cse6242-2018fall-online-hw1

CSE 6242 OAN, O01, O3: Data and Visual Analytics | Georgia Tech | Fall 2018

Homework 1: Analyzing The MovieDB data; SQLite; D3 Warmup; Gephi; OpenRefine

Due: Friday, September 14, 2018, 14:00 UTC

Prepared by (in alphabetical order): Anmol Chhabria, Mansi Mathur, Matthew Hull, Michael Petrey, Nilaksh Das, 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 (version number at bottom right of this document).
- ❏ Submit a single zipped file, called "HW1-{GT username}.zip", containing all the deliverables including source code/scripts, data files, and readme. Example: "HW1-jdoe3.zip" if GT account username is "jdoe3". Only .zip is allowed (no other format will be accepted). Your GT username is the one with letters and numbers.
- ❏ You may discuss high-level ideas with other students at the "whiteboard" level (e.g., how cross validation works, use hashmap instead of array) and review any relevant materials online. However, each student must write up and submit his or her own answers.
- ❏ All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures (e.g., reported to and directly handled by the Office of Student Integrity (OSI)). Consequences can be severe, e.g., academic probation or dismissal, grade penalties, a 0 grade for assignments concerned, and prohibition from withdrawing from the class**.**
- ❏ 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.
- ❏ In your final zip file, 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).
- ❏ We may use auto-grading scripts to grade some of your deliverables, so it is extremely important that you strictly follow our requirements.
- ❏ Wherever you are asked to write down an explanation for the task you perform, stay within the word limit or you may lose points.
- ❏ Every homework assignment deliverable and every project deliverable comes with a 48-hour "grace period". Any deliverable submitted after the grace period will get zero credit.
- ❏ 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. (The Q1 and Q3 folder folders are initially empty.)

Grading

The maximum possible score for this homework is 100 points.

Q1 [40 points] Collecting and visualizing The Movie DB (TMDb) data

Q1.1 [25 points] Collecting Movie Data

You will use "The Movie DB" API to: (1) download data about movies and (2) for each movie, download its 5 similar movies.

You will write some **Python 3** code (not Python 2.x) in **`script.py`** in this question. You will need an API key to use the TMDb data. Your API key will be an input to **`script.py`** so that we can run your code with our own API key to check the results. Running the following command should generate the CSV files specified in part b and part c:

```
python3 script.py <API_KEY>
```

Please refer to this tutorial to learn how to parse command line arguments. Please DO NOT leave your API key written in the code.

Note: The Python Standard Library and the requests library are allowed. Python wrappers (or modules) for the TMDb API may **NOT** be used for this assignment. Pandas also may **NOT** be used --- we are aware that it is a useful library to learn. However, to make grading more manageable and to enable our TAs to provide better, more consistent support to our students, we have decided to restrict the libraries to the more "essential" ones mentioned above.

a. How to use TheMovieDB API:

- Create a TMDb account and request for an API key - https://www.themoviedb.org/account/signup. Refer to this document for detailed instructions.
- Refer to the API documentation https://developers.themoviedb.org/3/getting-started/introduction , as you work on this question.

Note:
- The API allows you to make **40 requests every 10 seconds**. Set appropriate timeout intervals in your code while making requests. We recommend you think about how much time your script will run for when solving this question, so you will complete it on time.
- The API endpoint may return different results for the same request.

b. [10 points] Search for movies in the "**Comedy**" genre released in the year 2000 or later. Retrieve the 300 most popular movies in this genre. The movies should be sorted from most popular to least popular. **Hint**: Sorting based on popularity can be done in the API call.

- Documentation for retrieving similar movies: https://developers.themoviedb.org/3/discover/movie-discover https://developers.themoviedb.org/3/genres/get-movie-list
- Save the results in **`movie_ID_name.csv`**.

Each line in the file should describe one movie, in the following format --- NO space after comma, and do not include any column headers:

```
movie-ID,movie-name
```

For example, a line in the file could look like:

```
353486,Jumanji: Welcome to the Jungle
```

Note:
- You may need to make multiple API calls to retrieve all 300 movies. For example, the results may be returned in “pages,” so you may need to retrieve them page by page.
- Please use the “primary_release_date” parameter instead of the “release_date” parameter in the API when retrieving movies released in the year 2000 or later. The “release_date” parameter will incorrectly return a movie if any of its release dates fall within the years listed.

c. [15 points] For each of the 300 movies, use the API to find its 5 similar movies. If a movie has fewer than 5 similar movies, the API will return as many as it can find. Your code should be flexible to work with however many movies the API returns.

- Documentation for obtaining similar movies: https://developers.themoviedb.org/3/movies/get-similar-movies
- Save the results in **movie_ID_sim_movie_ID.csv**.

Each line in the file should describe one pair of similar movies --- NO space after comma, and do not include any column headers:

```
movie-ID,similar-movie-ID
```

Note: You should remove all duplicate pairs after the similar movies have been found. That is, if both the pairs `A,B` and `B,A` are present, only keep `A,B` where A < B. For example, if movie A has three similar movies X, Y and Z; and movie X has two similar movies A and B, then there should only be four lines in the file.

```
A,X
A,Y
A,Z
X,B
```

You do not need to fetch additional similar movies for a given movie, if one or more of its pairs were removed due to duplication.

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

- **movie_ID_name.csv:** The text file that contains the output to part b.
- **movie_ID_sim_movie_ID.csv:** The text file that contains the output to part c.
- **script.py**: The **Python 3** (not Python 2.x) script you write that generates both **movie_ID_name.csv** and **movie_ID_sim_movie_ID.csv**.

Note : Q1.2 builds on the results of Q1.1. Specifically, Q1.2 asks that the “Source,Target” be added to the resulting file from Q1.1. If you have completed both Q1.1 and Q1.2, your csv would have the header row — please submit this file. If you have completed only Q1.1, but not Q1.2 (for any reasons), then please submit the csv file without the header row.

Q1.2 [15 points] Visualizing Movie Similarity Graph

Using Gephi, visualize the network of similar movies obtained. You can download Gephi here. Ensure your system fulfills all requirements for running Gephi.

a. Go through the Gephi quick-start guide.

b. [2 points] Insert `Source,Target` as the first line in **movie_ID_sim_movie_ID.csv**. Each line now represents a directed edge with the format `Source,Target`. Import all the edges contained in the file using **Data Laboratory in Gephi**.

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

c. [8 points] 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.

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

- Average node degree (run the function called "Average Degree")
- Diameter of the graph (run the function called "Network Diameter")
- Average path length (run the function called "Avg. 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: `movie_ID_sim_movie_ID.csv`** (with `Source,Target` as its first line).
- **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 points] SQLite

SQLite is a lightweight, serverless, embedded database that can easily handle multiple gigabytes of data. It is one of the world's most popular embedded database systems. It is convenient to share data stored in an SQLite database --- just one cross-platform file which doesn't need to be parsed explicitly (unlike CSV files, which have to be loaded and parsed).

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

We will autograde your solution by running the following command that generates **Q2.db** and **Q2.OUT.txt** (assuming the current directory contains the data files).

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

- You may **not receive any points** if we are unable to generate the 2 output files.
- You may also **lose points** if you do not strictly follow the output format specified in each question below. The output format corresponds to the headers/column names for your SQL command output.

We have also added some lines of code to the **Q2.SQL.txt** file which are for the purpose of autograding. DO NOT REMOVE/MODIFY THESE LINES. You may **not receive any points** if these statements are modified in any way (our autograder will check for changes). There are clearly marked regions in the **Q2.SQL.txt** file where you should add your code. We have also provided a **Q2.OUT.SAMPLE.txt** which gives an example of how your final **Q2.OUT.txt** should look like after running the above command. Please avoid printing unnecessary items in your final submission as it may affect autograding and you may **lose points**.

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 only use INNER JOIN when performing a join between two tables. Other types of joins may result in incorrect results.

a. *Create tables and import data*.

i. [2 points] Create the following two tables ("movies" and "cast") with columns having the indicated data types:

- `movies`
  - `id` (integer)
  - `name` (text)
  - `score` (integer)
- `cast`
  - `movie_id` (integer)
  - `cast_id` (integer)
  - `cast_name` (text)

ii. [1 point] Import the provided **movie-name-score.txt** file into the `movies` table, and **movie-cast.txt** into the `cast` table. You can use SQLite's `.import` command for this. Please use relative paths while importing files since absolute/local paths are specific locations that exist only on your computer and will cause the autograder to fail right in the beginning.

b. [2 points] *Create indexes.* Create the following indexes which would speed up subsequent operations (improvement in speed may be negligible for this small database, but significant for larger databases):

i. `scores_index` for the `score` column in `movies` table
ii. `cast_index` for the `cast_id` column in `cast` table
iii. `movie_index` for the `id` column in `movies` table

c. [2 points] *Calculate average score*. Find the average score of all movies having a score >= 5

Output format:
`average_score`

d. [3 points] *Find poor movies*. List the five worst movies (lowest scores). Sort your output by score from lowest to highest, then by name in alphabetical order.

Output format:
`id,name,score`

e. [4 points] *Find laid back actors*. List ten cast members (alphabetically by `cast_name`) with *exactly* two movie appearances.

Output format:
`cast_id,cast_name,movie_count`

f. [6 points] *Get high scoring actors.* Find the top ten cast members who have the highest average movie scores. Sort your output by score (from high to low). In case of a tie in the score, sort the results based on the name of the cast member in alphabetical order. Skip movies with score <40 in the average score calculation. Exclude cast members who have appeared in two or fewer movies.

Output format:
`cast_id,cast_name,average_score`

g. [8 points] *Creating views.* Create a view (virtual table) called `good_collaboration` that lists pairs of actors who have had a good collaboration as defined here. Each row in the view describes one pair of actors who have appeared in at least three movies together AND the average score of these movies is >= 50.

The view should have the format:

```
good_collaboration(
        cast_member_id1,
        cast_member_id2,
        movie_count,
        average_movie_score)
```

For symmetrical or mirror pairs, only keep the row in which `cast_member_id1` has a lower numeric value. For example, for ID pairs (1, 2) and (2, 1), keep the row with IDs (1, 2). There should not be any self pairs (`cast_member_id1 == cast_member_id2`).

Full points will only be awarded for queries that use joins.

Remember that creating a view will not produce any output, so you should test your view with a few simple select statements during development. One such test has already been added to the code as part of the autograding.

Optional Reading: Why create views?

h. [4 points] *Find the best collaborators.* Get the five cast members with the highest average scores from the `good_collaboration` view made in the last part, and call this score the `collaboration_score`. This score is the average of the `average_movie_score` corresponding to each cast member, including actors in `cast_member_id1` as well as `cast_member_id2`. Sort your output in a descending order of this score (and alphabetically in case of a tie).

Output format:
`cast_id,cast_name,collaboration_score`

i. SQLite supports simple but powerful Full Text Search (FTS) for fast text-based querying (FTS documentation). Import movie overview data from the **movie-overview.txt** into a new FTS table 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. Also note that keywords like NEAR, AND, OR and NOT are case sensitive in FTS queries.

1. [1 point] Count the number of movies whose `overview` field contains the word “fight”.

   Output format:
   `count_overview`

2. [2 points] List the `id`’s of the movies that contain the terms “love” and “story” in the `overview` field with no more than 5 intervening terms in between.

   Output format:
   `id`

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

- **`Q2.SQL.txt`:** Modified file containing all the SQL statements and SQLite commands you have used to answer parts a - i in the appropriate sequence.

Q3 [15 points] 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.
- Ensure that you are using v3 of the D3 lib.

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 points] There should be 100 points using a rect shape that are randomly generated and placed on the plot. Each point's x coordinate should be a random number between 10 and 250 inclusively (i.e., [10, 250]), and so is each point's y coordinate. A point's x and y coordinates should be independently computed.)

  b. [2 points] 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 scatter-plot points.

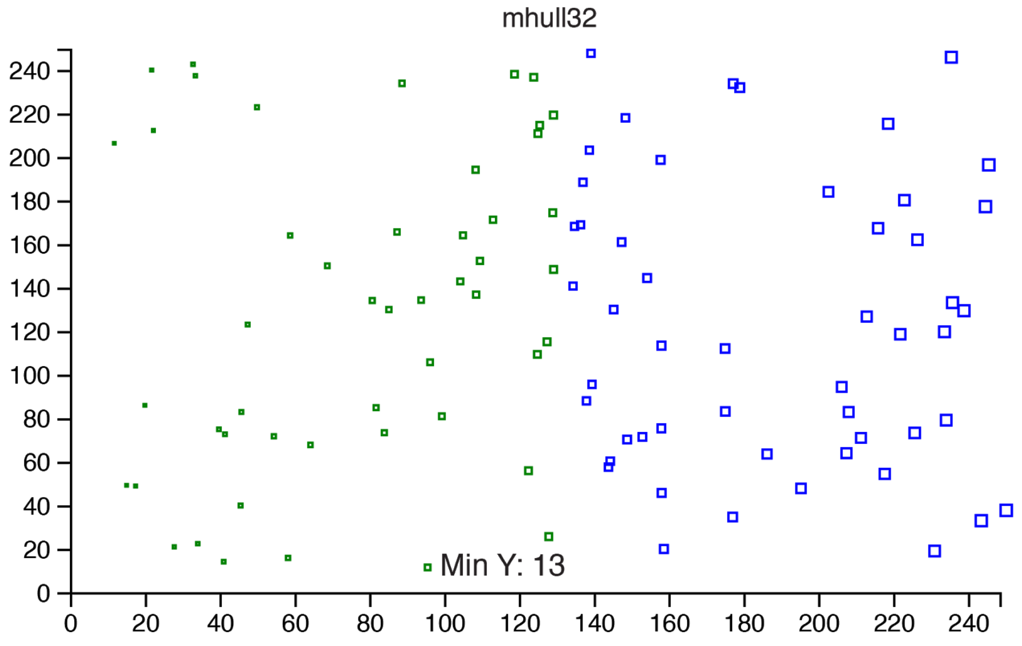
  c. [2 points] Each rect point's width and height will be of equivalent size. Set each point's width and height to the same value, between 1 and 5 inclusively, determined by the point's x coordinate. Use a single linear scale and apply it to both width and height to map the domain of X values to the range of [1,5].

  d. [3 points] All points with a width greater than the average width of all scatter-plot rect points should be outlined in blue. All other points should be outlined green. The points should use a transparent fill to enable visualization of overlapping points.
  Hint: Modify the 'stroke' parameter of the rect.

  e. [3 points] It is often desirable to emphasize some important data points in a dataset. Accomplish this by displaying a text annotation for the point that contains the smallest y value. The annotation should read "Min Y: <value>" where <value> is the minimum y. If there are multiple points that contain the same minimum y value, you can pick whichever point you like to annotate. Optional: experiment with different approaches to style the annotation text, e.g., try using different positioning settings, font-sizes, font-weights, and/or color to make it stand out.

  f. Your GT username (e.g., jdoe3) should appear above the scatterplot. Also set the title tag to your GT username.

The scatterplot should appear similar to but not exactly equivalent to the sample plot provided below. Remember that the plot will contain random data.



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 points] OpenRefine

a. Watch the videos on the OpenRefine's homepage for an overview of its features.
Download and install OpenRefine (version : 2.8)

b. Import Dataset:

- Launch OpenRefine. It opens in a browser (127.0.0.1:3333).
- You will use a subset of Fast Food Restaurants dataset derived from a Kaggle competition (Fast Food Restaurants). If you are interested in the details, please refer to the data description page. To reduce the demand for memory and disk space that you may need, we have sampled a subset of the dataset as "restaurants.csv", which you will use in this question. Each row in this file is a restaurant.
- Choose "Create Project" -> This Computer -> "restaurants.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 points] Clean the "*name*" column (select the column to be a Text Facet, and cluster the data, and select "key collision" as the clustering method and "fingerprint" as the keying function). Our goal in this step is to merge values that point to the same object but have tiny difference, for example "McDonald's" and "McDonalds". 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 "*name*" 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 points] Some of the string values in the "*websites*" column contain multiple websites for each restaurant. In the cases where the restaurant has more than one website, the websites' URLs are separated by commas (","). Replace the current values for the "*websites*" column (Edit Cells → Transform) with only the first website in the cell value using GREL (General Refine Evaluation Language). Provide the GREL expression used in the observations text file.

Example

Cell value **before** running GREL expression:
```
"http://mcdonalds.com,http://www.mcdonalds.com/?
cid=RF:YXT_FM:TP::Yext:Referral"
```

Cell value **after** running GREL expression:
```
"http://mcdonalds.com"
```

iii. [2 points] Name a column in the dataset that contains only nominal data, then name another column that contains only ordinal data (if no such column exists, write "none" as the column name), in the following format (i.e., your answers should contain two lines). For the definitions of "nominal" and "ordinal", please see this document.

```
nominal: columnX
ordinal: columnY
```

iv. [2 points] In each row of the data file, the "*province*" column stores the U.S. state in which the restaurant resides. Create a new column called "GAflag" that records a boolean value for each restaurant: a cell value should be set to `true` if the restaurant's "province" column contains "GA", `false` otherwise. Use the Add column feature (under Edit Columns → Add column based on this column…) and a GREL expression to create the new column. Record the GREL expression you used in the observations text file.

v. [2 points] The string values in the "*websites*" column contain "http://" or "https://". Use the Transform feature and a GREL expression to remove any "http://" or "https://" from the values. Please record the expression you used in the observations text file.

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

- **restaurants_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).

Important Instructions on Folder structure

The directory structure must be:

```
HW1-{GT username}/
        |---    Q1/
                |----   movie_ID_name.csv
                |----   movie_ID_sim_movie_ID.csv
                |----   graph.png / graph.svg
                |----   graph_explanation.txt
                |----   metrics.txt
                |----   script.py
        |---    Q2/
                |----   movie-cast.txt
                |----   movie-name-score.txt
                |----   movie-overview.txt
                |----   Q2.SQL.txt
        |---    Q3/
```

```
            |---- index.html
            |---- d3/
                  |---- d3.v3.min.js
    |---    Q4/
            |----     restaurants_clean.csv
            |----     changes.json
            |----     Q4Observations.txt
```

Version 5