CS 513: Theory & Practice of Data Cleaning Final Project
University of Illinois at Urbana-Champaign - Summer 2023
Phase 2 Report

Purpose

The purpose of this report is to summarize the steps that were performed to clean and organize the Chicago Food Inspection dataset. After performing data profiling, quality assessment, and quality improvement, the main use case mentioned in the initial assessment can be implemented. We have designed the dataset in such a way that additional analytics can also be performed.

Team Details

Team-ID: 120
Members:

- Jeremy Ahn (jcahn2@illinois.edu)
- Nithin Nathan (nnatha3@illinois.edu)
- Ratul Saha (ratuls2@illinois.edu)

Table of Contents

Description of Data Cleaning Performed

The provided dataset contains inspection details of around 24,000 facilities located in Chicago over the course of 8 years (2010-2017). The business details contain the business name, legal name, address details, and facility type. Location is identified by address, city, state, zip code, latitude and longitude. Inspection details capture inspection date, observation, risk type and result.

We are arranged the data of inspection details into 3 files, as described below:

1. Business: This file contains business identifiers, business names and facility type for every business listed. We will consider this as a master dataset.
2. Location: This file contains all the locations and their associated details, irrespective of which facility or what business is operating from there. This file contains geographical location and address details. We will consider this as another master dataset.
3. Inspection: This file contains all the inspection details performed on the businesses. We will consider this file as a relationship/fact dataset.

At the first level of data cleaning, we cleaned the following fields using OpenRefine:

Inspection ID: Unique ID for each inspection case performed on a business at a location. This is already cleaned and contains unique values pertaining to an unique inspection event. We did not perform any cleaning on this field.

Doing Business As (DBA) Name: The name of a business doing business as or also known as legal/registered name. We derived a new variable 'DBA Name - Clean' from 'DBA Name'.

- Cleaning leading and trailing spaces, convert all the text into uppercase, collapse multiple spaces into a single space, remove any special characters that were not part of the Basic Latin set
  - Cleaning Step: This step was performed by using `value.trim().toUppercase().replace(/[^\u0020-\u007F]/,"").replace(" "," ")`
  - Rationale: Leading or trailing spaces can interfere with clustering text values into the correct cluster.
  - Useful / Required for use case U1: When establishing the analysis against U1, identifying the business properly by its name is necessary. This also ensures that business license # is uniquely paired with the facility DBA name. Uppercase helps in better matching. Special characters interfere with text matching.

Also Known As (AKA) Name: Common name of the business. We found 2543 inspection records where AKA Name was blank. We derived a new variable 'AKA Name - Clean' from 'DBA Name'. We performed the same cleaning we did for DBA Name. Also, when AKA Name is clean, we populated with DBA Name.

- Cleaning leading and trailing spaces, convert all the text into uppercase, collapse multiple spaces into a single space, remove any special characters that were not part of the Basic Latin set

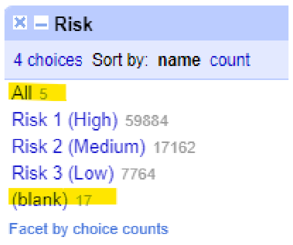
    - Cleaning Step: This step was performed by using value.trim().toUppercase().replace(/[^\u0020-\u007F]/,"").replace(" "," ")
    - Rationale: Leading or trailing spaces can interfere with clustering text values into the correct cluster.
    - Useful / Required for use case U1: When establishing the analysis against U1, identifying the business properly by its name is necessary. This also ensures that business license # is uniquely paired with the facility DBA name. Uppercase helps in better matching. Special characters interfere with text matching.
- Replace with DBA Name when blank:
    - Cleaning Step: This step was performed by using if(value==null,cells['DBA Name - Clean'].value,value)
    - Rationale: Blank values were replaced by an actual value from DBA Name - Clean, which is already cleaned and non-blank.
    - Useful / Required for use case U1: Analysis on this value would be useful since there are no blanks.

<u>License #</u>: Unique licensing number assigned by the Department of Business Affairs and Consumer Protection

- Flag incorrect License # values based on rules: We created a new field ‘License # - Flag’ to mark either ‘Correct’ or ‘Incorrect’. Records with incorrect License # will be cleaned at a later stage of the project.
    - Cleaning Step: This step was performed by using if(value==null,'Incorrect',if(value=='0','Incorrect',if(value.contains("[a-zA-Z~!@#$%^&*()_+\][{}|';:/.,<>?]+") == true,'Incorrect','Correct')))
    - Rationale: License # being the unique identifier of the business, blank , ‘0’ or any other non-numeric values are marked as incorrect.
    - Useful / Required for use case U1: Unique identification of each business is important to perform U1 effectively.
- Cleaning leading and trailing spaces, collapse multiple spaces into a single space, convert values to numbers. We created a new field ‘License # - Clean’
    - Cleaning Step: This step was performed by using value.trim().replace(" "," ").toNumber()
    - Rationale: Leading or trailing spaces can interfere with clustering.
    - Useful / Required for use case U1: When establishing the analysis against U1, identifying the business properly by its unique identifier is necessary.

<u>Risk</u>: Each establishment is categorized as to its risk of adversely affecting the public’s health, with 1 being the highest and 3 the lowest. The frequency of inspection is tied to this risk, with risk 1 establishments inspected most frequently and risk 3 least frequently.
There are 85 records where Risk values are either ‘All’ or blank. These are unrecognized category values as per definition of this dataset. We planned to mark these records as ‘Incorrect’.

- Text Facet: We found 19 records with ‘All’ as Risk, and 66 records with blank as Risk.

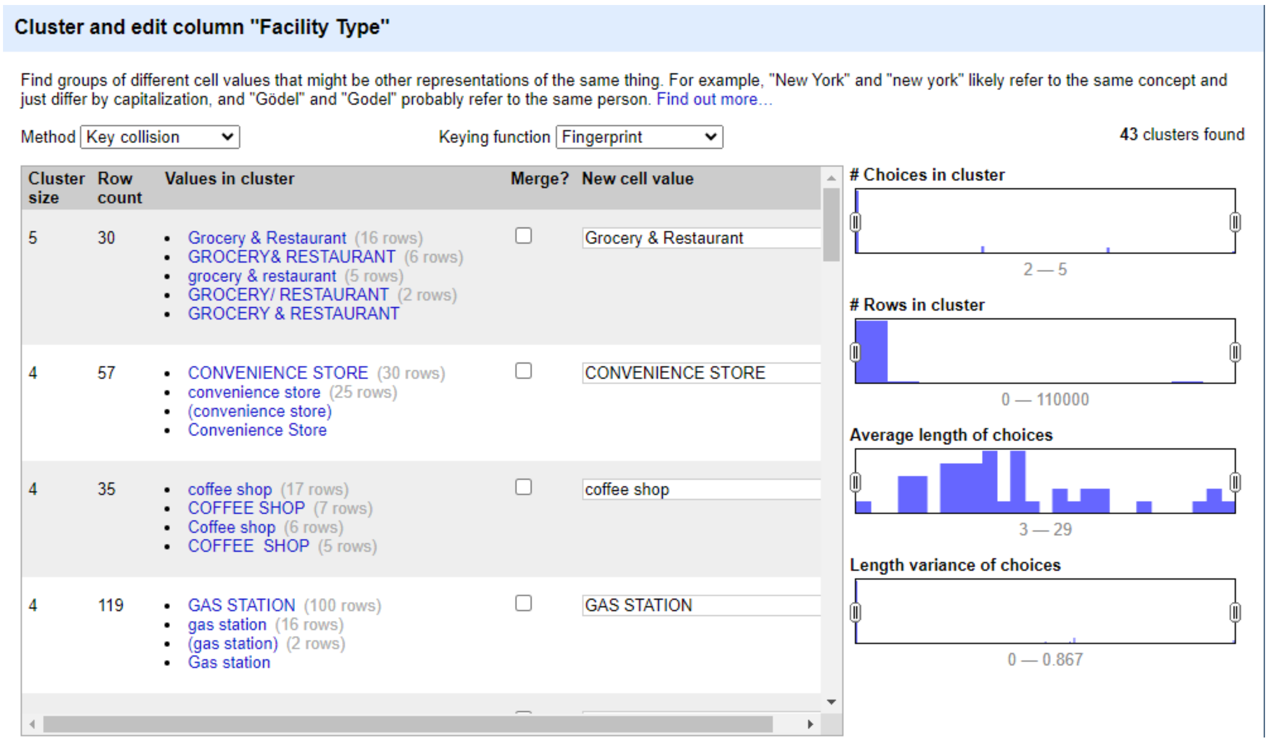

Risk
4 choices Sort by: name count

All	5
Risk 1 (High)	59884
Risk 2 (Medium)	17162
Risk 3 (Low)	7764
(blank)	17

- Flag incorrect Risk values based on rules: We created a new field 'Risk - Flag' to mark either 'Correct' or 'Incorrect'. Records with incorrect Risk values will be cleaned at a later stage of the project.
  - Cleaning Step: This step was performed by using if(value==null,'Incorrect',if(value=='','Incorrect',if(value=='Risk 1 (High)','Correct',if(value=='Risk 2 (Medium)','Correct',if(value=='Risk 3 (Low)','Correct','Incorrect')))))
  - Rationale: any invalid risk value should be discarded, otherwise we can not determine the correct risk factor of the inspection.
  - Useful / Required for use case U1: Only recognized category values can be useful for meaningful analysis.
- Cleaning leading and trailing spaces, convert all the text into uppercase, collapse multiple spaces into a single space, remove any special characters that were not part of the Basic Latin set. We created a new field 'Risk Category - Clean'.
  - Cleaning Step: This step was performed by using value.trim().toUppercase().replace(/[^\u0020-\u007F]/,"").replace(" "," ")
  - Rationale: We wanted to standardize the Risk field values if at all there are any discrepancies.
  - Useful / Required for use case U1: This was not really required since after flagging incorrect values, the data was clean. We did this for best practice.

Facility Type: Type of the business facility.

- Clustering and Merge: When ran the clustering process in the Inspection Type - Clean field, we found the below cluster. We merged the records into a single cluster.
  - Before:



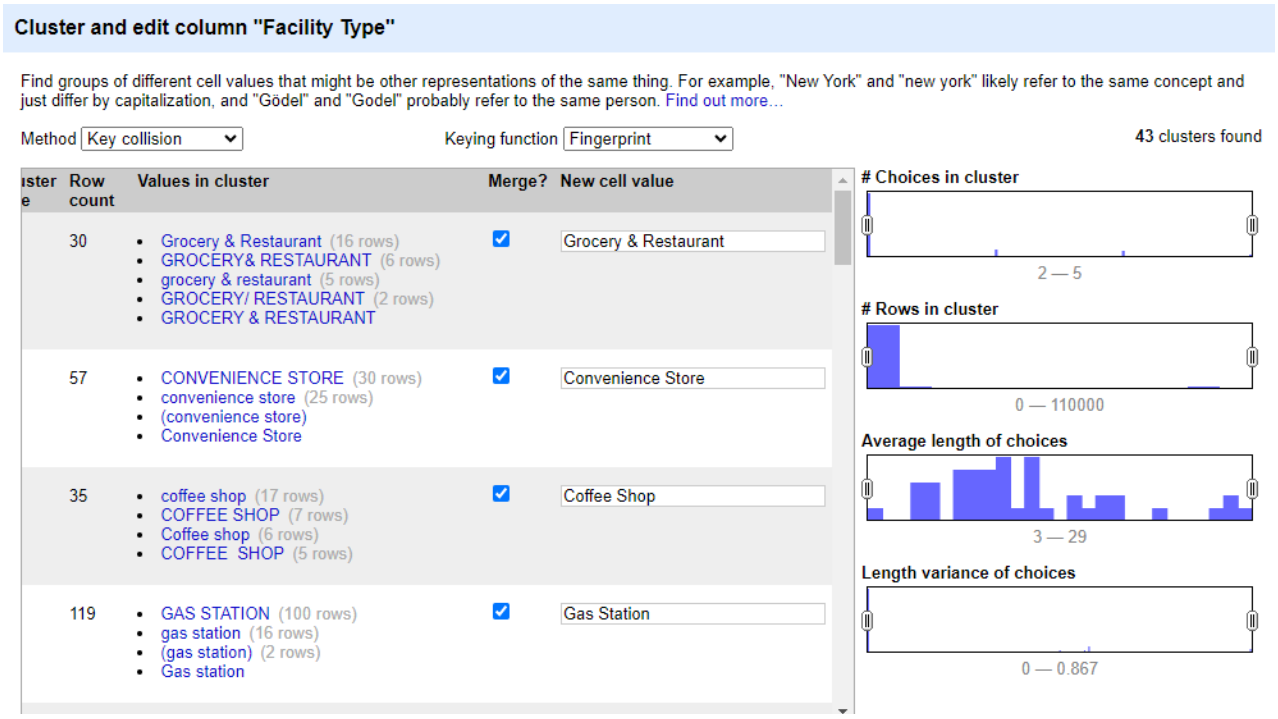
Cluster and edit column "Facility Type"

Find groups of different cell values that might be other representations of the same thing. For example, "New York" and "new york" likely refer to the same concept and just differ by capitalization, and "Gödel" and "Godel" probably refer to the same person. Find out more...

Method: Key collision — Keying function: Fingerprint — 43 clusters found

Cluster size	Row count	Values in cluster	Merge?	New cell value
5	30	Grocery & Restaurant (16 rows); GROCERY& RESTAURANT (6 rows); grocery & restaurant (5 rows); GROCERY/ RESTAURANT (2 rows); GROCERY & RESTAURANT	☐	Grocery & Restaurant
4	57	CONVENIENCE STORE (30 rows); convenience store (25 rows); (convenience store); Convenience Store	☐	CONVENIENCE STORE
4	35	coffee shop (17 rows); COFFEE SHOP (7 rows); Coffee shop (6 rows); COFFEE SHOP (5 rows)	☐	coffee shop
4	119	GAS STATION (100 rows); gas station (16 rows); (gas station) (2 rows); Gas station	☐	GAS STATION

- Clean Clustering Step:



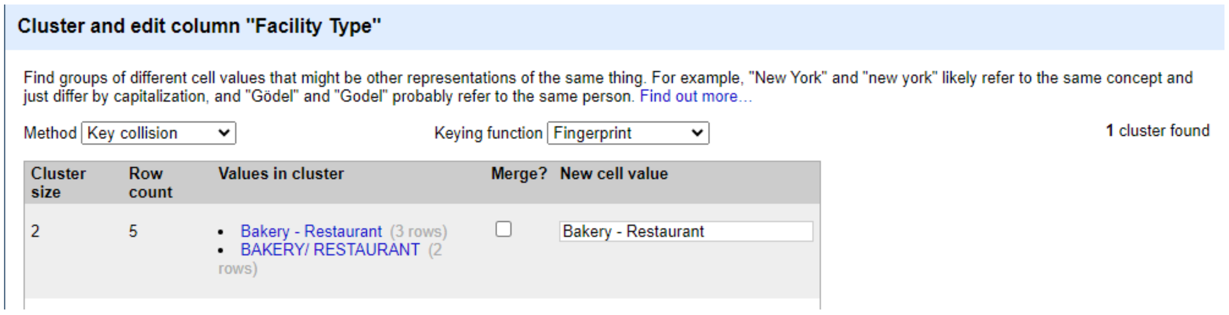
Cluster and edit column "Facility Type"

Find groups of different cell values that might be other representations of the same thing. For example, "New York" and "new york" likely refer to the same concept and just differ by capitalization, and "Gödel" and "Godel" probably refer to the same person. Find out more...

Method: Key collision — Keying function: Fingerprint — 43 clusters found

ıster e	Row count	Values in cluster	Merge?	New cell value
	30	Grocery & Restaurant (16 rows); GROCERY& RESTAURANT (6 rows); grocery & restaurant (5 rows); GROCERY/ RESTAURANT (2 rows); GROCERY & RESTAURANT	✓	Grocery & Restaurant
	57	CONVENIENCE STORE (30 rows); convenience store (25 rows); (convenience store); Convenience Store	✓	Convenience Store
	35	coffee shop (17 rows); COFFEE SHOP (7 rows); Coffee shop (6 rows); COFFEE SHOP (5 rows)	✓	Coffee Shop
	119	GAS STATION (100 rows); gas station (16 rows); (gas station) (2 rows); Gas station	✓	Gas Station

- Reclustering:



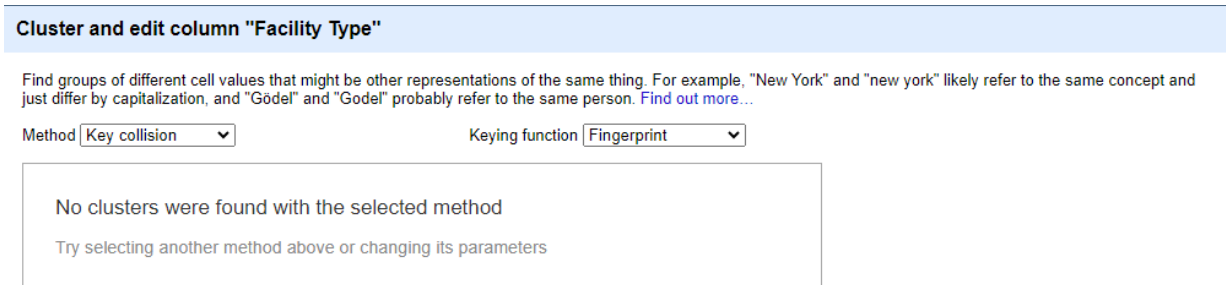
Cluster and edit column "Facility Type"

Find groups of different cell values that might be other representations of the same thing. For example, "New York" and "new york" likely refer to the same concept and just differ by capitalization, and "Gödel" and "Godel" probably refer to the same person. Find out more…

Method: Key collision — Keying function: Fingerprint — 1 cluster found

Cluster size	Row count	Values in cluster	Merge?	New cell value
2	5	• Bakery - Restaurant (3 rows) • BAKERY/ RESTAURANT (2 rows)	☐	Bakery - Restaurant

- After



Cluster and edit column "Facility Type"

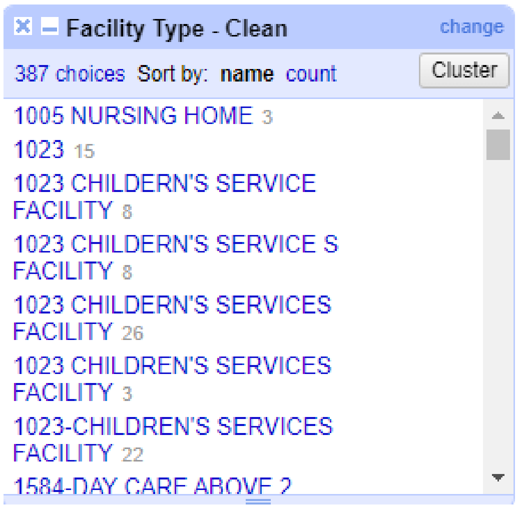
Find groups of different cell values that might be other representations of the same thing. For example, "New York" and "new york" likely refer to the same concept and just differ by capitalization, and "Gödel" and "Godel" probably refer to the same person. Find out more…

Method: Key collision — Keying function: Fingerprint

No clusters were found with the selected method

Try selecting another method above or changing its parameters

  - Rationale: Clustering revealed many different variations of the same text. We used this method to standardize the values for better analysis.
  - Useful / Required for use case U1: Standardization helps to consolidate the values into limited sets which is useful for the analysis.
- Cleaning leading and trailing spaces, convert all the text into uppercase, collapse multiple spaces into a single space, remove any special characters that were not part of the Basic Latin set. We created a new field Facility Type - Clean'.
  - Cleaning Step: This step was performed by using `value.trim().toUppercase().replace(/[^\u0020-\u007F]/,"").replace(" "," ")`
  - Rationale: We wanted to standardize the Risk field values if at all there are any discrepancies.
  - Useful / Required for use case U1: This was not really required since after flagging incorrect values, the data was clean. We did this for best practice.
- Cleaning through Text Facet: We ran the text facet process and updated 387 choices to fewer choices on 'Facility Type - Clean' field.
  - Before:

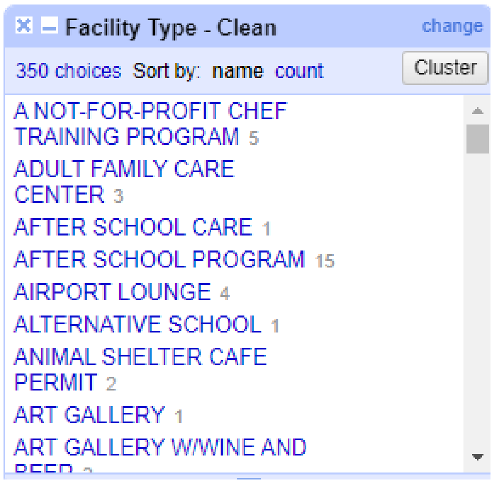


Facility Type - Clean change

387 choices Sort by: name count Cluster

1005 NURSING HOME	3
1023	15
1023 CHILDERN'S SERVICE FACILITY	8
1023 CHILDERN'S SERVICE S FACILITY	8
1023 CHILDERN'S SERVICES FACILITY	26
1023 CHILDREN'S SERVICES FACILITY	3
1023-CHILDREN'S SERVICES FACILITY	22
1584-DAY CARE ABOVE 2	

- After



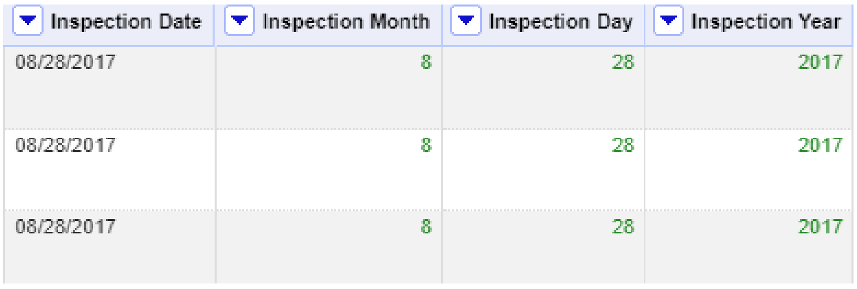
Facility Type - Clean change

350 choices Sort by: name count Cluster

A NOT-FOR-PROFIT CHEF TRAINING PROGRAM	5
ADULT FAMILY CARE CENTER	3
AFTER SCHOOL CARE	1
AFTER SCHOOL PROGRAM	15
AIRPORT LOUNGE	4
ALTERNATIVE SCHOOL	1
ANIMAL SHELTER CAFE PERMIT	2
ART GALLERY	1
ART GALLERY W/WINE AND	

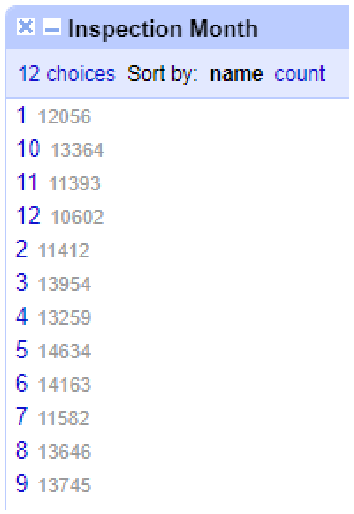
- Rationale: Text facet shows if there are values close enough which we can further standardize that are not captured after clustering.
- Useful / Required for use case U1: Standardization helps to consolidate the values into limited sets which is useful for the analysis.

Inspection Date: Date when an inspection was performed.
We create 3 additional columns from the date field: Month, Day and Year.



Inspection Date	Inspection Month	Inspection Day	Inspection Year
08/28/2017	8	28	2017
08/28/2017	8	28	2017
08/28/2017	8	28	2017

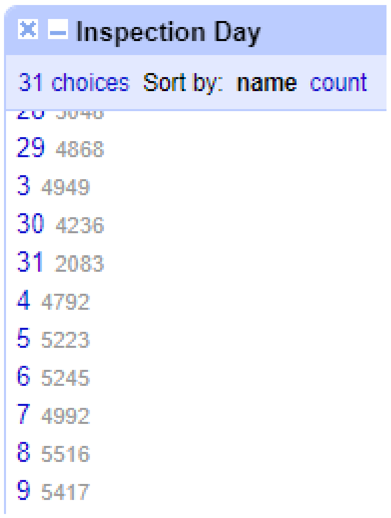
We ran text facets on these 3 newly created fields.



Inspection Month

12 choices Sort by: name count

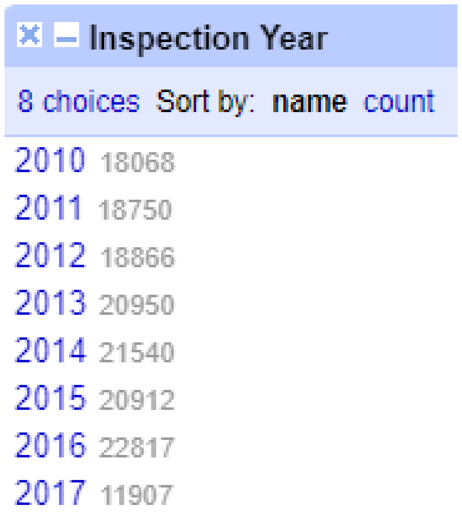
1	12056
10	13364
11	11393
12	10602
2	11412
3	13954
4	13259
5	14634
6	14163
7	11582
8	13646
9	13745



Inspection Day

31 choices Sort by: name count

29	4868
3	4949
30	4236
31	2083
4	4792
5	5223
6	5245
7	4992
8	5516
9	5417



Inspection Year

8 choices Sort by: name count

2010	18068
2011	18750
2012	18866
2013	20950
2014	21540
2015	20912
2016	22817
2017	11907

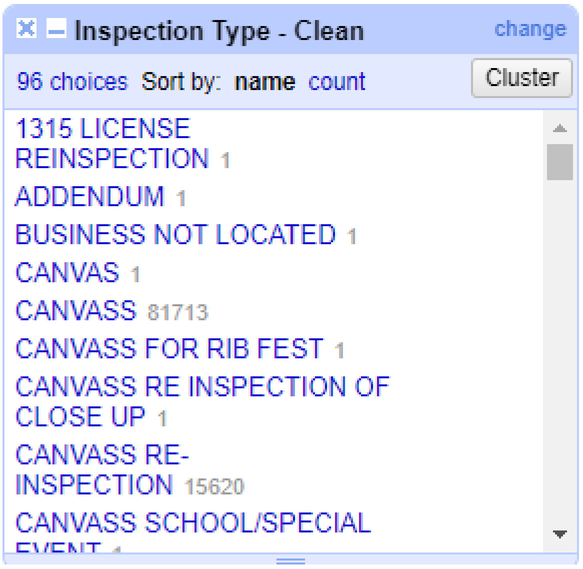
All the date values are clean. We did not need to perform any cleaning for this field.

Rationale: We wanted to make sure all the dates are correct.
Useful / Required for use case U1: Incorrect records can be identified and cleaned for accuracy of our analysis.

Inspection Type: Various purposes of inspection like license, complaint, canvass, expansion etc.

- Cleaning leading and trailing spaces, convert all the text into uppercase, collapse multiple spaces into a single space, remove any special characters that were not part of the Basic Latin set. We created a new field 'Inspection Type - Clean'.
  - Cleaning Step: This step was performed by using value.trim().toUppercase().replace(/[^\u0020-\u007F]/,"").replace(" "," ")
  - Rationale: We wanted to standardize the Risk field values if at all there are any discrepancies.
  - Useful / Required for use case U1: This was not really required since after flagging incorrect values, the data was clean. We did this for best practice.
- We ran the text facet on the newly created 'Inspection Type - Clean' field. We see there are 96 choices and some can still be standardized.

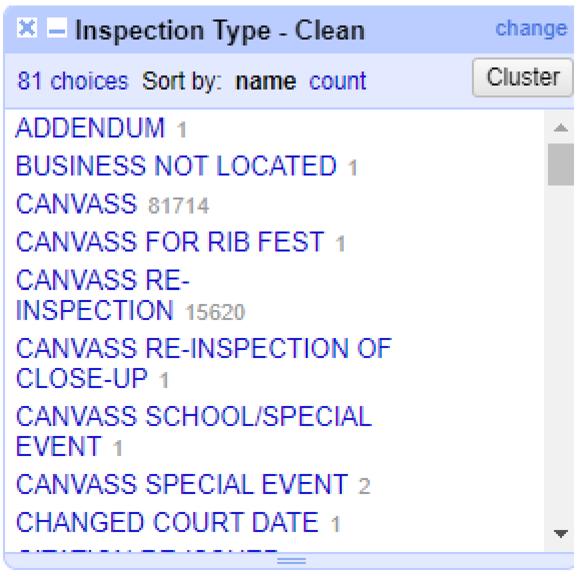
- Before



Inspection Type - Clean change
96 choices Sort by: name count Cluster

1315 LICENSE REINSPECTION	1
ADDENDUM	1
BUSINESS NOT LOCATED	1
CANVAS	1
CANVASS	81713
CANVASS FOR RIB FEST	1
CANVASS RE INSPECTION OF CLOSE UP	1
CANVASS RE-INSPECTION	15620
CANVASS SCHOOL/SPECIAL EVENT	1

- After



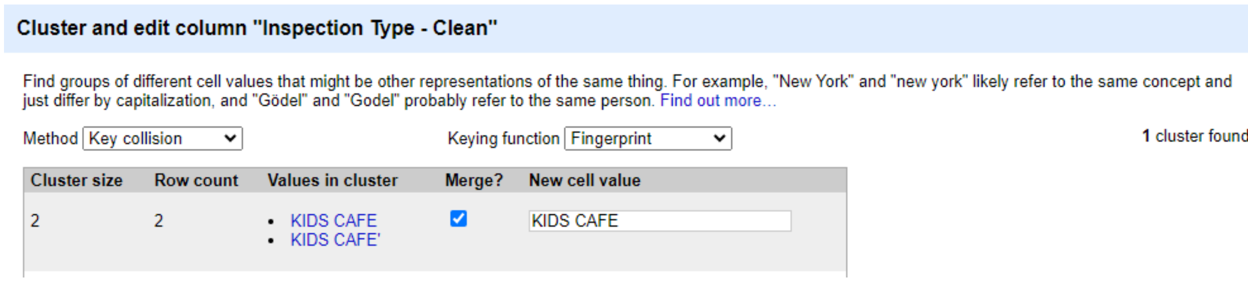
Inspection Type - Clean change
81 choices Sort by: name count Cluster

ADDENDUM	1
BUSINESS NOT LOCATED	1
CANVASS	81714
CANVASS FOR RIB FEST	1
CANVASS RE-INSPECTION	15620
CANVASS RE-INSPECTION OF CLOSE-UP	1
CANVASS SCHOOL/SPECIAL EVENT	1
CANVASS SPECIAL EVENT	2
CHANGED COURT DATE	1

- Rationale: Text facet shows if there are values close enough which we can further standardize that are not captured after clustering.
- Useful / Required for use case U1: Standardization helps to consolidate the values into limited sets which is useful for the analysis.

- Clustering and Merge: When ran the clustering process in the Inspection Type - Clean field, we found the below cluster. We merged the records into a single cluster.

- Before



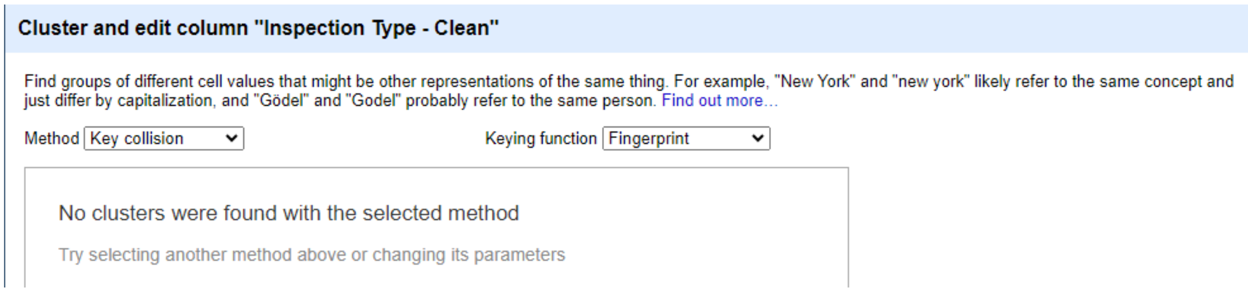
Cluster and edit column "Inspection Type - Clean"

Find groups of different cell values that might be other representations of the same thing. For example, "New York" and "new york" likely refer to the same concept and just differ by capitalization, and "Gödel" and "Godel" probably refer to the same person. Find out more...

Method: Key collision   Keying function: Fingerprint   1 cluster found

Cluster size	Row count	Values in cluster	Merge?	New cell value
2	2	• KIDS CAFE • KIDS CAFE'	✓	KIDS CAFE

- After



Cluster and edit column "Inspection Type - Clean"

Find groups of different cell values that might be other representations of the same thing. For example, "New York" and "new york" likely refer to the same concept and just differ by capitalization, and "Gödel" and "Godel" probably refer to the same person. Find out more...

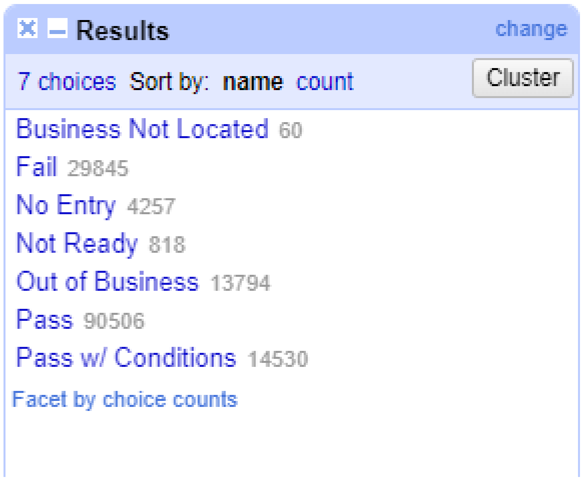
Method: Key collision   Keying function: Fingerprint

No clusters were found with the selected method

Try selecting another method above or changing its parameters

- Rationale: Clustering revealed variations of the same text. We used this method to standardize the values for better analysis.
- Useful / Required for use case U1: Standardization helps to consolidate the values into limited sets which is useful for the analysis.

Results: Final inspection outcome like pass, fail, not ready etc.
The results field data is clean. We did not need to perform any further data cleaning. Below is the text facet for confirmation.



Results   change

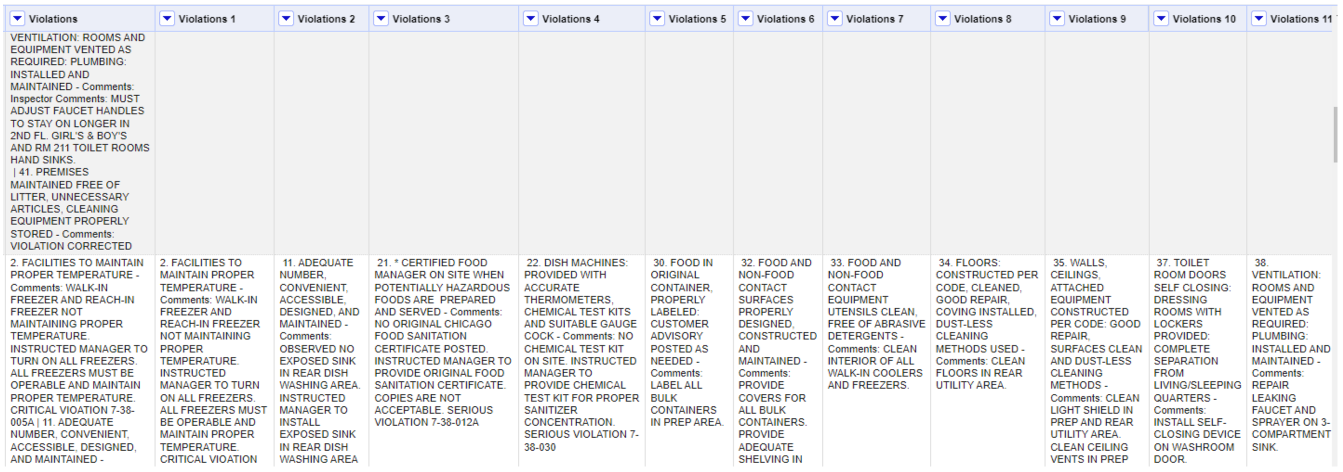
7 choices Sort by: name count   Cluster

Business Not Located	60
Fail	29845
No Entry	4257
Not Ready	818
Out of Business	13794
Pass	90506
Pass w/ Conditions	14530

Facet by choice counts

Rationale: We wanted to make sure all the values are standardized.
Useful / Required for use case U1: Slightly different values can be identified and merged for standardization.

Violations: The textual description of the inspection results. An establishment can receive one or more of 45 distinct violations (violation numbers 1-44 and 70).
Since the Violations field is a single placeholder for multiple violation information, separated by '|' character, we split this field into multiple fields.



Violations	Violations 1	Violations 2	Violations 3	Violations 4	Violations 5	Violations 6	Violations 7	Violations 8	Violations 9	Violations 10	Violations 11
VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED AND MAINTAINED - Comments: Inspector Comments: MUST ADJUST FAUCET HANDLES TO STAY ON LONGER IN 2ND FL. GIRL'S & BOY'S AND RM 211 TOILET ROOMS HAND SINKS. | 41. PREMISES MAINTAINED FREE OF LITTER, UNNECESSARY ARTICLES, CLEANING EQUIPMENT PROPERLY STORED - Comments: VIOLATION CORRECTED											
2. FACILITIES TO MAINTAIN PROPER TEMPERATURE - Comments: WALK-IN FREEZER AND REACH-IN FREEZER NOT MAINTAINING PROPER TEMPERATURE. INSTRUCTED MANAGER TO TURN ON ALL FREEZERS. ALL FREEZERS MUST BE OPERABLE AND MAINTAIN PROPER TEMPERATURE. CRITICAL VIOATION 7-38-005A | 11. ADEQUATE NUMBER, CONVENIENT, ACCESSIBLE, DESIGNED, AND MAINTAINED -	2. FACILITIES TO MAINTAIN PROPER TEMPERATURE - Comments: WALK-IN FREEZER AND REACH-IN FREEZER NOT MAINTAINING PROPER TEMPERATURE. INSTRUCTED MANAGER TO TURN ON ALL FREEZERS. ALL FREEZERS MUST BE OPERABLE AND MAINTAIN PROPER TEMPERATURE. CRITICAL VIOATION	11. ADEQUATE NUMBER, CONVENIENT, ACCESSIBLE, DESIGNED, AND MAINTAINED - Comments: OBSERVED NO EXPOSED SINK IN REAR DISH WASHING AREA. INSTRUCTED MANAGER TO INSTALL EXPOSED SINK IN REAR DISH WASHING AREA	21. * CERTIFIED FOOD MANAGER ON SITE WHEN POTENTIALLY HAZARDOUS FOODS ARE PREPARED AND SERVED - Comments: NO ORIGINAL CHICAGO FOOD SANITATION CERTIFICATE POSTED. INSTRUCTED MANAGER TO PROVIDE ORIGINAL FOOD SANITATION CERTIFICATE. COPIES ARE NOT ACCEPTABLE. SERIOUS VIOLATION 7-38-012A	22. DISH MACHINES: PROVIDED WITH ACCURATE THERMOMETERS, CHEMICAL TEST KITS AND SUITABLE GAUGE COCK - Comments: NO CHEMICAL TEST KIT ON SITE. INSTRUCTED MANAGER TO PROVIDE CHEMICAL TEST KIT FOR PROPER SANITIZER CONCENTRATION. SERIOUS VIOLATION 7-38-030	30. FOOD IN ORIGINAL CONTAINER, PROPERLY LABELED: CUSTOMER ADVISORY POSTED AS NEEDED - Comments: LABEL ALL BULK CONTAINERS IN PREP AREA.	32. FOOD AND NON-FOOD CONTACT SURFACES PROPERLY DESIGNED, CONSTRUCTED AND MAINTAINED - Comments: PROVIDE COVERS FOR ALL BULK CONTAINERS. PROVIDE ADEQUATE SHELVING IN	33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSILS CLEAN, FREE OF ABRASIVE DETERGENTS - Comments: CLEAN INTERIOR OF ALL WALK-IN COOLERS AND FREEZERS.	34. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUST-LESS CLEANING METHODS USED - Comments: CLEAN FLOORS IN REAR UTILITY AREA.	35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Comments: CLEAN LIGHT SHIELD IN PREP AND REAR UTILITY AREA. CLEAN CEILING VENTS IN PREP	37. TOILET ROOM DOORS SELF CLOSING: DRESSING ROOMS WITH LOCKERS PROVIDED: COMPLETE SEPARATION FROM LIVING/SLEEPING QUARTERS - Comments: INSTALL SELF-CLOSING DEVICE ON WASHROOM DOOR.	38. VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED AND MAINTAINED - Comments: REPAIR LEAKING FAUCET AND SPRAYER ON 3-COMPARTMENT SINK.

We saw as many as 23 violations (citations) listed for some facilities during an inspection event. Now, we cleaned each of these individual violations and derived violation code from them.

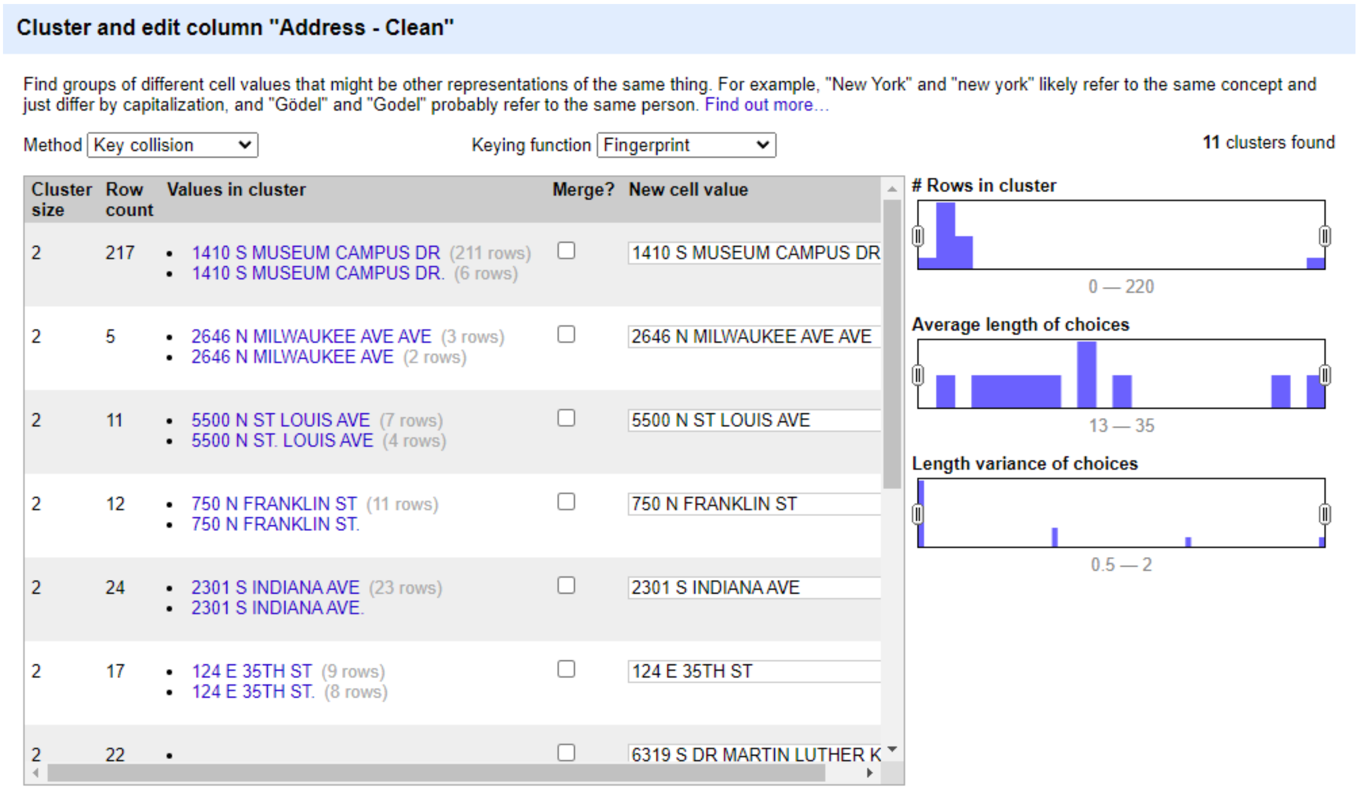
- Cleaning leading and trailing spaces, convert all the text into uppercase, collapse multiple spaces into a single space, remove any special characters that were not part of the Basic Latin set. We performed the same operations on the split fields only, we did not create any new field.
    - Cleaning Step: This step was performed by using common cell transformations on all the 23 new violation fields.
        - Trim leading and trailing whitespace
        - Collapse consecutive whitespace
        - To uppercase
    - Rationale: We wanted to gather all the textual details of the inspection results. Since each inspection record can contain more than one violation observation, we wanted to parse them out first for further transformation.
    - Useful / Required for use case U1: The parsed violation texts will help us get more information about the inspection results, e.g., what category of violation it was.

Address: Building and street name. We created the new field as 'Address - Clean'.

- Cleaning leading and trailing spaces, convert all the text into uppercase, collapse multiple spaces into a single space, remove any special characters that were not part of the Basic Latin set
    - Cleaning Step: This step was performed by using
      1.value.trim().toUppercase().replace(/[^\u0020-\u007F]/,"").replace(" "," ");
      2.value.replace(/[\p{Zs}\s]+/,' ')
    - Rationale: Leading or trailing spaces can interfere with clustering text values into the correct cluster.

- Useful / Required for use case U1: Unique and consistent identification for business location and address is necessary for smooth efficient analysis.

- Clustering and Merge: When ran the clustering process in the Address - Clean field, we found the below clusters. In total, there were 19 variations of Address spellings that either included repeated street indications or a period punctuation "." after commonly abbreviated words like 'Street' to 'ST' or 'ST.' and 'Drive' to 'DR' or 'DR.'. All spellings were merged to exclude punctuation and remove duplicates.
    - Before:



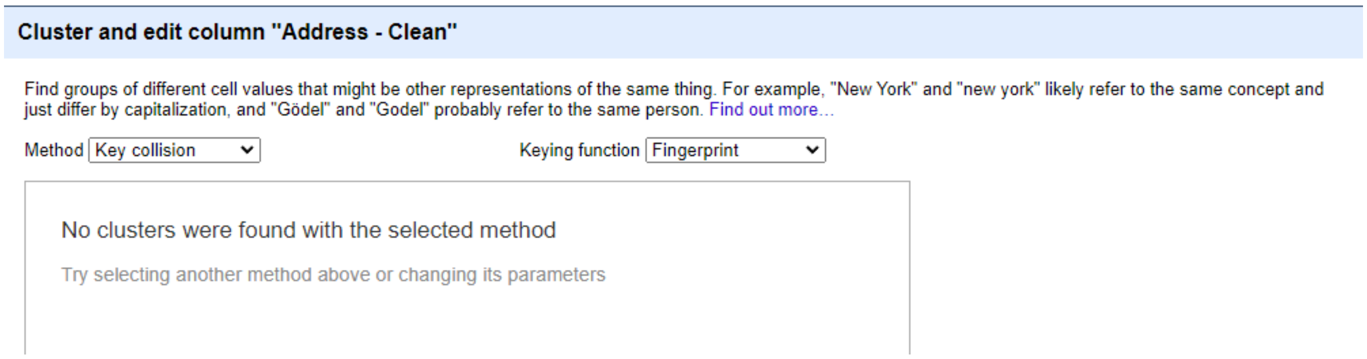
Cluster and edit column "Address - Clean"

Find groups of different cell values that might be other representations of the same thing. For example, "New York" and "new york" likely refer to the same concept and just differ by capitalization, and "Gödel" and "Godel" probably refer to the same person. Find out more...

Method: Key collision   Keying function: Fingerprint   **11** clusters found

Cluster size	Row count	Values in cluster	Merge?	New cell value
2	217	• 1410 S MUSEUM CAMPUS DR (211 rows) • 1410 S MUSEUM CAMPUS DR. (6 rows)	☐	1410 S MUSEUM CAMPUS DR
2	5	• 2646 N MILWAUKEE AVE AVE (3 rows) • 2646 N MILWAUKEE AVE (2 rows)	☐	2646 N MILWAUKEE AVE AVE
2	11	• 5500 N ST LOUIS AVE (7 rows) • 5500 N ST. LOUIS AVE (4 rows)	☐	5500 N ST LOUIS AVE
2	12	• 750 N FRANKLIN ST (11 rows) • 750 N FRANKLIN ST.	☐	750 N FRANKLIN ST
2	24	• 2301 S INDIANA AVE (23 rows) • 2301 S INDIANA AVE.	☐	2301 S INDIANA AVE
2	17	• 124 E 35TH ST (9 rows) • 124 E 35TH ST. (8 rows)	☐	124 E 35TH ST
2	22	•	☐	6319 S DR MARTIN LUTHER K

- After:



Cluster and edit column "Address - Clean"

Find groups of different cell values that might be other representations of the same thing. For example, "New York" and "new york" likely refer to the same concept and just differ by capitalization, and "Gödel" and "Godel" probably refer to the same person. Find out more...

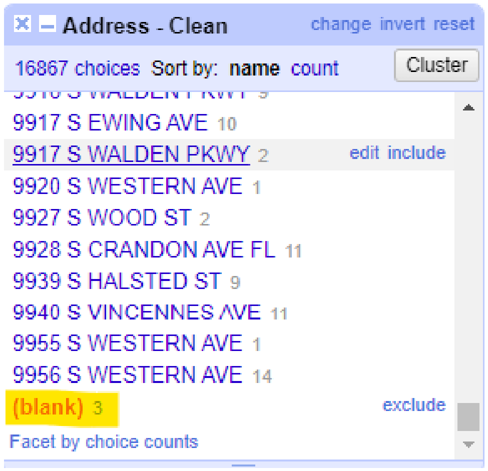
Method: Key collision   Keying function: Fingerprint

No clusters were found with the selected method

Try selecting another method above or changing its parameters

- Rationale: Clustering revealed many different variations of the same text. We used this method to standardize the values for better analysis.
- Useful / Required for use case U1: Standardization helps to consolidate the values into limited sets which is useful for the analysis.

- Flagged blank Address entries and created a new field called ‘Address - Flag’ to indicate empty addresses.
    - Cleaning Step: This step was performed by using if(isBlank(value),’Missing’,’Exists’)
    - Rationale: If there are missing address fields, it will be difficult to analyze based on the location. On the other hand, in the chance that another record based on the same business name (or other identifier) does include the address, we can fill in the missing address from there. This can be done later in the project.
    - Useful / Required for use case U1: As mentioned before, having missing address fields will make it difficult to use the location for analysis. If missing addresses cannot be filled in, they would likely be excluded. Having the flag field will make it efficient to filter which records have missing addresses.

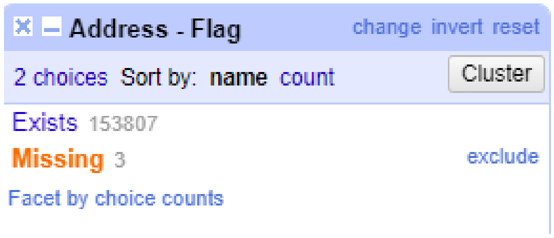


Address - Clean change invert reset
16867 choices Sort by: name count Cluster

```
9917 S EWING AVE 10
9917 S WALDEN PKWY 2          edit include
9920 S WESTERN AVE 1
9927 S WOOD ST 2
9928 S CRANDON AVE FL 11
9939 S HALSTED ST 9
9940 S VINCENNES AVE 11
9955 S WESTERN AVE 1
9956 S WESTERN AVE 14
(blank) 3                     exclude
```

Facet by choice counts

Added field ‘Address - Flag’



Address - Flag change invert reset
2 choices Sort by: name count Cluster

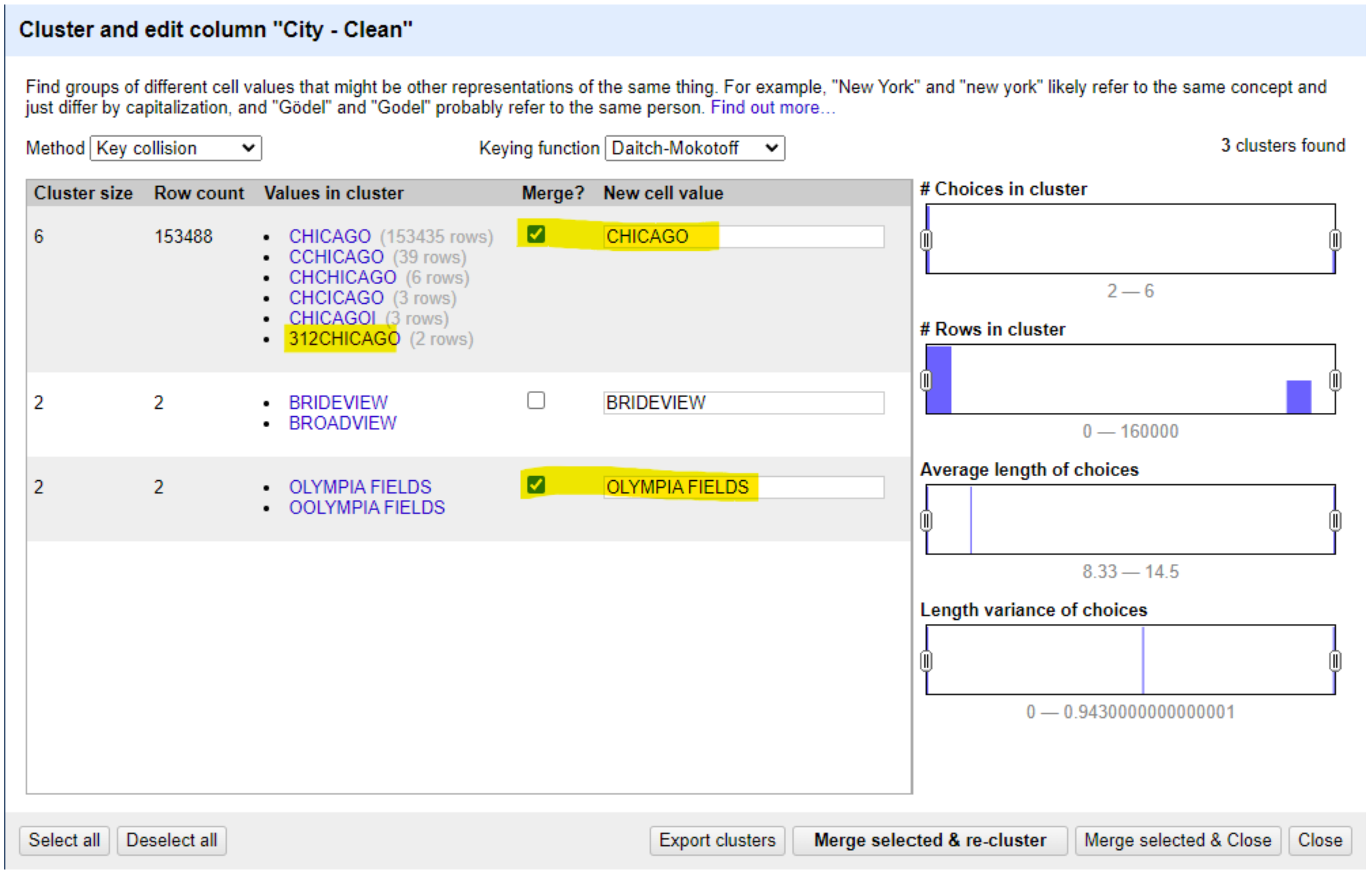
```
Exists 153807
Missing 3                     exclude
```

Facet by choice counts

<u>City:</u> City name. We created the new field as ‘City - Clean’.

- Cleaning leading and trailing spaces, convert all the text into uppercase, collapse multiple spaces into a single space, remove any special characters that were not part of the Basic Latin set
    - Cleaning Step: This step was performed by using value.trim().toUppercase().replace(/[^\u0020-\u007F]/,"").replace(" "," ").replace(/[\p{Zs}\s]+/,’ ‘);
    - Rationale: Leading or trailing spaces can interfere with clustering text values into the correct cluster.

    - Useful / Required for use case U1: Unique and consistent identification for business city location is necessary for smooth efficient analysis.
- Clustering and Merge: When clustering based on Key Collision and Daitch-Mokotoff as keying function, several misspellings were found, particularly of ‘CHICAGO’ and ‘OLYMPIA FIELDS’. With these clustering parameters, we were able to capture the most misspellings together. However, it also grouped ‘BRIDEVIEW’ and ‘BROADVIEW’, which are separate cities. As a result, we did not merge those cities.
    - Before:



Cluster and edit column "City - Clean"

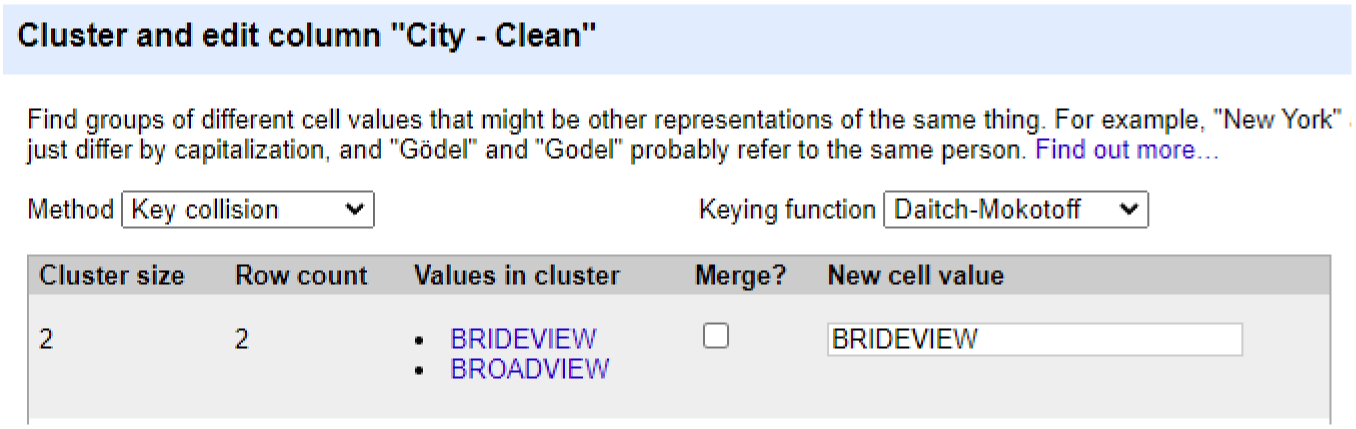
Find groups of different cell values that might be other representations of the same thing. For example, "New York" and "new york" likely refer to the same concept and just differ by capitalization, and "Gödel" and "Godel" probably refer to the same person. Find out more…

Method: Key collision | Keying function: Daitch-Mokotoff | 3 clusters found

Cluster size	Row count	Values in cluster	Merge?	New cell value
6	153488	CHICAGO (153435 rows) CCHICAGO (39 rows) CHCHICAGO (6 rows) CHCICAGO (3 rows) CHICAGOI (3 rows) 312CHICAGO (2 rows)	✓	CHICAGO
2	2	BRIDEVIEW BROADVIEW	☐	BRIDEVIEW
2	2	OLYMPIA FIELDS OOLYMPIA FIELDS	✓	OLYMPIA FIELDS

Note: The ‘312’ refers to the city of Chicago’s phone area code, which will likely be unneeded in a location based analysis where the address and coordinates are already given.

    - After:

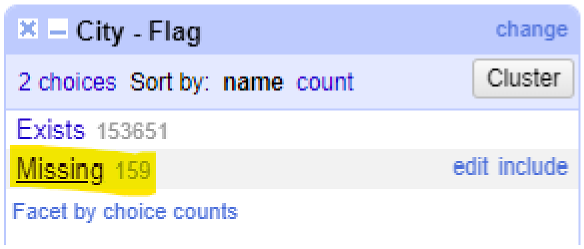

Cluster and edit column "City - Clean"

Find groups of different cell values that might be other representations of the same thing. For example, "New York"
just differ by capitalization, and "Gödel" and "Godel" probably refer to the same person. Find out more…

Method: Key collision — Keying function: Daitch-Mokotoff

Cluster size	Row count	Values in cluster	Merge?	New cell value
2	2	• BRIDEVIEW • BROADVIEW	☐	BRIDEVIEW

- Flagged blank City entries and created a new field called ‘City - Flag’ to indicate empty addresses.
  - Cleaning Step: This step was performed by using if(isBlank(value),’Missing’,’Exists’)
  - Rationale: Similar to missing addresses, If there are missing city fields, it will be difficult to analyze based on the location.
  - Useful / Required for use case U1: Being a part of the full address, the city is a necessary part of location analysis based on the address. Using records with an existing city entry will make analysis much more efficient.


City - Flag — change

2 choices Sort by: name count — Cluster

Exists	153651
Missing	159

Facet by choice counts

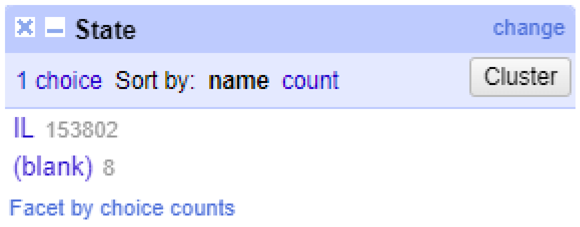
There were 159 records with missing city entries.

State: Reference to the state, all of which should be IL, assuming this a dataset of Chicago food business.

- Cleaning leading and trailing spaces, convert all the text into uppercase, collapse multiple spaces into a single space, remove any special characters that were not part of the Basic Latin set.
  - Cleaning Step: This step was performed by using value.trim().toUppercase().replace(/[^\u0020-\u007F]/,"").replace(" "," ").replace(/[\p{Zs}\s]+/,’ ‘);
  - Rationale: Leading or trailing spaces can interfere with clustering text values into the correct cluster.
- Replaced blank entries with ‘IL’.
  - Cleaning Step: This step was performed by using if(isBlank(value),’IL’,’IL’)
  - Rationale: Assuming every inspection is done within the state of IL for Chicago-based food business, the state will automatically be IL.

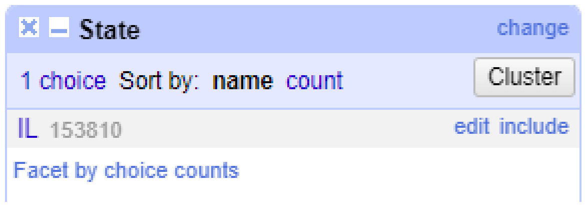
- Useful / Required for use case U1: When using the address for location based analysis, having the state correctly set as ‘IL’ can prove useful for obtaining a completed address for all Chicago-based food businesses.

  Before:

  State change
  1 choice Sort by: name count Cluster
  IL 153802
  (blank) 8
  Facet by choice counts

  After:

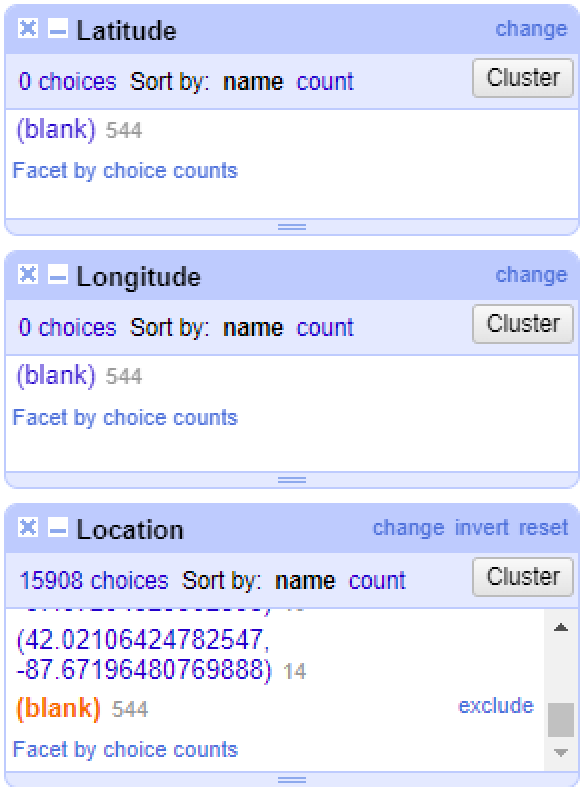
  

  State change
  1 choice Sort by: name count Cluster
  IL 153810 edit include
  Facet by choice counts

Location: The location is split into 3 separate fields: Latitude, Longitude, and (geo) Location. For the latitude and longitude, we were able to easily convert the already separated text into numbers.

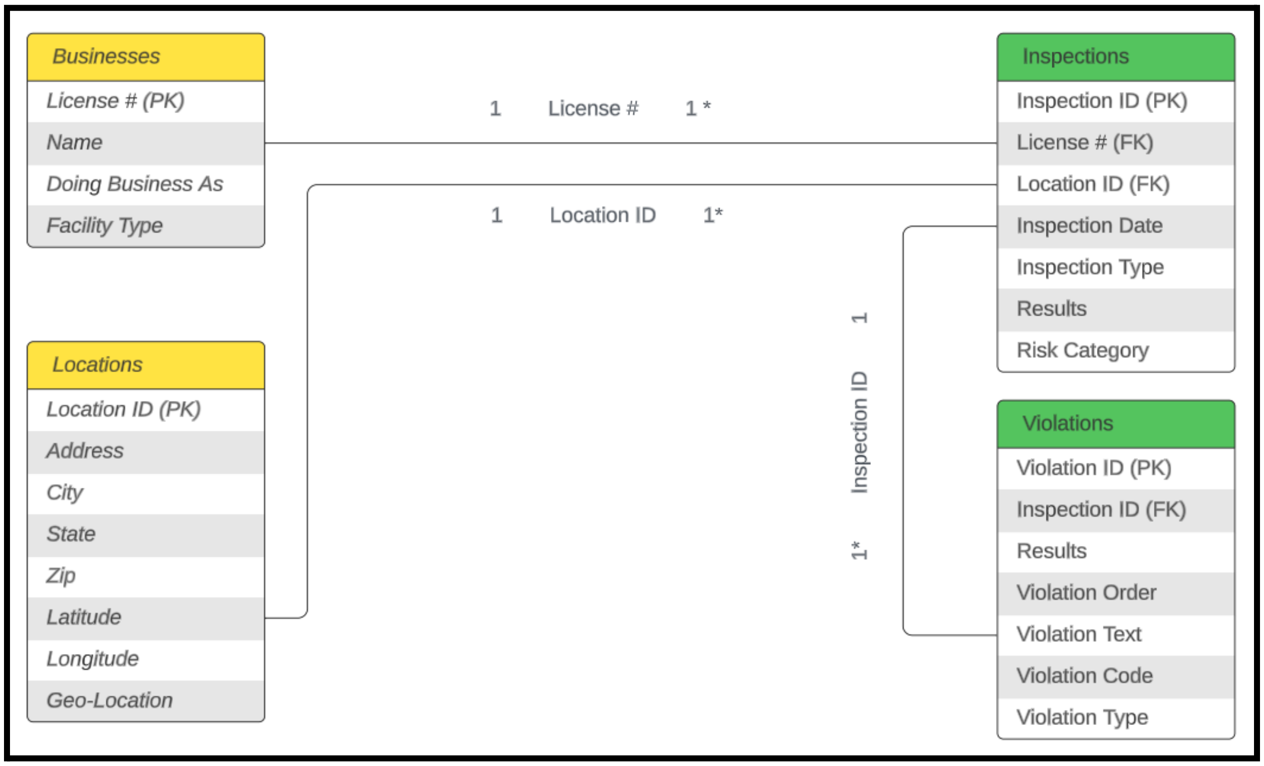
- Cleaning leading and trailing spaces, convert all the text into uppercase, collapse multiple spaces into a single space, remove any special characters that were not part of the Basic Latin set.
  - Cleaning Step: This step was performed on all 3 fields by using value.trim().toUppercase().replace(/[^\u0020-\u007F]/,"").replace(" "," ").replace(/[\p{Zs}\s]+/,’ ‘);
  - Rationale: Leading or trailing spaces can interfere with clustering text values into the correct cluster.
- Converting the already separated text into numbers for latitude and longitude.
  - Cleaning Step: This step was performed on all 3 fields by using value.toNumber()
  - Rationale: Converting the text into numbers can allow for analysis should the numbers rather than the text be needed.
- Removing the parenthesis ‘(‘ and ‘)’ and creating a new field called ‘Location - Clean’
  - Cleaning Step: This step was performed on all 3 fields by using value.replace(/\(|\)/,'')
  - Rationale: Removing the parenthesis can make it easier to read.
- Creating a new field called ‘Location - Flag’ to indicate whether a set of coordinates is missing.
  - Cleaning Step: This step was performed on all 3 fields by using asdf

- Rationale: As shown below, there were about 544 blank entries that seem to be consistently present in all 3 fields:



- To make this clear, a new flag with an indication of whether the location is ‘Missing’ can provide a more efficient way of filtering rows without coordinates to have a more complete location-based analysis. Entries with a location are flagged as ‘Exists’.
- Useful / Required for use case U1: Removing the missing entries and coordinate strings to their separate numeric values can provide a much more efficient and flexible location-based analysis for future steps in such a project.

The new data model is designed into the below ER Diagram.



“**Businesses**” object is a master dataset (dimension table in star schema) of all DBA names. All unique businesses are listed in this file. No business is duplicated. Each business is uniquely identified through License #, which is a valid non-zero / non-blank number.

Cleaned CSV file is located here:
https://raw.githubusercontent.com/ratulsaha778/cs513-final-project/main/business.csv

“**Locations**” object is a master dataset (dimension table in star schema) of all unique addresses. All unique businesses are located at one or more unique locations over the 8 years period (inspection date). No location is duplicated. Each location is uniquely identified by its own identifier ‘Location ID’, which is a valid non-blank number starting from zero.

Cleaned CSV file is located here:
https://raw.githubusercontent.com/ratulsaha778/cs513-final-project/main/locations.csv

“**Violations**” is a transactional dataset (fact table in star schema) of all violations that were documented during each inspection. Every violation observation is mapped to a single inspection event (Inspection ID). Each violation observation is uniquely identified by a non-blank identifier ‘Violation ID’ starting from zero. ‘Violation Order’ mentioned which number observation or sequence it is (e.g., observation # 1, observation # 2 etc.). Each violation observation (as ‘Violation Text’) is linked to a unique code (‘Violation Code’) ranging from 1-44 and 70 (total 45 values). Violation codes 1-14 are marked as ‘Critical’, violation codes 15-29 are marked as ‘Serious’, and the rest are all marked as ‘Other’ in the field ‘Violation Type’. Inspection ID to Violation ID is 1-to-many mapped.

Cleaned CSV file is located here:
https://raw.githubusercontent.com/ratulsaha778/cs513-final-project/main/violations-partial.csv
(Disclaimer: the full data was too large to load into github through UI, so we loaded a partial dataset with limited records).

“**Inspections**” is a transactional dataset (fact table in star schema) that contains the outcomes of each inspection. ‘Inspection ID’ uniquely identifies each inspection record. Each business from “Businesses” is 1-to-many mapped to “Inspections”, which means a single business is inspected one or more times. Each location from “Locations” is 1-to-many mapped to “Inspections”, which means the same facility was inspected one or more times (same or under different business).

Cleaned CSV file is located here:
https://raw.githubusercontent.com/ratulsaha778/cs513-final-project/main/inspections.csv

Data Quality Changes

We performed the data quality operations in 2 phases:

1. OpenRefine
    a. Initial data cleaning of multiple fields by trimming, collapse of consecutive spaces, special character removal, upcasing, and type conversion
    b. Clustering and merging
    c. Mass data update through the use of facets
    d. Flagging of incorrect values
2. Python
    a. Separated the records from a single CSV file into 4 files (businesses, locations, inspections, violations)
    b. Separated the records based on flag fields
        i. License # - Flag is incorrect
        ii. Risk - Flag is incorrect
    c. Separated the records that have invalid data
        i. Invalid violation code records
        ii. Blank violation text in 'Fail' inspections
    d. Separated the records that have IC violations
        i. License # and DBA Name dependency

The OpenRefine JSON file is attached as evidence of these data quality steps taken place.
https://raw.githubusercontent.com/ratulsaha778/cs513-final-project/main/OpenRefine%20Recipe.json

We performed some data quality operations further using Python. A link to the Python script is provided and the pseudo logic is described below.

The python script is located here:
https://raw.githubusercontent.com/ratulsaha778/cs513-final-project/main/Data%20Cleaning%20and%20Validation.py

Python Pseudo Logic:

- Read the OpenRefine output cleaned CSV file that has all 153,810 records
- Separate the business entities and make its own master dataset "Businesses". Retain the provided unique id 'License #', but cleaned. Further clean and enrich the data:
    - Removed incorrect License #
    - Selected the latest DBA Name and AKA Name from all inspection records
- Separate the locations entities and make its own master dataset "Locations". Created a new unique identifier 'Location ID' for each location. Further clean and enrich the data:
    - Populate missing Zip codes based on present Address, City and State
    - Populate missing City codes based on present Zip code
    - Populate missing Address, City, Zip based on latitude and longitude
    - Populate Address, City, Zip based on partial Address

- Separate the inspections instances and make its own fact dataset “Inspections”. Retain the provided unique identifier ‘Inspection ID’ for each inspection. Further clean and enrich the data:
    - Remove inspection records linked to incorrect License #
    - Remove inspection records linked to incorrect Risk
    - Retain the integrity (checked constraints) by mapping with ‘License #’ and ‘Location ID’
- Separate the violation instances and make its own fact dataset “Violations”. Created a new unique identifier Violation ID’ for each violation observation under each inspection event. Further clean and enrich the data:
    - Tag Violation Type: Critical / Serious / Other
    - Derive Violation Code: 1-44, 70
    - Remove incorrect violation context (Pass inspections cannot have Critical / Serious observations)
    - Convert combined and dirty inspection observations into a cleaner tall-order format that is easy for further analysis through natural language processing

The below table shows what columns were changed and how many cells were changed during the quality improvement process. White cells' actions are performed in OpenRefine and yellow cells’ actions are performed in Python.

SL#	Field Name	Operation	# of cells changed
OpenRefine			
1	DBA Name	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces, remove special characters	153,810
2	AKA Name	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces, remove special characters	153,810
3	AKA Name	Populate blank value cells by DBA Name values	2,543
4	License #	Prepare License # - Flag field	153,810
5	License #	Trimming leading and trailing whitespace, to number, collapsing consecutive whitespaces	153,795
6	Risk	Prepare Risk - Flag field	153,810
7	Risk	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	153,744
8	Facility Type	Cleaning through text facet	291
9	Facility Type	Trimming leading and trailing whitespace,	149,250

		uppercasing, collapsing consecutive whitespaces	
10	Facility Type	Cluster and Merge	114,899
11	Inspection Date	Split into 3 separate fields: month, day and year	153,810
12	Inspection Type	Cleaning through text facet	21,343
13	Inspection Type	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	153,809
14	Inspection Type	Cluster and Merge	7,228
15	Violations	Field split into 23 separate fields	123,012
16	Violation 1	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	277,103
17	Violation 2	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	146,315
18	Violation 3	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	218,149
19	Violation 4	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	171,089
20	Violation 5	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	127,238
21	Violation 6	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	89,739
22	Violation 7	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	60,402
23	Violation 8	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	39,759
24	Violation 9	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	24,762
25	Violation 10	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	12,654
26	Violation 11	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	8,481
27	Violation 12	Trimming leading and trailing whitespace,	4,720

		uppercasing, collapsing consecutive whitespaces	
28	Violation 13	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	2,550
29	Violation 14	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	1,260
30	Violation 15	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	612
31	Violation 16	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	296
32	Violation 17	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	141
33	Violation 18	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	71
34	Violation 19	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	37
35	Violation 20	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	19
36	Violation 21	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	14
37	Violation 22	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	2
38	Violation 23	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	2
39	Address	Trimming leading and trailing whitespace, uppercasing	153,810
40	Address	Collapsing consecutive whitespaces	618
41	Address	Cluster and merge	344
42	Address	Prepare Address - Flag field	153,810
43	City	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	153,651
44	City	Cluster and merge	153,490
45	City	Prepare City - Flag field	153,810
46	State	Trimming leading and trailing whitespace,	0

		uppercasing, collapsing consecutive whitespaces	
47	State	Replace blanks with ‘IL’	8
48	Latitude	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	0
49	Longitude	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	0
50	Location	Trimming leading and trailing whitespace, uppercasing, collapsing consecutive whitespaces	0
51	Latitude	Convert string text to number	153,266
52	Longitude	Convert string text to number	153,266
53	Location	Remove parenthesis	153,266
54	Location	Prepare Location - Flag	153,810

SL#	Field Name	Operation	# of cells changed
		Python	
1	Risk - Flag	Separate ‘Incorrect’ records based on Risk - Flag field	85 records cleaned
2	License # - Flag	Separate ‘Incorrect’ records based on License # - Flag field	454 records cleaned
3	Violation Code	Creation of Violation Code field by consolidating the code from each of the fields Violation 1 - 23	153,810 records updated
4	Violation Code	Prepare Violation - Flag field	568,654 records (transposed)
5	Violation Code	Separate ‘Incorrect’ records based on Violation - Flag field	33,524 records cleaned
6	DBA Name	A standard and same DBA Name for businesses with the same License #	1339 records adjusted
7	Inspection ID, License # - Clean, Location ID, Violation Code	Split the single dataset into 4 separate datasets: businesses, locations, inspections, violations	**Businesses**: 32,849 records **Locations**: 16,875 records

			Inspections: 153,299 records **Violations**: 535,130 records
8	Incomplete Location details	Clean incorrect addresses, Enrichment of Address, City, State, Zip, Location	270 records cleaned

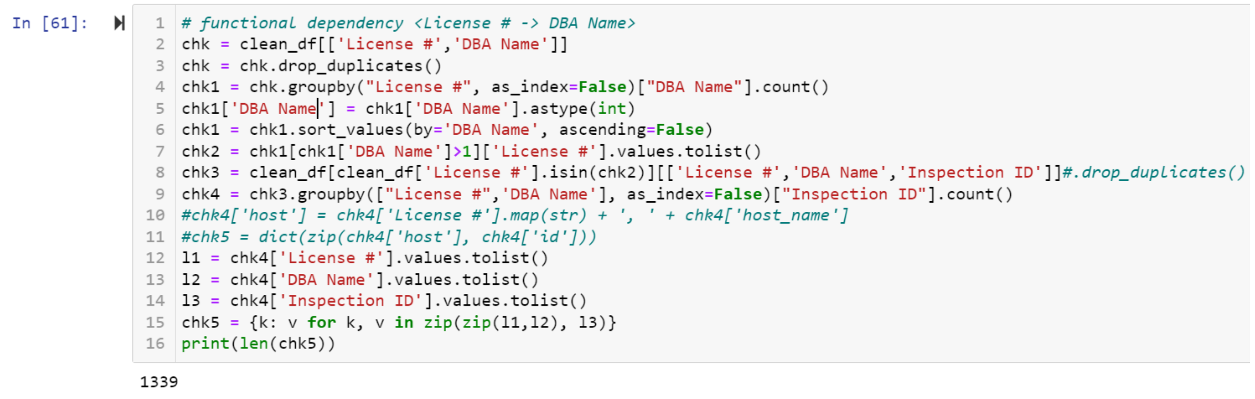
The below table shows the before and after metrics of data quality improvements:

SL#	Description	Before Cleaning	After Cleaning
1	Records with Blank or Zero License #	454	0
2	Non-standard location City names	57	46
3	Blank location State	8	0
4	Blank location Zip Code	98	1
5	Blank location Longitude / Latitude	544	123
6	Different DBA Names for businesses with the same License #	1339	0
7	Facet choices for Inspection Type	108	81
8	Facet choices for Facility Type	387	350
9	Facet choices for Risk	5	3
10	Blank AKA Name	2,543	0
11	Incorrect Violation Codes	If Results = 'Pass', 5404 observations were Critical and 19770 observations were Serious	If Results = 'Pass', no observations are Critical or Serious
12	Incorrect Risk data	85	0
13	Unique DBA Names	24,685	23,904
14	Unique License #	32,851	32,849
15	Unique locations	16,870	16,875 (enriched)
16	Unique inspections	153,810	153,299

17	Unique violations	153,810 records of combined unparsable violation texts	535,130 records of clean, parsed and unique violation texts mapped with violation type

IC-Violation Check in Python:

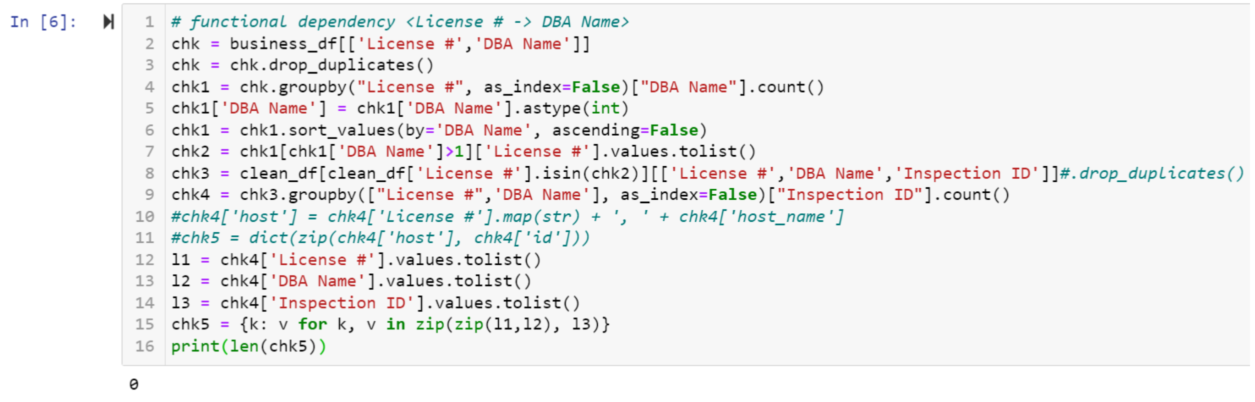
Before (1339 violations were present in business dataset):



```
In [61]:
# functional dependency <License # -> DBA Name>
chk = clean_df[['License #','DBA Name']]
chk = chk.drop_duplicates()
chk1 = chk.groupby("License #", as_index=False)["DBA Name"].count()
chk1['DBA Name'] = chk1['DBA Name'].astype(int)
chk1 = chk1.sort_values(by='DBA Name', ascending=False)
chk2 = chk1[chk1['DBA Name']>1]['License #'].values.tolist()
chk3 = clean_df[clean_df['License #'].isin(chk2)][['License #','DBA Name','Inspection ID']]#.drop_duplicates()
chk4 = chk3.groupby(["License #",'DBA Name'], as_index=False)["Inspection ID"].count()
#chk4['host'] = chk4['License #'].map(str) + ', ' + chk4['host_name']
#chk5 = dict(zip(chk4['host'], chk4['id']))
l1 = chk4['License #'].values.tolist()
l2 = chk4['DBA Name'].values.tolist()
l3 = chk4['Inspection ID'].values.tolist()
chk5 = {k: v for k, v in zip(zip(l1,l2), l3)}
print(len(chk5))

1339
```

After (no violations are present in business dataset):



```
In [6]:
# functional dependency <License # -> DBA Name>
chk = business_df[['License #','DBA Name']]
chk = chk.drop_duplicates()
chk1 = chk.groupby("License #", as_index=False)["DBA Name"].count()
chk1['DBA Name'] = chk1['DBA Name'].astype(int)
chk1 = chk1.sort_values(by='DBA Name', ascending=False)
chk2 = chk1[chk1['DBA Name']>1]['License #'].values.tolist()
chk3 = clean_df[clean_df['License #'].isin(chk2)][['License #','DBA Name','Inspection ID']]#.drop_duplicates()
chk4 = chk3.groupby(["License #",'DBA Name'], as_index=False)["Inspection ID"].count()
#chk4['host'] = chk4['License #'].map(str) + ', ' + chk4['host_name']
#chk5 = dict(zip(chk4['host'], chk4['id']))
l1 = chk4['License #'].values.tolist()
l2 = chk4['DBA Name'].values.tolist()
l3 = chk4['Inspection ID'].values.tolist()
chk5 = {k: v for k, v in zip(zip(l1,l2), l3)}
print(len(chk5))

0
```

The output of SQLite queries for data structure and data validation checks:

- The database tables schema is as per the output of the data cleaning Python program

  CREATE TABLE IF NOT EXISTS "businesses"(
  "License #" TEXT, "Dt" TEXT, "DBA Name" TEXT, "AKA Name" TEXT,
  "Facility Type" TEXT, "Inspection Date" TEXT, "License # - Flag" TEXT);

  CREATE TABLE IF NOT EXISTS "locations"(
  "Location ID" TEXT, "Address" TEXT, "City" TEXT, "State" TEXT, "Zip" TEXT, "Latitude" TEXT, "Longitude" TEXT, "Location" TEXT);

```
CREATE TABLE IF NOT EXISTS "violations"(
"Violation ID" TEXT, "Inspection ID" TEXT, "Results" TEXT, "Violation Order" TEXT,
"Violation Text" TEXT, "Violation Code" TEXT, "Violation Type" TEXT, "Violation Flag"
TEXT);

CREATE TABLE IF NOT EXISTS "inspections"(
"License #" TEXT, "Inspection ID" TEXT, "Inspection Date" TEXT, "Inspection Type"
TEXT, "Risk Category" TEXT, "Results" TEXT, "Location ID" TEXT);
```

Result: The field names and format are as expected.
Schema output is located here:
https://raw.githubusercontent.com/ratulsaha778/cs513-final-project/main/inspections_database_schema.sql

- No business present with blank / zero License #

```
sqlite> select count(*) from businesses where "License #" = '0' or "License #" = '';
0

sqlite> select count(*) from businesses where "DBA Name" = '' or "AKA Name" = '';
0
```

Result: There are no records with blank or zero as License #.

- No business present with more than one License #

```
sqlite> select * from businesses where "License #" in (select distinct "License #" from
(select "License #", count(*) from businesses group by "License #" having count(*)>1) )
order by "License #";
sqlite>

sqlite> select "License #", "DBA Name", count(*) as cnt from businesses group by
"License #", "DBA Name" having count(*) > 1;
sqlite>
```

Result: The output is blank as expected, which means every business known by DBA Name has exactly one License #.

- There are no invalid violation code

```
sqlite> select count(distinct "Violation Code") as cnt_vio_code from violations;
cnt_vio_code
46
```

```
sqlite> select distinct "Violation Code" from violations where "Violation Code"*1 > 45;
"Violation Code"
70

sqlite> select min("Violation Code"*1), max("Violation Code"*1) from violations where
"Violation Code"*1 < 70;
"min(""Violation Code""*1)","max(""Violation Code""*1)"
1,45
```

Result: Only the 46 possible different violation codes (1-45,70) are present in the data.

- Only validated violation inspection records are present

```
sqlite> select distinct "Violation Flag" from violations;
"Violation Flag"
Correct
```

Result: No ‘Invalid’ records are present.

- All inspections should be linked a valid business entity

```
sqlite> select i."Inspection ID", b."License #" from inspections i left join businesses b on
i."License #"=b."License #"
 where b."License #" is NULL;
sqlite>
```

Result: The result set is blank, which means all License # present in Inspections, is also present in Businesses.

- All inspections should be linked a valid location where an entity is present

```
sqlite> select i."Inspection ID", l."Location ID" from inspections i left join locations l on
i."Location ID"=l."Location ID" where l."Location ID" is NULL;
sqlite>
```

Result: The result set is blank, which means all Location ID present in Inspections, is also present in Locations.

- Each business identifier and location identifier combination should lead to valid business entities and locations respectively

```
sqlite> select distinct i."Inspection ID", b."License #", l."Location ID" from inspections i
left join businesses b on i."License #"=b."License #" left join locations l on i."Location
ID"=l."Location ID"  where b."License #" is NULL or l."Location ID" is NULL;
```

```
sqlite>
```

Result: The result set is blank, which means all Business License #s and Location IDs present in Inspections, are also present in Businesses and Locations dataset.

The SQL commands are located here:
https://raw.githubusercontent.com/ratulsaha778/cs513-final-project/main/SQL%20Commands.txt

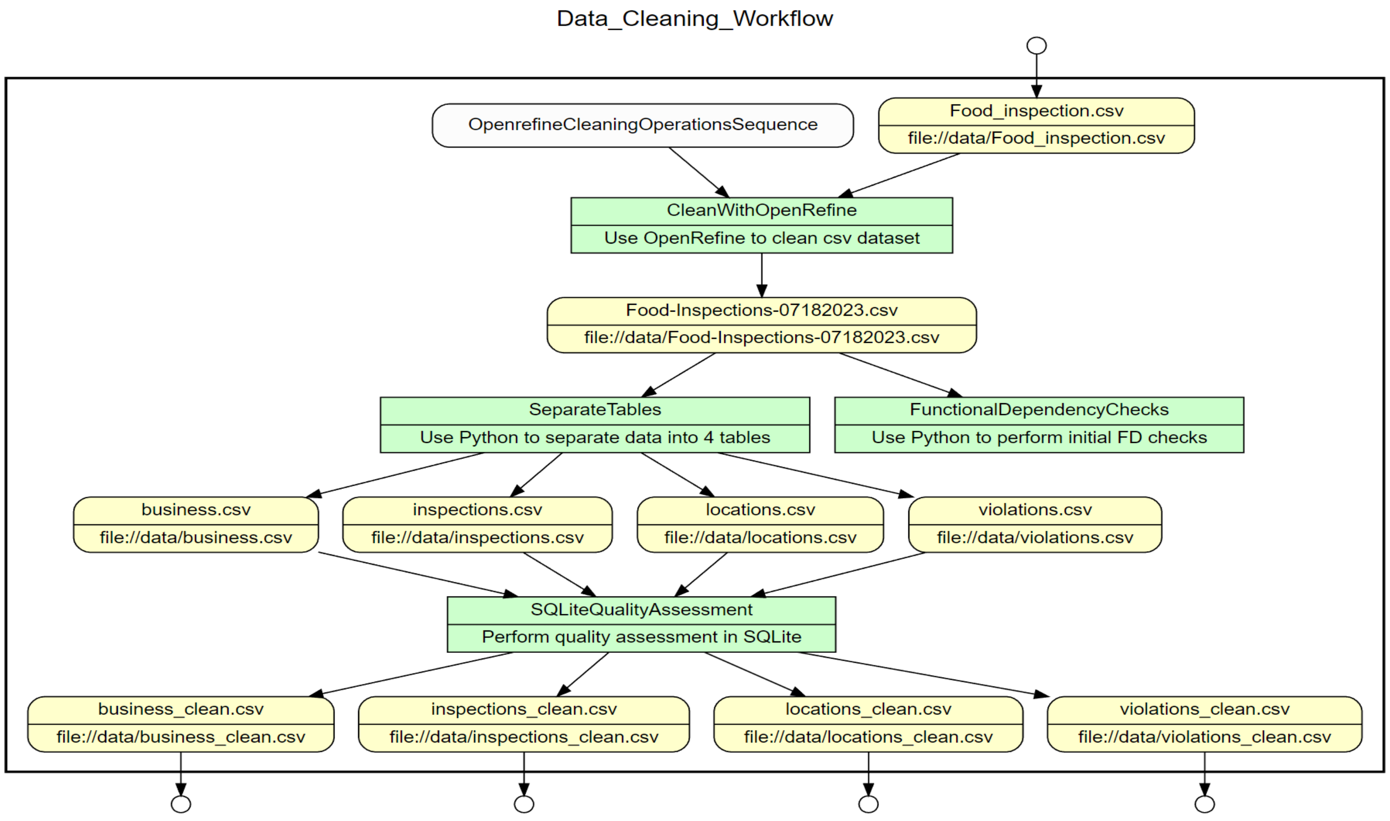
Workflow Models

<u>Outer Workflow:</u>
Cleaning Tools: OpenRefine, Python, SQLite

- Step 1 - Data Loading + Profiling:
    - Load the single source dataset D (Food-Inspections.csv) into a OpenRefine project
    - Perform data profiling and analysis in OpenRefine
    - Create new fields from existing fields
    - Export the dataset into a single file D' (Food-Inspections-07182023.csv)
- Step 2 - Data Cleaning:
    - Load the processed dataset D' into Python
    - Perform data cleaning operations in Python
    - Perform data enrichment operations in Python
    - Create 4 datasets as per ER Diagram D" retaining data integrity between datasets
        - Businesses (business.csv contains all business names)
        - Locations (locations.csv contains all geographics locations notified)
        - Inspections (inspections.csv contains details about all inspection records)
        - Violations (violations.csv contains all the violation details for each inspection)
- Step 3 - Quality Assessment:
    - Perform quality assessment before cleaning in OpenRefine and Python
    - Load all the 4 datasets D" into SQLite
    - Perform quality assessment over all the 4 datasets D" in SQLite

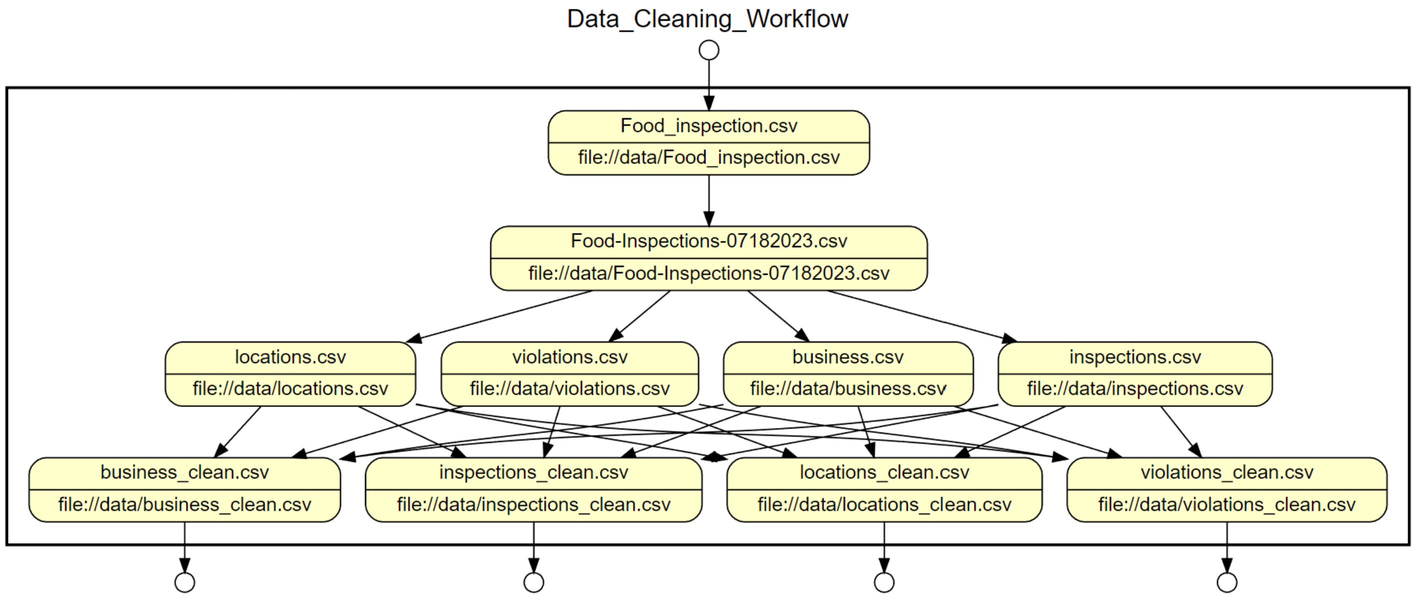
YesWorkflow Graphics:

Complete View:



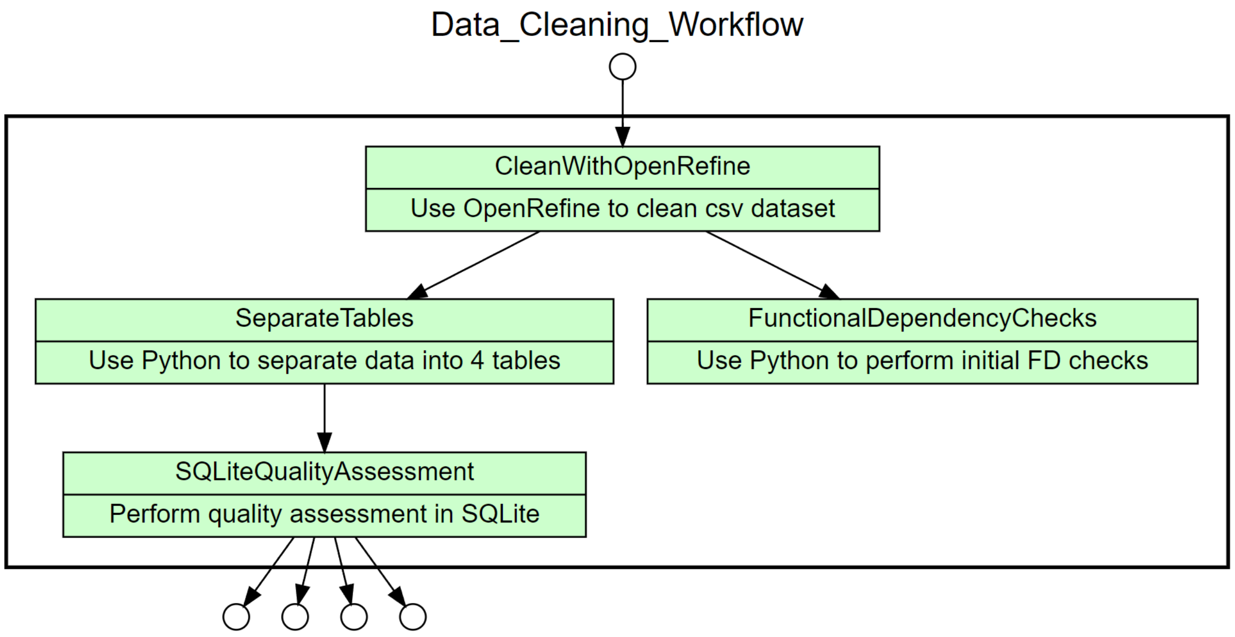
Data_Cleaning_Workflow
OpenrefineCleaningOperationsSequence
Food_inspection.csv
file://data/Food_inspection.csv
CleanWithOpenRefine
Use OpenRefine to clean csv dataset
Food-Inspections-07182023.csv
file://data/Food-Inspections-07182023.csv
SeparateTables
Use Python to separate data into 4 tables
FunctionalDependencyChecks
Use Python to perform initial FD checks
business.csv
file://data/business.csv
inspections.csv
file://data/inspections.csv
locations.csv
file://data/locations.csv
violations.csv
file://data/violations.csv
SQLiteQualityAssessment
Perform quality assessment in SQLite
business_clean.csv
file://data/business_clean.csv
inspections_clean.csv
file://data/inspections_clean.csv
locations_clean.csv
file://data/locations_clean.csv
violations_clean.csv
file://data/violations_clean.csv

Steps View:



Data_Cleaning_Workflow
Food_inspection.csv
file://data/Food_inspection.csv
Food-Inspections-07182023.csv
file://data/Food-Inspections-07182023.csv
locations.csv
file://data/locations.csv
violations.csv
file://data/violations.csv
business.csv
file://data/business.csv
inspections.csv
file://data/inspections.csv
business_clean.csv
file://data/business_clean.csv
inspections_clean.csv
file://data/inspections_clean.csv
locations_clean.csv
file://data/locations_clean.csv
violations_clean.csv
file://data/violations_clean.csv

Data View:


Data_Cleaning_Workflow
CleanWithOpenRefine
Use OpenRefine to clean csv dataset
SeparateTables
Use Python to separate data into 4 tables
FunctionalDependencyChecks
Use Python to perform initial FD checks
SQLiteQualityAssessment
Perform quality assessment in SQLite

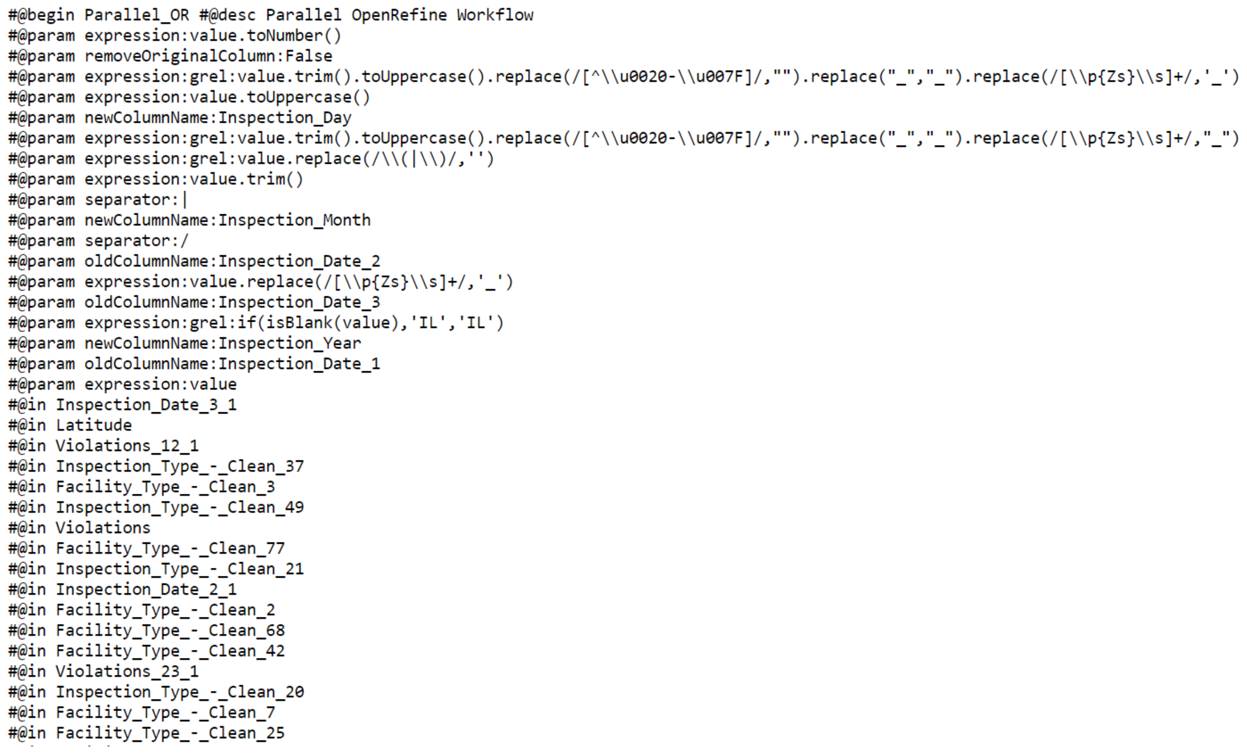
<u>Inner Workflow:</u>
Cleaning Tools: OpenRefine, Python, SQLite

- Step 1:
    - Remove the special characters, converting to uppercase.
    - Trim leading and trailing white spaces and collapse consecutive white spaces
    - Make a facet and perform the cluster operation using the key-collison method and fingerprint function. Merge the relevant clusters. Mass data updates from facets.
    - Create new fields from existing fields
        - DBA Name - Clean from DBA Name
        - AKA Name - Clean from AKA Name
        - License # - Clean from License #
        - Address - Clean from Address
        - City - Clean from City
        - Facility Type - Clean from Facility Type
        - Risk Category - Clean from Risk
        - Inspection Month, Inspection Day and Inspection Year from Inspection Date
        - Inspection Type - Clean from Inspection Type
        - Violations 1 to Violations 23 from Violations
    - Flag record as Correct / Incorrect for further cleaning
        - Address - Flag: Exists / Missing
        - License # - Flag: Correct / Incorrect
        - City - Flag: Exists / Missing
        - Risk - Flag: Correct / Incorrect
        - Location - Flag: Exists / Missing
- Step 2:
    - Create new fields:
        - Location ID - unique identifier for determining a unique location
        - Violation ID - unique identifier for each violation observation
        - Violation Order - index of the violation observation for each inspection
        - Violation Code - unique code for each observation
        - Violation Type - severity type for each violation
    - Remove records with zero/blank values at key fields (dates, license #, inspection ID etc.) with the help of flag field
    - Remove records with invalid data values (incorrect violation codes, incorrect risk) with the help of flag field
    - Avoid functional dependency violations by taking first/last record in the sequence
    - Split the large fact table into 2 master and 2 fact tables (per star schema)
    - Establish primary and foreign key constraints between the final datasets
    - Enrich some missing location data issues through publicly available internet sources
- Step 3:

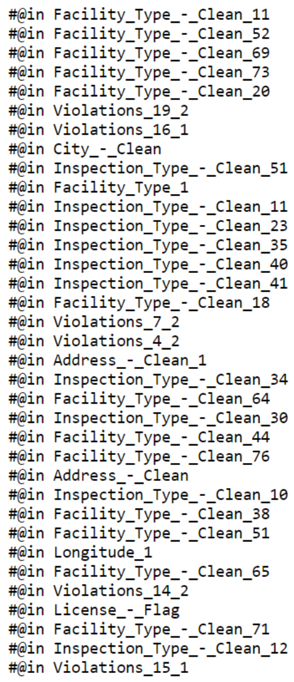
- Before and after view of data quality issues identified
- Load D file into SQLite
- Run queries to measure data quality (Q-before)
- Load D" files (4 csv files loaded into their individual tables) into SQLite
- Run queries to measure data quality (Q-after)
- Comparison if Q-before and Q-after and observe the improvements

The YW document is located at github here:
https://raw.githubusercontent.com/ratulsaha778/cs513-final-project/main/Workflow/Inner_Workflow/openrefine-workflow.yw

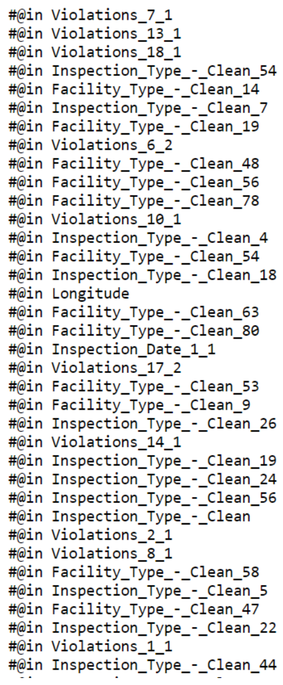
The salient components of the document is also presented below as screenshots:



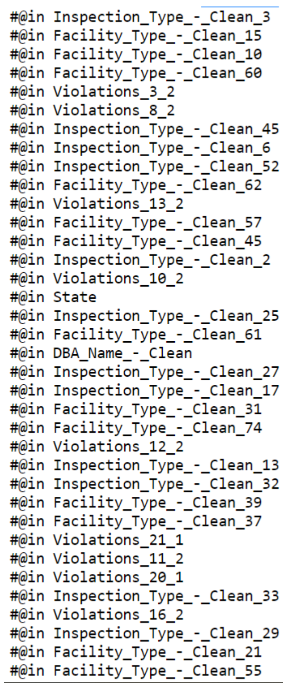
```
#@begin Parallel_OR #@desc Parallel OpenRefine Workflow
#@param expression:value.toNumber()
#@param removeOriginalColumn:False
#@param expression:grel:value.trim().toUppercase().replace(/[^\\u0020-\\u007F]/,"").replace("_","_").replace(/[\\p{Zs}\\s]+/,'_')
#@param expression:value.toUppercase()
#@param newColumnName:Inspection_Day
#@param expression:grel:value.trim().toUppercase().replace(/[^\\u0020-\\u007F]/,"").replace("_","_").replace(/[\\p{Zs}\\s]+/,"_")
#@param expression:grel:value.replace(/\\(|\\)/,'')
#@param expression:value.trim()
#@param separator:|
#@param newColumnName:Inspection_Month
#@param separator:/
#@param oldColumnName:Inspection_Date_2
#@param expression:value.replace(/[\\p{Zs}\\s]+/,'_')
#@param oldColumnName:Inspection_Date_3
#@param expression:grel:if(isBlank(value),'IL','IL')
#@param newColumnName:Inspection_Year
#@param oldColumnName:Inspection_Date_1
#@param expression:value
#@in Inspection_Date_3_1
#@in Latitude
#@in Violations_12_1
#@in Inspection_Type_-_Clean_37
#@in Facility_Type_-_Clean_3
#@in Inspection_Type_-_Clean_49
#@in Violations
#@in Facility_Type_-_Clean_77
#@in Inspection_Type_-_Clean_21
#@in Inspection_Date_2_1
#@in Facility_Type_-_Clean_2
#@in Facility_Type_-_Clean_68
#@in Facility_Type_-_Clean_42
#@in Violations_23_1
#@in Inspection_Type_-_Clean_20
#@in Facility_Type_-_Clean_7
#@in Facility_Type_-_Clean_25
```



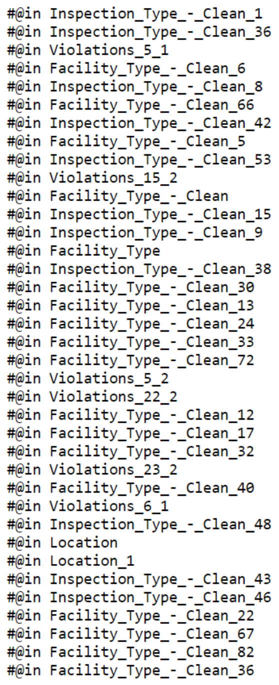
```
#@in Facility_Type_-_Clean_11
#@in Facility_Type_-_Clean_52
#@in Facility_Type_-_Clean_69
#@in Facility_Type_-_Clean_73
#@in Facility_Type_-_Clean_20
#@in Violations_19_2
#@in Violations_16_1
#@in City_-_Clean
#@in Inspection_Type_-_Clean_51
#@in Facility_Type_1
#@in Inspection_Type_-_Clean_11
#@in Inspection_Type_-_Clean_23
#@in Inspection_Type_-_Clean_35
#@in Inspection_Type_-_Clean_40
#@in Inspection_Type_-_Clean_41
#@in Facility_Type_-_Clean_18
#@in Violations_7_2
#@in Violations_4_2
#@in Address_-_Clean_1
#@in Inspection_Type_-_Clean_34
#@in Facility_Type_-_Clean_64
#@in Inspection_Type_-_Clean_30
#@in Facility_Type_-_Clean_44
#@in Facility_Type_-_Clean_76
#@in Address_-_Clean
#@in Inspection_Type_-_Clean_10
#@in Facility_Type_-_Clean_38
#@in Facility_Type_-_Clean_51
#@in Longitude_1
#@in Facility_Type_-_Clean_65
#@in Violations_14_2
#@in License_-_Flag
#@in Facility_Type_-_Clean_71
#@in Inspection_Type_-_Clean_12
#@in Violations_15_1
```



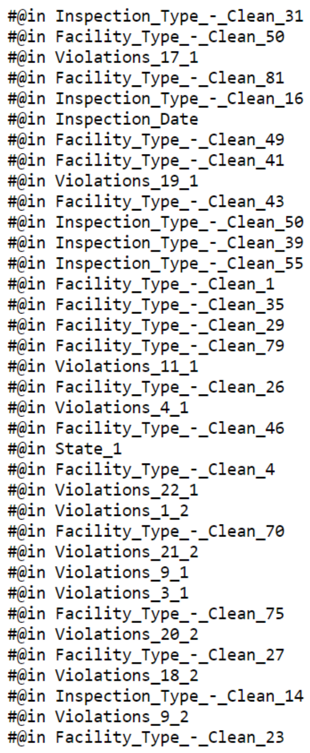
```
#@in Violations_7_1
#@in Violations_13_1
#@in Violations_18_1
#@in Inspection_Type_-_Clean_54
#@in Facility_Type_-_Clean_14
#@in Inspection_Type_-_Clean_7
#@in Facility_Type_-_Clean_19
#@in Violations_6_2
#@in Facility_Type_-_Clean_48
#@in Facility_Type_-_Clean_56
#@in Facility_Type_-_Clean_78
#@in Violations_10_1
#@in Inspection_Type_-_Clean_4
#@in Facility_Type_-_Clean_54
#@in Inspection_Type_-_Clean_18
#@in Longitude
#@in Facility_Type_-_Clean_63
#@in Facility_Type_-_Clean_80
#@in Inspection_Date_1_1
#@in Violations_17_2
#@in Facility_Type_-_Clean_53
#@in Facility_Type_-_Clean_9
#@in Inspection_Type_-_Clean_26
#@in Violations_14_1
#@in Inspection_Type_-_Clean_19
#@in Inspection_Type_-_Clean_24
#@in Inspection_Type_-_Clean_56
#@in Inspection_Type_-_Clean
#@in Violations_2_1
#@in Violations_8_1
#@in Facility_Type_-_Clean_58
#@in Inspection_Type_-_Clean_5
#@in Facility_Type_-_Clean_47
#@in Inspection_Type_-_Clean_22
#@in Violations_1_1
#@in Inspection_Type_-_Clean_44
```



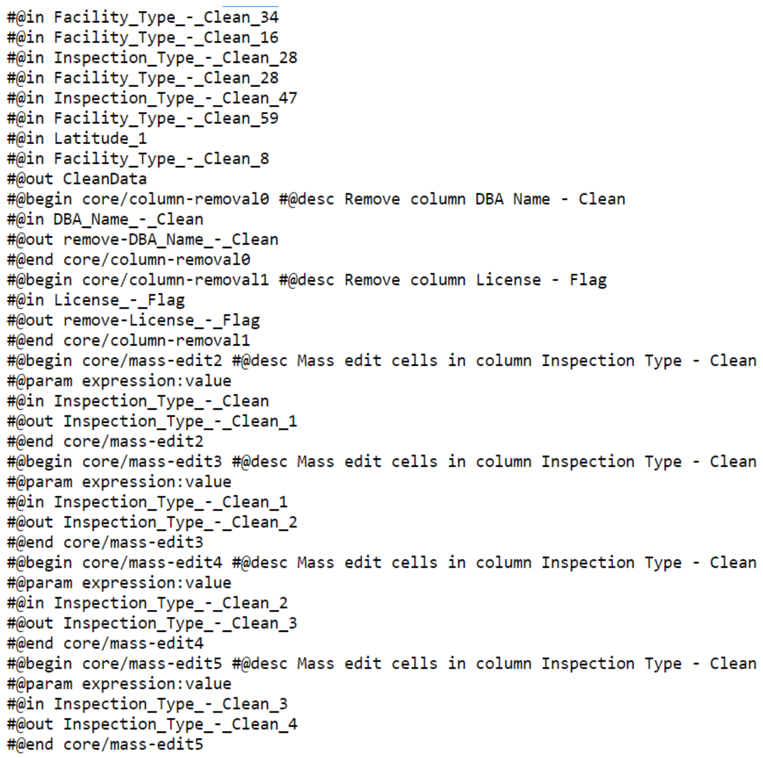
```
#@in Inspection_Type_-_Clean_3
#@in Facility_Type_-_Clean_15
#@in Facility_Type_-_Clean_10
#@in Facility_Type_-_Clean_60
#@in Violations_3_2
#@in Violations_8_2
#@in Inspection_Type_-_Clean_45
#@in Inspection_Type_-_Clean_6
#@in Inspection_Type_-_Clean_52
#@in Facility_Type_-_Clean_62
#@in Violations_13_2
#@in Facility_Type_-_Clean_57
#@in Facility_Type_-_Clean_45
#@in Inspection_Type_-_Clean_2
#@in Violations_10_2
#@in State
#@in Inspection_Type_-_Clean_25
#@in Facility_Type_-_Clean_61
#@in DBA_Name_-_Clean
#@in Inspection_Type_-_Clean_27
#@in Inspection_Type_-_Clean_17
#@in Facility_Type_-_Clean_31
#@in Facility_Type_-_Clean_74
#@in Violations_12_2
#@in Inspection_Type_-_Clean_13
#@in Inspection_Type_-_Clean_32
#@in Facility_Type_-_Clean_39
#@in Facility_Type_-_Clean_37
#@in Violations_21_1
#@in Violations_11_2
#@in Violations_20_1
#@in Inspection_Type_-_Clean_33
#@in Violations_16_2
#@in Inspection_Type_-_Clean_29
#@in Facility_Type_-_Clean_21
#@in Facility_Type_-_Clean_55
```



```
#@in Inspection_Type_-_Clean_1
#@in Inspection_Type_-_Clean_36
#@in Violations_5_1
#@in Facility_Type_-_Clean_6
#@in Inspection_Type_-_Clean_8
#@in Facility_Type_-_Clean_66
#@in Inspection_Type_-_Clean_42
#@in Facility_Type_-_Clean_5
#@in Inspection_Type_-_Clean_53
#@in Violations_15_2
#@in Facility_Type_-_Clean
#@in Inspection_Type_-_Clean_15
#@in Inspection_Type_-_Clean_9
#@in Facility_Type
#@in Inspection_Type_-_Clean_38
#@in Facility_Type_-_Clean_30
#@in Facility_Type_-_Clean_13
#@in Facility_Type_-_Clean_24
#@in Facility_Type_-_Clean_33
#@in Facility_Type_-_Clean_72
#@in Violations_5_2
#@in Violations_22_2
#@in Facility_Type_-_Clean_12
#@in Facility_Type_-_Clean_17
#@in Facility_Type_-_Clean_32
#@in Violations_23_2
#@in Facility_Type_-_Clean_40
#@in Violations_6_1
#@in Inspection_Type_-_Clean_48
#@in Location
#@in Location_1
#@in Inspection_Type_-_Clean_43
#@in Inspection_Type_-_Clean_46
#@in Facility_Type_-_Clean_22
#@in Facility_Type_-_Clean_67
#@in Facility_Type_-_Clean_82
#@in Facility_Type_-_Clean_36
```



```
#@in Inspection_Type_-_Clean_31
#@in Facility_Type_-_Clean_50
#@in Violations_17_1
#@in Facility_Type_-_Clean_81
#@in Inspection_Type_-_Clean_16
#@in Inspection_Date
#@in Facility_Type_-_Clean_49
#@in Facility_Type_-_Clean_41
#@in Violations_19_1
#@in Facility_Type_-_Clean_43
#@in Inspection_Type_-_Clean_50
#@in Inspection_Type_-_Clean_39
#@in Inspection_Type_-_Clean_55
#@in Facility_Type_-_Clean_1
#@in Facility_Type_-_Clean_35
#@in Facility_Type_-_Clean_29
#@in Facility_Type_-_Clean_79
#@in Violations_11_1
#@in Facility_Type_-_Clean_26
#@in Violations_4_1
#@in Facility_Type_-_Clean_46
#@in State_1
#@in Facility_Type_-_Clean_4
#@in Violations_22_1
#@in Violations_1_2
#@in Facility_Type_-_Clean_70
#@in Violations_21_2
#@in Violations_9_1
#@in Violations_3_1
#@in Facility_Type_-_Clean_75
#@in Violations_20_2
#@in Facility_Type_-_Clean_27
#@in Violations_18_2
#@in Inspection_Type_-_Clean_14
#@in Violations_9_2
#@in Facility_Type_-_Clean_23
```



```
#@in Facility_Type_-_Clean_34
#@in Facility_Type_-_Clean_16
#@in Inspection_Type_-_Clean_28
#@in Facility_Type_-_Clean_28
#@in Inspection_Type_-_Clean_47
#@in Facility_Type_-_Clean_59
#@in Latitude_1
#@in Facility_Type_-_Clean_8
#@out CleanData
#@begin core/column-removal0 #@desc Remove column DBA Name - Clean
#@in DBA_Name_-_Clean
#@out remove-DBA_Name_-_Clean
#@end core/column-removal0
#@begin core/column-removal1 #@desc Remove column License - Flag
#@in License_-_Flag
#@out remove-License_-_Flag
#@end core/column-removal1
#@begin core/mass-edit2 #@desc Mass edit cells in column Inspection Type - Clean
#@param expression:value
#@in Inspection_Type_-_Clean
#@out Inspection_Type_-_Clean_1
#@end core/mass-edit2
#@begin core/mass-edit3 #@desc Mass edit cells in column Inspection Type - Clean
#@param expression:value
#@in Inspection_Type_-_Clean_1
#@out Inspection_Type_-_Clean_2
#@end core/mass-edit3
#@begin core/mass-edit4 #@desc Mass edit cells in column Inspection Type - Clean
#@param expression:value
#@in Inspection_Type_-_Clean_2
#@out Inspection_Type_-_Clean_3
#@end core/mass-edit4
#@begin core/mass-edit5 #@desc Mass edit cells in column Inspection Type - Clean
#@param expression:value
#@in Inspection_Type_-_Clean_3
#@out Inspection_Type_-_Clean_4
#@end core/mass-edit5
```

Mass edit lines were skipped (present in the original document loaded at github)

```
#@begin core/column-split59 #@desc Split column Inspection Date by separator
#@param removeOriginalColumn:False
#@param separator:/
#@in Inspection_Date
#@out Inspection_Date_1
#@out Inspection_Date_1_1
#@out Inspection_Date_2_1
#@out Inspection_Date_3_1
#@end core/column-split59
#@begin core/column-rename60 #@desc Rename column Inspection Date 1 to Inspection Month
#@param oldColumnName:Inspection_Date_1
#@param newColumnName:Inspection_Month
#@in Inspection_Date_1_1
#@out Inspection_Month
#@end core/column-rename60
#@begin core/column-rename61 #@desc Rename column Inspection Date 2 to Inspection Day
#@param oldColumnName:Inspection_Date_2
#@param newColumnName:Inspection_Day
#@in Inspection_Date_2_1
#@out Inspection_Day
#@end core/column-rename61
#@begin core/column-rename62 #@desc Rename column Inspection Date 3 to Inspection Year
#@param oldColumnName:Inspection_Date_3
#@param newColumnName:Inspection_Year
#@in Inspection_Date_3_1
#@out Inspection_Year
```

```
#@begin core/column-split149 #@desc Split column Violations by separator
#@param removeOriginalColumn:False
#@param separator:|
#@in Violations
#@out Violations_1
#@out Violations_1_1
#@out Violations_2_1
#@out Violations_23_1
#@out Violations_22_1
#@out Violations_21_1
#@out Violations_20_1
#@out Violations_19_1
#@out Violations_18_1
#@out Violations_17_1
#@out Violations_16_1
#@out Violations_15_1
#@out Violations_13_1
#@out Violations_14_1
#@out Violations_12_1
#@out Violations_11_1
#@out Violations_10_1
#@out Violations_9_1
#@out Violations_8_1
#@out Violations_7_1
#@out Violations_6_1
#@out Violations_5_1
#@out Violations_4_1
#@out Violations_3_1
#@end core/column-split149
```

```
#@begin core/text-transform150 #@desc Text transform on cells in column Violations 1 using expression value.trim()
#@param expression:value.trim()
#@in Violations_1_1
#@out Violations_1_1
#@end core/text-transform150
#@begin core/text-transform151 #@desc Text transform on cells in column Violations 1 using expression value.replace(/[\\p{Zs}\\s]+/,' ')
#@param expression:value.replace(/[\\p{Zs}\\s]+/,'_')
#@in Violations_1_1
```

```
#@begin core/text-transform218 #@desc Text transform on cells in column Address - Clean using expression value.replace(/[\\p{Zs}\\s]+/,' ')
#@param expression:value.replace(/[\\p{Zs}\\s]+/,'_')
#@in Address_-_Clean
#@out Address_-_Clean_1
#@end core/text-transform218
#@begin core/mass-edit219 #@desc Mass edit cells in column Address - Clean
#@param expression:value
#@in Address_-_Clean_1
#@out Address_-_Clean_2
#@end core/mass-edit219
#@begin core/mass-edit220 #@desc Mass edit cells in column City - Clean
#@param expression:value
#@in City_-_Clean
#@out City_-_Clean_1
#@end core/mass-edit220
#@begin core/text-transform221 #@desc Text transform on cells in column State using expression grel:value.trim().toUppercase().replace(/[^\\u0020-\\u007F]/,\"\").replace(\"
\",\" \").replace(/[\\p{Zs}\\s]+/,\" \")
#@param expression:grel:value.trim().toUppercase().replace(/[^\\u0020-\\u007F]/,"").replace("_","_").replace(/[\\p{Zs}\\s]+/,"_")
#@in State
#@out State_1
#@end core/text-transform221
#@begin core/text-transform222 #@desc Text transform on cells in column State using expression grel:if(isBlank(value),'IL','IL')
#@param expression:grel:if(isBlank(value),'IL','IL')
#@in State_1
#@out State_2
#@end core/text-transform222
#@begin core/text-transform223 #@desc Text transform on cells in column Latitude using expression grel:value.trim().toUppercase().replace(/[^\\u0020-
\\u007F]/,\"\").replace(\" \",\" \").replace(/[\\p{Zs}\\s]+/,' ')
#@param expression:grel:value.trim().toUppercase().replace(/[^\\u0020-\\u007F]/,"").replace("_","_").replace(/[\\p{Zs}\\s]+/,'_')
#@in Latitude
#@out Latitude_1
#@end core/text-transform223
```

```
#@begin core/text-transform224 #@desc Text transform on cells in column Longitude using expression grel:value.trim().toUppercase().replace(/[^\\u0020-
\\u007F]/,\"\").replace(\" \",\" \").replace(/[\\p{Zs}\\s]+/,' ')
#@param expression:grel:value.trim().toUppercase().replace(/[^\\u0020-\\u007F]/,"").replace("_","_").replace(/[\\p{Zs}\\s]+/,'_')
#@in Longitude
#@out Longitude_1
#@end core/text-transform224
#@begin core/text-transform225 #@desc Text transform on cells in column Location using expression grel:value.trim().toUppercase().replace(/[^\\u0020-
\\u007F]/,\"\").replace(\" \",\" \").replace(/[\\p{Zs}\\s]+/,' ')
#@param expression:grel:value.trim().toUppercase().replace(/[^\\u0020-\\u007F]/,"").replace("_","_").replace(/[\\p{Zs}\\s]+/,'_')
#@in Location
#@out Location_1
#@end core/text-transform225
#@begin core/text-transform226 #@desc Text transform on cells in column Latitude using expression value.toNumber()
#@param expression:value.toNumber()
#@in Latitude_1
#@out Latitude_2
#@end core/text-transform226
#@begin core/text-transform227 #@desc Text transform on cells in column Longitude using expression value.toNumber()
#@param expression:value.toNumber()
#@in Longitude_1
#@out Longitude_2
#@end core/text-transform227
#@begin core/text-transform228 #@desc Text transform on cells in column Location using expression grel:value.replace(/\\(|\\)/,'')
#@param expression:grel:value.replace(/\\(|\\)/,'')
#@in Location_1
#@out Location_2
#@end core/text-transform228
```

Conclusions & Summary

We believe our analysis is as good as the quality of our data. As a data scientist, it would be a nightmare if the analytical results were wrong due to some incomplete or incorrect data values. Hence, checking data quality and identifying problem areas is of utmost importance before we can run any analysis on our data.

Data engineers spend a substantial amount of time and effort cleaning their data before it can be considered usable for their analysis. Without these actions, the research or analysis does not guarantee correct results. Detailing steps around how the data was gathered, how it was profiled, identified data quality issues, and how the data was cleaned and modeled for future access are all steps that are necessary to enrich and improve the data quality and also required in order to manage the provenance for replicating the results later.

We chose the Chicago Food Inspection dataset for our project. We used OpenRefine and Python as major tools to clean and enrich the data. We also used SQLite for performing data quality checks over cleaned data.

We first found a lot of discrepancies and data quality issues in the Chicago Food Inspection data we chose to be presented. It had IC violations, functional dependency issues, invalid data values, incomplete records, unidentified category values, non-standard names, and so on. As per our main use case (U1), our objective was to explore the effects of location on the risk evaluation of Chicago food inspections. This needed to make sure we cleaned up the main entities in our dataset: businesses and locations. After that, we performed our analysis on inspections and violations data. We had one large dataset with redundancies. Separating businesses and locations with their individual identifiers took the most effort and gave the highest quality rate, since these two will tie back to their factual data: inspections and violations. After this, we established the correct relationship between each inspection and its corresponding multiple violation observations by introducing primary key and foreign key relationships between them. We parsed the combined violation texts into separate violation text, violation code and severity so we can analyze them separately. We made sure the inspections data tied all the entities together, such that each inspection record connected a business to a unique valid location and listed all their zero-or-more violations. We believe this data is now clean enough to perform our analysis.

The challenges we faced were generally handling all the duplicate names or invalid or blank entries. We enriched data as much as possible to prevent data loss, but we had to delete some invalid records because we did not have much information on how to fill the gaps.

As the dataset is large, it was hard to check the integrity of each clustering step. Sometimes, we did not get all the expected clusters from OpenRefine, and had to rely on manual modification through facets.

We also faced slow run time and manual effort on imputing GREL commands, reviewing cluster groups, performing mass update, documenting all the data cleaning and enrichment steps. Even though tools like OpenRefine helped us perform these data cleaning steps efficiently, the whole process still required a lot of time. We used Python to improve the data operation performance and create the final, cleaned dataset. Additionally, running IC checks and checking validity are fast and easy in Python.

Considering our work distribution, we as a team planned the operations steps while preparing the Phase 1 document. After this, we distributed the work between all 3 team members based on the steps/tools used. The first team member worked on OpenRefine components and documented the steps, sharing the outcome for the next step. The second team member then took the outcome from the first step and ran Python scripts to further clean and model the data. The second member also shared the final cleaned datasets and documented the steps and ER diagram. The third team member ran the data quality checks on the final cleaned datasets and documented the queries. All the team members collaborated to prepare the final report.

Overall, there were no shortcuts to clean our data. A blend of various tools and techniques helped clean our data through a series of steps. We can guarantee that the data is good enough for our use cases.

Supplementary Materials

Provided below are some reference materials which we consulted while working with the data cleaning tools and arriving at a solution:

1. Dataset Metadata: https://data.cityofchicago.org/api/assets/BAD5301B-681A-4202-9D25-51B2CAE672FF
2. OpenRefine: https://gist.github.com/pmgreen/6e133c5dcde65762d29c
3. GREL: https://openrefine.org/docs/manual/grelfunctions#testing-string-characteristics
4. YesWorkflow: https://try.yesworkflow.org
5. SQLite: https://www.sqlite.org/docs.html

Along with web resources, the threads on CampusWire, weekly video course contents, work hour discussions and fireside chats helped us immensely in formalizing our thoughts and plans for this project.

Our final project artifacts along with Phase 1 and Phase 2 reports are located on Github: https://github.com/ratulsaha778/cs513-final-project.