CSE 6242 / CX 4242: Data and Visual Analytics | Georgia Tech | Fall 2025

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

Download the HW1 Skeleton before you begin.

Homework Overview

Vast amounts of digital data are generated each day, but raw data is often not immediately "usable". Instead, we are interested in the information content of the data such as what patterns are captured? This assignment covers useful tools for acquiring, cleaning, storing, and visualizing datasets. In question 1, we'll perform a simple end-to-end analysis using data from *The Movie Database* (TMDb). We will collect movie data via API, store the data in csv files. Q2 analyzes data using SQL queries with SQLite in Python. For Q3, we will complete a D3 warmup to prepare our students for visualization questions in HW2. Q4 & 5 will provide an opportunity to explore other industry tools used to acquire, store, and clean datasets.

The maximum possible score for this homework is **100 points.**

Contents

Important Notes

A. Submit your work by the due date on the course schedule.
    a. Every assignment has a generous 48-hour grace period, allowing students to address unexpected minor issues without facing penalties. You may use it without asking.
    b. Before the grace period expires, you may resubmit as many times as you need.
    c. TA assistance is not guaranteed during the grace period.
    d. Submissions during the grace period will display as “late” but **will not** incur a penalty.
    e. **We will not accept any submissions executed after the grace period ends**.
B. Always use the **most up-to-date assignment** (version number at the bottom right of this document). The latest version will be listed in Ed Discussion.
C. You may discuss ideas with other students at the "whiteboard" level (e.g., how cross-validation works, use HashMap instead of an array) and review any relevant materials online. However, **each student must write up and submit the student’s own answers**.
D. 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**.**

Submission Notes

A. All questions are graded on the Gradescope platform, accessible through Canvas.
B. We will not accept submissions anywhere else outside of Gradescope.
C. Submit all required files as specified in each question. Make sure they are named correctly.
D. You may upload your code periodically to Gradescope to obtain feedback on your code. **There are no hidden test cases.** The score you see on Gradescope is what you will receive.
E. You must **not** use Gradescope as the primary way to test your code. 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.
F. **Gradescope cannot run code that contains syntax errors**. If you get the “The autograder failed to execute correctly” error, verify:
    a. The code is free of syntax errors (by running locally)
    b. All methods have been implemented
    c. The correct file was submitted with the correct name
    d. No extra packages or files were imported
G. When many students use Gradescope simultaneously, it may slow down or fail. It can become even slower as the deadline approaches. You are responsible for submitting your work on time.
H. Each submission and its score will be recorded and saved by Gradescope. **By default, your last submission is used for grading.** To use a different submission, **you MUST “activate” it** (click the “Submission History” button at the bottom toolbar, then “Activate”).

Do I need to use the specific version of the software listed?

Under each question, you will see a set of technologies with specific versions - this is what is installed on the autograder and what it will run your code with. Thus, installing those specific versions on your computer to complete the question is highly recommended. You may be able to complete the question with different versions installed locally, but you are responsible for determining the compatibility of your code. **We will not award points for code that works locally but not on the autograder.**

Q1 [40 points] Collect data from TMDb to build a co-actor network

Leveraging the power of APIs for data acquisition, you will build a co-actor network of highly rated movies using information from The Movie Database (TMDb). Through data collection and analysis, you will create a graph showing the relationships between actors based on their highly rated movies. This will not only highlight the practical application of APIs in collecting rich datasets, but also introduce the importance of graphs in understanding and visualizing the real-world dataset.

Technology	• Python **3.10.x** • TMDb API version 3
Allowed Libraries	The Python Standard Library and Requests **only**.
Max runtime	10 minutes. Submissions exceeding this will receive **zero** credit.
Deliverables	• **Q1.py**: The completed Python file • **nodes.csv**: The csv file containing nodes • **edges.csv**: The csv file containing edges

Follow the instructions found in **`Q1.py`** to complete the Graph class, the TMDbAPIUtils class, and the one global function. 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

1. [10 pts] Implementation of the Graph class according to the instructions in **`Q1.py`**.
    a. **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/ out-degrees are not defined for undirected graphs.

2. [10 pts] Implementation of the `TMDbAPIUtils` class according to instructions in **`Q1.py`**. Use version 3 of the TMDb API to download data about actors and their co-actors. To use the API:
    a. Create a TMDb account and follow the instructions on this document to obtain an API key.
    b. **Be sure to use the key, not the token. This is the shorter of the two**.
    c. Refer to the TMDB API Documentation as you work on this question.

3. [20 pts] Build a co-actor network for movies released in 1999 according to the instructions in **`Q1.py`** and produce the correct **`nodes.csv`** and **`edges.csv`**.
    a. If an actor's name has comma characters (","), remove those characters before writing that name into the CSV files.

Q2 [30 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 that does not need to be parsed explicitly (unlike CSV files, which must be parsed). You can find instructions to install SQLite here. In this question, you will construct a TMDb database in SQLite, partition it, and combine information within tables to answer questions.

In this question, you will work with a dataset provided by TRAFFIC, an NGO working to ensure the global wildlife trade is legal and sustainable. TRAFFIC provides data through their interactive Wildlife Trade Portal, some of which we have already downloaded and pre-processed for you to utilize in Q2. You will modify the given **`Q2.py`** file by adding SQL statements to it. We suggest testing your SQL locally on your computer using interactive tools to speed up testing and debugging, such as DB Browser for SQLite.

Technology	• SQLite release **3.37.2** • Python **3.10.x**
Allowed Libraries	**Do not modify import statements**. Everything you need to complete this question has been imported for you. **Do not** use other libraries for this question.
Max runtime	10 minutes. Submissions exceeding this will receive **zero** credit.
Deliverables	• **`Q2.py`:** Modified file containing all the SQL statements you have used to answer parts a - h in the proper sequence.

IMPORTANT NOTES:

- If the **final** output asks for a decimal column, round it to 2 decimal places using the `ROUND()` SQL function
- 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.
- In this question, you must only use INNER JOIN when performing a join between two tables. Other types of joins may result in incorrect results.

Tasks and point breakdown

1. [6 pts] *Create tables and import data*.
    a. [3 pts] Create three tables named `incidents`, `details` and `outcomes` with columns having the indicated data types:
        i. `incidents`
            - `report_id (text)`
            - `category (text)`
            - `date (text)`
        ii. `details`
            - `report_id (text)`
            - `subject (text)`
            - `transport_mode (text)`
            - `detection (text)`
        iii. `outcomes`
            - `report_id (text)`
            - `outcome (text)`
            - `num_ppl_fined (integer)`
            - `fine (real)`
            - `num_ppl_arrested (integer)`
            - `prison_time (real)`
            - `prison_time_unit (text)`

    b. [3 pts] Import the three provided `.csv` files into the three tables created above.

i. Insert `incidents.csv` into the `incidents` table.
ii. Insert `details.csv` into the `details` table.
iii. Insert `outcomes.csv` into the `outcomes` table.

Write Python code that imports the `.csv` files into the individual tables. This will include looping through the file and using the **'INSERT INTO'** SQL command. You **must** only use relative paths to the Q2 directory while importing files, since absolute/local paths are specific locations that exist only on your computer; the auto-grader cannot access such locations and will fail.

2. [1 pt] *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.
   a. `incident_index` for the `report_id` column in the `incidents` table
   b. `detail_index` for the `report_id` column in the `details` table
   c. `outcomes_index` for the `report_id` column in the `outcomes` table

3. [2 pts] *Calculate a percentage*. Find the percentage of incidents that occurred between January 1st, 2018, and December 31st, 2020 (inclusive).
   - Output format and example value `(percentage)`: `50.41`

4. [3 pts] *Find the most common transport modes.* Identify the three most common transport modes that were detected by intelligence. Filter the detection column in the details table where the value is 'Intelligence'. Sort by most common transport mode to least common. Do not include null or empty transport modes.
   - Output format and example row values (`transport_mode, count`):
     `Sea 15`

5. [4 pts] *Identify detection methods with high arrest rates.* Identify the three detection methods with the highest number of average arrests across incidents. Only include incidents with one or more arrests in the average calculation. Only include detection methods with at least 100 incidents (with one or more arrests) in the final list. Sort by highest average to lowest.
   - Output format and example row values (`detection, count, avg_ppl_arrested`):
     `Dogs 15 4.21`

6. [4 pts] *List categories with the longest prison sentences.* Identify the incident categories with the highest average prison sentences in days. Use the `prison_time_unit` column to convert `prison_time` into days if necessary (assume 365 days in a year, 30 in a month, 7 in a week). Only include incident categories with more than 50 incidents. Sort by highest to lowest average.
   a. Output format and example row values (`category, count, avg_prison_time_days`):
      `"1. Seizure" 52 365.88`

7. [4 pts] *Creating a view.*
   a. [2 pts] Create a view (virtual table) called fines that lists report IDs, dates, the number of people fined, and the fine amount. Only include incidents where at least one person was fined. The view should have the following columns:
      - `report_id`
      - `date`
      - `num_ppl_fined`
      - `fine`

      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.

b. [2 pts] *Find years with the highest fine amounts*. Using the **fines** view, identify the 3 years with the most total fines given out and how many people they fined. Sort by highest to lowest fine amount.
   Output format and example row values (year, total_ppl_fined, total_fine_amount):`2017 300 8000000.55`

Optional Reading: Why create views?

8. [6 pts] *FTS*
   SQLite supports simple but powerful Full Text Search (FTS) for fast text-based querying (FTS documentation).

   a. Create a virtual table called incident_overviews using fts5 with the following columns
      - `report_id`
      - `subject`

NOTE: If you have issues that FTS is not enabled, try the following steps
- Go to sqlite3 downloads page: https://www.sqlite.org/download.html
- Download the dll file for your system
- Navigate to your Python packages folder, e.g., C:\Users\... ...\Anaconda3\pkgs\sqlite-3.29.0-he774522_0\Library\bin
- Drop the downloaded .dll file in the bin.

   b. Insert data from the `id` and `subject` columns in the details table into `incident_overviews`.
   c. Count the number of incidents that contain the terms 'dead' and 'pangolin' in the subject text with no more than two intervening terms between them. Matches are not case sensitive and should be full words, not substrings.
   Example:
   Allowed: "1 dead Sunda pangolin"
   Disallowed: "1 live pangolin seized and 2 dead birds found".
      - Output format and example (`count`):
        `15`

Q3 [15 points] D3 Warmup - Visualizing Wildlife Trafficking by Species

Using species-related data from the Wildlife Trade Portal, you will build a bar chart to visualize the most frequently illegally trafficked species between 2015 and 2023. Using D3, you will get firsthand experience with how interactive plots can make data more visually appealing, engaging, and easier to parse.

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). This reading provides an important foundation you will need for Homework 2. The question and autograder have been developed and tested for D3 version 5 (v5), while the book covers v4. What you learn from the book is transferable to v5, as v5 introduced few breaking changes. We also suggest briefly reviewing chapters 1-3 for background information on web development.

TRAFFIC International (2025) Wildlife Trade Portal. Available at www.wildlifetradeportal.org.

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.
Deliverables	• **`Q3.html`**: Modified file containing all html, javascript, and any css code required to produce the bar plot. Do not include the D3 libraries or q3.csv dataset.

IMPORTANT NOTES:

- Setup an HTTP server to run your D3 visualizations as discussed in the D3 lecture (OMS students: watch lecture video. 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.**
- We have provided sections of skeleton code and comments to help you complete the implementation. While you do not need to remove them, you need to write additional code to make things work.
- All d3*.js files are provided in the **lib** folder and referenced using **relative paths** in your html file. For example, since the file "Q3/Q3.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:
  a. `lib/d3/d3.min.js`
  b. `lib/d3-dsv/d3-dsv.min.js`
  c. `lib/d3-fetch/d3-fetch.min.js`
- 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 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. **As file paths are case-sensitive, ensure you correctly provide the relative path**.
- 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**.**
- **VERY IMPORTANT:** Use the Margin Convention guide to specify chart dimensions and layout.

Tasks and point breakdown

Q3.html: When run in a browser, should display a **horizontal** bar plot with the following specifications:

1. [3.5 pts] The bar plot must display one bar for each of the five most trafficked species by count. Each bar's length corresponds to the number of wildlife trafficking incidents involving that species between 2015 and 2023, represented by the 'count' column in our dataset.

2. [1 pt] The bars must have the same fixed thickness, and there must be some space between the bars, so they do not overlap.

3. [3 pts] 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. **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"`.

4. [2 pts] Set x-axis label to 'Count' and y-axis label to 'Species'. 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"`.

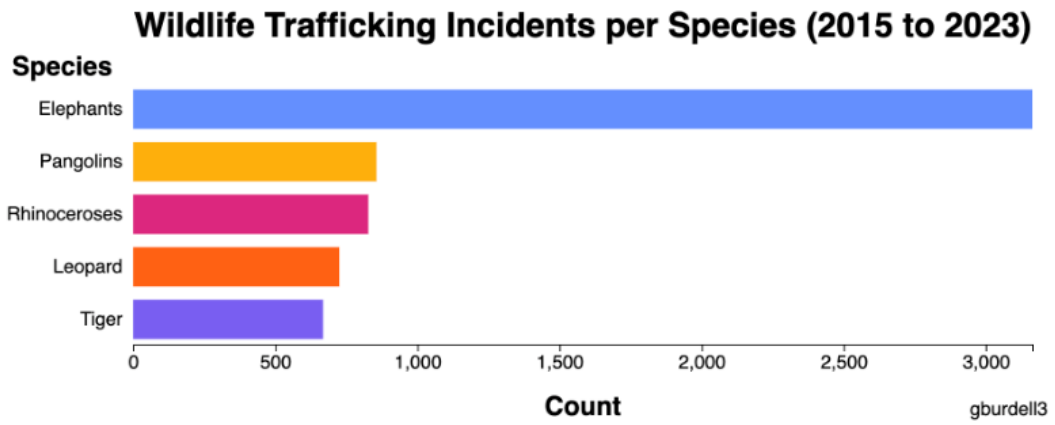
5. [2 pts] Use a linear scale for the X-axis to represent the count (recommended function: `d3.scaleLinear()`). Only display ticks and labels at every 500 interval. The X-axis must be displayed below the plot.

6. [2 pts] Use a categorical scale for the Y-axis to represent the species names (recommended function: `d3.scaleBand()`). Order the species names from greatest to least on 'Count' and limit the output to the top 5 species. The Y-axis must be displayed to the left of the plot.

7. [1 pt] Set the HTML title tag and display a title for the plot. **Those two titles are independent of each other and need to be set separately.** Set the HTML title tag (i.e., `<title>` Wildlife Trafficking Incidents per Species (2015 to 2023)`</title>`). Position the title "Wildlife Trafficking Incidents per Species (2015 to 2023)" above the bar plot. The title must be a `<text>` element having the `id: "title"`.

8. [0.25 pts] Add your GT username (usually includes a mix of letters and numbers) to the area beneath the bottom-right of the plot. The GT username must be a `<text>` element having the `id: "credit"`

9. [0.25 pts] Fill each bar with a unique color. We recommend using a colorblind-safe pallete.



NOTE: Gradescope will render your plot using Chrome and present you with a Dropbox link to view the screenshot of your plot as the autograder sees it. This visual feedback helps you adjust and identify errors, e.g., a blank plot indicates a serious error. Your design does not need to replicate the solution plot. However, the autograder **requires** the following DOM structure (including using **correct IDs** for elements) and sizing attributes to know how your chart is built.

```
<svg id="svg1"> plot
      | width: 900
      | height: 370
```

```
|
+-- <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 [10 points] OpenRefine

OpenRefine is a powerful tool for working with messy data, allowing users to clean and transform data efficiently. Use OpenRefine in this question to clean data around wildlife trafficking incidents. Construct GREL queries to identify incidents that occurred in airports.

OpenRefine is a Java application that requires Java JRE to run. However, OpenRefine v.3.6.2 comes with a compatible Java version embedded with the installer. So, there is no need to install Java separately when working with this version. Go through the main features on OpenRefine's homepage. Then, download and install OpenRefine 3.6.2. The link to release 3.6.2 is https://github.com/OpenRefine/OpenRefine/releases/tag/3.6.2

Technology	• OpenRefine **3.6.2**
Deliverables	• **`Q4.csv`**: Export the final table as a csv file. • **`history.json`**: Submit a list of changes made to file in json format. Go to 'Undo/Redo' Tab → 'Extract' → 'Export'. This downloads 'history.json'.

Tasks and point breakdown

Import Dataset

a. Run OpenRefine and point your browser at https://127.0.0.1:3333.
b. Choose "Create Project" → This Computer → `incidents.csv`. Click "Next".
c. You will now see a preview of the data. Click "Create Project" at the upper right corner.

1. [3 pts] Filter and Remove Data
   a. Use one facet to remove blank (null or empty) values from the 'Common Name' column.
   b. Use one text filter to remove any rows that do not mention "Airport" (case sensitive) in the 'Subject' column.
2. [7 pts] Identify Airports
   a. Create a column named 'Airport' based on the 'Subject' column that lists the full airport name mentioned.
      i. For example, a 'Subject' value of "77kg of sandalwood at the Chhatrapati Shivaji Maharaj International Airport, one Sudanese national" should create an 'Airport' value of "Chhatrapati Shivaji Maharaj International Airport".
      ii. There may not be a way to parse all airport names 100% correctly. Try to identify a heuristic that works in most cases.
      iii. There should not be any blank values in the 'Airport' column. Adjust your heuristic if it's a common issue. If there are rare edge cases, you can manually edit the offending cells.
   b. Cluster and merge similar airport names in the 'Airport' column. Select the 'Airport' column, go to 'Edit cells', then 'Cluster and edit'. Explore the various keying functions under the 'key collision' method and use the resulting clusters to merge variations of the same airport name.
      i. For example, "Detroit International Airport" and "Detroit Airport" can be merged into "Detroit International Airport". The value you choose to merge into does not matter – you can choose to merge into either "Detroit International Airport" or "Detroit Airport", but both values should not be present.

Q5 [5 points] Introduction to Python Flask

In this question, you will build a web application using Flask. Flask is a lightweight web application framework written in Python that provides you with tools, libraries, and technologies to build a web application quickly and scale it up as needed. The website will display wildlife trafficking data and allow users to filter and explore trafficking volume by different species classes. You will modify the given file: **`wrangling_scripts/Q5.py`**

Technology	Python **3.10.x** Flask
Allowed Libraries	Python standard libraries Libraries already imported in Q5.py
Deliverables	**`Q5.py`**: Completed Python file with your changes

Tasks and point breakdown

1. **username()** - Update the username() method inside **`Q5.py`** by including your GT username.

2. Install Flask on your machine by running `$ pip install Flask`
    a. You can optionally create a virtual environment by following the steps here. Creating a virtual environment is purely optional and can be skipped.

3. To run the code, 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 `index.html`, showing a table in which the rows returned by `data_wrangling()` are displayed. You can then choose different species classes from the dropdown and see how the data table updates dynamically.

4. You must solve the following two sub-questions:
    a. [2 pts] Generate a list of unique classes for options in the dropdown menu. Sort the list alphabetically.

    b. [3 pts] Filter, sort, and limit the data
        i. First, filter the data to only the specified class. If no class is specified, include all the data. You should still sort and limit if no class is specified.
        ii. Next, sort the data by the count column in descending order. You do not need to worry about tiebreaks.
        iii. Last, limit the data to only the top 10 rows. If the number of rows is fewer than 10 then return all rows.