (sum1 * sum2) / n
    den = math.sqrt((sum1_sq - (sum1 ** 2) / n) * (sum2_sq - (sum2 **
   2) / n))
    if den == 0:
        return 0
    r = num / den
    return r

def top_matches(
    prefs: dict[str, dict[str, float]],
    person: str,
    n: int = 5,
    similarity: callable = sim_pearson,
) -> list[tuple[float, str]]:
    '''
    Returns the top matches for person from the prefs dictionary.
    Number of results and similarity function are optional params.
    '''
    scores = [(similarity(prefs, person, other), other) for other in
   prefs if other != person]
    scores.sort(reverse=True)
    return scores[:n]

def least_matches(
    prefs: dict[str, dict[str, float]],
    person: str,
    n: int = 5,
    similarity: callable = sim_pearson,
) -> list[tuple[float, str]]:
    '''
    Returns the least matches for person from the prefs dictionary.
    Number of results and similarity function are optional params.
    '''
    scores = [(similarity(prefs, person, other), other) for other in
   prefs if other != person]
    scores.sort()
    return scores[:n]


def load_movie_lens():
    # Get movie titles
    with open('u.item.txt', encoding='utf-8') as f:
        movies = {}
        for line in f:
            id, title = line.split('|')[:2]
            movies[id] = title

    # Load data
    with open('u.data.txt', encoding='utf-8') as f:
        prefs = {}
        for line in f:
            user, movie_id, rating, _ = line.split('\t')
            prefs.setdefault(user, {})
            prefs[user][movies[movie_id]] = float(rating)

    return prefs


prefs = load_movie_lens()

list_user = top_matches(prefs, '774', n=5)
print(list_user)

list_user = top_matches(prefs, '774', n=5, similarity=sim_distance)
print(list_user)

list_user = least_matches(prefs, '774', n=5)
print(list_user)

list_user = least_matches(prefs, '774', n=5, similarity=sim_distance)
print(list_user)
```

Listing 2: Finding users with my preferences.

It can be observed that I have modified the topMatch function into a LeastMatch function by altering the sorting code to maintain the order instead of reversing it. I will use both the topMatch and leastMatch functions, with user 774 as my substitute, to identify the users that are most and least correlated to my substitute.

The results for the top match and least match users of my substitute are provided below.

Table 5: Top Match Correlated User

User	Correlation Result
428	1.0000000000000033
300	1.0000000000000007
170	1.0000000000000007
909	1.0
762	1.0

Table 6: Top Match Correlated User

User	**Correlation Result**
284	-1.0000000000000007
107	-1.0
155	-1.0
240	-1.0
252	-1.0

Analysis

The following users have been identified as being more similar to my substitute, and their movie preferences have been analyzed. Although some of the movie genres aligned with my interests, others did not. I attempted to use Euclidean Distance to find correlated users as well, and while some users matched with Pearson, not all of them did.

Q3 - Compute Ratings, Top 5 and Bottom 5 recommendations for movies

Answer

We will utilize the code provided in Chapter 2 of the book Programming Collective Intelligence. Specifically, we will employ the getRecommendation function, which utilizes the Pearson function to identify the movies that my substitute should watch based on the closest distance.

The implementation of the current process is presented below.

```python
from numpy import sqrt

def distance_similarity(prefs, p1, p2):
    si = {}
    for item in prefs[p1]:
        if item in prefs[p2]:
            si[item] = 1
    if len(si) == 0:
        return 0
    sum_of_squares = sum([pow(prefs[p1][item] - prefs[p2][item], 2) for item in
                          prefs[p1] if item in prefs[p2]])
    return 1 / (1 + sqrt(sum_of_squares))

def pearson_similarity(prefs, p1, p2):
    si = {}
    for item in prefs[p1]:
        if item in prefs[p2]:
            si[item] = 1
    if len(si) == 0:
        return 0
    n = len(si)
    sum1 = sum([prefs[p1][it] for it in si])
    sum2 = sum([prefs[p2][it] for it in si])
    sum1Sq = sum([pow(prefs[p1][it], 2) for it in si])
    sum2Sq = sum([pow(prefs[p2][it], 2) for it in si])
    pSum = sum([prefs[p1][it] * prefs[p2][it] for it in si])
    num = pSum - sum1 * sum2 / n
    den = sqrt((sum1Sq - pow(sum1, 2) / n) * (sum2Sq - pow(sum2, 2) / n
))
    if den == 0:
        return 0
    r = num / den
    return r

```

```python
def get_recommendations(prefs, person, similarity=pearson_similarity):
    totals = {}
    sim_sums = {}
    for other in prefs:
        if other == person:
            continue
        sim = similarity(prefs, person, other)
        if sim <= 0:
            continue
        for item in prefs[other]:
            if item not in prefs[person] or prefs[person][item] == 0:
                totals.setdefault(item, 0)
                totals[item] += prefs[other][item] * sim
                sim_sums.setdefault(item, 0)
                sim_sums[item] += sim
    rankings = [(total / sim_sums[item], item) for (item, total) in
                totals.items()]
    rankings.sort()
    rankings.reverse()
    return rankings

def load_movie_lens():
    movies = {}
    for line in open('u.item.txt'):
        (id, title) = line.split('|')[0:2]
        movies[id] = title
    prefs = {}
    for line in open('u.data.txt'):
        (user, movieid, rating, ts) = line.split('\t')
        prefs.setdefault(user, {})
        prefs[user][movies[movieid]] = float(rating)
    return prefs

prefs = load_movie_lens()
list_user = get_recommendations(prefs, '774')

print(list_user[0:5])

print(list_user[-5:])
```

Listing 3: Recommendation movies code

After receiving the list of suggested movies, we will store the recommendations in a list and select the top 5 and bottom 5 movies. This will allow us to display the movies that are highly recommended as well as those that are least recommended. The results of this process are presented below.

Table 7: Top 5 recommended movies

Movies	**Rating**
Prefontaine (1997)	5.0
Tough and Deadly (1995)	5.0
They Made Me a Criminal (1939)	5.0
Star Kid (1997)	5.0
Someone Else's America (1995)	5.0

Table 8: Least 5 recommended movies

Movies	**Rating**
Amityville: A New Generation (1993)	1.0
Amityville Curse, The (1990)	1.0
Amityville 3-D (1983)	1.0
Amityville 1992: It's About Time	1.0
3 Ninjas: High Noon At Mega Mountain	1.0

Analysis

Upon reviewing the top 5 recommended movies, I searched for some of them and found that most of them seem intriguing, and I assume they will be good. However, after examining the least recommended movies, I found that none of them appear interesting to me at all.

Q4 - Correlated Movies.

Answer

We will utilize the code we used in Q1 and add a new function that will transform the movie data into a format similar to the user data. This will enable us to compute the distance between movies and identify the movies that are most correlated.

The implementation of this process is presented below.

```python
import numpy as np

def sim_distance(prefs, p1, p2):
    si = {}
    for item in prefs[p1]:
        if item in prefs[p2]:
            si[item] = 1
    if not si:
        return 0
    sum_of_squares = sum([pow(prefs[p1][item] - prefs[p2][item], 2) for
    item in si])
    return 1 / (1 + np.sqrt(sum_of_squares))

def sim_pearson(prefs, p1, p2):
    si = {}
    for item in prefs[p1]:
        if item in prefs[p2]:
            si[item] = 1
    if not si:
        return 0
    n = len(si)
    sum1 = sum([prefs[p1][it] for it in si])
    sum2 = sum([prefs[p2][it] for it in si])
    sum1_sq = sum([pow(prefs[p1][it], 2) for it in si])
    sum2_sq = sum([pow(prefs[p2][it], 2) for it in si])
    p_sum = sum([prefs[p1][it] * prefs[p2][it] for it in si])
    num = p_sum - (sum1 * sum2 / n)
    den = np.sqrt((sum1_sq - pow(sum1, 2) / n) * (sum2_sq - pow(sum2,
  2) / n))
    if den == 0:
        return 0
    r = num / den
    return r

def top_matches(prefs, person, n=5, similarity=sim_pearson):
    scores = [(similarity(prefs, person, other), other) for other in
```

```
   prefs if other != person]
    scores.sort(reverse=True)
    return scores[:n]

def least_matches(prefs, person, n=5, similarity=sim_pearson):
    scores = [(similarity(prefs, person, other), other) for other in
   prefs if other != person]
    scores.sort()
    return scores[:n]

def transform_prefs(prefs):
    result = {}
    for person in prefs:
        for item in prefs[person]:
            result.setdefault(item, {})
            result[item][person] = prefs[person][item]
    return result

def load_movie_lens():
    movies = {}
    with open('u.item.txt', encoding='ISO-8859-1') as f:
        for line in f:
            (id, title) = line.split('|')[0:2]
            movies[id] = title
    prefs = {}
    with open('u.data.txt', encoding='ISO-8859-1') as f:
        for line in f:
            (user, movieid, rating, ts) = line.split('\t')
            prefs.setdefault(user, {})
            prefs[user][movies[movieid]] = float(rating)
    return prefs

prefs = load_movie_lens()
mod_prefs = transform_prefs(prefs)

list_movies = top_matches(mod_prefs, 'Silence of the Lambs, The (1991)'
   , n=5)
print(list_movies)

list_movies = least_matches(mod_prefs, 'Silence of the Lambs, The
   (1991)', n=5)
print(list_movies)

list_movies = top_matches(mod_prefs, 'Pulp Fiction (1994)', n=5)
print(list_movies)

list_movies = least_matches(mod_prefs, 'Pulp Fiction (1994)', n=5)
```

```
print(list_movies)
```

Listing 4: Further tweet analysis

As you can see, I have created a new function called leastMatch by modifying the topMatch function and removing the reverse functionality. This function allows us to identify the least correlated movies and most correlated movies. Once we have obtained the list of movies from these functions, we will display them. Additionally, we are sending n=5 into the function to specify the number of data that we require.

The result of this process is presented below.

Table 9: Most correlated movies of my Favorite Movies

Movies	**Correlation Values**
Mrs. Dalloway (1997)	1.0
One Night Stand (1997)	1.0
Smile Like Yours, A (1997)	1.0
Year of the Horse (1997)	1.0
Total Eclipse (1995)	1.0

Table 10: Least correlated movies of my Favorite Movies

Movies	**Correlation Values**
Children of the Revolution (1996)	-1.0
My Favorite Season (1993)	-1.0
Half Baked (1998)	-1.0
Broken English (1996)	-1.0
Caro Diario (Dear Diary) (1994)	-1.0

Table 11: Most correlated movies of my Least Favorite Movies

Movies	**Correlation Values**
’Maya Lin: A Strong Clear Vision (1994)	1.0
’Crossfire (1947)	1.0
’Zeus and Roxanne (1997)	1.0
’Wedding Gift, The (1994)	1.0
’Two Deaths (1995)	1.0

Table 12: Least correlated movies of my Least Favorite Movies

Movies	**Correlation Values**
'Mr. Magoo (1997)	-1.0
'Telling Lies in America (1997)	-1.0
'American Dream (1990)	-1.0
Anna (1996)	-1.0
'Bewegte Mann, Der (1994)	-1.0

Analysis

I am satisfied with the results as the recommended movies are of my liking and interest. I researched the movies on the internet and read their plot to confirm my preferences.

I am not a fan of movies that contain unrealistic elements, such as Godzilla, and this can be seen in the least favorite most correlated table where there is a Mr. Magoo movie that is related to fictional stories that I am not interested in. This demonstrates that the Pearson method of calculating the distance is effective in determining the correlation between objects.

Q6 - Rank all 1682 movies

Answer

The process for this task involves loading both the movie data and movie rating data, similarly to the previous example. After loading the data, each movie will be looped through, and the total count of ratings and average rating will be determined.

Once all the necessary data is obtained, a separate DataFrame will be created to perform further calculations on. A new column will be added to this DataFrame, which calculates the rating weight by multiplying the count of votes with the average rating.

Implementation of the current process is shared below,

```
import pandas as pd
from pandas import DataFrame
import numpy as np

col = ['movie_id','movie_title','release_date','video_release_date','
    IMDb_URL','unknown','Action','Adventure','Animation','Childrens','
    Comedy','Crime','Documentary','Drama','Fantasy','Film_Noir','Horror'
    ,'Musical','Mystery','Romance','Sci_Fi','Thriller','War','Western']
item_dataset = pd.read_csv('u.item.txt', delimiter = "|", header=None,
    names=col)

rating_item_list = []
for line in open('u.data.txt'):
    (user_id, item_id, rating, timestamp) = line.split('\t')

    rating_item = {'user_id': user_id, 'item_id': item_id,
                   'rating': rating, 'timestamp': timestamp
                   }
    rating_item_list.append(rating_item)

rating_dataset = DataFrame(rating_item_list)
rating_dataset['rating'] = pd.to_numeric(rating_dataset['rating'])

rating_list = []
for movie in item_dataset.iterrows():
    ratings = rating_dataset[rating_dataset['item_id'] == str(movie[1][
    'movie_id'])]
    average_rating = round(np.sum(ratings.iloc[:, 2]) / len(ratings.
    index), 2)
    rating_count = len(ratings.index)

    rating_object = {
        'movie_id'  : movie[1]['movie_id'],
        'movie_title': movie[1]['movie_title'],
        'average_rating': average_rating,
        'rating_count': rating_count,
        'weighted_rating'  : average_rating * rating_count
    }
    rating_list.append(rating_object)

rating_dataset = DataFrame(rating_list)

rating_dataset = rating_dataset.sort_values('weighted_rating',
    ascending=False)
rating_dataset.to_csv('rating_sorted_dataset.csv')
print(rating_dataset.iloc[0:10,:])
print(rating_dataset.iloc[-10:])
```

Listing 5: Ranking the movies

After loading the movie and movie rating data, we will iterate through each movie to calculate the total count of ratings and the average rating. Then, we will create a separate DataFrame and calculate the rating weight by multiplying the count of votes with the average rating.

After calculating the rating weight, we will sort the DataFrame based on the weighted average rating column and retrieve the top and bottom 10 using the iloc function of DataFrame. Finally, we will save the ranked results in a separate file.

The results are shown below.

Table 13: Top 10 ranked movies

Movies	**Avg. Rating**	**Count**	**Weighted Average**
Star Wars (1977))	4.36	583	2541.88
Fargo (1996)	4.16	508	2113.28
Return of the Jedi (1983)	4.01	507	2033.07
Contact (1997)	3.80	509	1934.20
Raiders of the Lost Ark (1981)	4.25	420	1785.00
Godfather, The (1972)	4.28	413	1767.64
English Patient, The (1996)	3.66	481	1760.46
Toy Story (1995)	3.88	452	1753.76
Silence of the Lambs, The (1991)	4.29	390	1673.10
Scream (1996)	3.44	478	1644.32

Table 14: Least 10 ranked movies

Movies	**Avg. Rating**	**Count**	**Weighted Average**
Hungarian Fairy Tale, A (1987)	1.0	1	1.0
Pharaoh's Army (1995)	1.0	1	1.0
Modern Affair, A (1995)	1.0	1	1.0
Wend Kuuni (God's Gift) (1982)	1.0	1	1.0
Touki Bouki (Journey of the Hyena) (1973)	1.0	1	1.0
Woman in Question, The (1950)	1.0	1	1.0
Quartier Mozart (1992)	1.0	1	1.0
Girl in the Cadillac (1995)	1.0	1	1.0
Nobody Loves Me (Keiner liebt mich) (1994)	1.0	1	1.0
Liebelei (1933)	1.0	1	1.0

Analysis

The purpose of multiplying the count with the average rating is to calculate the weighted average rating of the movies, which makes it easier to rank them. For instance, in table 13, even though The Godfather has a higher average rating of 4.28, it has only 413 user ratings compared to Fargo, which has a lower average rating of 4.16 but a higher count of user ratings at 508. That's why Fargo is ranked higher than The Godfather.

Q7 - Rank the same 1682 movies according to today's IMDB data

Answer

To complete this task, we will start by downloading two files, basic.csv and rating.csv, from today's Imdb movies data. Then, we will use pandas' read_csv function to load these CSV files. After loading the files, we will load the u.item.txt file so that we can extract records from the basic.csv dataset for the movies in the u.item.txt file.

We will use the DataFrame isin() function to find all the movies from the imdb movies file. Once we have identified all the movies, we will loop through each movie and use its ID (tconst) to find its rating and total vote count. We will then calculate the weighted average rating for each movie in the same way we did in Q5.

Implementation of the current process is shared below,

```python
import pandas as pd
from pandas import DataFrame
import numpy as np

load_movie_data = pd.read_csv('basic.tsv', delimiter="\t")
load_rating_data = pd.read_csv('rating.tsv', delimiter="\t")

col = ['movie_id', 'movie_title', 'release_date', 'video_release_date',
    'IMDb_URL', 'unknown', 'Action', 'Adventure', 'Animation', '
   Childrens',
       'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film_Noir
   ', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci_Fi', 'Thriller',
   'War', 'Western']
item_dataset = pd.read_csv('u.item.txt', delimiter="|", header=None,
   names=col)

list_movies = []

list_titles = item_dataset.iloc[:, 1]

filtered_movies = load_movie_data[load_movie_data['primaryTitle'].isin(
   list_titles)]

print(filtered_movies)

list_rating = []

for movie in filtered_movies.iterrows():
    filtered_rating = load_rating_data[load_rating_data['tconst'] ==
   movie[1]['tconst']]
    if len(filtered_rating.index) > 0:
        avg_rating = filtered_rating.iloc[0,1]
        num_Votes = filtered_rating.iloc[0,2]
        rating_item = {
            'tconst': movie[1]['tconst'],
            'movie_title': movie[1]['primaryTitle'],
            'avg_rating': avg_rating,
            'num_votes': num_Votes,
            'weighted_rating':  avg_rating * num_Votes
        }
        list_rating.append(rating_item)

final_rating_dataset = DataFrame(list_rating)
final_rating_dataset = final_rating_dataset.sort_values('
   weighted_rating', ascending=False)
print(final_rating_dataset.iloc[0:10,:])
print(final_rating_dataset.iloc[-10:])
```

Listing 6: Ranking the movies

After calculating the weighted average rating of each movie, we sorted the dataframe based on this column and then used the iloc function to get the top and bottom 10 movies. Finally, we saved the complete ranked result in a separate file. The output of the top and bottom 10 movies is shown below.

Table 15: Top 10 ranked movies

Movies	**Avg. Rating**	**Votes**	**Weighted Average**
Batman (1989)	7.6	64	486.4
Cinderella (1950)	7.6	63	478.8
Beauty and the Beast (1991)	7.2	57	410.4
Scream 2 (1997)	8.4	21	176.4
Wes Craven's New Nightmare (1994)	8.8	20	176.0
Scream (1996)	8.5	19	161.5
Alien (1979)	7.0	22	154.0
Night of the Living Dead (1968)	8.4	18	151.2
Halloween: The Curse of Michael Myers (1995)	8.3	18	149.4
Jurassic Park (1993)	8.6	17	146.2

Table 16: Least 10 ranked movies

Movies	**Avg. Rating**	**Count**	**Weighted Average**
Akira (1988)	7.2	11	79.2
Super Mario Bros. (1993)	7.4	9	66.6
Spawn (1997)	7.1	9	63.9
Deep Rising (1998)	8.8	6	52.8
Blade Runner (1982)	7.4	7	51.8
Top Gun (1986)	7.3	7	51.1
Star Trek VI: The Undiscovered Country (1991)	7.7	6	46.2
Rumble in the Bronx (1995)	8.6	5	43.0
Return of the Jedi (1983)	8.6	5	43.0
Bride of Frankenstein (1935)	4.2	5	21.0

References

- Code Reference, `https://github.com/arthur-e/Programming-Collective-Intelligence/blob/master/chapter2/recommendations.py`
- DataFrame operations, `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html`
- Tweepy, `https://docs.tweepy.org/en/latest/`
- Numpy, `https://numpy.org/doc/stable/reference/generated/numpy.log2.html`