INLS 490-172, Capra Project 2 Spring 2018
Assigned: March 31 Data Analysis

Checkpoint #1: Friday, April 13, 5:00pm
Full project due: Friday, April 20, 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, Due Friday, April 13, 5:00pm

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
```

Full project: Queries 4-8, Due Friday, April 20, 5:00pm

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.

6. *Do users with five or more friends listen to more songs?* To answer this question, compute the total number of song plays for all users who have five or more friends. Divide this by the total number of

users who have five or more friends to arrive at an average number of song plays for these users. Do the same for the set of users who have less than five friends. Print both numbers with clear labels.

Advanced Functionality

If you complete the functionality described above for queries #1-#6 using built-in Python data structures (e.g., lists, dictionaries, tuples, etc.) you can earn up to 86 points for the assignment. To earn the remaining points, you will need to implement the additional functionality described in this section.

7. *[5 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 names of the two artists 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)
```

8. [4 points] *Analysis of top tagged artists* – For the 10 artists with the highest overall number of tags, list: a) the first month they entered the top 10 in terms of number of tags, and b) the number of months they were in the top 10 in terms of number of tags. The list should be sorted in order of the overall number of tags (e.g., Britney Spears is first with 931 total tags). Example output is shown below.

```
Britney Spears(289):  num tags = 931
  first month in top10 = Sep 2006
  months in top10 =  25

Lady Gaga(89):  num tags = 767
  first month in top10 = Dec 2008
  months in top10 =  19
```

9. [5 points] *Use Pandas data objects* – As you implement parts of this project, you have a choice of using either Python's built-in data types (e.g., lists, dictionaries, tuples, etc.) or Pandas data objects (e.g., Series, DataFrames). On the one hand, you are probably be more familiar with Python's built-in data types. However, the Pandas data objects have powerful functions that may reduce the amount of code you need to write. To encourage you to use the Pandas objects, if you use Pandas DataFrames as a primary part of the implementation for queries #1, #2, and #3, you will earn the 5 points for this advanced functionality item. Of course, you may wish to use DataFrames for other queries as well.

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.

If you are using built-in Python data types, I suggest an approach similar to what is shown below:

```
import codecs
fp = codecs.open("artists.dat", encoding="utf-8")
fp.readline()  #skip first line of headers
for line in fp:
    line = line.strip()
    fields = line.split('\t')
    aid = int(fields[0])
    name = fields[1]
    # do other processing
```

If you are using Pandas DataFrames, 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 store them in Python lists or dictionaries, I recommend explicitly casting them as int() in your Python code (as shown above with fields[0]) so that they will not be read in as strings. If you leave them as strings, your program will probably slow down quite a bit. 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

If you choose to use Pandas DataFrames for this project, here are a few hints that you may find helpful.

- 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()

Python built-in data type hints
If you choose to use Python built-in data types (e.g., lists, dictionaries, tuples, etc.), here are a few hints that you may find helpful.

- Based on the artists.dat data, create an `aid2name` dictionary to map aids to the artist name. This will make printing out the artist names and interpreting the data much easier. For example, here are some examples from my aid2name dictionary:

  ```
  In [2]: aid2name[289]
  Out[2]: 'Britney Spears'

  In [3]: aid2name[51]
  Out[3]: 'Duran Duran'
  ```

  When you read data from the other files, you may find aids that are not in the artists.dat file. For this assignment, you should IGNORE any aids that you find in other files that are not in the artists.dat file. As you build subsequent data structures from the data in the other files, you may wish to check to see if each of the aids is in the aid2names dictionary. If not, you should ignore the aid (e.g., don't include it).

- Using the data from the users_artists.dat file, I created a dictionary called `aid2numplays` in which the keys were the aids and the values were the corresponding number of total number of play counts for that artist. I computed the play counts using methods we have discussed in class. Similarly, I created a dictionary called `uid2numplays` that maps user ids onto computed counts of the total number of plays made by each user.

- I created a data structure called `sorted_aidnumplays` using `aid2numplays` and sorted(). I did this using methods based on the examples shown in the lecture slides. I similarly created `sorted_uidnumplays` based on the `uid2numplays` dictionary. Here is an example of these data structures:

  ```
  In [12]: sorted_aidnumplays[:3]
  Out[12]: [(289, 2393140), (72, 1301308), (89, 1291387)]

  In [13]: sorted_uidnumplays[:3]
  Out[13]: [(757, 480039), (2000, 468409), (1418, 416349)]
  ```

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 3.6, 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.