cse6242-2018fall-campus-hw1

CSE 6242 A, Q / CX 4242 A: Data and Visual Analytics | Georgia Tech | Fall 2018

Homework 1: Analyzing The MovieDB Data; Gephi; SQLite; D3 Warmup; OpenRefine; Flask; jQuery

Due: Friday, September 14, 2018, 11:55 PM EST

Prepared by Arathi Arivayutham, Siddharth Gulati, Jennifer Ma, Mansi Mathur, Vineet Vinayak Pasupulety, Neetha Ravishankar, 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 120 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 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 pt] SQLite

The following questions will 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 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
```

We will generate the **Q2.OUT.txt** using the above command.

- 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 added some lines of code in 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**. Purpose of some lines of code in file Q2.SQL.txt are as follows:

- `.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 a 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.

Note: Do not use `.mode csv` in your Q2.SQL.txt file. This will cause quotes to be printed in the output of each `select '';` statement.

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 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 before attempting this question.
- 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:

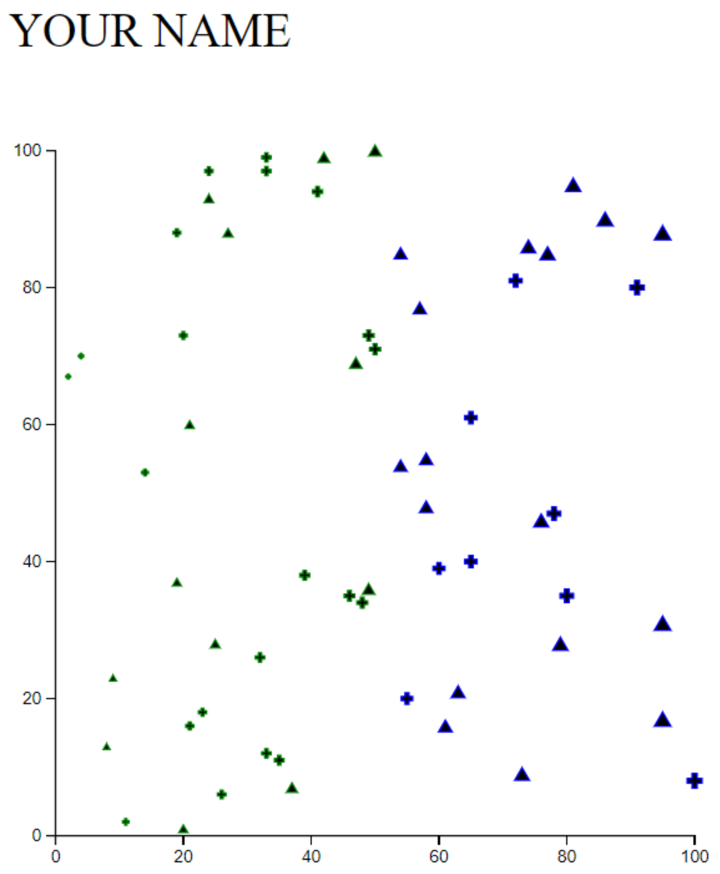
  a. [12 pt] Generate and plot 60 objects: 30 upward-pointing equilateral triangles and 30 crosses. Each object's X and Y coordinates should be a random integer between 0 and 100 inclusively (i.e., [0, 100]). An object's X and Y coordinates should be independently computed.

  Each object's size will be a value between 5 and 50 inclusively (i.e., [5, 50]). You should use the "symbol.size()" function of d3 to adjust the size of the object. Use the object's X coordinate to determine the size of the object. You should use a linear scale for the size, to map the domain of X values to the range of [5,50]. Objects with larger x coordinate values should have larger sizes.
  This link explains how size is interpreted by symbol.size(). You may want to look at this example for the usage of "symbol.size()" function.

  All objects with size greater than the average size of all scatterplot objects should be colored blue and all other objects should be colored green.
  All these objects should be filled (Please see the figure below) and the entire graph should fit in the browser window (no scrolling).

  b. [2 pt] The plot must have visible X and Y axes that scale according to the generated objects. The ticks on these axes should adjust automatically based on the randomly generated scatterplot objects.

  c. [1 pt] Your full name (in upper case) should appear above the scatterplot. Set the HTML title tag (<title>) to your GT account username (jdoe3) in lower case. **Do not** use your alias.



Example scatter plot

Notes:

- We show an example result of how your scatter plot may look like in the above figure. Your plot will likely be different (e.g., different locations for the points).
- No external libraries should be used. The index.html file can **only** refer to d3.v3.min.js within the d3 folder.
- While accessing d3.v3.min.js from index.html, please make sure that the path used is relative.

Q4 [10 pt] OpenRefine

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

b. Import Dataset:

- Launch OpenRefine. It opens in a browser (127.0.0.1:3333).
- We use a transactions dataset from Kaggle (Black Friday). 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 "properties.csv", which you will use in this question. Each row in this file is of the form <User_ID, Product_ID, Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status, Product_Category_1, Product_Category_2, Product_Category_3, Purchase>.
- 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] Use the Transform feature (under Edit Cells → Transform) and a General Refine Evaluation Language (GREL) expression to replace the values M and F in the column "Gender" with 0 and 1 respectively. Record the GREL expression you used in the observations text file.

ii. [2 pt] Use the Transform feature and a GREL expression to replace the null values in column "Product_Category_2" with the string "0". Record the GREL expression you used in the observations text file. Perform the same operation on "Product_Category_3" column to replace null values with "0".

iii. [2 pt] Use the Transform feature (under Edit Cells → Transform) and a GREL expression to replace the value "55+" in the column "Age" with the value "56-100". Record the GREL expression you used in the observations text file.

iv. [2 pt] Create a new column "High_Priced" with the values 0 or 1 based on the "Purchase" column (Edit column → Add column based on this column) with the following conditions: If the purchase value is greater than 1000, "high_priced" should be set as 1, else 0. (Ensure that the GREL expression handles converting the string value in "Purchase" column to numeric form). Record the GREL expression you used in the observations text file.

v. [2 pt] Create a text facet (Facet → Text facet) from the "Product_ID" column and find the number of unique products in the dataset. Then, sort the unique names by their counts to find the product with the highest count value (i.e., most frequently bought). Record the number of unique products and the Product_Id of this most frequently purchased product, separated by a comma, in the observations text file. For example, if the number of unique products is 100 and the product bought the most is P00000001, then your answer would be "100,P00000001" (without double quotes).

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 and c.v. Provide each answer in a new line.

Q5 [20 pt] Web Development with Flask and jQuery

In this question, we are going to create an interactive web-based application from scratch using web frameworks. It is recommended that you learn about the basics of HTML and Python before attempting this question. We believe this question will be helpful to students/teams who wish to develop web-based applications for their class projects. We recommend attempting this question using **Python 3+**.

Flask is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. A **web framework** or **web application framework** is a software framework designed to support the development of web applications including web services, web resources, and web APIs. Web frameworks aim to automate the overhead associated with common activities performed in web development. Flask is called a micro framework because it does not require particular tools or libraries. However, Flask supports extensions that can add application features as if they were implemented in Flask itself.

jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML. jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications. jQuery also provides capabilities for developers to create plugins on top of the JavaScript library. **Ajax** is a set of Web development techniques using many Web technologies on the client side to create **asynchronous Web applications**. With Ajax, Web applications can send and retrieve data from a web server asynchronously (in the background) without interfering with the display and behavior of the existing page. By decoupling the data interchange layer from the presentation layer, Ajax allows for Web pages, and by extension Web applications, to change content dynamically without the need to reload the entire page.

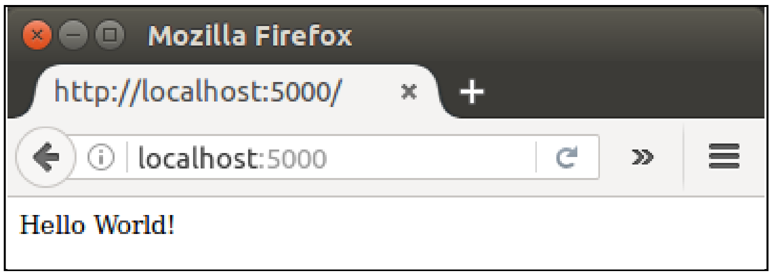
A few good tutorials for working with Flask, MySQL and jQuery in specific applications are listed below. You do not have to go through all of them to solve this question, however, they may give you clues that will help you solve the tasks at hand. These links are only for educational purposes - you do not have to implement them to get credit for this question.

- Implementing a simple Web application with Flask
- A tutorial to connect JQuery and Flask to handle REST requests
- A tutorial to connect a MySQL backend database for Flask
- A tutorial on understanding Flask in depth
- Advanced full stack development for a fully functional website

In the following questions, strive to create user interfaces that have both good usability (easy to understand, and to use) and are aesthetically pleasing (it is a valued industry skill!). However, you will not lose or gain points due to user interface aesthetics. **Strictly adhere to the letter casing of file names (all small letters)** - Flask and operating systems like Ubuntu are sensitive to casing of file names. If wrong casing is used, this might cause your submission to fail on case-sensitive operating systems, and points may be deducted.

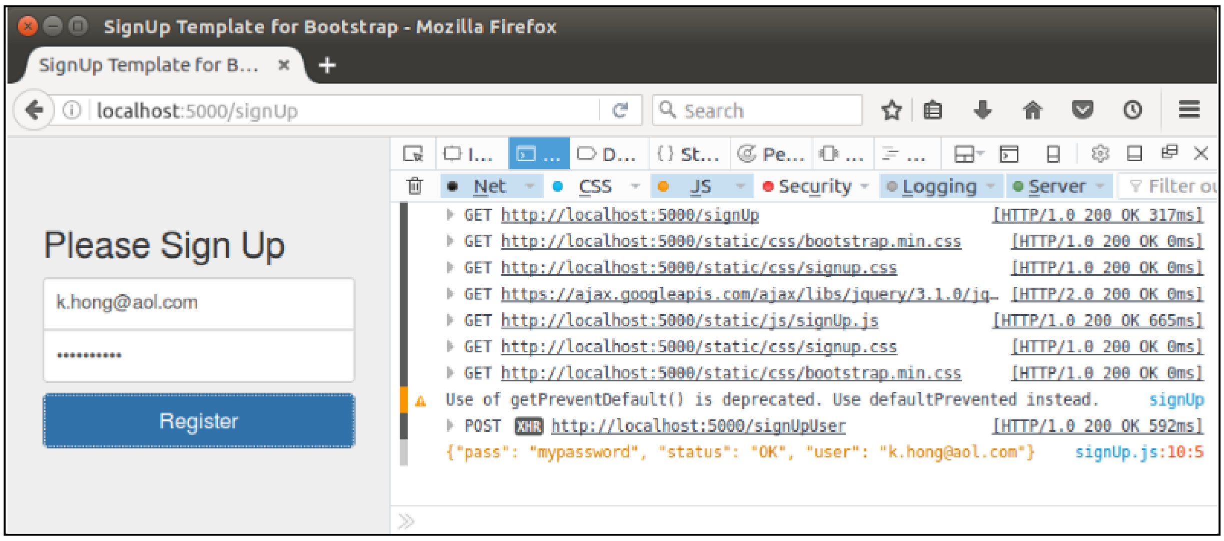
a. [5 pt] Follow the steps mentioned in this tutorial to create a signup website for CSE6242. The tutorial will help you create an HTML page called **signup.html.** Also, create an html page called **hello.html** which displays the phrase “Hello World!”. Visiting http://127.0.0.1:5000/ will display **hello.html.**

Example screenshots are shown below:



Hello World!

hello.html



```
GET http://localhost:5000/signUp                              [HTTP/1.0 200 OK 317ms]
GET http://localhost:5000/static/css/bootstrap.min.css          [HTTP/1.0 200 OK 0ms]
GET http://localhost:5000/static/css/signup.css                 [HTTP/1.0 200 OK 0ms]
GET https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jq…      [HTTP/2.0 200 OK 0ms]
GET http://localhost:5000/static/js/signUp.js                 [HTTP/1.0 200 OK 665ms]
GET http://localhost:5000/static/css/signup.css                 [HTTP/1.0 200 OK 0ms]
GET http://localhost:5000/static/css/bootstrap.min.css          [HTTP/1.0 200 OK 0ms]
Use of getPreventDefault() is deprecated. Use defaultPrevented instead.       signUp
POST XHR http://localhost:5000/signUpUser                     [HTTP/1.0 200 OK 592ms]
{"pass": "mypassword", "status": "OK", "user": "k.hong@aol.com"}      signUp.js:10:5
```

signup.html (with the console debugger opened on the right)

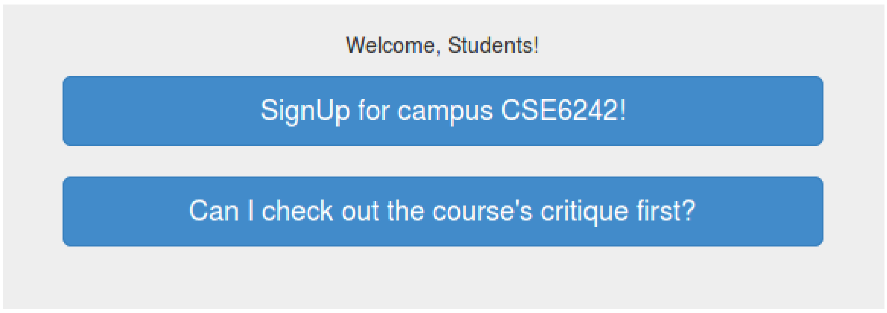
To run your website:

1. Go to the directory that contains your python script, **app.py,** and run it in the terminal.
2. Copy paste the URL printed in the terminal after the server (python script **app.py**) starts.
3. Go to http://127.0.0.1:5000/, the landing page, and check that it displays “Hello World!”.
4. Go to http://127.0.0.1:5000/signup and test filling in the form on the website.
5. Right click on your browser, click ‘Inspect Element’ and then press the console tab to see whether your entered details are visible there.

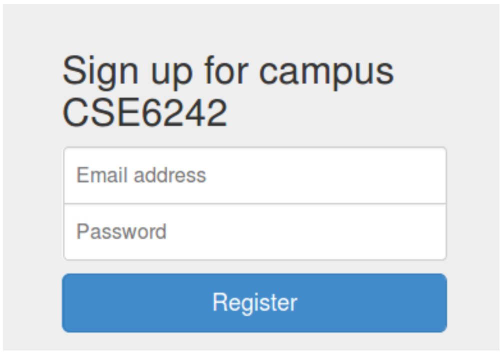
b. [3 pt] You will modify the landing page (**hello.html**) created in Q5.a to help potential CSE6242 students more easily navigate to the sign up page. You will create two buttons, one for signing up campus students, and another for prospective students to check out the course’s critique.

1. Create a button on the **hello.html** page (created in part a) which when clicked would redirect to the sign up page (**signup.html**) for the campus CSE6242 course.
2. Create a second button on **hello.html** that when clicked will redirect students to the course critique page of CSE6242 (http://critique.gatech.edu/course.php?id=CSE6242) (you probably saw it already, before signing up for this course).

The image below shows an example design for modified **hello.html**. Your design does not need to follow the example exactly.


Welcome, Students!
SignUp for campus CSE6242!
Can I check out the course's critique first?

Updated hello.html



Updated signup.html

c. [7 pt] We would now like to create a confirmation for users who have completed registering. Hence, we want users to see a simple confirmation message at the bottom of the signup page after they register. However, we are student-centric, and want to say everything with a personal touch! Hence, we would like to print the email address (not the password!) of the registered user in the confirmation message. Do the following:

1. Print the following confirmation message underneath the registration form only after the user enters his details and clicks "Register". Try to keep the message clearly visible and center aligned.

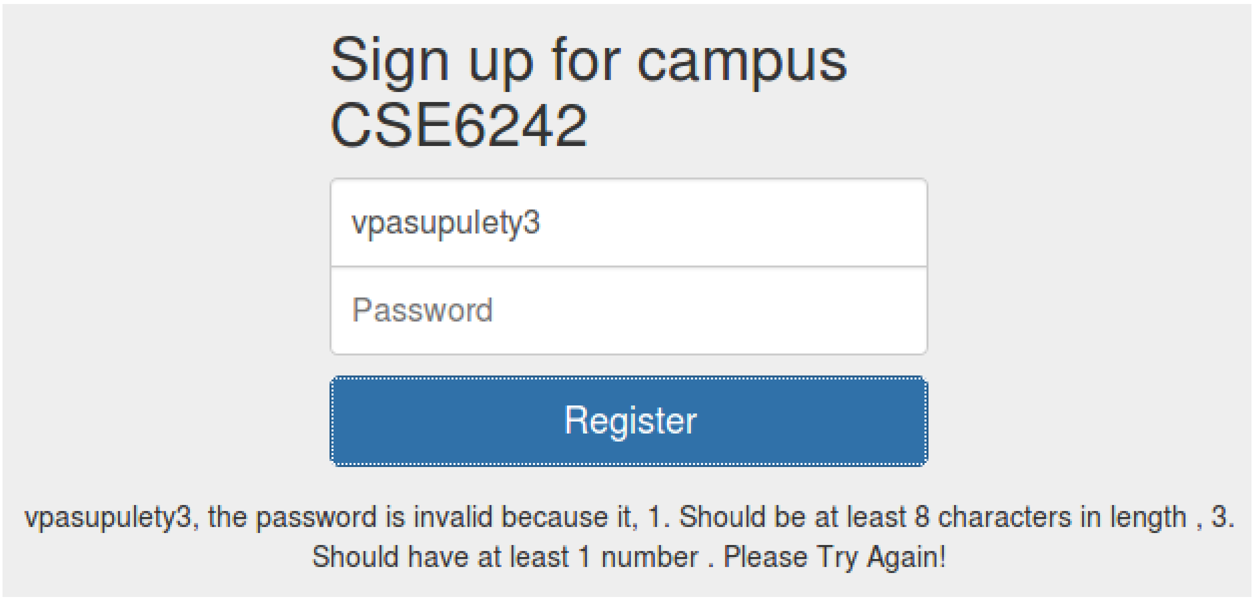
   "Congratulations on registering for CSE6242, <email address>! Redirecting you to the course homepage...".

2. After the confirmation message is printed, clear the previously entered inputs from the form.
   Hint: You only need one function to do this.
3. After displaying the above mentioned confirmation message, wait for 3 seconds, and then redirect to the CSE6242 website (http://poloclub.gatech.edu/cse6242/).
   Hint: Consider using window.setTimeout and document.location.

d. [5 pt] Like all other websites, we would like that students choose passwords that can not be easily compromised. We would like to check that all students have passwords that satisfy the following security criterion before accepting the registration of the student:

1. Should be at least 8 characters in length
2. Should have at least 1 uppercase character
3. Should have at least 1 number

Write a function that will check this input. If all criteria are met, then go ahead and send the confirmation message. But if they are not met, store the criteria that were violated and store them in the password that is to be returned. For example, if criteria 1 and 3 were not met, then password = [1,3] and status = 'BAD' (instead of 'OK'). In such failure cases, return status = 'BAD', user = <username> and password = [1,3]. On the webpage, mention the criteria that have not been met --- use the exact criterion wording, as written in the bullet list above. Also, encourage them to try again. Clear the password input of the form only for failure cases. Using the above example, the output should look similar to:



vpasupulety3, the password is invalid because it, 1. Should be at least 8 characters in length , 3. Should have at least 1 number . Please Try Again!

Output when entering an invalid password

Deliverables: Place all the files listed below in the **Q5** folder (carefully follow the folder structure mentioned at the end of this assignment):

- **hello.html** : The landing page of the website
- **signup.html** : The registration page for the campus course
- **signup.css**: Provided by us. You are encouraged to make changes that make the site aesthetically pleasing
- **bootstrap.min.css**: Provided by us. Please do not edit this!
- **signup.js:** Javascript file holding the AJAX requests
- **app.py**: Server backend that is responsible for routing, etc.

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/
                          |----      Q2.SQL.txt
                          |----      Q2.OUT.txt
                    |---      Q3/
                          |---- index.html
                          |---- d3/
                               |---- d3.v3.min.js
                    |---      Q4/
                          |----      properties_clean.csv
                          |----      changes.json
                          |----      Q4Observations.txt
                    |--- Q5/
                        |----  app.py
                        |----  templates/
                                |-----  signup.html
                                |-----  hello.html
                        |----  static/
                                |-----  css/
                                      |------  signup.css
                                      |------  bootstrap.min.css
                                 |-----  js/
                                      |------  signup.js
```

Version 0