CS 513 - Theory & Practice of data cleaning
Summer 2023

Team Project: Phase 1 Report
(Team - 30)

Prof: Bertram Ludaescher

Project Members:
Xiaojing Xu (xx33@illinois.edu)
Xinyu Fu (xinyufu2@illinois.edu)
Jake Xu (cxu54@illinois.edu)

July,9,2023

1. Dataset Chosen

We have chosen the provided Chicago Food Inspection dataset, which is about restaurant inspections conducted by the Food Protection Division of the Chicago Department of Public Health (CDPH).

2. Description of Dataset

The chosen dataset constitutes a comprehensive inquiry into the inspection of different types of facilities in Chicago, aiming to ascertain specific details such as address, license number, and incurred violations.

The dataset includes information related to inspections of various food establishments in Chicago, such as restaurants, grocery stores, schools, bakeries, hospitals, and temporary food service events. The data covers a specific period from January 4, 2010, to August 28, 2017. The available data provides details about each inspection, including the inspection date, inspection results, violations noted during the inspection, business names, license numbers, risk levels, and geographical information in the form of latitude and longitude coordinates. This information allows for analysis and monitoring of food safety practices, compliance with regulations, and identification of any potential health risks or violations observed during the inspections. These records indicate whether the City's food sanitation regulations were faithfully implemented and provide valuable insights for more effectively imposing food safety requirements in Chicago, with distinctive strategies across different areas.

Below are the column descriptions in our dataset:

Column Name	Descriptions
Inspection ID	A unique identifier for each food inspection conducted.
DBA Name	The "doing business as" name of the food establishment.
AKA Name	Also known as the "public name" of the food establishment, and is different from DBA Name in some cases.
License Number	The license number assigned to the food establishment, and is expected to be unique for each food establishment.
Facility Type	The type of facility, such as a restaurant, grocery store, school, etc.
Risk	The categorization for each food establishment's risk of adversely affecting public health, ranging from risk 1-3. The frequency of inspection is tied to this risk, with risk 1 establishments inspected most frequently and risk 3 least frequently.
Address	The street address of the food establishment.
City	The city of the food establishment's location, and should be Chicago.

State	The state of the food establishment's location, and should be IL.
ZIP	The ZIP code of the food establishment's location.
Inspection Date	The date on which the inspection took place.
Inspection Type	The type of inspection conducted, such as a routine inspection, re-inspection, complaint inspection, etc.
Results	The outcome of the inspection, indicating whether the establishment passed, failed, or passed w/ condition
Violations	Details of any violations found during the inspection, including the specific code of regulation violated and comments about the violations.
Latitude	The geographical coordinates of the food establishment's location.
Longitude	The geographical coordinates of the food establishment's location.
Location	A combined field representing the latitude and longitude.

```
Inspection ID        int64
DBA Name            object
AKA Name            object
License #          float64
Facility Type       object
Risk                object
Address             object
City                object
State               object
Zip                float64
Inspection Date     object
Inspection Type     object
Results             object
Violations          object
Latitude           float64
Longitude          float64
Location            object
dtype: object
```

Figure 1. Schema description from original dataset

The dataset is a single table consisting above 17 columns and 153810 rows. To manage all the records in a more readable and efficient way, we break down the data into 3 tables as Facilities, Inspections, and Addresses. We grouped related information together and built up the relationship as the following ER model diagram.

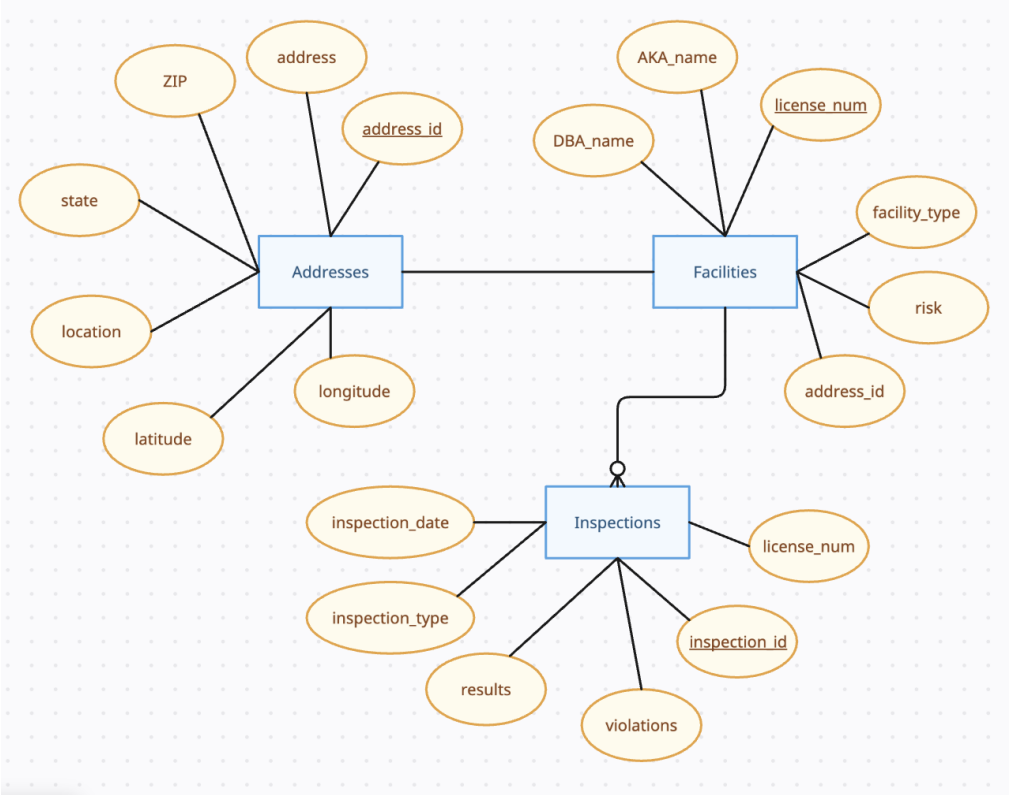

Figure 2. An Entity-Relationship model that describes the interrelated structure of the Chicago Food Inspections' columns

The data structure we redesigned is based on the ER approach instead of the non-value approach the dataset used before. It will reduce the size of each table, and when doing the basic data wrangling it will be more efficient.

Our relational schema:
Facilities(license #, DBA_name, AKA_name, facility_type, risk, address_id)
Addresses(address.id, address, ZIP, state, street, latitude, longitude, location)
Inspections(inspection.id, inspection_type, inspection_date, license #, results, violations)

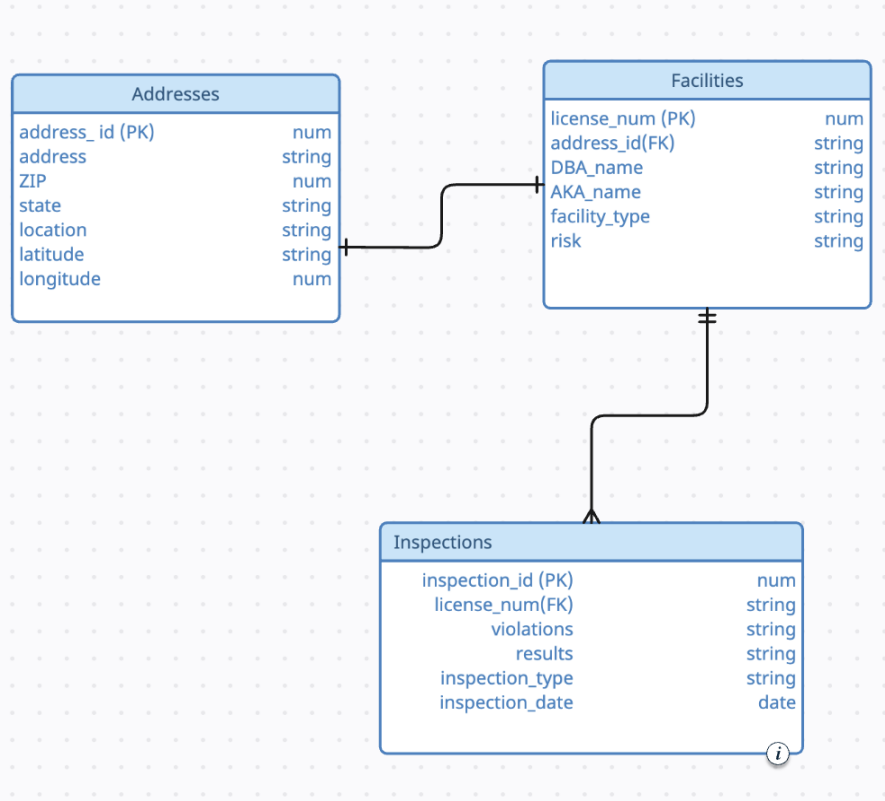

Figure 3. An Entity Relation Model with data type that conceptually depicts the Chicago Food Inspections dataset as a whole

When we got the general information about the database, we could define the type of each field on basic analysis. For the primary key of each table, the data type will be an id number, and it uniquely determines all other attributes for each table (see above).
Most of the fields own the data type as string. In the table Addresses, the ZIP code, latitude, and longitude columns are number type, and so as to the table Facilities. In the table Inspections, the inspection_date has a date data type, which possibly can be converted to ISO format during phase II.

3. Use Cases

We used OpenRefine and Tableau as our tools to help us analyze data from the original table, based on the information we got to design the different use cases.

a. “Zero cleaning” use case U0 - Number of Inspections Conducted per Year

It is unnecessary to perform data cleaning on U0 because:

a. The dataset includes dates of inspection and there are no missing rows for this column. We can analyze the number of inspections conducted each year without worrying about the incompleteness of such numbers.
b. Furthermore, it is unlikely that the number of inspections per year got mismatched or misrecorded since the inspection was conducted annually and the inspection_id uniquely determines each related attribute.

Thus, we hold that it is reasonable to perform analysis on the current dataset for U0 without data cleaning.

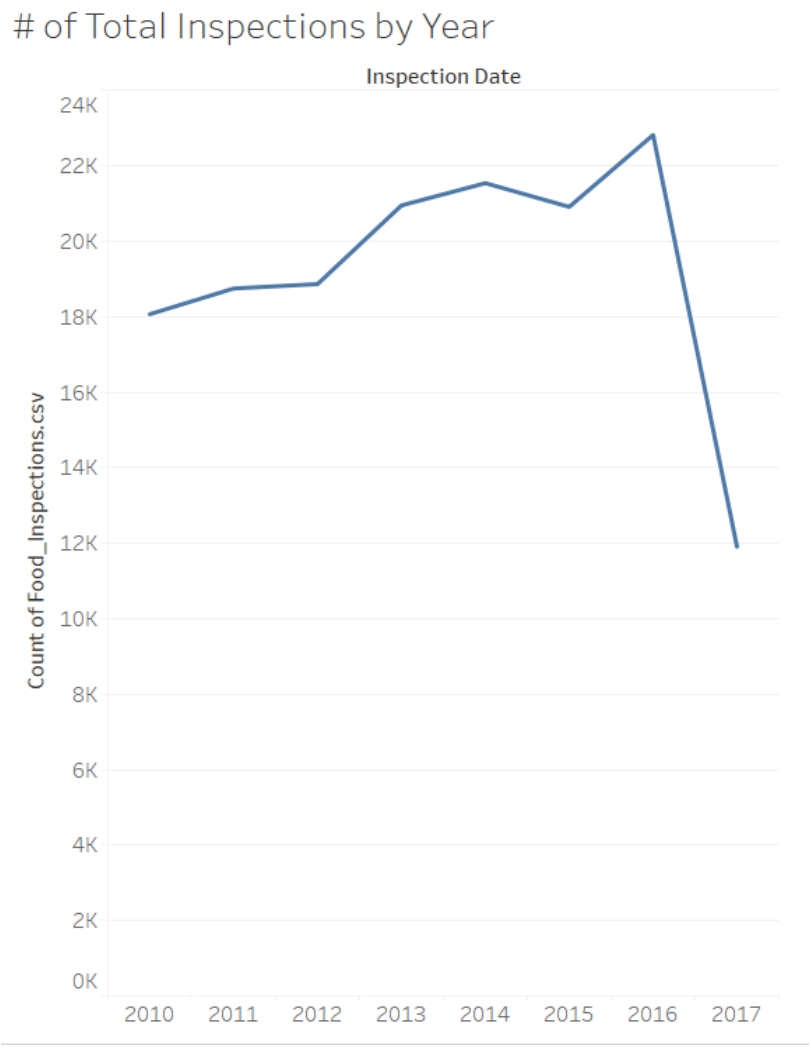

Figure 4. A simple line chart generated by Tableau, with x-axis being the year and y-axis being the number of total inspections associated with each year

The annual number of inspections ranges from 12K-22K, a relatively normal scope for a big city like Chicago, and there is no outlier. On top of this, there is clearly some pattern deserving a further investigation, therefore data cleaning won't be necessary for this use case.

b. Target (main) use cases U1a-c

U1a - Rank the passed/passed with conditions inspection numbers based on different ZIP codes.

It is a general way to evaluate the frequency of passed inspections and analyze their distribution on different physical areas. We could select the records with 'Result' value equal to 'Pass' and

‘Pass w/ Conditions’ and sum them by each ZIP column, then sort the results by descending order tp rank. The necessity of cleaning data is as following:

a. For column ‘ZIP’ , we need to estimate the proportion of null values and remove the null if possible.
b. It is necessary to verify if each facility matches a unique ZIP code. This is the way to check the validity of of ZIP
c. We need to check the value in the ‘Results’ column for each record, if its value matches the situation of ‘Violations’, which is also a necessary check for the validity of Results.

U1b - Find out the proportions of Risk Category for Each Facility Type.

We want to examine what is the proportion of each risk category in different types of facilities, such as restaurants, grocery stores, or bakeries. The necessity of cleaning data is as following:

a. Facility type values need to be standardized. For example, there are multiple expressions of “convenience store” in the current dataset, such as “CONVENIENCE STORE” and “Convenience Store.” A meaningful representation of this facility type requires data cleaning techniques, such as clustering in OpenRefine. Otherwise, they would account for 2 different facility types and blur the true proportion of risk level under “convenience store”.
b. For those facility type values that are rather confusing to categorize, we need to cluster them under reasonable assumptions. Descriptions for these assumptions need to be laid out in the first place and implemented throughout the cleaning process.
c. Risk category values need to be standardized. Some establishment’s risk value is null; others have results merely showing an undefined “All”, which possibly means that these establishments harbor risks at all 3 levels or the completely opposite. Both of the values require us to either permanently remove these establishments due to their invalidity or to temporarily adjust the size of each facility type so that we can limit these anomalies which easily throw off the total proportions for this use case, without damaging the data completeness. This ignoring functionality would be easily implemented through data cleaning, such as the INSERT IGNORE statement in SQL.

U1c - Display Yearly Trends of Inspection Results of McDonald's.

We want to examine the changing trends of McDonald's inspection results, specifically the number of passed inspections over the years. To analyze the trend of passed inspections, it is necessary to discover how many valid results McDonald's got in the past years. The necessity of cleaning data is as following:

a. Values in DBA name and AKA name need to be standardized. There are consecutive spaces between the words in some of the values. And like the use case above, we need to combine similar values into one value.
b. Data type of ‘Inspection_date’ needs to adjust to ‘date’. Since we need to categorize results by different year, it is necessary to add the year column based on the ‘Inspection_date’ column.
c. We need to check the value in the ‘Results’ column for each record, if its value matches the situation of ‘Violations’, which is also a necessary check for the validity of Results.

d. Some McDonald's inspections were categorized as "Pass w/ conditions" without specifying any violations under the "Violations" column. These results could be misleading as to whether it indeed passed without any violations or its violations got corrected during the inspection, just like the data source originally described the column "Pass w/ conditions." We need data cleaning to either combine these results with the correct passed inspections, or to simply eliminate them to capture the true passing trends under our assumptions.

For further investigation and example DQ problems identification, see Section 4.

c. "Never enough" use case U2 - A Complete Understanding, with an Extrapolation, of Compliance History

Based on the Inspection type, we know that there are re-inspections for food establishments, which means they have the opportunity to address violations and improve their compliance following an inspection. A use case is to analyze and quantify an establishment's overall adherence to maintaining food safety standards. How does one establishment's compliance history compare to other similar establishments? However, current data only provides a snapshot of compliance between 2010 to 2017 and does not offer a comprehensive long-term compliance history for each establishment. It does not reflect any improvements made by the establishment before 2010 or after 2017 (Figure 5). If we want to gain a more complete understanding of an establishment's compliance history and ongoing commitment to food safety, we need a more extensive dataset covering a longer period of time, which can not be obtained by cleaning current data.

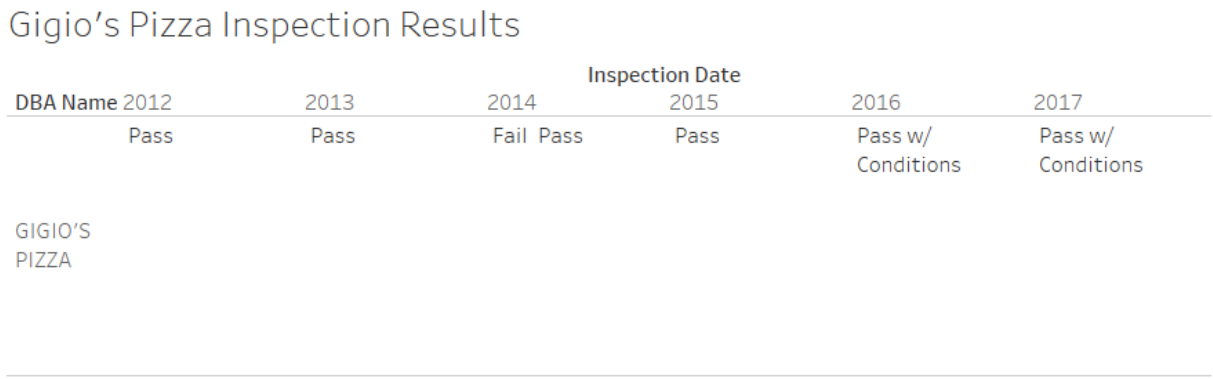

Figure 5. A single restaurant's inspection results throughout the years of this dataset

It is unnecessary to do data cleaning because:

a. We can see that there's really no pattern of pass/fail, with or without the influence of wrangled data since this is a qualitative feature. It would be hard to extrapolate the history outside of the year 2010-2017.
b. In addition, even for the result "Pass", there are 2 types: Pass or Pass w/ Conditions. There are no columns of this dataset to indicate what these conditions are, and in reality these conditions probably vary for each restaurant per inspection. Thus, it would be hard

to analyze each food establishment's compliance history and improvements without knowing the details of such conditions.

Data cleaning cannot deal with these qualitative inquiries especially with missing textual references, therefore we can never perform meaningful analysis on U2 even after data cleaning.

4. Data Quality Problems

The Chicago Food Inspection dataset suffers from many common data quality issues, including missing data, inconsistent formatting, ambiguous categorization, and redundancy.
For some attributes like DBA Name and Facility Type, the format and spelling of the same string value are inconsistent. Some attributes are redundant.
The attributes of State are redundant because we know the collected data are specifically from Chicago which is located in state IL. Listed below are data quality problems associated with our main use cases, highlighting the importance of data cleaning to support these use cases.

4.1 Data quality problem: Missing data

We could see from Figure 6 that for columns such as inspection id, DBA name, address, inspection date and results, there are no missing values. This means the dataset is comparably informative for the next steps.
But there are some columns containing null values less than 1%, which could be considered to be removed or handled in the future.

	Missing Value number	Missing Value Percentage(%)
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	0	0
City	159	0.1
State	8	0.01
Zip	98	0.06
Inspection Date	0	0
Inspection Type	1	0
Results	0	0
Violations	30798	20.02
Latitude	544	0.35
Longitude	544	0.35
Location	544	0.35

Figure 6. Data quantity analysis of showing missing value for each attribute

Application on Main Use case 1:
While the column "Inspection ID" is the primary key for table Inspection, we checked if records are unique in each record. We got a true return which means "Inspection ID" is neither null nor duplicated and it is capable of meeting the requirement of a primary key.

```
+----+---------------+--------------------------------+
|    | Column        | Unique Values in each record   |
|----+---------------+--------------------------------|
|  0 | inspection_id | Yes                            |
+----+---------------+--------------------------------+
```

Figure 7. Constraints check on Primary key for Inspection table

However, there are 15 records missing ‘License #’ attribute, which is considered as a primary key in table Facility, so these records need to be removed.
In use case U1-a, there are 98 records missing ZIP value which would affect the performance of the use case.
In use case U1-b, we need to observe the null data in the risk column, and there are 66 missing values. We also found out there is 1 record missing value of Inspection Type.
After discussion, we considered to remove the rows with those null values in License #, Risk, ZIP, Inspection Type. For records with null value in state, it could be filled as ‘IL’.

4.2 Data quality problem: Unstandardized data

The issue is we found inconsistent or misspelled facility types, resulting in multiple variations of the same facility type. We can use OpenRefine to merge similar values in a cluster.

Application on Main Use case 1:

In use case U1-b, The original data has 423 choices for the facility type, and we aim to reduce the number of choices after data cleaning, making it easier to interpret and visualize the different facility types effectively.

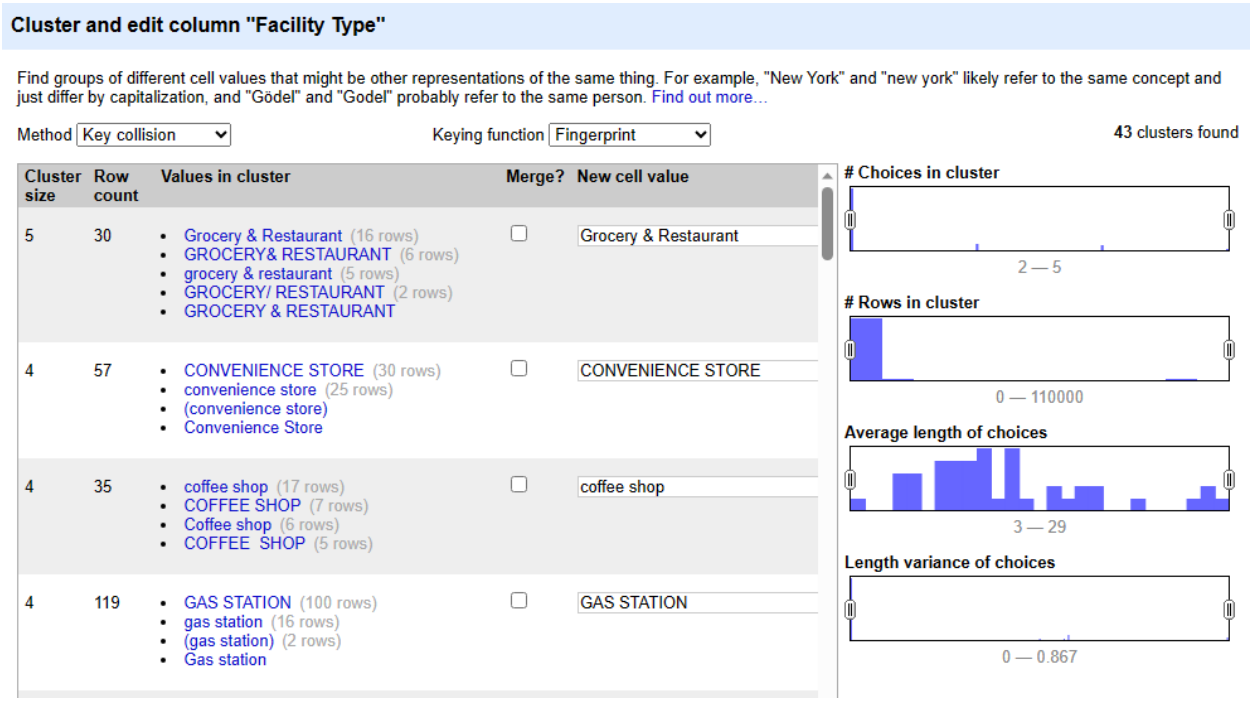

Figure 8. Clusters of the Facility Type column, showing inconsistency of naming conventions for the same facility type (i.e., capitalization, brackets, symbols etc.)

In use case U1-c, in the cluster of McDonald's, there are variations in capitalization and spacing between the words. Comparing the clustering results from different keying functions can provide valuable insights and enable a comprehensive analysis. The n-Gram fingerprint keying function yields more counts of the major clusters than the fingerprint keying function. We can combine the clusters obtained from different keying functions into a unified set of clusters. By merging clusters step by step, we will be able to reduce the variations and standardize the DBA names, resulting in a more accurate analysis of the popular food chains.

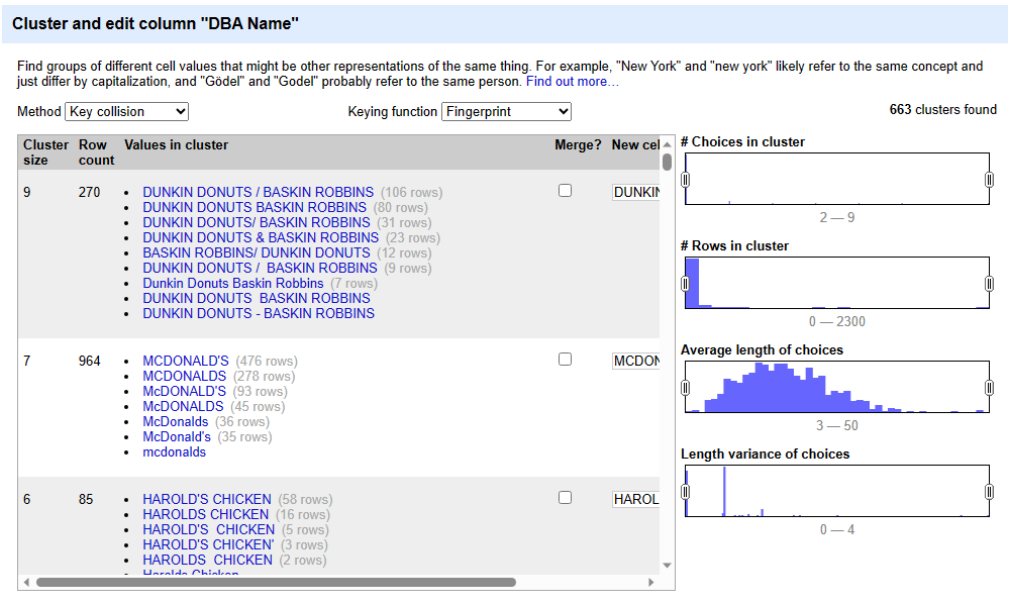

Figure 9. Clusters of the DBA Name column using the Key Collision method and Fingerprint Keying function

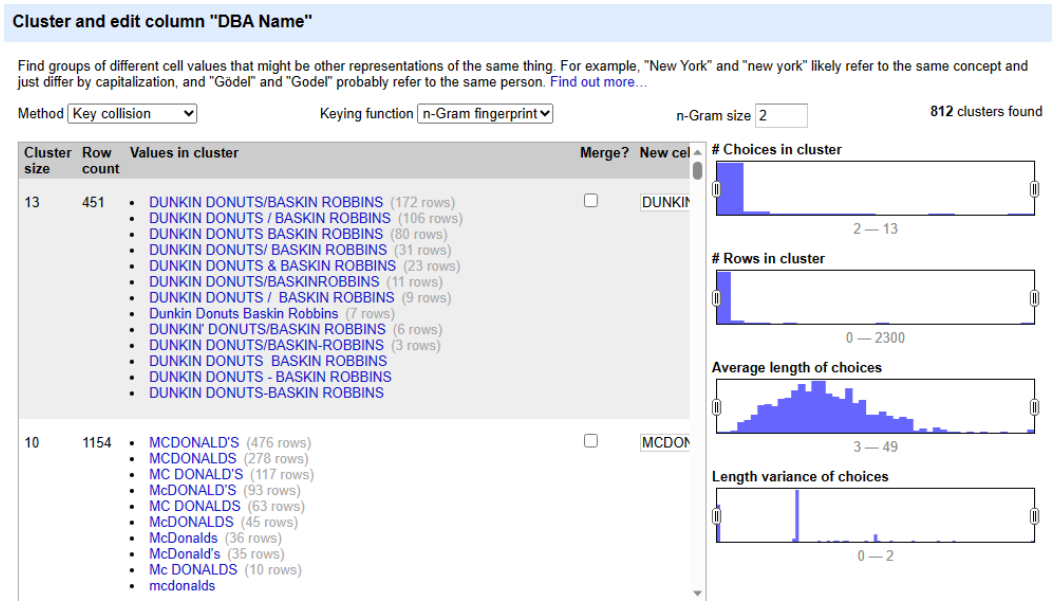

Figure 10. Clusters of the DBA Name column using the Key Collision method and n-Gram fingerprint keying function

It's important to have consistent naming conventions to identify restaurants of the same food chain. The DBA Name column may be more suitable for our analysis than the AKA Name column because the DBA Name represents the official business name.
The DBA Name column is particularly messy and contains a large number of variations of the same restaurant name. By investigating clustering results based on different keying functions using OpenRefine (Figure 9 and Figure 10), we identified many data quality issues of these text strings.

4.3 Data quality problem: Data Inconsistency
Inconsistencies occur when there are contradictory or conflicting data within the dataset.
It is a common data quality problem that arises when there are contradictory or conflicting data within a dataset. We could find it occurs when different parts of the dataset provide conflicting information.

Application on Main Use case 1:
In use case u1a and u1c, we need to verify the values in 'Results' column, and when we looked into the columns of 'Results' and 'Violations', we found an inconsistency indicating that even if there is no violation, some facilities got results of 'Pass with conditions'.
If the facility did not own any violations, the result of its inspection should be 'Pass' instead of 'Pass/w conditions', which is defined in the attribute dictionary. For the 113 records we found inconsistent, they need to be changed to 'Pass'.

	Condition	Record Count
0	'violations' is not null and 'result' is 'Pass with conditions'	14417
1	'violations' is null and 'result' is 'Pass with conditions'	113

Figure 11. Got the confliction comparing values in 'Results' column and 'Violations' column

5. Initial Plan for Phase-II
S1: Review and update your use case description and dataset description
We described the dataset in detail and developed the Entity-Relationship model that depicts the dataset as shown in Section 2. We have identified various use cases and showed figures to support our use cases as shown in Section 3. We'll keep updating these during phase-II. (Completed and will be updated by JX, XX and XF)

S2: Profile D to identify DQ problems
As shown in Section 4, we identified DQ problems by combining column values to see if analysis is feasible, and generating preliminary data visualization of different use cases to see whether the result is meaningful already, or has serious noises that impede further inquiries. We used OpenRefine to address the data quality concerns pertaining to the columns relevant to our main use cases, such as missing values and inconsistencies. We also proposed initial data cleaning strategies for these concerning columns. (Completed by XF, XX, and JX)

S3: Perform DC "proper":

- To initiate the data cleaning process, we will employ OpenRefine as our primary tool. OpenRefine will help us tackle initial cleaning tasks such as addressing inconsistent symbols, syntax errors, and typos within the dataset. We will leverage different clustering techniques and RegEx syntax available in OpenRefine to merge similar values into clusters to reduce the number of options in columns such as the DBA Name column. We will also use OpenRefine Json operation history to generate a YesWorkFlow diagram. (Assigned to XF)
- We will further enhance the data cleaning process using Python. Python will enable us to address specific data cleaning tasks that may not be achievable with OpenRefine. We plan to populate missing Latitude and Longitude values by leveraging Zip or Address information in a .py script. Furthermore, Python will be utilized to estimate the risk level of each facility by examining the annual inspection numbers associated with them and the spread of pass/fail results throughout the years. Python is also more suitable for creating new columns, such as the foreign key address.id we created for the Addresses table. Its advanced profiling capabilities (usually a divide-and-conquer approach) and its flexible package operations (Pandas, numpy, etc.) will provide unique advantages over other tools. For instance, we may desire to clean and organize the proportion of risk categories across different regions by recursively tracing back notable spatial patterns year by year. This can be complicated to do in SQL or OpenRefine. (Assigned to JX and XX)
- We will create an SQLite database and run SQL queries to both check integrity constraint violations to contrast and compare the dirty/cleaned dataset, and demonstrate the results of our main use case. These queries are ones of the best to verify that our dataset has been significantly improved and ready to generate meaningful results for analysis. One such query example would be the standard SQL SELECT FROM WHERE AND ORDER BY commands to generate the results for McDonald's passed inspections over the years before/after data cleaning (our main use case U1c). (Assigned to JX and XX)

S4: Data quality checking

- We will develop test examples / demos to show the summary statistics of the data before and after cleaning. (Assigned to JX, XX, and XF)
- On a given piece of use case, we perform queries on both the dirty and clean dataset to show how effective our cleaning methods will be regarding our main use case U1. (Assigned to XX)

S5: Document and quantify change

- We will document the number of changes in columns and cells, and any IC violations detected before and after data cleaning. We will generate workflow diagrams to record changes we made using OpenRefine, Python, and SQL. (Assigned to XF and JX)
- We will use Jupyter Notebook or R Markdown to analyze and visualize the data before and after cleaning. (Assigned to XF and JX)