CSE 6242/CX 4242: Data and Visual Analytics | Georgia Tech | Spring 2017

Homework 1: Analyzing Twitter dataset; SQLite; D3 Warmup; Gephi; OpenRefine

Due: Friday, February 3, 2017, 11:55 PM EST

Prepared by Meghna Natraj, Bhanu Verma, Fred Hohman, Kiran Sudhir,
Varun Bezzam, Chirag Tailor, Polo Chau

Submission Instructions and Important Notes:

It is important that you read the following instructions carefully and also those about the deliverables at the end of each question or you may lose points.

- Submit a single zipped file, called "HW1-{YOUR_LAST_NAME}-{YOUR_FIRST_NAME}.zip", containing all the deliverables including source code/scripts, data files, and readme. Example: 'HW1-Doe-John.zip' if your name is John Doe. **Only .zip is allowed** (no other format will be accepted)
- You may collaborate with other students on this assignment, but you must write your own code and give the explanations in your own words, and also mention the collaborators' names on T-Square's submission page. All GT students must observe the honor code. **Suspected plagiarism and academic misconduct will be reported to and directly handled** by the Office of Student Integrity (OSI). Here are some examples similar to Prof. Jacob Eisenstein's NLP course page (grading policy):
  - **OK:** discuss concepts (e.g., how cross-validation works) and strategies (e.g., use hashmap instead of array)
  - **Not OK:** several students work on one master copy together (e.g., by dividing it up), sharing solutions, or using solution from previous years or from the web.
- If you use any "*slip days*", you must write down the number of days used in the T-square submission page. For example, "Slip days used: 1". Each slip day equals 24 hours. E.g., if a submission is late for 30 hours, that counts as 2 slip days.
- At the end of this assignment, we have specified a folder structure about how to organize your files in a single zipped file. 5 points will be deducted for not following this strictly.
- We will use auto-grading scripts to grade some of your deliverables (there are hundreds of students), so it is extremely important that you strictly follow our requirements. Marks may be deducted if our grading scripts cannot execute on your deliverables.
- Wherever you are asked to write down an explanation for the task you perform, stay within the word limit or you may lose points.
- In your final zip file, please do not include any intermediate files you may have generated to work on the task, unless your script is absolutely dependent on it to get the final result (which it ideally should not be).
- After all slip days are used up, 5% deduction for every 24 hours of delay. (e.g., 5 points for a 100-point homework)
- We will not consider late submission of any missing parts of a homework assignment or project deliverable. To make sure you have submitted everything, download your submitted files to double check.

Download the HW1 Skeleton before you begin.

Q1 [45 pts] Collecting and visualizing Twitter data

1. **[30 pts]** You will use the Twitter REST API to retrieve (1) *followers*, (2) *followers of followers*, (3) *friends* and (4) *friends of friends* of a user on Twitter (a Twitter *friend* is someone you follow and a Twitter *follower* is someone who follows you).

a. The Twitter REST API allows developers to retrieve data from Twitter. It uses the OAuth mechanism to authenticate developers who request access to data. Here's how you can set up your own developer account to get started:

- *Twitter* : Create a Twitter account, if you don't already have one.
- *Authentication*: You need to get API keys and access tokens that uniquely authenticate you. Sign into Twitter Apps with your Twitter account credentials. Click '*Create New App*'. While requesting access keys and tokens, enter:

Name	dva_hw1_<your-student-id> (eg: dva_hw1_jdoe3)
Description	"For CSE 6242 at Georgia Tech"
Website	http://poloclub.gatech.edu/cse6242/2017spring/
Callback URL	field should be left empty as we will not need it

Check the developer agreement checkbox and click on *'Create your Twitter application'*. Once your request is approved, you can click *'Keys and Access Tokens'* to view your *'API key'* and *'API secret'*. Generate your access token by clicking the *'Create my access token'* button. Now, you are ready to make authenticated API calls to fetch data.

- *keys.json* : Store the credentials in a file named **keys.json** in the format given below. To prevent any possible errors due to your JSON file format, it is recommended that you validate your file using a JSON formatter. The format is:

```
{
        "api_key": "your api key here",
        "api_secret": "your api secret here",
        "token": "your access token here",
        "token_secret": "your access token secret here"
}
```

Note:

- Twitter limits how fast you can make API calls. For example, the limit while making GET calls for friends is 15 requests per 15 minutes.
- Refer to the rate limits chart for different API calls.
- **Set appropriate timeout intervals** in the code while making requests.
- An API endpoint **may return different results for the same request**.

You will use **Python 2.7.x** (not Python 3.0+) and the tweepy library to modify parts of the boilerplate script (script.py). If you are new to Python, here are few useful links to help you get started: tutorialspoint , file reading and writing methods

b. [15 pts] Search for followers of the Twitter username "*PoloChau*". Use the API to retrieve the first 10 followers. Further, for each of them, use the API to find their 10 followers.

- Read the documentation for getting followers of a Twitter user. Note that in tweepy, the 'screen_name' parameter represents the Twitter username.
- You code will write the results to **followers.csv**.
  Each line in the file should describe one relationship in the format:
  `follower-username, username`
- Grading distribution is given in the boilerplate code.

Note: *follower-username* represents the Source and *username* represents the Target for an edge in a directed graph, which you will use in a later question.

c. [15 pts] Search for friends of the Twitter screen name "*PoloChau*". Use the API to retrieve the first 10 friends. Further, for each of the 10 friends, use the API to find their 10 friends.

- Read the documentation for getting friends of a Twitter user.
- You code will write the results to **friends.csv.**
  Each line in the file should describe one pair of relationship in the format:
  `username, friend-username`
- Grading distribution is given in the boilerplate code.

Note: *username* represents the Source and *friend-username* represents the Target for an edge in a directed graph.

If a user has fewer than 10 followers or friends, the API will return as many as it can find. Your code should be flexible to work with whatever data the API endpoint returns.

Note: Some users may be protected, which means you won't be able to fetch their followers or friends. Such users can be ignored in the secondary list in the code. However, they should be present in the primary list.

Deliverables: Create a directory called **Q1** to store all the files listed below.

Note: Do **NOT** submit your API credentials (**keys.json**). They should not be shared. We will use our own keys and tokens to grade your work.

- **script.py:** The boilerplate code modified by you. The submitted code should run as is. That is, no extra installation or configuration should be required other than the specified libraries. Also specify the python version in the code.
- **followers.csv** and **friends.csv** produced in step b and c respectively. Please note that these files will be modified in task 2b shortly.

2. **[15 pts]** Visualize the network of friends and followers obtained using Gephi, which you can download here. Ensure your system fulfils all requirements for running Gephi.

a. Go through the Gephi quick-start guide.

b. [2 pts] Insert `Source,Target` as the first line in both **followers.csv** and **friends.csv**. Each line in both files now represents a directed edge with the format `Source,Target`. Import all the edges contained in these files using **Data Laboratory**.

Note: Remember to check the **"create missing nodes"** option while importing since we do not have an explicit nodes file.

c. [8 pts] Using the following guidelines, create a visually meaningful graph:
- Keep edge crossing to a minimum, and avoid as much node overlap as possible.
- Keep the graph compact and symmetric if possible.
- Whenever possible, show node labels. If showing all node labels create too much visual complexity, try showing those for the "important" nodes.
- Using nodes' spatial positions to convey information (e.g., "clusters" or groups).

Experiment with Gephi's features, such as graph layouts, changing node size and color, edge thickness, etc. The objective of this task is to familiarize yourself with Gephi and hence is a fairly open ended task.

> We (course staff) will select some of the most visually meaningful and beautiful graphs from you all and share them with the class on Piazza.

d. [5 pts] Using Gephi's built-in functions, compute the following metrics for your graph:

- Average node degree
- Diameter of the graph
- Average path length

Briefly explain the intuitive meaning of each metric in your own words.
You will learn about these metrics in the "graphs" lectures.

Deliverables: Place all the files listed below in the **Q1** folder.

- **For part b: followers.csv** and **friends.csv** (with `Source,Target` as their first lines).
- **For part c:** an image file named "**graph.png**" (or "**graph.svg**") containing your visualization and a text file named "**graph_explanation.txt**" describing your design choices, using no more than 50 words.
- **For part d:** a text file named "**metrics.txt**" containing the three metrics and your intuitive explanation for each of them, using no more than 100 words.

Q2 [35 pt] SQLite

The following questions help refresh your memory about SQL and get you started with SQLite --- a lightweight, serverless embedded database that can easily handle up to multiple GBs of data. As mentioned in class, SQLite is the world's most popular embedded database. It is convenient to share data stored in an SQLite database --- just one cross-platform file, and no need to parse (unlike CSV files).

You will modify the given **Q2.SQL.txt** file to add SQL statements and SQLite commands to it.

We will test the correctness of your answers by running your modified Q2.SQL.txt against olympics.db, which generates Q2.OUT.txt (assuming the data files are present in the current directory).

```
$ sqlite3 olympics.db < Q2.SQL.txt > Q2.OUT.txt
```

We will generate the **Q2.OUT.txt** using above command. You may **not receive any points** if we are unable to generate the file, or if you do not strictly follow the output formats specified in each question.

We have added some additional lines of code in the Q2.SQL.txt file as follows:

- `.headers off. :` After each question, an output format has been given with a list of column names/headers. This command ensures that the headers are not displayed in the output.
- `.separator ',' :` To specify that the input file and the output are comma-separated.
- `select '':` This command prints a blank line. After each question's query, this command ensures that there is new line between each result in the output file.

WARNING: Do not copy and paste any code/command from this PDF to the sqlite command prompt, because PDFs sometimes introduce hidden/special characters, causing SQL error. Manually type out the commands instead.

a. [2 pt] *Import data.* Create an SQLite database called **olympics.db** and provide the SQL code (and SQLite commands) used to create the following tables. Use SQLite's dot commands (*.separator STRING* and *.import FILE TABLE*) to import data from files. Data used in this question was derived from https://www.kaggle.com/rio2016/olympic-games.

   Import the olympic athlete data from athletes.csv (in the Q2 Folder) into a new table (in olympic.db) called **athletes** with the schema:

```
athletes (
      id integer
      name text,
      nationality text,
```

```
        gender text,
        dob numeric,
        height real,
        weight integer,
        sport text,
        gold integer,
        silver integer,
        bronze integer
        )
```

Import the olympic countries data from countries.csv (in the Q2 Folder) into a new table (in olympic.db) called **countries** with the schema:

```
countries (
        country text,
        code text,
        population integer,
        gdp_per_capita real
        )
```

b. [2 pt] *Build indexes*. Create two indexes that will speed up subsequent join operations:

- An index called *athletes_country_index* in the **athletes** table for the `nationality` column.
- An index called *countries_country_index* in **countries** table for the `code` column.

c. [2 pt] *Quick computations.* Find the total number of unique female athletes who won gold medals. Then find the total number of unique male athletes who won silver medals.

Output format (i.e., each line contains one number):
`count_female_athletes_won_who_gold`
`count_male_athletes_won_who_silver`

d. [4 pt] *Who won the most medals?* Find the top ten athletes who won the most total medals. Sort by descending order with respect to the total medal count, then sort by ascending order with respect to the athletes' names (this will sort the athletes who won the same amount of medals alphabetically). Note `country` in the output should be the country name, not the three letter code.

Output format:
`athlete_name, country, total_medals`

e. [4 pt] *Worldwide medal leaderboard.* List the top ten countries that won the most medals, and their medal counts in each category (bronze, silver, and gold). Just like above, sort descending by the countries that have won the most medals in total, then sort ascending by country name. Note, also like before, `country` in the output should be the country name, not the three letter code.

Output format:
`country, gold_medal_sum, silver_medal_sum, bronze_medal_sum`

f. [6 pt] *Performance leaderboard*: Find the top 10 countries with the best performance ratio (total number of medals*1000/number of athletes). Display the `country` (the country name, not the three letter code), `performance_ratio, gdp_per_capita` and `avg_bmi` (average body mass index of each country). Sort the results by best performance ratio (high to low) and then by country name (alphabetical order).

Note: The body mass index of an athlete is calculated as weight/(height*height). `avg_bmi` of a country is the average over all athletes of a country. The country with the highest performance ratio is the best performing country.

Output format:
`country, performance_ratio, gdp_per_capita, avg_bmi`

[2pt] Write down the following questions' answers in **observation.txt**.

1. The name of the country with the highest `gdp_per_capita.`
2. The name of the country with the lowest `avg_bmi.`

Example content of **observation.txt** (for reference purposes only)
Greece Austria

g. [7 pt] *Creating view*: Create a view (virtual table) called *most_played_sports*, where each row contains a sport that has more than 500 athletes. One column stores the sport name, and another the total number of medals for that sport.

The format of the view is:

```
most_played_sports(sport, total_medals)
```

Using this view, write a query to find all distinct pairs of sports such that the total medals of `sport_1` is strictly fewer than the total medals of `sport_2.` Ensure that you do not include pairs where both sports have the same medal counts. Sort the results by `sport_1` (alphabetical order) and then by `sport_2` (alphabetical order).

Output format:
`sport_1, sport_2`

Note: Remember that creating a view will produce no output. Full points will only be awarded for queries that use joins.

Optional reading: Why create views?

h. [2 pt] Calculate the total number of such pairs created from the view made in part g.

Output format:
`count_total_pairs`

i. [4 pt] SQLite supports simple but powerful Full Text Search (FTS) for fast text-based querying (FTS documentation).

Import the movie overview data from movie-overview.txt (in the Q2 folder) into a new FTS table (in rt.db) called **movie_overview** with the schema:

```
movie_overview (
      id integer,
      name text,
      year integer,
      overview text,
      popularity decimal
      )
```

1. [2pt] Count the number of movies whose *overview* fields contain the word “love” but not “hate”.

   Output format:
   `count_overview_love_not_hate`

2. [2pt] List, in ascending order, the *id*s of the movies that contain the terms “love” and “war” in their *overview* fields with no fewer than 7 intervening terms in between.

   Output format:
   `id`

Deliverables: Place all the files listed below in the **Q2** folder

- **`Q2.SQL.txt:`** Modified file additionally containing all the SQL statements and SQLite commands you have used to answer questions a - i in the appropriate sequence.
- **`Q2.OUT.txt:`** Output of the questions above. See below for how to generate this file.
- **`observation.txt:`** contains the two lines of answers in part f.

Q3 [15 pt] D3 Warmup and Tutorial

- Go through the D3 tutorial here.
- Complete steps 01-16 (Complete through “16. Axes”).
- This is a simple and important tutorial which lays the groundwork for Homework 2.

Note: We recommend using Mozilla Firefox or Google Chrome, since they have relatively robust built-in developer tools.

Deliverables: Place all the files/folders listed below in the **Q3** folder

- A folder named **d3** containing file **d3.v3.min.js** (download)
- **index.html** : When run in a browser, it should display a scatterplot with the following specifications:
    a. [5 pt] There should be 50 points that are randomly generated and placed on the plot. Each point’s x coordinate should be a random number between 0 and 100 inclusively (i.e., [0, 100]), and so is each point’s y coordinate. A point’s x and y coordinates should be independently computed.)
    b. [2 pt] The plot must have visible X and Y axes that scale according to the generated points. The ticks on these axes should adjust automatically based on the randomly generated scatterplot points.
    c. [5 pt] Each point’s radius will be a value between 1 and 5 inclusively, determined by the point’s x coordinate. Use a linear scale for the radius, to map the domain of X values to the range of [1,5].
    d. [3 pt] All points with radii greater than the average radius of all scatterplot points should be colored blue. All other points should be colored green.
    e. Your full name which can appear above or below the scatterplot.

Note: No external libraries should be used. The index.html file can **only** refer to d3.v3.min.js within the d3 folder.

Q4 [10 pt] OpenRefine

a. Watch the videos on the OpenRefine’s homepage for an overview of its features. Download OpenRefine (latest release : 2.6 r.c2)

b. Import Dataset:
    - Launch OpenRefine. It opens in a browser (127.0.0.1:3333).
    - Choose "Create Project" -> This Computer -> "menu.csv". Click "Next".
    - You will now see a preview of the dataset. Click "Create Project" in the upper right corner.

c. Clean/Refine the data:
Note: OpenRefine maintains a log of all changes. You can undo changes. See the "Undo/Redo" button on the upper left corner.

i. [2 pt] Clean the "Event" and "Venue" columns (Select the column to be a Text Facet, and cluster the data. *Note: You can choose different "methods" and "keying functions" while clustering).* A clean "Event" column should have no more than 1650 unique values, and a clean "Venue" column should have no more 150 unique values. Record the number of unique values for each column *after* they have been cleaned in your observations. *Note: The number of unique values for a column is shown in the facet box under the title.*

ii. [2 pt] Use the Transform feature (under Edit Cells → Transform) and a General Refine Evaluation Language (GREL) expression to represent the dates in column ("date") in a format such that "2000-12-31" is converted to "Sunday, December 31, 2000". Record the GREL expression you used in the observations text file.

iii. [1 pt] List a column in the dataset that contains only nominal data, and another column that contains only ordinal data. (Refer to their definitions here)

iv. [2 pt] The "Note" column records whether the menu is present in English, German, or Both. Create a new column to record all the menus that are both in German and English as a boolean value. Use the Add column feature (under Edit Columns → Add column based on this column…) and a GREL expression to create the new column. Notice that the note may specify the menu as "GERMAN AND ENGLISH" or "ENGLISH AND GERMAN". The column should record "true" if the menu is bilingual and "false" if the menu is not. Record the GREL expression you use in the observations text file.

v. [1 pt] Some call numbers have "_wotm" appended to the end of the 8 digit number. Use the Transform feature and a GREL expression to remove the text at the end of the number. Record the GREL expression you used in the observations text file.

vi. [2 pt] Experiment with Open Refine, and list a feature (apart from the ones used above) you could additionally use to clean/refine the data, and comment on how it would be beneficial in fewer than 50 words. (*Basic operations like editing a cell or deleting a row do not count.)*

Deliverables: Place all the files listed below in the **Q4** folder

- **Q4Menu.csv** : Export the final table.
- **changes.json** : Submit a list of changes made to file in json format. Use the "*Extract Operation History*" option under the Undo/Redo tab to create this file.
- **Q4Observations.txt** : A text file with answers to parts c(i), c(ii), c(iii), c(iv), c(v), and c(vi)

Important Instructions on Folder structure

The directory structure must be:

```
HW1-LastName-FirstName/
    |--- Q1/
        |----  followers.csv
        |----  friends.csv
        |----  graph.png / graph.svg
        |----  graph_explanation.txt
        |----  metrics.txt
        |----  script.py
    |--- Q2/
        |----  Q2.SQL.txt
        |----  Q2.OUT.txt
        |----  observation.txt
    |--- Q3/
        |----   index.html
        |----   d3/
            |----    d3.v3.min.js
    |--- Q4/
        |----  changes.json
        |----  Q4Menu.csv
        |----  Q4Observations.txt
```