CSE 6242 / CX 4242: Data and Visual Analytics | Georgia Tech | Spring 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 questions 1 & 2, 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, and analyze data using SQL queries. 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 [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 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.

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, format it to **two** places using `printf()`. Do **NOT** use the `ROUND()` function, as in rare cases, it works differently on different platforms. If you need to sort that column, be sure you sort it using the actual decimal value and not the string returned by printf.
- 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, except for part 7 and 8. Other types of joins may result in incorrect results.

Tasks and point breakdown

1. [9 points] *Create tables and import data*.
    a. [2 points] Create two tables (via two separate methods, part_ai_1 and part_ai_2, in **`Q2.py`**) named `movies` and `movie_cast` with columns having the indicated data types:
        i. `movies`
            1. `id (integer)`
            2. `title (text)`
            3. `score (real)`
        ii. `movie_cast`
            1. `movie_id (integer)`
            2. `cast_id (integer)`
            3. `cast_name (text)`
            4. `birthday (text)`
            5. `popularity (real)`

    b. [2 points] Import the provided **movies.csv** file into the `movies` table and **movie_cast.csv** into the `movie_cast` table
        i. 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. Make sure you use paths relative to the Q2 directory.

    c. [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. 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.

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

2. [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.
   a. `movie_index` for the `id` column in `movies` table
   b. `cast_index` for the `cast_id` column in `movie_cast` table
   c. `cast_bio_index` for the `cast_id` column in `cast_bio` table

3. [3 points] *Calculate a proportion*. Find the proportion of movies with a score between 7 and 20 (both limits inclusive). The proportion should be calculated as a percentage.
   a. Output format and example value:
      ```
      7.70
      ```

4. [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.
   a. Output format and example row values (`cast_name,appearance_count`):
      ```
      Harrison Ford,2
      ```

5. [4 points] *List the 5 highest-scoring movies*. In the case of a tie, prioritize movies with fewer cast members. Sort the result by score in descending order, then by number of cast members in ascending order, then by movie name in alphabetical order.
   a. Output format and example values (`movie_title,score,cast_count`):
      ```
      Star Wars: Holiday Special,75.01,12
      Games,58.49,33
      ```

6. [4 points] *Get high scoring actors.* Find the top ten cast members who have the highest average movie scores. Sort the output by `average_score` in descending order, then by `cast_name` alphabetically.
   a. Exclude movies with score < 25 before calculating `average_score`.
   b. Include only cast members who have appeared in three or more movies with score >= 25.
      i. Output format and example value (`cast_id,cast_name,average_score`):
         ```
         8822,Julia Roberts,53.00
         ```

7. [2 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 2 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 `cast_member_id1` is the same as `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?

8. [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`.
   a. Order your output by `collaboration_score` in descending order, then by `cast_name` alphabetically.
   b. Output format and example values(`cast_id,cast_name,collaboration_score`):
   ```
   2,Mark Hamil,99.32
   1920,Winoa Ryder,88.32
   ```

9. [4 points] SQLite supports simple but powerful Full Text Search (FTS) for fast text-based querying (FTS documentation).
   a. [1 point] 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
   - 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.
   - In your IDE, import sqlite3 again, fts should be enabled.

   b. [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.
      i. Example:
         Allowed: 'FIGHT', 'Fight', 'fight', 'fight.'
         Disallowed: 'gunfight', 'fighting', etc.
      ii. Output format and example value:
         12
   c. [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.
      i. Example:
         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.'
      ii. Output format and example value:
         6

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

In this question, you will utilize a dataset provided by TRAFFIC, an NGO working to ensure the global trade of wildlife 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 Q3. Using species-related data, 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 points] 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 point] The bars must have the same fixed thickness, and there must be some space between the bars, so they do not overlap.

3. [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. **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 points] 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"`.

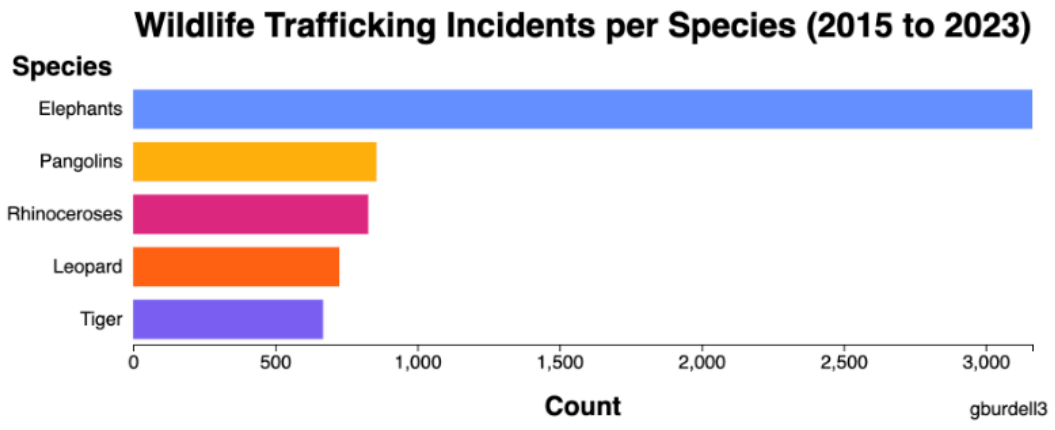
5. [2 points] Use a linear scale for the X-axis to represent the count (recommended function: `d3.scaleLinear()`). Only display ticks and labels at every 500 interval. The X-axis must be displayed below the plot.

6. [2 points] 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 point] 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 points] 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 points] 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 [5 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 from Mercari. Construct GREL queries to filter the entries in this dataset. 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	• **`properties_clean.csv`**: Export the final table as a csv file. • **`changes.json`**: Submit a list of changes made to file in json format. Go to 'Undo/Redo' Tab → 'Extract' → 'Export'. This downloads 'history.json' . Rename it to 'changes.json'. • **`Q4Observations.txt`**: A text file with answers to parts b.i, b.ii, b.iii, b.iv, b.v, b.vi. Provide each answer in a new line in the 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.

Tasks and point breakdown

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

2. [5 points] Clean/Refine the Data
   a. [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
      ```

   NOTE: OpenRefine maintains a log of all changes. You can undo changes by the "Undo/Redo" button at the upper left corner. You must follow all the steps in order and submit the final cleaned data file `properties_clean.csv`. The changes made by this step need to be present in the final submission. If they are not done at the beginning, the final number of rows can be incorrect and raise errors by the autograder.

   b. [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: While multiple methods can split data, ensure new columns aren't empty. Validate by sorting and checking for null values after using our suggested method in step b.

c. [0.5 points] Select the column `name` and apply the Text Facet (Facet → Text Facet). Cluster by using (Edit Cells → Cluster and Edit …), and then 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 "Merged Selected & Close." Provide the name of the keying function and number of clusters produced.

   Output format and sample values:

   `iii.function: fingerprint, 200`

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

d. [1 point] Replace the null values in the `brand_name` column with the text "Unknown" (Edit Cells → Transform). Provide the expression used.

   Output format and sample values:

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

NOTE: "Unknown" is case and space sensitive ("Unknown" is different from "unknown" and "Unknown ".)

e. [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")`

f. [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

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 points] Generate a list of unique animal classes for options in the dropdown menu. Sort the list alphabetically.

    b. [3 points] Filter, sort, and limit the data
        i. First, filter the data to only the specified class of animal. If no class is specified, include all the data.
        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.