|  |  |
| --- | --- |
| Image result for Data Cleaning  CS513: Data Cleaning Project Report | Abstract  Project report on data cleaning performed on New York Public Library’s crowd-sourced historical menus dataset to make the dataset usable for data analytics purpose  Team  Rohit Bansal  Parul Mainwal |

Table of Contents

[Introduction 2](#_Toc531959100)

[Initial assessment 2](#_Toc531959101)

[Structure Summary: 4](#_Toc531959102)

[Quality issues 4](#_Toc531959103)

[Use cases of the dataset 5](#_Toc531959104)

[Data cleaning goals 5](#_Toc531959105)

[Data cleaning process 6](#_Toc531959106)

[Step1: Python Script 6](#_Toc531959107)

[Step2: OpenRefine 7](#_Toc531959108)

[File: Menu.csv 7](#_Toc531959109)

[File: Dish.csv 15](#_Toc531959110)

[File: MenuItem.csv 17](#_Toc531959111)

[File: MenuPage.csv 17](#_Toc531959112)

[Step 3: Integrity Checks using SQLite database constraints 17](#_Toc531959113)

[Database Model 17](#_Toc531959114)

[Tables Creation and Data Import 18](#_Toc531959115)

[Integrity Constraints check 19](#_Toc531959116)

[Key Data Duplication Check 20](#_Toc531959117)

[Foreign Key Constraint Check 20](#_Toc531959118)

[Not Null Check 21](#_Toc531959119)

[Other Checks 21](#_Toc531959120)

[YesWorkflow 21](#_Toc531959121)

[Key Inputs/Outputs 21](#_Toc531959122)

[Dependencies 21](#_Toc531959123)

[Diagram 21](#_Toc531959124)

[Challenges 22](#_Toc531959125)

[Ambiguity 22](#_Toc531959126)

[OpenRefine Efficiency 22](#_Toc531959127)

[Tableau 23](#_Toc531959128)

[Appendix 24](#_Toc531959129)

[Tools/Software Used 24](#_Toc531959130)

[Project Deliverables 24](#_Toc531959131)

# Introduction

This document is a project report for the end-to-end data cleaning process performed on New York Public Library’s crowd-sourced historical menus dataset. It captures purpose of the data cleaning, details of the step performed for data cleaning and the findings during the process. This document also highlights the provenance of the steps performed using a tool called “Yes Workflow”.

The New York Public Library’s restaurant menu collection holds data about menus and dishes from 1840 to present. This is a crowdsourced dataset collected through the spreadsheets and APIs. Since the data crowdsourced and is collected via various means the data quality is very poor. The objective of this project is to apply various data cleaning technique and analyze their effectiveness on such a large data set. After performing data cleaning, it is expected that data will be usable for various data analytics purpose effectively.

# 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. It has almost half a million dish items listed in the database along with almost 20 thousand menus of various eateries. We downloaded the data in comma separated values format. As a result, we got 4 files:

* Dish.csv

This file has name of the dish along with lowest price and highest price. It also has some popularity related data where it informs when the dish was first appeared in a menu and when it was last appeared and in how many menus it was appeared. Here is the list of columns:

* + id
  + name
  + description
  + menus\_appeared
  + times\_appeared
  + first\_appeared
  + last\_appeared
  + lowest\_price
  + highest\_price
* Menu.csv

This file has details of menus. The details include the name, place, occasion, event, location, venue, sponsor of the menu. It also has some other information such as number of pages in the menu, number of dishes in the menu, language of the menu, currency of payment and some other descriptive fields. Here is the list of columns:

* + Id
  + Name
  + Sponsor
  + Event
  + Venue
  + Place
  + physical\_description
  + occasion
  + notes
  + call\_number
  + keywords
  + language
  + date
  + location
  + location\_type
  + currency
  + currency\_symbol
  + status
  + page\_count
  + dish\_count
* MenuPage.csv

This file refers to menu file and provides details of menu page. It provides the image of the menu page, height and width of the page and page number. It also provides the menu id from the menu file so that page could be associated to specific menu. Here is the list of columns:

* + id
  + menu\_id
  + page\_number
  + image\_id
  + full\_height
  + full\_width
  + uuid
* MenuItem.csv

This file refers to the menu, dish and menu page files and provides the information related to each menu item. For each menu item, it provides the dish id from dish.csv and menu page id from menupage.csv. It also provides other details of menu items such as price, creation type, and the location coordinate of the place. Here is the list of columns:

* + Id
  + menu\_page\_id
  + price
  + high\_price
  + dish\_id
  + created\_at
  + updated\_at
  + xpos
  + ypos

Structure Summary:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Features** | **Menu** | **Dish** | **MenuPage** | **MenuItem** |
| Row count | 17547 | 423400 | 66937 | 1332726 |
| Attribute count | 20 | 9 | 7 | 9 |
| Unique column | id | id | id | id |
| Refers |  |  | Menu\_id | Menu\_Page\_id  Dish\_Id |
| Description | File with record for each menu and its details | File with records of dishes | File with menu page information along with reference to menu | File with menu item details along with reference to dish and menu page |

Quality issues

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

* Leading/trailing spaces
* Additional spaces
* Inconsistent case
* Blank values for certain fields
* Inconsistent representation of missing information
* Special Characters like #%?\()[]
* Spelling mistakes
* In consistent format for values
* Some fields enclosed in double quotes.
* Inconsistent data -abbreviations/full form present for same field.
* Inconsistent date format for date type field.
* 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
* Ambiguity

Use cases of the dataset

As highlighted above, the data had many data quality issues. It had to be cleaned in order to make it usable for any use case. Cleaned data could be used for various data analytics purpose. 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 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 was not fit enough for being used in any application.

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 some basic data analysis could be performed. The goal of our cleaning 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.

We did not intend to remove the ambiguity in the data. We did not want to remove ambiguity by making assumptions. For example, if we found an item as omelette 1 and omelette 2 then we did not try to merge it to “omelette”. The reason behind that was that these two could be a variant of omelette with different ingredient and sometime different size of servings. Assuming them to represent one dish could hamper the data integrity. So, we left those cases as it is. We only cleaned data when it has following issues:

* Leading/trailing spaces
* Additional spaces
* Inconsistent case
* Blank values for certain fields
* Inconsistent representation of missing information
* Special Characters like #%?\()[]
* Spelling mistakes
* In consistent format for values
* Some fields enclosed in double quotes.
* Inconsistent data -abbreviations/full form present for same field.
* Inconsistent date format for date type field.
* 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

# Data cleaning process

## Step1: Python Script

We developed a python script in order to get rid of few of the common issues with various fields. These issues included:

* Leading/trailing spaces
* Additional spaces
* Inconsistent case
* Blank values for certain fields
* Inconsistent representation of missing information
* Special Characters like #%?\()[]

The reason for going for python script instead of Open Refine was that these issues were present with multiple columns in different files. With parallel processing architecture using map-reduce, these cleanups could be done very efficiently in one pass via python script. Using the script, we cleaned following items in one pass:

* File: Menu.csv (12 columns)
  + Name
  + Sponsor
  + Event
  + Venue
  + Place
  + Physical\_Decription
  + Occasion
  + Notes
  + Date
  + Location
  + Status
  + Currency\_Symbol
* File: Dish.csv (1 column)
  + Name

It is noticeable that menu.csv contains 17547 rows while dish.csv contains 423400 rows. Thus, with the script (12\*17547) + (1\*423400) = (210564 + 423400) = 633964 items were cleaned in one step within few seconds. This turned out to be most efficient then any other method we considered including open refine transformations.

## Step2: OpenRefine

We used open refine to further clean the data. Menu, MenuItem and Dish file was cleaned using OpenRefine. MenuPage did not had any issues which could be removed by clustering of facet. Hence it was not considered for cleaning with OpenRefine. We used clustering and facet feature of open refine extensively. We considered following methods for clustering:

* Method: Key-Collision, Function: Fingerprint
* Method: Key-Collision, Function: n-gram fingerprint (n=2)
* Method: Key-Collision, Function: metaphone3
* Method: Key-Collision, Function: cologne-phonetic
* Method: Nearest Neighbor, Function: PPM (Radius: 1, Block Chars: 6)
* Method: Nearest Neighbor, Function: Levenshtein (Radius: 1, Block Chars: 6)

Not all the method returned good candidates and sometimes method did not return any candidate. We picked and choose only good cluster to perform data cleaning while ignore the other clusters. Here is the list of detailed steps performed on each of the file:

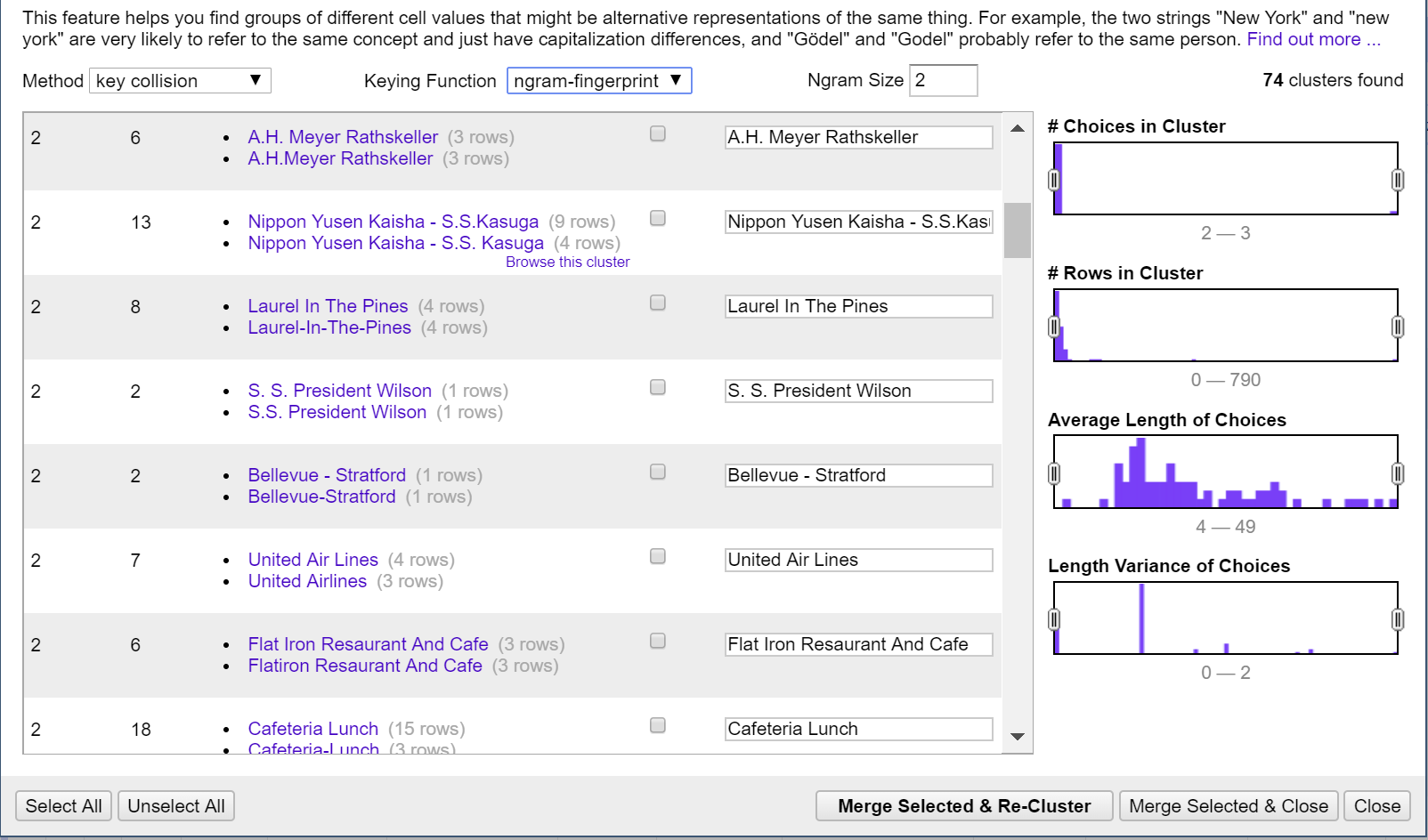
File: Menu.csv

Step 1: Converted Date Field to Date

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

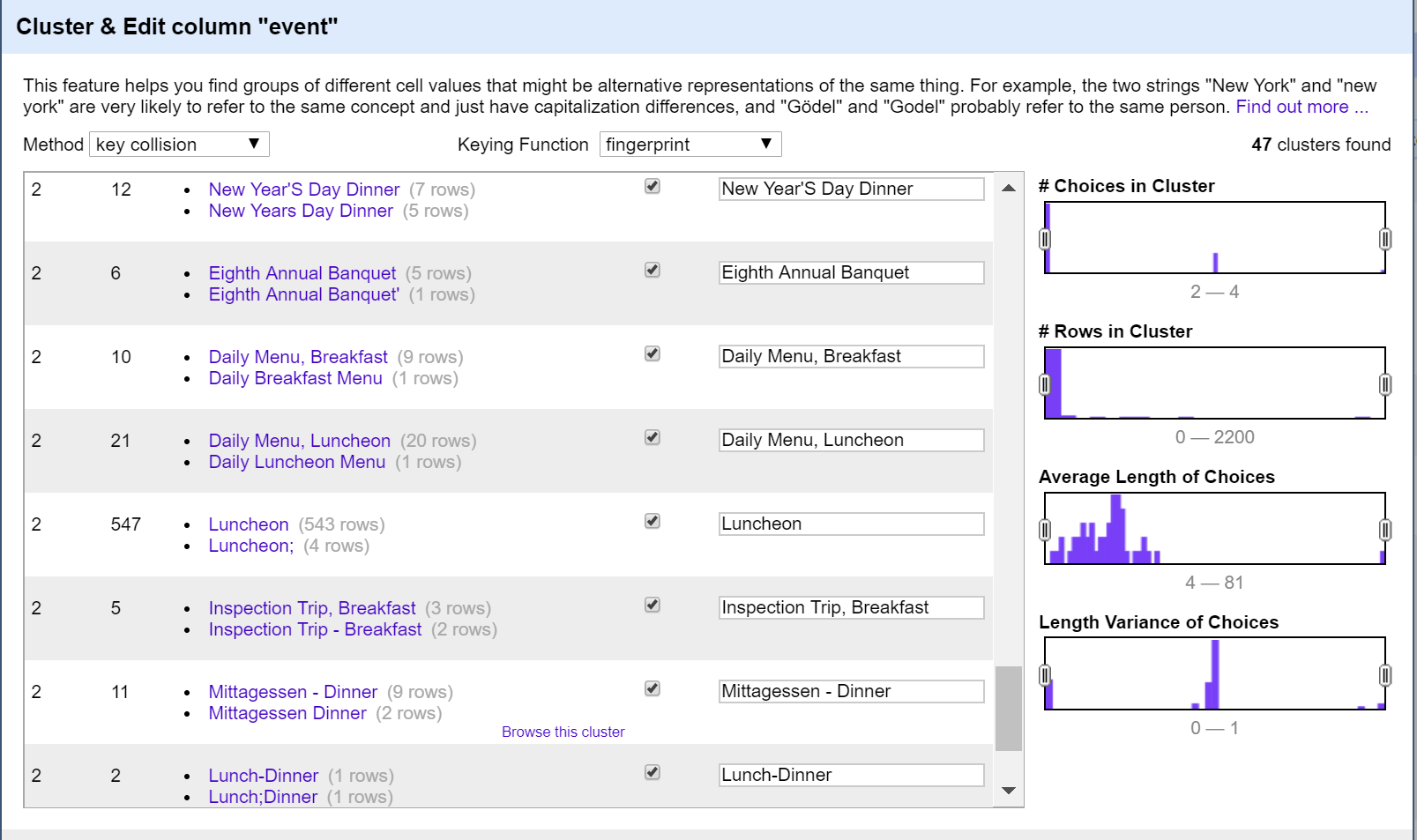
Step 3: Used Clustering from open refine on column “Sponsor”. Following methods were considered:

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:

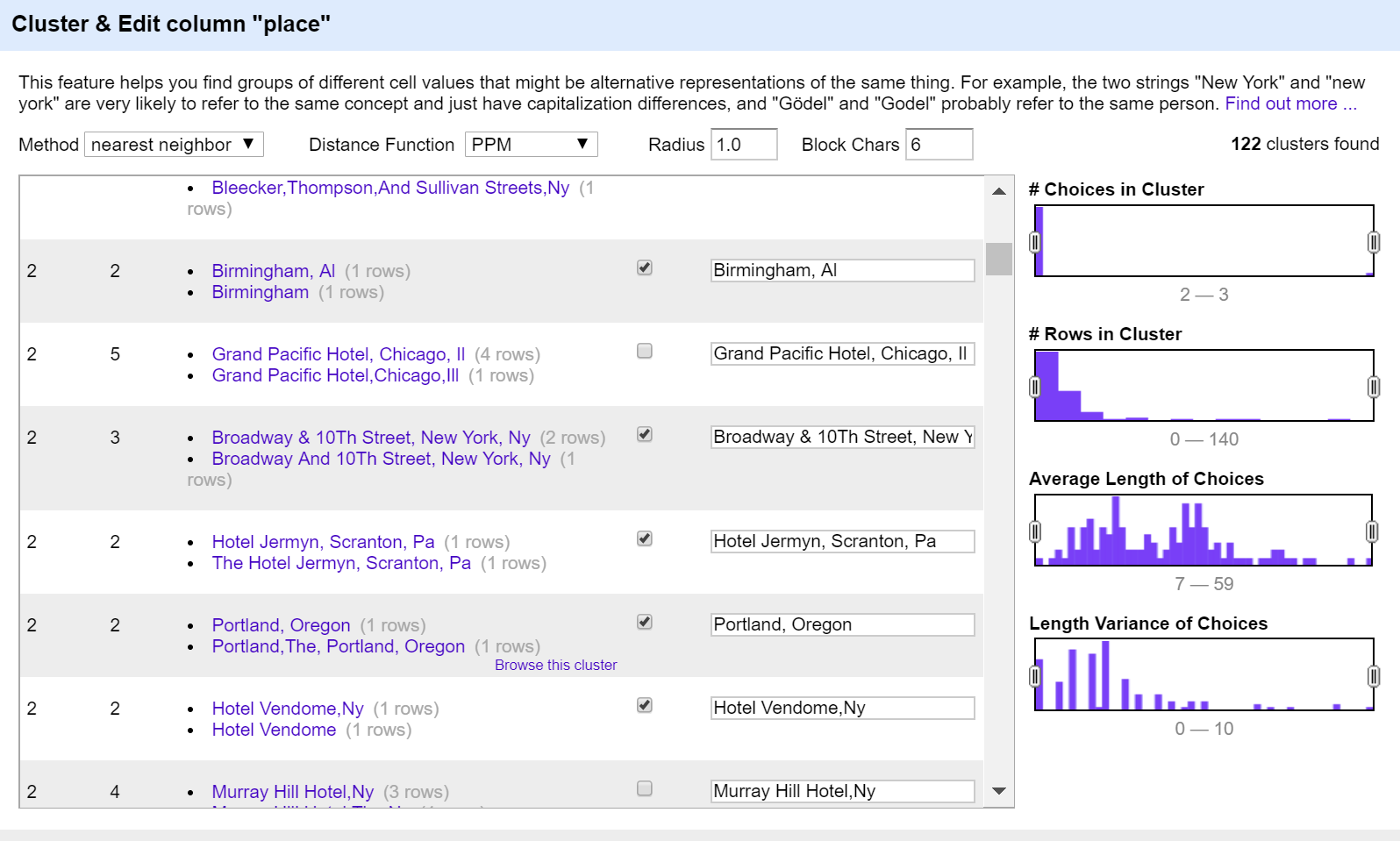
Step 4: Used Clustering from open refine on column “Event”. Following methods were used:

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:

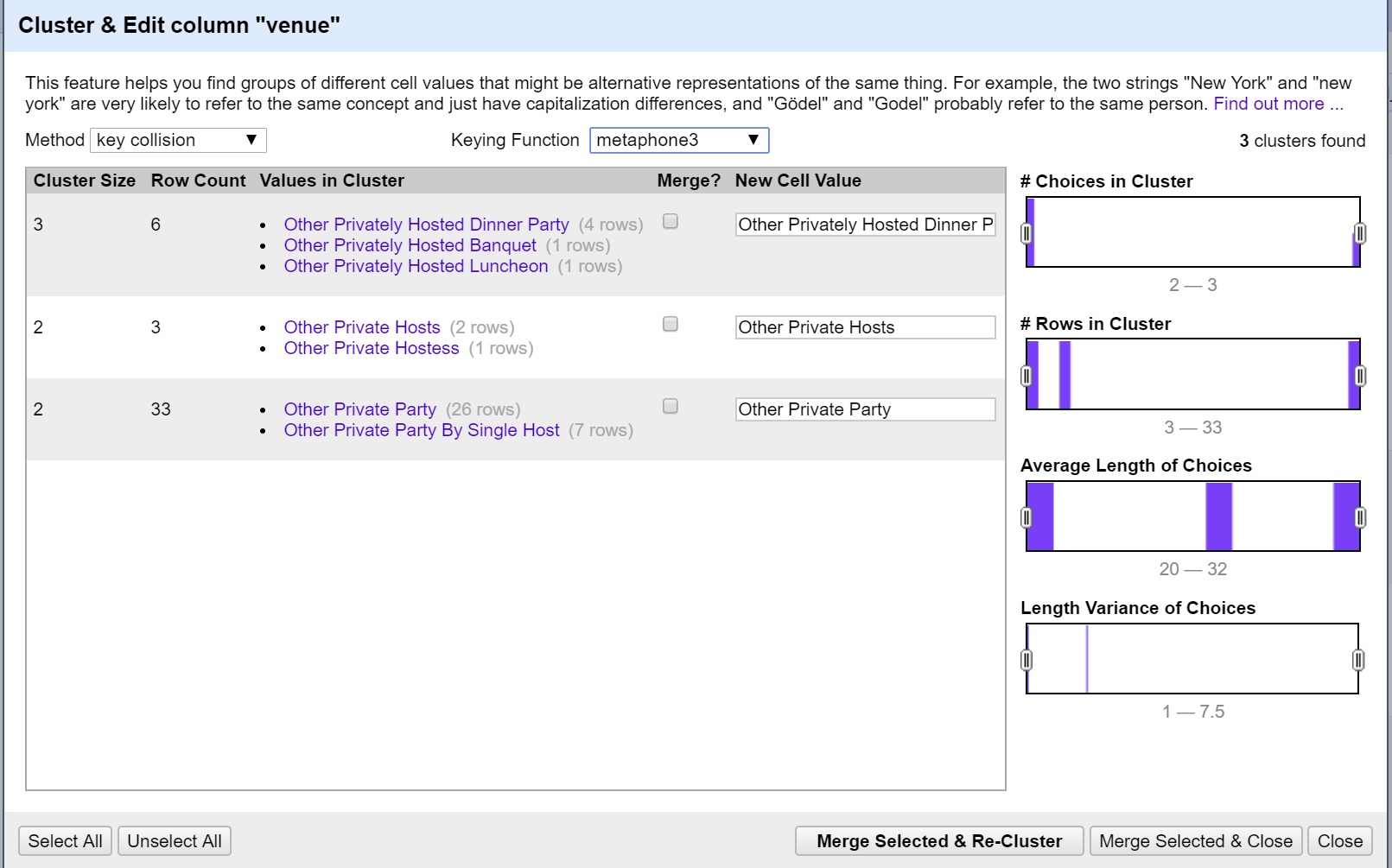
1. Method: Key-Collision, Function: Fingerprint
2. Method: Key-Collision, Function: n-gram fingerprint (n=2)
3. Method: Key-Collision, Function: metaphone3
4. Method: Key-Collision, Function: cologne-phonetic
5. Method: Nearest Neighbor, Function: PPM (Radius: 1, Block Chars: 6)
6. Method: Nearest Neighbor,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

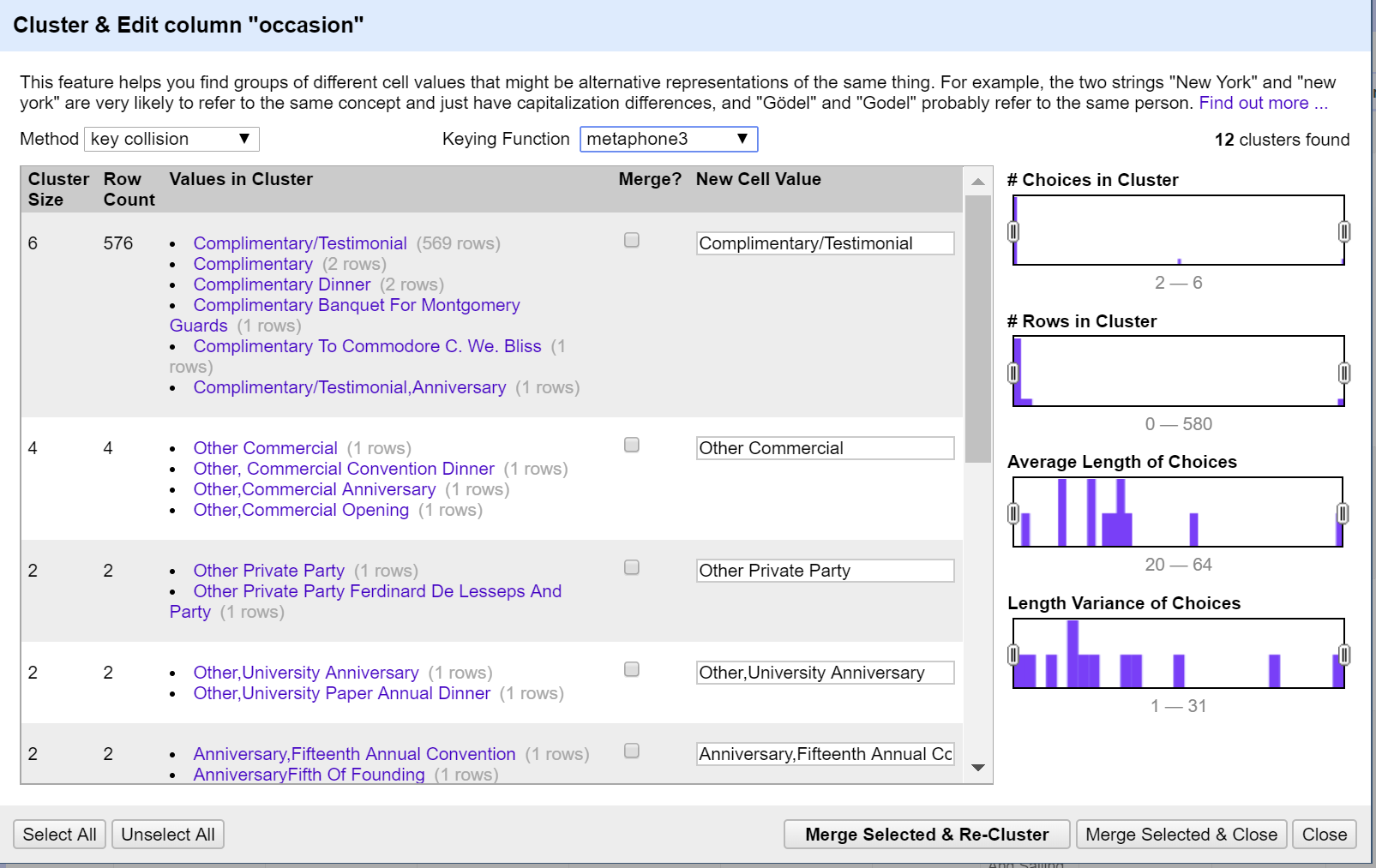
Screenshot:



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

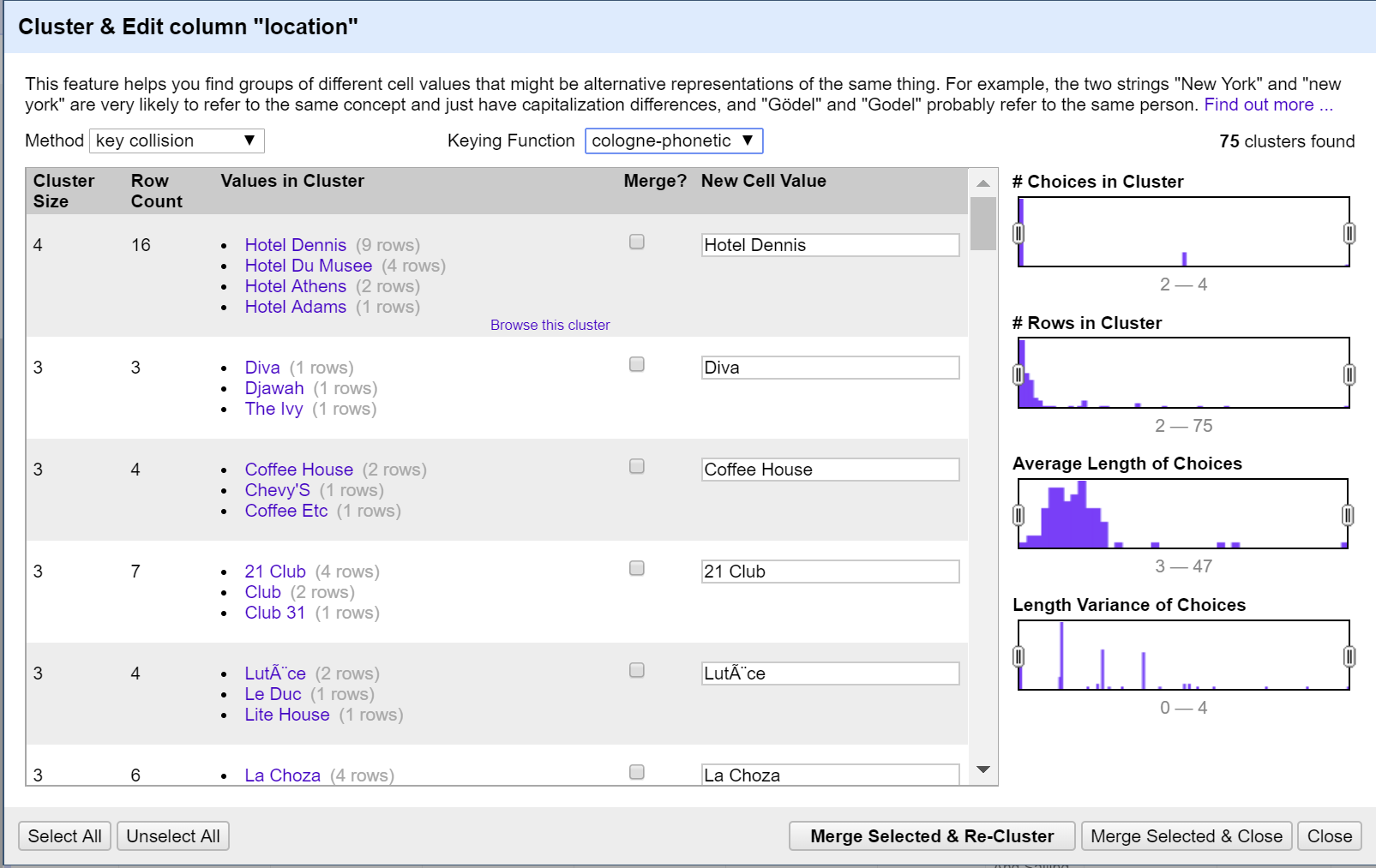
Screenshot:



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

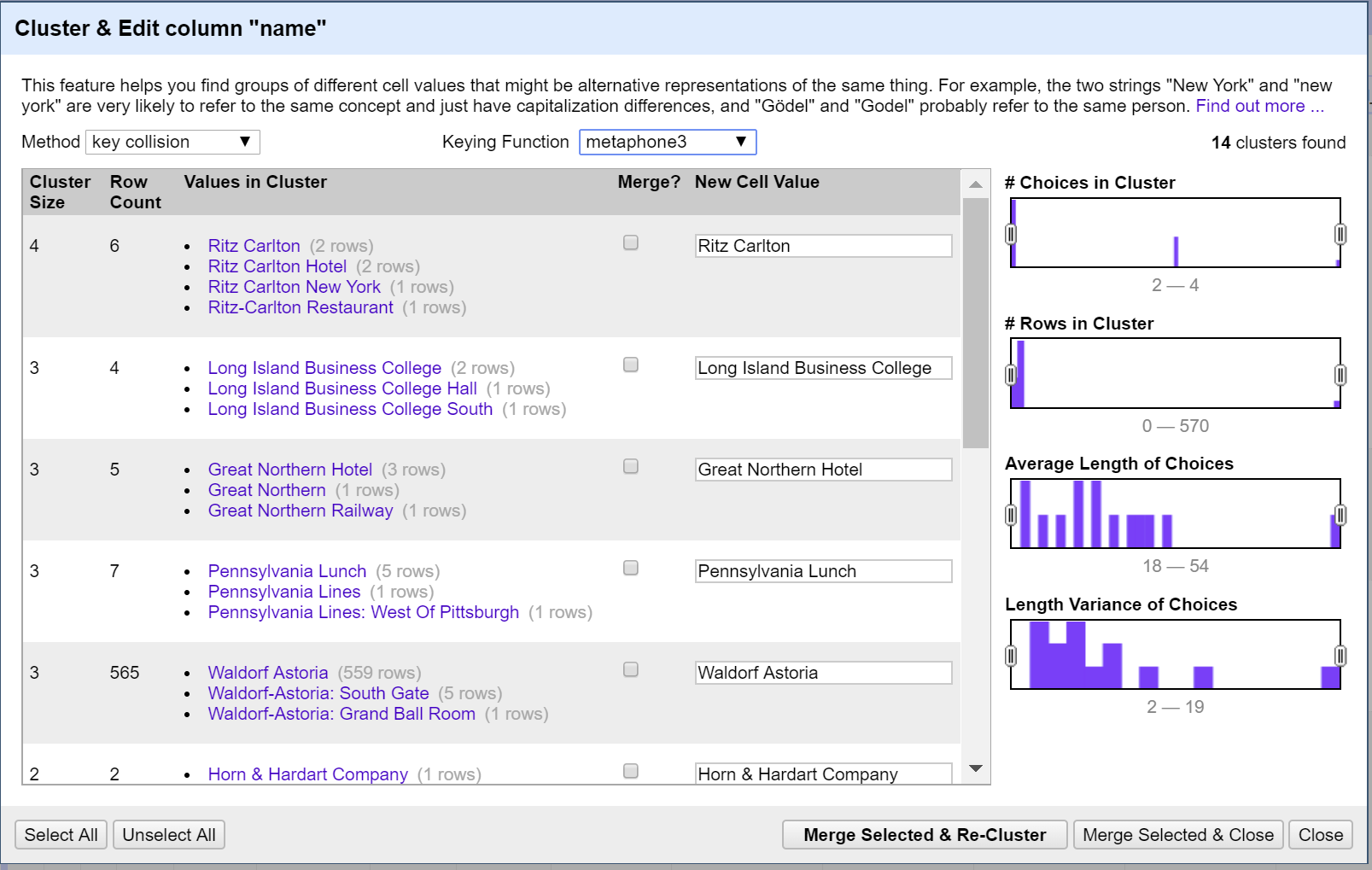
Screenshot:



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

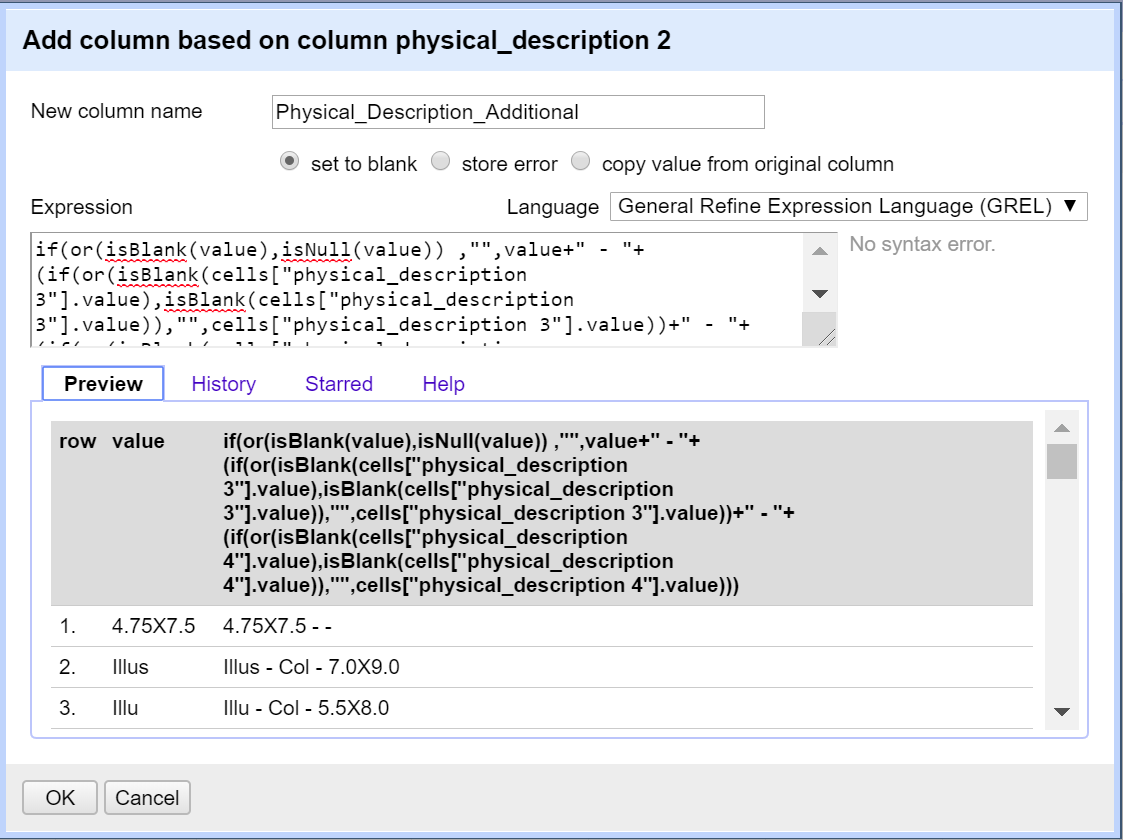
We updated the column physical\_description 1 to physical description type and then created a new column Physical\_description\_additional by merging following columns:

physical\_description 2

physical\_description 3

physical\_description 4

Screenshot:



File: Dish.csv

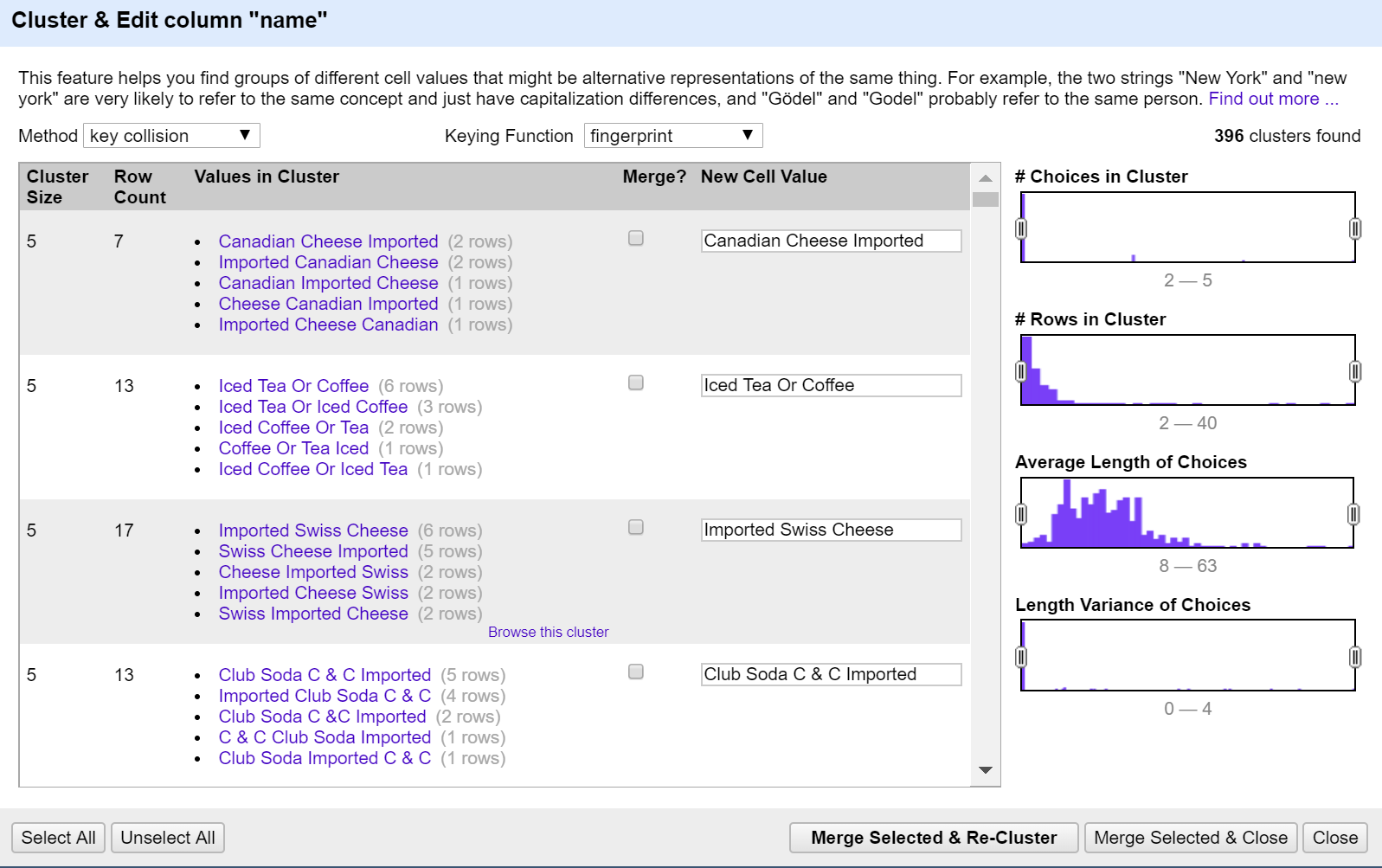
Step 1: We found even after cleaning with python script, there were few different kinds of special characters were present in the file. So, we transform data to remove unwanted characters. Formula:

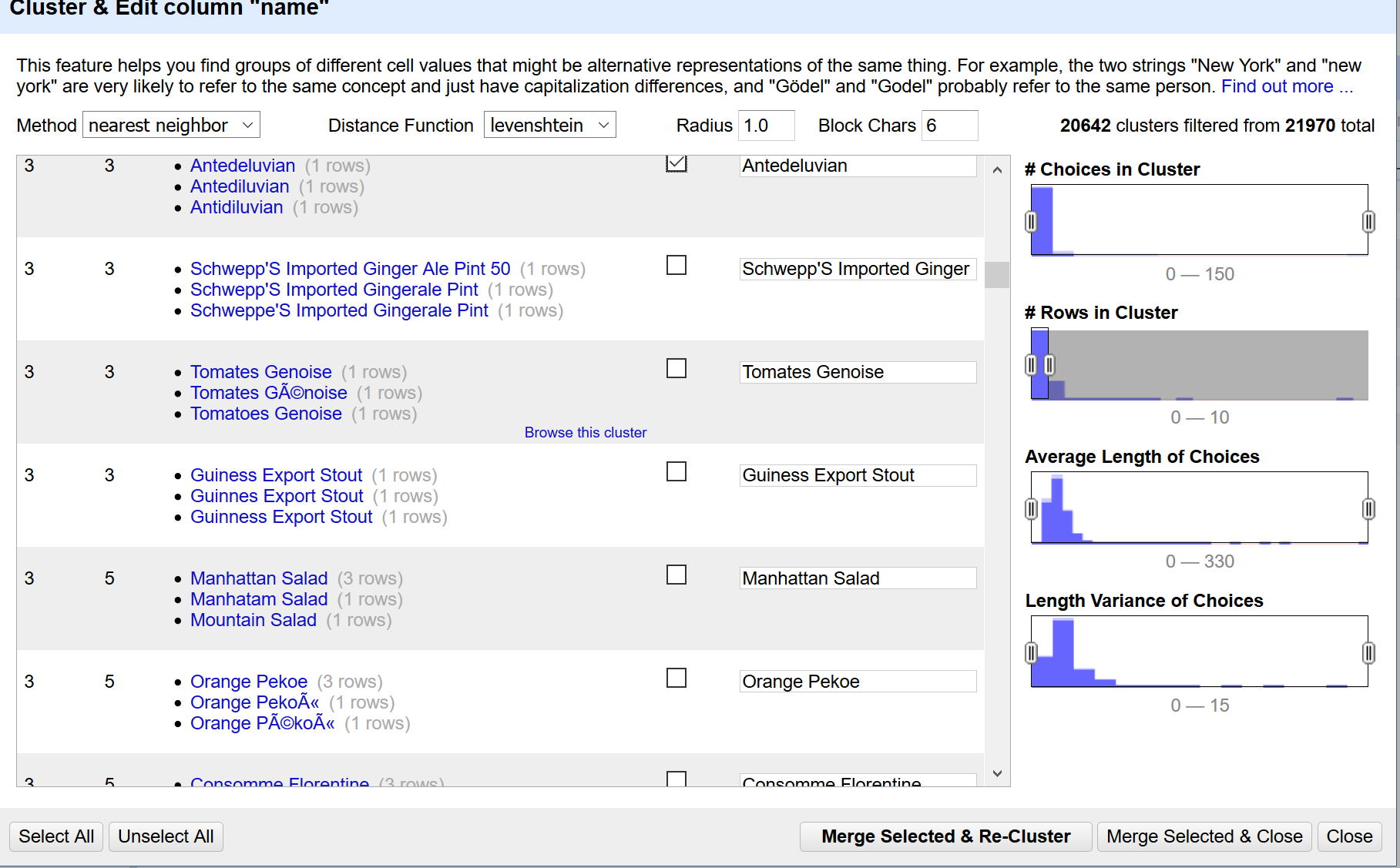
value.replace(/[>:<%#@!\\()\[\]\?\"\-\\*,\.\+]/, " ").replace(/\s+/," ").trim()

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:





File: 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

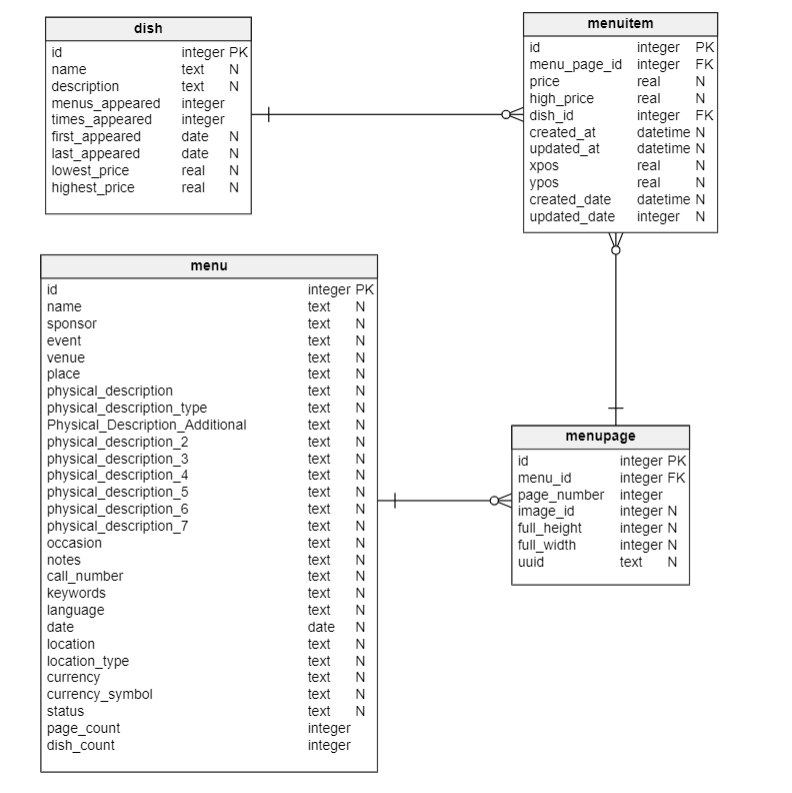
File: MenuPage.csv

No transformations were performed for this csv in open refine.

# Step 3: Integrity Checks using SQLite database constraints

Database Model

Based on the cleaned data from step 2, we created a database model which included the logical integrity constraints of ‘Primary Key’, ‘Foreign Key’ and ‘Not Null’. The database model also defined the one-to-one and one-to-many relationships between the four tables. Building the schema based on this model ensured that only the records satisfying all the defined constraints are loaded into the tables and the dirty data is discarded. Here is the ER diagram for our DB model:

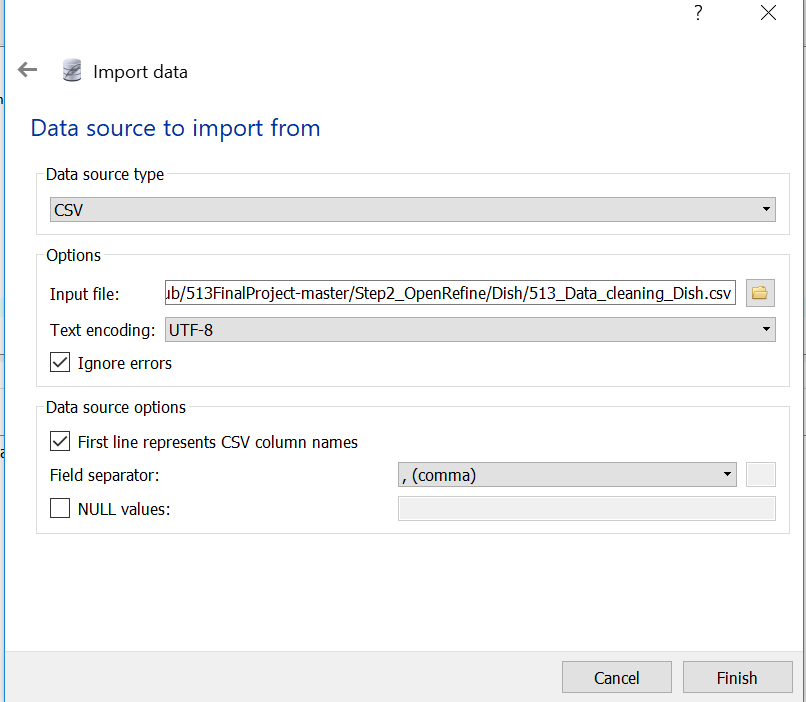


Tables Creation and Data Import

A database schema ‘nypl’ was created using SQLiteStudio. We created 4 tables and imported data into these tables from the cleaned csv files. As a result, following 4 tables were created:

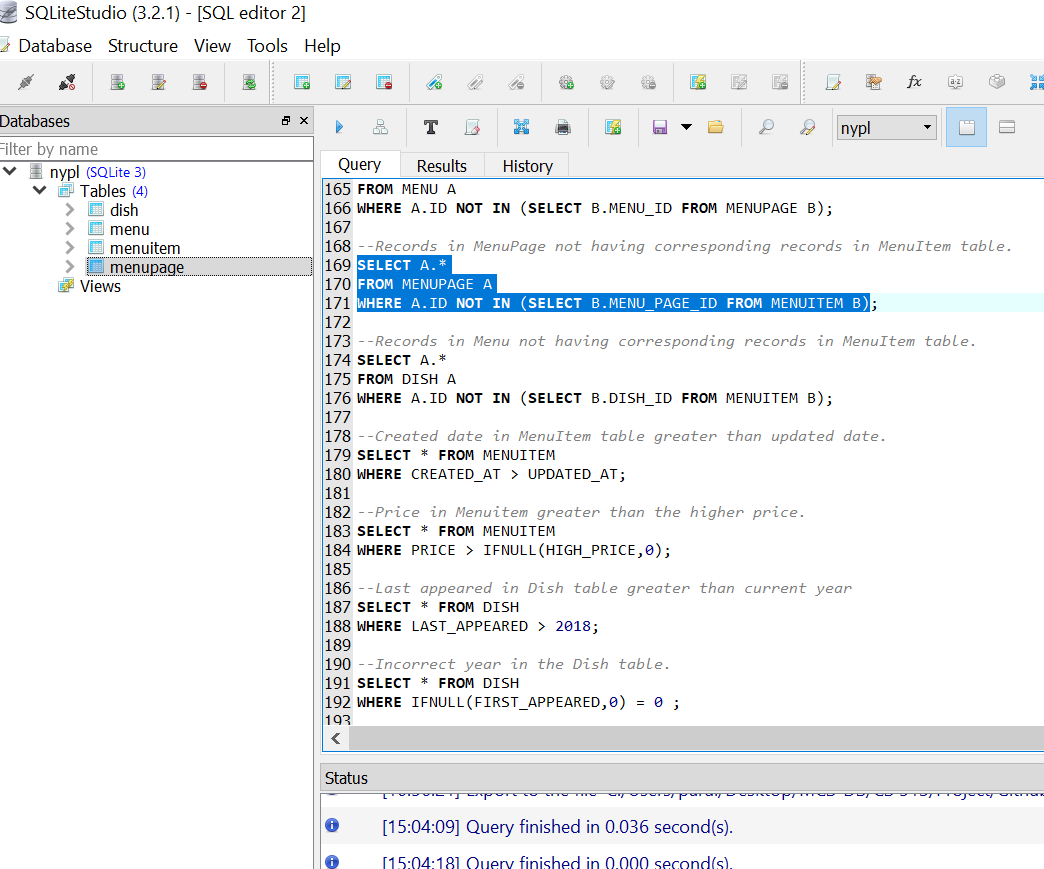
* Dish
* Menu
* Menupage
* MenuItem

Screenshot



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.

### 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.

### 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.

### 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.

### 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.

# YesWorkflow

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

Key Inputs/Outputs

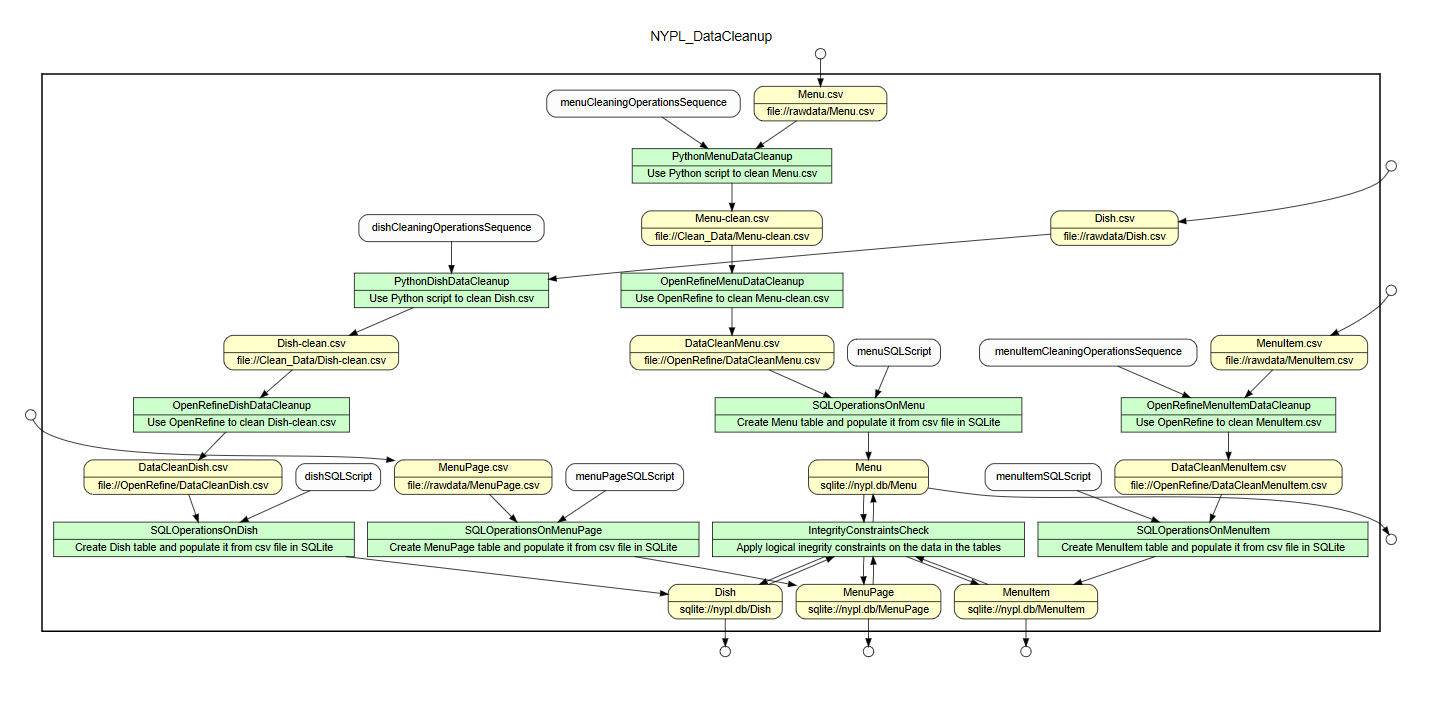
The key inputs to the workflow process are the four input data files namely, menu.csv, dish.csv, menuitem.csv, menupage.csv along with the data cleaning operation comprising of Openrefine and Python script.

## Dependencies

The complete workflow includes several 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.

## Diagram

Below is a snapshot of the Workflow Diagram:



# Challenges

## Ambiguity

One of the key challenges we faced was ambiguity in data. We did not intend to remove the ambiguity in the data as that would have required to make assumption. For example, if we found an item as omelette 1 and omelette 2 then we did not try to merge it to “omelette”. The reason behind that was that these two could be a variant of omelette with different ingredient and sometime different size of servings. Assuming them to represent one dish could hamper the data integrity. Thus, there was a tradeoff between ambiguity and integrity. So, we left those cases as it is

## OpenRefine Efficiency

Open refine was not able to load large dataset such as dish.csv. We had to change following configurations in refine.ini

REFINE\_MAX\_FORM\_CONTENT\_SIZE=99999999

REFINE\_MEMORY=4096M

Same can be achieved by following command: openrefine /m 4096M.

Further, for some of the operations, especially facet creation on large dataset, open refine was not able to perform optimally. We had to reduce the data size by applying a filter and then we were able to create the facet. We also found that it is more time efficient to apply clustering directly instead of first creating facet and then applying the clustering.

# Tableau

# Appendix

Dataset

New York Public Library Menus data

## Tools/Software Used

|  |  |  |
| --- | --- | --- |
| **Software** | **Version/Url** | **Usage** |
| Python | 3.6.0 | Scripting Language used for Data Cleaning |
| OpenRefine | 2.6-rc.2 | Data Cleaning Tool |
| SQLiteStudio | 3.2.1 | UI tool for working with SQLite |
| YesWorkflow | http://try.yesworkflow.org/ | Provenance Tool |
| Tableau | 2018.1.2 | Visualization Representation |
| Vetabelo | https://my.vertabelo.com/ | Database Modeler |
| GitHub | https://github.com/rohitbansal83/513FinalProject | Source control repository |
| Slack | https://uiuc-mcsds.slack.com/ | Communication channel used for project work |
| zoom | https://www.zoom.us/ | Video Conferencing tool for coordination |

## 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).