CS 598 Theory & Practice of Data Cleaning

Final Project Report

Cleaning the NYPL menus' Dataset

Abhisek Jana (abhisek2)
Ashish Kumar(ashishk2)
Aruna Neervannan(arunarn2)

Table of Contents

1 Introduction

We have selected the **New York Public Library's** crowd sourced **historical menus dataset** available here for our project (our local copy). This dataset is an interesting collection of historical menus compiled and curated by The New York Public Library as part of their digitizing effort. The collection includes about 45,000 menus from the 1840s to the present, and the goal of the digitization project is to transcribe each page of each menu, creating an enormous database of dishes, prices, locations, and so on. As of 07/01/2017, the transcribed database contains 423,388 dishes from 17,546 menus. Here are some interesting on details on the NYPL project and the purpose of this transcription.

The dataset consists of following CSV files:

- Menu
- MenuPage
- MenuItem
- Dish

2 Structure of Dataset

Menu is the key table of this database. Each row in this table represents a menu and has following columns:

- id – An int type field used to uniquely identify the record
- name – A string field and there are lots of empty entries. When not empty, it appears to be a duplicate of the "sponsor" field
- sponsor – A string field typically representing the name of the hotel or a venue. All entries are in upper case.
- event – A string field representing the name event for the menu was created. Typical values are Breakfast, Lunch, Dinner etc.
- venue – A string field representing the type of facility – commercial, social. This column has missing entries.
- place – A string field representing the location of the sponsor.
- physical Description – This is a compound field reflecting type of the menu and the dimensions of the menu
- notes – Additional details about the menu card are stored in this string field
- call_number – A range of numbers and its intent is not clear
- keywords – empty field
- language – empty field
- date – MM/DD/YY format. Some entries are in YYYY/MM/DD format and several are missing
- location – A string field, appears to be a duplicate of sponsor except entries are in the mixed case
- location_type – An empty string field

- currency – Field to capture currency on the menu. The field is either empty or has Dollars value
- currency_symbol – Field to capture currency's sign. The field is either empty or has a $ symbol
- status – A categorical value to reflect whether entry is complete or is it under view
- page_count – A numeric field to store number of pages in the menu
- dish_count – A numeric field to store number of dishes in a menu

Dish table represents the dishes that appeared on a menu and has following columns

- id – A numeric field to store id of the entry
- name – A string field to store the name of the dish
- description – A string field, can be empty or stores description of the dish
- menus_appeared – A numeric field, purpose unclear
- time_appeared – a numeric field, purpose unclear
- first_appeared– A number field storing the year when dish appeared first in a menu
- last_appeared – A number field storing the year when dish appeared last in a menu
- lowest_price – A floating point number to store lowest price, has no current symbol.
- highest_price– A floating point number to store lowest price, has no current symbol.

MenuItem is the one that links a MenuPage to a Dish and has following columns

- id – table entry ID
- menu_page – numeric field to identify menu page, can be used as a key for the menu page dataset
- price – floating number
- high_price – highest price compared to original price
- dish_id – numeric field to identify the dish, can be used to identify dish in the dish dataset
- created_at - Timestamp field shows when entry was created
- updated_at – Timestamp field shows when entry was last updated
- xpos - floating number representing location of item on the menu on the x-axis
- ypos - floating number representing location of item on the menu on the y-axis

Each row here represents a menu item on the menu page.

MenuPage represent a page on a menu (linked via menu_id column) and has following columns

- id – table entry identifier
- menu_id – numeric field to store menu id, can be used as a key for the menu dataset
- page_number – numeric field to store the specific page number of a given menu
- image_id – identifier for the image of the menu page. Images of menu pages are stored in some cases.
- full_height – numeric field, no unit of measure provided, assumed to be in millimeters
- fullwidth– numeric field, no unit of measure provided, assumed to be in millimeters
- uuid – A composite of 5 field of alpha-numeric values separated by a dash

2.1 Potential Use Cases

- Create a chronicle of food evolution in the 19th and 20th century.
- What were the most popular dishes? Were entrees more popular than dessert?
- Is there a relationship between dish and the venue?
- Were certain dishes popular on certain occasions?

2.2 Use Cases requiring no clean up

- A deeper analysis of dish dataset can help uncover popular dishes (based on how many different menus listed these dishes) and how long they were popular by calculating the difference between first appeared and last appeared values
- MenuPage csv files for the most part require no major cleanup and can provide historical information on structure of menu in the late 19th and early 20th century.

2.3 Will it ever be good enough?

- The MenuItem dataset has lot of missing values in the high_price and dish_id column.
- Since there were too many values missing we did not fill the column with the median price of an item.
- The created_at field and updated_at field may be useful for the data curator, however it has a very limited value for a data analyst.
- Several columns in the Menu CSV have missing values too.

3 OpenRefine

3.1 Environment Setup

For this project, we used OpenRefine 2.7. After install, we edited the refine.ini accommodate the large transformations in the csv files and modified the following defaults to:

```
REFINE_MAX_FORM_CONTENT_SIZE=14521430
REFINE_MEMORY=4096M
```

Launched the OpenRefine webapp using the refine command line script.

3.2 Changes in Menu CSV

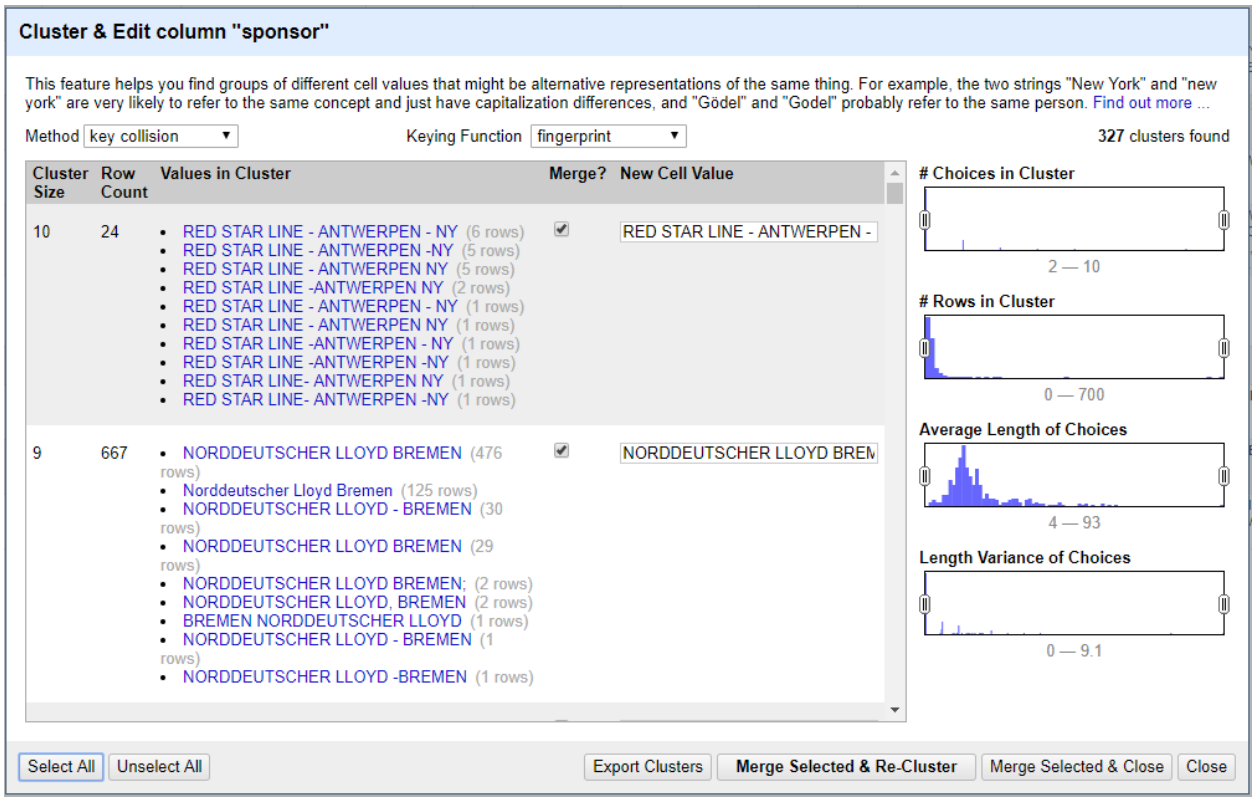
3.2.1 Transformations on "sponsor" column

1. Clustering operation with keying function as "fingerprint". For example, this would replace the following values:

```
"RED STAR LINE - ANTWERPEN -NY",
"RED STAR LINE - ANTWERPEN NY",
```

"RED STAR LINE -ANTWERPEN NY",
"RED STAR LINE - ANTWERPEN - NY",
"RED STAR LINE - ANTWERPEN - NY",
"RED STAR LINE - ANTWERPEN NY",
"RED STAR LINE -ANTWERPEN - NY",
"RED STAR LINE -ANTWERPEN -NY",
"RED STAR LINE- ANTWERPEN NY",
"RED STAR LINE- ANTWERPEN -NY"

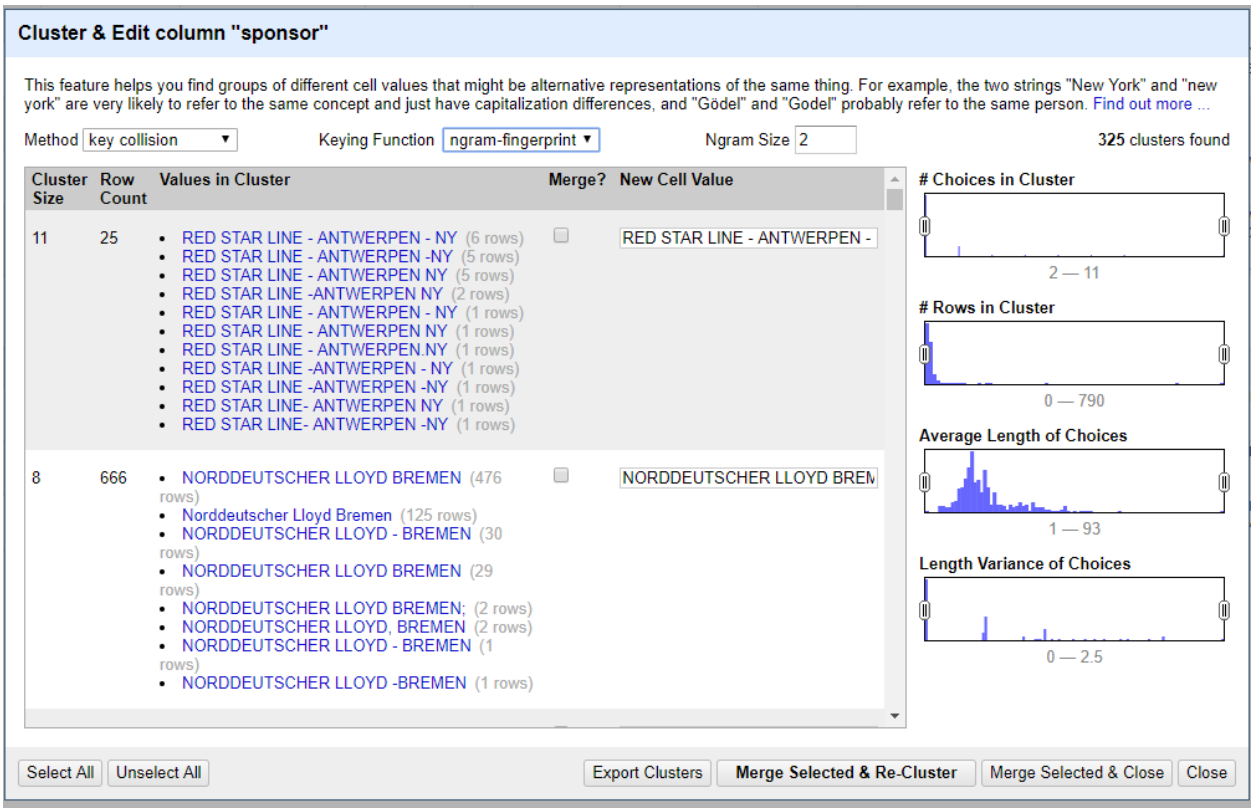
with "RED STAR LINE - ANTWERPEN - NY"



2. Clustering operation with keying function as “ngram-fingerprint” with ngram size as 2. For example, this would replace the following values:

   "NIPPON YUSEN KAISHA - S.S.KOBE MARU",
   "NIPPON YUSEN KAISHA - S.S. KOBE MARU",
   "NIPPPON YUSEN KAISHA - S.S. KOBE MARU"

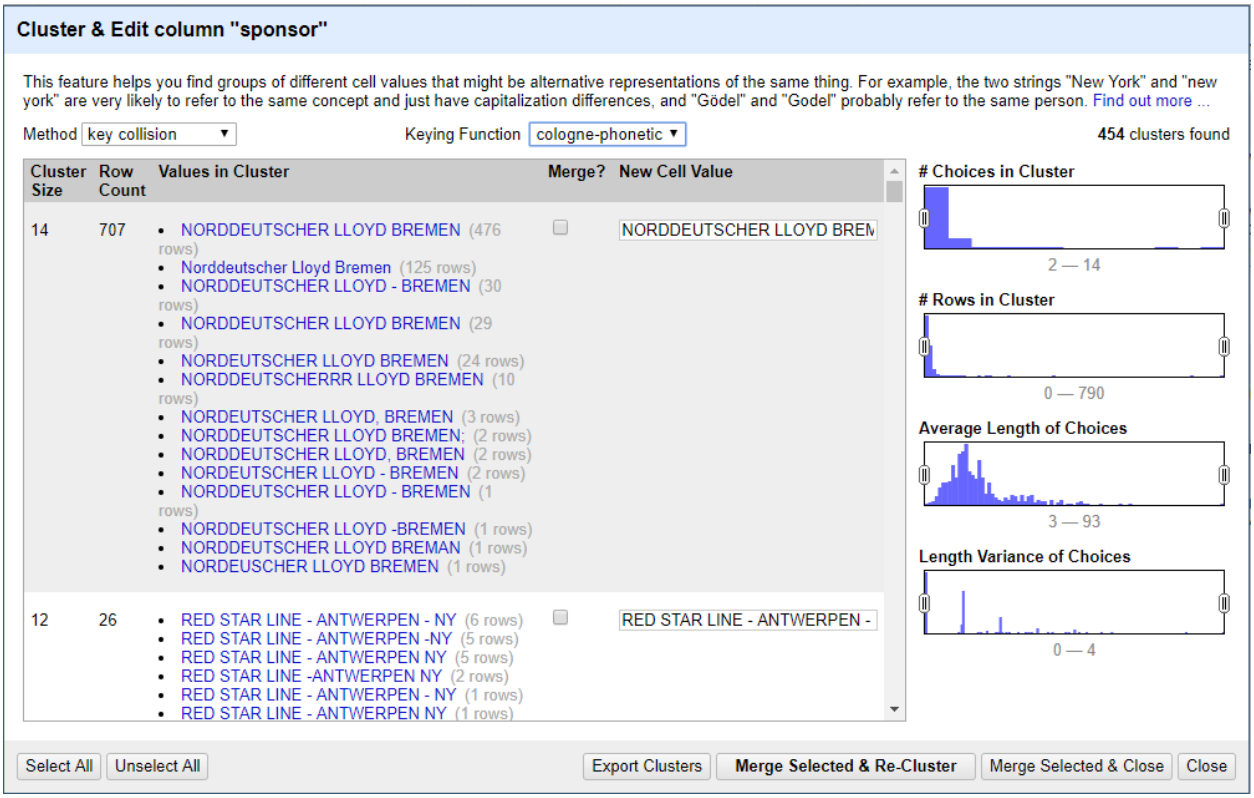
   with "NIPPON YUSEN KAISHA - S.S.KOBE MARU"



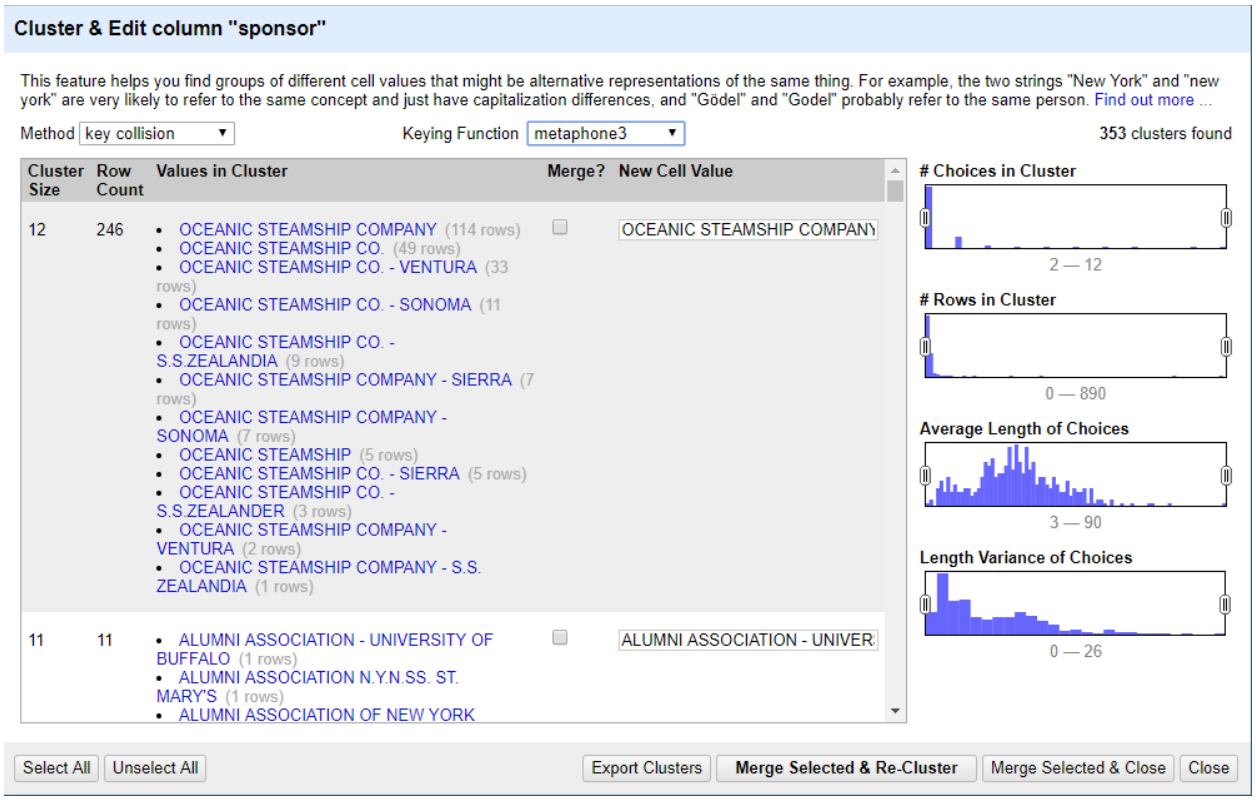
3. Clustering operation with keying function as “colgne-phonetic”. For example, this would replace the following values (using some common-sense and logic to make sure that the values made sense for the scenario):

    "THE LOTOS CLUB",
    "THE LOTUS CLUB"

    with "THE LOTUS CLUB"



4. Trim leading and trailing space, collapse consecutive whitespaces and convert all to UPPERCASE.
5. Replace special characters "[,],(,),*,%," with "".
6. We **did not choose** the clustering operation with keying function as “metaphone3” as it results in losing the existing organization breakdown per state or per county.

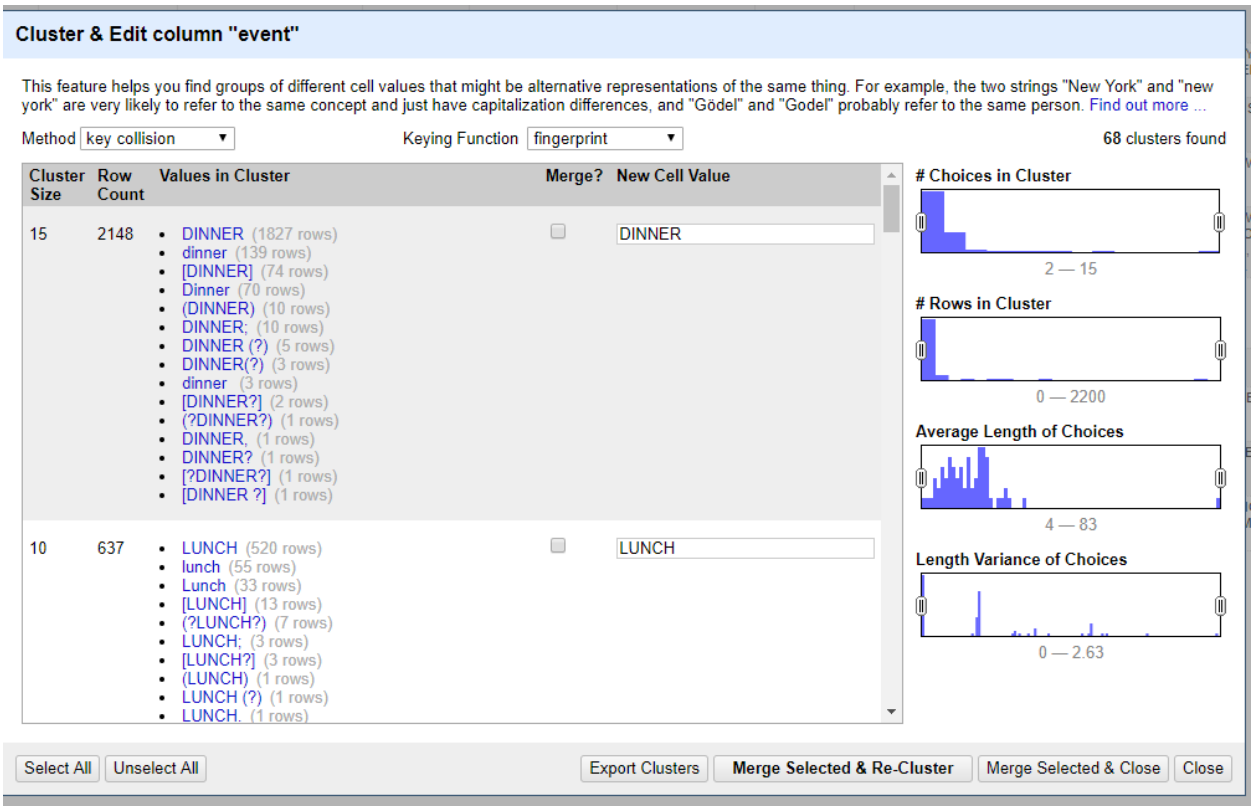


3.2.2 Transformations on "event" column

1. Clustering operation with keying function as "fingerprint". For example, this would replace the following values:

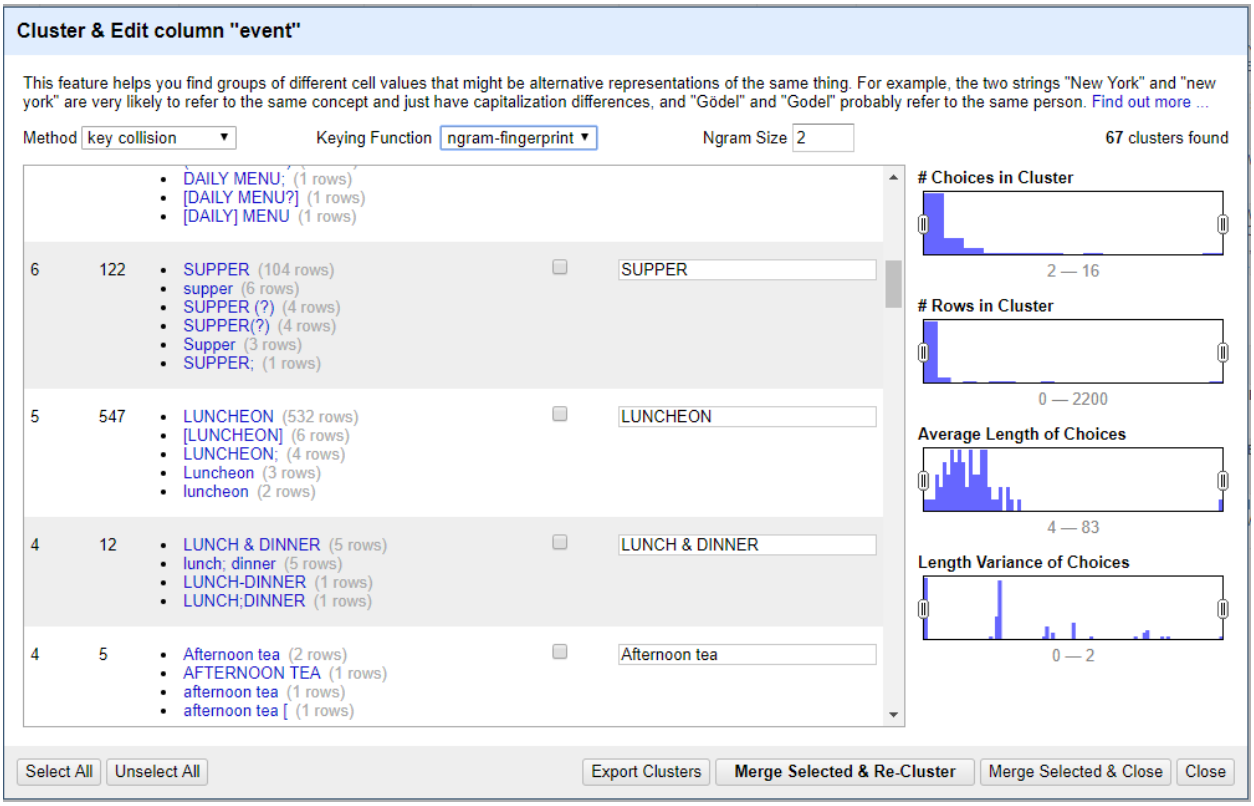
"BREAKFAST",
"Breakfast",
"breakfast",
"[BREAKFAST]",
"BREAKFAST;",
"BREAKFAST (?)",
"BREAKFAST(?)",
"[BREAKFAST ?]"

with "BREAKFAST"



2. Clustering operation with keying function as “ngram-fingerprint” with ngram size as 2. For example, this would replace the following values:
"DINNER/DINER",
"DINER/DINNER",
"DINNER, DINER"

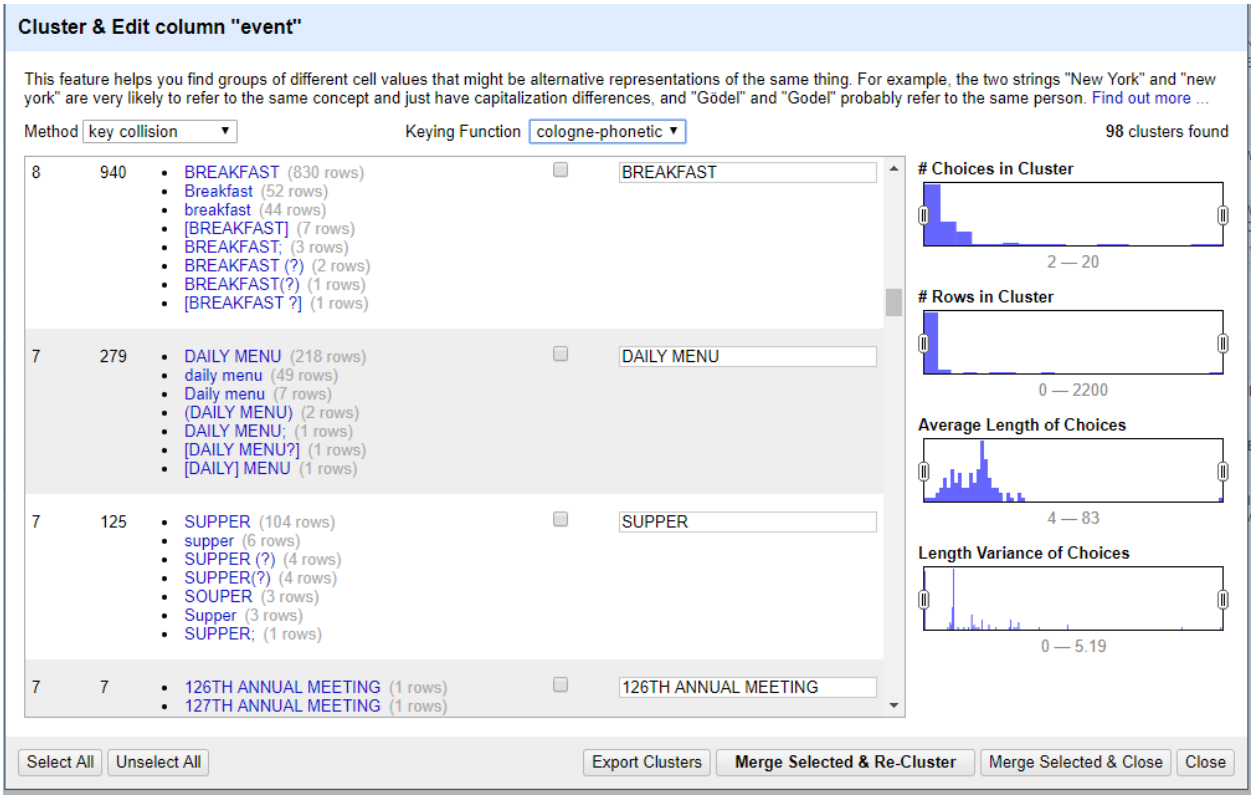
with "DINNER"



3. Clustering operation with keying function as “colgne-phonetic”. For example, this would replace the following values (using some common-sense and logic to make sure that the values made sense for the scenario):

   "LUNCH AND DINNER",
   "LUMCH AND DINNER"

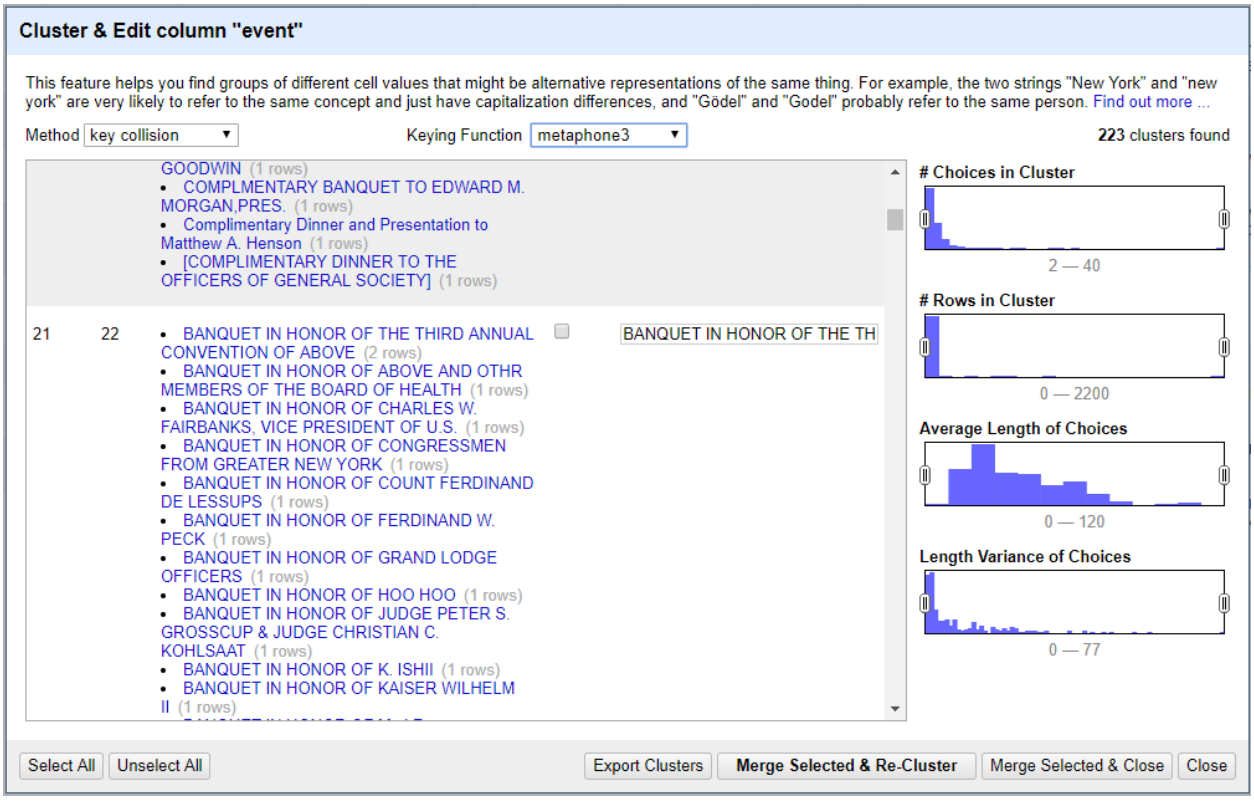
   with "LUNCH AND DINNER"



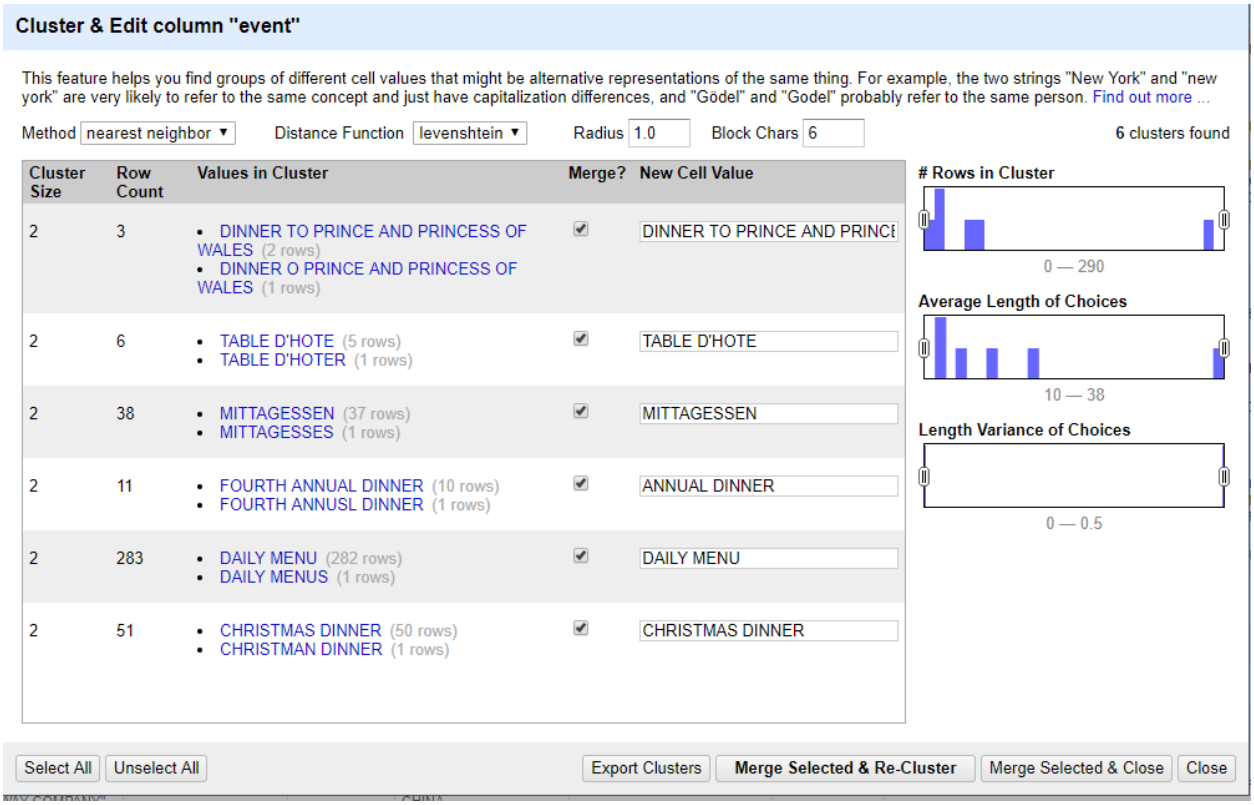
4. Trim leading and trailing space, collapse consecutive whitespaces and convert all to UPPERCASE.
5. Replace special characters "[,],(,),*,%," with "".
6. Clustering operation with keying function as “metaphone3” to merge some of obvious ones, for example:

"11TH ANNUAL DINNER",
"15TH ANNUAL DINNER",
"4TH ANNUAL DINNER",
"14TH ANNUAL DINNER",
"18TH ANNUAL DINNER",
"20TH ANNUAL DINNER",
"12TH ANNUAL DINNER",
"13TH ANNUAL DINNER",
"16TH ANNUAL DINNER",
"25TH ANNUAL DINNER",
"28TH ANNUAL DINNER",
"30TH ANNUAL DINNER",
"34TH ANNUAL DINNER",
"37TH ANNUAL DINNER",
"57TH ANNUAL DINNER",
"5TH ANNUAL DINNER",
"7TH ANNUAL DINNER",
were replaced with "ANNUAL DINNER"

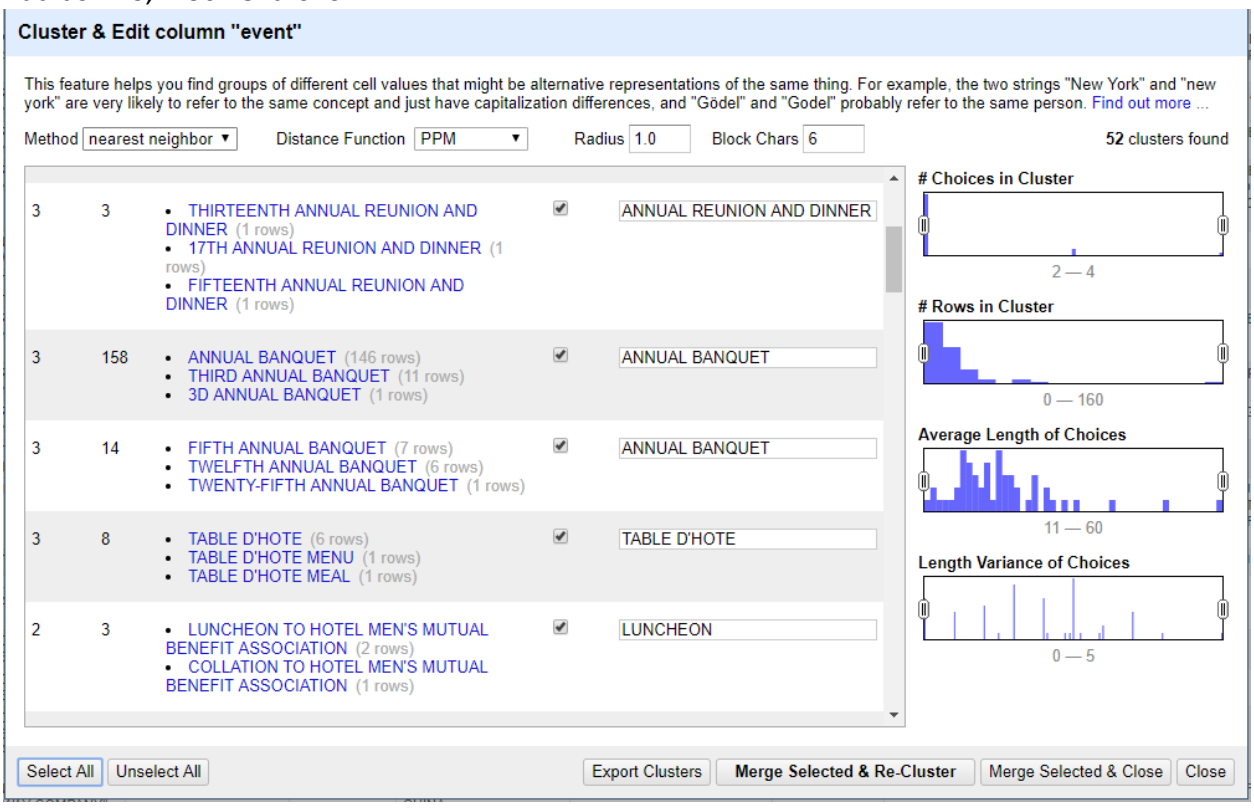
Some of the recommendations from OpenRefine were not taken as they did not make sense to our use case.



7. Clustering operation with Method as “nearest neighbor”, distance function as” levenshtein”, Radius=1.0, Block Chars=6



8. Clustering operation with Method as “nearest neighbor”, distance function as” PPM”, Radius=1.0, Block Chars=6

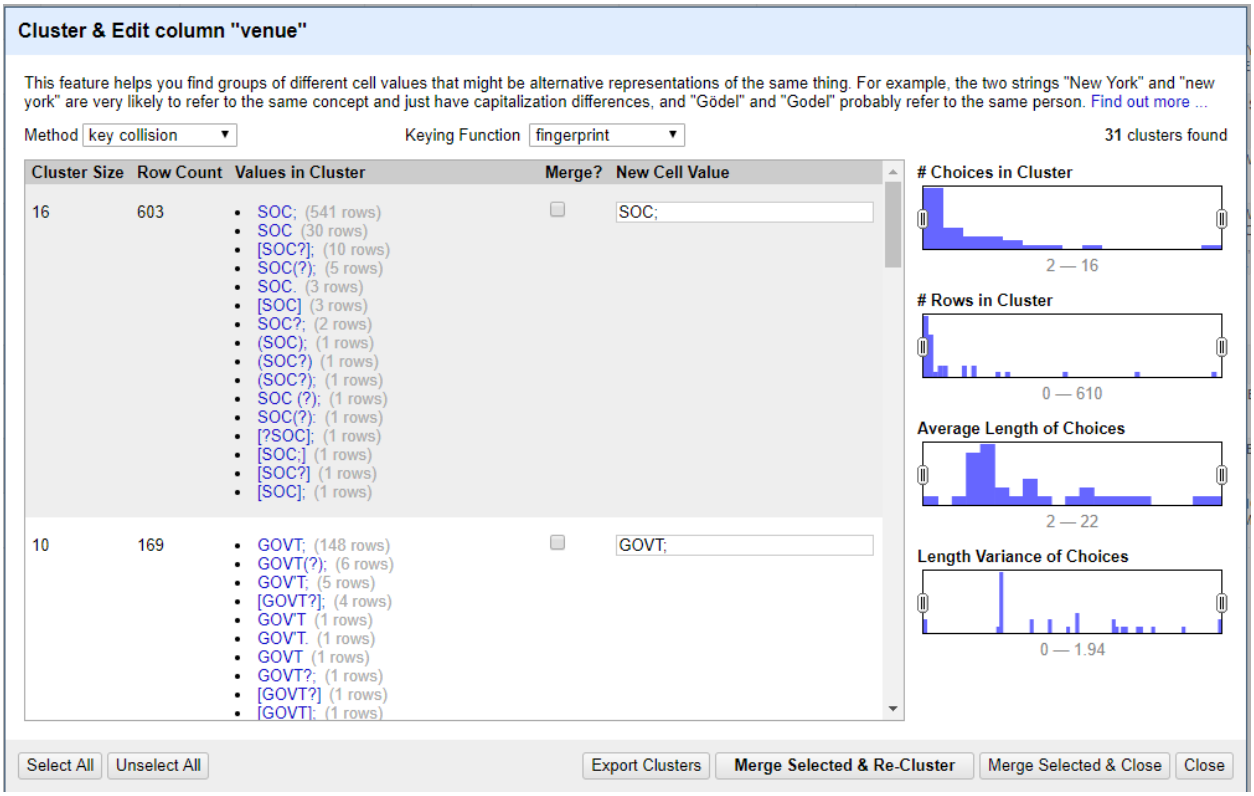


3.2.3 Transformations on “venue” column

1. Clustering operation with keying function as “fingerprint”. For example, this would replace the following values:

   "EDUC;",
   "EDUC",
   "EDUC (?);",
   "[EDUC];",
   "(EDUC);",
   "[EDUC?];"

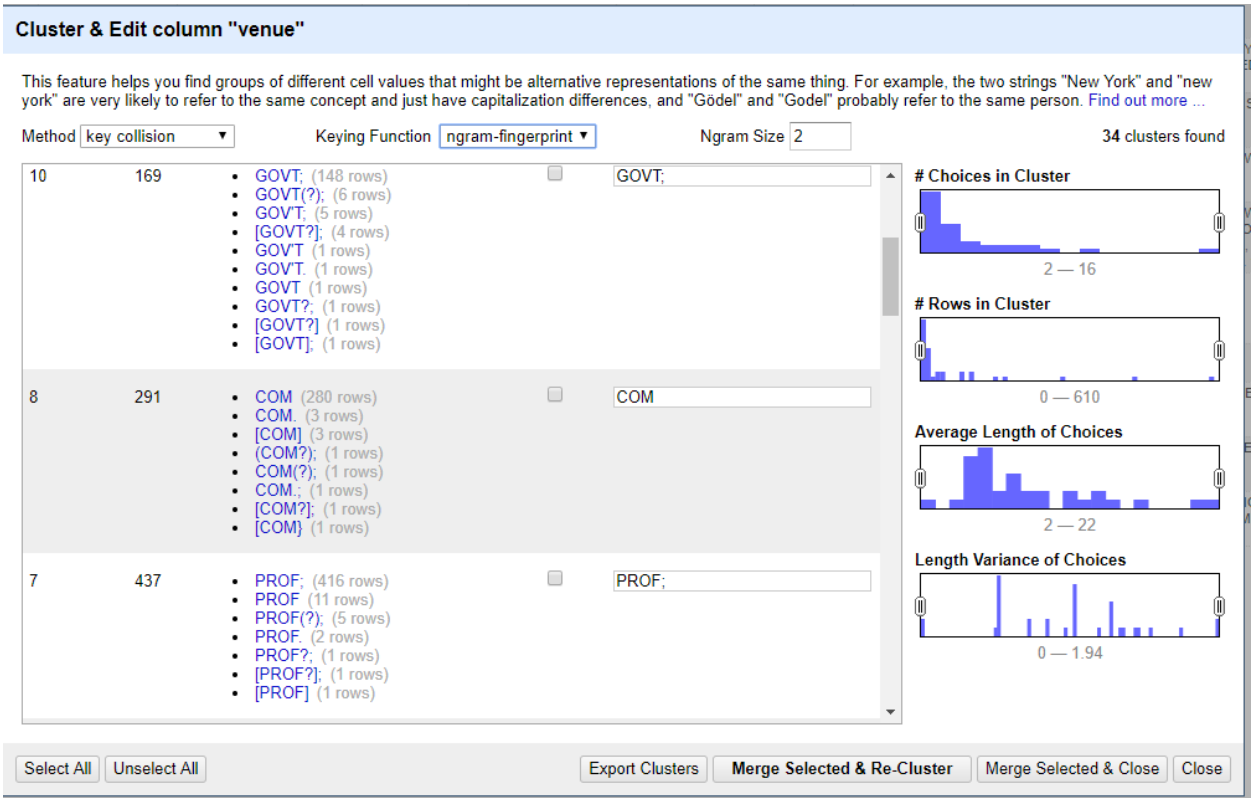
   with "EDU"



2. Clustering operation with keying function as “ngram-fingerprint” with ngram size as 2. For example, this would replace the following values:

   "SECULAR HOLIDAY;",
   "SECULAR HOLIDAY",
   "SECULAR HOLIDAY. HOLIDAY",
   "SECULAR HOLIDAY. HOLIDAY;",
   "SECULAR HOLIDAY HOLIDAY;",
   "SECULAR HOLIDAY HOLIDAY"

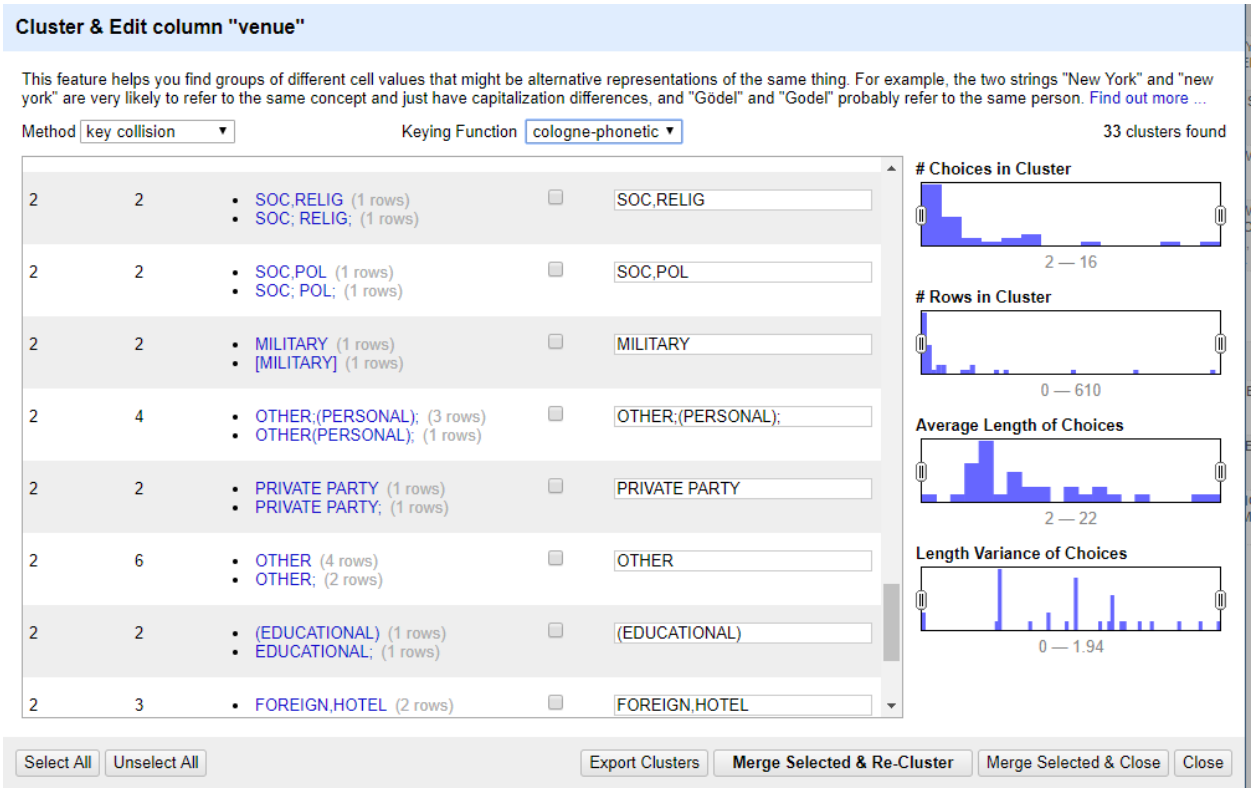
   with "SECULAR HOLIDAY"



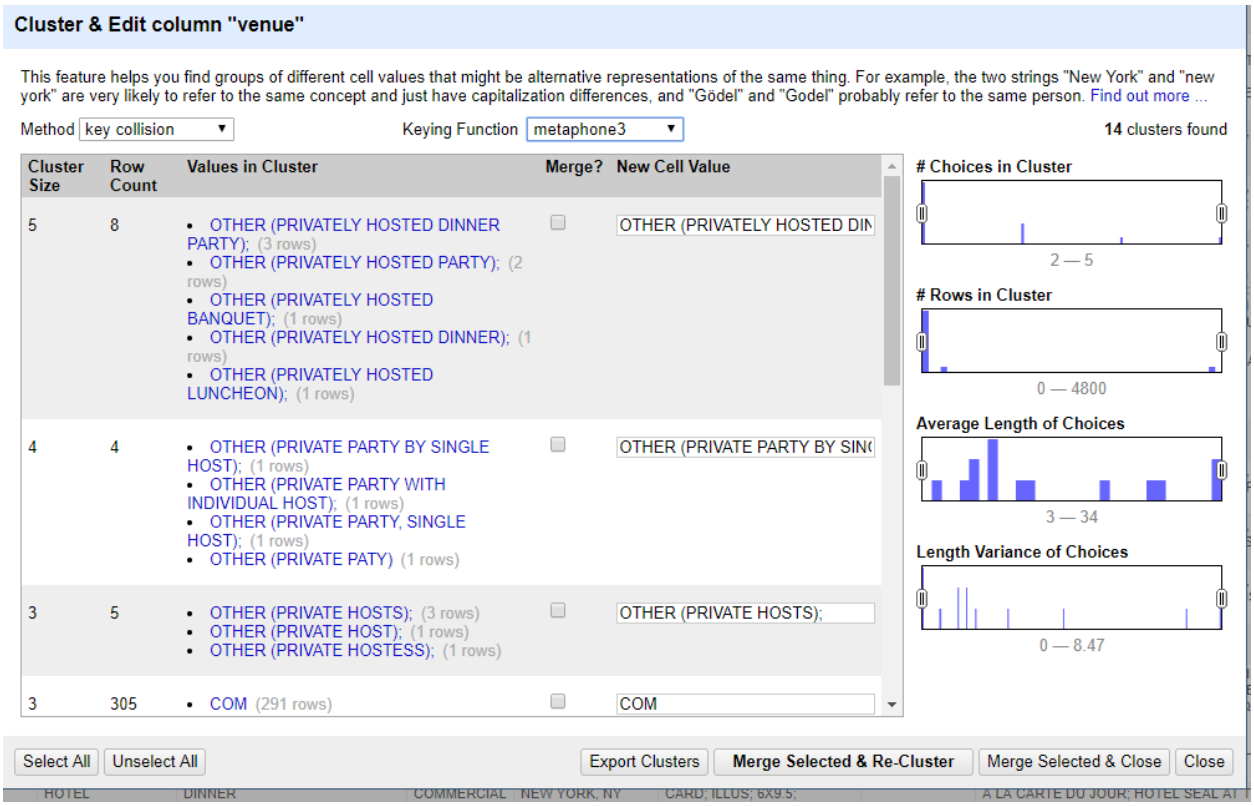
3. Clustering operation with keying function as “colgne-phonetic”. For example, this would replace the following values (using some common-sense and logic to make sure that the values made sense for the scenario):

   "DINNER A LA CARTE",
   "DINNER-A LA CARTE"

   with “DINNER A LA CARTE"



4. Trim leading and trailing space, collapse consecutive whitespaces and convert all to UPPERCASE.
5. Replace special characters "[,],(,),*,%," with "".
6. We did not choose the clustering operation with keying function as “metaphone3” as it results in losing the existing event breakdown.

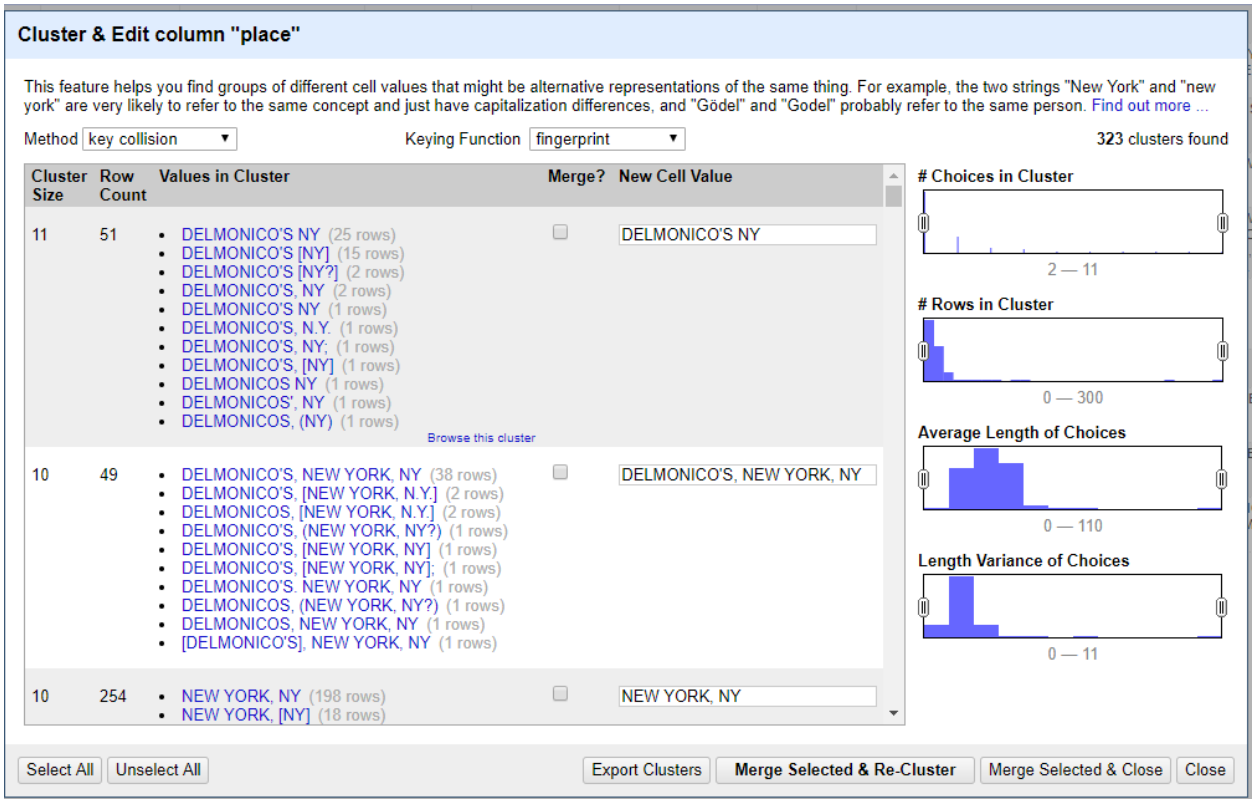


3.2.4 Transformation on “place” column

1. Clustering operation with keying function as “fingerprint”. For example, this would replace the following values:

   "DELMONICO'S NY",
   "DELMONICO'S [NY]",
   "DELMONICO'S [NY?]",
   "DELMONICO'S, NY",
   "DELMONICO'S NY",
   "DELMONICO'S, N.Y.",
   "DELMONICO'S, NY;",
   "DELMONICO'S, [NY]",
   "DELMONICOS NY",
   "DELMONICOS', NY",
   "DELMONICOS, (NY)"

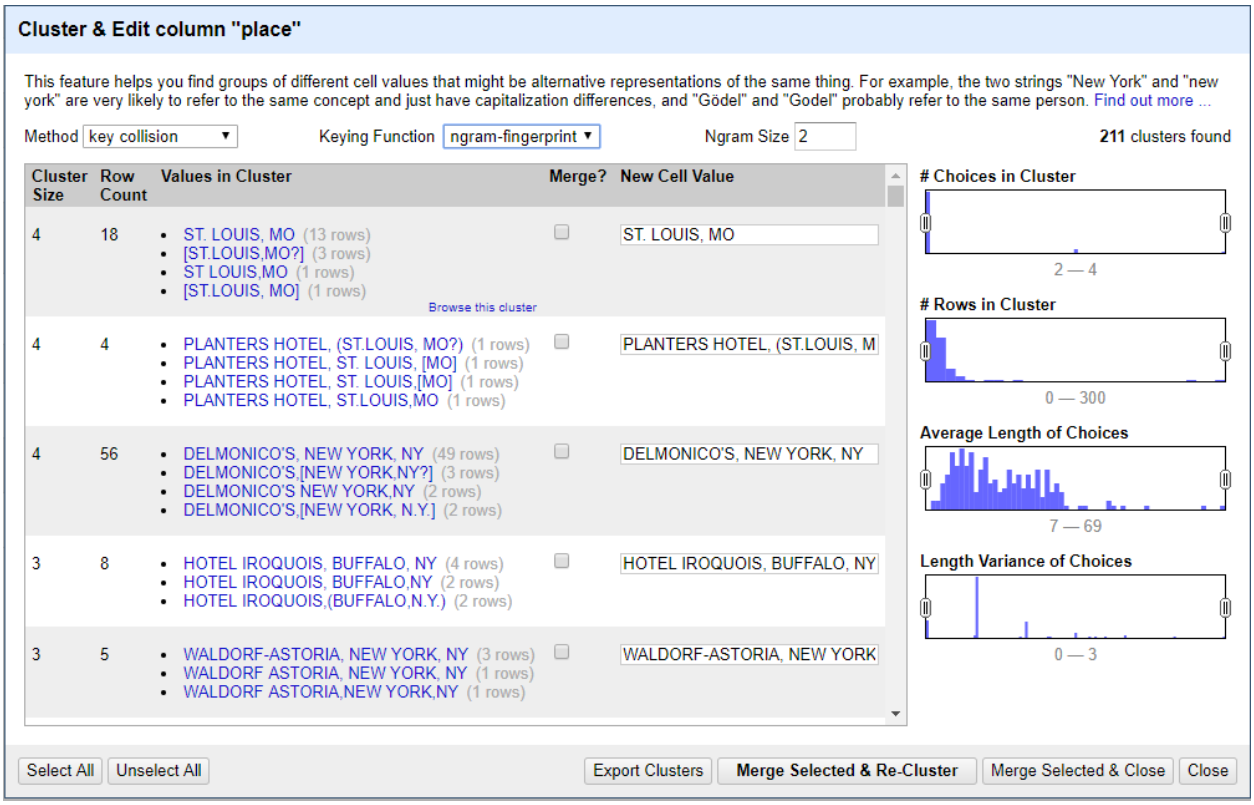
   with "DELMONICO'S, NEW YORK, NY"



2. Clustering operation with keying function as “ngram-fingerprint” with ngram size as 2.
   For example, this would replace the following values:

   "PLANTERS HOTEL, (ST.LOUIS, MO?)",
   "PLANTERS HOTEL, ST. LOUIS, [MO]",
   "PLANTERS HOTEL, ST. LOUIS,[MO]",
   "PLANTERS HOTEL, ST.LOUIS,MO"

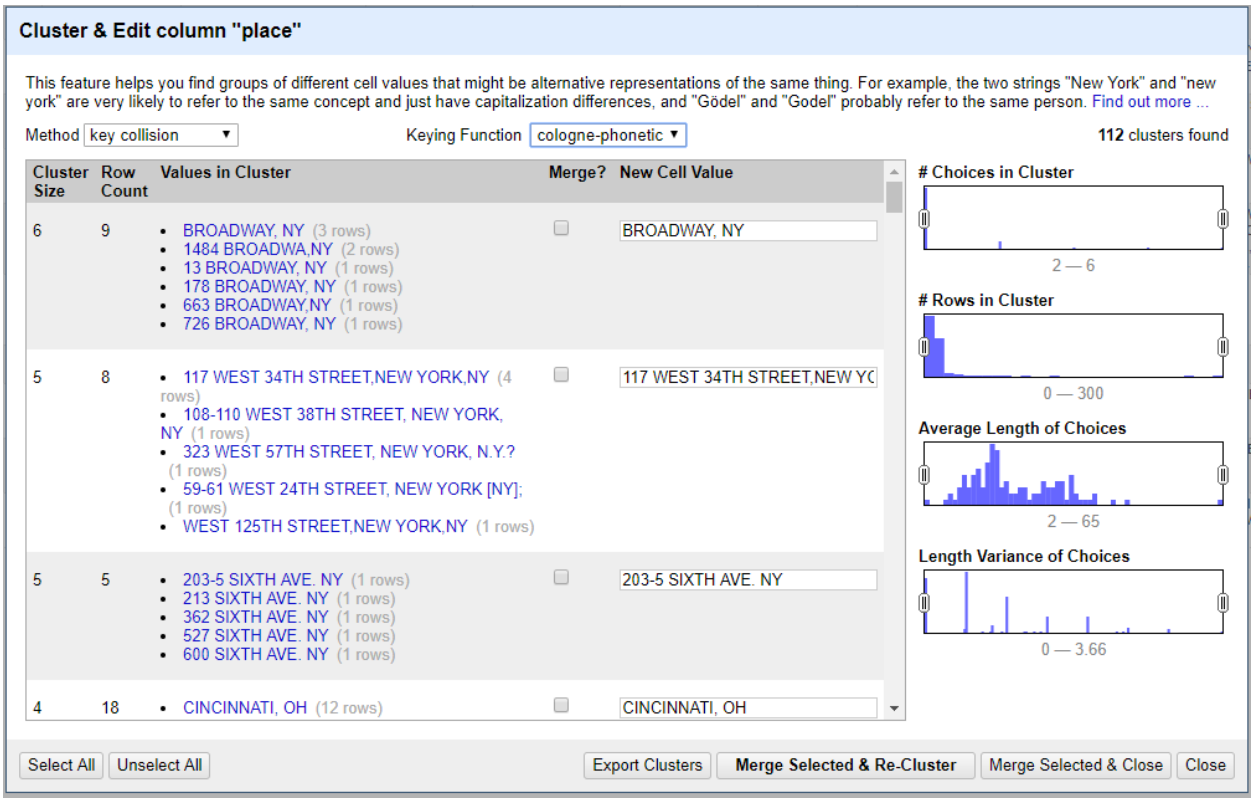
   with "PLANTERS HOTEL, ST.LOUIS, MO"



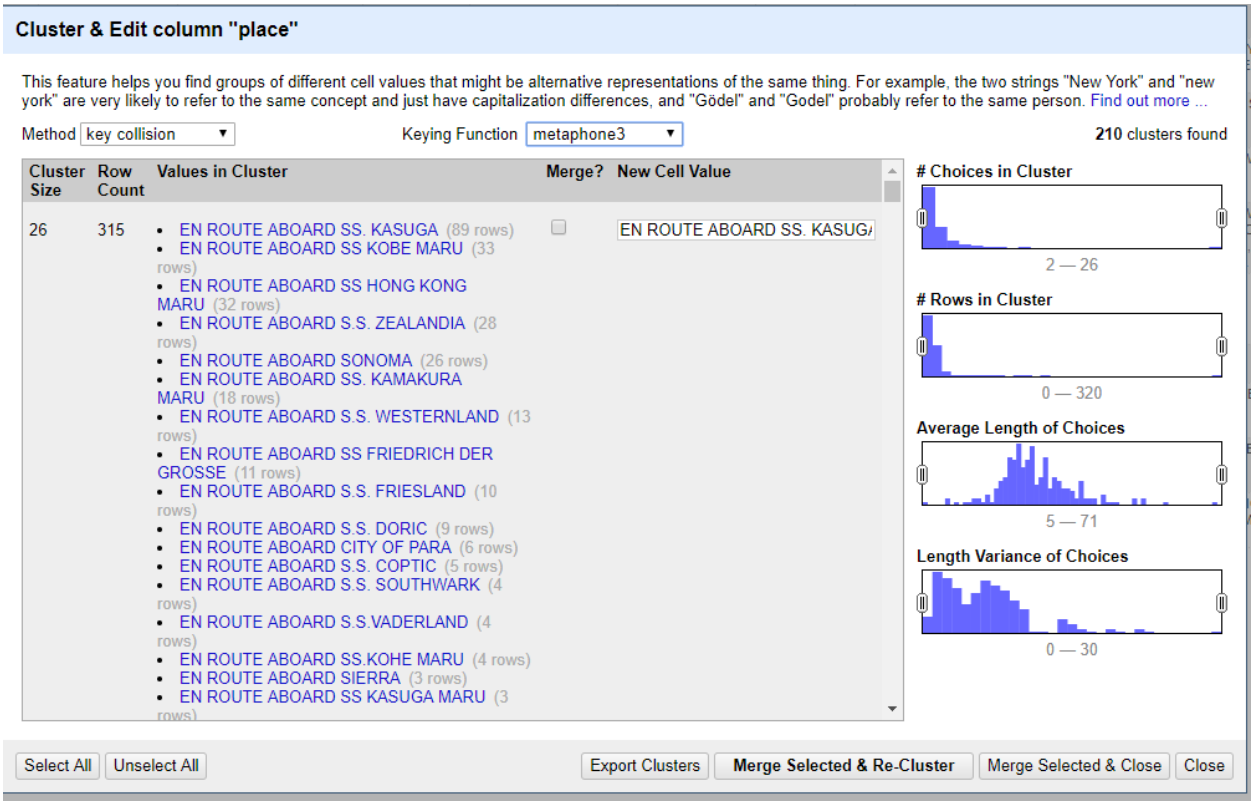
3. Clustering operation with keying function as “colgne-phonetic”. For example, this would replace the following values (using some common-sense and logic to make sure that the values made sense for the scenario):

   "SUMMERVILLE, S.C.",
   "SOMMERVILLE, S.C.",
   "SUMERVILLE,S.C."

   with "SUMMERVILLE, SC"



4. Trim leading and trailing space, collapse consecutive whitespaces and convert all to UPPERCASE.
5. Replace special characters with this transformation: value.replace(/[%@#!\?\\\(\)\[\]]+/,"").replace('*','').replace('"','')
6. We did not choose the clustering operation with keying function as “metaphone3” as it results in losing the existing organization breakdown per state or per county.



3.2.5 Transformations on "physical_description" column

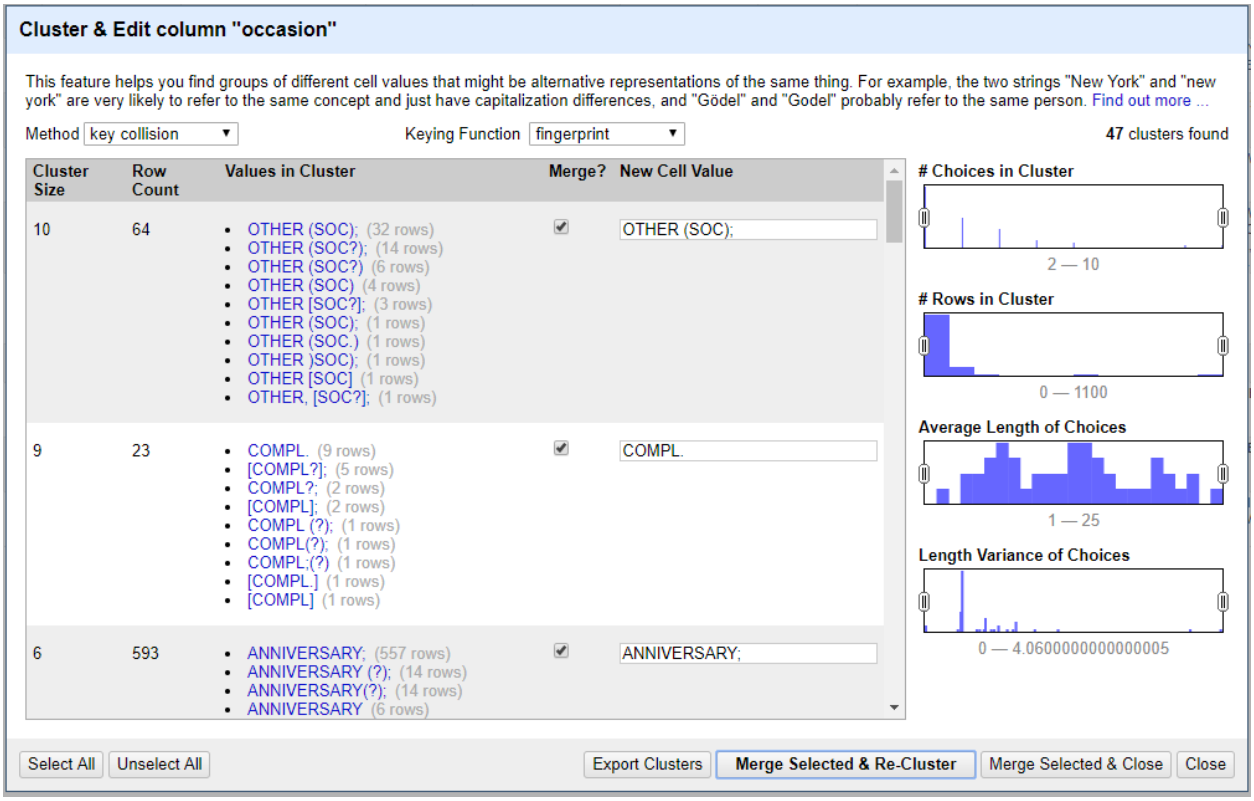
1. Trim leading and trailing space, collapse consecutive whitespaces and convert all to UPPERCASE.

3.2.6 Transformations on "occasion" column

1. Clustering operation with keying function as "fingerprint". For example, this would replace the following values:

"ANNIVERSARY;",
"ANNIVERSARY (?);",
"ANNIVERSARY(?);",
"ANNIVERSARY",
"ANNIVERSARY.",
"ANNIVERSARY?"

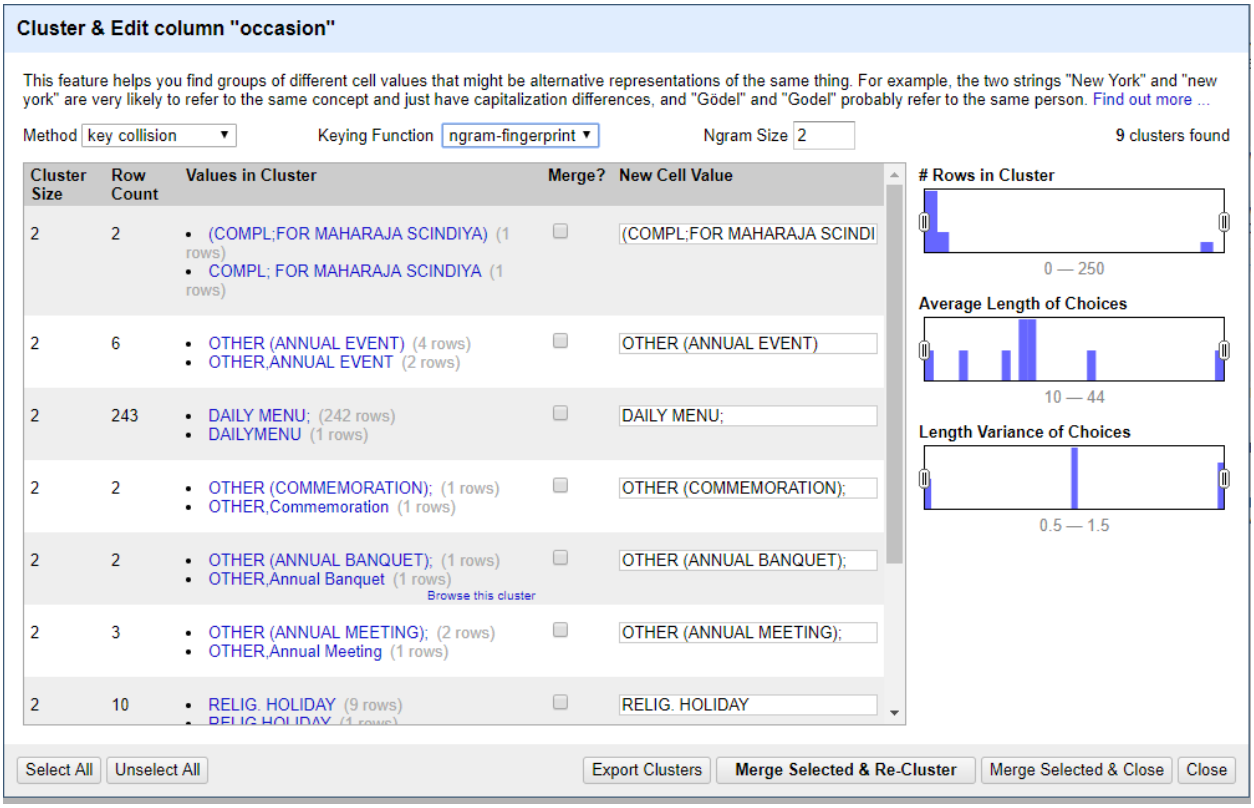
with "ANNIVERSARY"



2. Clustering operation with keying function as “ngram-fingerprint” with ngram size as 2. For example, this would replace the following values:

   "RELIG. HOLIDAY",
   "RELIG. HOLIDAY;",
   "RELIG HOLIDAY;"

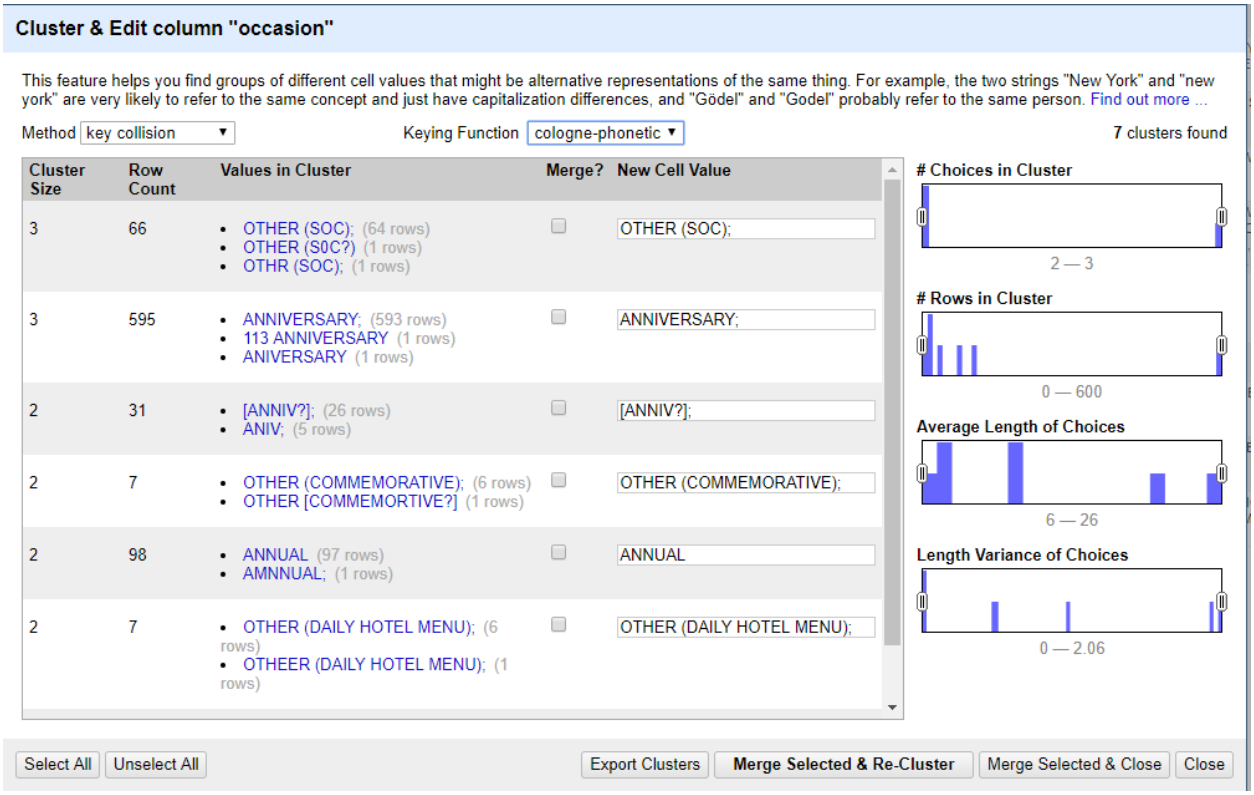
   with "RELIGIOUS HOLIDAY"



3. Clustering operation with keying function as “colgne-phonetic”. For example, this would replace the following values (using some common-sense and logic to make sure that the values made sense for the scenario):

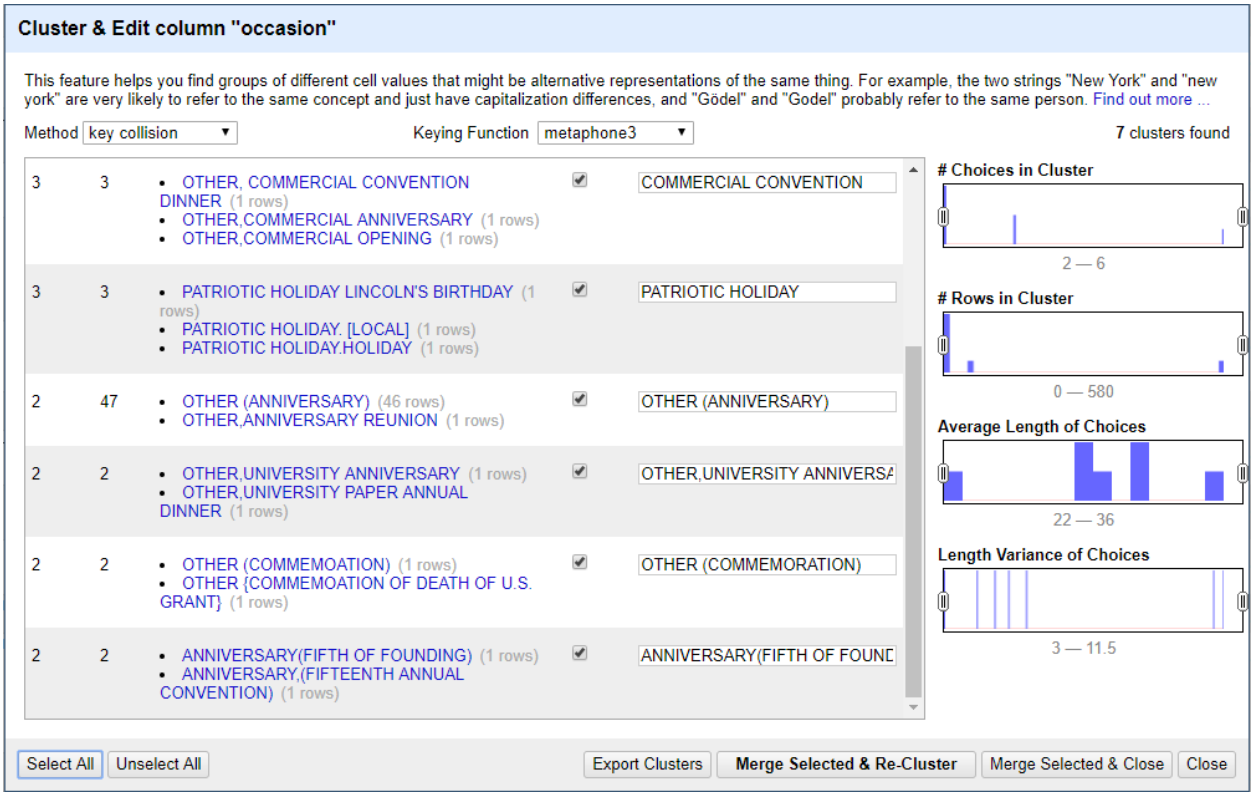
   "OTHER (COMMEMORATIVE)",
   "OTHER [COMMEMORTIVE?]"

   with "OTHER (COMMEMORATIVE)"



4. Trim leading and trailing space, collapse consecutive whitespaces and convert all to UPPERCASE.
5. Replace special characters "[,],(,),*,%," with "".

6. Clustering operation with keying function as “metaphone3”



3.2.7 Transformation on “notes” column

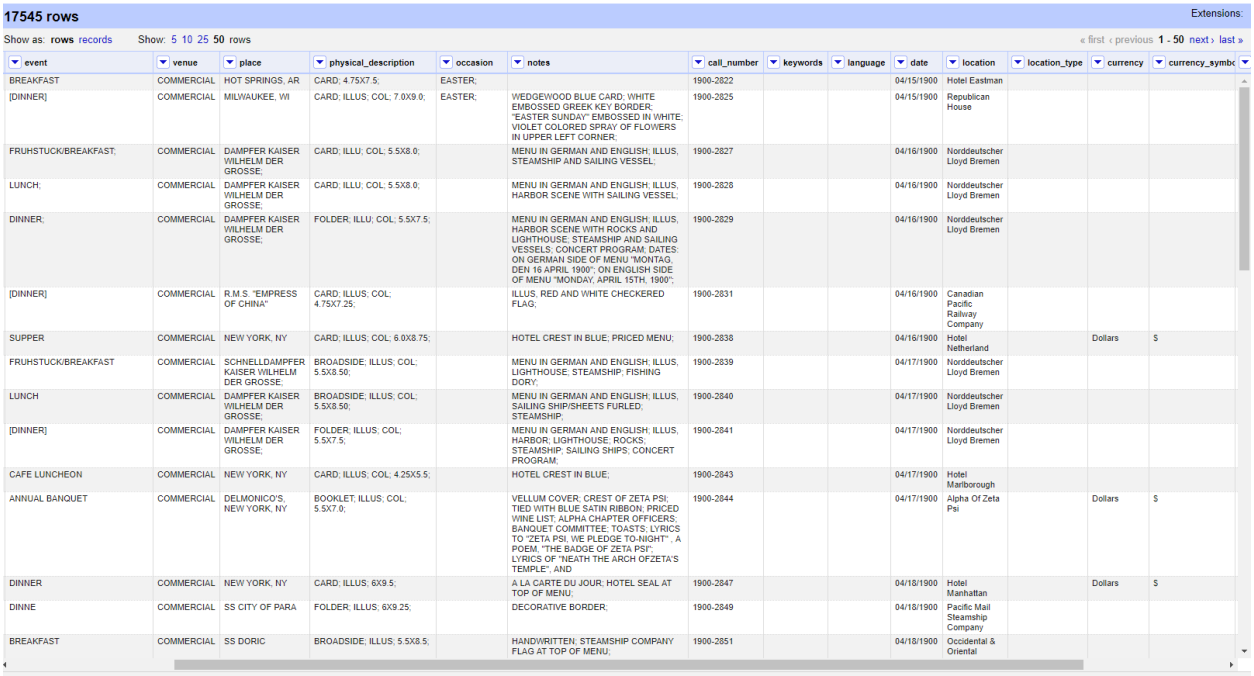
1. Trim leading and trailing space, collapse consecutive whitespaces and convert all to UPPERCASE.
2. Replace special characters "[,],(,),*,%," with "".

3.2.8 Transformation on “call_number” column

1. Trim leading and trailing space, collapse consecutive whitespaces and convert all to UPPERCASE.

3.2.9 Transformation on “date” column

1. Convert column type to “date”.
2. Use transformation “value.toString("MM/dd/yyyy")” to keep MM/DD/YYYY standard on the date column.
3. Convert column type to “text”.



17545 rows

Extensions:

Show as: **rows** records Show: 5 10 25 **50** rows « first ‹ previous **1 - 50** next › last »

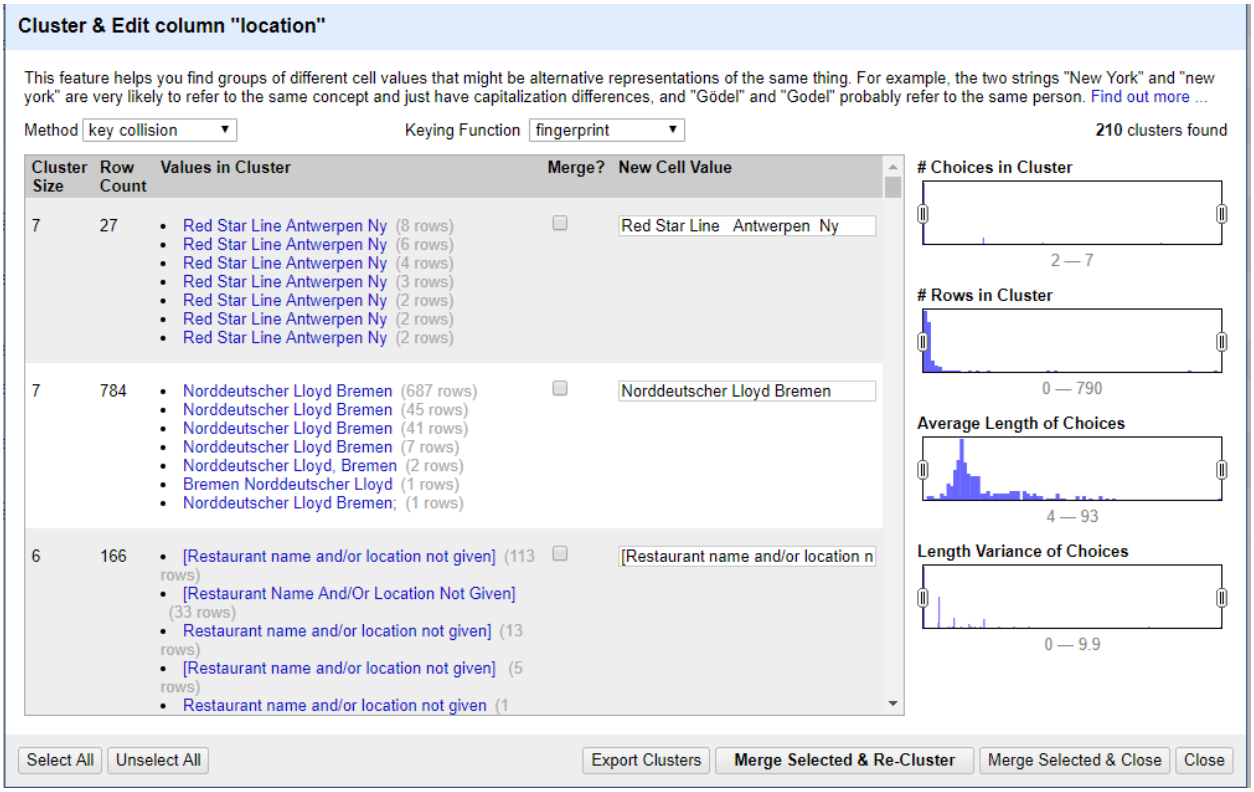
event	venue	place	physical_description	occasion	notes	call_number	keywords	language	date	location	location_type	currency	currency_symbc
BREAKFAST	COMMERCIAL	HOT SPRINGS, AR	CARD; 4.75X7.5;	EASTER;		1900-2822			04/15/1900	Hotel Eastman			
[DINNER]	COMMERCIAL	MILWAUKEE, WI	CARD; ILLUS; COL; 7.0X9.0;	EASTER;	WEDGEWOOD BLUE CARD; WHITE EMBOSSED GREEK KEY BORDER; "EASTER SUNDAY" EMBOSSED IN WHITE; VIOLET COLORED SPRAY OF FLOWERS IN UPPER LEFT CORNER;	1900-2825			04/15/1900	Republican House			
FRUHSTUCK/BREAKFAST;	COMMERCIAL	DAMPFER KAISER WILHELM DER GROSSE;	CARD; ILLU; COL; 5.5X8.0;		MENU IN GERMAN AND ENGLISH; ILLUS, STEAMSHIP AND SAILING VESSEL;	1900-2827			04/16/1900	Norddeutscher Lloyd Bremen			
LUNCH;	COMMERCIAL	DAMPFER KAISER WILHELM DER GROSSE;	CARD; ILLU; COL; 5.5X8.0;		MENU IN GERMAN AND ENGLISH; ILLUS, HARBOR SCENE WITH SAILING VESSEL;	1900-2828			04/16/1900	Norddeutscher Lloyd Bremen			
DINNER;	COMMERCIAL	DAMPFER KAISER WILHELM DER GROSSE;	FOLDER; ILLU; COL; 5.5X7.5;		MENU IN GERMAN AND ENGLISH; ILLUS, HARBOR SCENE WITH ROCKS AND LIGHTHOUSE; STEAMSHIP AND SAILING VESSELS; CONCERT PROGRAM; DATES: ON GERMAN SIDE OF MENU "MONTAG, DEN 16 APRIL 1900"; ON ENGLISH SIDE OF MENU "MONDAY, APRIL 15TH, 1900";	1900-2829			04/16/1900	Norddeutscher Lloyd Bremen			
[DINNER]	COMMERCIAL	R.M.S. "EMPRESS OF CHINA"	CARD; ILLUS; COL; 4.75X7.25;		ILLUS, RED AND WHITE CHECKERED FLAG;	1900-2831			04/16/1900	Canadian Pacific Railway Company			
SUPPER	COMMERCIAL	NEW YORK, NY	CARD; ILLUS; COL; 6.0X8.75;		HOTEL CREST IN BLUE; PRICED MENU;	1900-2838			04/16/1900	Hotel Netherland		Dollars	$
FRUHSTUCK/BREAKFAST	COMMERCIAL	SCHNELLDAMPFER KAISER WILHELM DER GROSSE;	BROADSIDE; ILLUS, COL; 5.5X8.50;		MENU IN GERMAN AND ENGLISH; ILLUS, LIGHTHOUSE, STEAMSHIP; FISHING DORY;	1900-2839			04/17/1900	Norddeutscher Lloyd Bremen			
LUNCH	COMMERCIAL	DAMPFER KAISER WILHELM DER GROSSE;	BROADSIDE; ILLUS; COL; 5.5X8.50;		MENU IN GERMAN AND ENGLISH; ILLUS, SAILING SHIP/SHEETS FURLED; STEAMSHIP;	1900-2840			04/17/1900	Norddeutscher Lloyd Bremen			
[DINNER]	COMMERCIAL	DAMPFER KAISER WILHELM DER GROSSE;	FOLDER; ILLUS; COL; 5.5X7.5;		MENU IN GERMAN AND ENGLISH; ILLUS, HARBOR; LIGHTHOUSE; ROCKS; STEAMSHIP; SAILING SHIPS; CONCERT PROGRAM;	1900-2841			04/17/1900	Norddeutscher Lloyd Bremen			
CAFE LUNCHEON	COMMERCIAL	NEW YORK, NY	CARD; ILLUS; COL; 4.25X5.5;		HOTEL CREST IN BLUE;	1900-2843			04/17/1900	Hotel Marlborough			
ANNUAL BANQUET	COMMERCIAL	DELMONICO'S, NEW YORK, NY	BOOKLET; ILLUS; COL; 5.5X7.0;		VELLUM COVER; CREST OF ZETA PSI; TIED WITH BLUE SATIN RIBBON; PRICED WINE LIST; ALPHA CHAPTER OFFICERS; BANQUET COMMITTEE; TOASTS; LYRICS TO "ZETA PSI, WE PLEDGE TO-NIGHT" , A POEM, "THE BADGE OF ZETA PSI"; LYRICS OF "NEATH THE ARCH OFZETA'S TEMPLE", AND	1900-2844			04/17/1900	Alpha Of Zeta Psi		Dollars	$
DINNER	COMMERCIAL	NEW YORK, NY	CARD; ILLUS; 6X9.5;		A LA CARTE DU JOUR; HOTEL SEAL AT TOP OF MENU;	1900-2847			04/18/1900	Hotel Manhattan		Dollars	$
DINNE	COMMERCIAL	SS CITY OF PARA	FOLDER; ILLUS; 6X9.25;		DECORATIVE BORDER;	1900-2849			04/18/1900	Pacific Mail Steamship Company			
BREAKFAST	COMMERCIAL	SS DORIC	BROADSIDE; ILLUS; 5.5X8.5;		HANDWRITTEN; STEAMSHIP COMPANY FLAG AT TOP OF MENU;	1900-2851			04/18/1900	Occidental & Oriental			

3.2.10 Transformation on “location” column

1. Clustering operation with keying function as “fingerprint”. For example, this would replace the following values:

"Red Star Line Antwerpen Ny",
"Red Star Line Antwerpen Ny",
"Red Star Line Antwerpen Ny",
"Red Star Line Antwerpen Ny",
"Red Star Line Antwerpen Ny",
"Red Star Line Antwerpen Ny",
"Red Star Line Antwerpen Ny"

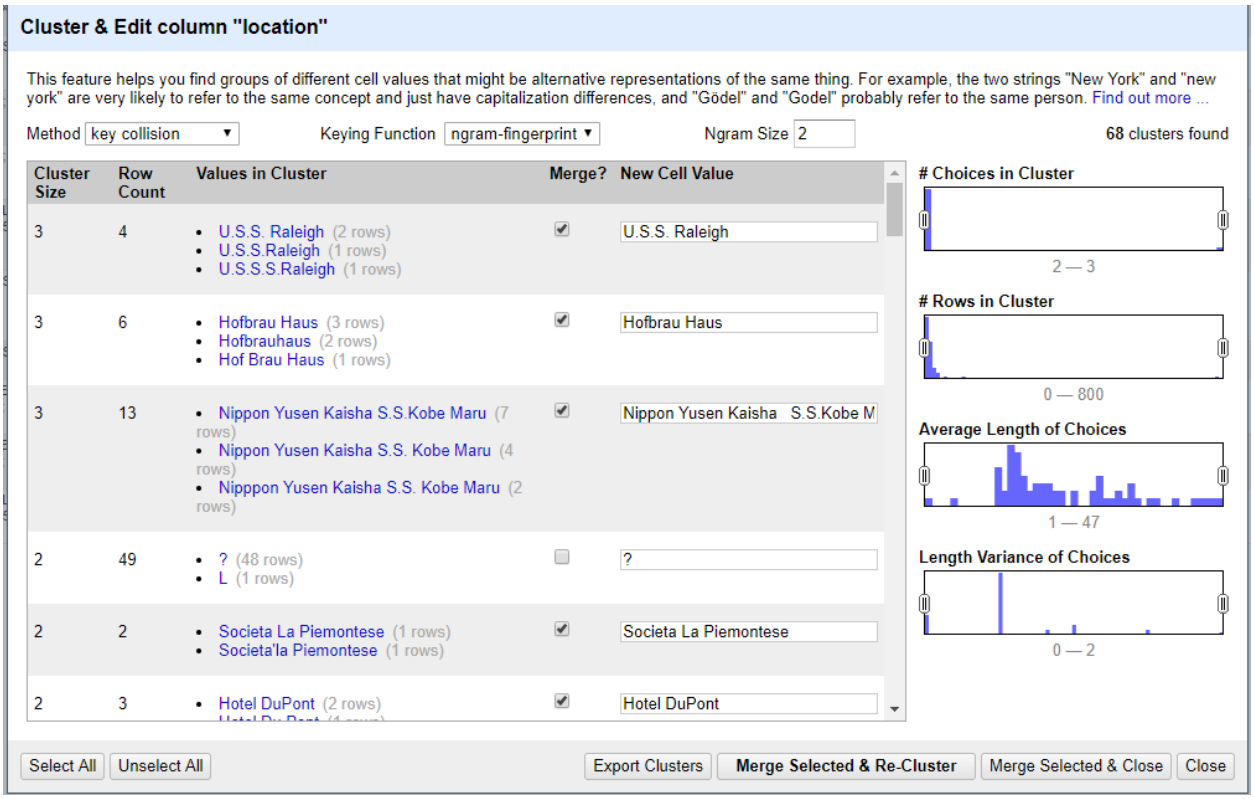
with "Red Star Line Antwerpen Ny"



4. Clustering operation with keying function as “ngram-fingerprint” with ngram size as 2. For example, this would replace the following values:

"U.S.S. Raleigh",
"U.S.S.Raleigh",
"U.S.S.S.Raleigh"

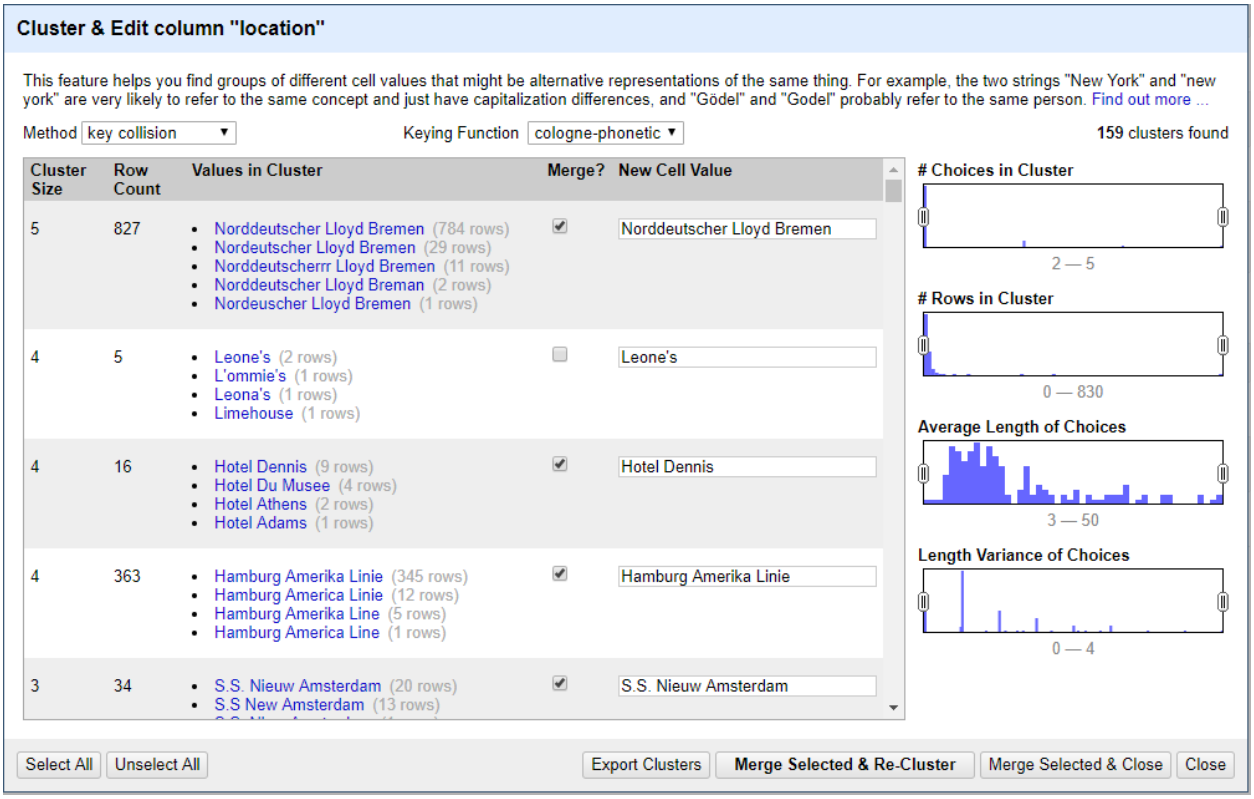
with "U.S.S. Raleigh"



5. Clustering operation with keying function as “colgne-phonetic”. For example, this would replace the following values (using some common-sense and logic to make sure that the values made sense for the scenario):

   "Norddeutscher Lloyd Bremen",
   "Nordeutscher Lloyd Bremen",
   "Norddeutscherrr Lloyd Bremen",
   "Norddeutscher Lloyd Breman",
   "Nordeuscher Lloyd Bremen"

   with "Norddeutscher Lloyd Bremen"



6. Trim leading and trailing space, collapse consecutive whitespaces and convert all to UPPERCASE.
7. Replace special characters "[,],(,),*,%," with "".

3.2.11 Columns Deleted or Merged

1. The following columns were eliminated as they had 100% missing values:
    a. name
    b. language
    c. keyword
    d. location_type
    e. currency_symbol
2. Location column was a duplicate of sponsor column and was empty in several cases. We merged sponsor and location with this transformation:

    if(isNull(cells["sponsor"].value), cells["sponsor"].value, cells["location"].value)

3. Deleted location column
4. Renamed sponsor to name.

3.2.12 Columns Untouched

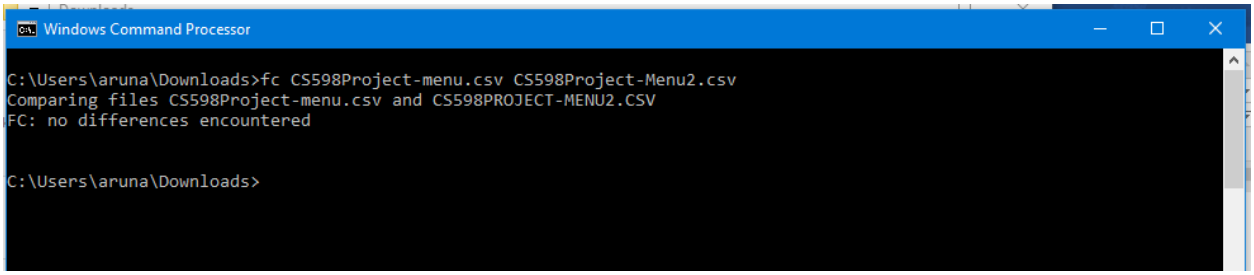
1. The following columns were not edited at all as they were deemed to contain valid values:

a. id
b. currency
c. status
d. page_count
e. dish_count

Export CS598Project-menu.csv from OpenRefine

3.2.13 Operation History & Provenance

The entire operation history has been exported from OpenRefine as a json file (**menuOperationHis.json**, attached with project files). To check the provenance, we created a new project in OpenRefine with the original menu.csv and applied this menuOperationHis.json. This automatically executes all the transformations on all the columns. We exported the new csv and compared to the CS598Project-menu.csv that was exported previously using the windows file comparator and the two csv files were identical proving that the transformation are 100% replicable.

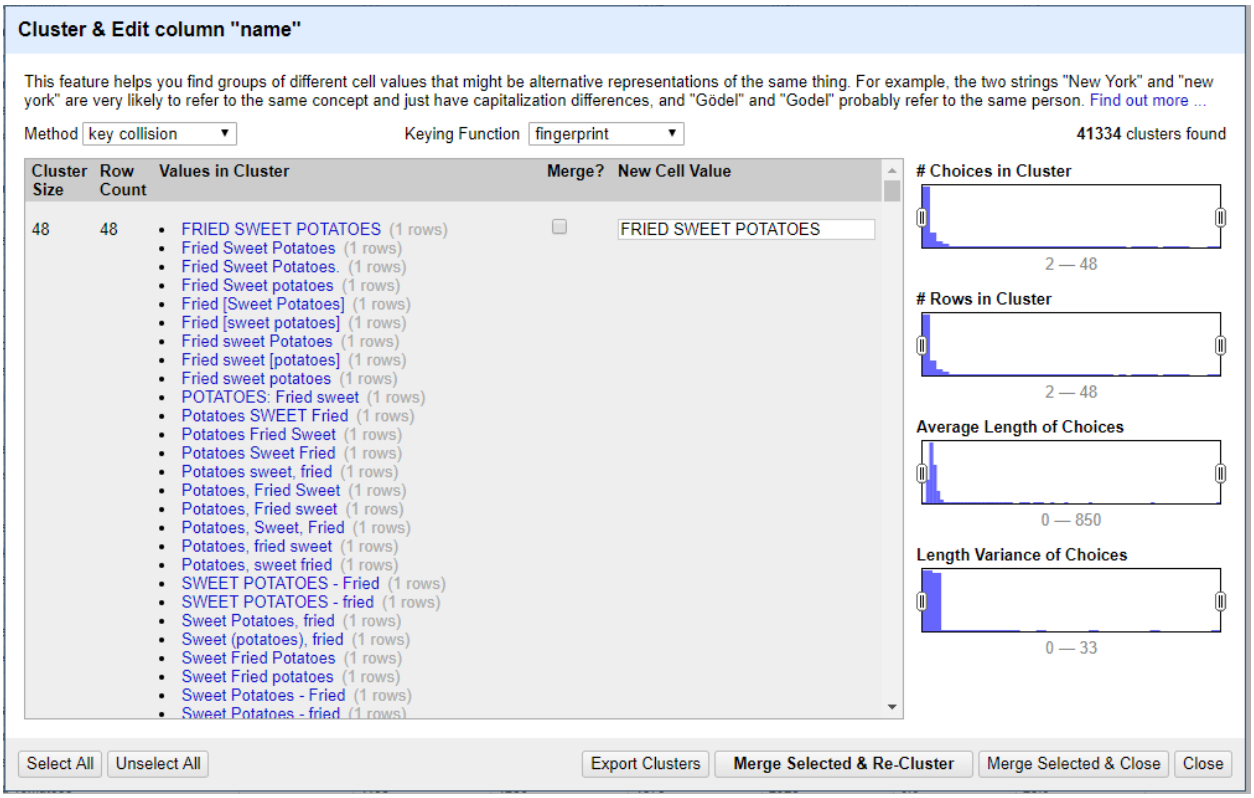


3.3 Changes in Dish CSV

3.3.1 Transformation on "name" column

1. Clustering operation with keying function as "fingerprint". For example, this would replace the following values:

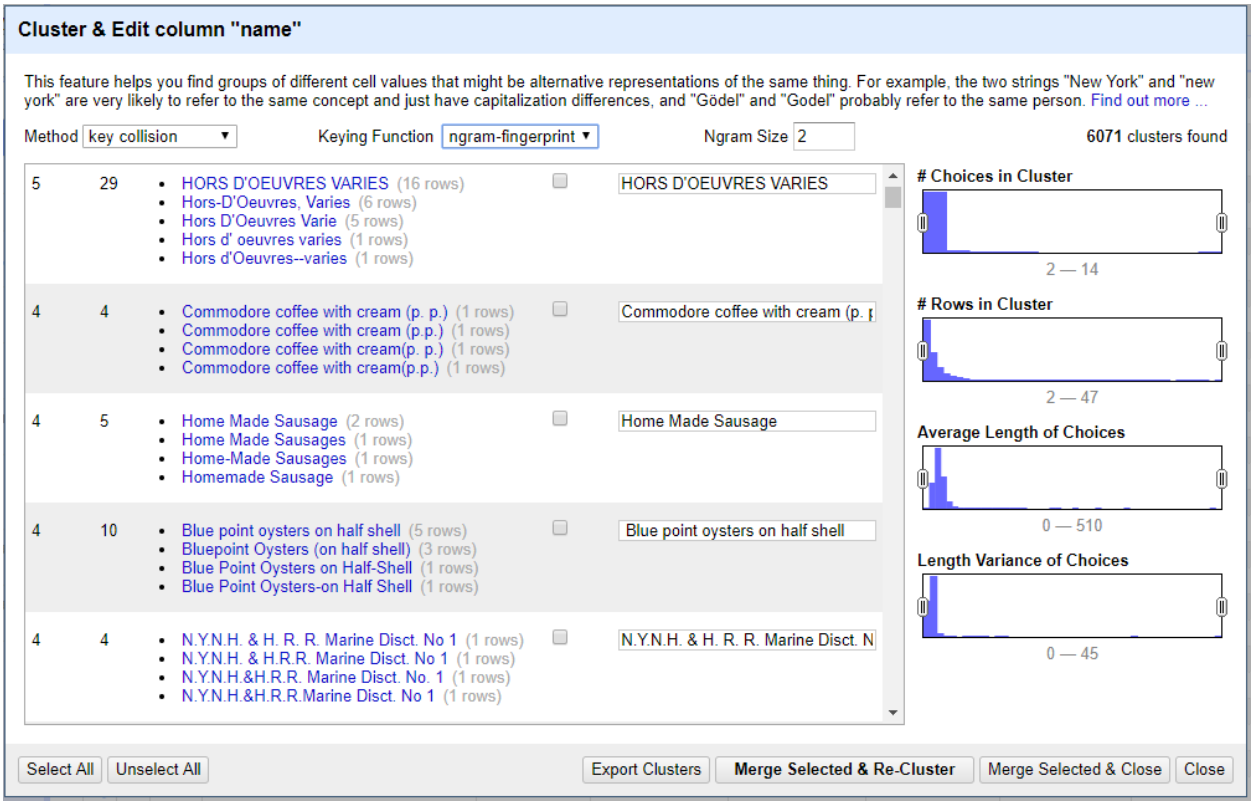
" French Fried Potatoes",
" Potatoes French Fried",
"FRENCH FRIED POTATOES",
"French fried potatoes",
"French Fried (potatoes)",
"French Fried Potatoes",
"French Fried Potatoes.",
"French Fried [Potatoes]",
"French Fried potatoes",
With " French Fried Potatoes"



2. Clustering operation with keying function as “ngram-fingerprint” with ngram size as 2. For example, this would replace the following values:

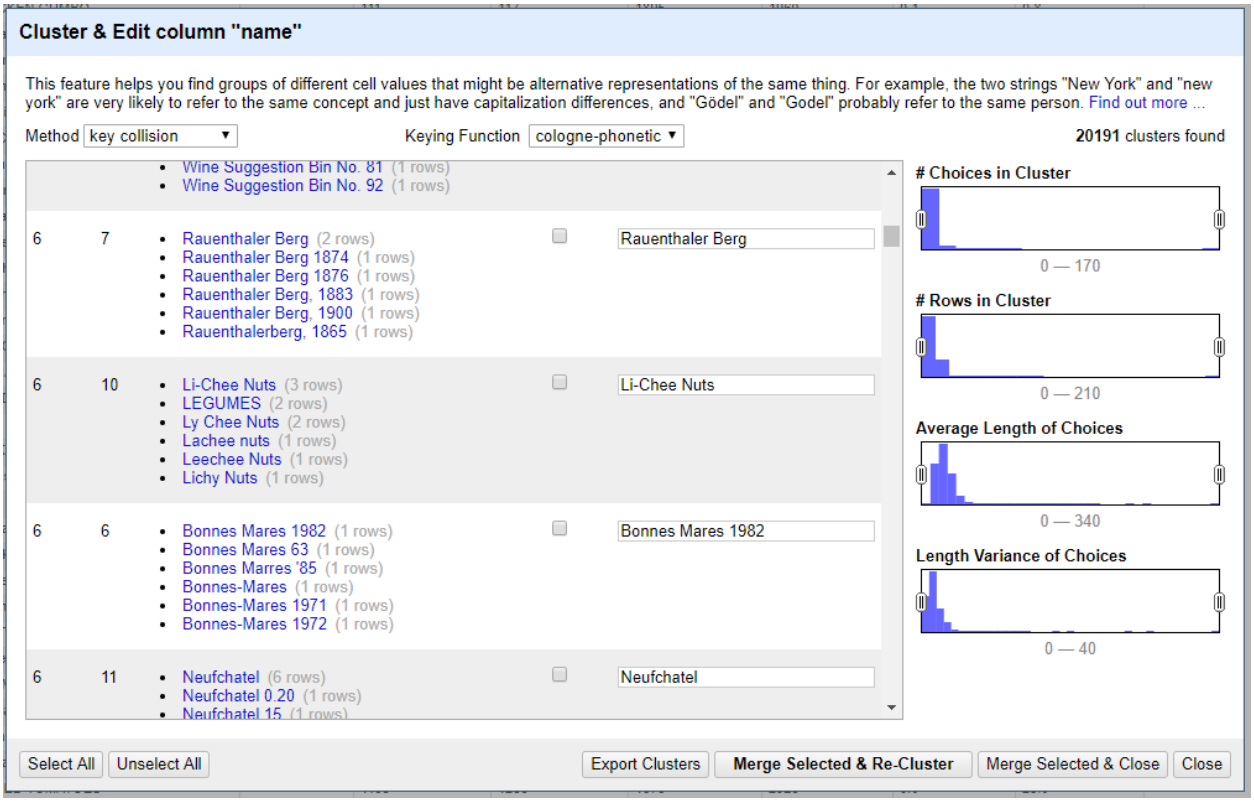
"Cream Potatoes",
"Cream potatoes",
"Cream, Potatoes",
"Potatoes & Cream"

with "Cream Potatoes"

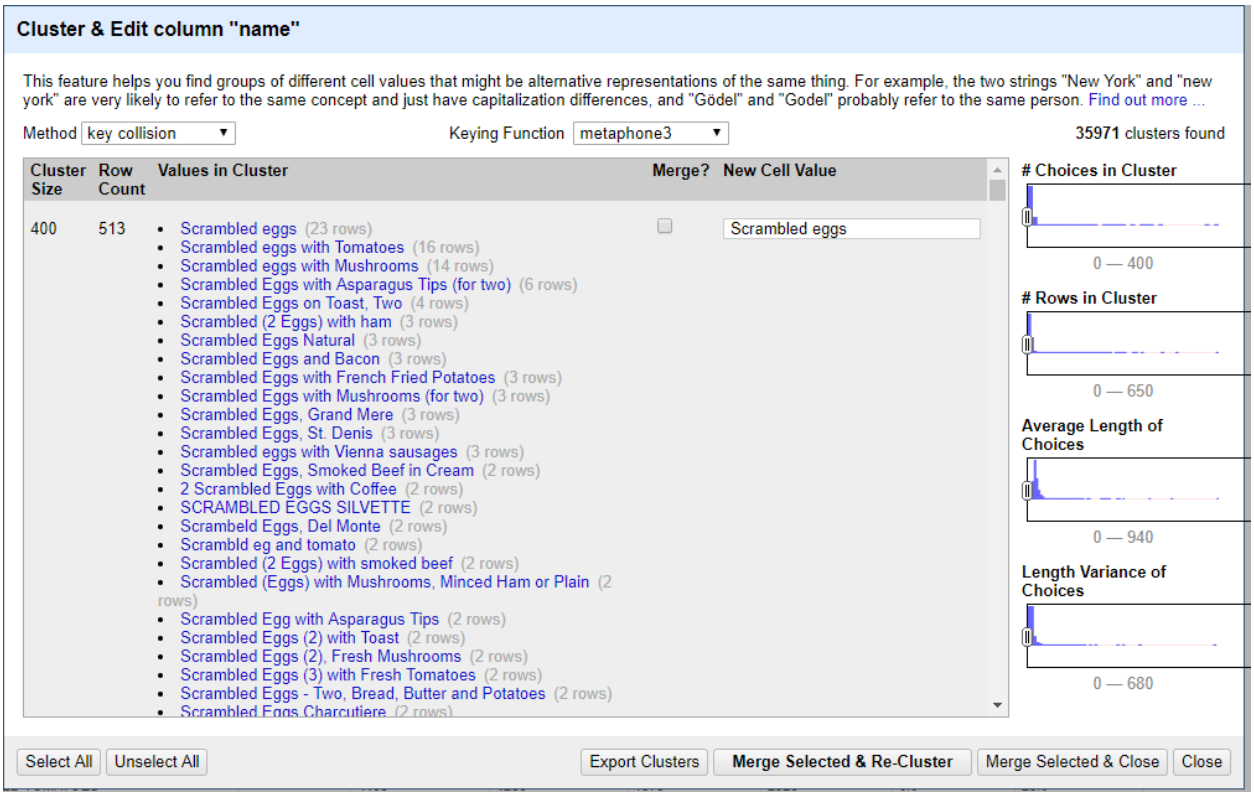


3. Clustering operation with keying function as “colgne-phonetic”. For example, this would replace the following values (a common sense based selective replacement was done):

"Coffee (Pot)",
"Coffe Pot",
"Coffee (pot - 1)",
"Cofee pot",
"Coffee (pot) .30",
"Coffee, Pota"
with "Coffee (Pot)"



4. We chose **not** to perform the clustering operation with keying function as “metaphone3” because the operation was resulting in a loss of feature selection in a dish. The screen shot shows an example where “Scrambled Eggs with Tomatoes” should not be listed as the same as “Scrambled Eggs with Mushrooms”



5. Replace special characters with this transformation:
   value.replace(/[%@#!\?\\\(\)\[\]]+/,"").replace('*','').replace('"','')
6. Trim leading and trailing space, collapse consecutive whitespaces and convert all to UPPERCASE.

3.3.2 Transformations on "first_appeared" column

1. Some of the columns in the date were entered as "2928" etc. These were fixed using the GREL transformation:
   value.replace(/^29/,"19").replace(/^28/,"18")
2. Set column to a number and value to 0 if null or blank with the following GREL transformation:
   if(or(isBlank(value,isNull(value)), 0.0, value
3. Convert the column type to number.

3.3.3 Transformations on "last_appeared" column

1. Some of the columns in the date were entered as "2928" etc. These were fixed using the GREL transformation:
   value.replace(/^29/,"19").replace(/^28/,"18")
2. Set column to a number and value to 0 if null or blank with the following GREL transformation:

if(or(isBlank(value,isNull(value)), 0.0, value

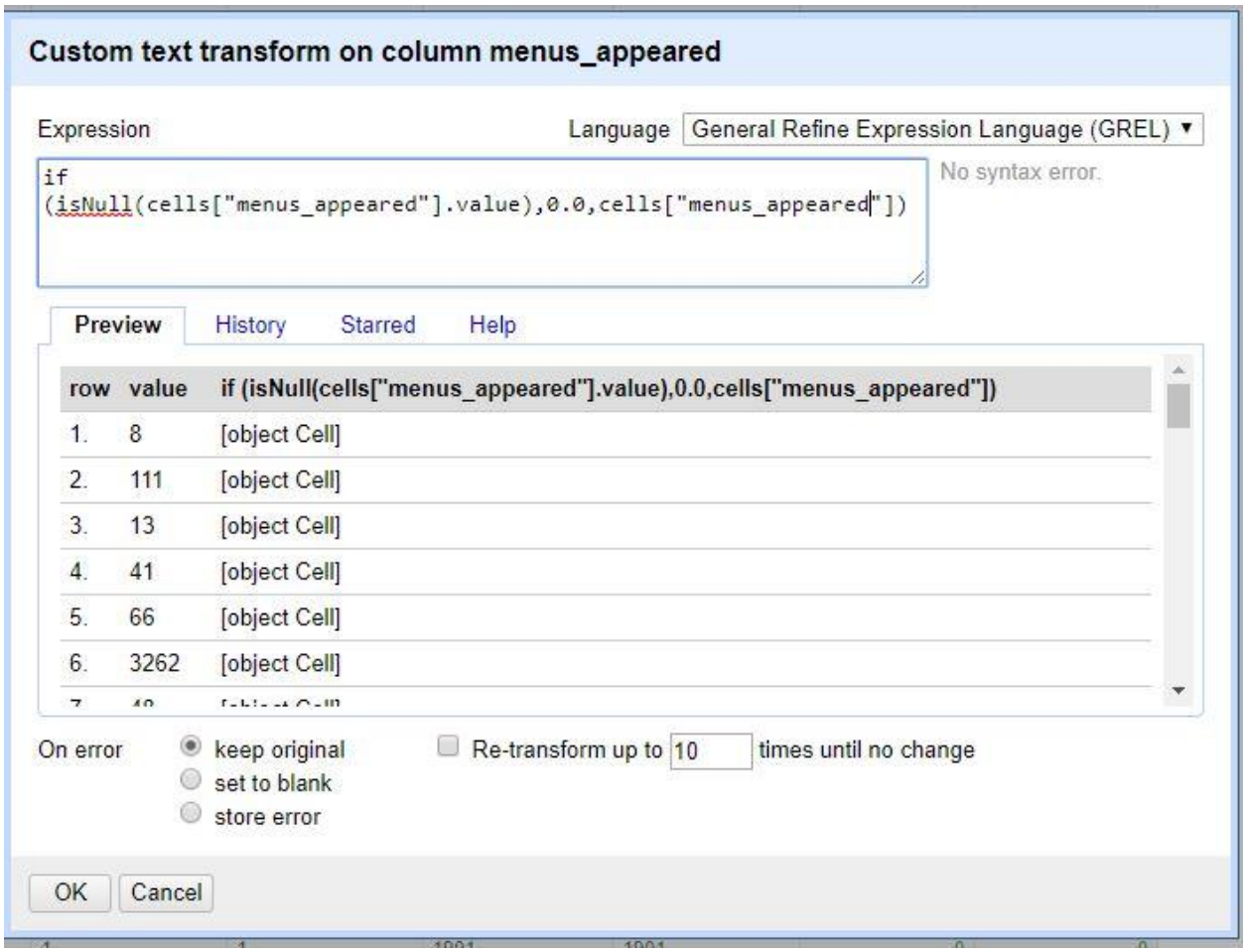
3. Convert the column type to number.

3.3.4 Transformations on "menus_appeared" column

1. Set column to a number and value to 0 if null or blank with the following GREL transformation:

   if(or(isBlank(value,isNull(value)), 0.0, value

2. Convert the column type to number.

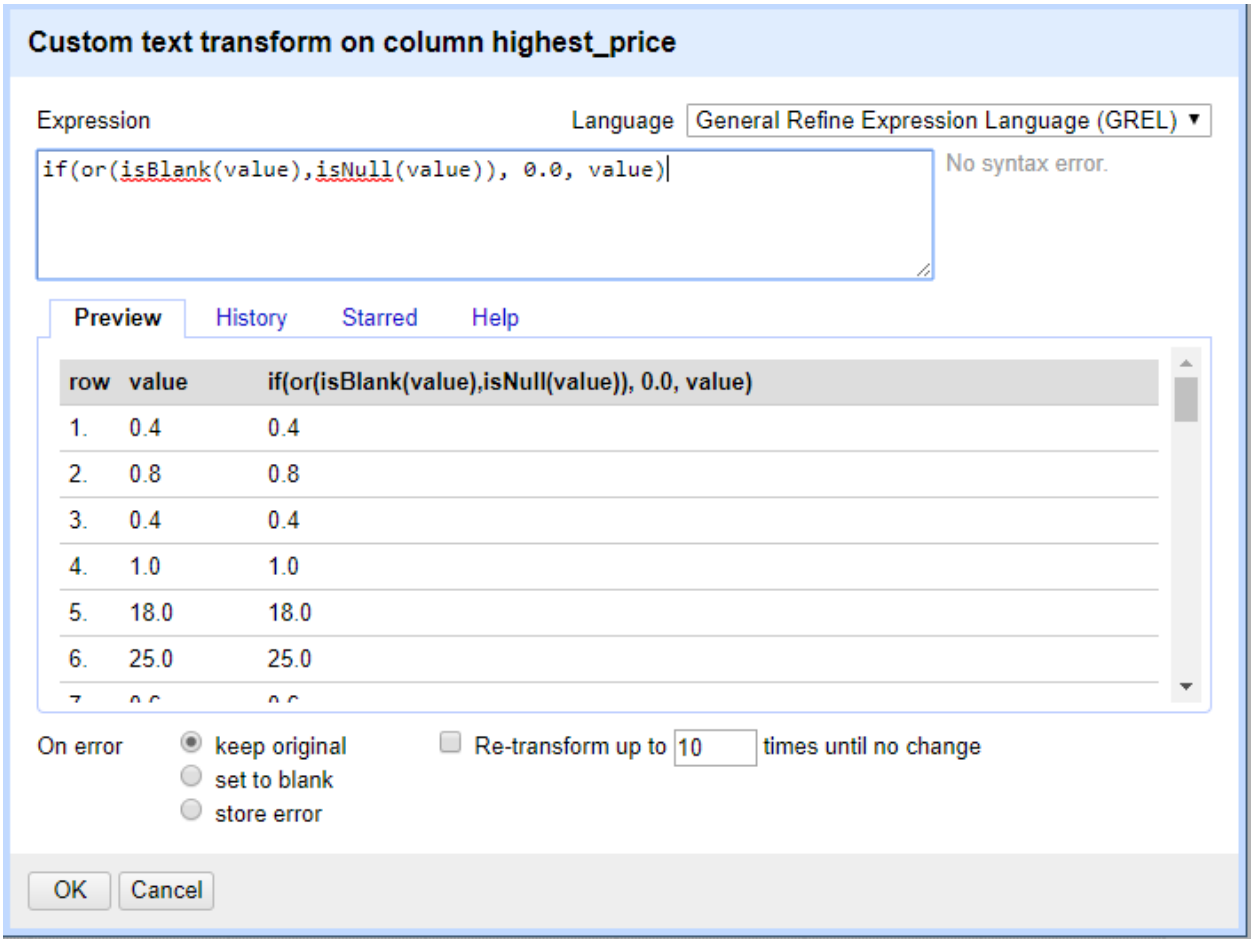


3.3.5 Transformations on "lowest_price" column

1. Set column to a number and value to 0 if null or blank with the following GREL transformation:

   if(or(isBlank(value,isNull(value)), 0.0, value

2. Convert the column type to number.



3.3.6 Transformations on "highest_price" column

1. Set column to a number and value to 0 if null or blank with the following GREL transformation:

   if(or(isBlank(value,isNull(value)), 0.0, value

2. Convert the column type to number.

3.3.7 Columns Deleted or Merged

The following columns were eliminated as they had 100% missing values:

- name

3.3.8 Columns left Untouched

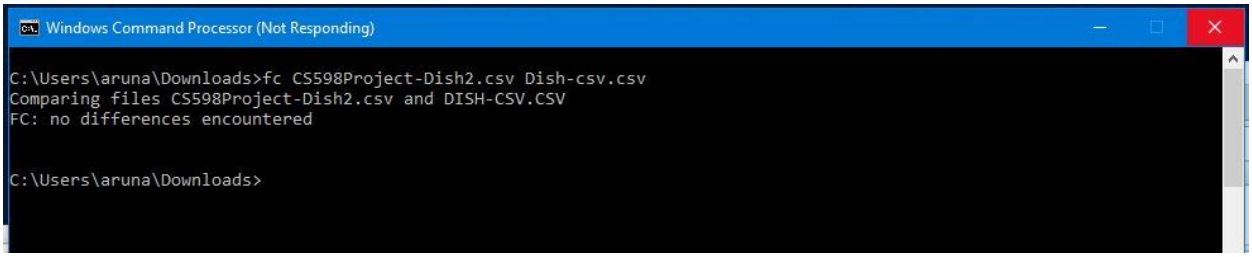
The following columns left untouched:

- id

Export the CS598Project-dish.csv from OpenRefine.

3.3.9 Operation History & Provenance

The operation history has been exported from OpenRefine as a json file (**dishOperationHis.json**, attached with project files). To check the provenance, we created a new project in OpenRefine with the original dish.csv and applied this dishOperationHis.json. This automatically performs all the transformations on all the columns. We exported the new csv and compared to the CS598Project-dish.csv using the windows file comparator and the two csv files were identical proving that the transformation are 100% replicable.



3.4 Changes in MenuPage CSV

Created a project in OpenRefine for the MenuPage.csv data and determined that all numeric columns need no further cleaning. The csv contained only numeric fields.

1. Set the page_number to 0 if null or blank using the GREL transformation:
   if(or(isBlank(value), isNull(value)), 0.0, value)
   and converted column type to number.
2. The image_id column for the most contained entirely numbers (almost 67K rows) except for 30 which contained text followed by numerals. We performed a GREL transformation to remove the alpabets and store it as a number:

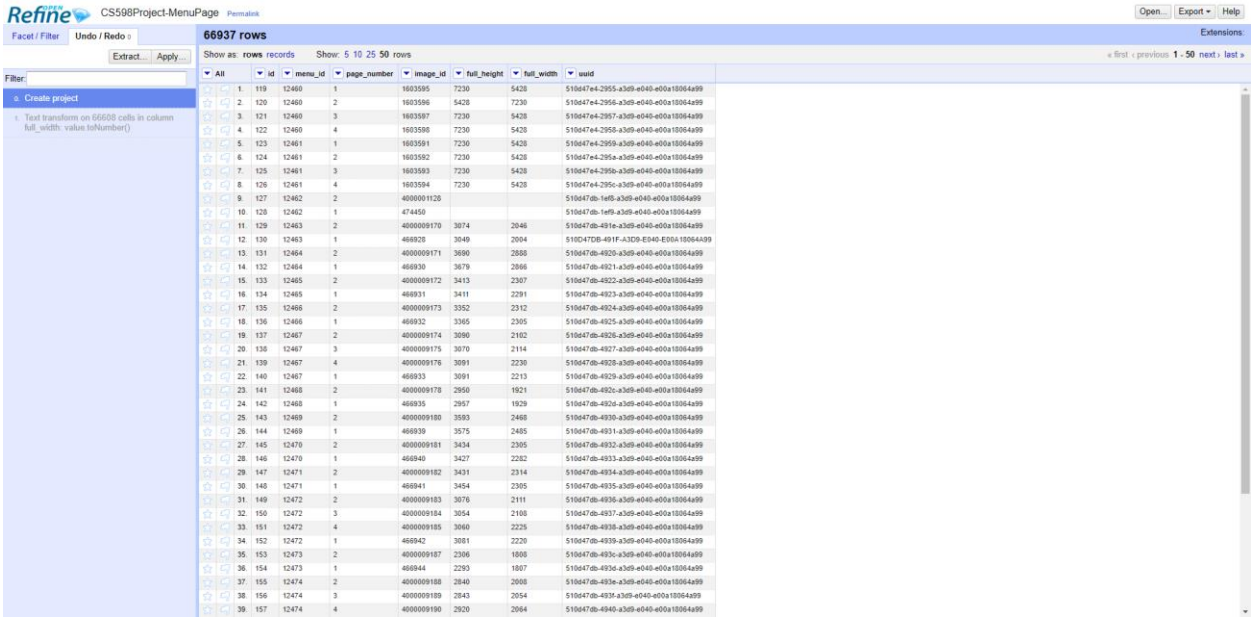
   value.replace('psnypl_rbk_','').replace('ps_rbk_','')

   It is possible that these 30 entries were a typo and should contain numbers. Since this column is likely to be a foreign key to a table which the OCR scanned images of the menu (not included in this dataset). **We did not remove these 30 entries because these items are referenced in the MenuPage table via the menu_page_id and removing them will result in an integrity constraint violation.**
3. Null and blank entries in full_height and full_width columns where set to 0 using the GREL transformation:
   if(or(isBlank(value),isNull(value)), 0, value)
4. The remaining columns were left untouched.
5. CS598Project-MenuPage.csv was exported from the project.

3.4.1 Operation History & Provenance

The operation history has been exported from OpenRefine as a json file (**menuPageOperationHis.json**, attached with project files). To check the provenance, we created

a new project in OpenRefine with the original dish.csv and applied menuPageOperationHis.json. This automatically performs all the transformations on all the columns. We exported the new csv and compared to the CS598Project-menuPage.csv using the windows file comparator and the two csv files were identical proving that the transformation are 100% replicable.



3.5 Changes in MenuItem CSV

Created a project in OpenRefine for the MenuItem.csv data and determined that all numeric columns need no further cleaning.

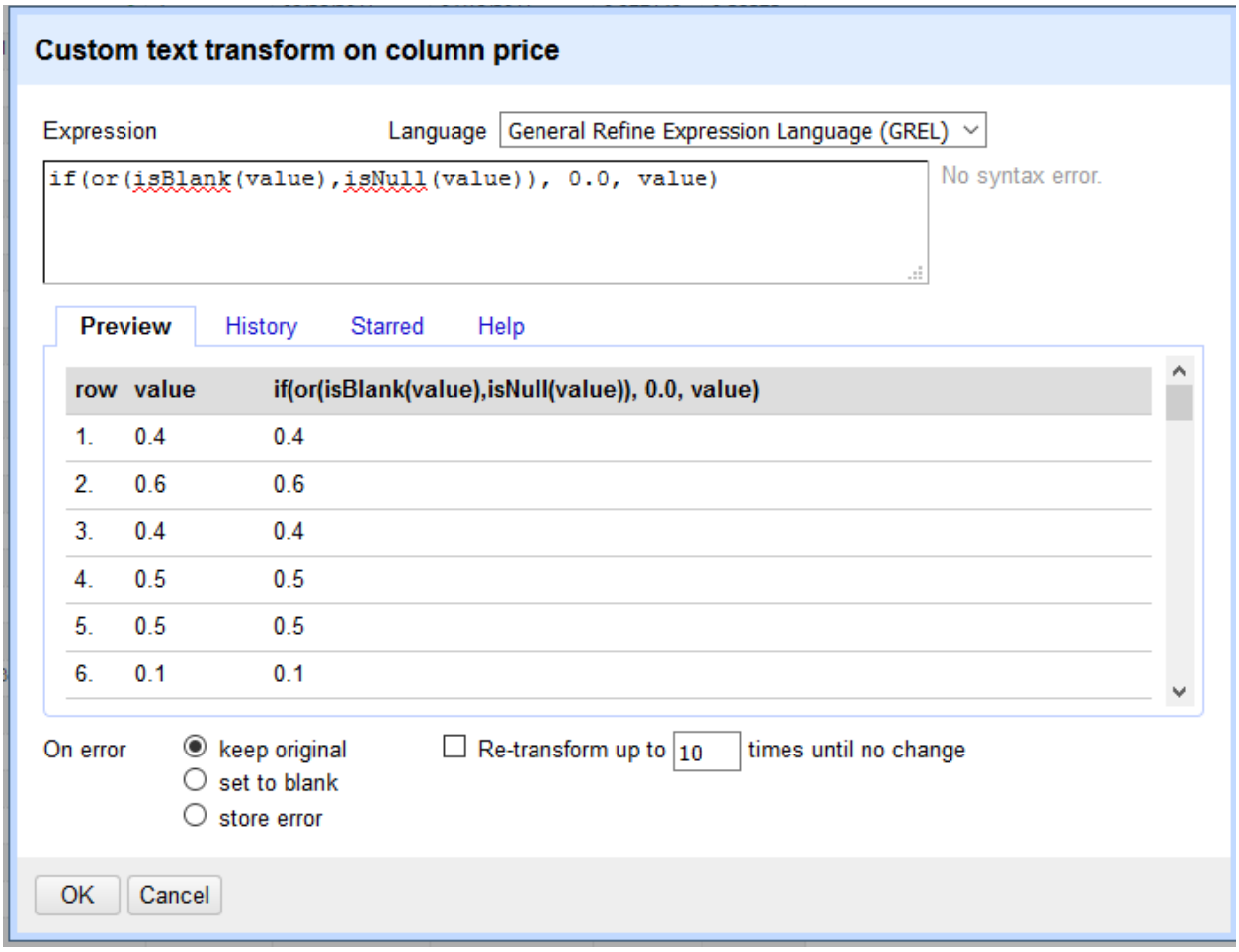
1. Used a combination of python and GREL transformations for the created_at and updated_at fields to convert them to the standard MM/DD/YYYY format:

   Python Transformation: a=value.split(" ")
   return a[0]

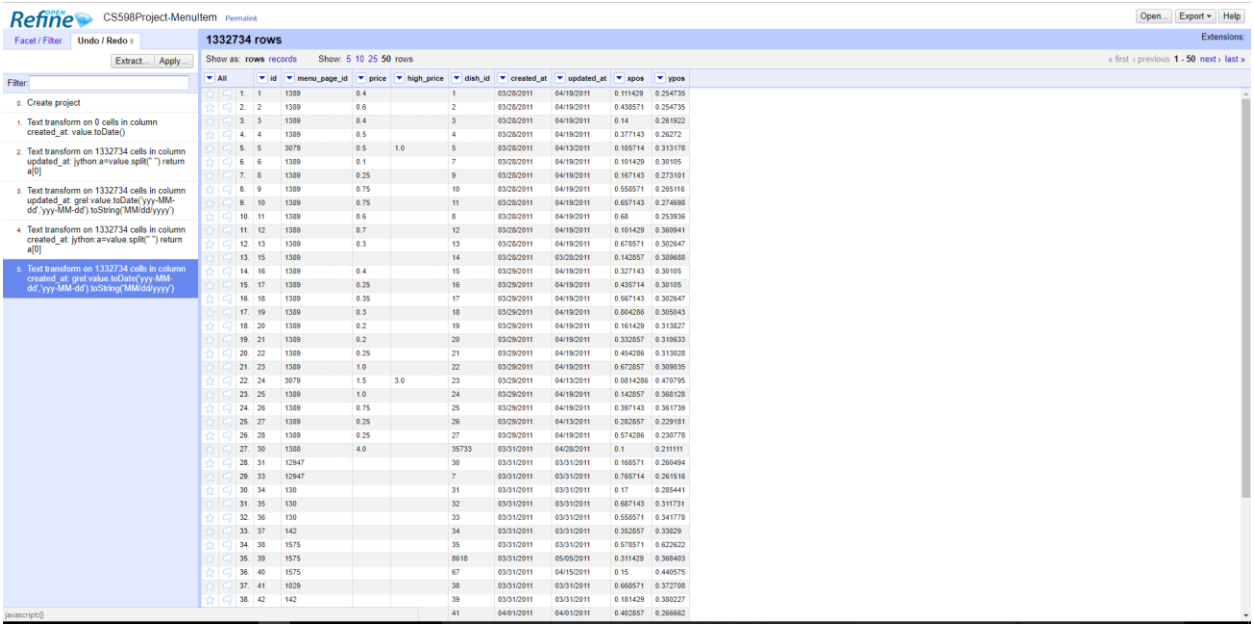
   GREL Transformation:
   value.toDate('yyy-MM-dd','yyy-MM-dd').toString('MM/dd/yyyy')

2. Set blank and null entries in the high_price column to 0 and changed the column type to number

3. Set blank and null entries in the price column to 0 and changed the column type to number



4. **dish_id column has blank entries we did not fill in the missing values as this is an identified integrity constraint violation.**
5. Final export of CS598Project-menuitem.csv from OpenRefine.



3.5.1 Operation History & Provenance

The operation history has been exported from OpenRefine as a json file (**menuItemOperationHis.json**, attached with project files). To check the provenance, we created a new project in OpenRefine with the original menuitem.csv and applied this menuItemOperationHis.json. This automatically performs all the transformations on all the columns. We exported the new csv and compared it to the CS598Project-menuitem.csv using the windows file comparator and the two csv files were identical proving that the transformation are 100% replicable.

3.6 Columns after cleanup with OpenRefine

3.6.1 Menu csv

- id – unique identified for each menu
- name – name of the menu with sponsor details
- event – event for which the menu was prepared
- venue – venue where the event was held
- place – location/address of event
- physical_description – characteristics of menu card; type of paper, size
- occasion – special occasion for which the menu was prepared.
- notes – details on menu, language used, interesting characteristics of menu
- call_number – 4-year digit followed by a hyphen by a 4-digit id
- date – date when the menu was created in the format MM/DD/YYYY
- currency – currency used the menu item pricing
- status – indicates whether the menu entry is complete
- page_count – number of pages on the menu
- dish_count – number of dishes on the menu

3.6.2 Dish csv

- id – unique identified for each dish
- name – name of the dish
- menus_appeared – menu on which the dish has appeared on
- times_appeared – number of times the dish has appeared on menus
- first_appeared – 4-digit year when the menu first appeared
- last_appeared – 4-digit year when the menu last appeared
- lowest_price – lowest prices for the dish
- highest_price – highest prices for the dish

3.6.3 MenuItem csv

- id – unique identifier for the menu item
- menu_page_id – page id on which menu appears (foreign key to menu page id)
- price – price of menu item
- high_price – highest price of men item
- dish_id – dish id for the menu item (foreign key to dish id)
- created_at – date on which the menu item was created
- updated_at – date on which the menu item was updated
- xpos – x coordinate of where the menu item appears on the scanned image
- ypos – y coordinate of where the menu item appears on the scanned image

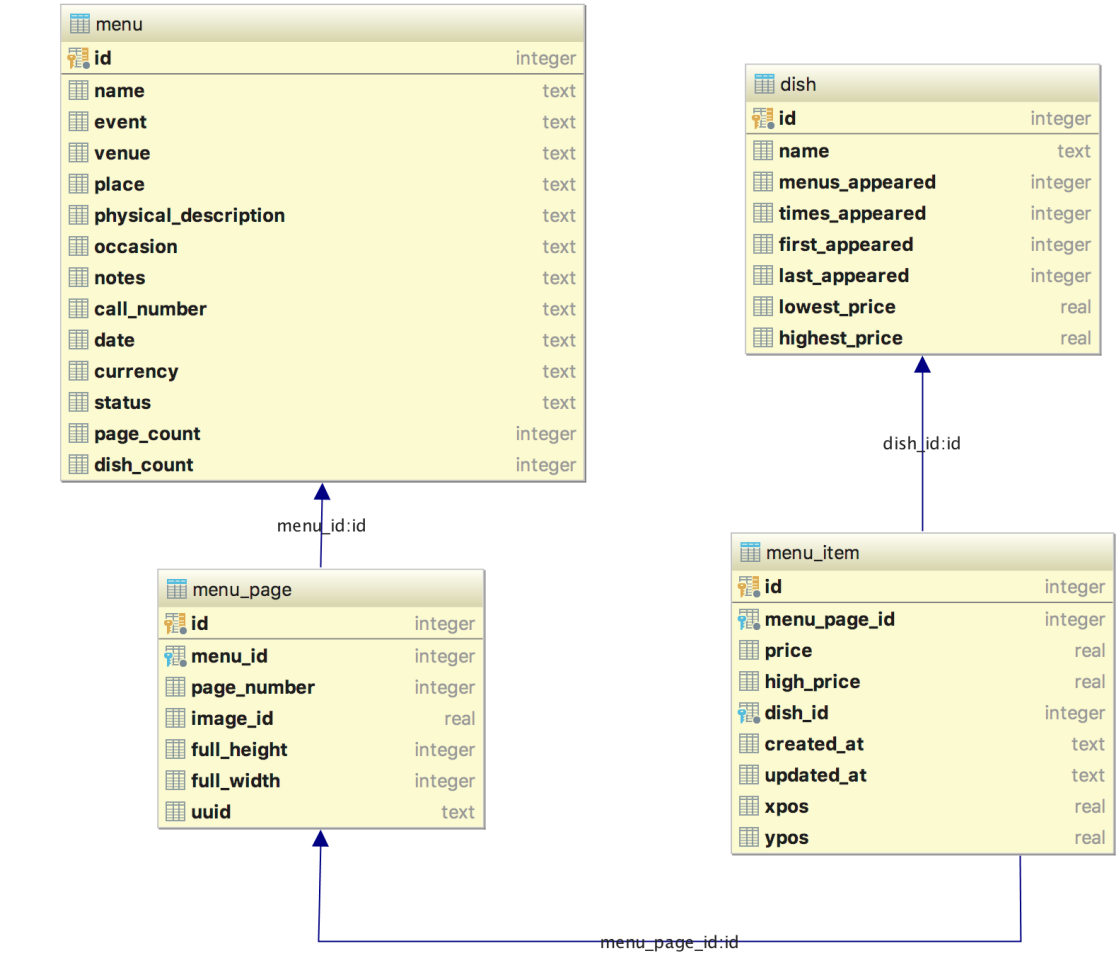
3.6.4 MenuPage csv

- id – unique identifier for a page on each menu
- menu_id – identifier to menu (foreign key to menu:id)
- page_number – page number for each menu
- image_id – id for OCR scanned image of the page
- full_height – height of the original page
- full_width – width of the original page
- uuid – unique alpha numeric identifier to the image

4 SQLite

We were successful in importing data into SQLite. We have reproduced the schema in the section 4.1. Section 4.2 contains data definition language used to create tables and constraints are introduced for the all four tables.

4.1 Schema



Files

4.1.1 Enable foreign key constraint support in SQLite

Before importing the data into "SQLite", we had to turn on the support for foreign key restrictions.

```
PRAGMA foreign_keys = ON;
```

4.2 Data Definition Language

Created the schema for all 4 tables with the foreign key relationship.

- The menu_page table has the **menu_id** as the foreign key of the menu table's primary key (id)
- The menu_item table has two foreign key relationship:
    - **menu_page_id** with menu_page table's primary key (id)
    - **dish_id** with dish table's primary key (id)

```
CREATE TABLE menu
(
   id INTEGER PRIMARY KEY NOT NULL,
   name TEXT,
   event TEXT,
   venue TEXT,
   place TEXT,
   physical_description TEXT,
   occasion TEXT,
   notes TEXT,
   call_number TEXT,
   date TEXT,
   currency TEXT,
   status TEXT,
   page_count INTEGER,
   dish_count INTEGER
)

CREATE TABLE menu_page
(
   id INTEGER PRIMARY KEY NOT NULL,
   menu_id INTEGER NOT NULL,
   page_number INTEGER,
   image_id REAL,
   full_height INTEGER,
   full_width INTEGER,
   uuid TEXT,
   CONSTRAINT menu_page_menu_id_fk FOREIGN KEY (menu_id) REFERENCES menu
(id) ON DELETE CASCADE ON UPDATE CASCADE
);
CREATE UNIQUE INDEX menu_page_id_uindex ON menu_page (id);

CREATE TABLE dish
(
   id INTEGER PRIMARY KEY NOT NULL,
   name TEXT,
   menus_appeared INTEGER,
   times_appeared INTEGER,
   first_appeared INTEGER,
```

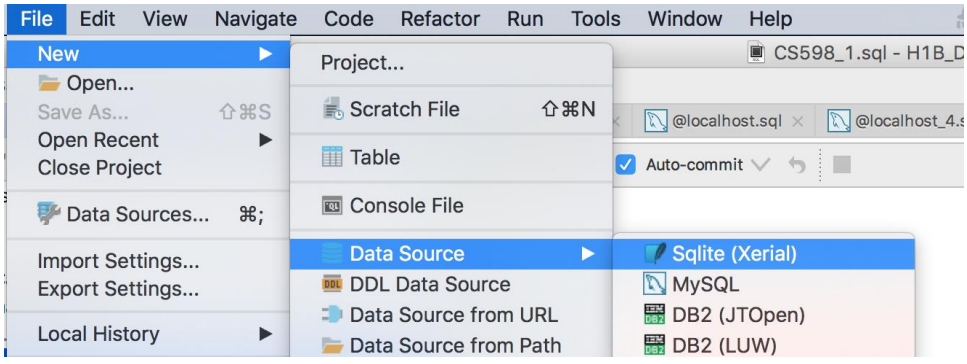
```
    last_appeared INTEGER,
    lowest_price REAL,
    highest_price REAL
);
CREATE UNIQUE INDEX dish_id_uindex ON dish (id);


CREATE TABLE menu_item
(
    id INTEGER PRIMARY KEY NOT NULL,
    menu_page_id INTEGER NOT NULL,
    price REAL,
    high_price REAL,
    dish_id INTEGER NOT NULL,
    created_at TEXT,
    updated_at TEXT,
    xpos REAL,
    ypos REAL,
    CONSTRAINT menu_item_menu_page_id_fk FOREIGN KEY (menu_page_id)
REFERENCES menu_page (id) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT menu_item_dish_id_fk FOREIGN KEY (dish_id) REFERENCES dish (id)
ON DELETE CASCADE ON UPDATE CASCADE
);
CREATE UNIQUE INDEX menu_item_id_uindex ON menu_item (id);
```

4.3 CSV Import

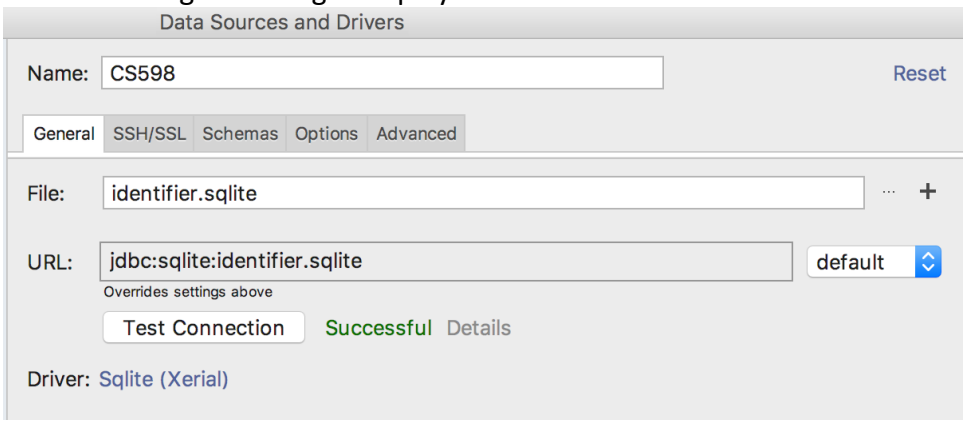
SQLite command line interface is not very robust to load complex CSV files. We used "DataGrip" to import the data. We created a data source and then imported the csv files.

4.3.1 Adding Data Source

- Click on File->New->Data Source->SQLite, as shown below.



- Then in the dialog enter the name as "CS598" and click on "Test Connection". The "Successful" message should get displayed.

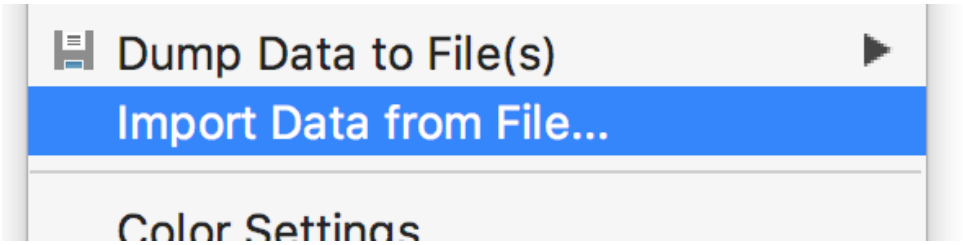


- Now click on apply to add the Data Source.

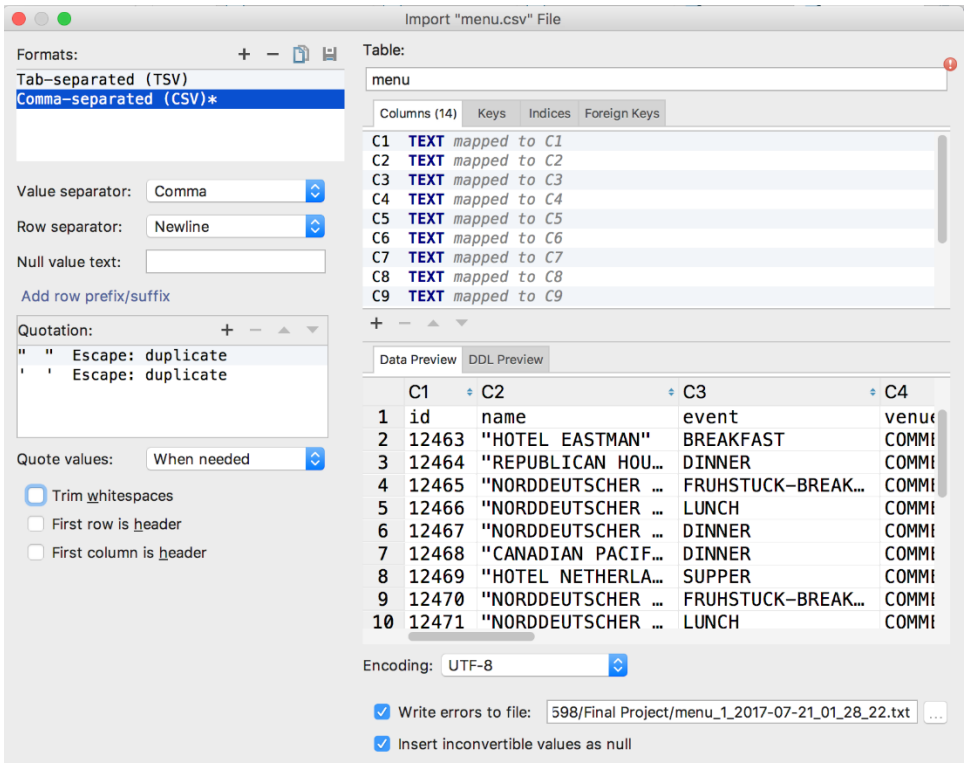
4.3.2 Import Data

4.3.2.1 Import menu.csv

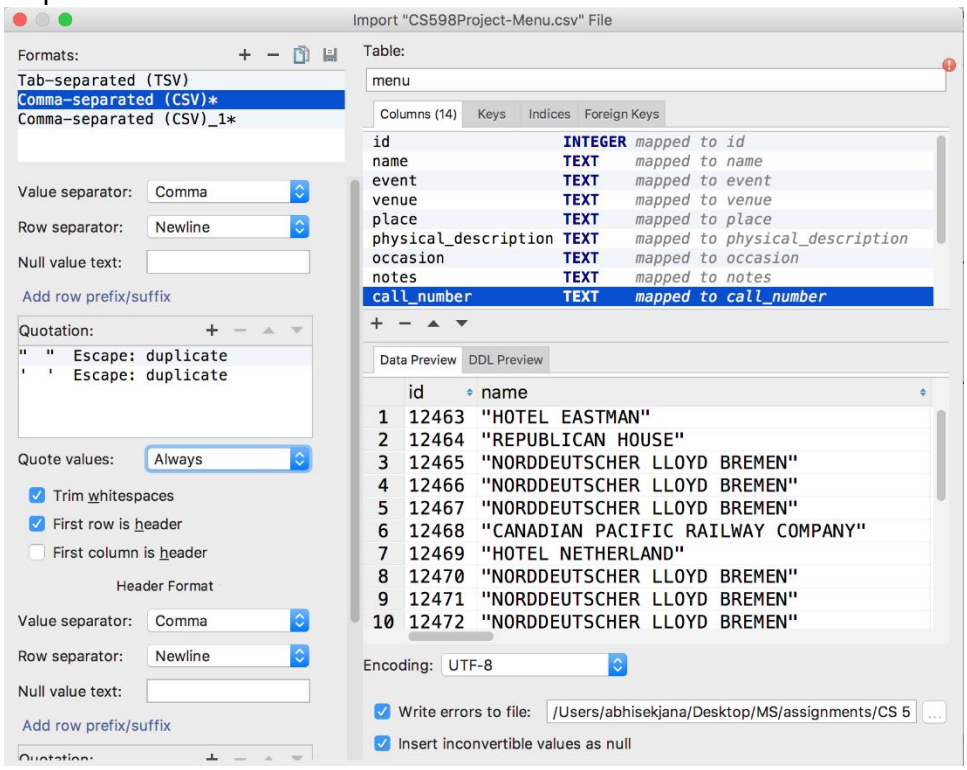
- Right click on the data source (CS598) and click on "Import Data from File..."



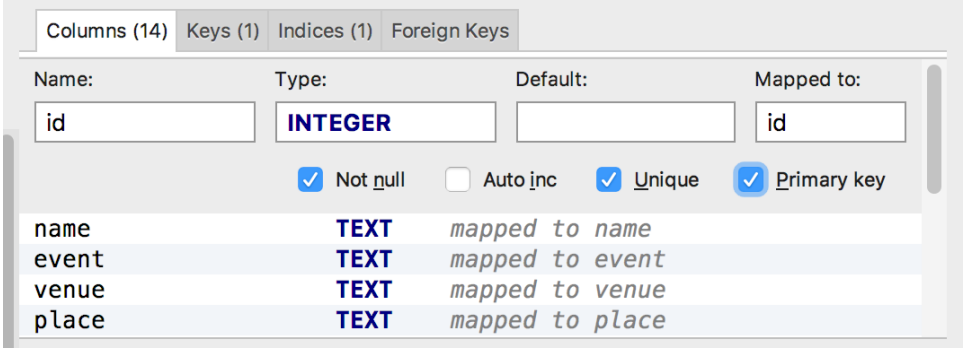
- Then select the menu.csv to load the file.
- Following window will be displayed.



- Select "Comma-separated (CSV)*" in the Format section, and check on "Trim whitespaces" and "First row is header"



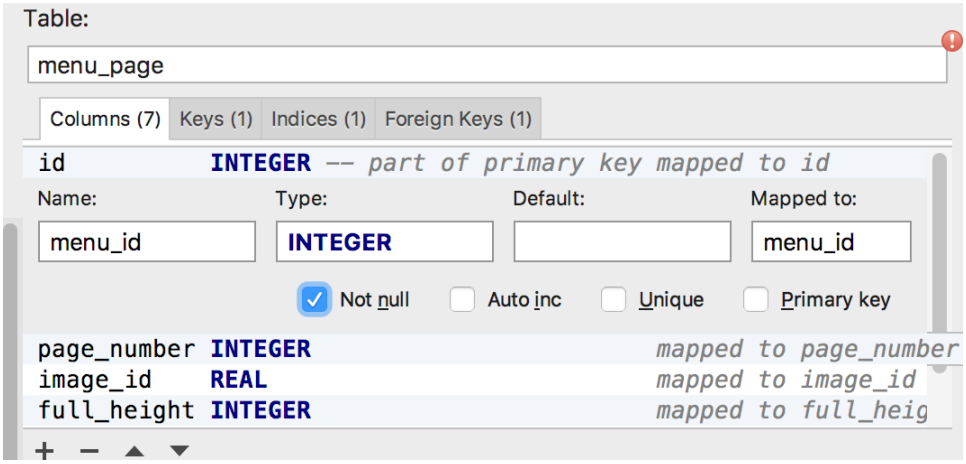
- We will define the Primary Key of the table now. Double-click on the id row in the Columns(14) tab. Select **Not Null**, **Unique** and **Primary key** check box.



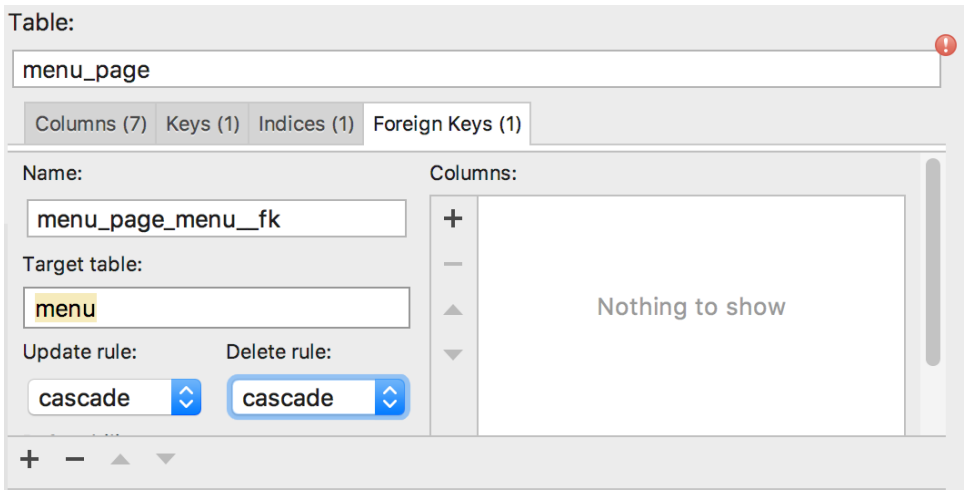
- Click on "OK" to import the data from menu.csv.

4.3.2.2 Import menu_page.csv

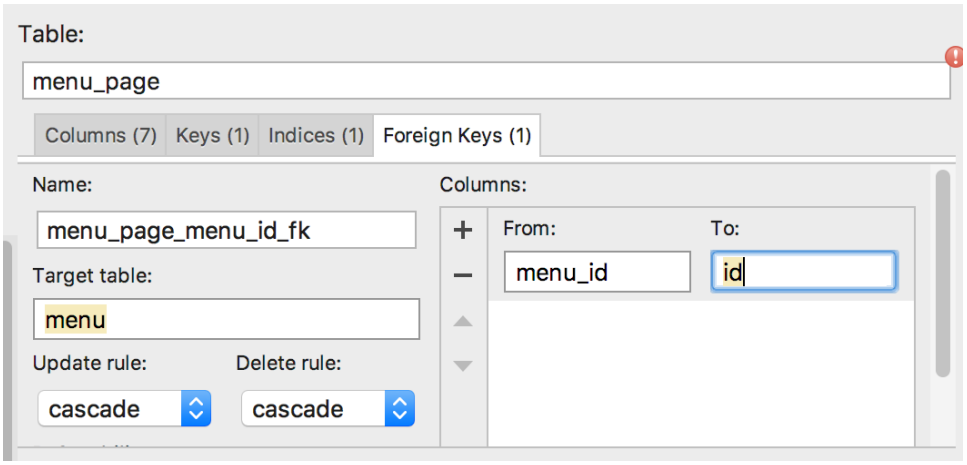
- We have imported the menu_page.csv the same way as the menu.csv.
- Configure the Table Name and the Primary Key.
- Since menu_page has foreign key constraint with the menu table, we need to define that.
- Double click on menu_id row and select Not Null check box.



- Click on the Foreign Keys tab and click on the + button at the left bottom.
- Enter "menu" in the Target table field. Then select both the Update & Delete rule as **cascade**.



- Now click on the + button in the top middle (Columns) section. Enter menu_id in the From field and id in the To field.



- Click on Ok to import the data.

4.3.2.3 Import dish.csv

- Importing dish.csv is same as the menu.csv.

4.3.2.4 Import menu_item.csv

- Importing menu_item.csv is same as the menu_page.csv.
- However, this time we need to create 2 foreign key constraints (menu_page & dish) instead of one.

4.4 Integrity Constraints

We imported the data from csv after enabling the foreign key support.

- When loading the menu_item table we found **241 integrity constraint violations** with the dish table. There are 241 rows in the menu_item csv where the dish id is empty. We could not load the data into the menu item table.
- The log of **241 integrity constraint violation errors** can be found in the following file.

- All 241 rows with empty dish_id from the menu_item table has been captured in a separate csv.
- The menu_page table has some menu_id which is not present in the menu table. The following query provided the list of **5803** rows from menu_page table which are violating the integrity constraint (There are no null in the name column of the menu table).
    - **SELECT** menu_page.* **from** menu_page
      LEFT OUTER JOIN menu
      on menu_page.**menu_id**=menu.**id**
      where menu.**name IS null**
    - The **5803** rows from menu_page table has been extracted to a separate csv file, which can be viewed here.
- Deleted the **5803** rows from the menu_page table using following query.
    - **delete from** menu_page **where id in** (
      SELECT menu_page.**id**
      FROM menu_page
      LEFT OUTER JOIN menu
      ON menu_page.**menu_id** = menu.**id**
      WHERE menu.**name IS NULL**
)
- Now menu_page has **61134** rows. Which matches with the count of the following query.
    - **SELECT** *count* (*) **from** menu
      LEFT OUTER JOIN menu_page
      on menu.**id** = menu_page.**menu_id**

- The menu_item table has **86347** rows where the dish id is violating the integrity constraint (there is no reference in the dish table). There are no null entries in the name column of the dish table)
    - **select** * **from** menu_item
      LEFT OUTER JOIN dish **on** menu_item.**dish_id** = dish.**id**
      where dish.**name is null**
    - The **86347** rows from menu_item table has been extracted to a separate csv file, which can be viewed here.

- Deleted the **86347** rows from the menu_item table using following query.
    - **DELETE FROM** menu_item **where id in** (
      SELECT menu_item.**id**
      FROM menu_item
      LEFT OUTER JOIN dish **ON** menu_item.**dish_id** = dish.**id**

```
  WHERE dish.name IS NULL
)
```

- Now the menu_item table has **4744** rows where the menu_page_id is violating the integrity constraint (there is no reference in the menu_page table).
    - ```
      select * from menu_item
      LEFT OUTER JOIN menu_page on menu_item.menu_page_id =
      menu_page.id
      where menu_page.menu_id is null
      ```
    - The **4744** rows from menu_item table has been extracted to a separate csv file, which can be viewed here.

- Deleted the **4744** rows from the menu_item table using following query.
    - ```
      DELETE FROM menu_item where id in (
        SELECT menu_item.id
        FROM menu_item
          LEFT OUTER JOIN menu_page ON menu_item.menu_page_id =
      menu_page.id
        WHERE menu_page.menu_id IS NULL
      )
      ```

4.4.1 Row Counts

Table Name	Row Count in Original CSV	Row Count after OpenRefine cleanup	Current Row Count in SQLite
menu	17,545	17,545	17,545
menu_page	66,937	66,937	61,134
dish	423,397	369,317	369,317
menu_item	1,332,726	1,332,726	1,241,394

4.5 Clean Scripts

Following three SQL commands are used to further clean up name and notes columns in menu dataset and name column in dish dataset. After cleaning the data set in Open Refine 3 columns had """ at the beginning and end. We had to clean up additional " (quote) using SQL query.

```
update menu set name=SUBSTR(name, 2,LENGTH(name)-2)
```

```
update menu set notes=SUBSTR(notes, 2,LENGTH(notes)-2)

update dish set name=SUBSTR(name, 2,LENGTH(name)-2)
```

4.6 Data Extraction

We have extracted the data based on the relevant columns available in the 4 data sets which can be used in further study. Extraction query that we used can be seen below.

4.6.1 Data Extraction Query

We have joined all 4 tables and included menu_item.price, dish.name, menu.event, menu.venue, menu.currency, menu.place, menu.date, menu.page_count, menu.dish_count in the select query when price is not equal to 0 and date is also not empty. This data will help us to perform time series analysis on Price and Date field.

```
select
menu_item.price,dish.name,menu.event,menu.venue,menu.currency,menu.place,
  menu.date,menu.page_count,menu.dish_count
from menu_item,dish,menu_page,menu
where menu_item.dish_id=dish.id and menu_item.price<>0
  and menu_page.id=menu_item.menu_page_id
  and menu.id=menu_page.menu_id
  and trim(menu.date) <> ''
order by menu_item.price desc
```

The extracted csv file can be found here.

5 Provenance

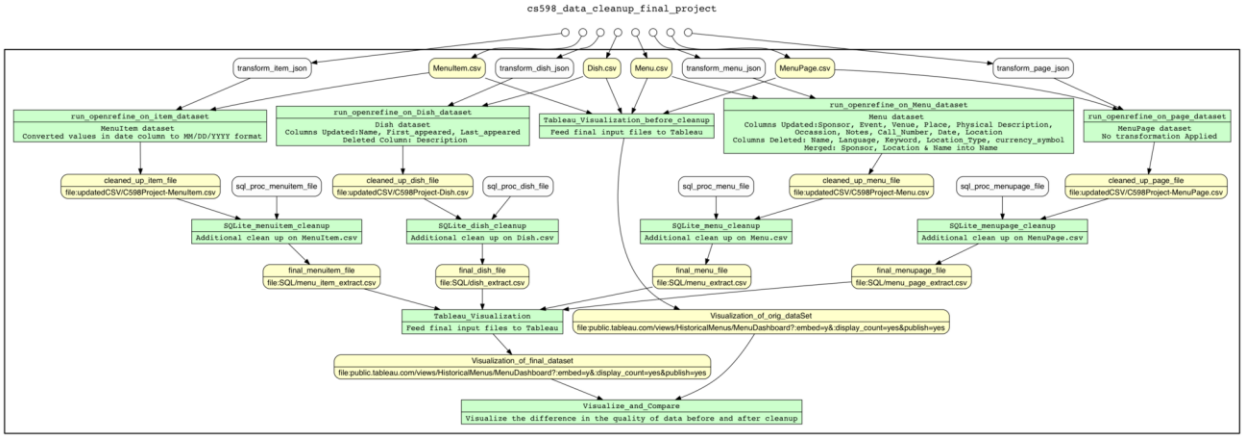
The end to end workflow was created using the YesWorkflow tool. The chart was created using offline version of YesWorkflow tool. We had to install dot tool from graphviz.com website to postprocess the gv file.

Following command was used to generate the image file of the workflow.

```
yw graph -c graph.view=COMBINED -c graph.layout=TB workflow1.py | dot –
Tpng -o workflow1.png
```

The first level of diagram shows the OpenRefine's json file and corresponding csv file as input to the workflow. See section 3 for more details.

The run operation produces a cleaned-up version of the csv files. These files are than imported into SQLite and addition SQL procs are run to create profile and do further cleanup of the data. See section 4 for additional details.

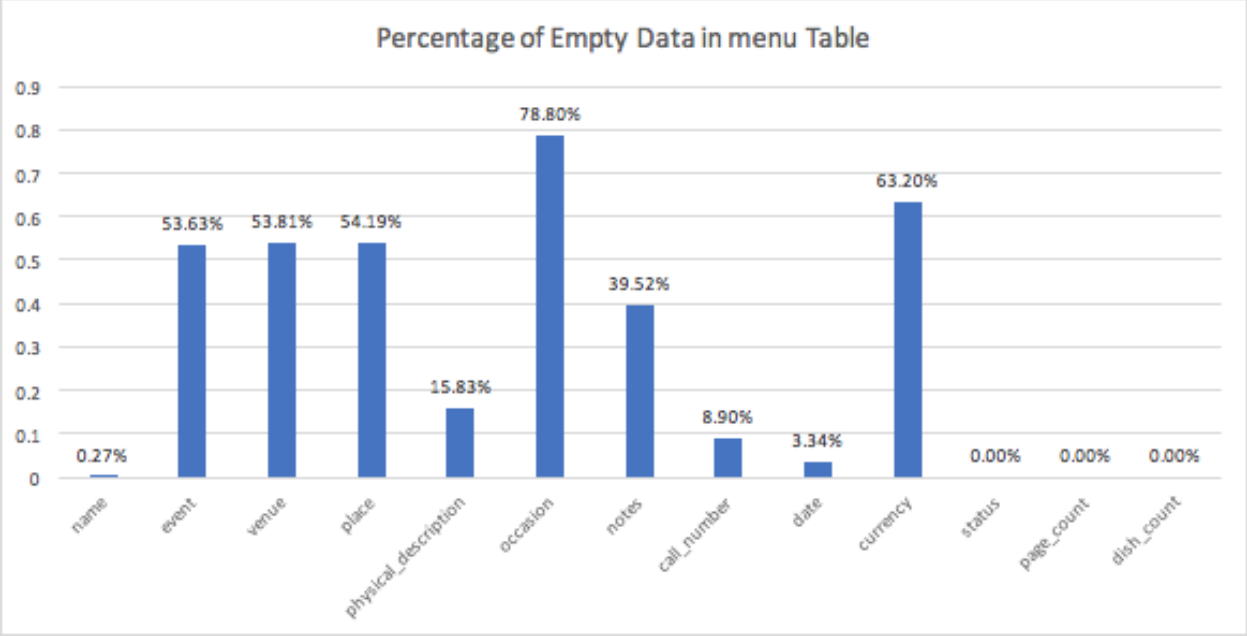


6 Dataset Quality Analysis

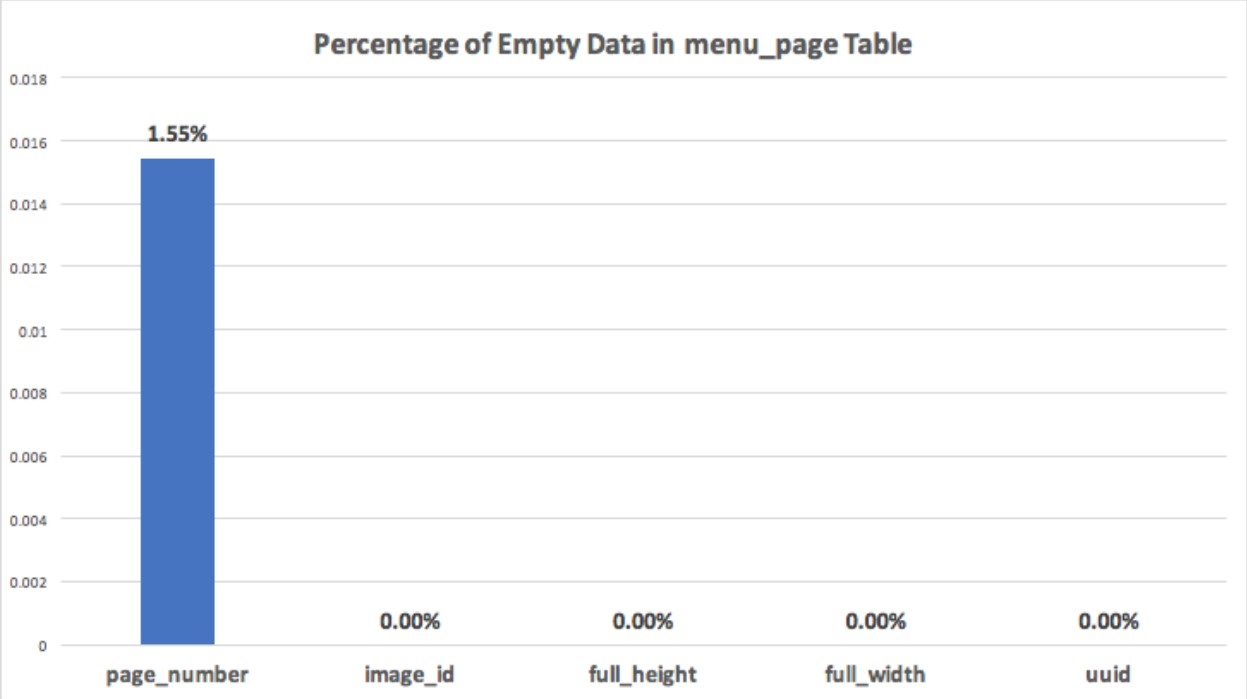
6.1 Analysis of empty data in the tables

The following 4 charts show the percentage of nulls and zeros in each of the columns in the tables after OpenRefine and SQLite cleanup.

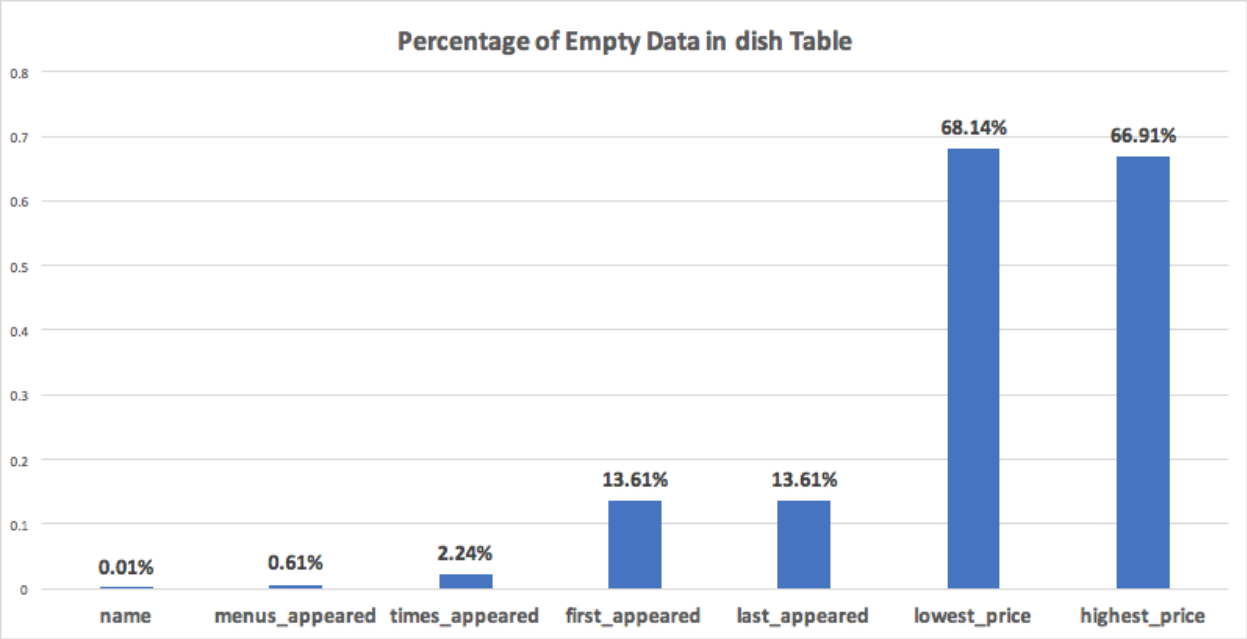
6.1.1 Percentage of empty data in menu table



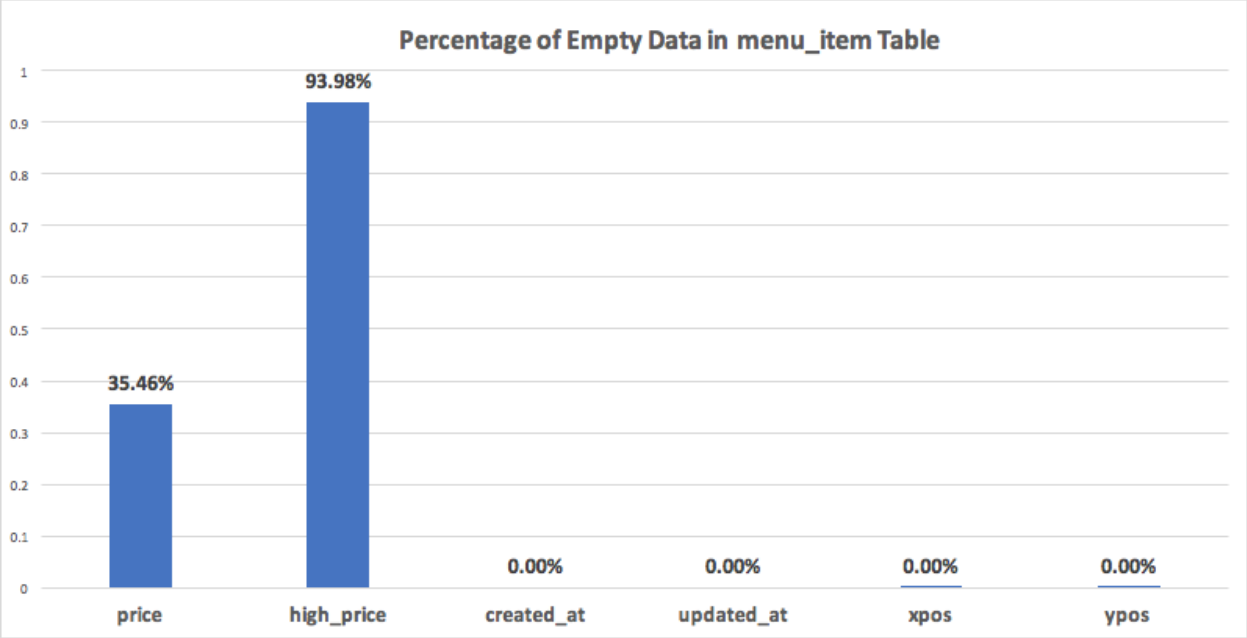
6.1.2 Percentage of empty data in menu_page table



6.1.3 Percentage of empty data in dish table



6.1.4 Percentage of empty data in menu_item table



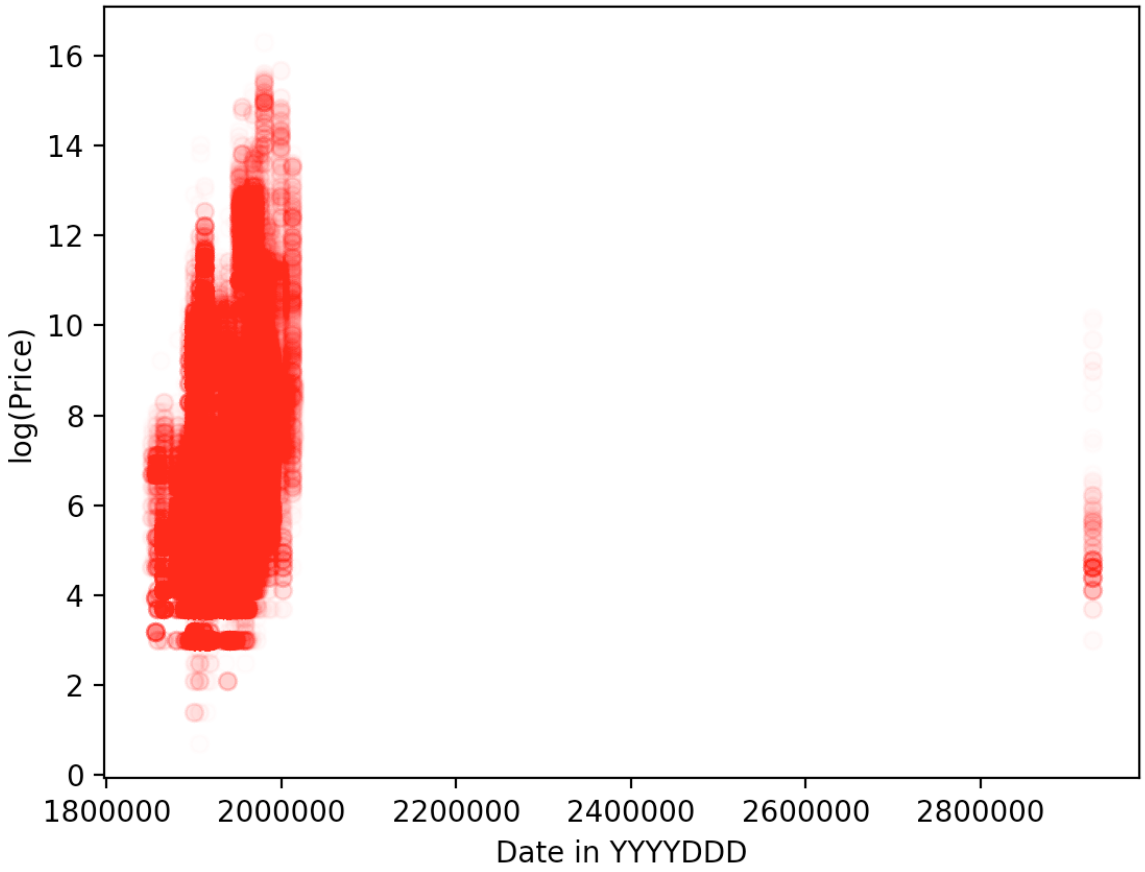
6.2 Regression Analysis Using Least Squares Method

We performed regression analysis to determine if any of the continuous variables (dish_count, page_count, data) can help predict the price (response variable).

We have transformed the date field to a continuous numeric variable (the transformation can be seen in the python code segment below). The input data file `menu_price_date_regression.csv` used for the regression can be downloaded here.

6.2.1 Price vs Date (logScale)

The converted date was plotted that against the log (price). We used a logScale because of the very high concentration of values in the low-price range. The plot clearly shows outliers for dates greater than 2800000 (clearly indicating user data entry errors).



6.2.2 Python code for Date Transformation and Outlier Detection

```
1.  import numpy as np
2.  import pandas as pd
3.  import datetime
4.  import matplotlib.pyplot as plt
5.
6.  def convertDate(old_dt):
7.      fmt = '%m/%d/%Y'
8.      dt = datetime.datetime.strptime(old_dt, fmt)
9.      tt = dt.timetuple()
10.     jd = tt.tm_year * 1000 + tt.tm_yday
11.     return jd
12.
13. df = pd.read_csv('menu_price_date_regression.csv')
14. df['n_date'] = df['date'].apply(convertDate)
15. df['n_price'] = np.log(df.price) + 6
16.
17. # Plot log(Price) Vs. Date
18. plt.plot(df.n_date,df.n_price, 'ro',alpha=.01)
19. plt.ylabel('log(Price)')
20. plt.xlabel('Date in YYYYDDD')
21. plt.show()
22.
23. # Remove outliers
```

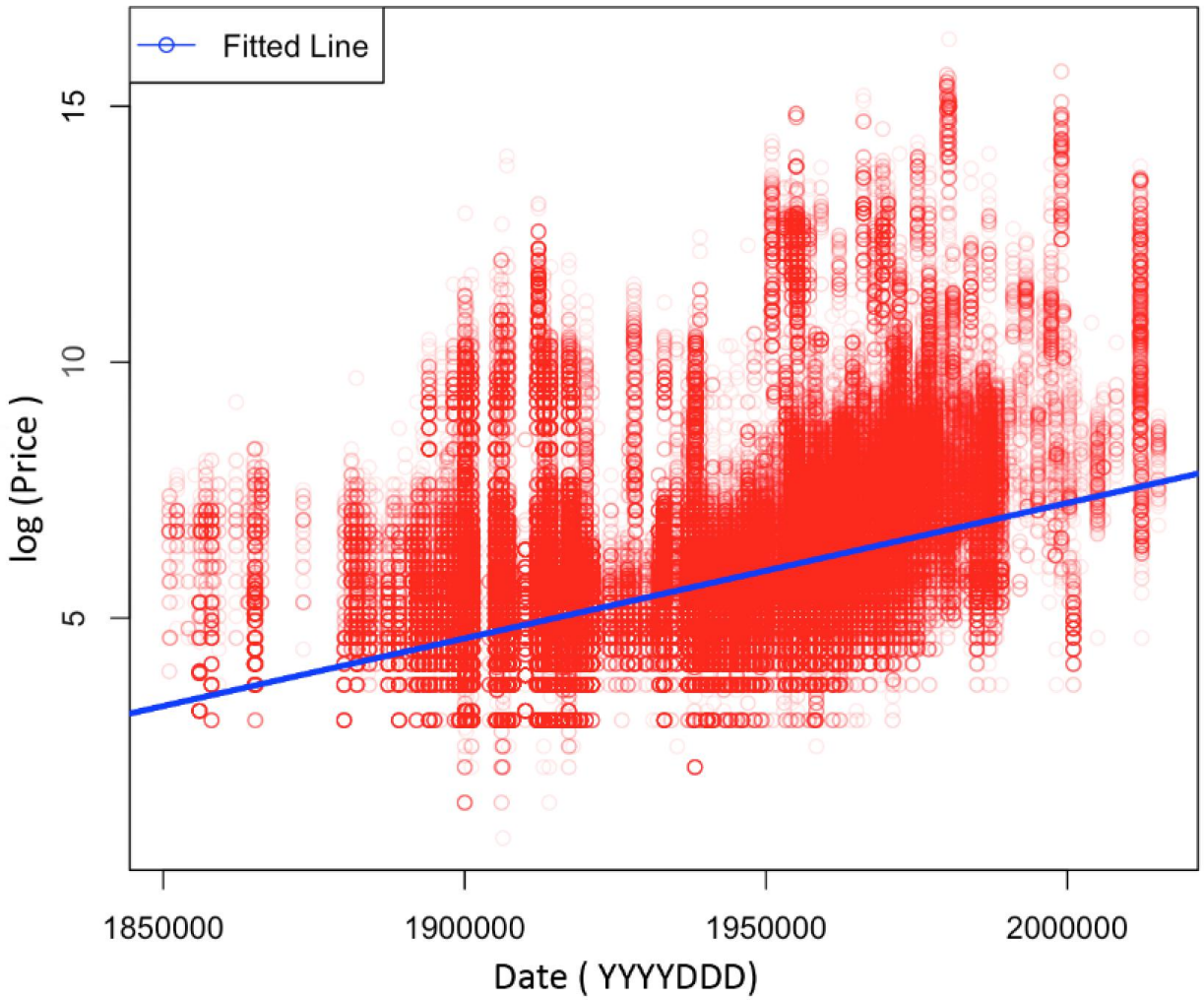
```
df = df[df.n_date < 2200000]
df.to_csv("menu_price_date_regression_final.csv")
```

We have removed the outliers and performed a regression analysis.

6.2.3 R Code for Linear Regression

We used R to perform the regression analysis and built a linear model regressing the log(Price) against the date. The input data file `menu_price_date_regression_final.csv` used for the regression can be downloaded here.

```
data=read.csv(file='menu_price_date_regression_final.csv')
model=lm(n_price~n_date,data=data)
plot(n_price~n_date,data=data,col=rgb(1,0,0,.1),xlab='Date',
     ylab='Price',)
abline(model,col=rgb(0,0,1),lwd=3)
```



6.2.4 Analysis

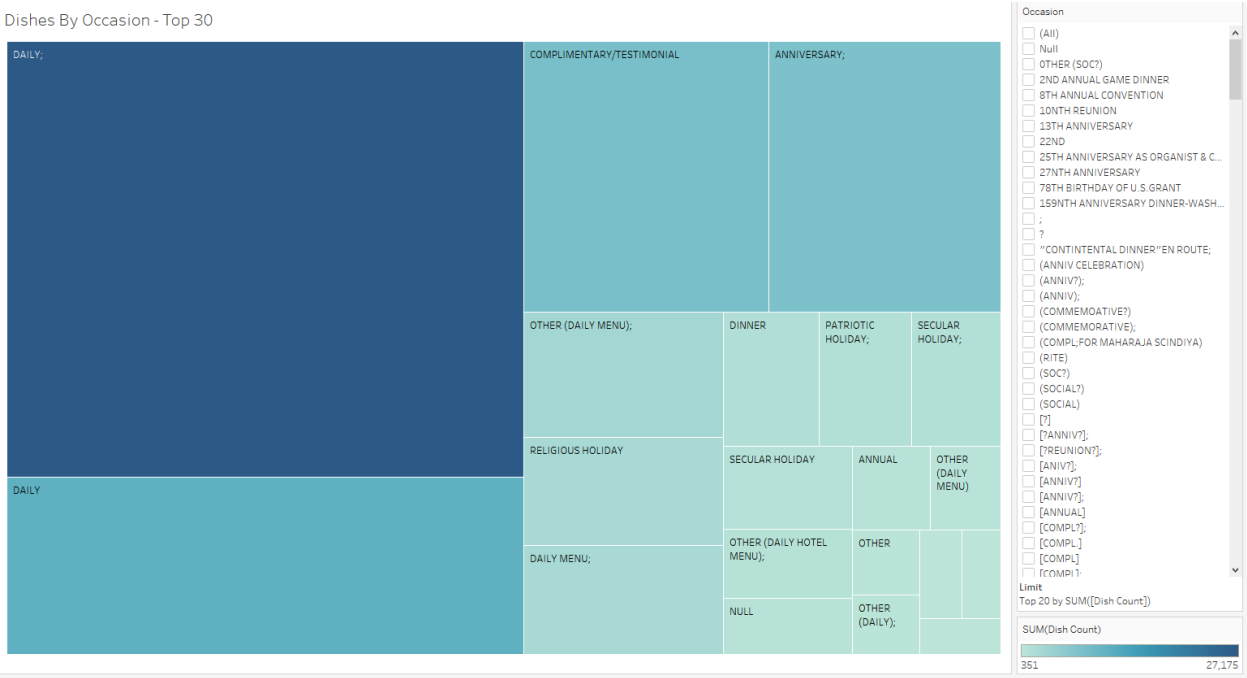
- We can say from the plotted data that there is a relationship between the price and date.
- The fitted regression line shows the estimated line to explain the change of price with date.

6.3 Visualization (Dirty Vs Clean Data)

We created visualizations in **Tableau** using the data extracted after the SQLite cleanup to draw a complete picture of how food evolved during the late 19th and early 20th century.
We have attempted to model the price, however there are still several items where the price was not valid. We addressed some of our initial use cases/questions, though not all could be addressed due to incomplete nature of the missing data. These complete visualization can be viewed on these links: [before cleanup] and [after cleanup].

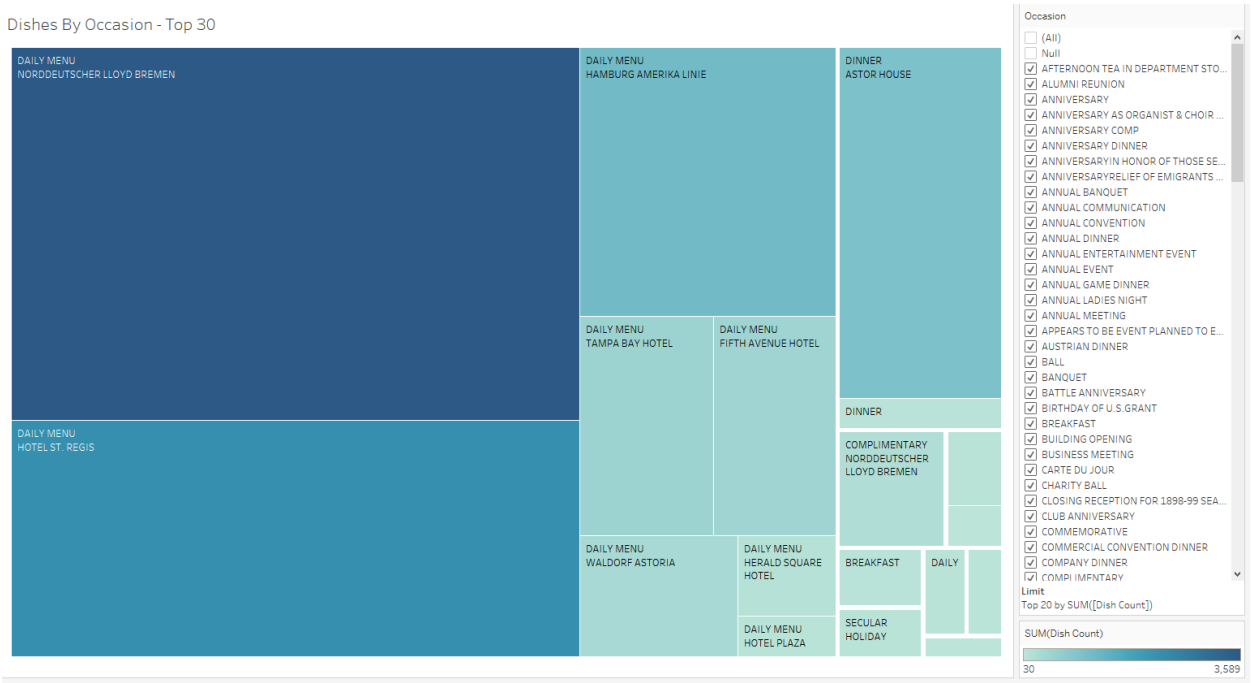
6.3.1 Top 30 Dishes organized by Occasion/Special Event

6.3.1.1 Prior to Cleanup



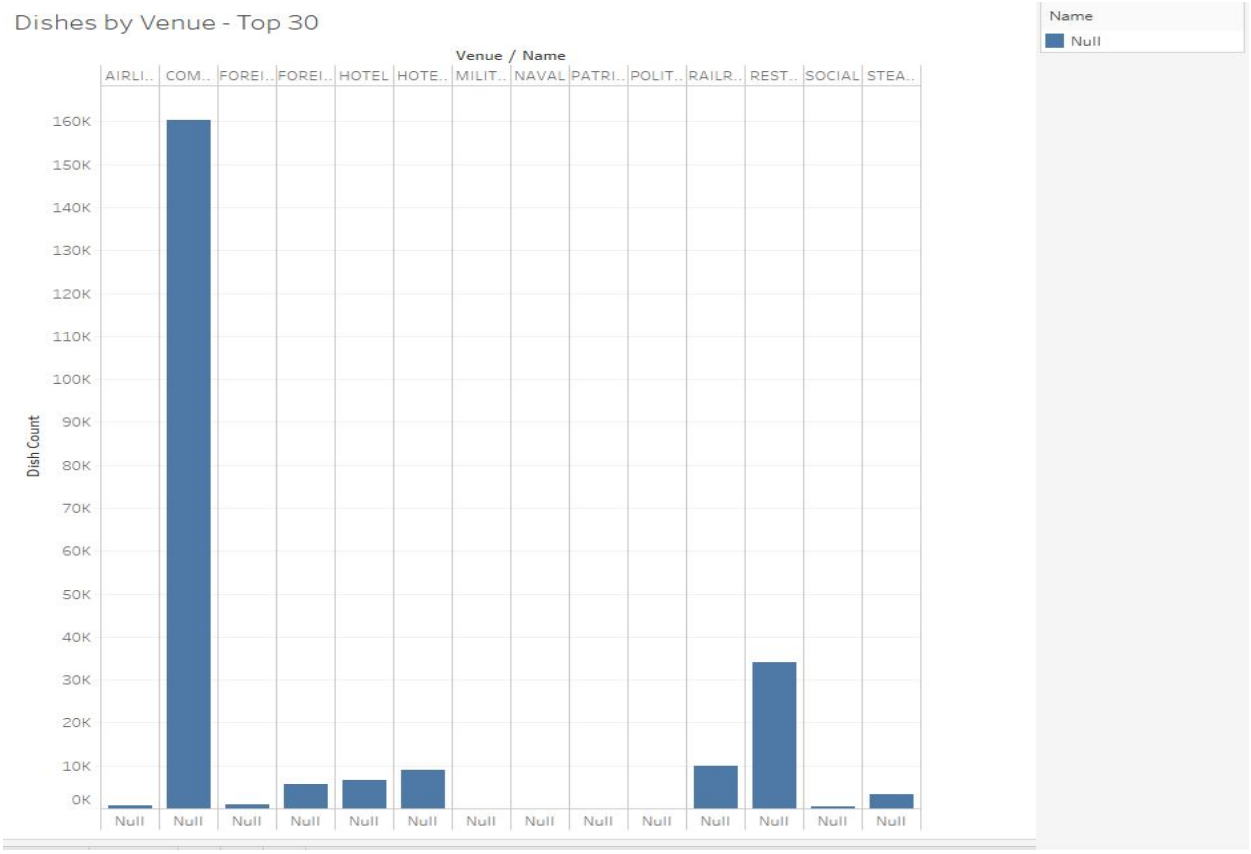
Has lot of null entries and the left-hand panel invalid and special characters that are not meaningful for extracting any results.

6.3.1.2 After Cleanup

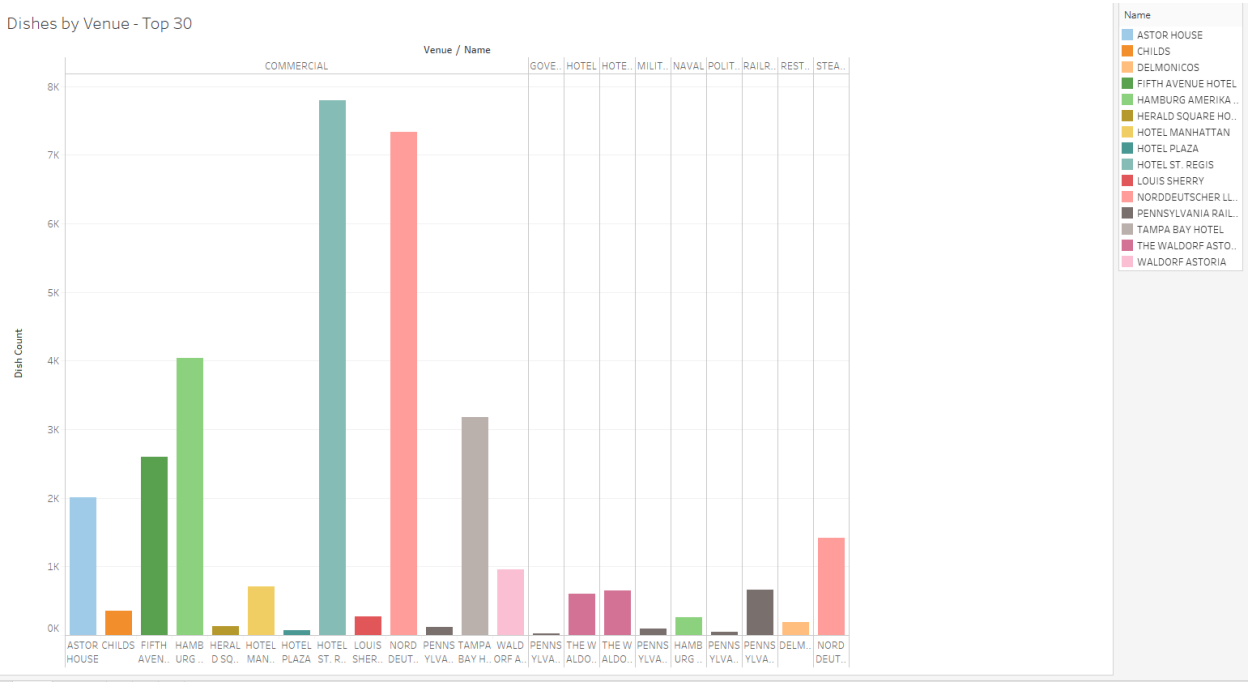


6.3.2 Top 30 Dishes and the venues where they appeared

6.3.2.1 Prior to Cleanup

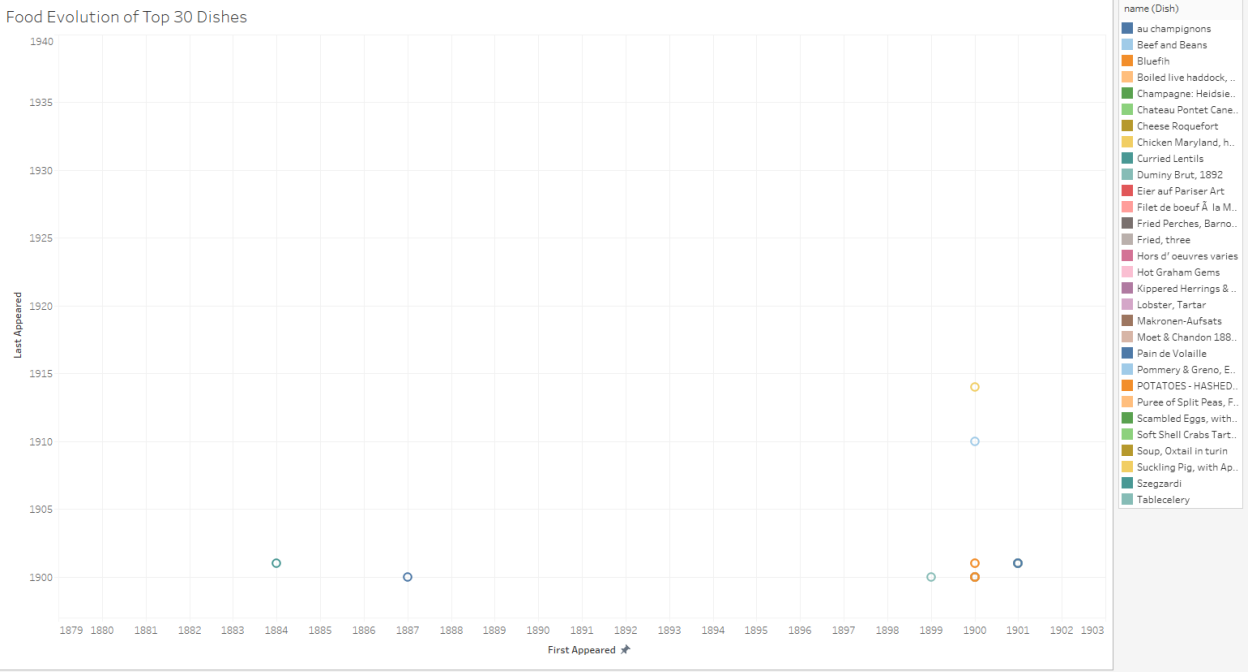


6.3.2.2 After Cleanup



6.3.3 Food Evolution of Top 30 Dishes

6.3.3.1 Prior to Cleanup



6.3.3.2 After Cleanup


Food Evolution of Top 30 Dishes
Last Appeared
2010
2000
1990
1980
1970
1960
1950
1940
1930
1920
1910
1900
1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903
First Appeared
Name (DishExtract)
->fruits.
Ale, Bass
Assorted Fresh Fruit
Broiled Squab
Candied Sweet Potat..
Cape Cod Oysters
Capon
Celery Stuffed
Cheddar Cheese
Cheese ..Gorgonzola
Cheese, Pineapple
Chocolate Parfait
Chocolate Sundae
Coffee - Special
Cream Of Tomato So..
Dressing, French
Dutch Cheese
Fancy Ices
Figs In Syrup
French Pastry
French Sardines
Green Turtle Soup
Grilled Sweet Potato..
Heart Of Lettuce Sal..
Hock, Sparkling
Ice Cream, Pistachio
Jelly Omelette
Raspberry Sherbet
Rolls And Butter
Stuffed Tomato

6.3.4 Top 30 popular Dishes of all times

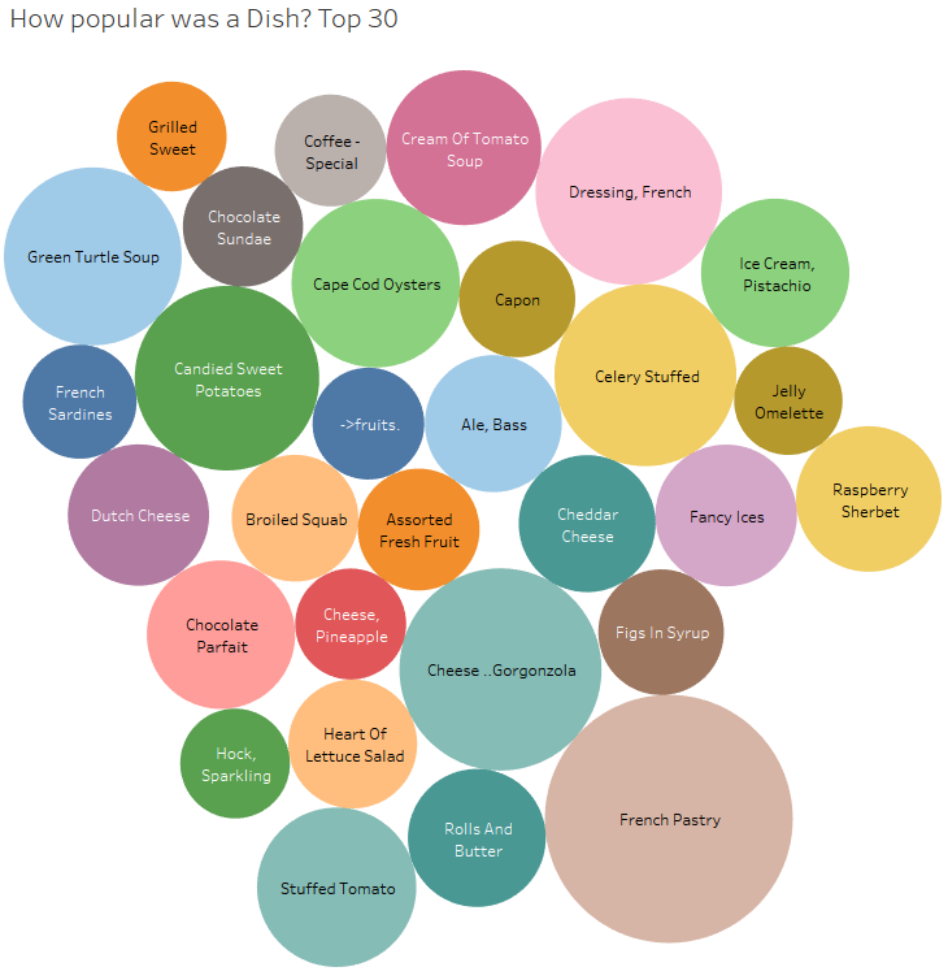
6.3.4.1 Prior to Cleanup

How popular was a Dish? Top 30

name (Dish)	
French Pastry	659
Gorgonzola Cheese	435
FRENCH DRESSING	373
Candied Sweet Potatoes	365
Stuffed celery	355
Green Turtle Soup	337
Cape Cod Oysters	300
Stuffed Tomato	261
Cream of tomato soup	257
Chocolate Parfait	239
Pistachio Ice Cream	236
Raspberry sherbet	218
Fancy Ices	215
Dutch cheese	214
Rolls and butter	204
Cheddar Cheese	201
Heart of Lettuce Salad	178
Figs in syrup	169
Broiled Squab	166
Assorted Fresh Fruit	159
Chocolate Sundae	155
Capon	145
French Sardines	139
Pineapple Cheese	132
Grilled sweet potatoes	128
Force	127
Beets in Butter	116
Two Fried Eggs	115
Kakao	114
Kraftbruhe	113

This is the only visualization that did not have any major missing entries mainly because we are viewing only the top 30 dishes. There were several missing but did not just fall under the top 30 visualized here. The cleanup helped fix all these gaps.

6.3.4.2 After Cleanup



Name (DishExtract)	
French Pastry	659
Cheese ..Gorgonzola	438
Dressing, French	373
Candied Sweet Potatoes	365
Celery Stuffed	355
Green Turtle Soup	339
Cape Cod Oysters	304
Stuffed Tomato	272
Cream Of Tomato Soup	257
Chocolate Parfait	239
Ice Cream, Pistachio	236
Raspberry Sherbet	227
Fancy Ices	215
Dutch Cheese	214
Rolls And Butter	204
Ale, Bass	202
Cheddar Cheese	201
Heart Of Lettuce Salad	178
Broiled Squab	172
Figs In Syrup	169
Assorted Fresh Fruit	159
Chocolate Sundae	155
Capon	145
French Sardines	139
->fruits.	136
Coffee - Special	134
Cheese, Pineapple	132
Hock, Sparkling	129
Grilled Sweet Potatoes	128
Jelly Omelette	126

Click on these links to view the entire visualization before cleanup and after cleanup.

7 Observations and Conclusions

7.1 Source and What We Cleaned

The quality of the originally provided dataset was very poor. It is not a surprise because the data was crowd sourced and in some cases the source is at least a few decades old. We used OpenRefine to introduce basic but essential transformation such as consistent formatting for dates, regular expressions to remove punctuation/non-alphanumeric characters and various clustering features to merge similar words and phrases. We took opportunity to eliminate columns with no values and merge duplicate columns. It helped reduced the size of the dataset without impacting the structure of the dataset.

7.2 Integrity Constraints and Other issues in SQLite

Despite the deep cleaning of various columns in OpenRefine, we encountered integrity issues after importing it into SQLite. The empty, null and zero value columns made it harder to import all rows of the dataset.

7.3 What we learned

This project made us appreciate the challenges and nuances of data cleaning. Data cleaning involves multiple steps and typically requires multiple iterations before data is clean enough to be used for further analytics.

The semantic understanding is just as important as the syntactic understanding of the dataset. While we can easily infer the intent of the date column and why it is part of the dataset, we had tough time figuring out the purpose and intent of other columns such as – call_number. There is no universal definition of clean data and it will vary from one user to another. We believe that we have improved quality of the dataset by a significant margin and it is ready to be used for deeper analytics.

7.4 How long did the process take?

The data cleaning process is a time-consuming process. It took us 16 hours to go through multiple iterations to clean up the data in OpenRefine. It took us another 8 hours to import data into SQLite, apply additional transformations and integrity constraints. The YesWorkflow portion took about 4 hours and tableau visualization process took another 4 hours.

7.5 Use cases answers

- The desserts and appetizers were more popular than the actual entrees.
- French Pastry was the most popular dish of all time.
- Hotel St. Regis served most dishes amongst all venues.
- Daily menus were more popular than meal only menus.
- Desserts and appetizers have shown remarkable resilience and continued to be a popular choice over multiple decades. For instance, Raspberry Sherbet and Pistachio ice cream lasted almost 90 years.
- Cream of tomato soup lasted 95 years.
- Lettuce Salad continued to be popular all the way through 2012.
- French dressing always accompanied lettuce salad. The likely explanation is that it was the only dressing offered on the menu.
- Many of the most popular dishes in the data set are single ingredients or beverages, rather than full entrées, something that is indicative of the food tastes of the upper class as most of the data is from high end hotels and cruise lines.
- We performed a basic regression analysis using least squares method and found a linear trend between the price on a log scale and the date.

8 Links

- Original dataset https://uofi.box.com/s/8i7ydcaf6s6h7ihypat4d7169g1h72h4
- OpenRefine's operation history can be downloaded from here:
    - menuOperationHis.json– history of changes for Menu CSV
    - dishOperationHis.json– history of changes for Dish CSV
    - menuItemOperationHis.json– history of changes for MenuItem CSV
    - menuPageOperationHis.json– history of changes for MenuPage CSV

- The output CSV files after OpenRefine transformation can be downloaded here:

    - CS598Project-Menu.csv– updated Menu CSV output from OpenRefine
    - CS598Project-Dish.csv– updated Dish CSV output from OpenRefine
    - CS598Project-MenuItem.csv– updated MenuItem CSV output from OpenRefine
    - CS598Project-MenuPage.csv – updated MenuPage CSV output from OpenRefine

- SQLite DB dump can be found here.
- Data Extract can be found here:
    - menu_extract.csv– Database extract for menu table
    - dish_extract.csv– Database extract for dish table
    - menu_item_extract.csv– Database extract for menu_item table
    - menu_page_extract.csv– Database extract for menu_page table

- YesWorkflow script.
- YesWorkflow diagram.
- Tableau visualization of the dataset
    - Before cleanup
    - After cleanup