CS 513: Theory and Practice of Data Cleaning

Data Cleaning Project: Phase-2

Summer 2022

By:Team 31

Candice Yan

Email: yan40@illinois.edu

Yilin Hou

Email: yilin17@illinois.edu

Fangsheng Yang

Email: fyang28@illinois.edu

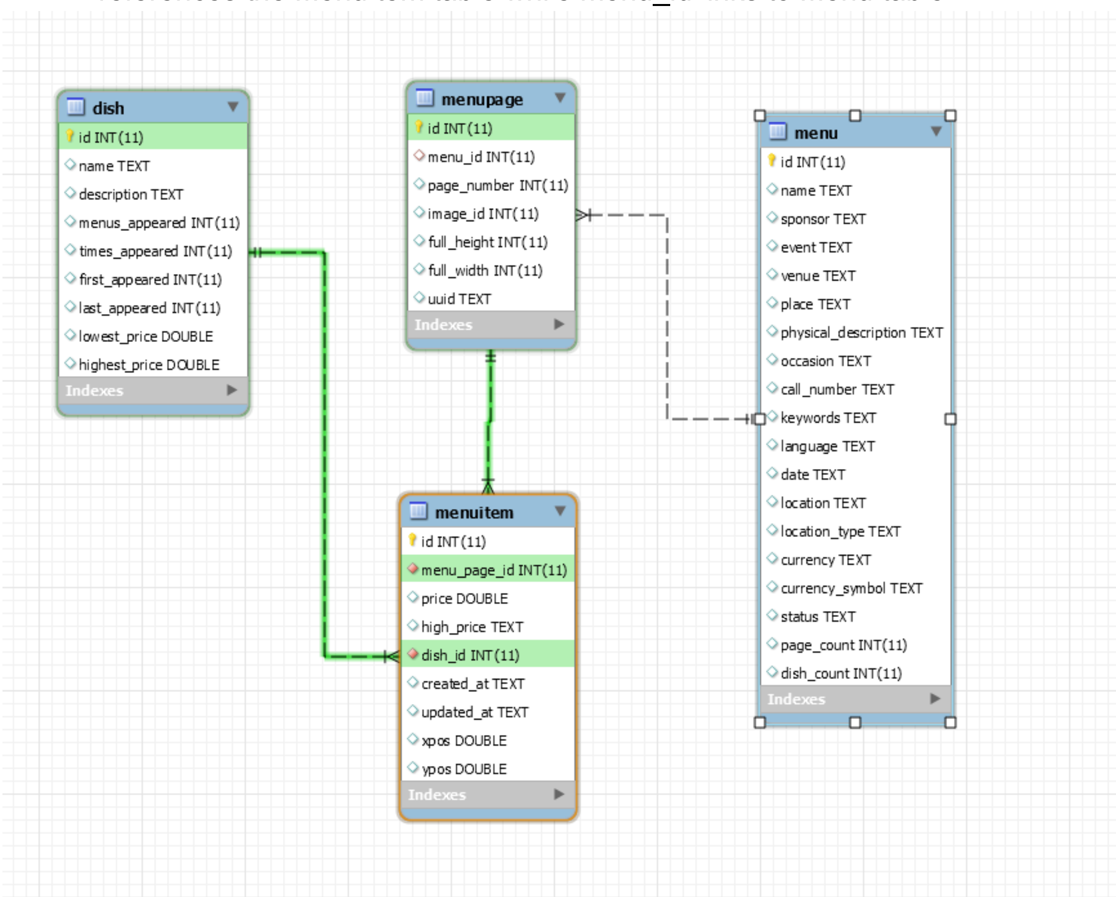
Table of Contents

1. Introduction

1.1 Dataset Description

Our team uses the dish and menu data from the New York Public Library. The four tables we used are:

1. Dish table which contains dish information including dish name, dish price, appeared in how many menus, appear times and dates. This table has 429,943 rows and 9 columns with id as primary key. Meanwhile, it’s a foreign key referring to the dish_id column in Menuitem table.
2. Menu table which contains menu information including menu name, sponsor, event, venue, pages, count of dishes on the menu and so on. This table has 17,550 rows and 20 columns with id as primary key. It’s also a foregin key referring to the menu_id in the MenuPage table.
3. MenuItem table has 1,048,576 rows and 9 columns with id as primary key. The two foreign keys menu_page_id and dish_id links MenuPage table and Dish table.
4. MenuPage table has 66,937 rows and 7 columns with id as primary key. Id also references the MenuItem table while menu_id links to Menu table.



1.2 Data Quality Issues

In order to get the dish table and menu table joint, we need to join through the menuitem and menupage table by menu_id, dish_id and menu_page_id (primary key). This will require integrity constraints not to be violated, thus a check on those id columns will be a must.

In the Menu table, venue and event are character fields, which contain ambiguous inputs and unconsolidated records. For example, “social” and “social club” can be consolidated into one venue name.
Since we are analyzing dishes over different time periods, date will be a very important feature to mark the timeline. The date column also suffers from a large outlier - a year 2928 seems obviously odd and should be removed.

In the name column of the Dish table, dish names have many issues, including but not limited to inconsistent letter cases, extra whitespaces, special characters, misspelling, word swapping, blank value, and so on. The appeared date column of the Dish table has some missing values (~ 60k), which is also worth to note. We can simply delete them as imputation could not be really appropriate.

1.3 Main Use Cases

The menu dataset can be used to extract useful information about dishes. For instance, we can investigate popular dishes during different time periods to figure out changes of people’s eating preference over time. This can be further broken down into more granular levels such as by venue or event. Because there are multiple data quality issues in the dataset, we cannot obtain the proper results for the main use cases. We will implement the following 2 use cases after data cleaning processes:

1. Select top 10 most popular dishes in a given time period (e.g. 1990 - 2000).
2. Select most 10 frequent venue menus and for each venue its most popular top 3 dishes.

1.4 Data Quality Change

The following table contains the summary of data changes showing the results of data quality changes:

<table>
<tr><td>Table</td><td>Column</td><td>Change</td></tr>
<tr><td>Menu</td><td>event</td><td>Clusters decrease from 1769 to 1450
Transformed 140 rows to uppercase
Edited 11,377 cells with following cluster transformations:
Fingerprint method to consolidate label cases and clean special characters
Metaphone3 to combine more complex labels
Daitch-mokotoff</td></tr>
</table>

		● Removed consecutive spaces in 4 rows ● Removed special characters in 216 cells ● Transform 9049 cells into string format
Menu	venue	● Clusters decrease from 234 to 59 ● Transformed 13 rows to uppercase ● Edited 24,129 cells with following cluster transformations: ○ Combine cluster with Daitch-mokotoff ○ Nearest neighbor (radius 20 block 10) to correct spelling issue ("PITHER" to "OTHER") ○ Combine "OTHER PRIVATE" cluster with metaphone3 ○ Nearest neighbor (radius 100 block 5) Cluster "OTHER" and "PRIVATE" ○ Manually update some labels to correct spelling errors and combine to clusters ● Removed special characters in 251 cells ● Transform 9336 cells into string format ● Drop 9358 rows missing Venue value ● update menu set venue = 'HOTEL' where venue like 'HOTEL%' ● update menu set venue = 'SOC' where venue like 'SOC%' ● update menu set venue = 'PROF' where venue like 'PROF%' ● update menu set venue = 'COM' where venue like 'COM%' ● update menu set venue = 'EDU' where venue like 'EDU%' ● update menu set venue = 'FOREIGN' where venue like 'FOREIGN%' ● update menu set venue = 'MIL' where venue like 'MIL%' ● update menu set venue = 'POL' where venue like 'POL%'
Menu	date	● Remove 4 rows with abnormal date value (i.e., 0001-01-01) ● Remove 586 rows with missing dates
Dish	name	● Trim all the leading and trailing whitespaces in Name Column (Text transform on 9045 cells in column name) ● Collapse consecutive whitespaces in Name Column (Text transform on 6415 cells in column name) ● Add a new column based on the name column, name the new column as name_case ● Transform the name_case column to Titlecase (Text

		transform on 281551 cells in column) ● Add a new column based on the name_case column name the new column as name_cleaned and transform data to remove unnecessary characters by using GREL value.replace(/[:%#@!<>\\()\[\]\?\"\=\-*,\.\+]/, " ").replace(/\s+/," ").trim() (Text transform on 203759 cells in column) ● Cluster and Edit column, method: Key-Collision, keying function: fingerprint (Mass edit 75132 cells in column) ● Cluster and Edit column, method: Key-Collision, keying function: ngram-fingerprint, Ngram Size 2 (Mass edit 15584 cells in column) ● Cluster and Edit column, method: Key-Collision, keying function: Metaphone3 (Mass edit 4997 cells in column)
Dish	id	● Drop 241 rows without dish_id
Dish	first_appeared & last_appeared	● Drop 6 rows that violate the integrity constraint: First_appeared date is smaller than last_appeared date
Menupage	full_height & full_width	● Fill null value in full_height & full_width default to ‘0’ (Impacted 329 rows)
Menupage	page_number	● Drop 1202 rows whose page_number is null value
MenuItem	dish_id	● Drop 241 rows whose dish_id is null value

Integrity constraint violation is checked for all the tables. The following summary table shows how data quality has been improved by comparing the differences between before data cleaning and after data cleaning:

Table	ICV Check	Change
Dish	ID is unique and has no null value.	None
Dish	First_appeared date is smaller than last_appeared date.	6 violations → 0 violations
Dish	Lowest price is lower than highest price	None
Menu	ID is unique and has no null value.	None
Menu	Venue column does not have invalid labels.	None

MenuItem	ID is unique and has no null value.	None
MenuItem	Dish_id and menu_page_id is not null.	dish _id has 241 null values → dish_id has 0 null values
MenuPage	ID is unique and has no null value.	None
MenuPage	Menu_id is not null.	None

2. Data cleaning Process

2.1 OpenRefine Data Cleaning - Menu Dataset

As described in the initial project plan, we are going to clean date, event, and venue columns in the menu dataset for our main use cases.

2.1.1 Event column

For Event column, the following steps are performed:

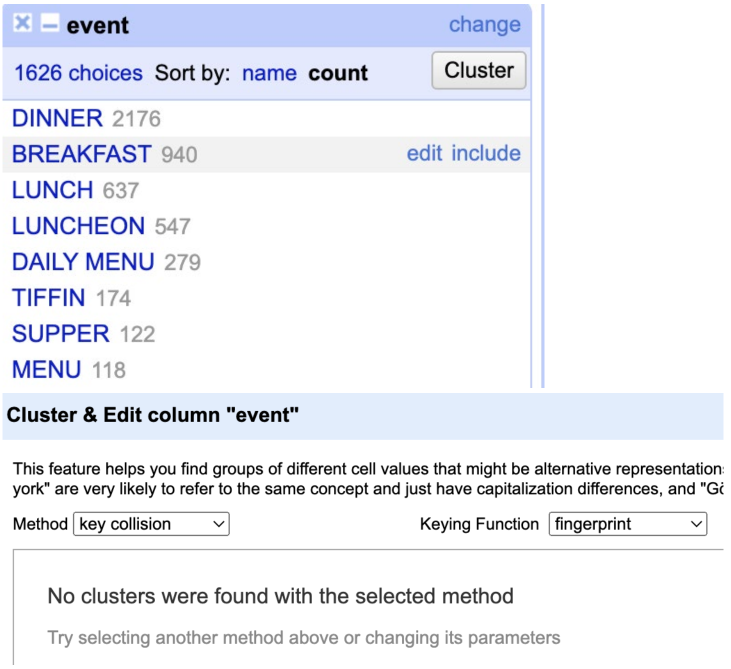
- Special characters are removed/replaced.

When exploring this column, some special characters catch our eyes, such as question mark, quoting mark, semicolon, brackets and parentheses. Unlike comma and dash mark, those special characters are not useful in bringing information and might cause errors in data import step. They are either removed or replaced with more proper marks. Take semicolon as an example, it is replaced by comma in the middle of the label and is removed at the end of label (i.e., “BREAKFAST; LUNCH; DINNER” to “BREAKFAST, LUNCH, DINNER”). This transformation reserves both the integrity and original information of the label.

- Leading and ending spaces are trimmed.
- Consecutive spaces are collapsed.
- All the characters are transformed to uppercase.

These following steps aim to format the column and reduce character length:

- Similar categories are clustered with key-collision method (metaphone3 and cologne-phonetic). This is to reduce the number of categories in the column as well as data ambiguity.



From the snippet above we can see that the frequent labels for the event are pretty normal, such as dinner, breakfast, lunch and luncheon. Those event names make sense and are self-explainable. When clustering with fingerprint as keying function, no further clusters could be found after removing special characters in prior.

However, we do detect some labels that share some extent of similarity using metaphone3 and cologne-phonetic as keying functions. For instance, the graph below represents a situation where annual dinner is very granular. We consolidate those labels to “ANNUAL DINNER”. The rationale to do so is that for our main use case, such granularity is not necessary and will cause information loss. Before re-clustering, granular labels only have one or two rows; after re-clustering they will increase the major label with 28 rows. Similar consolidations are done for “ANNUAL BANQUET”, “ANNIVERSARY”, “ANNUAL MEETING”, and etc.

Some apparent spelling errors are also corrected in this step, such as “DINER” and “MITAGESSEN” which are shown in below snippet.

- DINNER (2176 rows)
- DINER (22 rows)
- DNNER (1 rows)

- MITTAGESSEN (33 rows)
- MITAGESSEN (2 rows)
- MITTAGESEN (2 rows)

Notice that some granular labels are not further combined. The reason is that we are not able to estimate whether they belong to the same group. For instance, below labels are grouped using Daitch-Mokotoff function. Nevertheless, “ANNUAL MEETING DINNER” and “ANNUAL MEETING LUNCH” are evidently different categories (dinner vs. lunch), so we decide not to cluster them. Besides, 24 rows form a small portion of total rows and thus the impact of not clustering should be minor.

8	24	• ANNUAL MEETING (14 rows) • ANNUAL MEETING LUNCHEON (3 rows) • ANNUAL MEETING & BANQUET (2 rows) • ANNUAL MEETING & DINNER (1 rows) • ANNUAL MEETING AND BANQUET (1 rows) • ANNUAL MEETING AND DINNER (1 rows) • ANNUAL MEETING LUNCH (1 rows) • ANNUAL MEETING ON THE 169TH ANNIVERSARY OF THE BIRTH OF GEORGE WASHINGTON (1 rows)

- Cells are transformed to string format

When loading the dataset to SQL database and, we encountered problems in sorting data by event field. This is attributed to the inconsistency of data types within the column. To solve this problem, toString() function is applied to the event column so as to reconcile data format.

2.1.2 Venue Column

- Special characters are removed/replaced.

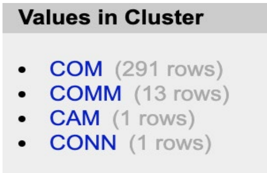
venue change
234 choices Sort by: name count Cluster

(SOC?) 1
(SOC?); 1
(SOC); 1
(SOCIAL?) 1
[?] 1
[?POSSIBLY A PRIVATE HOST?]; 1
[?SOC]; 1
[COM?]; 1
[COM] 3
[COM} 1
[EDUC?]; 1

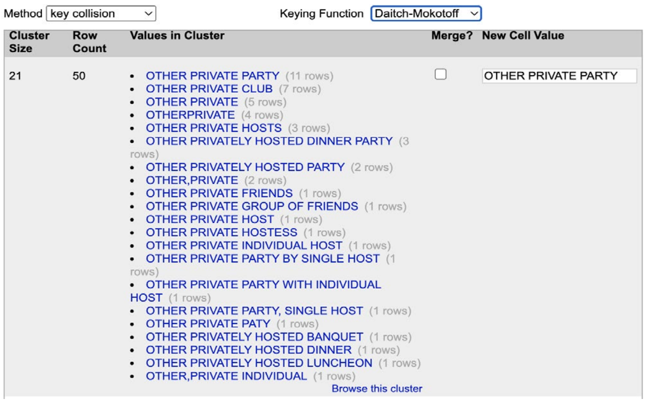
Similarly, special characters appear in the venue column, including brackets, parentheses, period, colons, braces, question marks and semicolons. They are either removed or replaced with more proper characters such as commas.

- Similar categories are then clustered with key-collision methods (Daitch-Mokotoff and metaphone3). This is to reduce the number of categories in the column as well as data ambiguity.

As an example shown in the table below, “COM” and “COMM” should refer to the same label and thus need to be clustered.



Another example is to use the nearest neighbor method (radius 100, block 5) and Daitch-Mokotoff to cluster “OTHER” and “PRIVATE”.



- Leading and ending spaces are trimmed.
- Consecutive spaces are collapsed.
- All the characters are transformed to uppercase.

These three steps aim to format the column and reduce character length.

- Spelling errors as below are manually corrected as the amount is small.

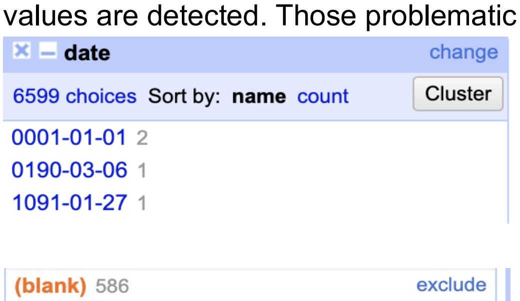
- Cells are transformed to string format.

Similar to the event column, data type in the venue column is not consistent. Again, toString() function is applied to the venue column so as to make it reconciled.

2.1.3 Date column

- Abnormal date values are removed.

Date columns are examined and cleaned, since it is used to select information to fulfill our main use case. Upon reviewing the dataset, problems are found as follows.
First, there are some abnormal date values such as “0001-01-01”. Second, multiple missing values are detected. Those problematic rows are deleted.



2.2 OpenRefine Data Cleaning - Dish Dataset

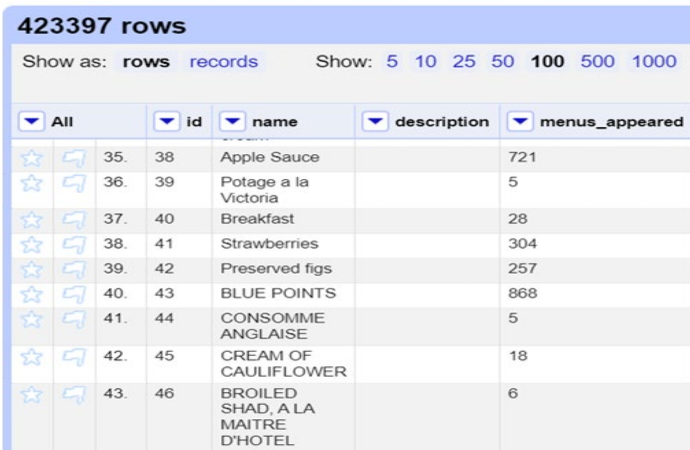
As described in the initial project plan, we are going to clean the name column in the dish dataset for our main use cases.

2.2.1 Name column

For the Name column, we first use OpenRefine to clean the data. The following two steps are performed because an extra whitespace or line-break character is difficult to identify using human eyes.

- Trim all the leading and trailing whitespaces in Name Column
- Collapse consecutive whitespaces in Name Column

Browsing the Name column data, we find out that the letter cases of dish names are not consistent. For example, “Apple Sauce” is a titlecase but “Cream of Cauliflower” is an uppercase as below.


423397 rows

Show as: **rows** records Show: 5 10 25 50 **100** 500 1000 r

All			id	name	description	menus_appeared
		35.	38	Apple Sauce		721
		36.	39	Potage a la Victoria		5
		37.	40	Breakfast		28
		38.	41	Strawberries		304
		39.	42	Preserved figs		257
		40.	43	BLUE POINTS		868
		41.	44	CONSOMME ANGLAISE		5
		42.	45	CREAM OF CAULIFLOWER		18
		43.	46	BROILED SHAD, A LA MAITRE D'HOTEL		6

To easily compare the cleaned column with original column data, we add a new column based on the name column renamed as name_case. Since the majority of name data is titlecase, we transform the whole name_case column to titlecase. All the letter cases of dish names are consistent after this data cleaning step. To be specific, the following two data cleaning steps are performed:

- Add a new column based on the name column, name the new column as name_case
- Transform the name_case column to Titlecase

Also, there are many unnecessary characters, such as question marks, quotation marks, parentheses, and square brackets, which will not only confuse the users but also affect the result of use case. After adding a new column based on name_case column, we use GREL to clean these special characters:

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

name_case	name_cleaned
Jam & Momialade (marmalade?)	Jam & Momialade marmalade
[g?] Plaumen U. Reis	g Plaumen U Reis
Cheese Cakes ?	Cheese Cakes
Egg Ala ?	Egg Ala
?	
Fruits & Deporto?	Fruits & Deporto
Juisdine? Truffes	Juisdine Truffes
Truite Au Bleu Sce L??	Truite Au Bleu Sce L
Ginur Ha?	Ginur Ha

Finally, we use different clustering methods to group similar names together because there are many inconsistencies in Name column data because of misspellings, word swapping, and non-standardized value formatting. For example, “Broiled Chicken Half” should be the same as “Broiled Half Chicken” and “Half Broiled Chicken”. Due to the size of inconsistent data, we cluster and merge the data multiple times using the following steps:

- Cluster and Edit column, method: Key-Collision, keying function: fingerprint

Cluster & Edit column "name_cleaned"

This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strin york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same pe

Method: key collision   Keying Function: fingerprint

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
16	40	• Imported Ginger Ale C & C (12 rows) • Ginger Ale Imported C & C (8 rows) • C & C Imported Ginger Ale (4 rows) • Ginger Ale C & C imported (2 rows) • Ginger Ale Imported C &c (2 rows) • Imported Ginger Ale c & C (2 rows) • C & C Ginger Ale Imported • C C Imported Ginger Ale • Ginger Ale C & C Imported • Ginger Ale c & C Imported • Ginger Ale imported C & C • Imported C & C Ginger Ale • Imported Ginger Ale c C • Imported c & C Ginger Ale • Imported c &c Ginger Ale • c & C Ginger Ale Imported	☑	Imported Ginger Ale C & C
13	39	• Broiled Chicken half (11 rows) • Broiled Chicken Half (5 rows) • Chicken Broiled Half (4 rows) • Half Broiled Chicken (4 rows) • Half Chicken Broiled (3 rows) • Broiled Half Chicken (2 rows) • Chicken Broiled half (2 rows)	☑	Broiled Chicken half

- Cluster and Edit column, method: Key-Collision, keying function: ngram-fingerprint, Ngram Size 2

Cluster & Edit column "name_cleaned"

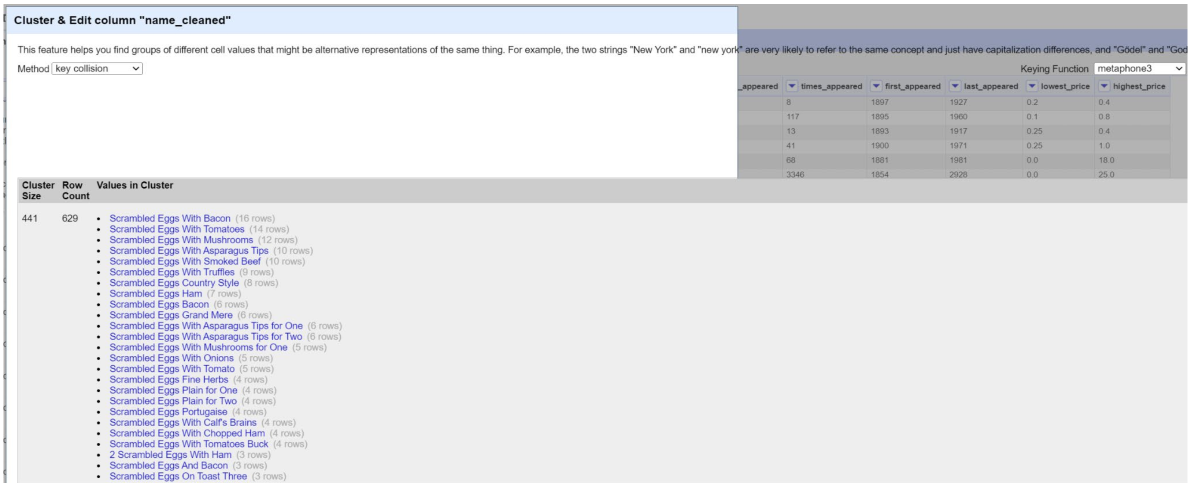
This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same person. Find out more

Method: key collision   Keying Function: ngram-fingerprint   Ngram Size: 2

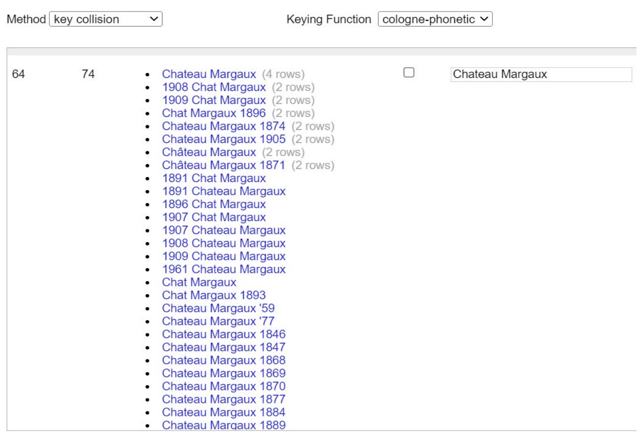
Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
2	3	• Campari 4 Cl (2 rows) • Campari 4cl Browse this cluster	☑	Campari 4 Cl
2	2	• Tournedos Of Fillet St Valier • Tournedos Of Fillet St Vallier	☑	Tournedos Of Fillet St Valier
2	2	• Blue Points On Half Shell Doz • Bluepoints On Half Shell Doz	☑	Blue Points On Half Shell Doz
2	6	• Won ton Soup (4 rows) • Wonton Soup (2 rows)	☑	Won ton Soup
2	2	• King William High Ball • King William Highball	☑	King William High Ball
2	2	• Vosne Romanee Malconsorts 1961 Moillard • Vosnee Romanee Malconsorts 1961 Moillard	☑	Vosne Romanee Malconsorts 19
2	4	• Deutsche Brat Kartoffeln (3 rows) • Deutsche Bratkartoffeln	☑	Deutsche Brat Kartoffeln

- Cluster and Edit column, method: Key-Collision, keying function: Metaphone3 (Cluster size > 200)

When the keying function is Metaphone3, we see a lot of long dish names like below. For example, there are so many different types of scrambled eggs. We consider “Scrambled Eggs With Tomatoes” and “Scrambled Eggs With Bacon” are all Scrambled Eggs, so we cluster these different types of scramble eggs together as “Scramble Eggs”.



When the keying function is cologne-phonetic, we see various wine names such as Chateau Margaux and decide not to cluster these wine names. The cluster size is quite small (<100), so they will not be selected from our main use cases even though we clean them.

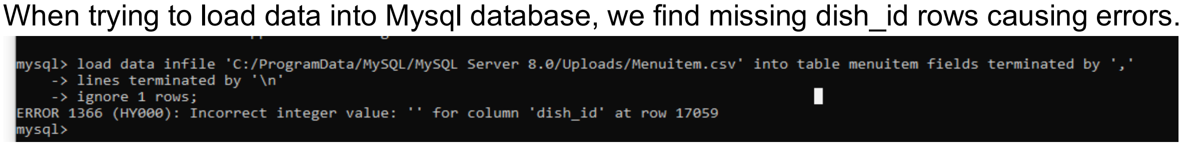


When we switch the clustering method to the nearest neighbor, it takes too long to obtain any cluster due to the size of the dataset. Thus, we only use the Key-Collision clustering method.

2.3 Python Data Cleaning

Further data problems occur when importing data to the database, so we decide to use Python to clean. Codes and queries are submitted in the appendix.

When trying to load data into Mysql database, we find missing dish_id rows causing errors.



Similarly, in the Menupage table, full_height & full_width fields have NaN values, and

Page_number field has NaN values.

```
    -> ignore 1 rows;
ERROR 1264 (22003): Out of range value for column 'image_id' at row 9
mysql> load data infile 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/MenuPage.csv' into table menupage fields terminated by ','
    -> lines terminated by '\n'
    -> ignore 1 rows;
ERROR 1366 (HY000): Incorrect integer value: '' for column 'full_height' at row 9
mysql> load data infile 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/Cleaned_Menupage.csv' into table menupage fields terminated by ','
    -> lines terminated by '\n'
    -> ignore 1 rows;
ERROR 1366 (HY000): Incorrect integer value: 'ps_rbk_637' for column 'image_id' at row 13943
mysql> load data infile 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/Cleaned_Menupage.csv' into table menupage fields terminated by ','
    -> lines terminated by '\n'
    -> ignore 1 rows;
ERROR 1366 (HY000): Incorrect integer value: '' for column 'page_number' at row 34097
mysql> load data infile 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/Cleaned_Menupage.csv' into table menupage fields terminated by ','
    -> lines terminated by '\n'
    -> ignore 1 rows;
Query OK, 65735 rows affected (0.56 sec)
Records: 65735  Deleted: 0  Skipped: 0  Warnings: 0
```

- Using pandas fill NaN in full_height & full_width default to ‘0’ and drop the rows whose page_number is NAN.

For the Menu table, there are 9358 rows missing Venue value.

- Drop venue fields = NaN since this will be used in the main use case.

Additionally, we use Mysql Workben to do further data cleaning for the venue field of the Menu table for use case 2 where the venue names need to be as clean as possible. Therefore, we run the following query to aggregate them to same venue names:

- update menu set venue = 'HOTEL' where venue like 'HOTEL%';
- update menu set venue = 'SOC' where venue like 'SOC%';
- update menu set venue = 'PROF' where venue like 'PROF%';
- update menu set venue = 'COM' where venue like 'COM%';
- update menu set venue = 'EDU' where venue like 'EDU%';
- update menu set venue = 'FOREIGN' where venue like 'FOREIGN%';
- update menu set venue = 'MIL' where venue like 'MIL%';
- update menu set venue = 'POL' where venue like 'POL%';

2.4 Integrity Constraint Violation Checks

2.4.1 Raw Data ICV check

Dish:

1. Check if ID is unique and has no null value

```
#Dish table Validation
#Check if ID is unique and no null value
print("id column contiains",dish['id'].isnull().sum(),"null values")
print("id column contains", sum(dish['id'].duplicated()),"duplicate values")
#First_appeared should be smaller than last_appeared.
dish[dish['first_appeared']>dish['last_appeared']]
```

```
id column contiains 0 null values
id column contains 0 duplicate values
```

2. First_appeared should be smaller than last_appeared.

```
#First_appeared should be smaller than last_appeared.
dish[dish['first_appeared']>dish['last_appeared']]
```

```
id column contiains 0 null values
id column contains 0 duplicate values
```

	id	name	description	menus_appeared	times_appeared	first_appeared	last_appeared	lowest_price	highest_price
131193	164029	Clear beef broth	NaN	0	1	1900	0	0.25	0.25
163257	204888	Hot roast beef with gravy	NaN	0	1	1900	0	0.25	0.25
197049	250693	SURI LEBERLI - Shredded Calf's Liver Flambe in...	NaN	0	1	1945	0	NaN	NaN
197052	250699	SWISS MINCED VEAL, ROESTI	NaN	0	1	1945	0	NaN	NaN
237739	301736	Cafe Glacee	NaN	0	2	1940	0	0.40	0.40
244533	309629	Garlic Butter	NaN	0	1	1947	0	0.40	0.40

Return 6 records. The validation is caused by missing values in last_appeared field.

3. Lowest_price should be smaller than highest_price.

```
dish[dish['lowest_price']>dish['highest_price']]
```

	id	name	description	menus_appeared	times_appeared	first_appeared	last_appeared	lowest_price	highest_price

Return 0 records. No violation is detected.

Menu:

1. Check if ID is unique and has no null value.

```
#Menu table validation
print("id column contiains",menu['id'].isnull().sum(),"null values")
print("id column contains", sum(menu['id'].duplicated()),"duplicate values")
```

```
id column contiains 0 null values
id column contains 0 duplicate values
```

Return 0 records.

2. Check venue field with a distinct select statement, if there are any invalid values.

There are some venue fields with similar venue value:

- HOTEL, FOR and HOTEL, RESTAURANT
- SOC%

Limit to 1000 rows

```
select distinct venue from menu
where venue like 'SOC%'
order by venue
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

venue
SOC
SOC, COM
SOC, MIL
SOC,GK
SOC,POL
SOC,RELIG
SOCIAL
SOCIAL CLUB

- PROF%

PRO
PROF
PROF, COM
PROF,SOC
PROG

- COM%

COM, POL
COM,SOC
COMMERCIAL

Menuitem:

1. Check if ID is unique and has no null value.

```
#MenuItem table validation
print("id column contains",menu_item['id'].isnull().sum(),"null values")
print("id column contains", sum(menu_item['id'].duplicated()),"duplicate values")
```

```
id column contains 0 null values
id column contains 0 duplicate values
```

2. Check dish_id and menu_page_id is null since dish_id and menu_page_id is a foreign key.

```
print("dish_id column contains",menu_item['dish_id'].isnull().sum(),"null values")
print("menu_page_id column contains",menu_item['menu_page_id'].isnull().sum(),"null values")
```

```
dish_id column contains 241 null values
menu_page_id column contains 0 null values
```

MenuPage:

1. Check if ID is unique and no null value

```
#MenuPage table validation
print("id column contiains",menu_page['id'].isnull().sum(),"null values")
print("id column contains", sum(menu_page['id'].duplicated()),"duplicate values")
```

```
id column contiains 0 null values
id column contains 0 duplicate values
```

2. Check menu_id field has no null value

```
print("id column contiains",menu_page['menu_id'].isnull().sum(),"null values")
```

```
id column contiains 0 null values
```

2.4.2 Cleaned Data ICV Check

Dish:

1. Check if ID is unique and has no null value
   select * from dish

where id is null;
Return 0 records;

select id,count(id) from dish
group by id
having count(id) > 1;

Return 0 records.

2. First_appeared should be smaller than last_appeared.
select * from dish
where first_appeared > last_appeared;

Returned 0 records.

3. Lowest_price should be smaller than the highest price.
select * from dish
where lowest_price > highest_price;

Return 0 records.

Menu:

1. Check if ID is unique and has no null value
select * from menu
where id is null;

select id,count(id) from menu
group by id
having count(id) > 1;

Both queries return 0 records.

2. Check venue field with a distinct select statement to see if there are any invalid values.

```
menu_cleaned['venue'].value_counts()

COM                                  5012
SOC                                   656
PROF                                  438
RESTAURANT                            195
GOVT                                  169
EDU                                   157
HOTEL                                 124
OTHER                                 107
RAILROAD                              102
NAV                                   102
PATR                                  102
POL                                    92
MIL                                    88
STEAMSHIP                              52
PAT                                    35
FOREIGN                                28
GOV                                    27
RELIG                                  26
SS, FOR                                21
GREEK LETTER FRATERNITY OR SORORITY    16
AIRLINE                                14
REL                                    10
PRIVATE                                 6
CLUB                                    4
NAVAL                                   3
PATRIOTIC                               2
PRO                                     2
DOM                                     1
PROG                                    1
REPORTERS OF EVENT                      1
INDIVIDUAL                              1
POSSIBLY A PRIVATE HOST                 1
RESORT                                  1
ALUMNI                                  1
CULTURAL                                1
UNKNOWN                                 1
GK                                      1
MUSICAL                                 1
NAC                                     1
```

Menuitem:

1. Check if ID is unique and has no null value
   ```
   select * from menuitem
   where id is null;

   select id,count(id) from menuitem
   group by id
   having count(id) > 1;
   ```
   Both queries return 0 records.

2. Check dish_id and menu_page_id is null since dish_id and menu_page_id is a foreign key.
   ```
   select * from menuitem
   where dish_id is null;

   select * from menuitem
   where menu_page_id is null;
   ```
   Both queries return 0 records.

MenuPage:

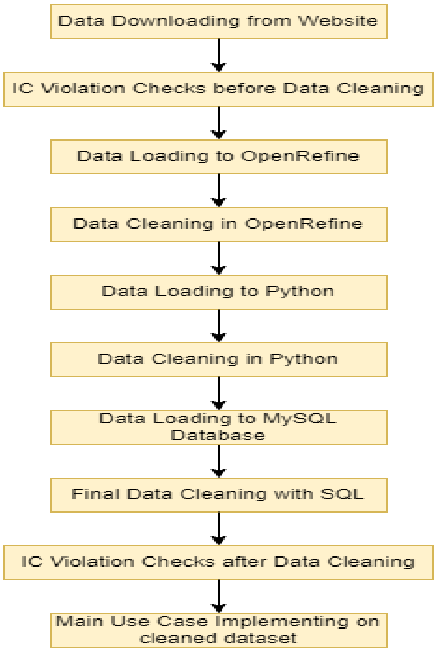
1. Check if ID is unique and has no null value
   ```
   select * from menuPage
   where id is null;

   select id,count(id) from menuPage
   group by id
   having count(id) > 1;
   ```
   Both queries return 0 records.

2. Check menu_id is unique and has no null value
   ```
   select * from menupage
   where menu_id is null;
   ```
   Returned 0 records.

3. Workflow Model

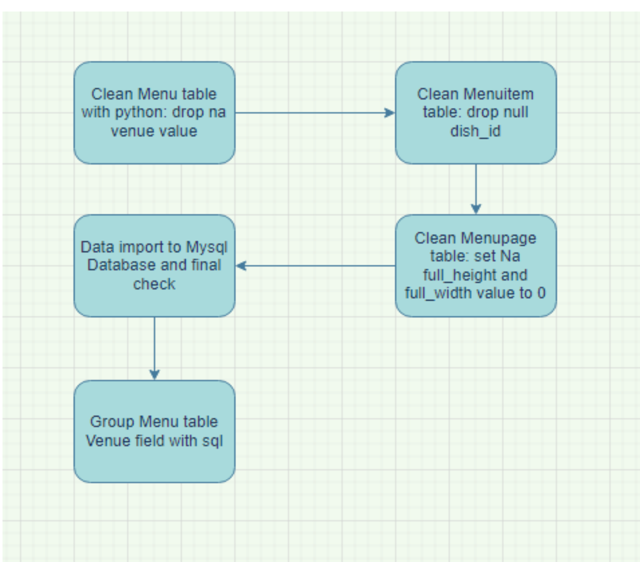
3.1 Outer Workflow



3.2 Inner Workflow

Workflows for steps operated in Openrefine are attached as supporting documents. Please refer to dish.pdf and menu.pdf.

Further Python data cleaning inner workflow as below:



4. Use Case Implementation

4.1 Use Case 1

Query result for Use Case 1: partition the result by 10 years and order by top 10 dish count. For better readability, the following screenshot only lists the top 3 most popular dishes before 1970. Full data result submitted with name usecase1.csv.

year_range	dish_id	dish_name	dish_count	Rank_in_ten_years
<1890	96	Coffee	5	1
	232	Roast Beef	4	2
	1544	Roast Chicken	4	2
	4320	Boiled Tongue	4	2
	2097	Boiled Ham	4	2
	163	Charlotte Russe	4	2
1880--1890	96	Coffee	67	1
	1177	Olives	56	2
	1215	Fruits	53	3
1890--1900	96	Coffee	681	1
	1177	Olives	474	2
	112	Fruit	354	3
1900--1910	96	Coffee	3276	1
	97	Tea	1760	2
	1177	Olives	1411	3
1910--1920	96	Coffee	56	1
	97	Tea	41	2
	808	Kaffee	26	3
1920--1930	1099	Ham	8	1
	96	Coffee	6	2
	15	Celery	6	2
1930--1940	97	Tea	55	1
	96	Coffee	55	1
	98	Milk	43	3
1940--1950	96	Coffee	53	1
	98	Milk	53	1
	97	Tea	44	3
1950--1960	96	Coffee	34	1
	97	Tea	24	2
	136438	Chef's Salad	14	3
1960--1970	96	Coffee	42	1
	98	Milk	40	2
	97	Tea	36	3

4.2 Use Case 2

Use case two is to find out most 10 frequent venue menus and for each venue its most popular top 3 dishes.

venue	dish_name	cnt
COM	Coffee	3174
	Tea	1865
	Potatoes Mashed	1414
EDU	Coffee	82
	Olives	81
	Radishes	48
GOVT	Coffee	56
	Olives	38
	Dessert	34
HOTEL	Coffee	41
	Tea	32
	Assorted Cold Cuts	32
NAV	Coffee	77
	Fruit	41
	Bread	40
OTHER	Coffee	38
	Celery	31
	Olives	30
PROF	Coffee	206
	Olives	203
	Cigars	163
RAILROAD	Coffee	111
	Milk	94
	Tea	93
RESTAURANT	Coffee	61
	Tea	50
	Milk	40
SOC	Olives	312
	Coffee	304
	Celery	269

5. Conclusion

5.1 Key Findings

By implementing the main use case, we find that in 1890-1910, coffee is the most popular dish on the menu. Following frequently appeared dishes are olives, fruit and tea.
For commercial venues, the most popular dish is coffee as well. Tea ranks second followed by mashed potatoes. Olive is the second most popular dish for most of the venues. We can infer that back in history, people loved olives and coffee. Coffee is still a very popular dish in many restaurants nowadays, but we seldom see the figure of olive. It’s a good example to tell the transition of people's preference on food and a micro of economy development.

5.2 Problem Encountered

Initial use case plans to utilize both event and venue columns. However, when checking the event column, a lot of problems occurred and we ended up using the venue column only. On the one hand, there are too many categories in the event column (1769 labels) which makes it hard to cluster. On the other hand, categories in the event column are extremely granular which further adds difficulty in cleaning procedure.

When using Openrefine to clean the venue column in the Manu table and name column in the Dish table, the text facet is not able to cluster all the labels very efficiently. The dataset still needs to be processed manually in other places. We further deal with this problem in python and combine other labels based on expert judgment.

5.2 Lessons Learned

Uncleaned dirty data results in wrong decision making and resource wasting. Data Cleaning is a vital step for any analysis, decision making and prediction.
This project offered a good opportunity for our team to dig deeper into the data clean process, understand how important data cleaning is, and apply some useful tools, OpenRefine, SQL and Pandas.
Some advantages and disadvantages of the tools we have been experienced:

1. OpenRefine is a useful tool for data cleaning: it offers a clear overview of data, efficiently resolving inconsistencies in a data set.
2. Relational database is efficient for checking the data quality and implementing the final result of use cases. It is useful for data accuracy, data integrity and normalization.
3. Limitation of RDBMS:
    - Data schema design is costly and difficult to maintain.
    - Improper data types; insufficient data loading process. Data must fit the column data type to be loaded.
    - Cost and performance of RDBMS: when running complex queries on large scales of data, the interface crashes due to memory issues.
4. Pandas is powerful and fast for data processing and analysis.

5.3 Team Contribution

Fangsheng Yang:

- Cleaned Menu table, dish table and menu_item table, conducted ICV check for cleaned datasets, created inner workflow for steps operated in python, uploaded datasets to database, wrote queries and implement main use case, and completed report

Yilin Hou:

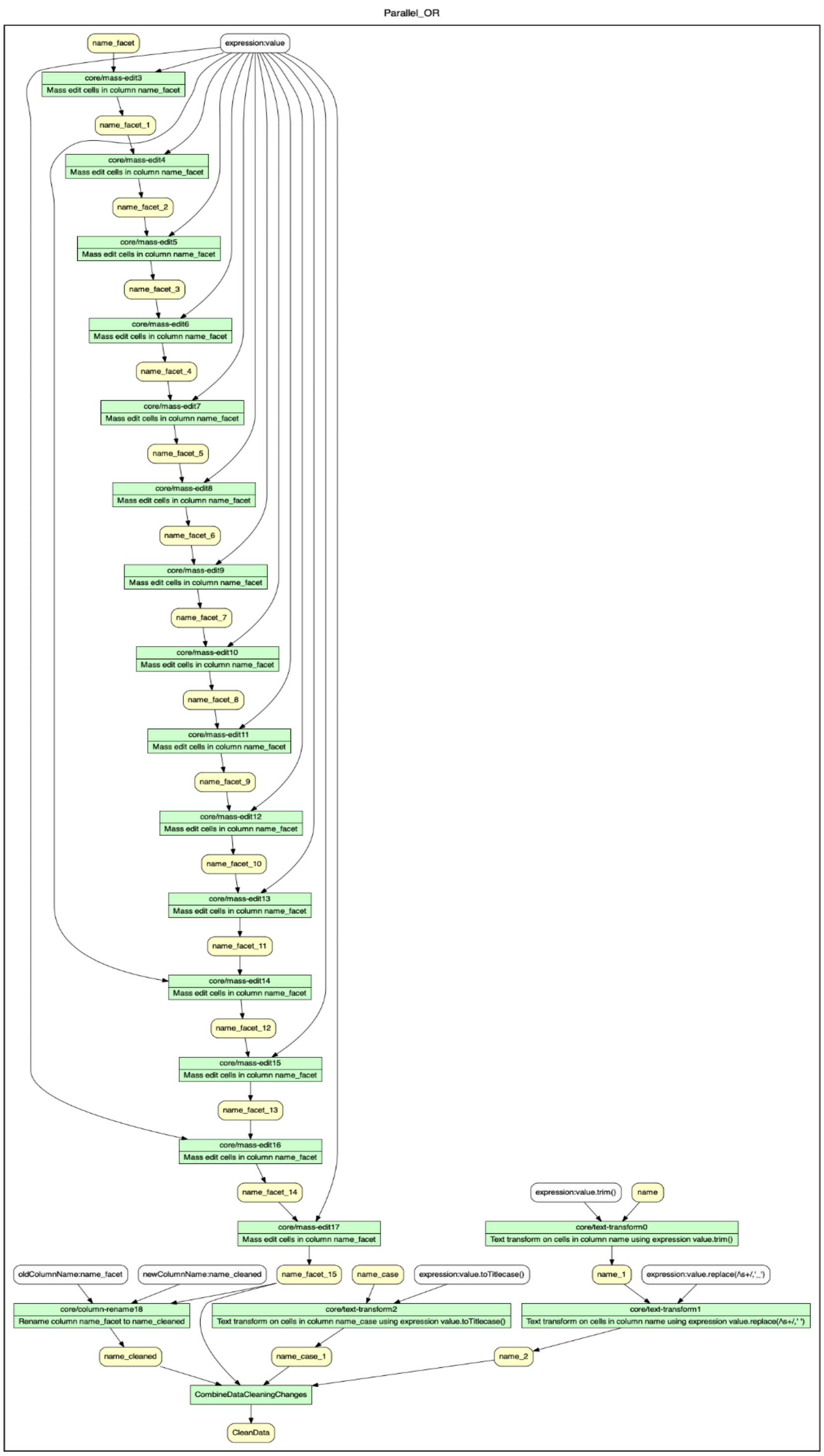
- Cleaned Menu table, created inner workflow for steps operated in openrefine, conducted ICV check for raw datasets, and completed report

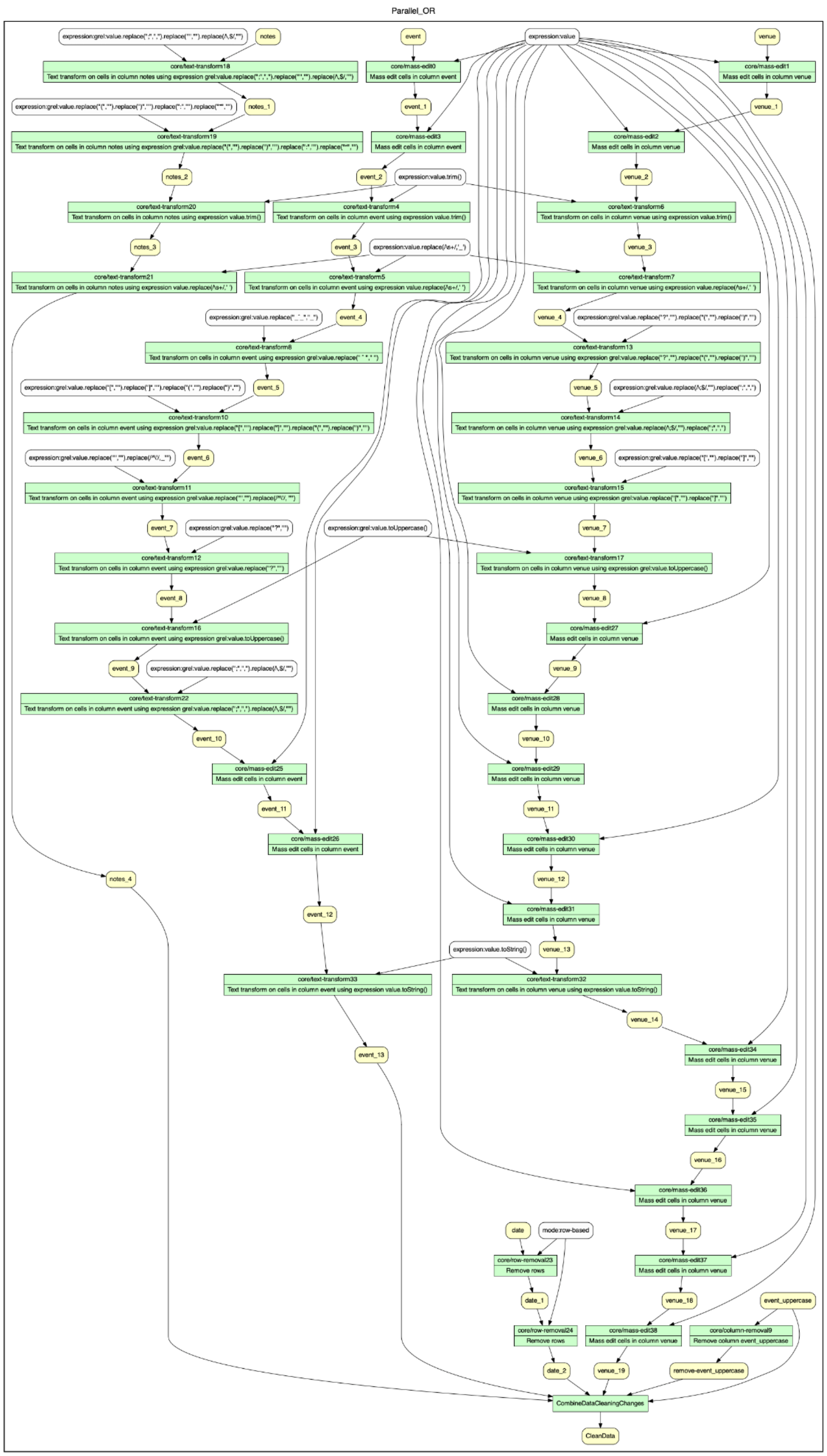
Candice Yan:

- Cleaned Dish table, created outer workflow for overall steps we performed, organized and completed report

6. Appendix

6.1 Inner Workflow for OpenRefine Process




Parallel_OR
expression:value
event
venue
notes
core/mass-edit0
Mass edit cells in column event
core/mass-edit1
Mass edit cells in column venue
core/text-transform18
core/text-transform19
core/text-transform20
Text transform on cells in column notes using expression value.trim()
core/text-transform21
core/text-transform4
Text transform on cells in column event using expression value.trim()
core/text-transform6
Text transform on cells in column venue using expression value.trim()
expression:value.trim()
expression:grel:value.toUppercase()
expression:value.toString()
core/text-transform33
Text transform on cells in column event using expression value.toString()
core/text-transform32
Text transform on cells in column venue using expression value.toString()
date
mode:row-based
core/row-removal23
Remove rows
core/row-removal24
core/column-removal9
Remove column event_uppercase
event_uppercase
remove-event_uppercase
CombineDataCleaningChanges
CleanData

6.2 Query Designed for Use Case 1

Limit to 1000 rows

```
select * from (
select cte3.year_range, cte3.dish_id, cte3.name as dish_name, cte3.dish_count,
rank() over(partition by cte3.year_range order by cte3.dish_count desc) as Rank_in_ten_years
from
(
select  cte2.year_range, cte2.dish_id, cte2.name,

count(cte2.dish_id) as dish_count
from
(
SELECT
    cte.year_range,cte.id as menu_id,  d.name, d.id as dish_id
FROM
    menupage p
        JOIN
    menuitem i ON p.id = i.menu_page_id
        JOIN
    dish d ON d.id = i.dish_id
        JOIN
    (SELECT
        id,
            date,
            (CASE
                WHEN date < 1880 THEN '<1890'
                WHEN date >= 1880 AND date < 1890 THEN '1880--1890'
                WHEN date >= 1890 AND date < 1900 THEN '1890--1900'
                WHEN date >= 1900 AND date < 1910 THEN '1900--1910'
                WHEN date >= 1910 AND date < 1920 THEN '1910--1920'
                WHEN date >= 1920 AND date < 1930 THEN '1920--1930'
                WHEN date >= 1930 AND date < 1940 THEN '1930--1940'
                WHEN date >= 1940 AND date < 1950 THEN '1940--1950'
                WHEN date >= 1950 AND date < 1960 THEN '1950--1960'
                WHEN date >= 1960 AND date < 1970 THEN '1960--1970'
                WHEN date >= 1970 AND date < 1980 THEN '1970--1980'
                WHEN date >= 1980 AND date < 1990 THEN '1980--1990'
                WHEN date >= 1990 AND date < 2000 THEN '1990--2000'
                WHEN date >= 2000 THEN '2000 AND AFTER'
            END) AS year_range
    FROM
        menu) cte ON p.menu_id = cte.id

) cte2
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

year_range	dish_id	dish_name	dish_count	Rank_in_ten_years
1880--1890	96	Coffee	67	1
1880--1890	1177	Olives	56	2
1880--1890	1215	Fruits	53	3
1880--1890	2450	Cafe	49	4
1880--1890	3436	Petits Fours	43	5
1880--1890	4434	Raisins	34	6
1880--1890	1052	Dessert	34	6
1880--1890	112	Fruit	32	8
1880--1890	15	Celery	30	9
1880--1890	13950	Huitres	28	10
1890--1900	96	Coffee	681	1
1890--1900	1177	Olives	474	2
1890--1900	112	Fruit	354	3
1890--1900	15	Celery	352	4
1890--1900	7	Radishes	339	5
1890--1900	795	Cheese	237	6
1890--1900	83	Potatoes Mashed	220	7
1890--1900	1215	Fruits	218	8
1890--1900	1763	Cucumbers	218	8
1890--1900	2450	Cafe	216	10
1900--1910	96	Coffee	3276	1
1900--1910	97	Tea	1760	2

Result 36

6.3 Query Designed for Use Case 2

```sql
select f.venue, f.dish_name, f.cnt from
(
select
top_menu_dish.venue, top_menu_dish.name as dish_name, top_menu_dish.cnt,
rank() over(partition by top_menu_dish.venue order by top_menu_dish.cnt desc) as dish_rnk
from
(
SELECT
    m.venue, d.name, count(d.name) as cnt

FROM
    menupage p
        JOIN
    menuitem i ON p.id = i.menu_page_id
        JOIN
    dish d ON d.id = i.dish_id
        JOIN
    menu m ON m.id = p.menu_id
        JOIN
    (SELECT
        m.venue
    FROM
        menu m
    GROUP BY m.venue
    ORDER BY COUNT(m.id) DESC
    LIMIT 10) top_menu ON m.venue = top_menu.venue
    group by m.venue, d.name
) top_menu_dish
) f
where f.dish_rnk <= 3
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

venue	dish_name	cnt
COM	Coffee	3174
COM	Tea	1865
COM	Potatoes Mashed	1414
EDU	Coffee	82
EDU	Olives	81
EDU	Radishes	48
GOVT	Coffee	56
GOVT	Olives	38
GOVT	Dessert	34
HOTEL	Coffee	41
HOTEL	Tea	32
HOTEL	Assorted Cold Cuts	32
NAV	Coffee	77
NAV	Fruit	41
NAV	Bread	40
OTHER	Coffee	38
OTHER	Celery	31
OTHER	Olives	30
PROF	Coffee	206
PROF	Olives	203
PROF	Cigars	163
RAIL...	Coffee	111
RAIL...	Milk	94
RAIL...	Tea	93
REST...	Coffee	61
REST...	Tea	50
REST...	Milk	40