(Team 137) Junyang Wang(jw111@illinois.edu)

CS513: Theory & Practice of Data Cleaning

University of Illinois Urbana-Champaign

Summer 2022

FINAL REPORT (Team 137)

Table of Contents

Final Project (Team 137) - Project Phase-I 2

Introduction and Overview 2

Data Set 2

Use Case 3

U0: sufficient with zero data cleaning 3

U1: target use case 3

U2: unrealistic use case 3

Dataset Description 4

Database Schema 4

Data Quality Problems 7

Final Project (Team 137) - Project Phase-II 10

Data Cleaning Process and Detail 10

Data Cleaning with OpenRefine 10

Data Cleaning with Python 14

Demonstrate Data Quality Improvements 17

Exploratory Data Analysis 19

Conclusions & Summary 20

Supplementary Materials 21

Final Project (Team 137) - Project Phase-I

Introduction and Overview

Restaurant inspections ensure that food served to the public at licensed food establishments follows food safety guidelines. The Food Protection Division of the Chicago Department of Public Health (CDPH) is committed to maintaining the safety of food bought, sold, or prepared for public consumption in Chicago by carrying out science-based inspections of all retail food establishments. These inspections promote public health in areas of food safety and sanitation and prevent the occurrence of food-borne illness. CDPH's licensed, accredited sanitarians inspect retail food establishments such as restaurants, grocery stores, bakeries, convenience stores, hospitals, nursing homes, day care facilities, shelters, schools, and temporary food service events. Inspections focus on food handling practices, product temperatures, personal hygiene, facility maintenance, and pest control.

All restaurants are subject to certain recurring inspections. Each year a restaurant is subject to annual inspections to ensure continued compliance with City ordinances and regulations. In addition to recurring inspections, restaurants may also be inspected in response to a complaint. Some of these recurring inspections, such as the inspection by the Buildings Department, will be scheduled, while others will not.

Generally, inspections are conducted by the Health Department for sanitation and safe food handling practices, the Buildings Department to ensure the safety of the structure, and the Fire Department to ensure safe fire exits. The City's Dumpster Task Force, a collaborative effort between the Health Department and Streets and Sanitation Department, also inspects restaurants to ensure compliance with sanitation regulations.

Data Set

For this project, I have selected the Chicago-Food-Inspection dataset from https://www.kaggle.com/datasets/chicago/chi-restaurant-inspections. This information is derived from inspections of restaurants and other food establishments in Chicago from January 1, 2010 to the present. Inspections are performed by staff from the Chicago Department of Public Health’s Food Protection Program using a standardized procedure.

Use Case

The usefulness of data can be judged by the potential use case it may serve. The raw data in its original form has many issues but may still be valuable for several analytic scenarios and use cases. However, once cleaned, further uses can be foreseen, some of them are listed below:

U0: sufficient with zero data cleaning

Although the data in raw form is not clean and has several anomalies, it may still serve several use cases, these are just a sample of some of the possible use cases with this dataset:

- How many Chicago food establishments have pass the food inspection?

  The dataset in its original form would support this query. Since the column of which has the result of the food inspection are entirely populated with no missing data, this use case can be easily answered without any data cleaning effort. Select count on column inspection results with "Pass" would yield the correct answer for the given question. Any refinement of the data would not have an effect on the final result to the given question.

- How many food establishments inspection in Chicago?
  We can answer this question without any data cleaning because each unique id corresponds to a single inspection record, and we simply want an understanding of the volume of total records regardless of the other features of the data, we do not need to do any cleaning to answer the question. Select count (*) from food_inspection.csv alone would yield the correct answer for the given question.

U1: target use case

Once the initial quality issues have been addressed the data will be fit for the following and other similar use cases:

- What is the number of Church Kitchen in Chicago have pass the food inspection?
  To answer the question, we need to select all the inspection record with facility type as restaurant and then get the percentage by count the number of selected inspection result with "Pass" and the total number of the record selected. With the dataset in the current uncleaned state, there are many different type of facility type (Restaurant, RESTAURANT/GROCERY STORE, GROCERY/RESTAURANT, RESTAURANT.BANQUET HALLS …..) . while the structure of the data gives us the capability to attempt to answer this question, the inconsistencies in the facility type filed largely affect the true number of the restaurant inspection records, thereby throwing off the percentage computations.

U2: unrealistic use case

- What are the conditional requirements on some of the food establishments?
  Even though some of the food business are issued with Pass w/ Conditions as the inspection result, there is no column corresponding to the condition data. This require outside knowledge of conditional requirements issued during the inspection, so data cleaning will not be sufficient to resolve this question.

Dataset Description

The food inspection data set consists of records from 01/02/2013 to 08/28/17 derived from inspections of restaurants and other food establishments in Chicago. There are 17 columns in total, and 153810 total records.

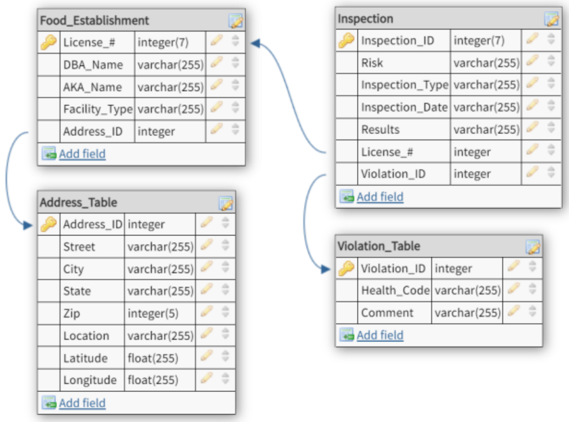


Figure 1: relational Schema

The food_inspection.csv dataset is a single database table; the large number of columns appears to reveal a bad design that was made unnecessarily complicated. I represented the data into 4 tables; food_establishment, address, Inspection and violation. I did this because the relations in each individual table pertains to one “thing”, which the table name describes. Tying together, the tables using foreign keys allows the 4 tables to share their attributes without duplicating the data across multiple tables.

Database Schema

For Food_Establishment schema, we used License_# attribute as the primary key as it is unique for every food business in the dataset. By default, the business License number should not be Null. The Address_ID is the foreign key for address table.

```
CREATE TABLE `Food_Establishment` (

        `License_#` INT(7) NOT NULL,

        `DBA_Name` varchar(255) NOT NULL,

        `AKA_Name` varchar(255),

        `Facility_Type` varchar(255) NOT NULL,

        `Address_id` INT NOT NULL,
	PRIMARY KEY (`License_#`)
);
```

Food_Establishment table Attributes:

- License_#: primary key, unique business license issued by government agencies
- DBA_Name: “doing business as,” or trade name
- AKA_Name: “Also Known As” name
- Facility_Type: type of food business (Restaurant, Special Event, Grocery Store, etc...)
- Address_ID: foreign key for the address table

For Address_Table schema, we used Address_ID attribute as the primary key. The Address_ID attribute is auto increment and should not be null.

```
CREATE TABLE `Address_Table` (
	`Address_ID` INT NOT NULL AUTO_INCREMENT,
	`City` varchar(255) NOT NULL,
	`State` varchar(255) NOT NULL,
	`Zip` INT(5) NOT NULL,
	`Location` varchar(255) NOT NULL,
	`Latitude` FLOAT(255) NOT NULL,
	`Longitude` FLOAT(255) NOT NULL,
	PRIMARY KEY (`Address_id`)
);
```

Address_Table Attributes:

- Address_ID: primary key,
- Street: street address for the food business
- City: city address for the food business
- State: state address for the food business
- Zip: address zip code
- Location: Latitude and Longitude of the address, hold repeating information as Latitude and Longitude attributes, redundant.
- Latitude: Latitude of the address
- Longitude: Longitude of the address

For Inspection schema, we used Inspection_ID attribute as the primary key. Inspection_ID is unique for every inspection records in the dataset. By default, the Inspection_ID number should not be Null. The License_# is the foreign key for Food_Establishment table.

```
CREATE TABLE `Inspection` (
        `Inspection_ID` INT(7) NOT NULL,
        `Risk` varchar(255) NOT NULL,
        `Inspection_Type` varchar(255) NOT NULL,
        `Inspection_Date` varchar(255) NOT NULL,
        `Results` varchar(255) NOT NULL,
        `License_#` INT NOT NULL,
        `Violations_ID` INT,
        PRIMARY KEY (`Inspection_ID`)
);
```

Inspection Table Attributes:

- Inspection_ID: primary key,
- Risk: risk level evaluated during the inspection (Risk 1 (High), Risk 2 (Medium), Risk 3 (Low))
- Inspection_Type: type of inspection
- Inspection_Date: date information
- Results: Inspection result (Pass, Pass w/ Conditions, Not Ready, Fail, Out of Business)
- License_#: Latitude of the address
- Violations_ID: Longitude of the address

For Violation_Table schema, we used Violations_ID attribute as the primary key. The Violations_ID attribute is auto increment and should not be null.

```
CREATE TABLE `Violation_Table` (
        `Violation_ID` INT NOT NULL AUTO_INCREMENT,
        `Health_Code` varchar(255),
        `Comment` varchar(255),
        PRIMARY KEY (`Violations_ID`)
);
```

Violation_Table Attributes:

- Violation_ID: primary key,

- Health_Code: health code violations by the food business
- Comment: additional comment about the violation

ALTER TABLE `Food_Establishment` ADD CONSTRAINT `Food_Establishment_fk0` FOREIGN KEY (`Address_id`) REFERENCES `Address_Table`(`Address_id`);

ALTER TABLE `Inspection` ADD CONSTRAINT `Inspection_fk0` FOREIGN KEY (`License_#`) REFERENCES `Food_Establishment`(`License_#`);

ALTER TABLE `Inspection` ADD CONSTRAINT `Inspection_fk1` FOREIGN KEY (`Violations_ID`) REFERENCES `Violation_Table`(`Health_Code`);

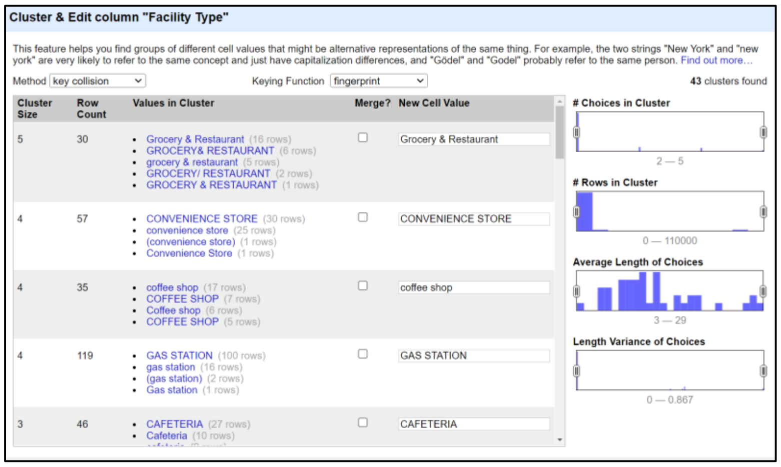
Data Quality Problems

A quick inspection of the dataset suggests there exists the data quality issues, such as missing data, redundant data, Format Issues such as date format, Data Type issues such as numeric columns represented as String and Data represented in different cases such as upper case, lower case etc.

Player	# of null values	% of null values
Inspection ID	0	0%
DBA Name	0	0%
AKA Name	2543	1.65%
License #	15	0.01%
Facility Type	4560	2.96%
Risk	66	0.04%
Address	1	0%
City	159	0.1%
State	8	0.01%
Zip	98	0.06%
Inspection Date	0	0%
Inspection Type	1	0.01%
Results	0	0%
Violations	30798	20.02%

Latitude	544	0.35%
Longitude	544	0.35%
Location	544	0.35%

Table 1: Table of Missing value counts and percentages for each attribute



Cluster & Edit column "Facility Type"

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…

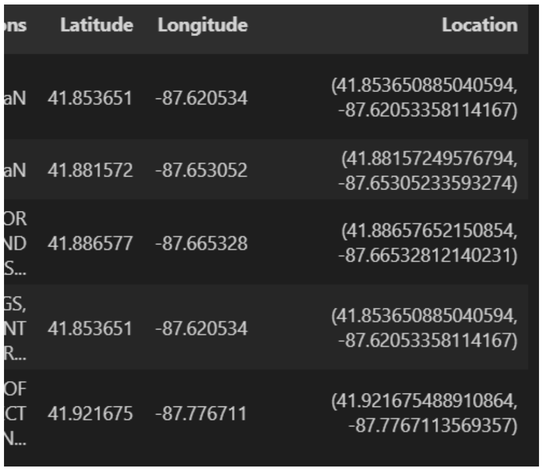
Method: key collision — Keying Function: fingerprint — **43** clusters found

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 (1 rows)	☐	Grocery & Restaurant
4	57	• CONVENIENCE STORE (30 rows) • convenience store (25 rows) • (convenience store) (1 rows) • Convenience Store (1 rows)	☐	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 (1 rows)	☐	GAS STATION
3	46	• CAFETERIA (27 rows) • Cafeteria (10 rows)	☐	CAFETERIA

Figure 2: Cluster for Facility Type column

In this table, we can see that the inspection id, business name, address, inspection data, type and result are mostly filled. license number is vital for identify different food business. however, there are 15 missing value in the license number attribute, which will need to be addressed in order for future analysis.

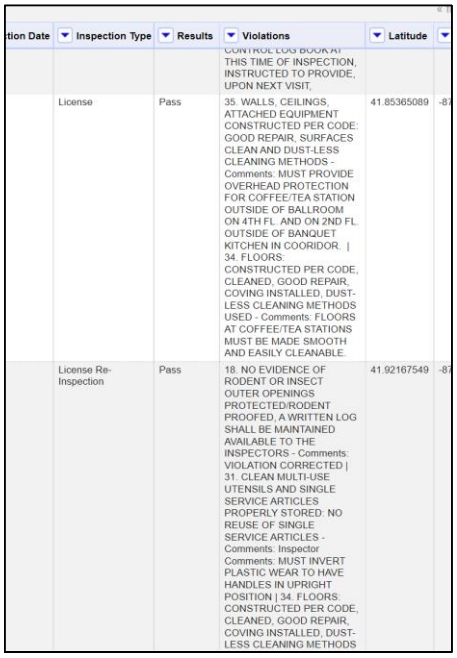
Another data quality issue is the format and spelling in the facility type column. There are multiple values with spelling but in different font upper or lowercase and punctuations.



ns	Latitude	Longitude	Location
aN	41.853651	-87.620534	(41.853650885040594, -87.62053358114167)
aN	41.881572	-87.653052	(41.88157249576794, -87.65305233593274)
OR ND S...	41.886577	-87.665328	(41.88657652150854, -87.66532812140231)
GS, NT R...	41.853651	-87.620534	(41.853650885040594, -87.62053358114167)
OF CT N...	41.921675	-87.776711	(41.921675488910864, -87.7767113569357)

Figure 3: latitude, longitude and location column

Location column hold same GPS coordinates data as latitude and longitude, so it is redundant and should be removed.



tion Date	Inspection Type	Results	Violations	Latitude	
			CONTROL LOG BOOK AT THIS TIME OF INSPECTION, INSTRUCTED TO PROVIDE, UPON NEXT VISIT,		
	License	Pass	35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Comments: MUST PROVIDE OVERHEAD PROTECTION FOR COFFEE/TEA STATION OUTSIDE OF BALLROOM ON 4TH FL. AND ON 2ND FL. OUTSIDE OF BANQUET KITCHEN IN COORIDOR. | 34. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUST-LESS CLEANING METHODS USED - Comments: FLOORS AT COFFEE/TEA STATIONS MUST BE MADE SMOOTH AND EASILY CLEANABLE.	41.85365089	-87
	License Re-Inspection	Pass	18. NO EVIDENCE OF RODENT OR INSECT OUTER OPENINGS PROTECTED/RODENT PROOFED, A WRITTEN LOG SHALL BE MAINTAINED AVAILABLE TO THE INSPECTORS - Comments: VIOLATION CORRECTED | 31. CLEAN MULTI-USE UTENSILS AND SINGLE SERVICE ARTICLES PROPERLY STORED: NO REUSE OF SINGLE SERVICE ARTICLES - Comments: Inspector Comments: MUST INVERT PLASTIC WEAR TO HAVE HANDLES IN UPRIGHT POSITION | 34. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUST-LESS CLEANING METHODS	41.92167549	-87

Figure 4: violations column

Finally, the violations seem to have both health code violations and addition comments in the same column, which could obfuscate the user`s ability to find and analysis useful information in the records. We may need to spilt the violations column into two separate columns.

Final Project (Team 137) - Project Phase-II

Data Cleaning Process and Detail

In order to facilitate our use case, we found that leveraging OpenRefine, Python, and SQL Server technologies would best suite establishing a clean data set. We incorporated two main steps to our data cleaning workflow which resembled a pipeline. From the simple outer workflow, each of these two steps has been further broken down into inner workflows using YesWorkFlow. Starting from the raw data file from food inspection, we began the process through OpenRefine. With the file generated from OpenRefine, we then used Python to conduct further cleaning.

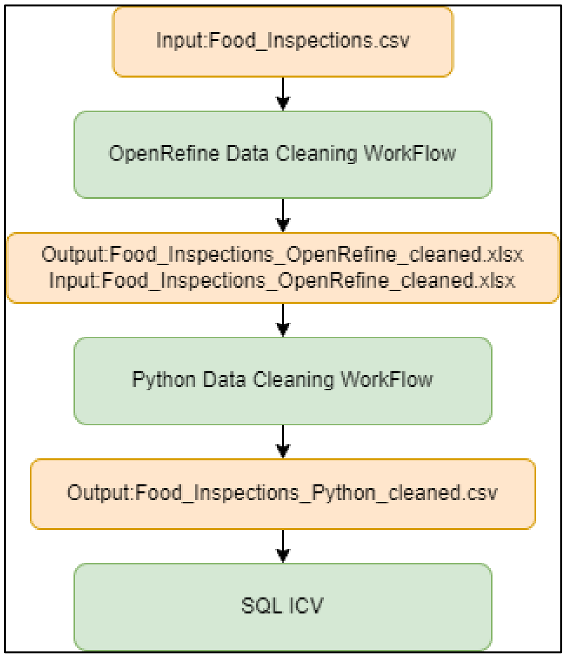

Input:Food_Inspections.csv
OpenRefine Data Cleaning WorkFlow
Output:Food_Inspections_OpenRefine_cleaned.xlsx
Input:Food_Inspections_OpenRefine_cleaned.xlsx
Python Data Cleaning WorkFlow
Output:Food_Inspections_Python_cleaned.csv
SQL ICV

Figure 1: Outer Workflow

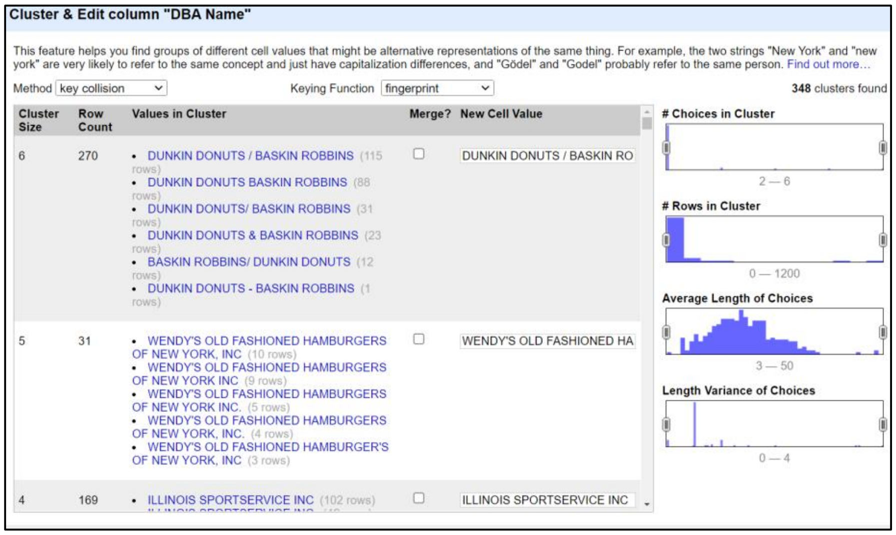
Data Cleaning with OpenRefine

Within the tool OpenRefine, the data was cleansed column by column. This was not required specifically to address the answer to our use case, but was required to support our efforts to establish a data set that would be suitable for ingesting into SQL Server where our integrity checks were to be established on the data set.
OpenRefine: Inconsistent naming, syntax, duplication, and typos were nearly all resolved with series of OpenRefine operations. The result was a nearly syntactically clean data set ready for schematic fixes.

- DBA Name

OpenRefine Operations	# of Cells modified
Trim leading and trailing white space. Collapse consecutive white spaces	3322
Text transform to Uppercase	10443

Mass edit by cluster using key collision method	10603
Mass edit by cluster using nearest neighbor method	38


Cluster & Edit column "DBA Name"

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: fingerprint — 348 clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
6	270	• DUNKIN DONUTS / BASKIN ROBBINS (115 rows) • DUNKIN DONUTS BASKIN ROBBINS (88 rows) • DUNKIN DONUTS/ BASKIN ROBBINS (31 rows) • DUNKIN DONUTS & BASKIN ROBBINS (23 rows) • BASKIN ROBBINS/ DUNKIN DONUTS (12 rows) • DUNKIN DONUTS - BASKIN ROBBINS (1 rows)	☐	DUNKIN DONUTS / BASKIN RO
5	31	• WENDY'S OLD FASHIONED HAMBURGERS OF NEW YORK, INC (10 rows) • WENDY'S OLD FASHIONED HAMBURGERS OF NEW YORK INC (9 rows) • WENDY'S OLD FASHIONED HAMBURGERS OF NEW YORK INC. (5 rows) • WENDY'S OLD FASHIONED HAMBURGERS OF NEW YORK, INC. (4 rows) • WENDY'S OLD FASHIONED HAMBURGER'S OF NEW YORK, INC (3 rows)	☐	WENDY'S OLD FASHIONED HA
4	169	• ILLINOIS SPORTSERVICE INC (102 rows)	☐	ILLINOIS SPORTSERVICE INC

Figure 2: Clustering and Cleaning DBA Name column

- AKA Name

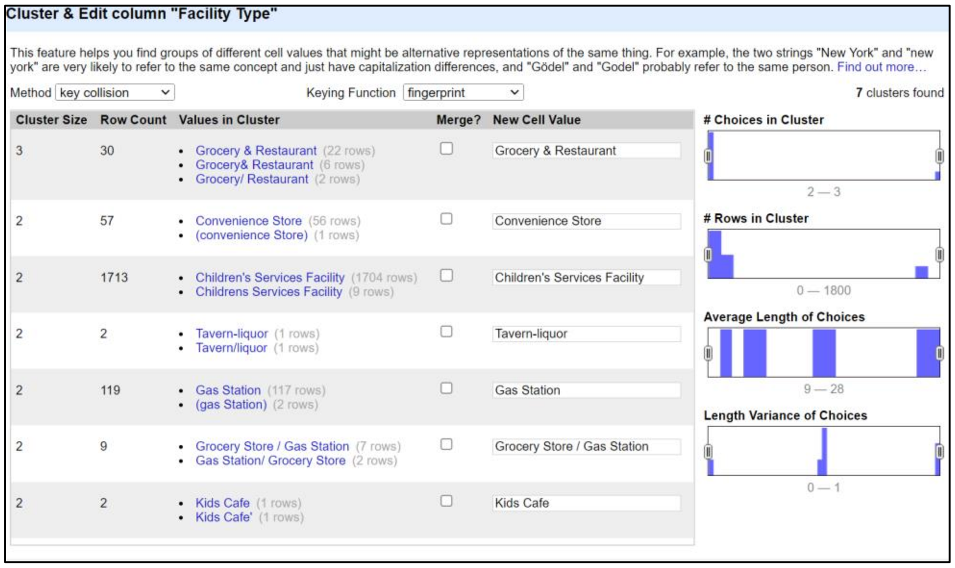
OpenRefine Operations	# of Cells modified
Trim leading and trailing white space. Collapse consecutive white spaces	3560
Text transform to Uppercase	9594

- License #

OpenRefine Operations	# of Cells modified
Filter and remove records with an empty License #	15 removed

- Facility Type

OpenRefine Operations	# of Cells modified
Trim leading and trailing white space. Collapse consecutive white spaces	13
Text transform to Titlecase	4984
Mass edit by cluster using key collision method	1932



Cluster & Edit column "Facility Type"

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: fingerprint — 7 clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
3	30	• Grocery & Restaurant (22 rows) • Grocery& Restaurant (6 rows) • Grocery/ Restaurant (2 rows)	☐	Grocery & Restaurant
2	57	• Convenience Store (56 rows) • (convenience Store) (1 rows)	☐	Convenience Store
2	1713	• Children's Services Facility (1704 rows) • Childrens Services Facility (9 rows)	☐	Children's Services Facility
2	2	• Tavern-liquor (1 rows) • Tavern/liquor (1 rows)	☐	Tavern-liquor
2	119	• Gas Station (117 rows) • (gas Station) (2 rows)	☐	Gas Station
2	9	• Grocery Store / Gas Station (7 rows) • Gas Station/ Grocery Store (2 rows)	☐	Grocery Store / Gas Station
2	2	• Kids Cafe (1 rows) • Kids Cafe' (1 rows)	☐	Kids Cafe

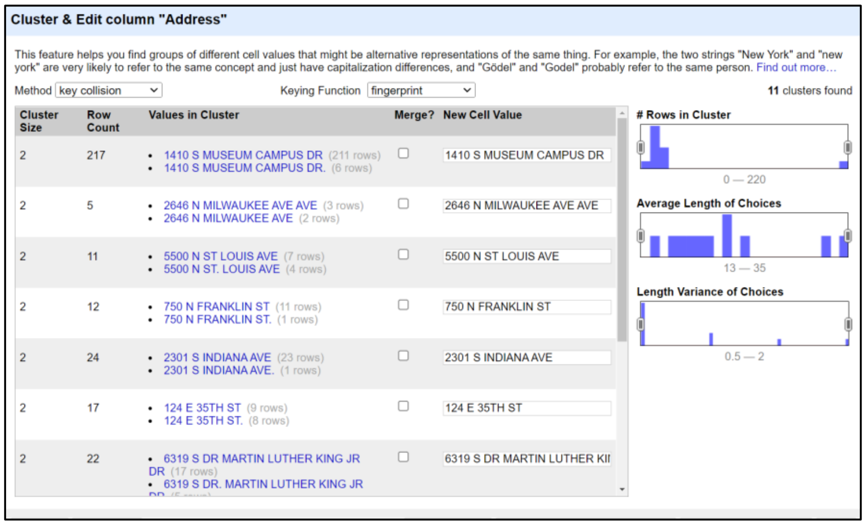
Figure 2: Clustering and Cleaning Facility Type column

- Risk

OpenRefine Operations	# of Cells modified
Filter and remove records with an empty Risk cell	66 removed

- Address

OpenRefine Operations	# of Cells modified
Trim leading and trailing white space. Collapse consecutive white spaces	618
Text transform to Uppercase	9488
Mass edit by cluster using key collision method	344



Cluster & Edit column "Address"

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…

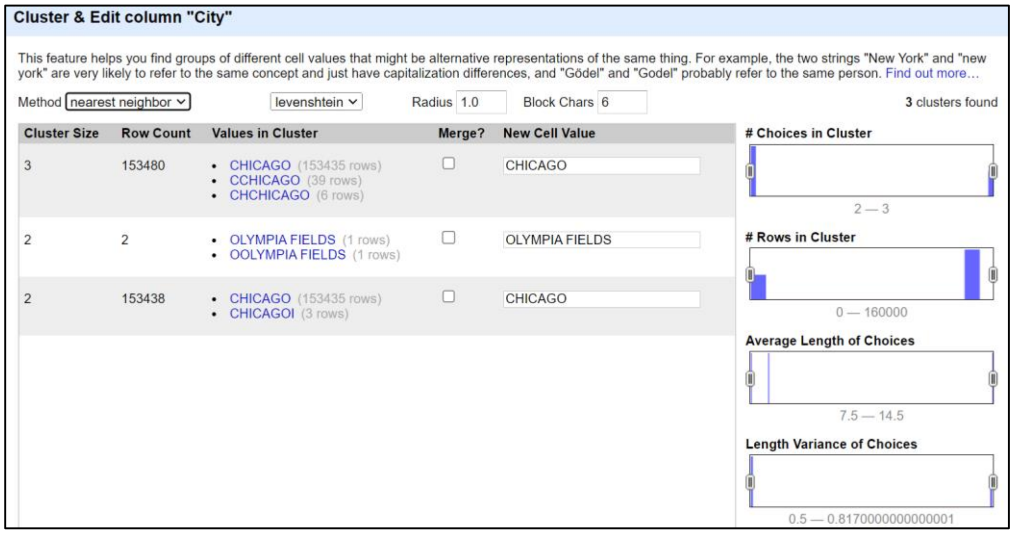
Method: key collision — Keying Function: fingerprint — 11 clusters found

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. (1 rows)	☐	750 N FRANKLIN ST
2	24	• 2301 S INDIANA AVE (23 rows) • 2301 S INDIANA AVE. (1 rows)	☐	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 KING JR DR (17 rows) • 6319 S DR. MARTIN LUTHER KING JR	☐	6319 S DR MARTIN LUTHER KII

Figure 3: Clustering and Cleaning Address column

- City

OpenRefine Operations	# of Cells modified
Text transform to Uppercase	348
Mass edit by cluster using nearest neighbor method	153485



Cluster & Edit column "City"

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: nearest neighbor — levenshtein — Radius 1.0 — Block Chars 6 — **3** clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
3	153480	• CHICAGO (153435 rows) • CCHICAGO (39 rows) • CHCHICAGO (6 rows)	☐	CHICAGO
2	2	• OLYMPIA FIELDS (1 rows) • OOLYMPIA FIELDS (1 rows)	☐	OLYMPIA FIELDS
2	153438	• CHICAGO (153435 rows) • CHICAGOI (3 rows)	☐	CHICAGO

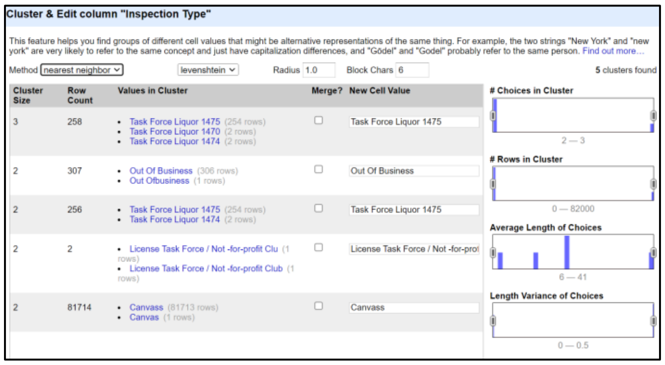
Figure 4: Clustering and Cleaning City column

- Zip

OpenRefine Operations	# of Cells modified
Filter and remove records with an empty Zip code	98 removed

- Inspection Type

OpenRefine Operations	# of Cells modified
Text transform to Titlecase	29926
Mass edit by cluster using nearest neighbor method	82023
Filter and remove records with an empty Inspection Type number	1 removed



Cluster & Edit column "Inspection Type"

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: nearest neighbor — levenshtein — Radius 1.0 — Block Chars 6 — **5** clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
3	258	• Task Force Liquor 1475 (254 rows) • Task Force Liquor 1470 (2 rows) • Task Force Liquor 1474 (2 rows)	☐	Task Force Liquor 1475
2	307	• Out Of Business (306 rows) • Out Ofbusiness (1 rows)	☐	Out Of Business
2	256	• Task Force Liquor 1475 (254 rows) • Task Force Liquor 1474 (2 rows)	☐	Task Force Liquor 1475
2	2	• License Task Force / Not -for-profit Clu (1 rows) • License Task Force / Not -for-profit Club (1 rows)	☐	License Task Force / Not -for-prof
2	81714	• Canvass (81713 rows) • Canvas (1 rows)	☐	Canvass

Figure 5: Clustering and Cleaning Inspection Type column

- Location

OpenRefine Operations	# of Cells modified
Remove Location column	153810 removed

The YesWorkFlow model for OpenRefine is shown in Figure 6 and provides each step in our data cleaning process. This YesWorkFlow diagram is translated from OpenRefine Json history using OR2YWTool.

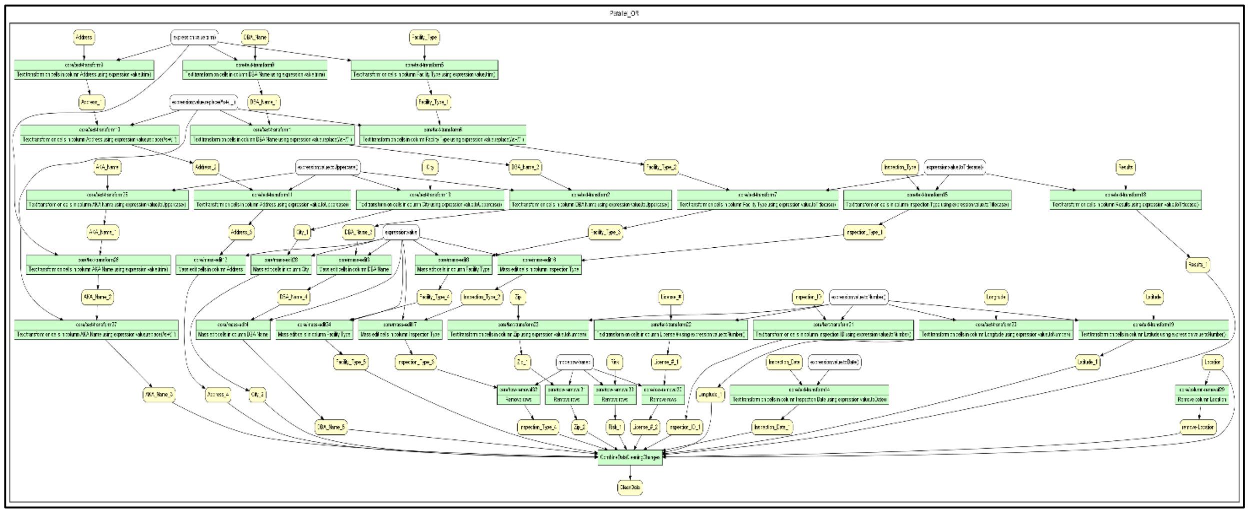
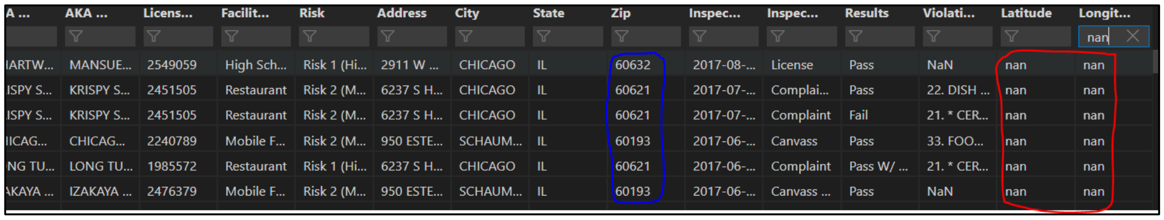


Figure 6: OpenRefine YesWorkFlow Diagram

Data Cleaning with Python

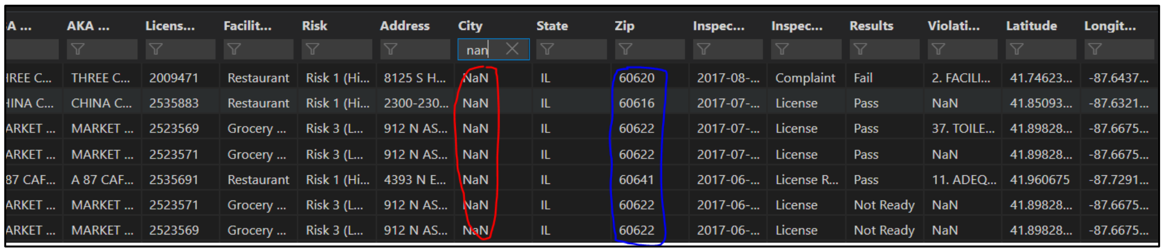
OpenRefine resolved many of the data quality issues required to solve our use case, but given the poor data quality that remained in the data set, additional processing was done leveraging Python.

The Chicago Food Inspection dataset has servals records with missing city, state, Latitude and longitude information. To answer the use case to find the location of food business, we must have all the location information available in the clean dataset.



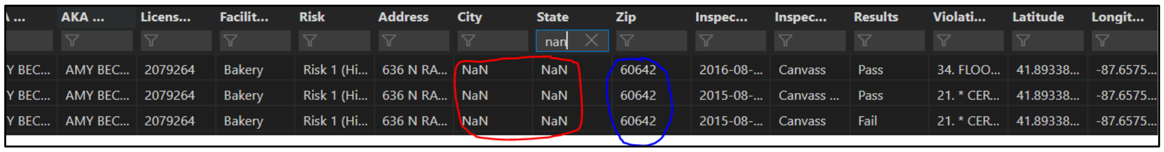
A ...	AKA ...	Licens...	Facilit...	Risk	Address	City	State	Zip	Inspec...	Inspec...	Results	Violati...	Latitude	Longit...
														nan
IARTW...	MANSUE...	2549059	High Sch...	Risk 1 (Hi...	2911 W ...	CHICAGO	IL	60632	2017-08-...	License	Pass	NaN	nan	nan
ISPY S...	KRISPY S...	2451505	Restaurant	Risk 2 (M...	6237 S H...	CHICAGO	IL	60621	2017-07-...	Complai...	Pass	22. DISH ...	nan	nan
ISPY S...	KRISPY S...	2451505	Restaurant	Risk 2 (M...	6237 S H...	CHICAGO	IL	60621	2017-07-...	Complaint	Fail	21. * CER...	nan	nan
IICAG...	CHICAG...	2240789	Mobile F...	Risk 2 (M...	950 ESTE...	SCHAUM...	IL	60193	2017-06-...	Canvass	Pass	33. FOO...	nan	nan
NG TU...	LONG TU...	1985572	Restaurant	Risk 1 (Hi...	6237 S H...	CHICAGO	IL	60621	2017-06-...	Complaint	Pass W/ ...	21. * CER...	nan	nan
^KAYA ...	IZAKAYA ...	2476379	Mobile F...	Risk 2 (M...	950 ESTE...	SCHAUM...	IL	60193	2017-06-...	Canvass ...	Pass	NaN	nan	nan

Figure 7: Missing GPS coordinates value



A ...	AKA ...	Licens...	Facilit...	Risk	Address	City	State	Zip	Inspec...	Inspec...	Results	Violati...	Latitude	Longit...
						nan								
IREE C...	THREE C...	2009471	Restaurant	Risk 1 (Hi...	8125 S H...	NaN	IL	60620	2017-08-...	Complaint	Fail	2. FACILI...	41.74623...	-87.6437...
IINA C...	CHINA C...	2535883	Restaurant	Risk 1 (Hi...	2300-230...	NaN	IL	60616	2017-07-...	License	Pass	NaN	41.85093...	-87.6321...
ARKET ...	MARKET ...	2523569	Grocery ...	Risk 3 (L...	912 N AS...	NaN	IL	60622	2017-07-...	License	Pass	37. TOILE...	41.89828...	-87.6675...
ARKET ...	MARKET ...	2523571	Grocery ...	Risk 3 (L...	912 N AS...	NaN	IL	60622	2017-07-...	License	Pass	NaN	41.89828...	-87.6675...
87 CAF...	A 87 CAF...	2535691	Restaurant	Risk 1 (Hi...	4393 N E...	NaN	IL	60641	2017-06-...	License R...	Pass	11. ADEQ...	41.960675	-87.7291...
ARKET ...	MARKET ...	2523571	Grocery ...	Risk 3 (L...	912 N AS...	NaN	IL	60622	2017-06-...	License	Not Ready	NaN	41.89828...	-87.6675...
ARKET ...	MARKET ...	2523569	Grocery ...	Risk 3 (L...	912 N AS...	NaN	IL	60622	2017-06-...	License	Not Ready	NaN	41.89828...	-87.6675...

Figure 8: Missing City value



...	AKA ...	Licens...	Facilit...	Risk	Address	City	State	Zip	Inspec...	Inspec...	Results	Violati...	Latitude	Longit...
							nan							
Y BEC...	AMY BEC...	2079264	Bakery	Risk 1 (Hi...	636 N RA...	NaN	NaN	60642	2016-08-...	Canvass	Pass	34. FLOO...	41.89338...	-87.6575...
Y BEC...	AMY BEC...	2079264	Bakery	Risk 1 (Hi...	636 N RA...	NaN	NaN	60642	2015-08-...	Canvass ...	Pass	21. * CER...	41.89338...	-87.6575...
Y BEC...	AMY BEC...	2079264	Bakery	Risk 1 (Hi...	636 N RA...	NaN	NaN	60642	2015-08-...	Canvass	Fail	21. * CER...	41.89338...	-87.6575...

Figure 9: Missing City and State value

Fortunately, the zip code of the food inspection record is complete with no missing data. I can populate the missing city, state, Latitude and longitude information based on the zip code. below are the major cleaning steps:

1. Find and create Dataframe Loc for dataset with missing key city, state or GPS coordinates fields but have zip code populated.
2. Populate the missing the Latitude and Longitude based on the Zip code.
3. Populate the missing City and State based on the Zip code.
4. Rescan the dataset to check if there still exists any missing key value.

Here is the end result of python data cleaning:

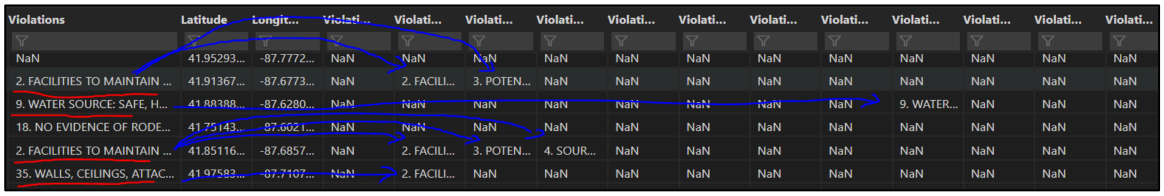
Player	# of null values	% of null values
Inspection ID	0	0%
DBA Name	0	0%
AKA Name	2487	1.62%
License #	0	0%
Facility Type	4508	2.93%
Risk	0	0%
Address	1	0.0001%
City	0	0%
State	0	0%
Zip	0	0%
Inspection Date	0	0%
Inspection Type	0	0%
Results	0	0%
Violations	30798	19.97%

Latitude	0	0%
Longitude	0	0%

Table 1: Table of Missing value counts and percentages for each attribute

The violations column seems to have both health code violations and addition comments in the same column, which could obfuscate the user`s ability to find and analysis useful information in the records. I can to spilt the violations column into 46 separate columns, each represent 1 violation code with its addition comment. below are the major cleaning steps:

1. Loop through violations column to find all the unique violations codes (46 total).
2. Add 46 new column to the dataset, each represent a corresponding violation code.
3. Loop through violations column again, split and add each violations code and comments to each corresponding new column.
4. Clean and remove the old violations column.



Violations	Latitude	Longit...	Violati...	Violati...	Violati...	Violati...	Violati...	Violati...	Violati...	Violati...	Violati...	Violati...	Violati...	Violati...
NaN	41.95293...	-87.7772...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2. FACILITIES TO MAINTAIN ...	41.91367...	-87.6773...	NaN	2. FACILI...	3. POTEN...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9. WATER SOURCE: SAFE, H...	41.88388...	-87.6280...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	9. WATER...	NaN	NaN	NaN
18. NO EVIDENCE OF RODE...	41.75143...	-87.6021...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2. FACILITIES TO MAINTAIN ...	41.85116...	-87.6857...	NaN	2. FACILI...	3. POTEN...	4. SOUR...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
35. WALLS, CEILINGS, ATTAC...	41.97583...	-87.7107...	NaN	2. FACILI...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 10: move violations column to corresponding new violation columns

The YesWorkFlow model for Python cleaning is shown in Figure 11 and provides each step in our data cleaning process.

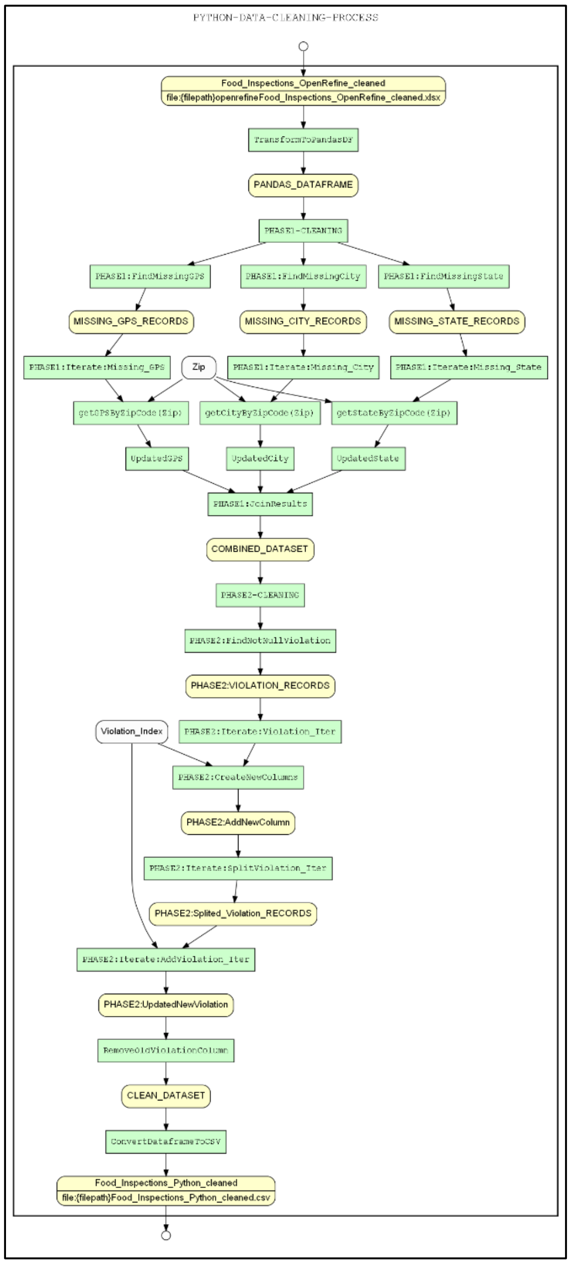

PYTHON-DATA-CLEANING-PROCESS
Food_Inspections_OpenRefine_cleaned
file:{filepath}openrefineFood_Inspections_OpenRefine_cleaned.xlsx
TransformToPandasDF
PANDAS_DATAFRAME
PHASE1-CLEANING
PHASE1:FindMissingGPS
PHASE1:FindMissingCity
PHASE1:FindMissingState
MISSING_GPS_RECORDS
MISSING_CITY_RECORDS
MISSING_STATE_RECORDS
PHASE1:Iterate:Missing_GPS
Zip
PHASE1:Iterate:Missing_City
PHASE1:Iterate:Missing_State
getGPSByZipCode(Zip)
getCityByZipCode(Zip)
getStateByZipCode(Zip)
UpdatedGPS
UpdatedCity
UpdatedState
PHASE1:JoinResults
COMBINED_DATASET
PHASE2-CLEANING
PHASE2:FindNotNullViolation
PHASE2:VIOLATION_RECORDS
Violation_Index
PHASE2:Iterate:Violation_Iter
PHASE2:CreateNewColumns
PHASE2:AddNewColumn
PHASE2:Iterate:SplitViolation_Iter
PHASE2:Splited_Violation_RECORDS
PHASE2:Iterate:AddViolation_Iter
PHASE2:UpdatedNewViolation
RemoveOldViolationColumn
CLEAN_DATASET
ConvertDataframeToCSV
Food_Inspections_Python_cleaned
file:{filepath}Food_Inspections_Python_cleaned.csv

Figure 11: Python YesWorkFlow Diagram

Demonstrate Data Quality Improvements

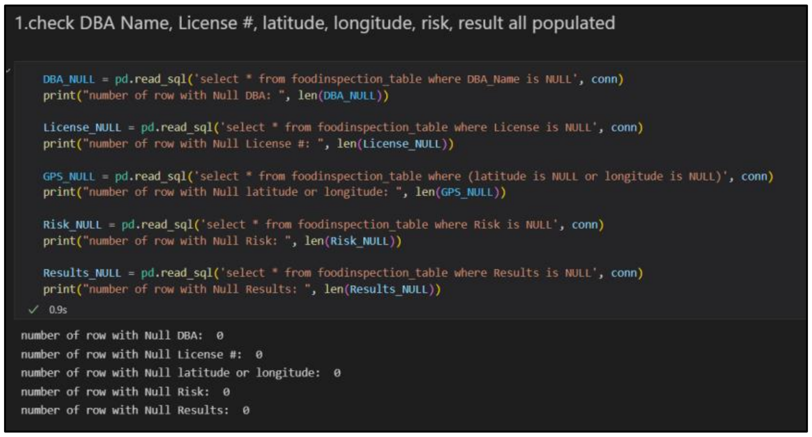
Integrity Constraints Violations Check was performed to ensure our queries for our main use case perform as expected. The sql query provided in this section has also been included in the sqlite.ipynb. below are few integrity constraints which we ran in sqlite.ipynb notebook:

• Ensure that data for major use case is non-null (specifically DBA name, License, latitude, longitude, risk, result)

• latitude must be in [0,90] and longitude should be [-180, 180]

• Every License number has unique address

The following screen shots provide validation that these integrity constraints hold.



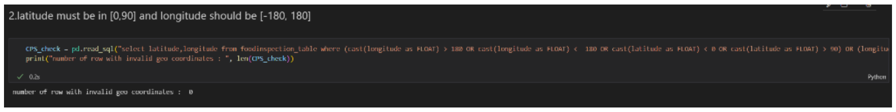
1.check DBA Name, License #, latitude, longitude, risk, result all populated

```
DBA_NULL = pd.read_sql('select * from foodinspection_table where DBA_Name is NULL', conn)
print("number of row with Null DBA: ", len(DBA_NULL))

License_NULL = pd.read_sql('select * from foodinspection_table where License is NULL', conn)
print("number of row with Null License #: ", len(License_NULL))

GPS_NULL = pd.read_sql('select * from foodinspection_table where (latitude is NULL or longitude is NULL)', conn)
print("number of row with Null latitude or longitude: ", len(GPS_NULL))

Risk_NULL = pd.read_sql('select * from foodinspection_table where Risk is NULL', conn)
print("number of row with Null Risk: ", len(Risk_NULL))

Results_NULL = pd.read_sql('select * from foodinspection_table where Results is NULL', conn)
print("number of row with Null Results: ", len(Results_NULL))
```

✓ 0.9s

```
number of row with Null DBA:  0
number of row with Null License #:  0
number of row with Null latitude or longitude:  0
number of row with Null Risk:  0
number of row with Null Results:  0
```

Figure 12:number of null rows for the key columns



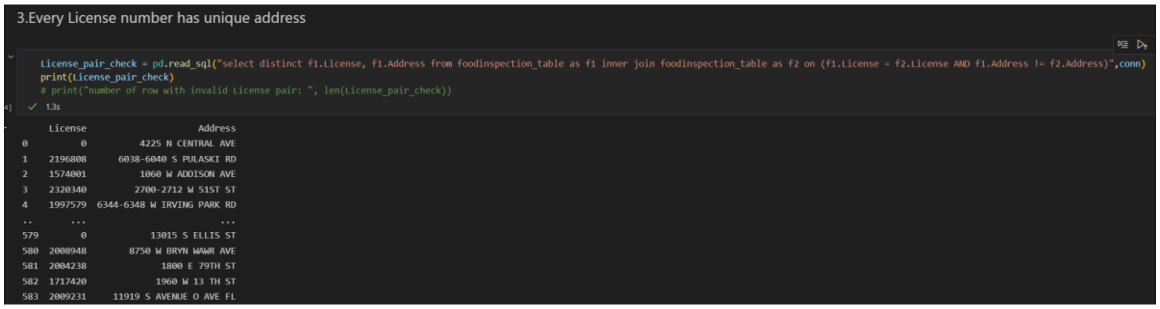
2.latitude must be in [0,90] and longitude should be [-180, 180]

```
GPS_check = pd.read_sql("select latitude,longitude from foodinspection_table where (cast(longitude as FLOAT) > 180 OR cast(longitude as FLOAT) < -180 OR cast(latitude as FLOAT) < 0 OR cast(latitude as FLOAT) > 90) OR (longitu
print("number of row with invalid geo coordinates : ", len(GPS_check))
```

✓ 0.2s Python

```
number of row with invalid geo coordinates :  0
```

Figure 13: number of rows with invalid geo coordinates



3.Every License number has unique address

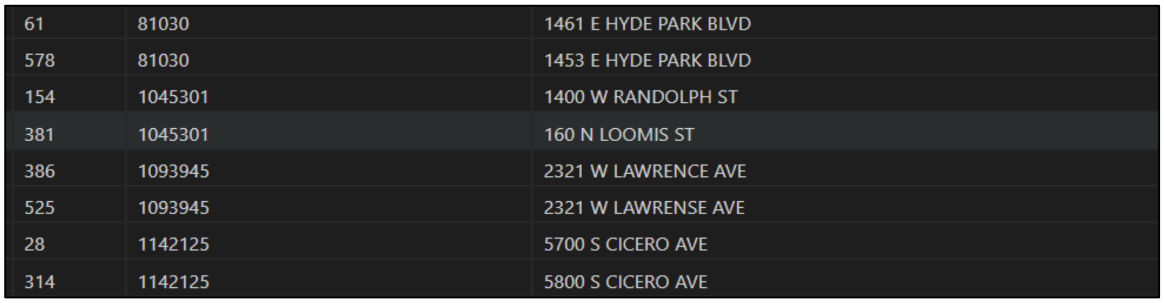
```
License_pair_check = pd.read_sql("select distinct f1.License, f1.Address from foodinspection_table as f1 inner join foodinspection_table as f2 on (f1.License = f2.License AND f1.Address != f2.Address)",conn)
print(License_pair_check)
# print("number of row with invalid License pair: ", len(License_pair_check))
```

✓ 1.3s

```
     License                Address
0          0        4225 N CENTRAL AVE
1    2196808      6038-6040 S PULASKI RD
2    1574001        1060 W ADDISON AVE
3    2320340        2700-2712 W 51ST ST
4    1997579  6344-6348 W IRVING PARK RD
..       ...                       ...
579        0           13015 S ELLIS ST
580  2008948        8750 W BRYN MAWR AVE
581  2004238             1800 E 79TH ST
582  1717420             1960 W 13 TH ST
583  2009231      11919 S AVENUE O AVE FL
```

Figure 14: records where one license with different address

Here, we can confirm that all of the data for major use case (DBA name, License, latitude, longitude, risk, result) is populated with no missing data. The all latitude and longitude values are within the correct range.



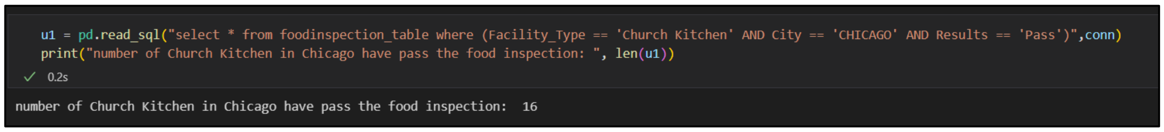
61	81030	1461 E HYDE PARK BLVD
578	81030	1453 E HYDE PARK BLVD
154	1045301	1400 W RANDOLPH ST
381	1045301	160 N LOOMIS ST
386	1093945	2321 W LAWRENCE AVE
525	1093945	2321 W LAWRENSE AVE
28	1142125	5700 S CICERO AVE
314	1142125	5800 S CICERO AVE

There are several records with license number paired with different address, most of them has typo error in the address column which would be dropped or ignored.

To illustrate data quality issues and demonstrate data cleaning improvements, the original data set was loaded into a SQL table to show the valuable information handled before and after data quality improvements were made. below are few use case which we ran in sqlite.ipynb notebook:

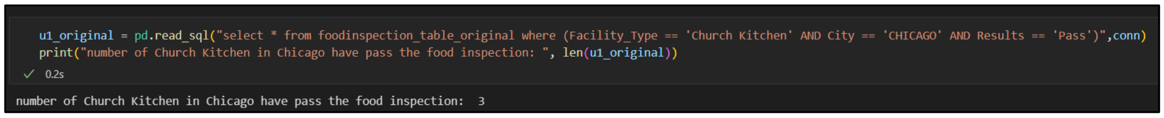
U1: Number of Church Kitchen in Chicago have pass the food inspection?

Cleaned:



```
u1 = pd.read_sql("select * from foodinspection_table where (Facility_Type == 'Church Kitchen' AND City == 'CHICAGO' AND Results == 'Pass')",conn)
print("number of Church Kitchen in Chicago have pass the food inspection: ", len(u1))
✓ 0.2s
number of Church Kitchen in Chicago have pass the food inspection:  16
```

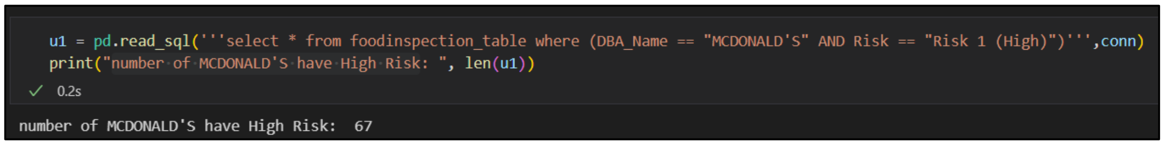
Original:



```
u1_original = pd.read_sql("select * from foodinspection_table_original where (Facility_Type == 'Church Kitchen' AND City == 'CHICAGO' AND Results == 'Pass')",conn)
print("number of Church Kitchen in Chicago have pass the food inspection: ", len(u1_original))
✓ 0.2s
number of Church Kitchen in Chicago have pass the food inspection:  3
```

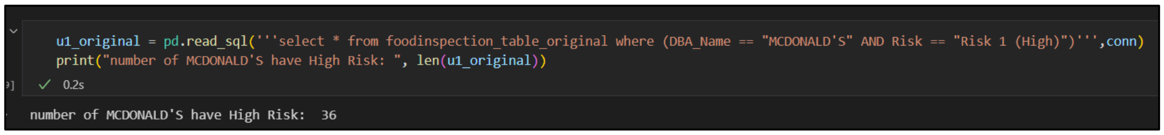
Number of MCDONALD'S have High Risk?

Cleaned:



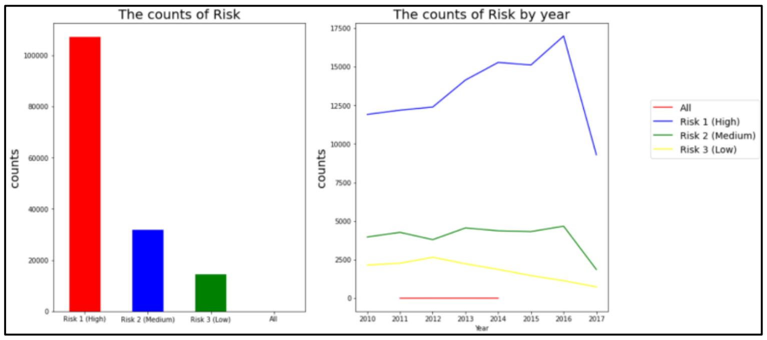
```
u1 = pd.read_sql('''select * from foodinspection_table where (DBA_Name == "MCDONALD'S" AND Risk == "Risk 1 (High)")''',conn)
print("number of MCDONALD'S have High Risk: ", len(u1))
✓ 0.2s
number of MCDONALD'S have High Risk:  67
```

Original:

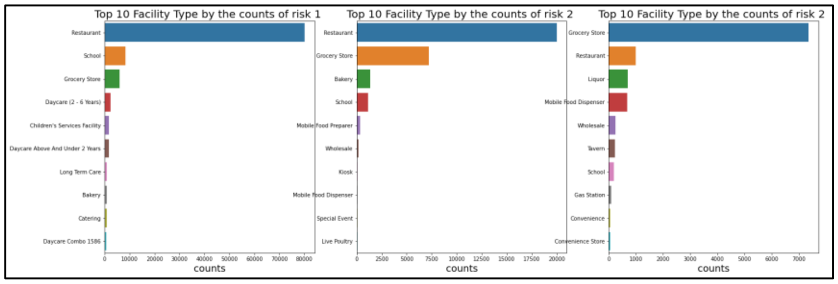


```
u1_original = pd.read_sql('''select * from foodinspection_table_original where (DBA_Name == "MCDONALD'S" AND Risk == "Risk 1 (High)")''',conn)
print("number of MCDONALD'S have High Risk: ", len(u1_original))
✓ 0.2s
number of MCDONALD'S have High Risk:  36
```

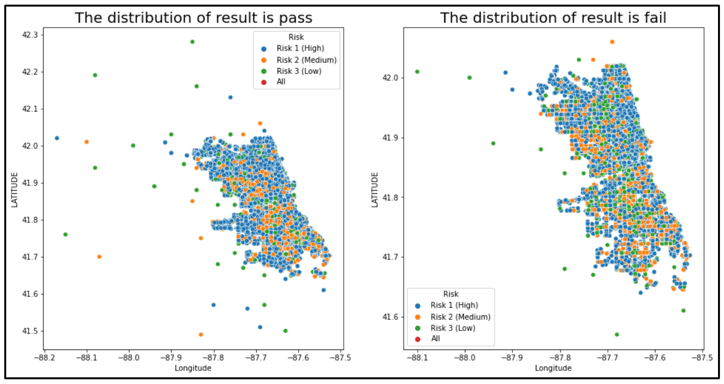
Exploratory Data Analysis



There are four types of risk: high, medium, low and All. Most (over 70%) audited spots had high risk. Second one is medium with around 20% participation and the low risk with around 10%. There are large number of food business get a rating of high and medium risk during 2015 and 2016.



restaurant has the most number of facility with rating of high and medium risk(over 50%), most of the Grocery Store has rating of medium(around 20%) and low risk(over 50%).


The distribution of result is pass
The distribution of result is fail
Risk
Risk 1 (High)
Risk 2 (Medium)
Risk 3 (Low)
All
LATITUDE
Longitude

I divide result on only two types: pass (normal and with conditions) and no pass (fail and other). Over 2/3 facilities pass the control, 1/3 fail or don't have control etc. As we can see in map, there is no spatial difference between facilities whose pass control and no pass. The center with the highest density of spots seems to be relatively safer than in other places.

Conclusions & Summary

This project was a great opportunity to understand all the nuances regarding data cleaning. Although some basic analysis could be done as is, more in depth analysis required cleaning from consistency and constraints among each column. The removal or marking of outliers or other significant data points was necessary too. I found that the best approach was to use a combination of tools, as used OpenRefine, Python, and SQL.

OpenRefine provided a baseline of cleaning that was necessary. The approach of cleaning column by column laid the foundation for our further processing with SQL. Although I quickly realized that OpenRefine alone was not enough and I had to go through a few more steps using Python.

I found that Python was helpful in identifying specific outliers within columns or marking specific characteristics of entries such as one business License number has several different address. Then with the use of queries, SQL was then ideal to identify any constraints within the data or make sure each entry was distinct.

In conclusion, through this project, I have learned that the best way to approach cleaning is through the use of several tools. I was able to select a dataset, create a target use case, and outline all the required steps to create the most accurate, clean and concise dataset that would allow us to answer our case use with the best customer experience.

Supplementary Materials

Workflow Model are stored in the OpenRefine and Python folder and can also be found directly in the GitHub repository.

The SQL queries for checking integrity constraints and exploratory data analysis for this project can be found in the IPython Notebook (sqlite.ipynb and Exploratory data analysis.ipynb), which can be found in the Sqlite SQL and EDA folder or directly in the GitHub repository.

The original data set is provided in a Box folder identified in the DataLinks.txt or directly through this url.