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

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

Due: Monday, September 18, 2017, 11:55 PM EST

Prepared by Kiran Sudhir, Varun Bezzam, Yuyu Zhang, Akanksha Bindal, Vishal Bhatnagar, Vivek Iyer, 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.

- Always check to make sure you are using the most up-to-date assignment PDF (e.g., re-download it from the course homepage if unsure).
- 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 and strategies (e.g., how cross-validation works, 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, 5% deduction for every 24 hours of delay. (e.g., 5 pts for 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.

Grading

The maximum possible score for this homework is 110 points (105 points + 5 bonus points). If a student scores 110 points, that student will receive (110 / 105) * 10 = 10.48 points towards the final course grade.

Q1 [45 pts + 5 bonus pts] Collecting and visualizing Twitter data

1. **[30 pts]** You will use the Twitter REST API to retrieve *friends* and *followers* on Twitter. A Twitter *friend* is someone you follow. A Twitter *follower* is someone who follows you.

a. [0 pts] The Twitter REST API allows developers to retrieve data from Twitter. It uses the OAuth mechanism to authenticate developers who request access to data. Follow the steps below to set up your own developer account to get started:

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

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

Check the developer agreement checkbox and click *'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, we 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**. We recommend that you to think about **how much time your script will run for** when solving this question, so you complete it on time.
- Refer to the rate limits chart for different API calls.
- **Set appropriate timeout intervals** in the code while making requests.
- Certain requests may fail with the **"130: Over Capacity" error code**. Re-running the script should resolve this error.
- An API endpoint **MAY return different results for the same request** (e.g., a Twitter user's friends and followers may change over time).

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 a few useful links to help you get started: tutorialspoint, file reading and writing methods

b. [28 pts] You will use the API to retrieve a subgraph of Twitter users, starting from the Twitter username *PoloChau*. Your tasks include:
(i) Retrieve *PoloChau*'s first 20 friends -- these are PoloChau's *primary friends*
(ii) For each *primary friend*, retrieve its first 20 friends
(iii) For each *primary friend*, retrieve its first 20 followers

"Retrieve first N friends/followers" means retrieving the first N friends/followers returned by the API.

- Read the documentation for getting the friends and followers of a Twitter user. Note that in tweepy, the 'screen_name' parameter represents the Twitter username.
- Your code will write the results to a file named **graph.csv**. Every line in **graph.csv** represents **one "following" relationship**.
  For example, suppose the file contains the following two lines:

```
Alice,Bob
Bob,Carol
```

  This means Alice follows Bob, and Bob follows Carol. In other words, the first user is the "source" of the following relationship, the second user the "target". You will use the graph constructed from these relationships (edges) in a later question.

  Therefore, when you write "friend information" to the file (in i and ii above), use the following format (NO space after comma):

username,friend-username

  Similarly, when you write "follower information" to the file (in iii above), use the following format (NO space after comma):

follower-username,username

- To simplify your implementation, your code for this part (part b) does **not** need to remove duplicate rows in **graph.csv**. Duplicate rows will be removed in part c.
- Grade distribution is indicated in the boilerplate code.

Note:

- If a user has fewer than 20 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.
- Some users may be protected. This means that you will not be able to fetch their followers or friends. Such users can be ignored when you retrieve the followers and friends of PoloChau's *primary friends* (indicated by points (ii) and (iii) above). However, they should be present in the *primary friends* list.

c. [2 pts] **Remove duplicate rows in graph.csv** using any methods/software/programs that you want, so that only unique relationships (rows) remain in the file.

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.
- **graph.csv** produced in step c. Please note that this file will be modified in Q1 task 2 and task 3 shortly.

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

a. Go through the Gephi quick-start guide.

b. [2 pts] Insert `Source,Target` as the first line in **graph.csv**. Each line now represents a directed edge with the format `Source,Target`. Import all the edges contained in the file 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: graph.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.

3. [5 bonus pts] You now have the opportunity to try out the Argo visualization tool under (heavy) development at the Polo Club of Data Science. Argo currently offers basic features. **Warning**: you may run into bugs when using Argo!

a. First, download Argo from the landing page at http://poloclub.gatech.edu/argo/
b. Then, run the Argo executable file to launch the program.
c. Once Argo has loaded, click the “Import CSV” button to import the **graph.csv** edge file from part 2. Make sure the `Source` and `Target` columns in the dialog box match their corresponding columns in the CSV file.
d. Use the same guidelines as in part 2c above to create a visually meaningful graph:

Experiment with Argo's features, such as changing node size and color, pinning and unpinning node locations, showing and hiding node labels, etc. You can use the "Save" and "Load" buttons on the toolbar to save your progress with Argo. The save file will contain nodes and edges in the graph as well as any modifications you made to the graph. This task is open ended and optional, hence the bonus points.

Deliverables: Take a screenshot of the visualization you have created using Argo, name it "**argo.png**" and place it in the **Q1** folder.

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 your answers' correctness by running your modified Q2.SQL.txt against flight_database.db to generate Q2.OUT.txt (assuming the current directory contains the data files).

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

We will generate the **Q2.OUT.txt** using the above command. You may **not receive any points** (1) if we are unable to generate the file, or (2) if you do not strictly follow the output formats specified in each question below.

We have added some lines of code in the Q2.SQL.txt file. Their purposes are:

- `.headers off.` : After each question, an output format has been given with a list of column names/headers. This command ensures that such 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 for use in the sqlite command prompt, because PDFs sometimes introduce hidden/special characters, causing SQL error. Manually type out the commands instead.

Note: For the questions in this section, you must use only INNER JOIN when you perform a join between two tables. Other types of join may result in incorrect outputs.

a. [2 pt] *Import data.* Create an SQLite database called **flight_database.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/usdot/flight-delays.

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

```
flights(
        airline text,
        flight_number integer,
        origin_airport text,
        destination_airport text,
        departure_delay integer,
        distance integer,
        arrival_delay integer
        )
```

Note: `departure_delay` refers to how late/early the flight has departed from the origin airport. The values are in minutes. A positive value indicates a delay, whereas a negative value indicates an early departure. For example, 20 means that the departure of the flight has been delayed by 20 minutes; -20 means departing 20 minutes earlier than scheduled. Similarly, `arrival_delay` refers to how late/early the flight has arrived at the destination airport.

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

```
airports(
        airport_code text,
        airport text,
        city text,
        state text,
        latitude real,
        longitude real
        )
```

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

```
airlines(
        airline_code text,
```

```
    airline text
    )
```

b. [2 pt] *Build indexes*. Create the following five indexes that will speed up subsequent join operations (speed improvement is negligible for this small database, but significant for larger databases):

   1. *flights_airline_index* for **flights** table's `airline` column
   2. *flights_origin_airport_index* for **flights** table's `origin_airport` column
   3. *flights_destination_airport_index* for **flights** table's `destination_airport` column
   4. *airport_airport_index* for **airports** table's `airport_code` column
   5. *airlines_airline_index* for **airlines** table's `airline_code` column

c. [2 pt] *Quick computations.* Find the total number of flights arriving at the destination airport 'SEA' with a delay of over 20 minutes. Then find the total number of flights departing from the origin airport 'SFO' with a delay of over 20 minutes.

   Output format (i.e., each line contains one number):
   `count_flights_arriving_at_sea`
   `count_flights_departing_from_sfo`

d. [4 pt] *Average delay of flights per airline.* List the top 5 airlines with longest average arrival delay. Sort by descending order of `avg_arrival_delay`. Note that `airline_name` in the output should be the name of the airline, not the 2-letter code.

   Format of each line in the output (5 lines total):
   `airline_name,avg_arrival_delay`

   Note: Include both early arrivals and delays in the computation of `avg_arrival_delay`.

e. [4 pt] *Airlines at each airport.* For every airport that has flights departing from it, list the airport name and the distinct airline names of those flights (of that airport). For example, if a flight has 'SEA' as the `origin_airport` and 'AS' as the `airline`, then the `airport_name` is 'Seattle-Tacoma International Airport' and the `airline_name` is 'Alaska Airlines Inc.'. Sort the output alphabetically, first by airport name (NOT the code), then by airline name (NOT the code). Limit the output to the first 30 entries.

   Format of each line in the output:
   `airport_name,airline_name`

f. [8 pt] *Percentage of delayed flights*: For each destination airport which has at least one arriving flight delayed by more than 30 minutes, calculate the percentage (out of 100) of arriving flights that have been delayed by more than 30 minutes. Note that `airport_name` refers to the

name of the airport and not the 3-letter code. Sort the output alphabetically by airport name. Limit the output to the first 20 entries.

Format of each line in the output (20 lines total):
`airport_name,percentage`

Hint: Consider performing an INNER JOIN on two SELECT statements (e.g., one for delayed flights, another for all flights).

g. [7 pt] *Creating a view*: Create a view (virtual table) called *airport_distances*, where each row consists of a distinct pair of airports and the corresponding distance between them. Please use the following method to calculate the distance between the latitudes of each pair of airports.

Distance between two airports:
(airport1_latitude - airport2_latitude)^2

Each row of the view should be of the form
`airport1_name,airport2_name,distance`

`airport1_name` should **strictly precede** `airport2_name` alphabetically.

The format of the view is:
`airport_distances(airport1_name,airport2_name,distance)`

Using this view, write a query to display the first 10 rows in the view after sorting the distance between the airports in descending order.

Format of each line in the output (10 lines total):
`airport1_name,airport2_name,distance`

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 flight_database.db) called **movie_overview** with the schema:

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

Note: Create the table using **fts3** or **fts4** only.

1. [2pt] Count the number of movies whose *overview* field starts with the word “when”.

   Output format:
   `count_overview_when`

2. [2pt] List, in ascending order, the *id*s of the movies that contain the terms “love” and “hate” in their *overview* fields with no more 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 queries in **`Q2.SQL.txt`**. Check above for how to generate this file.

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 should appear above the scatterplot. Set the title tag to your GA Tech ID (susan7). **Do not** use your alias.

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 and install OpenRefine (latest release : 2.7)

b. Import Dataset:
    - Launch OpenRefine. It opens in a browser (127.0.0.1:3333).
    - We use a property dataset from Zillow, which comes from a running competition on Kaggle (Zillow Prize: Zillow's Home Value Prediction). If you are interested in the details, please refer to the data description page. In order to save the cost of memory and disk space, we sampled a subset of the dataset as "properties.csv".
    - Choose "Create Project" -> This Computer -> "properties.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 "propertyzoningdesc" column (Select the column to be a Text Facet, and cluster the data. *Note: You can choose different "methods" and "keying functions" while clustering).* Our goal in this step is to merge values that point to the same object but have tiny difference, for example "EMR1" and "EMR1*". A clean "propertyzoningdesc" column should have no more than 950 unique values. All clusters having multiple values should be merged. You may use the default new cell value for each cluster. Record the number of unique values for the "propertyzoningdesc" column *after* it has 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 ("transactiondate") in a format such that "2016-01-01" is converted to "Friday, January 01, 2016". 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 "bedroomcnt" column records the number of bedrooms in the property. Create a new column to record all the properties that have strictly more than 3 bedrooms (not inclusive) as a boolean value. You may name the new column as "bedroomflag". Use the Add column feature (under Edit Columns → Add column based on this column…) and a GREL expression to create the new column.

v. [1 pt] Some values in the "propertyzoningdesc" column have "*" in the string (this happens even after the clustering step). Use the Transform feature and a GREL expression to remove the "*" in the string. Please record the expression you used in the observations text file.

vi. [2 pt] Experiment with OpenRefine, 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

- **properties_clean.csv** : Export the final table as a comma-separated values (CSV) file.
- **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/
        |----  argo.png
        |----  graph.csv
        |----  graph.png / graph.svg
        |----  graph_explanation.txt
        |----  metrics.txt
        |----  script.py
    |--- Q2/
        |----  Q2.SQL.txt
        |----  Q2.OUT.txt
    |--- Q3/
        |---- index.html
        |---- d3/
            |---- d3.v3.min.js
    |--- Q4/
        |----  properties_clean.csv
        |----  changes.json
        |----  Q4Observations.txt
```