cse6242-2019fall-hw1

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

Homework 1: Analyzing Rebrickable Lego data; SQLite; D3 Warmup; Gephi; OpenRefine

Prepared by our 26+ wonderful TAs of CSE6242A,Q,OAN,O01,O3/CX4242A for our 1100+ 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-GTusername.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.

❏ Every homework assignment deliverable and every project deliverable comes with a 48-hour “grace period”. The *Due* date on Canvas represents the official deadline for the deliverable. The *Available Until* date on Canvas represents the official deadline with 48-hour grace period for the deliverable. 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.

❏ Every homework assignment deliverable and every project deliverable comes with a 48-hour “grace period”. Any deliverable submitted after the grace period will get zero credit. We recommend that you plan to finish by the beginning of the grace period in order to leave yourself time for any unexpected issues which might arise.

❏ We will not consider late submission of any missing parts of a homework assignment or project deliverable. To make sure you have submitted everything, download your submitted files to double check. You may re-submit your work before the grace period expires. Canvas automatically appends a “version number” to files that you re-submit. You do not need to worry about these version numbers, and there is no need to delete old submissions. We will only grade the most recent submission.

Download the HW1 Skeleton before you begin.

Grading

The maximum possible score for this homework is 100 points.

Introduction

In Questions 1, 2, and 3, you will perform data collection, exploration, and visualization of the extensive LEGO database available from 'Rebrickable'. In the following tasks, you will build both current and historical domain knowledge about LEGO themes, sets, and parts.

In Q1, we focus on collecting data using an API and then building a graph that shows the relationships between Sets and Parts. From this we can gain insights into what LEGO part is used the most frequently throughout various LEGO sets.

In Q2, you will work directly with the data files to build a portion of the Rebrickable database locally using SQLite. Next, you will explore the hierarchy of LEGO themes / sub-themes, as well as the historical growth of LEGO sets over time.

In Q3, you will visualize the growth of LEGO sets through the years. This will serve as an introduction to D3.

Q4 focuses on cleaning and preparing data for visualization.

Q1 [40 points] Collecting and visualizing Rebrickable Lego data

Q1.1 [25 points] Collecting Rebrickable Lego Data

You will use the "Rebrickable" API version 3 to: (1) download data about the Lego sets and (2) for each set, download the parts that comprise it.

You will write code using Python 3.7.x in **`script.py`** in this question. You will need an API key to use the Rebrickable 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 a **`.gexf`** graph file specified in Q1.1.d.

```
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. In general, it is good practice to not store any sensitive information like API keys and passwords as part of your 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. A module for creating a **`.gexf`** graph file is included in the skeleton and also imported at the top of the **`script.py`** file. Pandas and Numpy **CANNOT** be used --- while we understand that they are useful libraries to learn, completing this question is not critically

dependent on their functionality. In addition, to 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.

How to use the Rebrickable API

- Create a Rebrickable account and generate an API Key. Refer to this document for detailed instructions.
- Refer to the API documentation at https://rebrickable.com/api/ as you work on this question. Within the documentation you will find a helpful 'try-it-out' feature for interacting with the API calls.

Note: The API allows you to make **1 request every second**. 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. You may be penalized for a runtime exceeding 10 minutes. When we grade, we will take into account what your code does, and aspects that may be out of your control. For example, sometimes the Rebrickable server may be under heavy load, which may significantly increase the response time (e.g., the closer it is to HW1 deadline, likely the longer the response time!).

a. [10 points] Using the Rebrickable API, retrieve the top LEGO sets that have the most parts. Since this is a live database, the results may vary as you implement your solution. Adjust the parameters of the API calls such that you retrieve at least 270 and no more than 300 sets. The sets should be ordered by the number of parts they contain. For each set, you will need its:

- set number
- set name

Hints:

- Sorting on number of parts can be accomplished in the API call
- Adjust the `min_parts` parameter
- Set the `page_size` to be larger than the expected number of results to avoid pagination issues

Complete the following functions (necessary for us to grade your work).

- **`min_parts()`** in **script.py**
- **`lego_sets()`** in **script.py**

b. [5 points] Retrieving Parts for Lego Sets. For each set returned in part a, use the API to get a list of all inventory parts in that set. Since we are only interested in the parts that are used most frequently in a set, attempt to retrieve up to but no more than the top 20 parts for each set. For each part, you will need its:

- part color
- part quantity
- part name
- part number

To address the fact that some parts for a set have the same part_num, construct a unique id by concatenating the part number and color. e.g., A part having a part_num = "3203" and a color "C9C9C9" would be concatenated into an id = "3203_C9C9C9". You will use this part id as the node id when you add it to the graph in part c.

Note: Not all sets have 20 different parts. It is allowable to have fewer than 20 parts for a set.

Hint: Set the `page_size` parameter = 1000 when retrieving parts to avoid pagination issues.

c. [10 points] Constructing a graph using the **`pygexf`** library (included in skeleton under Q1→ gexf/) . Use the **`pygexf`** module to construct a graph that can be imported into the Gephi Open Graph Viz Platform software. You can review details about the .**gexf** file format here. You may also review a simple and a more complex graph created using the module. Instantiate and construct a static, undirected graph as follows:

Note: `script.py` includes an import statement for the pygexf library.

d. Declare a string-valued node attribute titled 'Type'. Add this attribute to each node in the graph. You will use node attributes to perform partitioning operations within Gephi.
- For nodes representing a [Lego] **set,** set the attribute value = "set"
- For nodes representing a [Lego] **part,** set the attribute value = "part"

- Each **set** should be added as a node to the graph
  - node id = set number retrieved in part a
  - node label = set name retrieved in part a
  - node color is set using RGB values of '0', '0', '0'
- Each **part** should be added as a node to the graph
  - node id = the part id you made by concatenating the part number and color in part b
  - node label = part name retrieved in part b
  - node color is set using the part color retrieved in part b. RGB values must be converted from the original hexadecimal representation in the data.
- Add an edge between each **part** and the **set**(s) it belongs to.
  - edge id = construct a unique id of your choosing.
  - edge source = set number retrieved in part a
  - edge target = the unique part id you constructed from the part number and color. retrieved in part b
  - edge weight = part quantity retrieved in part b

Note: Ensure that you do not add the same node more than once to the graph. **`pygexf`** has some functions that you can use to check this.

Use **`pygexf's`** `.write()` command to output a file named **`bricks_graph.gexf`**

Complete the following function (necessary for us to grade your work).
- **`gexf_graph()`** in **script.py**

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

Q1.2 [15 points] Visualizing a Lego Sets and Parts Graph

Using Gephi version 0.9.2, visualize the network of the Lego sets and their most used-parts. 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] Start Gephi and then use File→ Open to open **`bricks_graph.gexf.`** Within the import report dialogue window, ensure the graph type is set to 'undirected'. Under 'more options...', ensure that these boxes are checked:
- 'Auto-Scale'
- 'Create missing nodes'
- 'Self-loops'
- Leave the Edges merge strategy selected to 'Sum'. Ignore the GEXF version 1.2 deprecation warning.

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 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 label color, edge thickness, etc. The objective of this task is to familiarize yourself with Gephi; therefore this is a fairly open-ended task. It is not possible to create a "perfect" visualization for most graph datasets. The above

guidelines are ones that generally help. However, like most design tasks, creating a visualization is about making selective design compromises. Some guidelines could create competing demands, and following all guidelines may not guarantee a "perfect" design.

Hint:
Install more Layout plugins/algorithms through Tools → Plugins → Available Plugins. Check and install plugins in the 'Layout' category.

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")

You will learn about these metrics in the "graphs" lectures.

Complete the following functions for auto-grading purposes.

- **`avg_node_degree()`** in **script.py**
- **`graph_diameter()`** in **script.py**
- **`avg_path_length()`** in **script.py**

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

- **`script.py`:** The **Python 3.7** script you write that generates **`bricks_graph.gexf`** contains the 6 completed functions described in Q1.1.b, Q1.1.d, and Q1.2.d
- **`bricks_graph.gexf`:** The Gephi graph file output from **`script.py`** from Q1.1.d.
- an image file named "**`graph.png`**" (or "**`graph.svg`**") containing your visualization created in Gephi for Q1.2.c.
- Do not include the `pygexf/` directory. We will supply our own copy of this library during grading.

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 perfect, 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 document for use in the sqlite command prompt, because the document rendering 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. [4 points]*Create tables and import data*.
   i. [2 points] Create three tables named:
      - `sets`
      - `themes`
      - `parts`

with columns having the indicated data types:
- `sets`
  - `set_num` (text)
  - `name` (text)
  - `year` (integer)
  - `theme_id` (integer)
  - `num_parts` (integer)
- `themes`
  - `id` (integer)
  - `name` (text)
  - `parent_id` (integer)
- `parts`
  - `part_num` (text)
  - `name` (text)
  - `part_cat_id` (integer)
  - `part_material_id` (integer)

   ii. [2 points] Import the provided files as follows:
      - **sets.csv** file into the `sets` table
      - **themes.csv** file into the `themes` table
      - **parts.csv** file into the `parts` table

Use SQLite's `.import` command for this. Only use relative paths, e.g., `data/<file>.csv` while importing files since absolute/local paths are specific locations that exist only on your computer and will cause the autograder to fail..

b. [3 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.

- `sets_index` for the `set_num` column in `sets` table
- `parts_index` for the `part_num` column in `parts` table
- `themes_index` for the `id` column in `themes` table

c. [4 points] Required domain knowledge: LEGO sets belong to either a top level theme, e.g., 'Castle', 'Town', 'Space' or a theme → sub-theme hierarchy, e.g.,Town → Classic Town, Town → Outback, Town → Race.

i. [2 points] Create a view (virtual table) called `top_level_themes` that contains only the top level themes. This view must contain the `id` and `name` of any theme in the `themes` table that does not have a parent theme. You can check this condition by querying where the `parent_id = ''`

NOTE: the view you create here must **NOT** be a 'TEMP' view, nor a 'TEMPORARY' view.

Optional Reading: Why create views?

ii. [2 points] After creating the view, write a query that shows the total number of top level themes as `count` in the view you created.

Output format and sample value:

```
count
  57
```

d. [4 points] *Finding top level themes with the most sets.* Using the `top_level_themes` view that you created in part c.i, find the 10 top level themes that have the greatest number of sets (no need to consider a top level theme's child themes). Sort the output descending order from highest to lowest.

Output format and sample value:

```
theme,num_sets
Space,777
Town,755
Castle,333
...
```

e. [7 points] *Calculate a percentage*. Continue exploring top level themes using the `top_level_themes` view and the `sets` table. Write a query that expresses the number of sets from above as a percentage of the total number of sets that belong only to top level themes. The total number of sets would be the sum of all (not limited to top 10) num_sets from the part d.

List the themes and percentages, limiting the output only to themes with a percentage >= 5.00. Format all decimal values to 2 decimal places.

Output format and sample value:

```
theme,percentage
Space,10.30
Town,7.33
Castle,5.00
...
```

Hint:

You can format your decimal output using `printf()` as mentioned here: https://stackoverflow.com/questions/9149063/sqlite-format-number-with-2-decimal-places-always'

As a sanity check, you may manually verify (do not submit any verification code/sql) your percentage calculations against the following values that should not be included in the result:

```
theme,percentage
Disney Princess,1.01
Exo-Force,0.99
Collectible Minifigures,0.90
Designer Sets,0.88
Elves,0.86
```

f. [4 points] *Summarize a theme and its sub-themes*. As LEGO released more sets, some themes were subdivided into sub-themes. List each sub-theme and its total number of sets for the 'Castle' theme (Use the Castle theme with an id = 186). Sort the output by number of sets highest to lowest, then alphabetically.

Output format and sample value:

```
sub_theme,num_sets
  City,116
  Space Port,28
  Extreme Team,21
  ...
```

g. [6 points] Explore the growth of LEGO sets over time. From a historical standpoint, it's interesting to see the cumulative number of LEGO sets that have been released over time.

i. First, create a new view called `sets_years` that contains the `ROWID`, `year`, and number of sets (`sets_count`) released each year.

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.

ii. *Find the cumulative number of sets in the Rebrickable database for each year*. Using the view `sets_years`, find the cumulative number of sets for each year. e.g., if the first 3 sets were released in 1949 and 4 more sets released in 1950, then the cumulative values would be:

1949,3
1950,7

Sort your output by years in ascending order.

Output format and sample value:

```
year,running_total
1949,3
1950,7
1951,11
...
```

NOTE: The output to g.ii should match the data found in Q3/q3.csv. When you work on Q3, you will build a visualization using the Q3/q3.csv that we have provided.

h. [3 points] SQLite supports simple but powerful Full Text Search (FTS) for fast text-based querying (FTS documentation). Import lego data from the **parts.csv** into a new FTS table called `parts_fts` with the schema:

```
parts_fts(part_num (text),
name (text),
part_cat_id (integer),
```

`part_material_id` (integer))

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

i. [1 point] Count the number of unique parts as "`count_overview`" whose `name` field begins with the prefix 'mini'. A unique part is identified by a unique `part_num`. Matches are not case sensitive. Match words that begin with that prefix only. e.g., Allowed: 'Mini', 'mini', 'minifig', 'Minifig', 'minidoll', 'Minidoll'. Disallowed: 'undermined', 'administer', etc.

Output format and sample value:

```
count_overview
52
```

ii. [1 points] List the `part_num`'s of the unique parts as "`part_num_boy_minidoll`" that contain the terms 'minidoll' and 'boy' in the `name` field with no more than 5 intervening terms. Matches are not case sensitive. Contrary to what you did in h(i), match full words, not word parts/sub-strings. e.g., Allowed: 'minidoll gray hair boy', 'minidoll freckles boy', 'boy blue shirt minidoll'. Disallowed: 'minidoll gray hair yellow shirt blue pants boy', 'minidolllego blue pants boy', 'boylego minidoll', etc.

Output format and sample values:

```
Part_num_boy_minidoll
103
104
...
```

iii. [1 points] List the `part_num`'s of the unique parts as "`part_num_girl_minidoll`" that contain the terms 'minidoll' and 'girl' in the `name` field with no more than 5 intervening terms. Matches are not case sensitive. Similar to what you did in h(ii), match full words, not word parts/sub-strings .

Output format and sample values:

```
part_num_girl_minidoll
101
102
...
```

Deliverable: Place all the files listed below in the **Q2** folder. Do **NOT** include the data/ directory. We will supply our own copy of data during grading.

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

Q3 [15 points] D3 (v5) Warmup

Use Georgia Tech's library to access S. Murray's Interactive Data Visualization for the Web, 2nd edition (free for GT students).

- You may be prompted to sign in using your GT account. Click the 'Online Access' and/or 'O'Reilly Safari Ebooks'.

- Read chapters 4-8. You may briefly review chapters 1-3 if you require some additional background on web development.

- This reading is a simple but important reference that lays the groundwork for Homework 2. This assignment uses D3 version v5, while the book covers only v4. What you learn is transferable to v5. In

Homework 2, you will work with D3 extensively.

Note: We highly recommend that you use the latest Firefox browser to complete this question. We will grade your work using **Firefox 68.0 (or newer)**.

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

You may need to setup an HTTP server to run your D3 visualizations (depending on which web browser you are using, as discussed in the D3 lecture (OMS students: the video "Week 5 - Data Visualization for the Web (D3) - Prerequisites: Javascript and SVG". Campus students: see lecture PDF.). The easiest way is to use http.server for Python 3.x, or SimpleHTTPServer for Python 2.x. **Run your local HTTP server in the hw1-skeleton/Q3 folder**

All d3*.js files in the **lib** folder must be referenced using **relative paths**, e.g., **"lib/d3/<filename>**" in your html files (e.g., those in folders Q3, etc.). For example, since the file "Q3/index.html" uses d3, its header should contain:
<script type="text/javascript" src="lib/d3.v5.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 may be used are:
- `lib/d3/d3.min.js`
- `lib/d3-dsv/d3-dsv.min.js`
- `lib/d3-fetch/d3-fetch.min.js`

All questions that require reading from a dataset require you to submit the dataset in the deliverables too. In your html/js code**,** use a **relative path** to read in the dataset file**.** For example, since Q3 requires reading data from the `q3.csv` file, the path should simply 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 won't run on our machines we grade (and you will lose points).

You can and are encouraged (though not required) to decouple the style, functionality and markup in the code for each question. That is, you can use separate files for css, javascript and html -- this is a good programming practice in general.

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

- A folder named **lib** containing folders **d3, d3-fetch, d3-dsv**

- **q3.csv**: The file that we have provided you, in the hw1 skeleton under Q3 folder, which contains the data that will be loaded into the D3 plot. (Make sure you are using the provided q3.csv; do **NOT** use any output from Q2.g.ii .)

- **index.(html / css / js)** : When run in a browser, it should display a barplot with the following specifications:

  a. [3 points] Load the data from `q3.csv` using D3 fetch methods. We recommend that you use `d3.dsv()`.

  b. [2 points] The barplot must display one bar per row in the `q3.csv` dataset. Each bar corresponds to the running total of Lego sets 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., 1949, 1950, 1951, …, 2019.

  c. [1 point] The bars must have a fixed width and some spacing in between each bar so that the bars do not overlap.

  d. [3 points] The plot must have visible X and Y axes that scale according to the generated bars; i.e., the axes are driven by the data that they are representing. Likewise, the ticks on these axes adjust automatically based on the values within the datasets, i.e., they must not be hard-coded.

e. [1.5 points] Use a linear scale for the Y axis to represent the `running_total`.

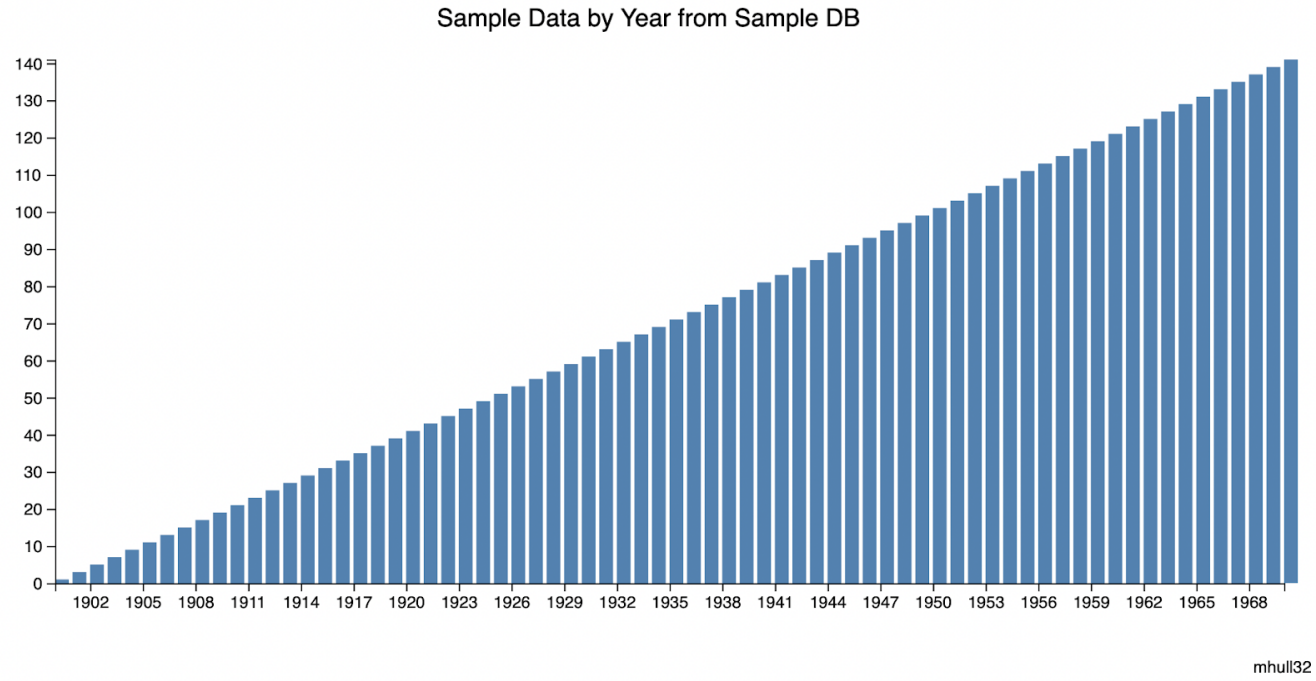
f. [3 points] Use a time scale for the X axis to represent the `year`. It may be necessary to use time parsing / formatting when you load and display the `year` data. The axis would be overcrowded if you display every year value so set the X-axis ticks to display one tick for every 3 years.

g. [1 point] Set the title tag and display a title for the plot.

- The title “Lego Sets by Year from Rebrickable” should appear above the barplot.
- Also set the HTML title tag (i.e., `<title>`Lego Sets by Year from Rebrickable`</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 barplot should appear similar in style to the sample data plot provided below.



Q4 [10 points] OpenRefine

a. Watch the videos on the OpenRefine’s homepage for an overview of its features.
Download and install OpenRefine (latest release: 3.2)

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 provided as "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. Follow the exact format requested for the output in each one of the parts below.

i. [1 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 in this operation, not including the original `category_name` column.

Output format and sample values:
```
ii.columns: 10
```

Note: There are many ways of splitting the data. While we have provided a specific way to accomplish this for 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, you should verify that there are no columns that are completely empty by sorting and checking for null values.

iii. [2 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 highest number of clusters under the "Key Collision" method. Click on 'Select All' and 'Merge Selected & Close'. Provide the name of the keying function that produces the highest number of clusters.

Output format and sample values:
```
iii.function: fingerprint
```

iv. [2 points] Replace the null values in the `brand_name`" with the text "Unbranded" (Edit Cells -> Transform). Provide the General Refine Evaluation Language (GREL) expression used.

Output format and sample values:
```
iv.GREL_brandname: endsWith("food", "ood")
```

v. [2 points] 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.

Output format and sample values:
```
v.GREL_highpriced: endsWith("food", "ood")
```

vi. [2 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. You will need to convert the text to lowercase before you search for the terms.

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

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

- **properties_clean.csv** : Export the final table as a comma-separated values (.csv) file.
- **changes.json** : Submit a list of changes made to file in json format. Use the "*Extract Operation History*" option under the Undo/Redo tab to create this file.

- **Q4Observations.txt** : A text file with answers to parts c.i, c.ii, c.iii, c.iv, c.v, c.vi. Provide each answer in a new line in the exact output format requested.

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-GTusername.zip** (e.g., HW1-jdoe3.zip).

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

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

```
HW1-GTusername/
      Q1/
            script.py
            bricks_graph.gexf
            graph.png or graph.svg

      Q2/
            Q2.SQL.txt
      Q3/
            index.(html / js / css)
            q3.csv
            lib/
                  d3/
                        d3.min.js
                  d3-fetch/
                        d3-fetch.min.js
                  d3-dsv/
                           d3-dsv.min.js

      Q4/
            properties_clean.csv
            changes.json
            Q4Observations.txt
```

Version 2