INLS 570, Capra | Project 2 | Spring 2021
Assigned: April 13 | Data Analysis

**Checkpoint #1: Monday, April 26, 5:00pm**
**Full project due: Monday, May 3, 5:00pm**

---

**Overview:** Write a Python program to analyze data from last.fm.

**Learning objectives:** Gain experience processing and cleaning datasets, building data structures to support analysis, and performing data analysis to answer specific questions.

**Project specification:**
For this project, you will analyze data from the on-line music service, last.fm. Specifically, we will use a dataset that was released as part of the HetRec Workshop at the 2011 ACM Conference on Recommender Systems. A zip file containing the data is made available by the Grouplens research team at the University of Minnesota. To download it, go to the page: http://grouplens.org/datasets/hetrec-2011/
and download the file: hetrec2011-lastfm-2k.zip

Unzip the file and you should get a folder with 7 items:

| **Filename** | **Notes** |
|---|---|
| artists.dat | This file contains artist ids, artist names, and other information such as URLs for artists on the last.fm site. |
| readme.txt | Important notes about the dataset and file structures. |
| user_artists.dat | This file contains lines that describe how many times a user has played songs by an artist. It contains lines with user ids (uid), artist ids (aid), and the number of times the user has played a song by that artist (weight). |
| user_friends.dat | You will need this file for question #6. |
| user_taggedartists.dat | You will need this file for question #8. |
| tags.dat | You may not need this file for this project. |
| user_taggedartists_timestamps.dat | You may not need this file for this project. |

This dataset is just a sampling of data from last.fm, but it is a good example of the type of real-world data that data analysts work with. Here are a few stats about the data files: there are 1892 user ids and 17,632 artists. There are 92,834 user -artist relationships, and 25,434 friend relationships.

*Data Analysis and Specific Queries*
To analyze this data, you will answer a specific set of questions (queries) given below. For each query, clearly comment the section of your code that addresses the question, and in your comments, provide a brief description of the approach that you took. Implement functions as appropriate.

There is no "user interface" for this project. I will run your program and it should produce output all at once for the questions below.

In your output for each question, print:

- A blank line
- A line of 40 exclamation points "!!!!!!!!"
- Another blank line
- A line with the question number and the brief description of the question (shown in *italics* below)
- Then print the output for that question.

**Checkpoint #1: Queries 1-3**

1. *Who are the top artists in terms of play counts?* Print a list of the 10 artists with the most song plays (across all users), sorted by number of song plays. For each of the top 10, print the artist name, the artist id, and the total number of song plays for that artist. See example output below.

```
1. Who are the top artists?
    Britney Spears  289    2393140
    Depeche Mode    72     1301308
    Lady Gaga       89     1291387
```

2. *What artists have the most listeners?* Print a list of the 10 artists with the highest number of users who have listened to at least one song by that artist, sorted by number of listeners. For each artist, print the artist name, artist id, and the number of distinct users who have listened to a song by that artist.

```
2. What artists have the most listeners?
    Lady Gaga       89     611
    Britney Spears  289    522
    Rihanna         288    484
```

3. *Who are the top users in terms of play counts?* Print the 10 user ids with the most song plays (across all artists), sorted by number of song plays. For each, print the user id and total number of song plays for that user.

```
3. Who are the top users?
    757      480039
    2000     468409
    1418     416349
```

*Requirement to use Pandas data objects* – For this assignment, you must use Pandas DataFrames as a *primary part* of the implementation for queries #1, #2, and #3. It is okay to use other data structures, but data frames should be a primary part of your implementation. Of course, you may wish to use DataFrames for other queries as well.

**Full project: Queries 4-7**

4. *What artists have the highest average number of plays per listener?* A previous question (#2) asked you to compute the artists with the most listeners, but only tells part of the story. If Person A only played a Britney Spears song once, and Person B played Britney Spears songs 10,000 times, they would both be counted equally in Question #2. For this question (#4), we want to compute the average number of plays per listener for each artist. This average is the number of total plays for that artist divided by the number of listeners for that artist. Print a list of the 10 artists with the highest average number of plays per listener. Include the artist name, artist id, total number of plays, total number of listeners, and the computed average number of plays per listener.

5. *What artists with at least 50 listeners have the highest average number of plays per listener?* When you finish question (#4), you may notice a problem – many of the "average number of plays" end up being based on the data from only *one* user. For example, according to my output for question #4, there is only one person who listened to the artist "Viking Quest", but that person really liked them, playing their songs 35,323 times. Since 35,323 / 1 = 35,323, Viking Quest got the highest average number of plays. Our metric from question #4 did not take into account that there are many artists that only have a few people who listen to them. To address this, in question (#5) we will require that artists have at least 50 listeners before we "trust" the average number of plays. For output, print a list of the 10 artists with the highest average number of plays per listener for artists with at least 50 listeners. Include the artist name, artist id, total number of plays, total number of listeners, and the computed average number of plays per listener. *Hint*: Depeche Mode was the top artist in my output, with an average of 4615 plays per listener.

***Advanced Functionality (Optional)***

If you complete the functionality described above for queries #1-#5 using built-in Python data structures (e.g., lists, dictionaries, tuples, etc.) you can earn up to 88 points for the assignment. To earn the remaining points, you will need to implement the additional functionality described in this section.

6. *[4 points] How similar are two artists?* There are many ways you might define "similar artists". Artists could be similar because of their musical style, the era they performed during, or based on the fans they have in common. For this question, we will define similarity based on common listeners. The *Jaccard index* is a statistic for measuring the similarity of two sets (http://en.wikipedia.org/wiki/Jaccard_index). Write a function called `artist_sim(aid1, aid2)` that takes two artist ids as arguments. The function should first compute the set of users who have listened to aid1, and the set of users who have listened to aid2. Then it should compute the Jaccard index of these two sets. The Jaccard index is the number of items in the intersection of the two sets divided by the number of items in the union of the two sets. You will need to use floating point numbers when computing the index, and you may also find it helpful to use Python's `set` data structure (http://docs.python.org/2/library/sets.html). Test your function by computing the similarity of the following pairs of artists. For each pair, print the artists names followed by the Jaccard index.

```
artist_sim(735,562)
artist_sim(735,89)
artist_sim(735,289)
artist_sim(89,289)
artist_sim(89,67)
artist_sim(67,735)
```

7. [8 points] *Recommend artists based on friends* – For this function, you will implement a feature that recommends artists to a user based on what their friends have been listening to. For example, suppose that user 1 is friends with users 2, 3, and 4 and we had the following data from user_artists.dat:

| userID | artistID | weight |
|---|---|---|
| 1 | 51 | 14 |
| 2 | 51 | 22 |
| 2 | 58 | 19 |
| 2 | 59 | 3 |
| 2 | 60 | 30 |
| 3 | 51 | 18 |
| 3 | 58 | 15 |
| 3 | 59 | 20 |
| 4 | 59 | 10 |

In this example, all users 1, 2, and 3 have all been listening to artist 51. However, users 2 and 3 have play counts for artist 58, but user 1 does not. This might indicate that users 2 and 3 have "discovered" and like artist 58, so this is a good candidate for us to recommend to user 1. We would not want to recommend artist 51 since user 1 already has play counts for that artist. For this feature, implement a function that will take a user id as input and will output a list of the top 10 artists that should be recommended to that user based on their friends' listening frequencies. For a user U with set of friends F, compute the set of candidate artists A who are artists that at least two friends have listened to but that user U has not listened to. Then, for each artist in set A, compute the average number of plays of each artist by all friends F who have listened to that artist. Rank the set of artists A according to the averages and display the top 10.

For instance, from the sample data above, consider user 1. Assume that user 1 has three friends, user 2, user 3, and user 4. Thus, U=1 and F={2, 3, 4}. Based on the user_artists.dat data, we compute the set of artists A to be {58, 59}. Note that artist 60 is not included since only one friend listens to artist 60. For each of the artists in set A, we compute the average number of plays by friends who listen to that artist. For artist 58, the average is 19+15= 34/2 = 17. For artist 59, the average is 3+20+10 = 34/3 = 11. This means we would recommend the artists in the following order: 58, 59.

The dataset includes a file user_friends.dat that indicates users who have "friended" another user. Note that the friend relationships in the user_friends.dat file are one directional (e.g., the top line of the file indicates that user 2 is friends with user 275, but that does not necessarily mean that user 275 is friends with user 2).

*Reading and Cleaning the Data*

Your program should start by reading the data in the artists.dat and user_artists.dat files described above and storing the data in Python or Pandas data structures. The .dat files are text files with fields separated by tab characters. The text is stored using "utf-8" encoding. To properly read utf-8 characters, you may need to use the "codecs" library.

I suggest an approach similar to what is shown below:

```
artists_df = pd.read_table('artists.dat', encoding="utf-8",
                           sep="\t", index_col='id')
```

When reading in ***integer*** values from the .dat files, you will need to make sure that you store them as integers in your Python (or Pandas) data structures. If you read the data into Pandas DataFrames using .read_table(), then it is likely that Pandas will automatically store the data as integers, but you will probably still need to specify `encoding="utf-8"` as shown above.

As with many real-world datasets, you may find a few problems and inconsistencies with the data. I will point out a few of these in class and can provide guidance about how to address them. For example, in the file user_taggedartists.dat there are a few records for tags that were made in the 1950s! Since this was well before last.fm existed, we will assume that these tag records are erroneous and will exclude them from our analysis. In the same file, there are records that indicate a tag was made for an artist id that does not exist in the artists.dat file. We will also exclude these from our tag data. As you work with the dataset, you may find other issues. If you run into data problems and have questions, please feel free to ask me.

*Pandas hints*

Here are a few hints that you may find helpful in using Pandas DataFrames for this assignment.

- Consider using a hierarchical index for the user_artists data:

```
user_artists_df = pd.read_table('user_artists.dat',
                          encoding="utf-8",
                          sep="\t",
                          index_col=['userID', 'artistID'])
```

- Explore the use of the DataFrame methods .merge(), .sort_values() and .iterrows()

**Grading:**
Your program will be evaluated based on its functionality, programming logic, and programming style. Functionality focuses on the question, "Does your program product the correct results?" Programming logic considers whether the approach you implemented in your code is correct (or close to correct). Programming style looks at how easy it is to understand your code – is it organized well, did you use functions appropriately, **did you include good comments**?

**How to turn in your assignment:**
Your program should be contained in a single Python file and be entirely code that you write yourself. Name your file according to the following convention:

```
youronyen_p2.py
```

Replace *youronyen* with your actual Onyen (e.g. my assignment would be `rcapra_p2.py`).

I will test your program by running it with Python 3.7, with the specified .dat files in the same directory.

Submit your file electronically through the Sakai by going to the Assignments area and finding the "P2" assignment. After you think you have submitted the assignment, I strongly recommend checking to be sure the file was uploaded correctly by clicking on it from within Sakai. Keep in mind that if I cannot access your file, I cannot grade it.

If for some reason you need to re-submit your file, you must add a version number to your filename. Sakai is configured so that it will accept up to 3 total submissions. Use the following file naming convention if you need to re-submit:

| | |
|---|---|
| Your first submission: | `youronyen_p2.py` |
| Your second submission: | `youronyen_p2_v2.py` |
| Your third submission: | `youronyen_p2_v3.py` |

Sakai is also configured with a due date and an "accept until" date. Submissions received after the due date may receive a late penalty.