CSE 6242 / CX 4242: Data and Visual Analytics
Georgia Tech, Fall 2020

HW 1: End-to-end analysis of TMDb data, Argo-Lite, SQLite, D3 Warmup, OpenRefine, Flask

By 32+ awesome TAs of CSE6242A,Q,QSZ,OAN,O01,O3/CX4242A for our 1200+ students

Submission Instructions and Important Notes

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

1. Always check to make sure you are using the **most up-to-date assignment** (version number at bottom right of this document).
2. Submit a single zipped file, called “HW1-GTusername.zip” that unzips to a folder called “HW1-GTusername”, containing all the deliverables including source code/scripts, data files, and readme. Example: “HW1-jdoe3.zip” if GT account username is “jdoe3”. Your GT username is the **one with letters and numbers**. Only .zip is allowed; no other format will be accepted.
    a. At the end of this assignment, we have specified a folder structure that you must use to organize your files. 5 points will be deducted for not following this folder structure strictly.
    b. Due to the large class size, we may need to use auto-grading scripts to grade some of your deliverables, to help speed up grading, so we can return graded work to you sooner. Thus, it is extremely important that you strictly follow the instructions.
    c. Do not include any intermediate files you may have generated while working on the task, unless your work is absolutely dependent on it to get the final result — there are rarely any situations that would justify such a need.
    d. Wherever you are asked to write down an explanation for the task you perform, stay within the word limit or you may lose points.
3. 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.
4. 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.
5. You should submit your work by the official **Due** date on Canvas, the same date specified on the course schedule.
    a. Every homework assignment deliverable comes with a 48-hour “grace period”. You do **not** need to ask before using this grace period.
    b. You may re-submit your work before the grace period expires **without penalty**, but Canvas will mark your submission as **"late"**.
    c. Canvas automatically appends a “version number” to files that you re-submit. You do not need to worry about these version numbers, and there is no need to delete old submissions. **We will only grade the most recent submission.**
    d. Any deliverable submitted after the grace period will get **0 credit**. We recommend that you submit your work before the grace period begins.
    e. We will **not** consider late submission of any missing parts of a deliverable. To make sure you have submitted everything, download your submitted files to double check. If your submitting large files, you are responsible for making sure they get uploaded to the system in time. You have 48 hours to verify your submissions!

Download the HW1 Skeleton before you begin.

Homework Overview

Vast amounts of digital data are generated each day, but raw data are often not immediately “usable.” Instead, we are interested in the information content of the data: what patterns are captured? This assignment covers a few useful tools for acquiring, cleaning, storing, and visualizing datasets.

In Question 1 (Q1), you will collect data using an API for *The Movie Database* (TMDb). You will construct a graph representation of this data that will show which actors have acted together in various movies, and use *Argo Lite* to visualize this graph and highlight patterns that you find. This exercise demonstrates how visualizing and interacting with data can help with discovery.

In Q2, you will construct a TMDb database in SQLite, with tables capturing information such as how well each movie did, which actors acted in each movie, and what the movie was about. You will also partition and combine information in these tables in order to more easily answer questions such as "which actors acted in the highest number of movies?".

In Q3, you will visualize temporal trends in movie releases, using a JavaScript-based library called D3. This part will show how creating interactive rather than static plots can make data more visually appealing, engaging and easier to parse.

Data analysis and visualization is only as good as the quality of the input data. Real-world data often contain missing values, invalid fields, or entries that are not relevant or of interest. In Q4, you will use OpenRefine to clean data from Mercari, and construct GREL queries to filter the entries in this dataset.

Finally, in Q5, you will build a simple web application that displays a table of TMDb data on a single-page website. To do this, you will use Flask, a Python framework for building web applications that allows you to connect Python data processing on the back end with serving a site that displays these results.

Grading and Feedback

The maximum possible score for this homework is 100 points.

We will auto-grade Q1 and Q2 using the Gradescope platform. We believe our students (you all!) may benefit from being able to use Gradescope to obtain feedback as you work on these questions. **Using Gradescope is optional — you can complete these questions without using it.** If you decide to use Gradescope, **keep the following important points in mind**.

1. Every student will receive an email invitation to join Gradescope via the student’s email address listed on Canvas. We expect the invitations will arrive at your inboxes within a few hours after the release of this homework.
2. You may upload your code periodically to Gradescope to obtain feedback for your code. This is accomplished by having Gradescope auto-grade your submission using the same test cases that we will use to grade your work. The test cases’ results may help inform you of potential errors and ways to improve your code.
3. Your grades for Q1 and Q2 will be determined only based on what you submit to Canvas, as some students may not choose to use Gradescope to test their code. **We will ignore any code that you may have uploaded to Gradescope (and we will not grade it).** In other words, **if you decide to use Gradescope to test your work — and when you are happy with your work, you must submit that work via Canvas for us to grade it.**

4. When a lot of students use Gradescope, it is possible for it slow down or fail to communicate with the tester, and it can become even slower as the submission deadline approaches. You are responsible for submitting your work in time.

Q1 [40 points] Collect data from TMDb and visualize co-actor network

Q1.1 [30 points] Collect data from TMDb and build a graph

For this Q1.1, you will be using and submitting a python file**.** Complete all tasks according to the instructions found in **submission.py** to complete the Graph class, the TMDbAPIUtils class, and the two global functions. The Graph class will serve as a re-usable way to represent and write out your collected graph data. The TMDbAPIUtils class will be used to work with the TMDB API for data retrieval.

NOTE: You **must** only use a version of Python ≥ 3.7.0 and < 3.8 for this question. This question has been developed, tested for these versions. You **must not** use any other versions (e.g., Python 3.8). While we want to be able to extend to more Python versions, the specified versions are what we can definitively support at this time.

NOTE: You must only use the modules and libraries provided at the top of **submission.py** and modules from the Python Standard Library. Pandas and Numpy **CANNOT** be used — while we understand that they are useful libraries to learn, completing this question is not critically dependent on their functionality. In addition, to enable our TAs to provide better, more consistent support to our students, we have decided to focus on the subset of libraries.

NOTE: We will call each function once in **submission.py** during grading. You may lose some points if your program runs for unreasonably long time, such as more than 10 minutes during "non-busy" times. The average runtime of the code during grading is expected to take approximately 4 seconds. When we grade, we will take into account what your code does, and aspects that may be out of your control. For example, sometimes the server may be under heavy load, which may significantly increase the response time (e.g., the closer it is to HW1 deadline, likely the longer the response time!).

a) [10 pts] Implementation of the `Graph` class according to the instructions in **submission.py**

b) [10 pts] Implementation of the `TMDbAPIUtils` class according to the instructions in **submission.py.** You will use version 3 of the TMDb API to download data about actors and their co-actors. To use the TMDb API:
   - Create a TMDb account and obtain your client id / client secret which are required to obtain an authentication Token. Refer to this document for detailed instructions (log in using your GT account).
   - Refer to the TMDB API Documentation as you work on this question. The documentation contains a helpful 'try-it-out' feature for interacting with the API calls.

c) [10 pts] Producing correct **nodes.csv** and **edges.csv.** You must upload your **nodes.csv** and **edges.csv** file as directed in Q1.2.

NOTE: Q1.2 builds on the results of Q1.1

Q1.2 [10 points] Visualizing a graph of co-actors using Argo-Lite

Using Argo Lite, visualize a network of actors and their co-actors. You can access Argo Lite here

You will produce an Argo Lite graph snapshot your **`edges.csv`** and **`nodes.csv`** from Q1.1.c.

a. To get started, review Argo Lite's readme on GitHub. Argo Lite has been open-sourced.

b. Importing your Graph
   - Launch Argo Lite
   - From the menu bar, click 'Graph' → 'Import CSV'. In the dialogue that appears:
     - Select 'I have both nodes and edges file'
   - Under Nodes, use 'Choose File' to select **`nodes.csv`** from your computer
     - Leave 'Has Headers' selected
     - Verify 'Column for Node ID' is 'id'
   - Under Edges, use 'Choose File' to select **`edges.csv`** from your computer
     - Verify 'Column for Source ID' is 'source'
     - Select 'Column for Target ID' to 'target'
     - Verify 'Selected Delimiter' is ','
   - At the bottom of the dialogue, verify that 'After import, show' is 'All Nodes'
   - The graph will load in the window. Note that the layout is paused by default; you can select to 'Resume' or 'Pause' layout as needed.
   - Dragging a node will 'pin' it, freezing its position. Selecting a pinned node, right clicking it, then choosing 'unpin selected' will unpin that node, so its position will once again be computed by the graph layout algorithm. Experiment with pinning and unpinning nodes.

c. [7 points] Setting graph display options
   - On "Graph Options" panel, under 'Nodes' → 'Modifying All Nodes', expand 'Color' menu
     - Select Color by 'degree', with scale: 'Linear Scale'
     - Select a color gradient of your choice that will assign lighter colors to nodes with higher node degrees, and darker colors to nodes with lower degrees
   - Collapse the 'Color' options, expand the 'Size' options.
     - Select 'Scale by' to 'degree', with scale: Linear Scale'
     - Select meaningful Size Range values of your choice or use the default range.
   - Collapse the 'Size' options
   - On the Menu, click 'Tools' → 'Data Sheet'
   - Within the 'Data Sheet' dialogue:
     - Click 'Hide All'
     - Set '10 more nodes with highest degree'
     - Click 'Show' and then close the 'Data Sheet' dialogue
   - Click and drag a rectangle selection around the visible nodes
   - With the nodes selected, configure their node visibility by setting the following:
     - Go to 'Graph Options' → 'Labels'
     - Click 'Show Labels of Selected Nodes'
     - At the bottom of the menu, select 'Label By' to 'degree'
     - Adjust the 'Label Length' so that the full text of the actor name is displayed
   - On the Menu, click 'Tools' -> 'Filters' -> 'Show All Nodes' The result of this workflow yields a graph with the sizing and coloring depending upon the node degree and the nodes with the highest degree are emphasized by showing their labels.
   -

d. [3 points] Designing a meaningful graph layout

Using the following guidelines, create a visually meaningful and appealing layout:

- Reduce as much edge crossing as possible
- Reduce node overlap as much as possible
- Keep the graph compact and symmetric as possible
- Use the nodes' spatial positions to convey information (e.g., "clusters" or groups)
- Experiment with showing additional node labels. If showing all node labels creates too much visual complexity, show at least 10 "important" nodes. You may decide what "importance" mean to you. For example, you may consider nodes (actors) having higher connectivity as potentially more "important" (based on how the graph is built).

The objective of this task is to familiarize yourself with basic, important graph visualization features. Therefore, this is an **open-ended task**, and most designs receive full marks. So please experiment with Argo Lite's features, changing node size and shape, etc. In practice, it is not possible to create "perfect" visualizations for most graph datasets. The above guidelines are ones that generally help. However, like most design tasks, creating a visualization is about making selective design compromises. Some guidelines could create competing demands and following all guidelines may not guarantee a "perfect" design.

If you want to save your Argo Lite graph visualization snapshot locally to your device, so you can continue working on it later, we recommend the following workflow.

- Select 'Graph' → 'Save Snapshot'
    - In the 'Save Snapshot` dialog, click 'Copy to Clipboard'
    - Open an external text editor program such as TextEdit or Notepad. Paste the clipboard contents of the graph snapshot, and save it to a file with a **`.json`** extension. You should be able to accomplish this with a default text editor on your computer by overriding the default file extension and manually entering '.json'.
    - You may save your progress by saving the snapshot and loading them into Argo Lite to continue your work.
- To load a snapshot, choose 'Graph' → 'Open Snapshot'
- Select the graph snapshot you created.

NOTE: Q1.2 (d) will not be graded on Gradescope. We will give a qualitative score on the overall design and presentation of your graph visualization in Argo Lite.

e. Publish and Share your graph snapshot

- Select 'Graph ' → 'Publish and Share Snapshot' → 'Share'
- Next, click 'Copy to Clipboard' to copy the generated URL
- Return the URL in the `return_argo_lite_snapshot()` function in **`submission.py`**

If you modify your graph after you publish and share a URL, you will need to re-publish and obtain a new URL of your latest graph. Only the graph snapshot shared via the URL will be graded.

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

⊄ **`submission.py`**: the completed Python file

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 does not need to be parsed explicitly (unlike CSV files, which have to be parsed).

You will modify the given **Q2_SQL.py** file by adding SQL statements to it.

NOTE: You **must** only use a version of Python ≥ 3.7.0 and < 3.8 for this question. This question has been developed, tested for these versions. You **must not** use any other versions (e.g., Python 3.8)

NOTE: Do not modify the import statements, everything you need to complete this question has been imported for you. You may not use other libraries for this assignment.

A Sample class has been provided for you to see some sample SQL statements, you can turn off this output by changing the global variable SHOW to False. **NOTE:** This must be set to *false* before uploading to Gradescope and turning it in to Canvas.

GTusername - Please update the method GTusername with your credentials

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. [9 points] *Create tables and import data*.

i. [2 points] Create two tables (via two separate methods) named `movies` and `movie_cast` with columns having the indicated data types:

```
1. movies
    1. id (integer)
    2. title (text)
    3. score (real)
2. movie_cast
    1. movie_id (integer)
    2. cast_id (integer)
    3. cast_name (text)
    4. birthday (text)
    5. popularity (real)
```

ii. [2 points] Import the provided **movies.csv** file into the `movies` table and **movie_cast.csv** into the `movie_cast` table

1. You will write Python code that imports the `.csv` files into the individual tables. This will include looping though the file and using the **'INSERT INTO'** SQL command. Only use relative paths while importing files since absolute/local paths are specific locations that exist only on your computer and will cause the auto-grader to fail.

iii. [5 points] *Vertical Database Partitioning*. Database partitioning is an important technique that divides large tables into smaller tables, which may help speed up queries. For this question you will create a new table `cast_bio` from the `movie_cast` table (i.e., columns in `cast_bio` will be a subset of those in `movie_cast`) Do not edit the `movie_cast` table. Be sure that when you insert into the new `cast_bio` that the values are unique. Please read this page for an example of vertical database partitioning.

```
cast_bio
    1. cast_id (integer)
    2. cast_name (text)
    3. birthday (date)
```

```
4. popularity (real)
```

b. [1 point] *Create indexes.* Create the following indexes for the tables specified below. This step increases the speed of subsequent operations; though the improvement in speed may be negligible for this small database, it is significant for larger databases.

   i. `movie_index` for the `id` column in `movies` table
   ii. `cast_index` for the `cast_id` column in `movie_cast` table
   iii. `cast_bio_index` for the `cast_id` column in `cast_bio` table

c. [3 points] *Calculate a proportion*. Find the proportion of movies having a score > 50 and that has 'war' in the name. Treat each row as a different movie. The proportion should only be based on the total number of rows in the movie table. Format all decimals to two places using `printf()`. Do **NOT** use the `ROUND()` function as it does not work the same on every OS.

Output format and sample value:

```
7.70
```

d. [4 points] *Find the most prolific actors*. List 5 cast members with the highest number of movie appearances that have a popularity > 10. Sort the results by the number of appearances in descending order, then by `cast_name` in alphabetical order.

Output format and sample values (`cast_name,appearance_count`):

```
Harrison Ford,2
```

e. [4 points] *Find the highest scoring movies with the smallest cast.* List the 5 highest-scoring movies that have the fewest cast members. Sort the results by score in descending order, then by number of cast members in ascending order, then by movie name in alphabetical order. Format all decimals to two places using `printf()`.

Output format and sample values (`movie_title,movie_score,cast_count`):

```
Star Wars: Holiday Special,75.01,12
War Games,58.49,33
```

f. [4 points] *Get high scoring actors.* Find the top ten cast members who have the highest average movie scores. Format all decimals to two places using `printf()`.

- Sort the output by average score in descending order, then by `cast_name` in alphabetical order.
- Do not include movies with score <25 in the average score calculation.
- Exclude cast members who have appeared in two or fewer movies.

Output format and sample values (`cast_id,cast_name,average_score`):

```
8822,Julia Roberts,53.00
```

g. [6 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 appeared in at least 3 movies together AND the average score of these movies is >= 40.

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 pair” where the value of `cast_member_id1` is the same as that of `cast_member_id2`.

NOTE: Full points will only be awarded for queries that use joins for part g.

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 auto-grading.

NOTE: Do not submit any code that creates a ‘TEMP’ or ‘TEMPORARY’ view that you may have used for testing.

Optional Reading: Why create views?

i. [4 points] *Find the best collaborators.* Get the 5 cast members with the highest average scores from the `good_collaboration` view, 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`. Format all decimals to two places using `printf()`.

- Sort your output by this score in descending order, then by `cast_name` alphabetically.

  Output format (`cast_id,cast_name,collaboration_score`):

  ```
  2,Mark Hamil,99.32
  1920,Winoa Ryder,88.32
  ```

h. [4 points] SQLite supports simple but powerful Full Text Search (FTS) for fast text-based querying (FTS documentation). Import movie overview data from the **movie_overview.csv** into a new FTS table called `movie_overview` with the schema:

```
movie_overview
  ▪ id (integer)
  ▪ overview (text)
```

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.

i. [1 point] Count the number of movies whose `overview` field contains the word ‘fight’. Matches are not case sensitive. Match full words, not word parts/sub-strings.
e.g., Allowed: ‘FIGHT’, ‘Fight’, ‘fight’, ‘fight.’. Disallowed: ‘gunfight’, ‘fighting’, etc.

Output format:

```
12
```

ii. [2 points] Count the number of movies that contain the terms ‘space’ and ‘program’ in the `overview` field with no more than 5 intervening terms in between. Matches are not case sensitive. As you did in h(i)(1), match full words, not word parts/sub-strings. e.g., Allowed: ‘In Space there was a program’, ‘In this space program’. Disallowed: ‘In space you are not subjected to the laws of gravity. A program.’, etc.

Output format:

```
6
```

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

1. **`Q2_SQL.py`:** Modified file containing all the SQL statements you have used to answer parts a - h in the proper sequence.

Q3 [15 points] D3 (v5) Warmup

Read chapters 4-8 of Scott Murray's Interactive Data Visualization for the Web, 2nd edition (sign in using your GT account, e.g., jdoe3@gatech.edu). You may also briefly review chapters 1-3 if you need additional background on web development. **This simple reading provides important foundation** you will need for Homework 2. This question uses D3 version v5, while the book covers D3 v4. What you learn from the book is transferable to v5. In Homework 2, you will work with D3 extensively.

NOTE the following important points:

1. We highly recommend that you use the latest Firefox browser to complete this question. We will grade your work using **Firefox 79.0 (or newer)**.

2. For this homework, the D3 library is provided to you in the **lib** folder. You must **NOT** use any D3 libraries (d3*.js) other than the ones provided.

3. You may need to setup an HTTP server to run your D3 visualizations (depending on which web browser you are using, as discussed in the D3 lecture (OMS students: the video “Week 5 - Data Visualization for the Web (D3) - Prerequisites: JavaScript and SVG”. Campus students: see lecture PDF.). The easiest way is to use http.server for Python 3.x. **Run your local HTTP server in the hw1-skeleton/Q3 folder.**

4. We have provided sections of code along with comments in the skeleton to help you complete the implementation. While you do not need to remove them, you may need to write additional code to make things work.

5. All d3*.js files in the **lib** folder are referenced using **relative paths** in your html file. For example, since the file “Q3/index.html” uses d3, its header contains:
<script type="text/javascript" src="lib/d3/d3.min.js"></script>
It is incorrect to use an absolute path such as:
<script type="text/javascript" src="http://d3js.org/d3.v5.min.js"></script>

The 3 files that are referenced are:

- `lib/d3/d3.min.js`
- `lib/d3-dsv/d3-dsv.min.js`
- `lib/d3-fetch/d3-fetch.min.js`

6. For a question that reads in a dataset, you are required to submit the dataset too (as part of your deliverable). In your html / js code**,** use a **relative path** to read in the dataset file**.** For example, since Q3 requires reading data from the `q3.csv` file, the path should be 'q3.csv' and **NOT** an absolute path such as "C:/Users/polo/HW1-skeleton/Q3/q3.csv". Absolute/local paths are specific locations that exist only on your computer, which means your code will **NOT** run on our machines when we grade (and you will lose points).

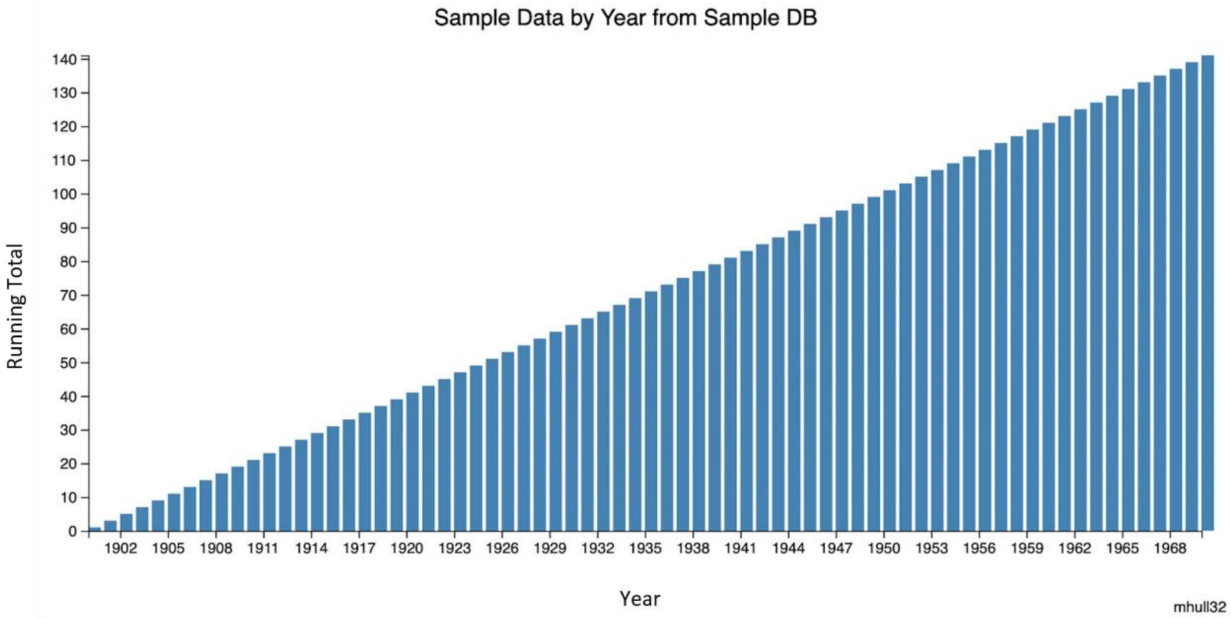
7. You can and are encouraged (though not required) to decouple the style, functionality and markup in the code for each question. That is, you can use separate files for CSS, JavaScript and HTML — this is a good programming practice in general.

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

- A folder named **lib** containing folders **d3, d3-fetch, d3-dsv**
- **q3.csv**: the file that we have provided you, in the hw1 skeleton under Q3 folder, which contains the data that will be loaded into the D3 plot.
- **index.(html / css / js)** : when run in a browser, it should display a barplot with the following specifications:
    a. [1.5 points] Load the data from `q3.csv` using D3 fetch methods. We recommend `d3.dsv()`.
    b. [2 points] The barplot must display one bar per row in the `q3.csv` dataset. Each bar corresponds to the running total of movies for a given year. The height of each bar represents the running total. The bars are ordered by ascending time with the earliest observation at the far left. i.e., 1880, 1890, ..., 2000
    c. [1 point] The bars must have the same fixed width, and there must be some space between two bars, so that the bars do not overlap.
    d. [3 points] The plot must have visible X and Y axes that scale according to the generated bars. That is, the axes are driven by the data that they are representing. Likewise, the ticks on these axes must adjust automatically based on the values within the datasets, i.e., they must not be hard-coded.
    e. [2 point] Set x-axis label to 'Year' and y-axis label to 'Running Total'.
    f. [1 point] Use a linear scale for the Y axis to represent the `running total` (recommended function: `d3.scaleLinear()`).
    g. [3 points] Use a time scale for the X axis to represent `year` (recommended function: `d3.scaleTime()`). It may be necessary to use time parsing / formatting when you load and display the `year` data. The axis would be overcrowded if you display every year value so set the X-axis ticks to display one tick for every 10 years.
    h. [1 point] Set the HTML title tag and display a title for the plot.
        - Position the title "Running Total of TMDb Movies by Year" above the barplot.
        - Set the HTML title tag (i.e., `<title>` Running Total of TMDb Movies by Year `</title>`).

i. [0.5 points] Add your GT username (usually includes a mix of letters and numbers) to the area beneath the bottom-right of the plot (see example image).

The barplot should appear similar in style to the sample data plot provided below.



Q4 [5 points] OpenRefine

OpenRefine is a Java application and requires Java JRE to run. Download and install Java if you do not have it (you can verify by typing ‘java -version’ in your computer’s terminal or command prompt).

a. Watch the videos on OpenRefine’s homepage for an overview of its features. Then, download and install OpenRefine release 3.3. Do not use version 3.4 (which is in beta status).

b. Import Dataset
- Run OpenRefine and point your browser at 127.0.0.1:3333.
- We use a products dataset from Mercari, derived from a Kaggle competition (Mercari Price Suggestion Challenge). If you are interested in the details, visit the data description page. We have sampled a subset of the dataset provided as "properties.csv".
- Choose "Create Project" → This Computer → `properties.csv`". Click "Next".
- You will now see a preview of the data. Click "Create Project" at the upper right corner.

c. Clean/Refine the data

NOTE: OpenRefine maintains a log of all changes. You can undo changes. Use the "Undo/Redo" button at the upper left corner. Follow the exact output format specified in every part below.

i. [0.5 point] 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 in `Q4Observations.txt`. Exclude these rows.

Output format and sample values:

```
i.rows: 500
```

ii. [1 point] 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 '/') and does not remove the original `category_name` column. Provide the number of new columns that are created by this operation, excluding the original `category_name` column.

Output format and sample values:

```
ii.columns: 10
```

NOTE: There are many possible ways to split the data. While we have provided one way to accomplish this in step ii, some methods could create columns that are completely empty. In this dataset, none of the new columns should be completely empty. Therefore, to validate your output, we recommend that you verify that there are no columns that are completely empty, by sorting and checking for null values.

iii. [0.5 points] Select the column `name` and apply the Text Facet (Facet → Text Facet). Cluster by using (Edit Cells → Cluster and Edit …) this opens a window where you can choose different "methods" and "keying functions" to use while clustering. Choose the keying function that produces the smallest number of clusters under the "Key Collision" method. Click 'Select All' and 'Merge Selected & Close'. Provide the name of the keying function and the number of clusters that was produced.

Output format and sample values:

```
iii.function: fingerprint, 200
```

NOTE: Use the default Ngram size when testing Ngram-fingerprint.

iv. [1 point] Replace the null values in the `brand_name` column with the text "Unknown" (Edit Cells -> Transform). Provide the General Refine Evaluation Language (GREL) expression used.

Output format and sample values:

```
iv.GREL_categoryname: endsWith("food", "ood")
```

v. [1 point] 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 90, `high_priced` should be set as 1, else 0. Provide the GREL expression used to perform this.

Output format and sample values:

```
v.GREL_highpriced: endsWith("food", "ood")
```

vi. [1 point] 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. Convert the text to lowercase before you search for the terms.

Output format and sample values:
`vi.GREL_hasoffer: endsWith("food", "ood")`

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

- **properties_clean.csv** : Export the final table as a comma-separated values (.csv) file.
- **changes.json** : Submit a list of changes made to file in json format. Use the "*Extract Operation History*" option under the Undo/Redo tab to create this file.
- **Q4Observations.txt** : A text file with answers to parts c.i, c.ii, c.iii, c.iv, c.v, c.vi. Provide each answer in a new line in the exact output format specified. Your file's final formatting should result in a .txt file that has each answer on a new line followed by one blank line (to help visually separately the answers)

Q5 [5 points] Introduction to Python Flask

Flask is a lightweight web application framework written in Python that provides you with tools, libraries and technologies to quickly build a web application. It allows you to scale up your application as needed.

You will modify the given file:

- wrangling_scripts/wrangling.py

NOTE: You **must** only use a version of Python ≥ 3.7.0 and < 3.8 for this question. This question has been developed, tested for these versions. You **must not** use any other versions (e.g., Python 3.8).

NOTE: You must only use the modules and libraries provided at the top of **submission.py** and modules from the Python Standard Library (except Flask). Pandas and Numpy **CANNOT** be used — while we understand that they are useful libraries to learn, completing this question is not critically dependent on their functionality. In addition, to enable our TAs to provide better, more consistent support to our students, we have decided to focus on the subset of libraries.

Username()- Update the username() method inside wrangling.py by including your GTUsername.

- Get started by installing Flask on your machine by running `pip install Flask` (Note that you can optionally create a virtual environment by following the steps here. Creating a virtual environment is purely optional and can be skipped.)

- To run the code, you must navigate to the Q5 folder in your terminal/command prompt and execute the following command: `python run.py`. After running the command go to http://127.0.0.1:3001/ on your browser. This will open up index.html showing a table in which the rows returned by data_wrangling() are displayed.

- You must solve the following 2 sub-questions:
    a. [2 points] Read the top 100 rows using the data_wrangling() method.

       NOTE: The skeleton code by default reads all the rows from movies.csv. You must add the required code to ensure reading only the **first** 100 rows. The skeleton code already handles reading the table header for you.

b. [3 points]: Sort the table in *descending* order of the values i.e., with larger values at the top and smaller values at the bottom of the table in the last (3rd) column.

Deliverables: Place the file listed below in the **Q5** folder

- **wrangling.py** : Submit wrangling.py file with your changes.

Extremely Important: folder structure & content of submission zip file

We understand that some of you may work on this assignment until just prior to the deadline, rushing to submit your work before the submission window closes. **Please take the time** to validate that **all files** are present in your submission and that you have not forgotten to include any deliverables! If a deliverable is not submitted, you will receive **zero** credit for the affected portion of the assignment — this is a very sad way to lose points, since you have already done the work!

You are submitting a single **zip** file named **HW1-GTusername.zip** (e.g., HW1-jdoe3.zip).

The files included in each question's folder have been clearly specified at the end of the question's problem description.

The zip file's folder structure must exactly be (when unzipped):

```
HW1-GTusername/
        Q1/
                submission.py
        Q2/
                Q2_SQL.py
        Q3/
                index.(html / js / css)
                q3.csv
                lib/
                        d3/
                                d3.min.js
                        d3-fetch/
                                d3-fetch.min.js
                        d3-dsv/
                                d3-dsv.min.js
        Q4/
                properties_clean.csv
                changes.json
                Q4Observations.txt
        Q5/
                wrangling.py
```