CS513: Data Cleaning Project Report

Dataset: New York Public Library Menus data

Tools: OpenRefine, Python, SQLite, YesWorkflow, Tableau

Team: Parul Mainwal (mainwal2@illinois.edu)

Rohit Bansal (rbansal3@illinois.edu)

**Table of Contents**

1. Introduction:

2. Overview and initial assessment

2.1 Structure & content of the dataset:

2.2 Quality issues

2.3 Use cases of the dataset

2.4 Data cleaning goals

3. Data cleaning with OpenRefine & Python:

3.1 Menu.csv

3.2 Dish.csv:

3.3 MenuItem.csv:

3.4 MenuPage.csv

4. SQLite Relational Database & Integrity Constraints:

4.1 Database Model:

4.2 Tables Creation and Data Import:

4.3 Integrity Constraints check:

4.3.1 Key Data Duplication Check:

4.3.2 Foreign Key Constraint Check:

4.3.3 Not Null Check:

4.3.4 Other Checks:

5. YesWorkflow:

5.1 Key Inputs/Outputs:

5.2 Dependencies:

5.3 Diagram:

6 – Tableau

Appendix

Specifications

Project Deliverables:

1. Introduction

This is a project report on the end-to-end data cleaning done for New York Public Library’s crowd-sourced historical menus dataset. It summarizes the complete workflow along with our findings during the process.

1. Overview and initial assessment

The New York Public Library’s restaurant menu collection is one of the largest in the world. It holds about 45,000 menus dating from the 1840s to the present. There is a lot of data made available to the public through spreadsheet and APIs. For our current project, we have used the spreadsheet exports data available in csv format.

2.1 Structure & content of the dataset

The raw data consisted of four csv files:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Features | Menu | MenuItem | MenuPage | Dish |
| Row count | 17547 | 1332726 | 66937 | 423400 |
| Attribute count | 20 | 9 | 7 | 9 |
| Unique column | id | id | id | id |
| Associated file/field | MenuPage/menu\_id | MenuPage/id,  Dish/id | Menu/id,  Menuitem/menu\_page\_id | MenuItem/dish\_id |
| Description | Main file with record for each menu and it’s details like the event it was created for, status, dish count it has, date on which it was created etc. | Covers the details of each menu item, it’s price, creation date, page number it is on etc. | Covers information about each menu page and it’s link with a menu and menuitem. | Covers details of each dish, which menuitem it is listed on, high/low price, name etc. |

* 1. Quality issues

We analyzed each of the data files and identified below data quality issues:

* Blank values for certain fields.
* Some fields enclosed in double quotes.
* Inconsistent lowercase/uppercase format.
* Special Characters like #%?\()[] present;.
* Leading/trailing spaces present.
* Inconsistent data -abbreviations/full form present for same field.
* Inconsistent date format for date type field.
* Additional spaces present.
* Difference in spellings of same word.
* Key values mismatch between associated tables.
* Negative/Incorrect values assigned e.g. 0 for price
* Data integrity issues e.g. Created Date>Updated Date.
* Poor formatting
  1. Use cases of the dataset

The data has many data quality issues and needs to be cleansed enough to be made usable for any use case. Once the data is cleaned, it can be uploaded into the database and made available via web search to the users. Some of the use cases that can be applied on the cleansed data are:

* Answering specific questions for researchers such as when did a dish first appeared on the menu, how has the price/location of a dish changed over the years.
* The dishes are contained in a Menu along with their price and other details.
* The change in demand of a dish over a period of time and possible contributing factors.
* Correlation between the demand of a dish with its price/location/other factors.

Because of the multiple issues with the data, the dataset is not fit enough for being used in an app that is used for ordering food online.

* 1. Data cleaning goals

After analysis of the dataset, we decided to clean the data to a point that it can be loaded into the database and be available for web search. The goal of our leaning process is to have it clean enough to be able to support basic user queries like menu structure, popularity of a dish over time etc.

1. Data cleaning with OpenRefine & Python

The cleaning of each of the files is described in this section. We cleaned the files for Menu and Dish with Python script first and then with OpenRefine. MenuItem file was cleaned with OpenRefine. MenuPage did not have any data quality issues as such, hence it was not considered for cleaning with OpenRefine/Python. Below are the steps executed to clean each of the files.

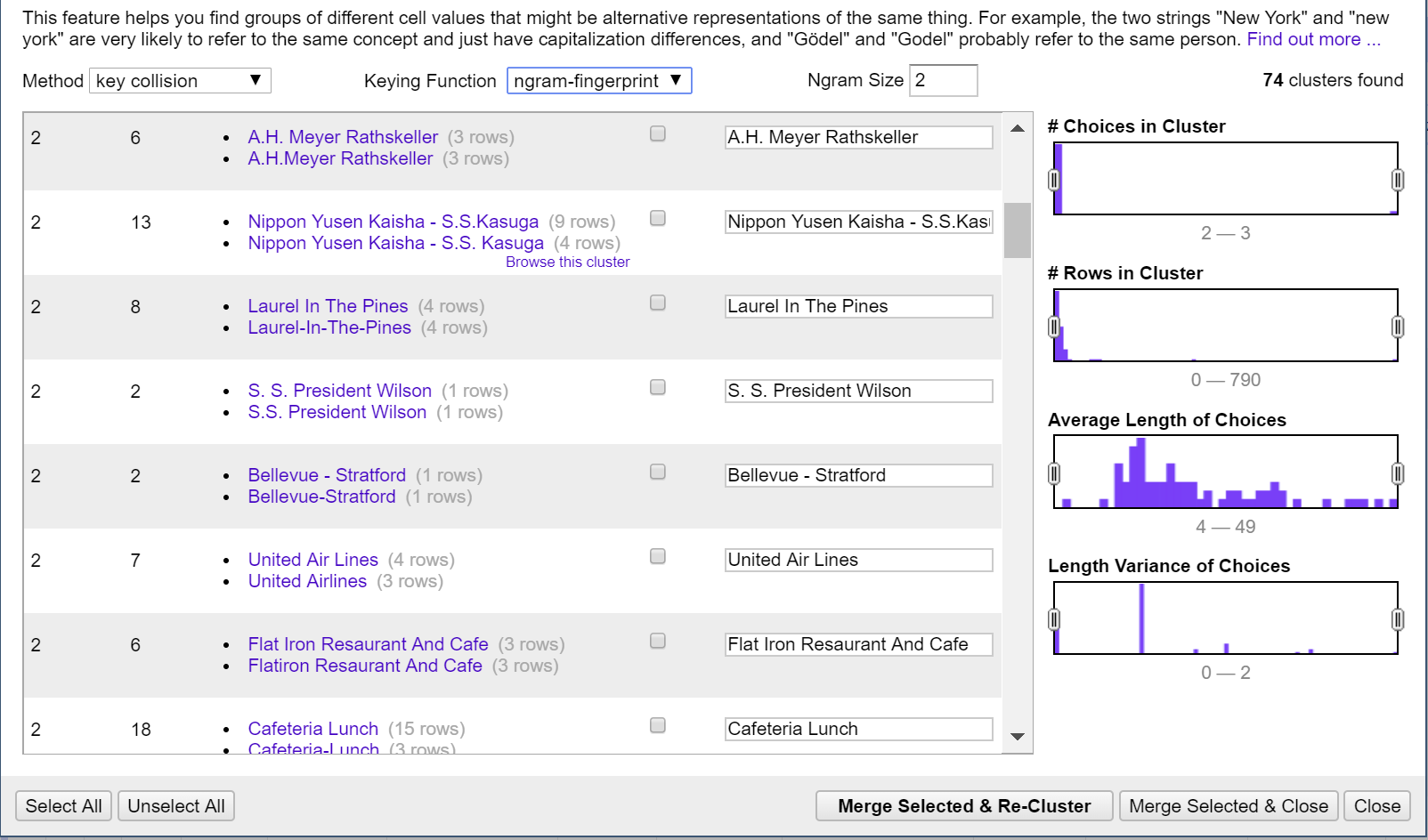
3.1 Menu.csv

Step 1: Converted Date Field to Date

Step 2: Analyzed Date field outliers and based on that removed outliers

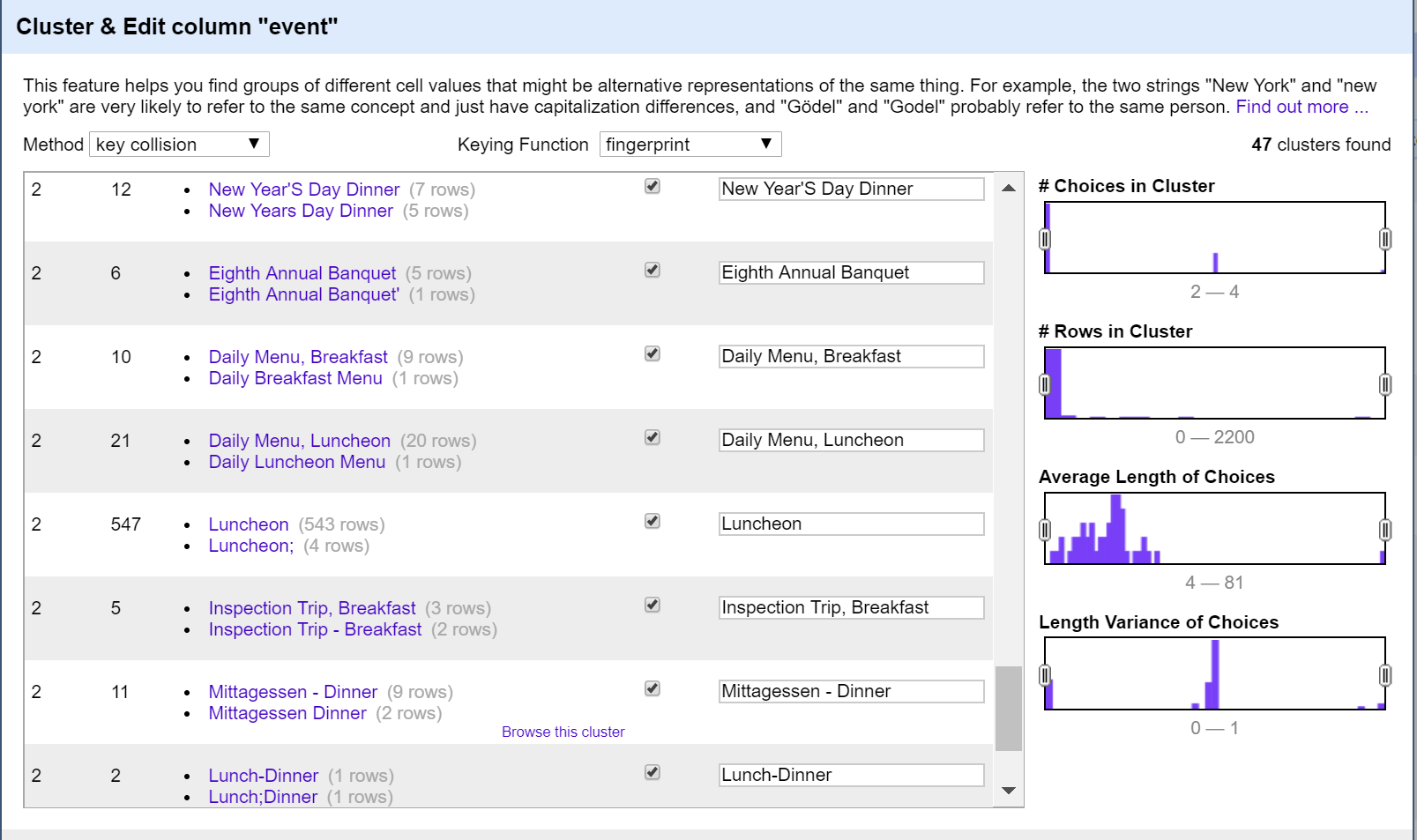
Step 3: Used Clustering from open refine on column “Sponsor”. Following methods were used:

1. Method: Key-Collision, Function: Fingerprint
2. Method: Key-Collision, Function: n-gram fingerprint (n=2)
3. Method: Key-Collision, Function: metaphone3
4. Method: Key-Collision, Function: cologne-phonetic
5. Method: Nearest Neighbor, Function: PPM (Radius: 1, Block Chars: 6)
6. Method: Nearest Neighbor, Function: Levenshtein (Radius: 1, Block Chars: 6)
7. Group updates

Screenshot:

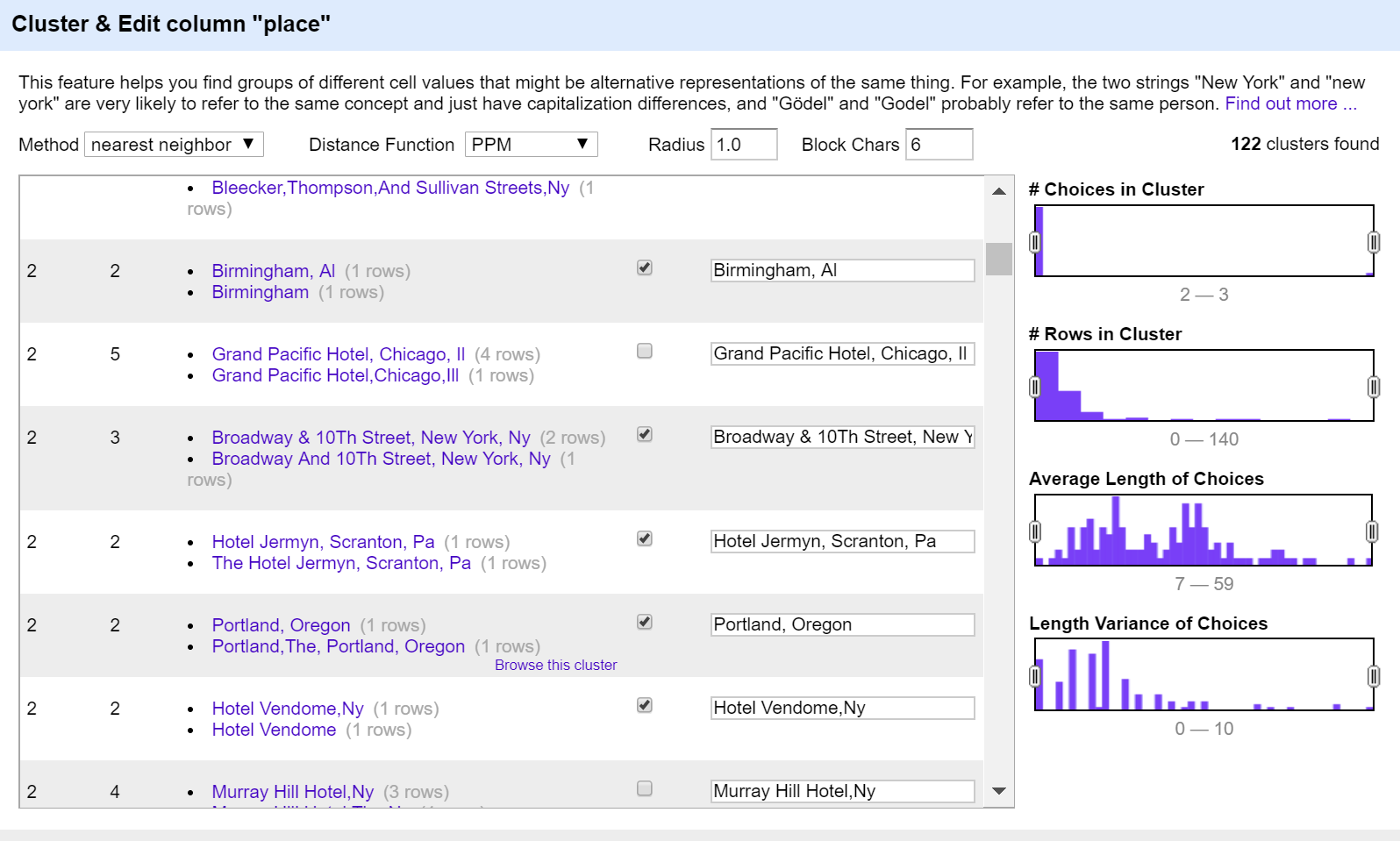
Step 4: Used Clustering from open refine on column “Event”. Following methods were used:

1. Method: Key-Collision, Function: Fingerprint
2. Method: Key-Collision, Function: n-gram fingerprint (n=2)
3. Method: Key-Collision, Function: metaphone3
4. Method: Key-Collision, Function: cologne-phonetic
5. Method: Nearest Neighbor, Function: PPM (Radius: 1, Block Chars: 6)
6. Method: Nearest Neighbor, Function: Levenshtein (Radius: 1, Block Chars: 6)
7. Group updates

Screenshot:

Step 5: Used Clustering from open refine on column “Place”. Following methods were used:

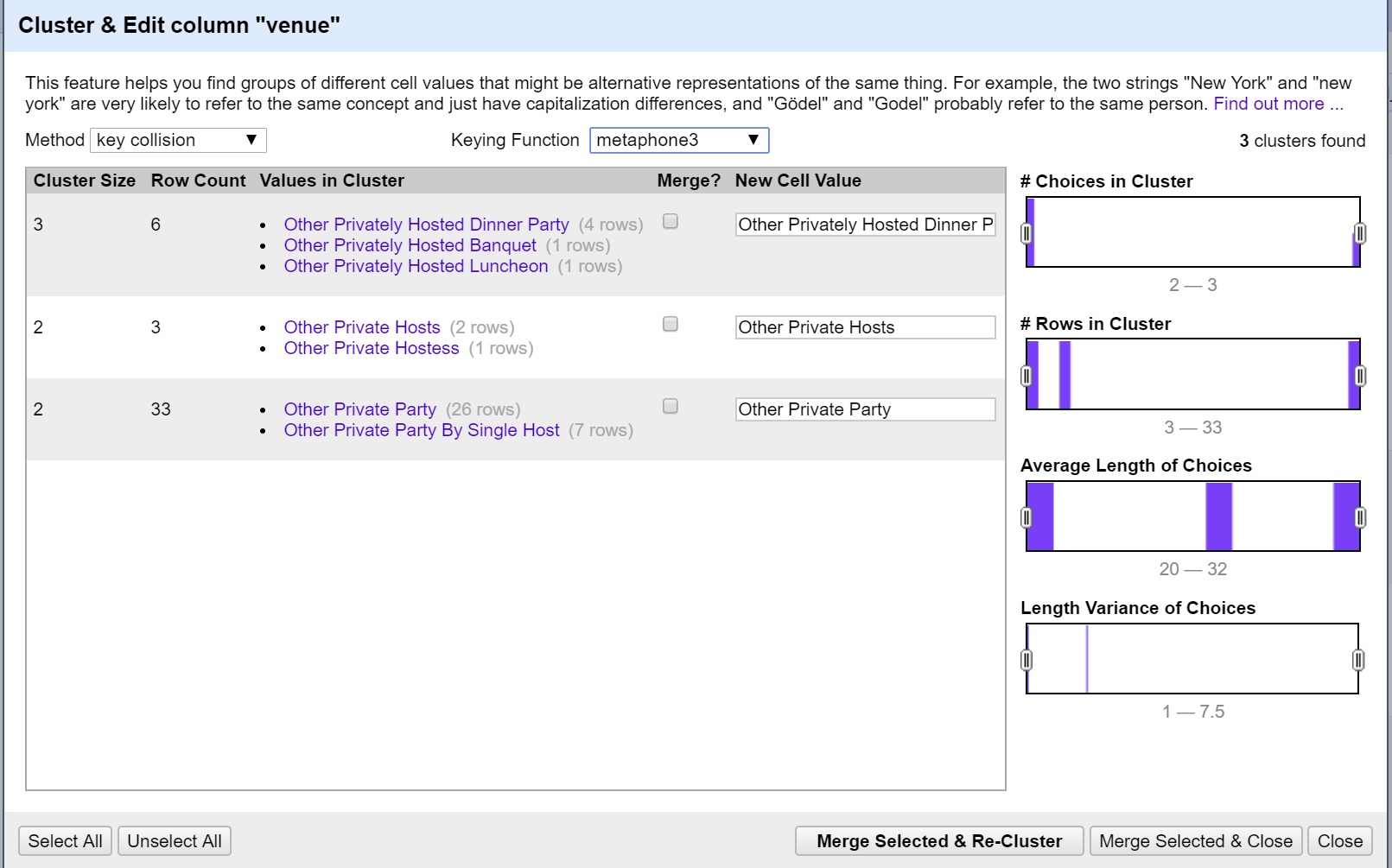
1. Method: Key-Collision, Function: Fingerprint
2. Method: Key-Collision, Function: n-gram fingerprint (n=2)
3. Method: Key-Collision, Function: metaphone3
4. Method: Key-Collision, Function: cologne-phonetic
5. Method: Nearest Neighbor, Function: PPM (Radius: 1, Block Chars: 6)
6. Method: Nearest Neighbor, Function: Levenshtein (Radius: 1, Block Chars: 6)
7. Group updates

Screenshot:

Step 6: Used Clustering from open refine on column “Venue”. Following methods were used:

1. Method: Key-Collision, Function: Fingerprint
2. Method: Key-Collision, Function: n-gram fingerprint (n=2)
3. Method: Key-Collision, Function: metaphone3
4. Method: Key-Collision, Function: cologne-phonetic
5. Method: Nearest Neighbor, Function: PPM (Radius: 1, Block Chars: 6)
6. Method: Nearest Neighbor, Function: Levenshtein (Radius: 1, Block Chars: 6)
7. Group updates

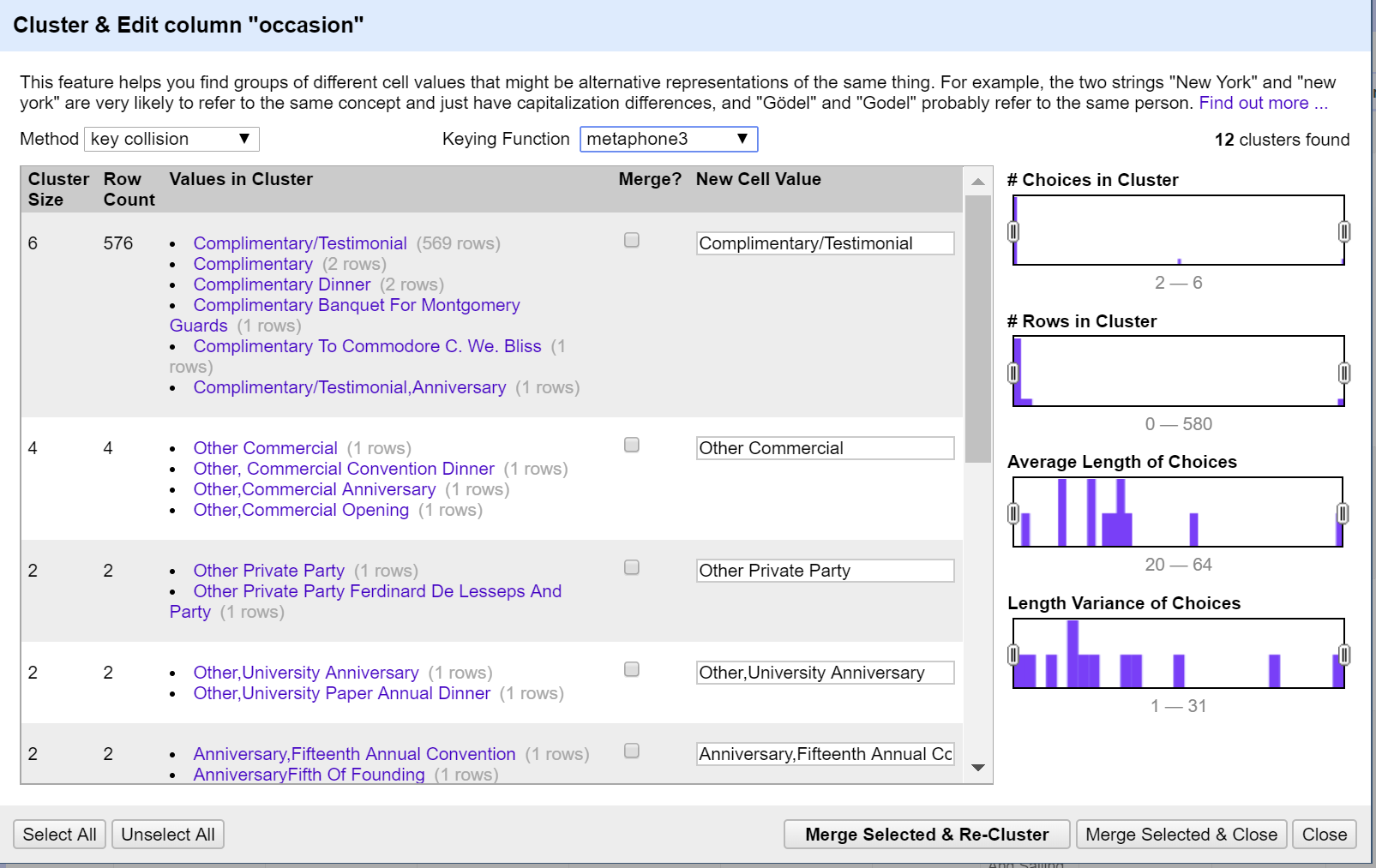
Screenshot:



Step 7: Used Clustering from open refine on column “Occasion”. Following methods were used:

1. Method: Key-Collision, Function: Fingerprint
2. Method: Key-Collision, Function: n-gram fingerprint (n=2)
3. Method: Key-Collision, Function: metaphone3
4. Method: Key-Collision, Function: cologne-phonetic
5. Method: Nearest Neighbor, Function: PPM (Radius: 1, Block Chars: 6)
6. Method: Nearest Neighbor, Function: Levenshtein (Radius: 1, Block Chars: 6)
7. Group updates

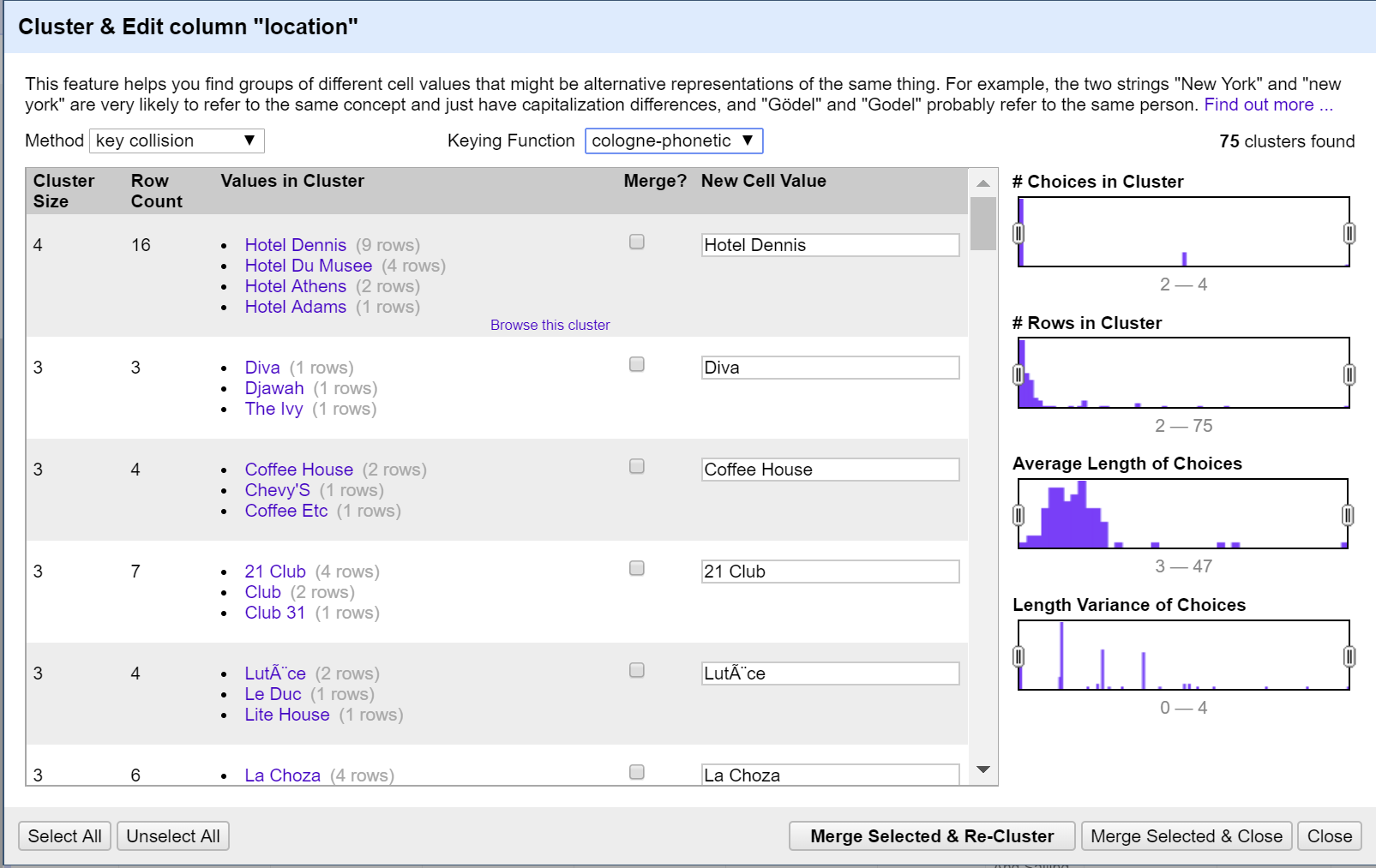
Screenshot:



Step 8: Used Clustering from open refine on column “Location”. Following methods were used:

1. Method: Key-Collision, Function: Fingerprint
2. Method: Key-Collision, Function: n-gram fingerprint (n=2)
3. Method: Key-Collision, Function: metaphone3
4. Method: Key-Collision, Function: cologne-phonetic
5. Method: Nearest Neighbor, Function: PPM (Radius: 1, Block Chars: 6)
6. Method: Nearest Neighbor, Function: Levenshtein (Radius: 1, Block Chars: 6)
7. Group updates

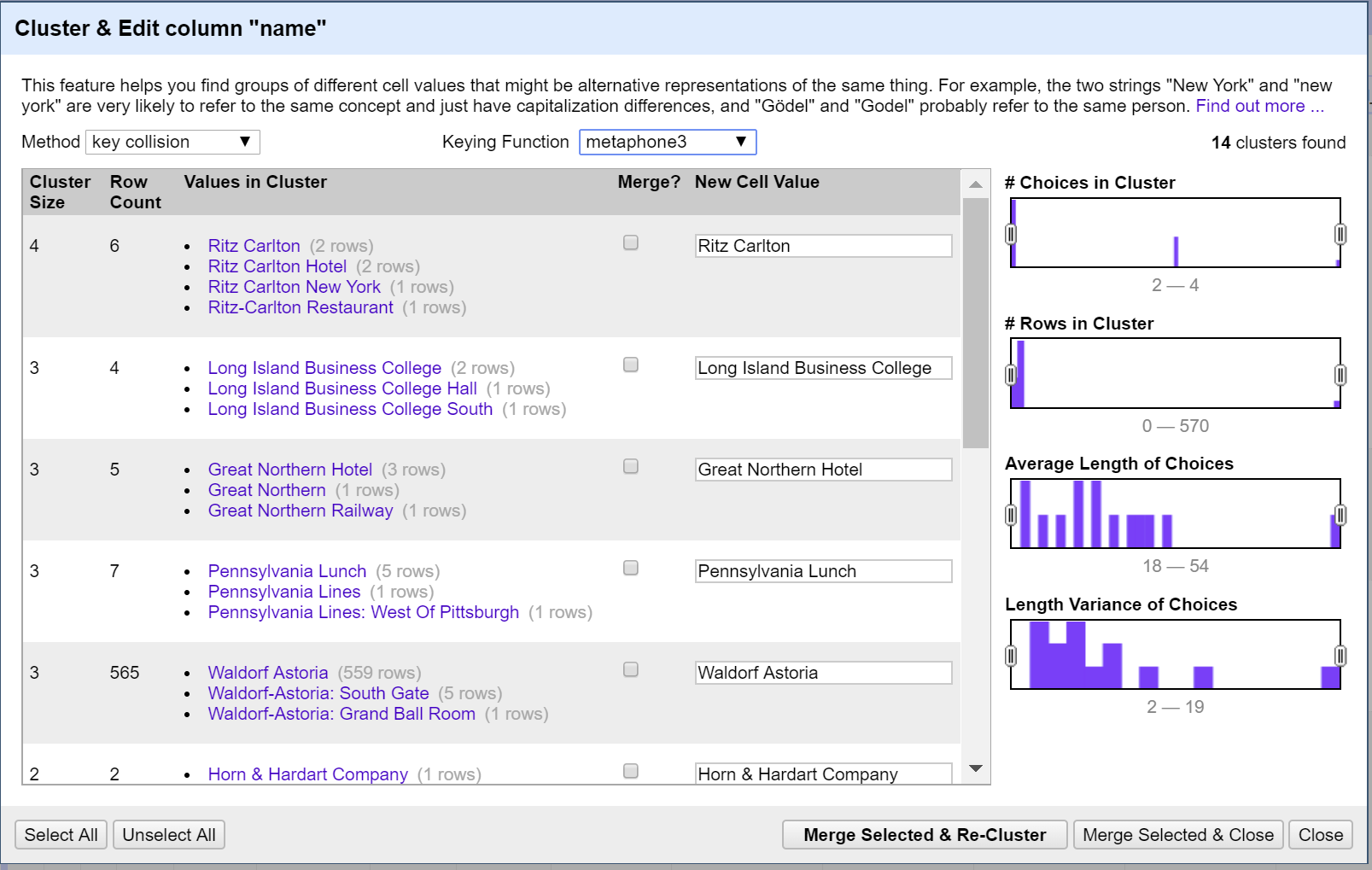
Screenshot:



Step 9: Used Clustering from open refine on column “Name”. Following methods were used:

1. Method: Key-Collision, Function: Fingerprint
2. Method: Key-Collision, Function: n-gram fingerprint (n=2)
3. Method: Key-Collision, Function: metaphone3
4. Method: Key-Collision, Function: cologne-phonetic
5. Method: Nearest Neighbor, Function: PPM (Radius: 1, Block Chars: 6)
6. Method: Nearest Neighbor, Function: Levenshtein (Radius: 1, Block Chars: 6)
7. Group updates

Screenshot:



Step 10: Split the physical\_description column. It resulted in 7 new columns:

physical\_description 1

physical\_description 2

physical\_description 3

physical\_description 4

physical\_description 5

physical\_description 6

physical\_description 7

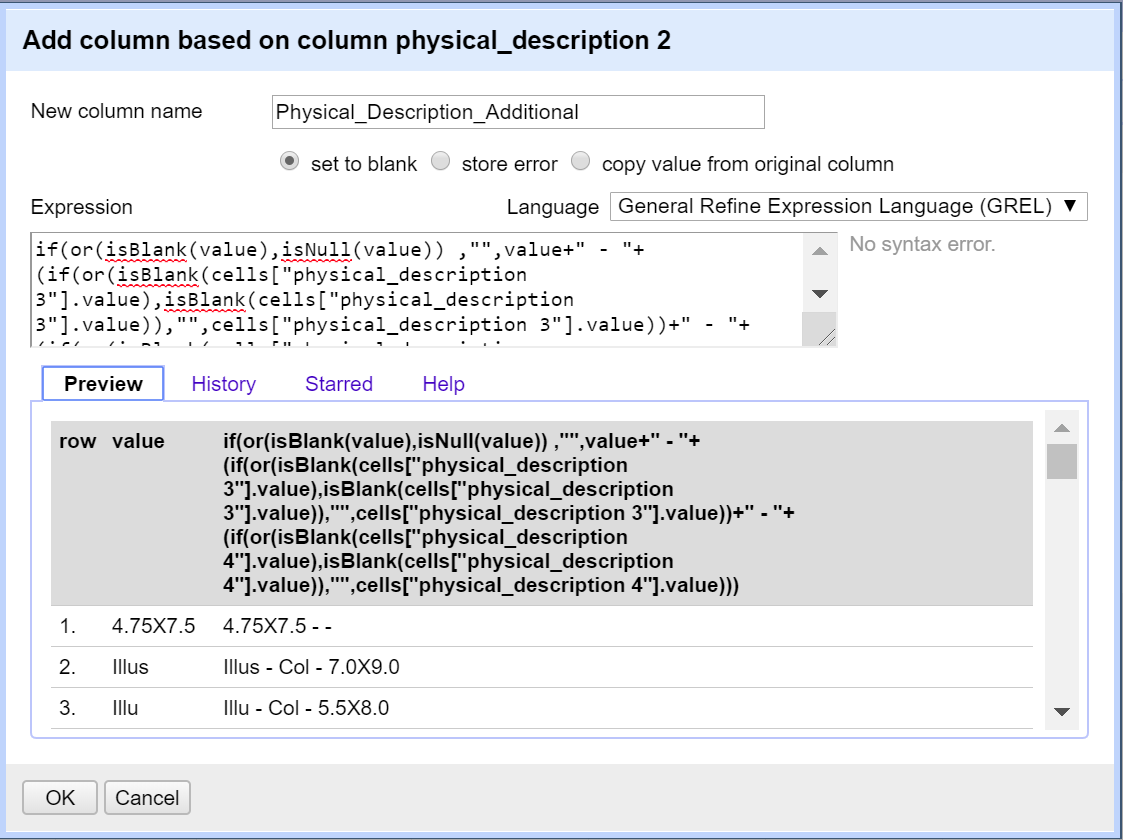
We updated the column physical\_description 1 to physical description type and then created a new column Physical\_description\_additional by merging following columns:

physical\_description 2

physical\_description 3

physical\_description 4

Screenshot:



3.2 Dish.csv

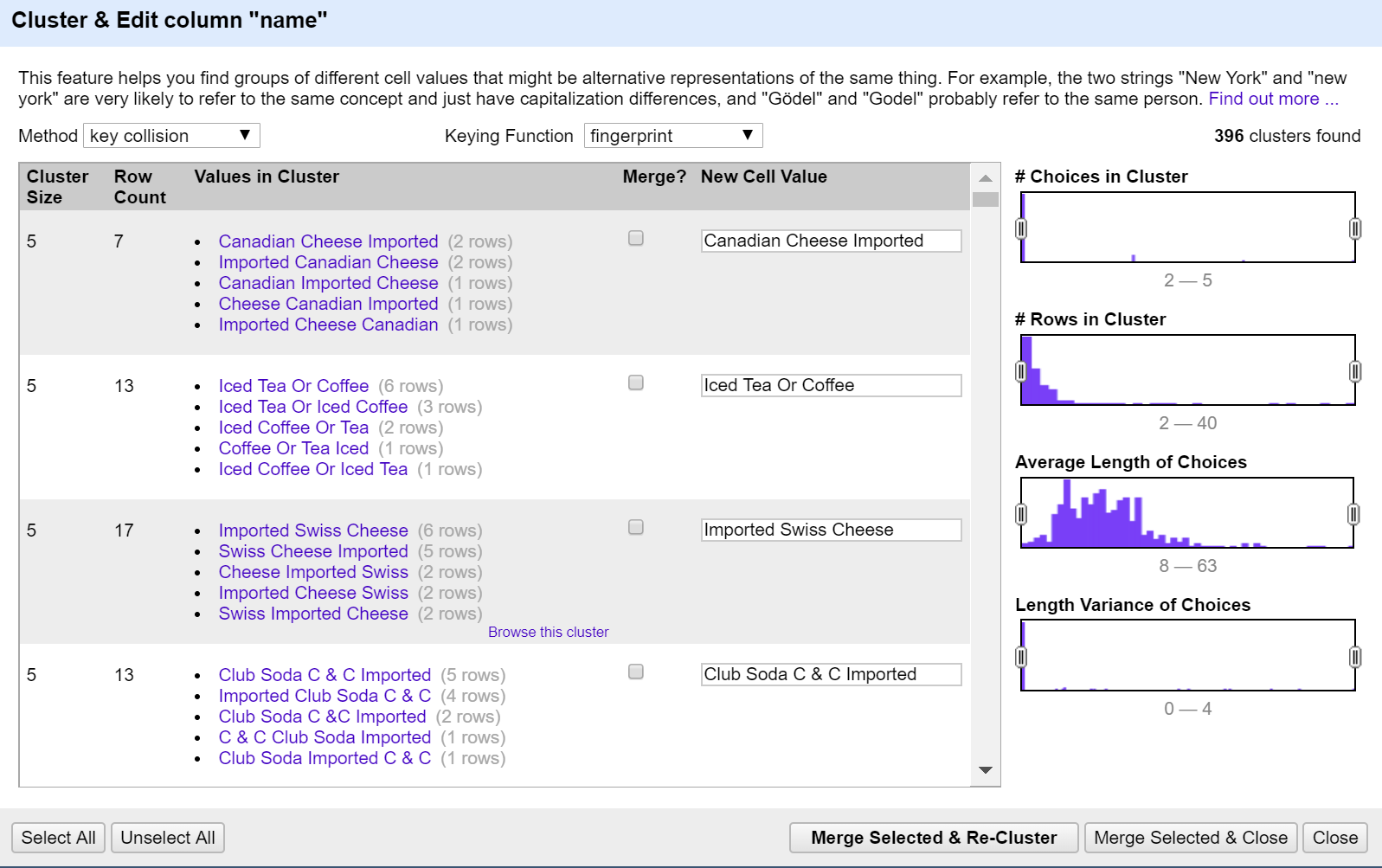
Step 1: Transform data to remove unwanted characters. Formula:

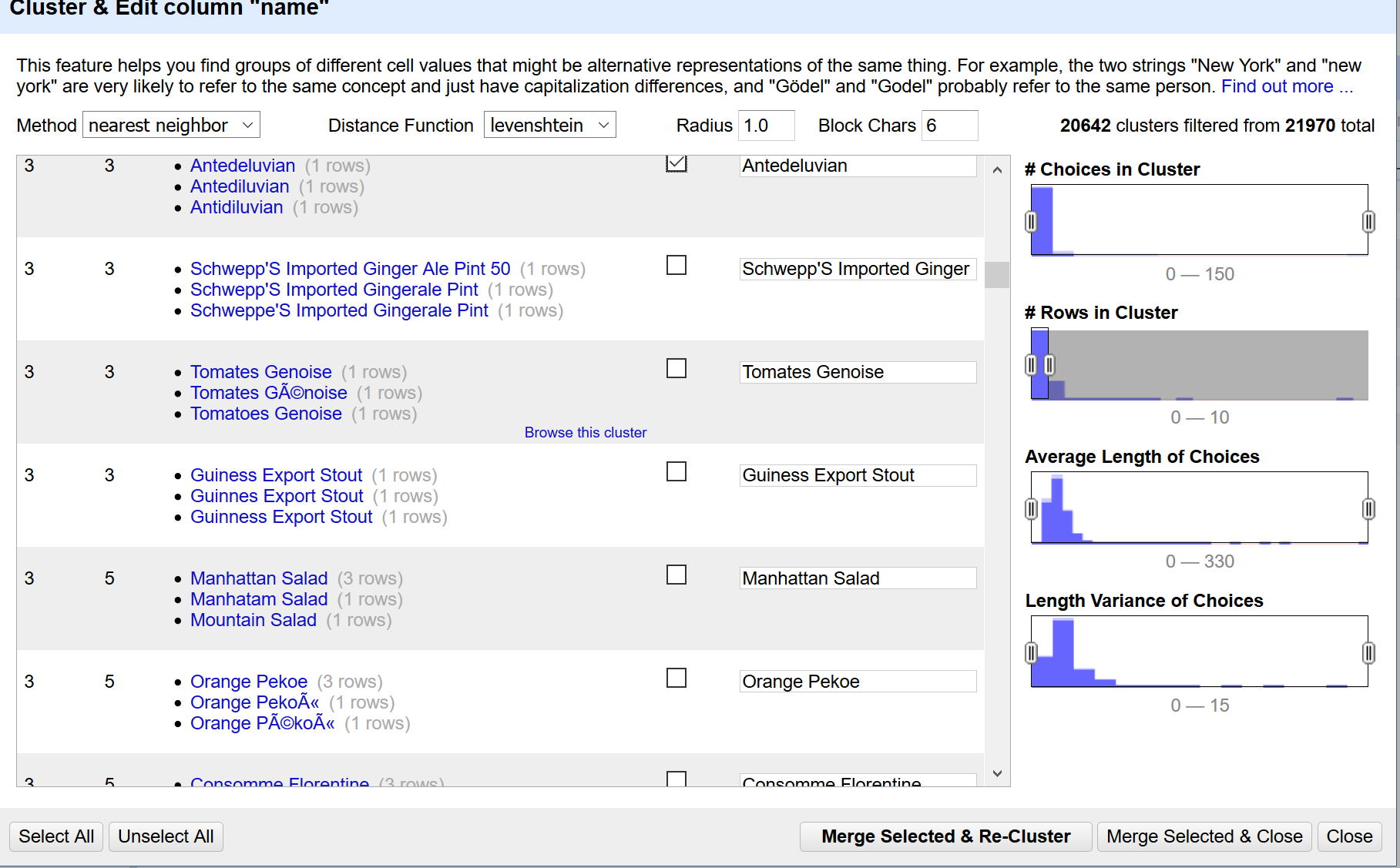
value.replace(/[>:<%#@!\\()\[\]\?\"\-\\*,\.\+]/, " ").replace(/\s+/," ").trim()

Step 2: Used Clustering from open refine on column “Name”. Following methods were used:

1. Method: Key-Collision, Function: Fingerprint
2. Method: Key-Collision, Function: n-gram fingerprint (n=2)
3. Method: Key-Collision, Function: metaphone3
4. Method: Key-Collision, Function: cologne-phonetic
5. Method: Nearest Neighbor, Function: PPM (Radius: 1, Block Chars: 6)
6. Method: Nearest Neighbor, Function: Levenshtein (Radius: 1, Block Chars: 6)
7. Group updates

Screenshot:





3.3 MenuItem.csv

Step 12: Transform created\_at field to date by creating a new field created\_date

Step 13: Transformed updated\_at field to date by creating a new field updated\_date.

(Note: both created\_at and created\_date field exists in the csv file. Similarly, both updated \_at and updated \_date field exists in the database.)

Step 14: Transform xpos field to number

Step 15: Transformed ypos field to number

3.4 MenuPage.csv

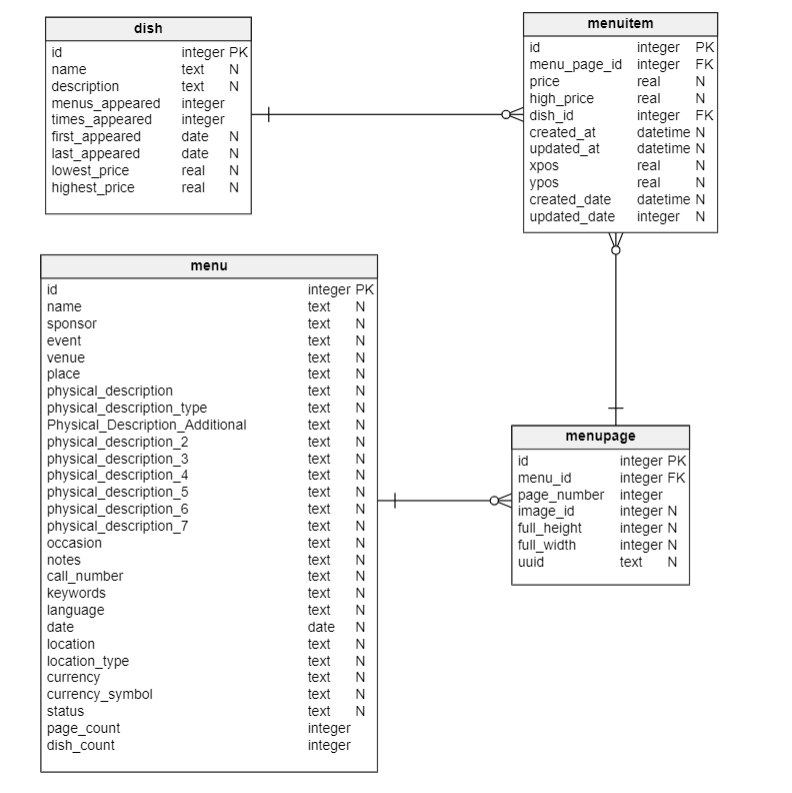
No transformations were performed for this csv in open refine.

Challenges faced during cleaning:

4. SQLite Relational Database & Integrity Constraints

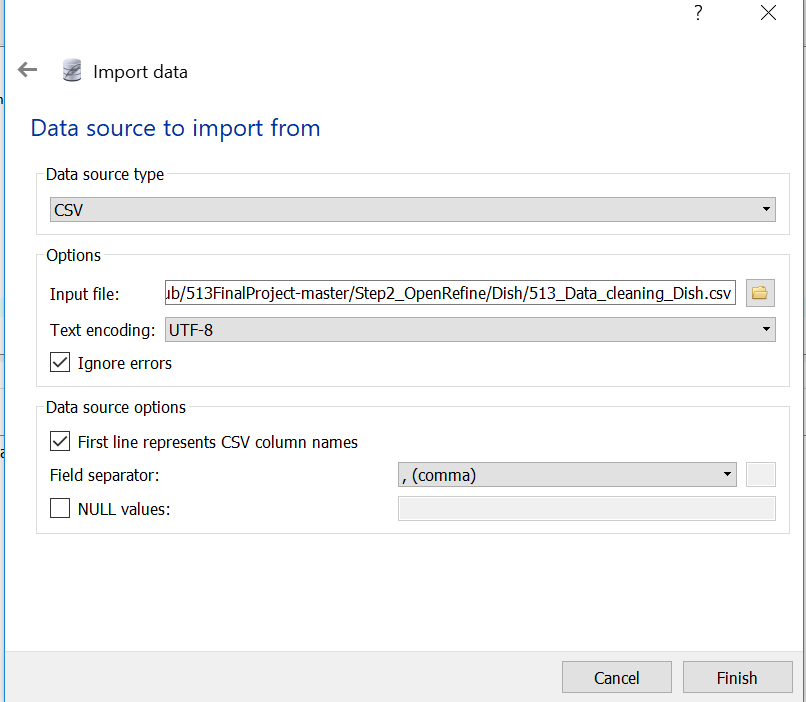
4.1 Database Model

Based on the data, we created below database model which includes the logical integrity constraints of ‘Primary Key’, ‘Foreign Key’ and ‘Not Null’. The database model also defines the one-to-one and one-to-many relationships between the four tables. Building the schema based on this model ensures that only those records that satisfy the defined constraints are loaded into the tables and the dirty data is discarded.



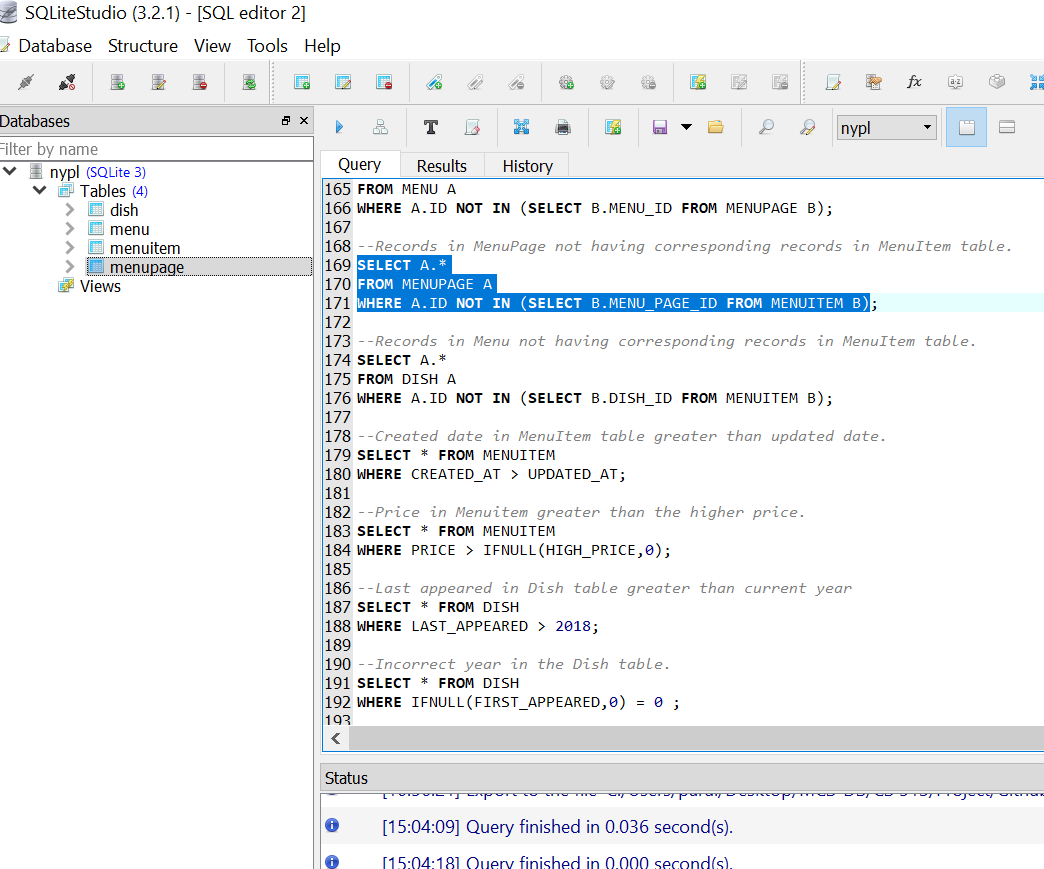
4.2 Tables Creation and Data Import

A database schema ‘nypl’ was created using SQLiteStudio. We created 4 tables and imported data into the tables from the cleaned csv files in nypl database.



4.3 Integrity Constraints check

We created several queries to check the logical integrity constraints on the tables. All the scripts for the schema/table creation and integrity constraints check are listed under the ‘SQL\_scripts’.sql file. SQLiteStudio was used to run the queries and obtain results.



Below is a brief description of the various checks done on the data.

* + 1. Key Data Duplication Check

This step checked presence of any duplicate data in the table on the basis of ID field. Since, we applied primary key constraint on ID field during the data load, the duplicate data was filtered out already from the tables. Our final data in the tables returned no duplicate records.

* + 1. Foreign Key Constraint Check

This step checked the foreign key constraints violation on the tables. That is if any table that used primary key of another table, had data that not matched with the primary key value in the source table. We applied this constraint during the data load itself for Menu, Dish and MenuItem tables. We had to disable the foreign key constraints only on the MenuItem table during the data load as it would have otherwise resulted in loss of too many rows of data and we wouldn’t get enough data for further analysis.

* + 1. Not Null Check

We applied not null constraint on certain table columns to ensure that only those records were loaded into the table that had values in these columns. This resulted in discarding irrelevant records which would not be useful for further analysis. This constraint was applied on all the tables during the data load itself.

* + 1. Other Checks

We ran queries to identify other integrity constraint violations, some of which are listed below:

* Created date in MenuItem table greater than updated date.
* Price in Menuitem greater than the higher price.
* Last appeared in Dish table greater than current year (2018)
* Incorrect year in the Dish table.
* First\_appeared greater than last\_appeared in Dish table.
* Lowest price greater than highest price in Dish table.
* Count of menus\_appeared in dish table not matching count obtained from menuitem and menupage tables.
* Mismatch of page count between menu and menupage tables.

1. YesWorkflow

We created a YesWorkflow for the whole process using the online YesWorkflow tool (link in appendix section).

5.1 Key Inputs/Outputs

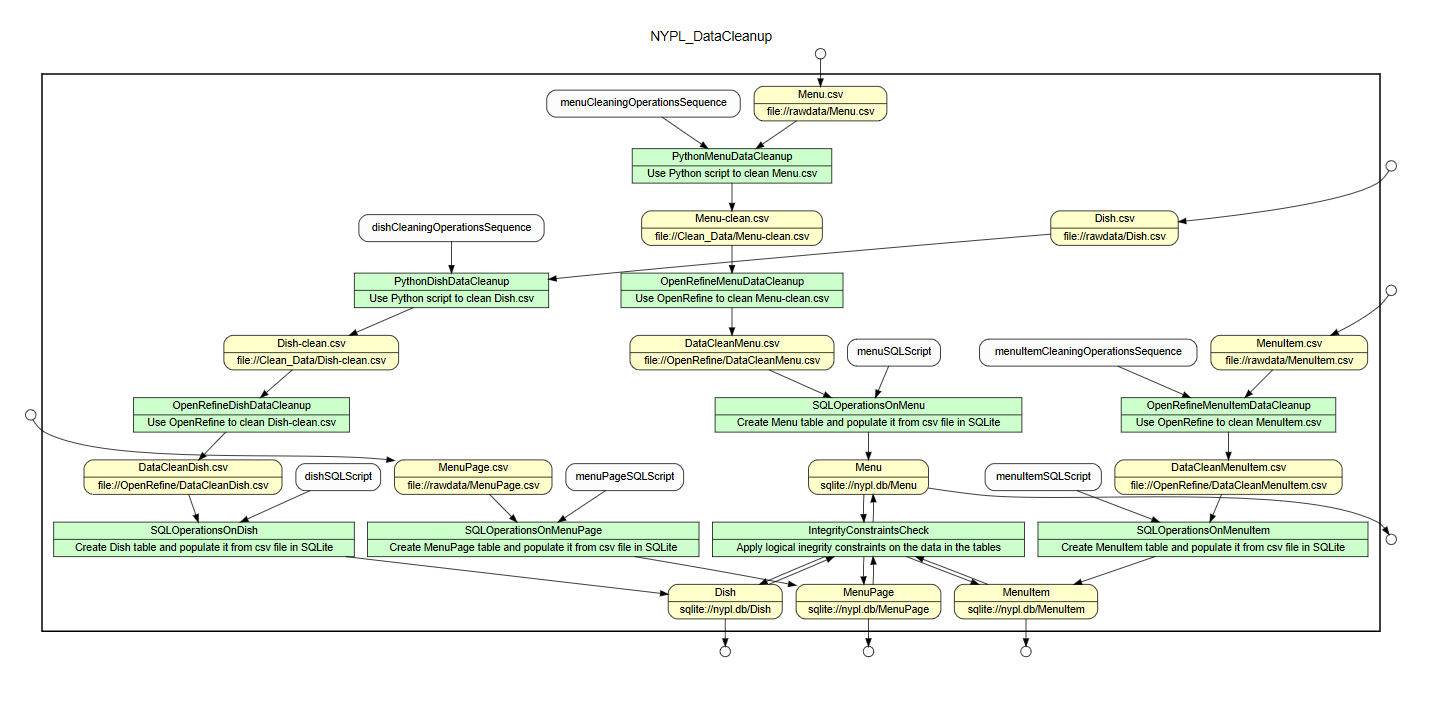
The key inputs to the workflow process are the four input data files namely, menu.csv, dish.csv, menuitem.csv, menupage.csv along with the datacleaning operation comprising of Openrefine and Python script.

* 1. Dependencies

The complete workflow includes a number of steps that are interdependent on each other. The data cleaning Operation done with OpenRefine and Python script is dependent on the raw data files. The SQLOperations of schema creation and integrity constraints is dependent on the final cleansed files.

5.3 Diagram

Below is a snapshot of the Workflow Diagram:



1. Tableau

7 Appendix

Software Specifications

|  |  |
| --- | --- |
| **Software** | **Version/Url** |
| Python |  |
| OpenRefine |  |
| SQLiteStudio | 3.2.1 |
| YesWorkflow | http://try.yesworkflow.org/ |
| Tableau |  |
| Vetabelo (Database Modeler) | https://my.vertabelo.com/ |

Project Deliverables

All the projects deliverables including the raw dataset, sql scripts, cleansed table data, Yesworkflow scripts can be accessed at this [link](https://github.com/rohitbansal83/513FinalProject).