CSE 6242 / CX 4242: Data and Visual Analytics | Georgia Tech | Spring 2022

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

Contents

Important Notes

1. Submit your work by this assignment's official **Due** date on the course schedule.
    a. Every assignment has a 48-hour grace period. You may use it without asking us.
    b. Before the grace period expires, you may resubmit as many times as you need to.
    c. Submissions during the grace period will display as "late" and will not incur a penalty.
    d. We will not accept any submissions after the grace period.
2. Always use the **most up-to-date assignment** (version number at bottom right of this document).
3. This advanced course expects students to submit code that runs and is free of syntax errors. Code that does not run successfully will receive **0 credit**.
4. 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 the student's own answers**.
5. All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures, directly handled by the Office of Student Integrity (OSI). Consequences can be severe, e.g., academic probation or dismissal, a 0 grade for assignments concerned, and prohibition from withdrawing from the class**.**
6. Wherever you are asked for a written response, stay within the word limit or you may lose points.

Submission Instructions

Carefully read and follow the high-level instructions below, and the detailed instructions in each question.

1. **Submit ALL deliverables via Gradescope.** We will not accept submissions via any other channels.
2. Submit all the required files, as specified at the beginning of each question. Any deviations from the specified format (extra files, misnamed files, etc) will cause your submission to not be graded.
3. Each submission and its score will be recorded and saved by Gradescope. **By default, Gradescope uses your last submission for grading.** To use a different submission, **you MUST "activate" it** (click "Submission History" button at bottom toolbar, then "Activate").

Grading and Feedback

The maximum possible score for this homework is 100 points.

We will auto-grade all questions using the Gradescope platform. Based on our experience, students (you all!) benefit from using Gradescope to obtain feedback as they work on this assignment. **Keep the following important points in mind:**

1. **You can access Gradescope through Canvas.**
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. You must **not** use Gradescope as the primary way to test your code's correctness, since it provides only a few test cases, and error messages may not be as informative as local debuggers. Iteratively develop and test your code locally, write more test cases, and follow good coding practices. Use Gradescope mainly as a "final" check.
4. **Gradescope cannot run code that contains syntax errors. If Gradescope is not running your code, before seeking help, verify that:**
    a. Your code is free of syntax errors (by running it locally)
    b. All methods have been implemented
    c. You have submitted the correct file with the correct name
5. When many students use Gradescope simultaneously, it may slow down or fail to communicate with the tester. It can become even slower as the submission deadline approaches. You are responsible for submitting your work on time.

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.

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

Technology	Python **3.7.x only** (question developed and tested for these versions) TMDb API version 3 Argo Lite
Allowed Libraries	Python standard libraries **only**. Other libraries (including but not limited to Pandas and Numpy) are **NOT** allowed
Max runtime	10 minutes. Submissions exceeding this will receive **zero** credit.
Deliverables	[Gradescope] **`submission.py`**: the completed Python file (it **must** include your Argo Lite snapshot's URL)

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

For this Q1.1, you will use and submit 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.

Tasks and point breakdown

a) [10 pts] Implementation of the `Graph` class according to the instructions in **`submission.py`. Note that the graph is undirected,** thus **{a, b} and {b, a}** refer to the **same undirected edge** in the graph; **keep only either {a, b} or {b, a}** in the `Graph` object. A node's degree is the number of (undirected) edges incident on it. In-degrees and out-degrees are not defined for undirected graphs.

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 API:
   - Create a TMDb account and follow the instructions on this document to obtain an authentication token.
   - 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`** files to Argo Lite as directed in Q1.2.
   - Reminder: as mentioned in the Python file, if an actor name has comma characters (","), remove those characters before writing that name into the csv files (so Argo Lite can parse that name correctly 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 will produce an Argo Lite graph snapshot using your **`edges.csv`** and **`nodes.csv`** from Q1.1.c.

Tasks and point breakdown

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 set to 'All Nodes'
   - Dragging a node will 'pin' it, freezing its position. Double-clicking a pinned node unpins it, so its position will once again be computed by the graph layout algorithm. Experiment with pinning and unpinning nodes.

NOTE: If a malformed .csv is uploaded, Argo Lite will try and identify the error to help you fix it. Reload Argo Lite before reattempting to import any revised .csv files. Errors not recognized by Argo Lite could cause

it to become un-responsive. If you suspect this is the case, open the developer tools for your browser and review any console error messages.

c. [10 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’
   - On the ‘Data Sheet’ dialogue:
     - Click ‘Hide All’
     - Set ‘10 more nodes with highest degree’
     - Click ‘Show’ and then close the ‘Data Sheet’ dialogue
     - Close the 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 Selected’ (if you see ‘Hide Selected’, click it to update it to ‘Show Selected’)
     - At the bottom of the menu, select 'Label By' to ‘name'
     - Adjust the ‘Label Length’ so that the full text of the actor name is displayed
   - Show only non-leaf vertices. On the Menu, click ‘Tools’ → Data Sheet→ ‘Show k More Nodes with Highest Degree’. (where k is the input number of nodes such that **only nodes with a degree > 1 are visible**). To make this easier, we suggest writing a utility function in your `Graph` class to find the count of leaf nodes in order to determine how many nodes should be shown.
   - Show the labels of at least 10 nodes, ensuring that labels are shown for the 5 nodes with the highest degree. Do not show all of the node labels since it can create extreme visual complexity.

   The result of this workflow yields a graph design with the sizing and coloring depend upon the node degree, and the nodes with the highest degrees are emphasized by showing their labels.

   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.

d. Publish and share your graph snapshot
   - Name your graph: On the top navigation bar, click on the label 'Untitled Graph'. In the 'Rename Snapshot' dialogue window that appears, enter your GTUsername as the 'Snapshot Name' and click 'Done'
   - Select 'Graph ' → 'Publish and Share Snapshot' → 'Continue'
   - Click 'Copy to Clipboard' to copy the generated shapshot URL
   - Return the URL in the `return_argo_lite_snapshot()` function in **`submission.py`**
     - **NOTE:** If this function returns a malformed or invalid URL, Gradescope may crash.

   NOTE: 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.

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 must be parsed).

You will modify the given **Q2_SQL.py** file by adding SQL statements to it. We suggest that you consider testing your SQL locally on your computer using interactive tools to speed up testing and debugging, such as DB Browser for SQLite (https://sqlitebrowser.org).

Technology	Python **3.7.x only** (question developed and tested for these versions) SQLite release **3.22 only**
Allowed Libraries	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.
Max runtime	10 minutes. Submissions exceeding this will receive **zero** credit.
Deliverables	[Gradescope] **`Q2_SQL.py`**: Modified file containing all the SQL statements you have used to answer parts a - h in the proper sequence.

Tasks and point breakdown

NOTE: A sample class has been provided to show example SQL statements; you can turn off this output by changing the global variable SHOW from True to False. This **must** be set to *False* before uploading to Gradescope.

NOTE: In this question, you must only use INNER JOIN when performing a join between two tables, except for part g. Other types of joins may result in incorrect results.

GTusername — update the method GTusername with your credentials

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

i. [2 points] Create two tables (via two separate methods, part_ai_1 and part_ai_2, respectively in Q2_SQL.py) 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. You **must** 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. 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 the values are unique when inserting into the new `cast_bio` table. Read this page for an example of vertical database partitioning.

cast_bio
1. cast_id (integer)
2. cast_name (text)
3. birthday (text)
4. popularity (real)

b. [1 point] *Create indexes.* Create the following indexes. Indexes increase data retrieval speed; though the speed improvement may be negligible for this small database, it is significant for larger databases.

1. `movie_index` for the `id` column in `movies` table
2. `cast_index` for the `cast_id` column in `movie_cast` table
3. `cast_bio_index` for the `cast_id` column in `cast_bio` table

c. [3 points] *Calculate a proportion*. Find the proportion of movies having both a score > 50 and the substring 'war' in the name. Note that the 'war' search should be case-insensitive. Treat each row as a different movie. The proportion should be calculated as a percentage and 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 in some rare cases it works differently on different platforms.

Output format example:

```
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 row 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 intermediate result 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
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 decimal places using `printf()`.

- Sort the output by average score in descending order, then by `cast_name` in alphabetical order.
- **First** exclude movies with score <25 in the average score calculation.
- **Next** 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`.

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()`.

- Order your output by the printf-formatted `collaboration_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.

NOTE: If you have issues that fts is not enabled, try the following steps
1) Go to sqlite3 downloads page: https://www.sqlite.org/download.html
2) Download the dll file for your system
3) Navigate to your python packages folder, e.g.,
C:\Users\... ...\Anaconda3\pkgs\sqlite-3.29.0-he774522_0\Library\bin
4) Drop the downloaded .dll file in the bin.
5) In your IDE, import sqlite3 again, fts should be enabled."

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
```

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.

Technology	D3 Version 5 (included in the lib folder) Chrome 97.0 (or newer): the browser for grading your code Python http server (for local testing)
Allowed Libraries	D3 library is provided to you in the **lib** folder. You must **NOT** use any D3 libraries (d3*.js) other than the ones provided. In Gradescope, these libraries will be provided for you in the auto-grading environment.
Max runtime	NA
Deliverables	[Gradescope] **`submission.html:`** Modified file containing all html, javascript, and any css code required to produce the barplot. Do not include the D3 libraries or q3.csv dataset.

NOTE the following important points:

1. You will need to setup an HTTP server to run your D3 visualizations 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.**

2. 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.

3. All d3*.js files in the **lib** folder are referenced using **relative paths** in your html file. For example, since the file "Q3/submission.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`

```
10
```

4. In your html / js code**,** use a **relative path** to read the dataset file**.** For example, since Q3 requires reading data from the `q3.csv` file, the path must 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). Gradescope will provide a copy of the q3.csv dataset using the same directory structure provided in the HW skeleton.

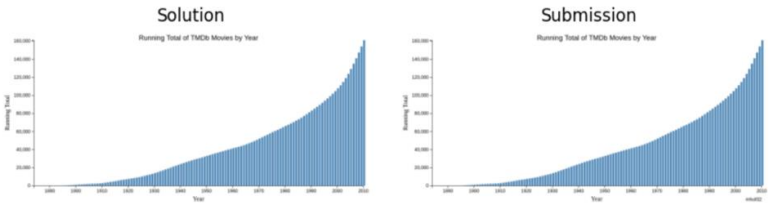
5. Load the data from `q3.csv` using D3 fetch methods. We recommend `d3.dsv()`. Handle any data conversions that might be needed, *e.g.,* strings that need to be converted to integer. See https://github.com/d3/d3-fetch#dsv

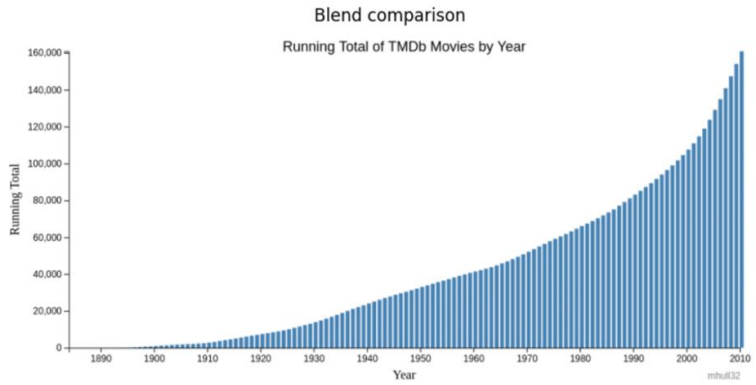
6. IMPORTANT: use the Margin Convention guide for specifying chart dimensions and layout. The autograder will assume this convention has been followed for grading purposes.

submission.html : when run in a browser, it should display a vertical barplot with the following specifications:

a. [3.5 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
b. [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.
c. [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. The x-axis must be a `<g>` element having the `id: "x_axis"` and the y-axis must be a `<g>` element having the `id: "y_axis"`.
d. [2 points] Set x-axis label to 'Year' and y-axis label to 'Running Total'. The x-axis label must be a `<text>` element having the `id: "x_axis_label"` and the y-axis label must be a `<text>` element having the `id: "y_axis_label"`.
e. [1 point] Use a linear scale for the Y axis to represent the `running total` (recommended function: `d3.scaleLinear()`).
f. [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.
g. [1 point] Set the HTML title tag and display a title for the plot. Set the HTML title tag (i.e., `<title>` Running Total of TMDb Movies by Year `</title>`). Position the title "Running Total of TMDb Movies by Year" above the barplot. The title must be a `<text>` element having the `id: "title"`
h. [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 GT username must be a `<text>` element having the `id: "credit"`

7. Gradescope will render your plot using Chrome and present you with a Dropbox link to view the screenshot of your plot with the solution plot in both a side-by-side and an overlay display.





The visual feedback helps you make adjustments and identify errors, *e.g.,* a blank plot likely indicates a serious error. It is not necessary that your design replicates the solution plot. We **require** the following DOM structure and sizing attributes for accurate comparisons:

```
<svg id="svg1"> plot
      | width: 960
      | height: 500
      |
      +-- <g id="container"> containing Q3.a plot elements
          |
          +-- <g id="bars"> containing bars
          |
          +-- <g id="x_axis"> x-axis
          |   |
          |   +-- (x-axis elements)
          |
          +-- <text id="x_axis_label"> x-axis label
          |
          +-- <g id="y_axis"> y-axis
          |   |
          |   +-- (y-axis elements)
          |
          +-- <text id="y_axis_label"> y-axis label
          |
          +-- <text id="credit"> GTUsername
          |
          +-- <text id="title"> chart title
```

Q4 [5 points] OpenRefine

Technology	OpenRefine 3.3 (DO NOT USE 3.4 OR 3.5)
Allowed Libraries	NA
Max runtime	NA
Deliverables	Gradescope: • **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)

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).

Watch the videos on OpenRefine's homepage for an overview of its features. Then, download and install OpenRefine release 3.3. The direct link to release 3.3 is https://github.com/OpenRefine/OpenRefine/releases/tag/3.3.

a. 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.

b. Clean/Refine the data

   NOTE: OpenRefine maintains a log of all changes. You can undo changes by 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. [0.5 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. [1.5 points] 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 (in the GREL expression) before you search for the terms.

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

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

Technology	Python **3.7.x only** (question developed and tested for these versions) Flask
Allowed Libraries	Python standard libraries Libraries already included in wrangling.py Any other libraries (including but not limited to Pandas and Numpy) are **NOT** allowed in this assignment
Max allowed runtime	NA
Deliverables	[Gradescope] **`wrangling.py:`** the completed python file with your changes

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 first 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 data 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.