cse6242-2020spring-hw1

CSE 6242 / CX 4242: Data and Visual Analytics Georgia Tech, Spring 2020

Homework 1: “End-to-end” analysis of Rebrickable Lego data; SQLite; D3 Warmup; Argo Lite; OpenRefine

Prepared by our 32+ wonderful TAs of CSE6242A,Q,QSZ,OAN,O01,O3/CX4242A for our 1100+ students

Submission Instructions and Important Notes

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

1. Always check to make sure you are using the **most up-to-date assignment** (version number at bottom right of this document).
2. Submit a single zipped file, called “HW1-GTusername.zip” that unzips to a folder called “HW1-GTusername”, containing all the deliverables including source code/scripts, data files, and readme. Example: “HW1-jdoe3.zip” if GT account username is “jdoe3”. Your GT username is the one with letters and numbers. Only .zip is allowed; no other format will be accepted.
    a. At the end of this assignment, we have specified a folder structure you must use to organize your files. 5 points will be deducted for not following this strictly.
    b. Due to the large class size, we may need to use auto-grading scripts to grade some of your deliverables, to help speed up grading, so we can return graded work to you sooner. Thus, it is extremely important that you strictly follow the instructions.
    c. Do not include any intermediate files you may have generated while working on the task, unless your work is absolutely dependent on it to get the final result — there are rarely any situations that would justify such a need.
    d. Wherever you are asked to write down an explanation for the task you perform, stay within the word limit or you may lose points.
3. 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.
4. 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.
5. You should submit your work by the official **Due** date on Canvas, the same date specified on the course schedule.
    a. Every homework assignment deliverable comes with a 48-hour “grace period”. You do **not** need to ask before using this grace period.
    b. You may re-submit your work before the grace period expires **without penalty**, but Canvas will mark your submission as **"late"**.

c. 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.**

d. Any deliverable submitted after the grace period will get **0 credit**. We recommend that you submit your work before the grace period begins.

e. 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 48 hours to verify your submissions!

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 learn to perform data collection, exploration, and visualization of the extensive Lego database available from 'Rebrickable'. 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 Lego Sets and Parts. From this, we can gain insights into which Lego part is used the most across Lego sets.

In Q2, you will work directly with the Rebrickable database data files to build a portion of the Rebrickable database locally using SQLite. Next, you will explore the hierarchy of Lego themes and sub-themes, as well as the 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 [30 points] Collecting Rebrickable Lego Data and Exploring Lego Graph

For this Q1.1, you will be using and submitting a **Jupyter notebook.** Complete all tasks according to the instructions in the **`rebrickable.ipynb`** notebook.

You **must** only use a version of Python ≥ 3.7.5 and < 3.8 for this question. This question has been developed, tested for these versions. You **must not** use any other versions (e.g., Python 3.7.0).

WARNING: Do NOT add any cells to the Jupyter Notebook, because that will crash the autograder.

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

Note: You must only use the modules and libraries provided at the top of the **`rebrickable.ipynb`** `Graph` class included in the skeleton for Q1 and modules from the Python Standard Library. 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 enable our TAs to provide better, more consistent support to our students, we have decided to focus on the subset of libraries.

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 Rebrickable API Documentation 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 that you think about how much time your script will run for when solving this question, so your program will run quickly while complying the rate limit. You may lose some points if your program runs for unreasonably long time, such as more than 20 minutes during "nonbusy" times. 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!).

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

Q1.2 [10 points] Visualizing a Lego Sets and Parts Graph using Argo Lite

Using Argo Lite, visualize the network of the Lego sets and their most used-parts. You can access Argo Lite here

You will produce an Argo Lite graph snapshot for **`graph.csv`**, the file you produced in Q1.1 (b)(iv)

a. To get started, review the readme file of the Argo Lite GitHub repository. Argo Lite has been open-sourced.

b. Importing your Graph

- Launch Argo Lite
- Click 'Graph' → 'Import from .csv'. In the dialog that appears:
  - Select 'I have only edges file'
  - Choose **`graph.csv`** from your computer
  - Leave 'Has Headers' selected
  - Verify 'Column for Source ID' is 'Source'
  - Select 'Column for Target ID' to 'Target'
  - Verify Select 'Delimiter' is ','
- The graph will load in the window. Note that the layout is paused, you can select to 'Resume' or 'Pause' layout as needed.
- Dragging a node will 'pin' it, freezing its position. Selecting a pinned node, right clicking it, then choosing 'unpin selected' will unpin that node, so its position will once again be computed by the graph layout algorithm. Experiment with pinning and unpinning nodes.

c. [4 pts.] Setting Graph Display Options

- On the "Graph Options" panel, under 'Nodes' → 'Modifying All Nodes', expand the 'Color' menu
  - Select Color by 'degree', with scale: Linear scale
  - Select a color gradient of your choice that will assign lighter colors to nodes with higher node degrees, and darker colors to nodes with lower degrees
- Collapse the 'Color' options, expand the 'Size' options.
  - Select 'Scale By' to 'degree', with scale: 'Log Scale'
  - Select a size range or use the default values

Collapse the 'Size' options

- Go to 'Graph Options' → 'Labels'

- Select 'Show All Labels'
- At the bottom of the menu, select 'Label By' to 'degree'
- Click 'Hide All Labels'
- In the main window, select a few nodes to emphasize by showing their Label. This can be done by right-clicking on the node and selecting 'Toggle Labels'

d. [4 pts.] Designing a meaningful graph layout
Using the following guidelines, create a visually meaningful and appealing layout:

- Reduce as much edge crossing as possible
- Reduce node overlap as much as possible
- Keep the graph compact and symmetric if possible
- Whenever possible, show node labels. If showing all node labels creates too much visual complexity, try showing those for the “important” nodes
- Use nodes’ spatial positions to convey information (e.g., “clusters” or groups)

Experiment with Argo Lite’s features, changing node size and shape, etc. The objective of this task is to familiarize yourself with basic, important graph visualization features. Therefore this is an 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.

e. [1 pt] Saving your graph and working with graph snapshots.

- Select 'Graph' → 'Save Snapshot'
  - In the 'Save Snapshot` dialog, click 'Copy to Clipboard'
  - Open an external text editor program such as TextEdit or Notepad. Paste the clipboard contents of the graph snapshot, and save it to a file named **`graph.json`**. You should be able to accomplish this with a default text editor on your computer by overriding the default file extension and manually entering ‘.json’.
  - You may save your progress by saving the snapshot and loading them into Argo Lite to continue your work.
- To load a snapshot, choose 'Graph' → 'Open Snapshot'
- Select the graph snapshot you created.

f. [1 pt] Publish and Share your graph

- Select 'Graph ' → 'Publish and Share Snapshot' → 'Share’
- Next, click 'Copy to Clipboard' to copy the generated URL
- Paste the URL in the text file **`argo_lite_url.txt`**

Only the graph shared via the URL will be graded. The snapshot .json file mainly serves as a backup and it will not be graded by default — we will only review the json file under extremely rare circumstances (e.g., URL is corrupt or server hosting the URL is down)

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

- **`rebrickable.ipynb`** : the completed Jupyter Notebook file
- **`graph.csv`** : the edges output you produced in **`rebrickable.ipynb`**
- **`graph.json`** : the graph snapshot you created in Argo Lite from **`graph.csv`**
- **`argo_lite_url.txt`** : the url of your shared graph in Argo Lite

Q2 [35 points] SQLite

SQLite is a lightweight, serverless, embedded database that can easily handle multiple gigabytes of data. It is one of the world’s most popular embedded database systems. It is convenient to share data stored in an SQLite database — just one cross-platform file which does not need to be parsed (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 want to grade your work and provide feedback to you as quickly as we can, to help you more effectively learn new materials from this course. Thus, 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 request that you be mindful of all the important information below, which can cause the auto-grader to return an error.

1. You will **not receive any points** if we are unable to generate the two output files above.
2. 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 OR MODIFY THOSE LINES.** You will **not receive any points** if those 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:

- Putting 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.

This semester, for your convenience and education purposes, we will provide you with an (experimental) output checker. This is our first time designing and developing such a tool, so it will not be perfect. We will announce the availability of the tool on Piazza.

IMPORTANT: The output checker does not grade your work. It mainly checks for potential glaring issues. A piece of code that passes the checker is not guaranteed to receive full credit since there can be many ways (e.g., programming logic) that would produce code that passes the checker. To help us focus on providing timely help to all students, the instruction team will not support the checker outside of Piazza (e.g., requests through emails and Slack messages will be ignored).

WARNING: Do not copy and paste any code or commands from this document for use in the sqlite command prompt, because the document rendering sometimes introduce hidden or special characters, causing SQL errors. This might cause the autograder to fail, leading to loss of points. 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 (if you can, otherwise as many as you can) top level themes that have the greatest number of sets (no need to consider a top level theme's child themes). Sort the output by descending order, from highest to lowest. Limit the results to those top level themes that have more than 25 sets.

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. Sort percentages in descending order.

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

f. [4 points] Summation of sub-theme sets. As Lego released more sets, some themes were subdivided into sub-themes. List each sub-theme and its total number of sets. Sort the output by number of sets highest to lowest, then alphabetically. Limit the number of sets to those that have a count greater than 5.

Output format and sample value:

```
sub_theme,num_sets
  Soccer,22
  4.5V,16
  Star Wars Rogue One,81
  …
```

g. [6 points] Explore the growth of Lego sets over time. From a historical standpoint, it is 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 we recommend that 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. For example, 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 and limit your results to those sets that were released in the 80's (1980 to 1989).

Output format and sample value:

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

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] Count the total number of `part_num`'s of the unique parts as "`total_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:

```
total_boy_minidoll
103
```

iii. [1 points] Count the total number of `part_num`'s of the unique parts as "`total_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:

```
total_girl_minidoll
101
```

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

Through Georgia Tech library, access Scott 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 '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 72.0 (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`

For a question that reads in a dataset, you are required to submit the dataset too (as part of your deliverable). 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 will **NOT** run on our machines when 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 there must be some space between two bars, 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. That is, the axes are driven by the data that they are representing. Likewise, the ticks on these axes must adjust automatically based on the values within the datasets, i.e., they must not be hard-coded.

  e. [1.5 points] Use a linear scale for the Y axis to represent the `running_total` (recommended function: d3.scaleLinear()).

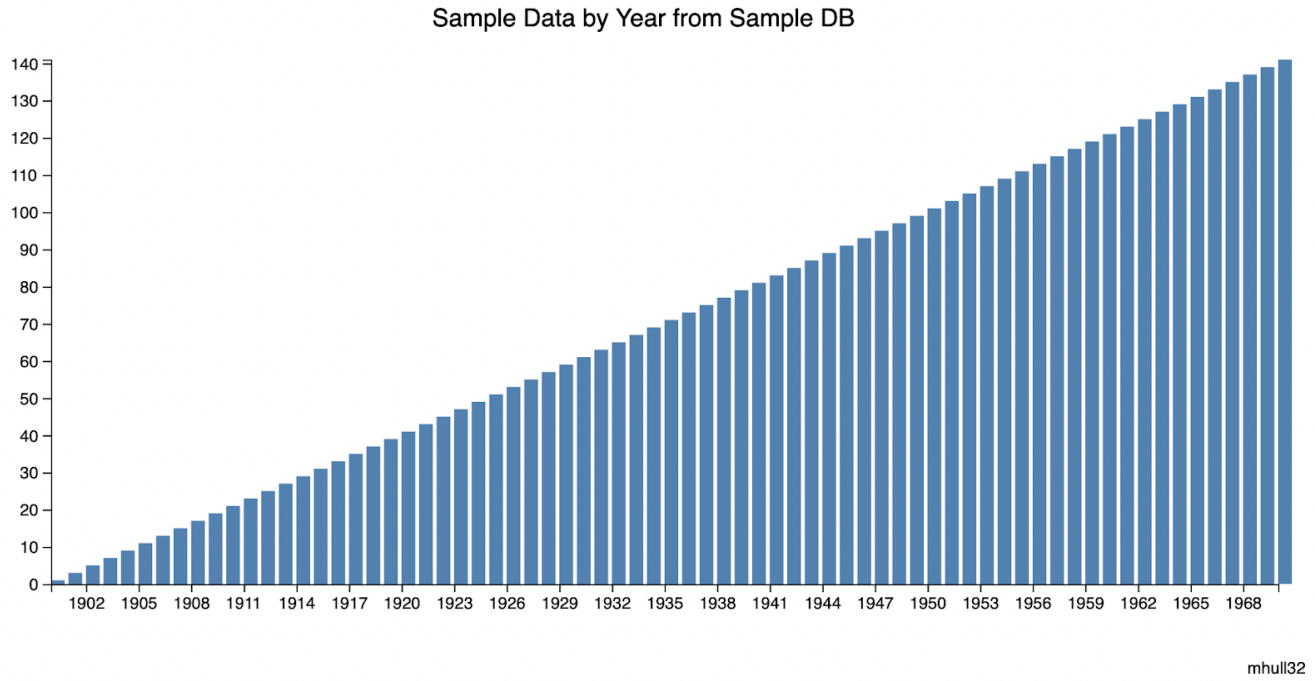
  f. [3 points] Use a time scale for the X axis to represent `year` (recommended function: d3.scaleTime()). It may be necessary to use time parsing / formatting when you load and display the `year` data. The axis would be overcrowded if you display every year value so set the X-axis ticks to display one tick for every 3 years.

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

- Position the title “Rebrickable Lego Sets by Year” above the barplot.
- Set the HTML title tag (i.e., `<title>Rebrickable Lego Sets by Year</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

OpenRefine is a Java application and requires Java JRE to run. Download and install Java if you do not have it (you can verify by typing ‘java -version’ in your computer’s terminal or command prompt).

a. Watch the videos on OpenRefine’s homepage for an overview of its features. Then, download and install OpenRefine (release: 3.2). Do not use version 3.3 RC1.

b. Import Dataset

- Run OpenRefine and point your browser at 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. Use the "Undo/Redo" button at 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 by this operation, excluding the original `category_name` column.

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

Note: There are many possible ways to split the data. While we have provided one way to accomplish this in step ii, some methods could create columns that are completely empty. In this dataset, none of the new columns should be completely empty. Therefore, to validate your output, we recommend that you verify that there are no columns that are completely empty, by sorting and checking for null values.

iii. [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. Use the default Ngram size when testing Ngram-fingerprint. Click ‘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` column 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 90, `high_priced` should be set as 1, else 0. Provide the GREL expression used to perform this.

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

vi. [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. **Please take the time** to validate that **all files** are present in your submission and that you have not forgotten 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 have 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/
            rebrickable.ipynb
            graph.csv
            graph.json
            argo_lite_url.txt

      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