CSE 6242/CX 4242: Data and Visual Analytics | Georgia Tech | Spring 2018

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

Due: Friday, February 2, 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 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 100 points.

Q1 [30 pt] Collecting and visualizing Twitter data

1. **[20 pt]** 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/
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. Or consider using one of the API's properties to handle rate limiting.
- 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

Note: Running Python < 2.7.9 might cause SSL certificate errors when trying to access the Twitter API.

b. [18 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.
- Ensure that you write strings, and not unicodes to the csv file.

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. Note that the edges are directed. "A,B" and "B,A" refer to different relationships on Twitter, so you would keep them both.

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 shortly.

2. [10 pt] 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. [1 pt] 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. [5 pt] 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. [4 pts] 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")
   - Graph density (run the function called "Graph Density")

   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 four metrics and your intuitive explanation for each of them, using no more than 125 words.

Q2 [30 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 test your answers' correctness by running your modified Q2.SQL.txt against olympics.db to generate Q2.OUT.txt (assuming the current directory contains the data files).

```
$ sqlite3 olympics.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 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. [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/the-guardian/olympic-games/data.

Note: Please use ONLY the data provided in Q2 folder and NOT from the Kaggle website as datasets have been modified for this question.

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

```
athletes(
        id integer,
        sex text,
        dob text,
        height float,
        weight integer,
        nationality text
        )
```

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

```
countries(
        country text,
        code text,
        population integer,
        gdp float
        )
```

Import the data on the games (such as athlete id, name, nationality information along with the sport name and number of different medals won) from games.csv (in the Q2 Folder) into a new table (in olympics.db) called **games** with the schema:

```
games(
        id integer,
        name text,
        nationality text,
        sport text,
        gold integer,
        silver integer,
        bronze integer
        )
```

b. [1 pt] *Build indices*. Create the following four indexes that will speed up subsequent join operations (speed improvement is negligible for this small database, but significant for larger databases):

1. *id_index* for **athletes** table's `id` column
2. *nationality_index* for **athletes** table's `nationality` column
3. *id_games_index* for **games** table's `id` column
4. *code_index* for **countries** table's `code` column

c. [1 pt] *Quick computation.* Find the total number of gold medals won by the athletes with nationality 'USA'.

Output format (a single line that contains one number):
`count_usa_gold`

d. [2 pt] *Distinct Sports:* List all distinct sports where 'USA' won gold medals, in alphabetical order. Use the DISTINCT keyword to avoid repetition of sports.

e. [2 pt] *Handling Empty Cells:* Notice that some of the cells in the `population` column of **countries** are empty strings. Use the following command to explicitly set these empty cells to `NULL` by updating the table as follows:

```
UPDATE countries SET population = NULL where population='';
```

With the empty cells filled with null values, you can use the IS NOT NULL keyword to exclude such rows in any query.

Once you have updated the `countries` table to fill the empty cells with null values, list the top 5 most populated countries along with the sum of all medals (i.e., gold, silver and bronze combined) won by the country. Your result should be ordered by country population, from highest to lowest.

For example: if '*United States*' has won 10 medals in total and '*United Kingdom*' has won 12, but *USA* is more populated than *UK*, then your result should order them as follows:
United States, 10
United Kingdom, 12

Format of each line in the output (5 lines total):
`full_country_name, sum_of_medals`

f. [2 pt] *Female Athletes:* Find the top 5 countries with the highest number of female athletes who have won at least one gold medal. List the names of these countries (full names and not the country code) along with their respective **female athlete counts** as output. Your output should be ordered by the count from highest to lowest.

Format of each line in the output (5 lines total):

`full_country_name, count_of_gold_winning_female_athletes`

g. [4 pt] *Fit Athletes:* The BMI of a person is calculated as the ratio of their weight (in kilograms) to their squared height (in meters). A person is considered 'fit' if his or her BMI is between 18.5 (inclusively) and 24.9 (inclusively).

   Find the athletes who are fit (according to the above definition) and list their names along with their BMI values. Sort the list first alphabetically by their names and then by their BMI values from lowest to highest, and limit the result to 10 rows.

   Note: The athletes table lists the `height` in meters and `weight` in kilograms.

   Format of each line in the output:
   `athlete_name, bmi_value`

   Hint: Consider using BETWEEN.

h. [5 pt] *Percentages of Gold & Silver Medals:* Calculate the percentage of gold medals and silver medals won by each country in **INTEGER FORMAT** only (if you use float in your intermediate calculations, then round down the final value using '*CAST(value as int)*'). Order the countries first by their gold medal percentage (highest to lowest) and then by their silver medal percentage (highest to lowest). List the full names of the countries along with their gold and silver medal percentages. Limit the result to the top 20 countries.

   Format of each line in the output (20 lines total):
   `full_country_name, gold_medal_percentage, silver_medal_percentage`

   Hint: Consider using SELECT statements to find the denominator in the percentage calculation.

   Note: Percentage of gold medals won by a country = Number of gold medals won by that country / Total number of gold medals in the olympics.

i. [4 pt] *Creating a view*: Create a view (virtual table) called *gdp_metrics*, where each row consists of distinct pairs of countries such that country1's GDP always exceeds country2's GDP by no more than 100 (note that the difference in the GDPs of the two countries should not be equal to 0). Please use the following method to calculate the difference:

   (country1_gdp - country2_gdp)

   In each pair *<full_country1_name, full_country2_name>* you form, *country1_gdp* should be greater than *country2_gdp*.

Each row of the view should be of the form:
`full_country1_name, full_country2_name, difference_gdp`

The format of the view is:
`gdp(full_country1_name, full_country2_name, diff)`

Using this view, write a query to display the first 5 rows in the view after sorting the gdp differences in descending order.

Format of each line in the output (5 lines total):
`country1_full_name, country2_full_name, diff`

Note: Remember that creating a view will produce no output.

Optional reading: Why create views?

j. [1 pt] Calculate the total number of such pairs created from the view made in part i.

Output format:
`count_total_pairs`

k. [6 pt] SQLite supports the 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 olympics.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. Also note that keywords like NEAR, AND, OR and NOT are case-sensitive in FTS queries.

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

   Output format:
   `count_overview_in`

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

   Output format (one or more lines):
   `id`

3. [2pt] List the names of the movies that contain partially/fully the word “geo” in their overview field. Look for words in the overview field that begin with the word “geo”. For example, “geography”, “Georgia”, and “geo” all partially/fully match and begin with the word “geo”.

   Output format (one or more lines):
   `movie_name`

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

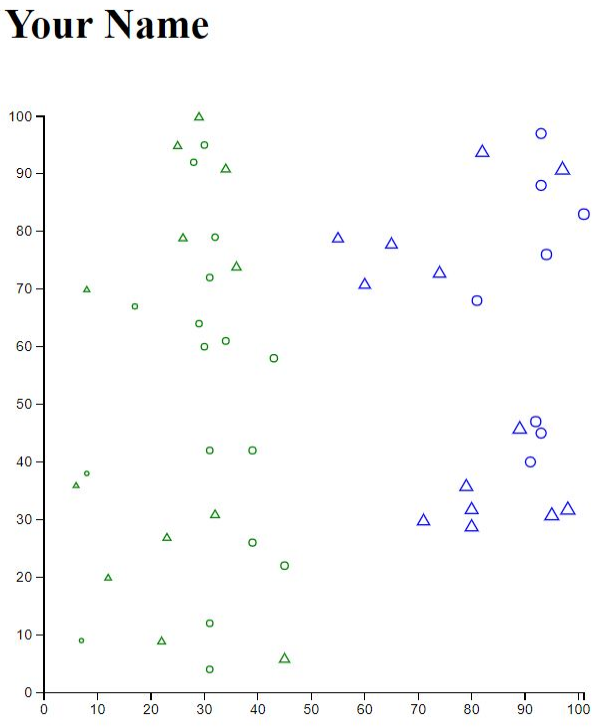
- A folder named **d3** containing file **d3.v3.min.js** (download)
- **index.html** : When run in a browser, it should display a scatterplot with the following specifications:
  a. [12 pt] Generate and plot 50 objects: 25 circles and 25 upward-pointing equilateral triangles. Each object’s X and Y coordinates 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 not be filled (Please see the figure below).

b. [2 pt] The plot must have visible X and Y axes whose range of values linearly scale according to the coordinates of 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.

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 : 2.8)

b. Import Dataset:
   - Launch OpenRefine. It opens in a browser (127.0.0.1:3333).
   - We use a products dataset from Mercari, derived from a competition on Kaggle (Mercari Price Suggestion Challenge). If you are interested in the details, please refer to the data description page. We have sampled a subset of the dataset as the given "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.a [1 pt] Select the "category_name" column and choose 'Facet by Blank' (Facet -> Customized Facets -> Facet by blank) to filter out the records that have blank values in this column. Provide the number of rows that return True. Remove these rows.

i.b [1 pt] Split the column "category_name" into multiple columns without removing the original column. For example, a row with "Kids/Toys/Dolls & Accessories" in the category_name column, would be split across the newly created columns as "Kids", "Toys" and "Dolls & Accessories". Use the existing functionality in OpenRefine that creates multiple columns from an existing column based on a separator (i.e. in this case '/'). Provide the number of columns that are created in this operation. Remove the newly created columns that do not have values in all rows.

ii. [2 pt] Select the column "name" and apply the Text Facet (Facet -> Text Facet). Click on the Cluster button which opens a window where you can choose different "methods" and "keying functions" to use while clustering. Choose the keying function that produces the highest number of clusters under the "Key Collision" method. Provide the number of clusters found using this keying function. Click on 'Select All' and 'Merge Selected & Close'.

iii. [2 pt] Replace the null values in the “brand_name column” with the text “Unbranded” (Edit Cells -> Transform). Provide the General Refine Evaluation Language (GREL) expression used.

iv. [2 pt] Create a new column “high_priced” with the values 0 or 1 based on the “price” column with the following conditions: If the price is greater than 100, “high_priced” should be set as 1, else 0. Provide the GREL expression used to perform this.

v. [2 pt] Create a new column “has_offer” with the values 0 or 1 based on the “item_description” column with the following conditions: If it contains the text “discount” or “offer” or “sale”, then set the value in “has_offer” as 1, else 0. Provide the GREL expression used to perform this.

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.a, c.i.b, c.ii, c.iii, c.iv and c.v. Provide each answer in a new line.

Q5 [15 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 2.7**.

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.

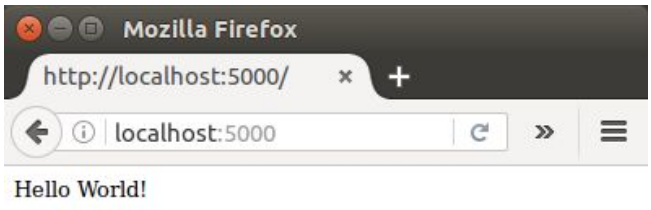
A few good tutorials for working with Flask, MySQL and jQuery in specific applications are listed below. You do not have to go through all of them to solve this question, however, they may give you clues that will help you solve the tasks at hand. These links are only for educational purposes - you do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

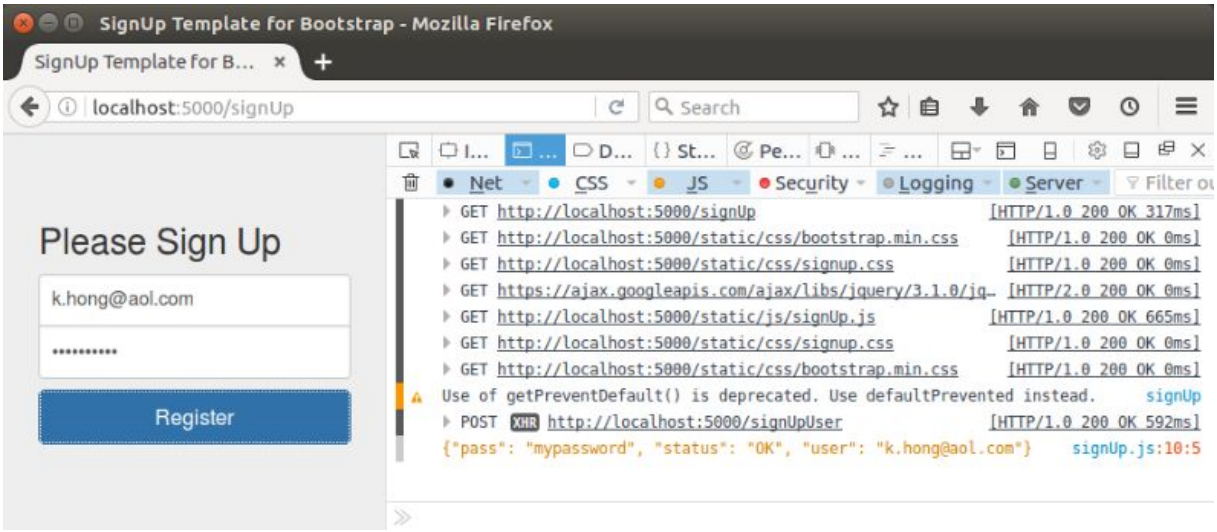
In the following questions, strive to create user interfaces that have both good usability (easy to use, easy to understand) and are aesthetically pleasing (it is a valued industry skill!). However, you will not lose credit for a poor looking website or gain more credit for a good looking one.

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!”. When you go to http://127.0.0.1:5000/, you should render **hello.html.**

   Example screenshots are available below:



hello.html



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

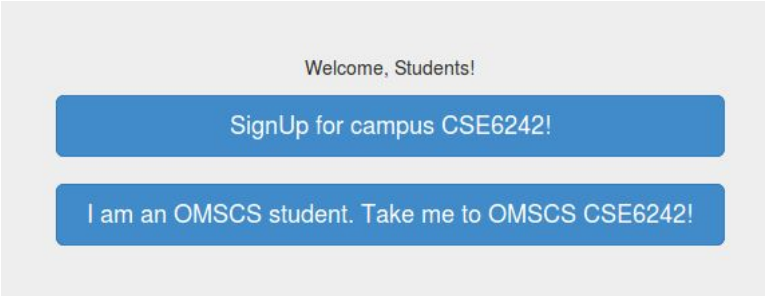
To run your website:

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.

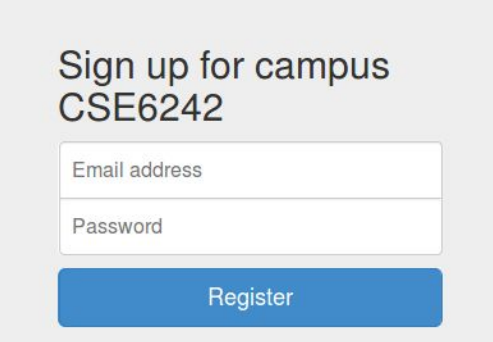
b. [3 pt] You will modify the landing page (**hello.html**) created in Q5.a to help potential CSE6242 students more easily navigate to the sign up page. You will create two buttons, one for campus students, and another for OMSCS students.

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 OMSCS students directly to the OMSCS version of CSE6242 (https://cse6242.gatech.edu/) (we do not handle that version of the course!).

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



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/cse6242/2018spring/).
      Hint: Consider using window.setTimeout and document.location.

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-LastName-FirstName/
    |--- Q1/
        |----  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
    |--- Q5/
        |----  app.py
        |----  templates/
            |-----  signup.html
            |-----  hello.html
        |----  static/
            |-----  css/
                |------  signup.css
                |------  bootstrap.min.css
            |-----  js/
                |------  signup.js
```