cse6242-2019spring-hw1

CSE 6242 / CX 4242: Data and Visual Analytics Georgia Tech, Spring 2019

Homework 1: Analyzing The MovieDB data; SQLite; D3 Warmup; Gephi; OpenRefine

Prepared by our 30+ wonderful TAs of CSE6242A,Q,OAN,O01,O3/CX4242A for our 1200+ students

Submission Instructions and Important Notes:

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

❏ Always check to make sure you are using the most up-to-date assignment (version number at bottom right of this document).

❏ Submit a single zipped file, called "HW1-{GT username}.zip", containing all the deliverables including source code/scripts, data files, and readme. Example: "HW1-jdoe3.zip" if GT account username is "jdoe3". Only .zip is allowed (no other format will be accepted). Your GT username is the one with letters and numbers.

❏ You may discuss high-level ideas with other students at the "whiteboard" level (e.g., how cross validation works, use hashmap instead of array) and review any relevant materials online. However, each student must write up and submit his or her own answers.

❏ All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures (e.g., reported to and directly handled by the Office of Student Integrity (OSI)). Consequences can be severe, e.g., academic probation or dismissal, grade penalties, a 0 grade for assignments concerned, and prohibition from withdrawing from the class**.**

❏ At the end of this assignment, we have specified a folder structure you must use to organize your files in a single zipped file. 5 points will be deducted for not following this strictly.

❏ In your final zip file, do not include any intermediate files you may have generated to work on the task, unless your script is absolutely dependent on it to get the final result (which it ideally should not be).

❏ We may use auto-grading scripts to grade some of your deliverables, so it is extremely important that you strictly follow our requirements.

❏ Wherever you are asked to write down an explanation for the task you perform, stay within the word limit or you may lose points.

❏ Every homework assignment deliverable and every project deliverable comes with a 48-hour "grace period". Such deliverable may be submitted (and resubmitted) up to 48 hours after the official deadline without penalty. You do not need to ask before using this grace period.

❏ Any deliverable submitted after the grace period will get 0 credit.

❏ We will not consider late submission of any missing parts of a deliverable. To make sure you have submitted everything, download your submitted files to double check. If your submitting large files, you are responsible for making sure they get uploaded to the system in time. You have the whole grace period to verify your submissions!

Download the HW1 Skeleton before you begin. (The Q3 folder is initially empty.)

Grading

The maximum possible score for this homework is 100 points.

Q1 [40 points] Collecting and visualizing The Movie DB (TMDb) data

Q1.1 [25 points] Collecting Movie Data

You will use "The Movie DB" API version 3 to: (1) download data about movies and (2) for each movie, download its 5 similar movies.

You will write code using Python 3.7.x in **`script.py`** in this question. You will need an API key to use the TMDb data. Your API key will be an input to **`script.py`** so that we can run your code with our own API key to check the results. Running the following command should generate the CSV files specified in part b and part c:

```
python3 script.py <API_KEY>
```

Please refer to this tutorial to learn how to parse command line arguments. DO NOT leave your API key written in the code.

Note: You may only use the modules and libraries provided at the top the script.py file included in the skeleton for Q1 and modules from the Python Standard Library. Python wrappers (or modules) for the TMDb API may **NOT** be used for this assignment. Pandas and Numpy also may **NOT** be used --- while we understand that they are useful libraries to learn, completing this question is not critically dependent on their functionality. In addition, to make grading more manageable and to enable our TAs to provide better, more consistent support to our students, we have decided to restrict the libraries accordingly.

a. How to use TheMovieDB API:

- Create a TMDb account and request for an API key - https://www.themoviedb.org/account/signup. Refer to this document for detailed instructions.
- Refer to the API documentation https://developers.themoviedb.org/3/getting-started/introduction , as you work on this question.
- Use v3 of the API. You will need to use the v3 API key in your requests.

Note:

- The API allows you to make **40 requests every 10 seconds**. Set appropriate timeout intervals in your code while making requests. We recommend you think about how much time your script will run for when solving this question, so you will complete it on time.
- The API endpoint may return different results for the same request.
- You will be penalized for a runtime exceeding 5 minutes.

b. [10 points] Search for movies in the '**Drama**' genre released in the year 2004 or later. Retrieve the 350 most popular movies in this genre. The movies should be sorted from most popular to least popular. **Hint**: Sorting based on popularity can be done in the API call.

- Documentation for retrieving similar movies: https://developers.themoviedb.org/3/discover/movie-discover https://developers.themoviedb.org/3/genres/get-movie-list
- Save the results in **`movie_ID_name.csv`**.

Each line in the file should describe one movie, in the following format --- NO space after comma, and do not include any column headers:

```
movie-ID,movie-name
```

For example, a line in the file could look like:

```
353486,Jumanji: Welcome to the Jungle
```

Note:

- You may need to make multiple API calls to retrieve all movies. For example, the results may be returned in "pages," so you may need to retrieve them page by page.
- Please use the "primary_release_date" parameter instead of the "release_date" parameter in the API when retrieving movies released in the time period specified above. The "release_date" parameter will incorrectly return a movie if any of its release dates fall within the years listed.

c. [15 points] **Step 1: similar movie retrieval.** For each movie retrieved in part b, use the API to find its 5 similar movies. If a movie has fewer than 5 similar movies, the API will return as many as it can find. Your code should be flexible to work with however many movies the API returns.

Step 2: deduplication. After all similar movies have been found, remove all duplicate movie pairs. That is, if both the pairs `A,B` and `B,A` are present, only keep `A,B` where `A` < `B`. If you remove a pair due to duplication, there is no need to fetch additional similar movies for a given movie (That is, you should NOT re-run any part of Step 1). For example, if movie `A` has three similar movies `X`, `Y` and `Z`; and movie `X` has two similar movies `A` and `B`, then there should only be four lines in the file.

```
A,X
A,Y
A,Z
X,B
```

- Documentation for obtaining similar movies: https://developers.themoviedb.org/3/movies/get-similar-movies
- Save the results in **movie_ID_sim_movie_ID.csv**.

Each line in the file should describe one pair of similar movies --- NO space after comma, and do not include any column headers:

```
movie-ID,similar-movie-ID
```

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

- **movie_ID_name.csv:** The text file that contains the output to part b.
- **movie_ID_sim_movie_ID.csv:** The text file that contains the output to part c.
- **script.py**: The **Python 3.7** script you write that generates both **movie_ID_name.csv** and **movie_ID_sim_movie_ID.csv**.

Note : Q1.2 builds on the results of Q1.1. Specifically, Q1.2 asks that the "Source,Target" be added to the resulting file from Q1.1. If you have completed both Q1.1 and Q1.2, your csv would have the header row — please submit this file. If you have completed only Q1.1, but not Q1.2 (for any reasons), then please submit the csv file without the header row.

Q1.2 [15 points] Visualizing Movie Similarity Graph

Using Gephi, visualize the network of similar movies obtained. You can download Gephi here. Ensure your system fulfills all requirements for running Gephi.

a. Go through the Gephi quick-start guide.

b. [2 points] Manually insert `Source,Target` as the first line in **movie_ID_sim_movie_ID.csv**. Each line now represents a directed edge with the format `Source,Target`. Import all the edges contained in the file using **Data Laboratory in Gephi**.

Note: Ensure that **"create missing nodes"** option is selected while importing since we do not have an explicit nodes file.

c. [8 points] Using the following guidelines, create a visually meaningful graph:

- Keep edge crossing to a minimum, and avoid as much node overlap as possible.

- Keep the graph compact and symmetric if possible.
- Whenever possible, show node labels. If showing all node labels create too much visual complexity, try showing those for the “important” nodes. We recommend that you first run the Gephi’s built-in stat functions to gain more insight about a given node.
- Using nodes’ spatial positions to convey information (e.g., “clusters” or groups).

Experiment with Gephi’s features, such as graph layouts, changing node size and color, edge thickness, etc. The objective of this task is to familiarize yourself with Gephi; therefore this is a fairly open-ended task.

d. [5 points] Using Gephi’s built-in functions, compute the following metrics for your graph:

- Average node degree (run the function called “Average Degree”)
- Diameter of the graph (run the function called “Network Diameter”)
- Average path length (run the function called “Avg. Path Length”)

Briefly explain the intuitive meaning of each metric in your own words.
You will learn about these metrics in the “graphs” lectures.

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

- **For part b: `movie_ID_sim_movie_ID.csv`** (with `Source,Target` as its first line).
- **For part c:** an image file named “**`graph.png`**” (or “**`graph.svg`**”) containing your visualization and a text file named “**`graph_explanation.txt`**” describing your design choices, using no more than 50 words.
- **For part d:** a text file named “**`metrics.txt`**” containing the three metrics and your intuitive explanation for each of them, using no more than 100 words.

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 doesn’t need to be parsed explicitly (unlike CSV files, which have to be loaded and parsed).

You will modify the given **Q2.SQL.txt** file by adding SQL statements and SQLite commands to it.

We will autograde your solution by running the following command that generates **Q2.db** and **Q2.OUT.txt** (assuming the current directory contains the data files).

```
$ sqlite3 Q2.db < Q2.SQL.txt > Q2.OUT.txt
```

Since no auto-grader is bullet-proof, we ask that you be mindful of all the important points and notes below, which can cause the auto-grader to return an error. Our goal is to efficiently grade your assignment and return it as quickly as we can, so you can receive feedback and learn from the experience that you'll gain in this course.

- You will **not receive any points** if we are unable to generate the two output files above.
- You will **lose points** if you do not strictly follow the output format specified in each question below. The output format corresponds to the headers/column names for your SQL command output.

We have added some lines of code to the **Q2.SQL.txt** file for autograding purposes. **DO NOT REMOVE/MODIFY THESE LINES.** You will **not receive any points** if these statements are modified in any way (our autograder will check for changes). There are clearly marked regions in the **Q2.SQL.txt** file where you should add your code.

Examples of modifying the autograder code that can cause you to lose points:

- Putting any code or text of any kind, intentionally or unintentionally, outside the designated regions.
- Modifying, updating, or removing the provided statements / instructions / text in any way.
- Leaving in unnecessary debug/print statements in your submission. You may desire to print out more output than required during your development and debugging, but make sure to remove all extra code and text before submission.

Regrettably, we will not be releasing the auto-grader for your testing since that will likely invite unwanted attempts to game it. However, we have provided you with **Q2.OUT.SAMPLE.txt** with sample data that gives an example of how your final **Q2.OUT.txt** should look like after running the above command. Note that the sample data should not be submitted or relied upon for any purpose other than output reference and format checking. Avoid printing unnecessary output in your final submission as it will affect autograding and you will **lose points**.

WARNING: Do not copy and paste any code/command from this PDF for use in the sqlite command prompt, because PDFs sometimes introduce hidden/special characters, causing SQL error. This might cause the autograder to fail and you will lose points if such a case. You should manually type out the commands instead.

NOTE: For the questions in this section, you must only use INNER JOIN when performing a join between two tables. Other types of joins may result in incorrect results.

NOTE: Do not use `.mode csv` in your Q2.SQL.txt file. This will cause quotes to be printed in the output of each `SELECT … ;` statement.

a. *Create tables and import data*.

i. [2 points] Create two tables named '`movies`' and '`movie_cast`' with columns having the indicated data types:

- `movies`
  - `id` (integer)
  - `name` (text)
  - `score` (integer)
- `movie_cast`
  - `movie_id` (integer)
  - `cast_id` (integer)
  - `cast_name` (text)

ii. [1 point] Import the provided **movie-name-score.txt** file into the `movies` table, and **movie-cast.txt** into the `movie_cast` table. Use SQLite's `.import` command for this. Only use relative paths while importing files since absolute/local paths are specific locations that exist only on your computer and will cause the autograder to fail..

b. [2 points] *Create indexes.* Create the following indexes for the tables specified below. This step increases the speed of subsequent operations; though the improvement in speed may be negligible for this small database, it is significant for larger databases.

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

c. [3 points] *Calculate a proportion.* Find the proportion of movies having a score > 50. Treat each row as a different movie. The proportion should only be based on the total number of rows in the movie table.

Output format and sample value:

```
prop
77.7
```

d. [3 points] *Find the highest scoring movies*. List the seven best movies (highest scores). Sort your output by score from highest to lowest, then by name in alphabetical order.

Output format and sample value:

```
id,name,score
7878,Kirk Krandall,44
```

e. [3 points] *Find the most prolific actors*. List 5 cast members with the highest number of movie appearances (`movie_count`).

- Sort the number of appearances from highest to lowest.
- In case of a tie in the number of appearances, sort the results of `cast_name` in alphabetical order

Output format and sample value:

```
cast_id,cast_name,movie_count
68686,Harrison Ford,2
```

f. [6 points] *Get high scoring actors.* Find the top ten cast members who have the highest average movie scores.

- Sort your output by average score (from high to low).
- In case of a tie in the average score, sort the results of `cast_name` in alphabetical order.
- Do not include movies with score <50 in the average score calculation.
- Exclude cast members who have appeared in two or fewer movies.

Output format:

```
cast_id,cast_name,average_score
8822,Julia Roberts,53
```

g. [8 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 have 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 pairs (`cast_member_id1 == cast_member_id2`).

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

Remember that creating a view will not produce any output, so you should test your view with a few simple select statements during development. One such test has already been added to the code as part of the autograding.

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?

h. [4 points] *Find the best collaborators.* Get the 5 cast members with the highest average scores from the `good_collaboration` view made in the last part, 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`.

- Sort your output in a descending order of this score
- Sort by alphabetically by `cast_name` in case of a tie in the score

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

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

```
movie_overview (
            id integer,
            name text,
            year integer,
            overview text,
            popularity decimal)
```

NOTE: Create the table using **fts3** or **fts4** only. Also note that keywords like NEAR, AND, OR and NOT are case sensitive in FTS queries.

1. [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:
   ```
   count_overview
   ```

2. [2 points] List the `id`’s of the movies that contain the terms ‘love’ and ‘story’ 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 .

   Output format:
   ```
   id
   ```

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

- **`Q2.SQL.txt:`** Modified file containing all the SQL statements and SQLite commands you have used to answer parts a - i in the appropriate sequence.

Q3 [15 points] D3 Warmup and Tutorial

- Go through the online D3 v3 tutorial here.
- Complete steps 01-16 (Complete through “16. Axes”).
- Ensure that you are using v3 of the D3 lib.

Note: This is a simple and important tutorial which lays the groundwork for Homework 2. The latest D3 version is v5, but only the v3 tutorial is available online. What you learn in this v3 tutorial is transferable to v5. In Homework 2, you will work with D3 a lot; we’re upgrading Homework 2 to v5. All Georgia Tech students have FREE access to Safari Books, which include this and its updated v4 tutorial. https://www.safaribooksonline.com. Just log in with your GT account.

Note: We recommend using Mozilla Firefox or Google Chrome, since they have relatively robust built-in developer tools.

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

- A folder named **d3** containing file **d3.v3.min.js** (download)

- **index.html** : When run in a browser, it should display a scatterplot with the following specifications:

  a. [4 points] There should be 100 points using a circle shape that are randomly generated and placed on the plot. Each point's x coordinate should be a random number between 10 and 250 inclusively (i.e., [10, 250]), and so should each point's y coordinate. A point's x and y coordinates should be independently computed.

  b. [2 points] The plot must have visible X and Y axes that scale according to the generated points. The ticks on these axes should adjust automatically based on the randomly generated scatter-plot points.

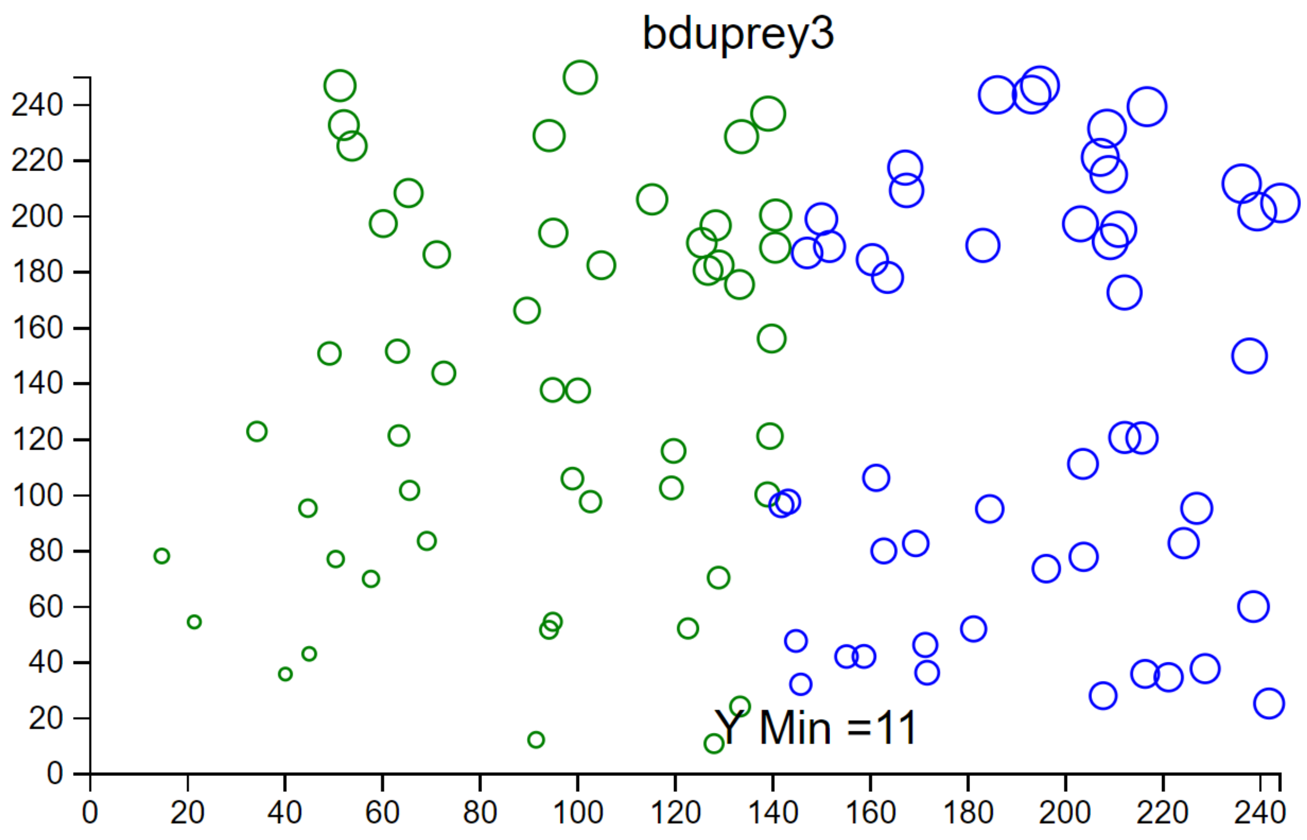
  c. [2 points] Use a single linear scale and apply it to both the X and Y coordinates of each circle to map the domain of X and Y values to the range of [1,5]. Set each circle's radius attribute to be the euclidean distance between the points (X, 0) and (0, Y) where X and Y denote the circle's **scaled** X and Y values.

  d. [3 points] All points with a scaled X value greater than the average of the scaled X value of all scatter-plot points should be outlined in **blue**. All other points should be outlined **green**. The points should use a transparent fill to enable visualization of overlapping points.
  Hint: Modify the 'stroke' parameter of the 'circle'.

  e. [3 points] It is often desirable to emphasize some important data points in a dataset. Accomplish this by displaying a text annotation for the point that contains the smallest y value. The annotation should read "Min Y: <value>" where <value> is the minimum y. If there are multiple points that contain the same minimum y value, you can pick whichever point you like to annotate. Optional: experiment with different approaches to style the annotation text, e.g., try using different positioning settings, font-sizes, font-weights, and/or color to make it stand out. **Use unscaled Y values here rather than scaled Y values.**

  f. [1 point] Your GT username, in lower-case, (e.g., jdoe3) should appear above the scatterplot. Also set the HTML title tag (e.g., <title>jdoe3</title>) to your GT username (also in lower-case).

The scatterplot should appear similar to but not exactly equivalent to the sample plot provided below. Remember that the plot will contain random data.



Note: No external libraries should be used. The index.html file can **only** refer to d3.v3.min.js within the d3 folder using the js file's **relative path**. Absolute/local paths are specific locations that exist only on your computer, which means your code won't run on our machines we grade (and you will lose points).

Q4 [10 pt] OpenRefine

a. Watch the videos on the OpenRefine's homepage for an overview of its features. Download and install OpenRefine (latest release : 3.1)

b. Import Dataset:

- Launch OpenRefine. It opens in a browser (127.0.0.1:3333).
- We use a products dataset from Mercari, derived from a competition on Kaggle (Mercari Price Suggestion Challenge). If you are interested in the details, please refer to the data description page. We have sampled a subset of the dataset as the given "properties.csv".
- Choose "Create Project" -> This Computer -> "properties.csv". Click "Next".
- You will now see a preview of the dataset. Click "Create Project" in the upper right corner.

c. Clean/Refine the data:
Note: OpenRefine maintains a log of all changes. You can undo changes. See the "Undo/Redo" button on the upper left corner.

i.a [1 pt] Select the "category_name" column and choose 'Facet by Blank' (Facet -> Customized Facets -> Facet by blank) to filter out the records that have blank values in this column. Provide the number of rows that return True. Remove these rows.

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

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

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

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

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

Note: There has been a slight confusion with c) v. and thus, we will be giving full credit to GREL statements that fulfill either of the following:
a) Look for the following words “sale”, “offer”, discount” without converting the original value in the “item description” column to lowercase
b) Look for the following words “sale”, “offer”, discount” after converting the original value in the “item description” column to lowercase

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

● **properties_clean.csv** : Export the final table as a comma-separated values (.csv) file.
● **changes.json** : Submit a list of changes made to file in json format. Use the “*Extract Operation History*” option under the Undo/Redo tab to create this file.
● **Q4Observations.txt** : A text file with answers to parts c.i.a, c.i.b, c.ii, c.iii, c.iv and c.v. Provide each answer in a new line.

Extremely Important: folder structure and content of submission zip file

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

You are submitting a single **zip** file named **HW1-{GT username}.zip**. The files included in each question’s folder have been clearly specified at the end of the question’s problem description.

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

```
HW1-{GT username}/
      Q1/
            movie_ID_name.csv
            movie_ID_sim_movie_ID.csv
            graph.png / graph.svg
            graph_explanation.txt
            metrics.txt
            script.py
      Q2/
            movie-cast.txt
            movie-name-score.txt
            movie-overview.txt
            Q2.SQL.txt
      Q3/
            index.html
            d3/
                  d3.v3.min.js
      Q4/
            properties_clean.csv
            changes.json
            Q4Observations.txt
```

Version 6