CS513: Theory & Practice of Data Cleaning

**Final Project Phase 1 Report**

**Team 59: Data Mavericks**

Avinash Baldeo, Ashley George, Sotheara Chea

[abaldeo2@illinois.edu,](mailto:abaldeo2@illinois.edu) [ageorge8@illinois.edu](mailto:ageorge8@illinois.edu), [chea3@illinois.edu](mailto:chea3@illinois.edu)

**Table of Contents**

[**1. Dataset Overview** 3](#_Toc138967167)

[**2. Dataset Description** 3](#_Toc138967168)

[2.1 Full Data Narrative 3](#_Toc138967169)

[2.2 Database Diagram & Schema 5](#_Toc138967170)

[**3. Use cases** 8](#_Toc138967171)

[3.1 U0: Zero Cleaning Use Case 8](#_Toc138967172)

[3.2 U1: Main Use Case 8](#_Toc138967173)

[3.3 U2: Never Enough Use Case 9](#_Toc138967174)

[**4. Data Quality Problems** 9](#_Toc138967175)

[4.1 List obvious data quality problems 9](#_Toc138967176)

[4.2 Why data cleaning is necessary for main use case U1 11](#_Toc138967177)

[**5. Phase-II Initial Plan** 11](#_Toc138967178)

[5.1 Data Cleaning Plan 11](#_Toc138967179)

[5.2 Who in the team will be responsible for which steps and Timeline 12](#_Toc138967180)

# **1. Dataset Overview**

For this project our group is using the Chicago food inspection dataset which is originally released on Kaggle by the City of Chicago: <https://www.kaggle.com/datasets/chicago/chicago-food-inspections>

The Chicago Department of Public Health's dataset contains information from restaurant inspections since January 1, 2010. As per the description given, the inspections are standardized and conducted by the Food Protection Program staff. The results get input into a database, reviewed, and approved by a Licensed Environmental Health Practitioner. The dataset provided includes a subset of the data elements extracted from the database. A disclaimer is given that the dataset on food inspections may contain duplicates, making it a suitable choice for data cleaning.

# **2. Dataset Description**

## 2.1 Full Data Narrative

Food establishments undergo annual and complaint-based inspections for compliance with City ordinances. The food inspections ensure food safety in licensed establishments such as restaurants, grocery stores, and bakeries.

The Chicago Department of Public Health (CDPH) conducts these science-based inspections of food establishments, promoting food safety, sanitation, and preventing food-borne illnesses. The inspections cover food handling, temperatures, hygiene, facility maintenance, and pest control.

While Inspections for sanitation are done by the Health Department, inspections are also conducted by the Buildings Department for structural safety and Fire Department for fire exits. The City's Dumpster Task Force also checks for trash disposal compliance with sanitation regulations.

The dataset provided is maintained using Socrata's API and Kaggle's API (Application Programming Interfaces), and the data source is the City of Chicago Data Portal <https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5>

Uncompressed, the dataset size is 176 MB. In total there are 17 columns and 153,810 records with inspection dates ranging from 01/04/2010 to 08/28/2017.

The table below gives a brief description of each column available in the food\_inspection.csv file.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Column Type** | **Description** |
| Inspection ID | integer | A unique number identifying the inspection occurrence |
| DBA Name | string | Stands for “Doing Business”, it is the legal name of the registered food establishment. |
| AKA Name | string | Stands for “Also Known As”, it is the publicly known name of the food establishment. |
| License # | integer | A unique license number assigned to the establishment for legal purposes by Department of Business Affairs and Consumer Protection |
| Facility Type | string | Describes the type/category of the establishment such as a bakery, restaurant, grocery store, etc. |
| Risk | string | The establishments’ risk level of adversely affecting public health (1 being the highest and 3 the lowest risk). Higher risk is inspected more frequently. |
| Address | string | The full street address of the establishment. |
| City | string | The city where the establishment is located. |
| Zip | integer | The zip code associated with the address. |
| Inspection Date | string | The date when the food inspection occurred. |
| Inspection Type | string | The type of inspection performed such canvass consultation, complaint, etc. |
| Results | string | Indicates whether the inspection passed, passed with conditions, or failed. |
| Violations | string | List of distinct health violations (46 distinct types) with descriptions, found during the inspection |
| Latitude | float | The GPS latitude coordinate of the establishment location |
| Longitude | float | The GPS longitude coordinate of the establishment location |
| Location | string | The GPS point coordinate (latitude, longitude) of the establishment location |

Table 2.1 – Food Inspection Dataset Description

## 2.2 Database Diagram & Schema

The following database diagram(s) represent a better designed & normalized view of the dataset with foreign key constraints enforcing the referential integrity and maintaining the relationships of the original data.

The database consists of the following 5 tables:

FoodEstablishment stores the key establishment information such as license, business name and type of facility.

EstablishmentLocation stores the address & location information of the food establishments.

FoodInspection holds food inspection events including type, date, and result.

InspectionViolation maps the inspection events to the list of violations (if multiple) received along with the health inspectors' comment about each violation.

ViolationCode is the master table that contains the unique list of health inspection violation codes and the descriptions.

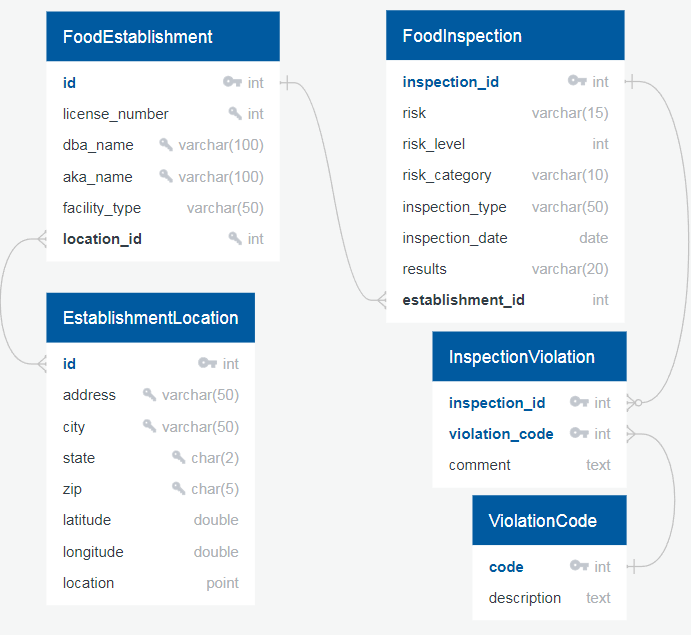


Figure 2A – Entity-Relationship (ER) Diagram of the normalized data

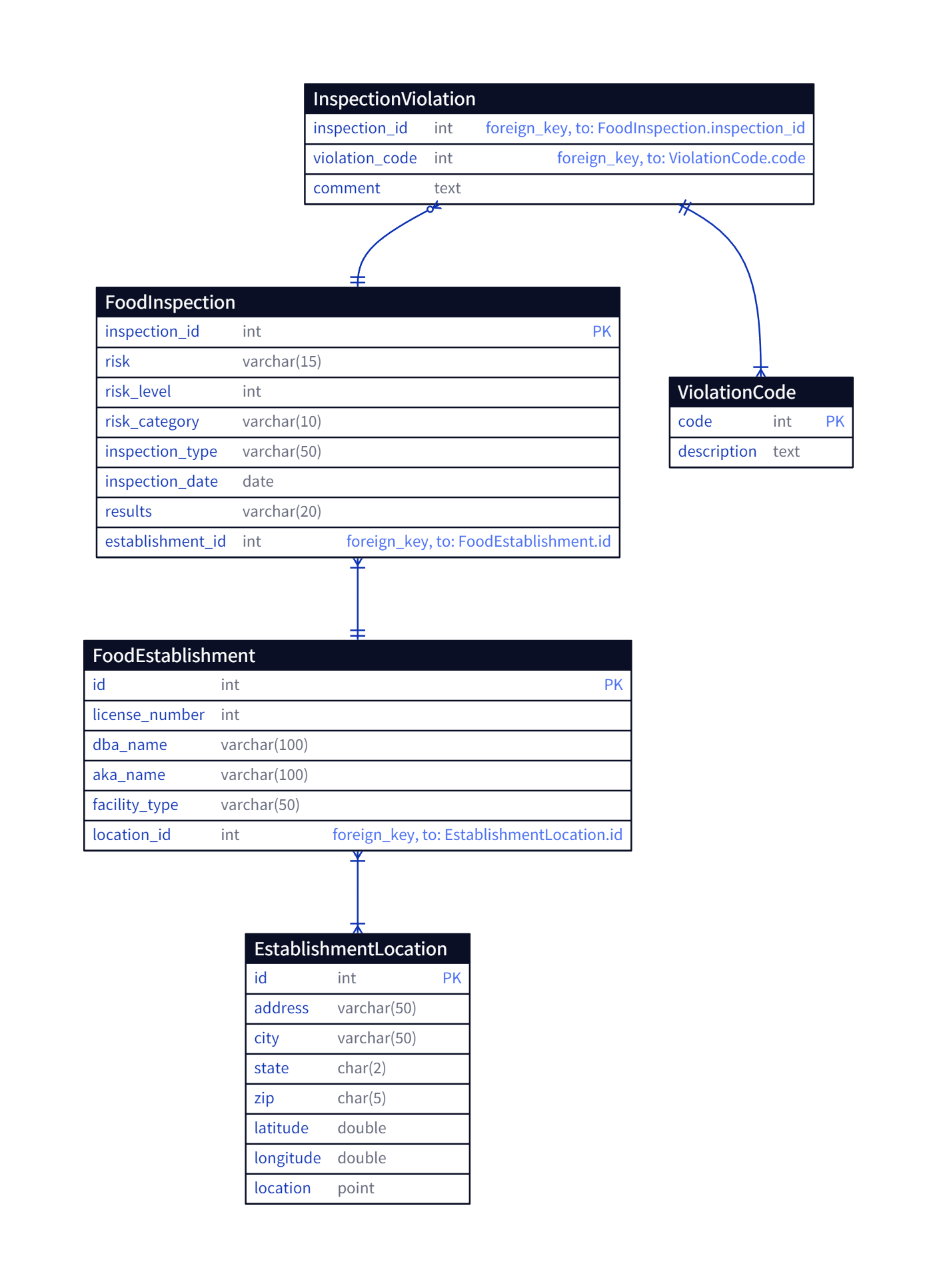


Figure 2B – Entity-Relationship (ER) Diagram with foreign keys

**CREATE TABLE EstablishmentLocation (**  
 **id INTEGER PRIMARY KEY AUTOINCREMENT,**  
 **address VARCHAR(50) NOT NULL,**  
 **city VARCHAR(50),**  
 **state CHAR(2),**  
 **zip CHAR(5),**  
 **latitude DOUBLE PRECISION,**  
 **longitude DOUBLE PRECISION,**  
 **location POINT**  
 **);**

Notes on EstablishmentLocation:

* + id is the autoincremented primary key.
  + address is always present in the dataset given.
  + A unique key constraint will be created on address, city, state, and zip.
  + Location (latitude, long) is redundant but kept for geospatial queries.
  + There can be multiple food establishments at the same location.

**CREATE TABLE FoodEstablishment (**  
 **id INTEGER PRIMARY KEY AUTOINCREMENT,**  
 **license\_number INTEGER,**  
 **dba\_name VARCHAR(100) NOT NULL,**  
 **aka\_name VARCHAR(100),**  
 **facility\_type VARCHAR(50),**  
 **location\_id INTEGER**  
 **);**

Notes on FoodEstablishment:

* + location\_id is foreign key to EstablishmentLocation table.
  + id is the autoincremented primary key.
  + dba\_name is always present in the dataset given.
  + A unique key constraint will be created on license\_number, dba\_name, aka\_name, and location\_Id after data\_cleaning.
  + Food establishments such as franchises can have multiple different locations.

**CREATE TABLE FoodInspection (**  
 **inspection\_id INTEGER PRIMARY KEY,**  
 **risk VARCHAR(15),**  
 **risk\_level INTEGER,**  
 **risk\_category VARCHAR(10),**  
 **inspection\_type VARCHAR(50),**  
 **inspection\_date DATE NOT NULL,**  
 **results VARCHAR(20) NOT NULL,**  
 **establishment\_id INTEGER**  
 **);**

Notes on FoodInspection:

* + inspection\_id is NOT autogenerated, it is the primary key that identifies the food inspection event and comes directly from Inspection ID column.
  + risk\_level is the numeric value assigned to the establishment's health risk (1, 2, 3). It is parsed from the Risk column.
  + risk\_category is the nominal value of risk assigned to establishments health risk (low, medium, high). It is parsed from the original dataset Risk column.
  + inpsection\_date and results are always present in the dataset given.

**CREATE TABLE ViolationCode (**  
 **code INTEGER PRIMARY KEY,**  
 **description TEXT NOT NULL**  
 **);**

Notes on ViolationCode:

* + code is NOT autogenerated, it is the primary key that identifies the food violation. It is parsed from the Violation column in the dataset.
  + description is parsed from the violation code. Each code description is present in the given dataset

**CREATE TABLE InspectionViolation (**  
 **inspection\_id INTEGER,**  
 **violation\_code INTEGER,**  
 **comment TEXT,**  
 **PRIMARY KEY (inspection\_id,violation\_code),**  
 **FOREIGN KEY (inspection\_id) REFERENCES FoodInspection(inspection\_id),**  
 **FOREIGN KEY (violation\_code) REFERENCES ViolationCode(code)**  
 **);**

The following database indices are created to help with analytical queries and joins between the related tables.

**CREATE INDEX idx\_location ON EstablishmentLocation (location);**  
  
**CREATE INDEX idx\_foodestablishment\_\_location\_id ON FoodEstablishment (location\_id);**  
  
**CREATE INDEX idx\_facility\_type ON FoodEstablishment (facility\_type);**  
  
**CREATE INDEX idx\_foodinspection\_\_establishment\_id ON FoodInspection (establishment\_id);**  
  
**CREATE INDEX idx\_risk\_category ON FoodInspection (risk\_category);**  
  
**CREATE INDEX idx\_inspection\_date ON FoodInspection (inspection\_date);**  
  
**CREATE INDEX idx\_inspection\_type ON FoodInspection (inspection\_type);**  
  
**CREATE INDEX idx\_results ON FoodInspection (results);**

Note, these unique index constraints will be added to the schemas in Phase 2 after cleaning, otherwise it will prevent loading the current dataset into the database.  
  
**CREATE UNIQUE INDEX idx\_uniq\_location ON EstablishmentLocation (**  
 **address,city,state,zip**  
 **);**

**CREATE UNIQUE INDEX idx\_uniq\_establishment ON FoodEstablishment (**  
 **license\_number,dba\_name,aka\_name,location\_id**  
 **);**

# **3. Use cases**

## 3.1 U0: Zero Cleaning Use Case

Upon initial the data inspection, it is apparent that the dataset as-is can provide answers to questions about trends in inspection results over time. Specifically, we can check for the following:

* Visualize the count of the results of inspections over a given timeframe (by year/month) as Line/Bar Chart.
* Create a Bar chart showing the count of each inspection result type.
* Create a Bar chart showing the count of the number of inspections by date.

Each record in the dataset represents an inspection event, as denoted by the Inspection ID field, which is unique and has no missing values. The dataset also contains both the Inspection Date for each inspection and the Results of the inspections. Both fields contain no missing values, and the inspection date follows a consistent date format dd/mm/yyyy which can easily be converted from string to date without any data cleaning.

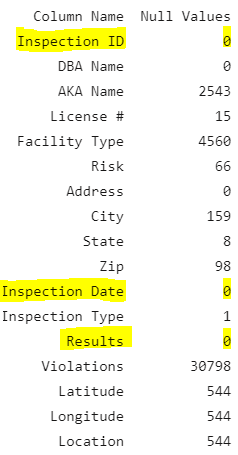


Figure 3.1a - U0 Columns

By using these fields to create these type visualizations, we should be able to see if there are there any seasonal patterns to inspection result outcomes (i.e., more failures during colder months vs. warmer months)

## 3.2 U1: Main Use Case

Consumers choose food establishments based on several factors. Food safety is one of, if not the most important factor for them. With this dataset on Chicago food inspections, for our main use we can build a visualization dashboard (using either Python or Tableau) to provide insights on the safety of Chicago food establishments based on the history of inspection violations, the frequency of each violation type, and the risk level for consumers. Specifically, we would hope to answer the following questions by performing this data cleaning and visualization:

* Is there a correlation between the type of facility and the risk level assessed from inspections?
* How do inspection results (pass, pass with conditions, fail, etc.) vary among different types of food establishment?
* What are the most common violation codes received based on type of inspection & type of facility?
* What is the distribution of inspection results by geographic locations (i.e., which areas have the highest number of failed inspections?)
* What is the distribution of risk severity by geographic locations?

The list of all the violations cited is currently included in the same column for each inspection. The description of each violation is duplicated as there are only 46 distinct violations. Also, comments providing the details of each violation are bundled together in the same column.

## 3.3 U2: Never Enough Use Case

Even with the data cleaning performed on this dataset, we still are unable to provide general food establishment recommendations to consumers as the data available is only a view of food safety, which is just one aspect in determining where to eat. The dataset currently does not incorporate customer reviews on food taste, indicate the type of cuisines available, or include indicators on the food prices. We would need to build a separate workflow to pull such information via external APIs such as Yelp, Foursquare, and Google and then we would have to combine it with this food inspection dataset to create a proper recommendation system.

# **4. Data Quality Problems**

## 4.1 List obvious data quality problems

1. Missing Values - there are values missing in some of the columns in the dataset such as Violations, Facility Type, AKA Name, and Longitude and Latitude. The most is the violations column with 30798 nulls.
   1. For the main use case, these inspections with null in the violation column may need to be filtered out if they truly represent that the inspection resulted in no violations.
   2. 4560 missing values in Facility Type column may or may not be an issue depending on the type of visualization being shown on the dashboard, but likely they will be filtered out.
   3. AKA Name has 2543 missing values. Since we are planning on using AKA Name as part of a composite key to identify establishments, we shall coalesce these with DBA Name to deal with the nulls in this column.
   4. Latitude and Longitude are missing for 548 address records. Depending on if these are mapped to valid businesses, we can try to backfill them (using address and zip) to increase the coverage for our geographic visualizations, otherwise these records may need to be dropped.
   5. For 3 records the address is blank (empty string) which can be dropped.

A picture containing text, screenshot, display, software

Description automatically generated

Figure 4.1a - Bar Chart for Count of Missing Values in Dataset

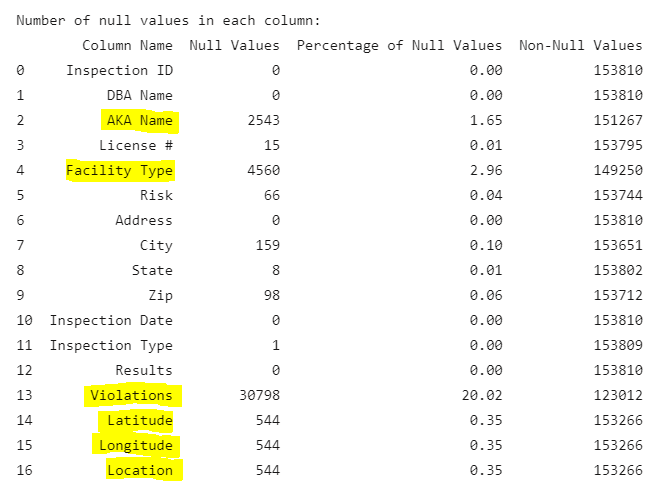


Figure 4.1b - Counts of Null Values in Dataset

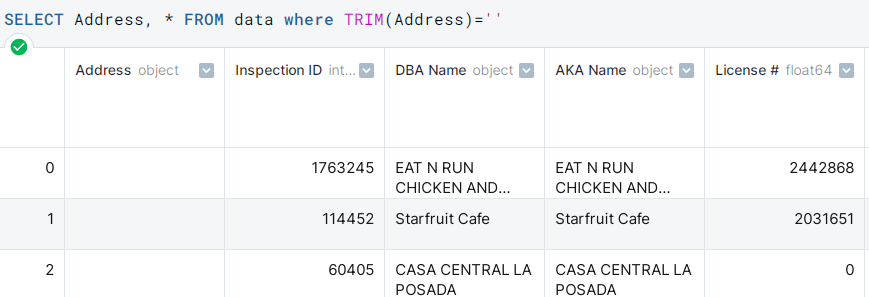


Figure 4.1c - Records with blank address

1. Duplicated Values – Since the dataset is given as a flat file of inspection events, there are many duplicates among food establishments and location entities. For example, as you can see in figure 4.1d below, License #, DBA Name, Address, and Location only have a fraction (10-20%) of unique values as compared to overall dataset size. Using the SQL queries shown below, we can see there should be around 17k unique locations and 34k food establishments. It is important to verify the duplicates are eliminated before loading the data into our database otherwise the numbers shown in our visualizations will be off due to double counting.

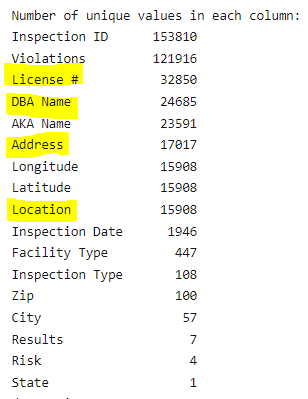


Figure 4.1d - Number of unique values in Dataset

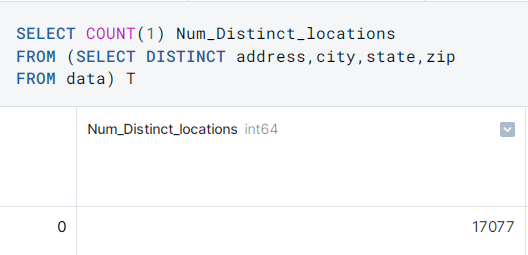


Figure 4.1e - Number of distinct locations in Dataset

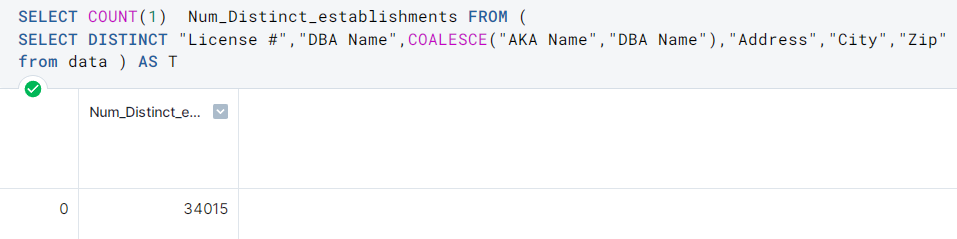


Figure 4.1f - Number of distinct establishments in Dataset

1. Inconsistent Naming and Typos- As shown in figure 4.1e below, the same Facility Types are listed in different ways. This will need to be clustered and combined using OpenRefine.
   1. Other string columns that can be clustered include Inspection Type, DBA Name, AKA Name, Address, and City.
   2. For string columns like DBA Name, Facility Type, Address and City the values should be made same case (uppercase/titlecase).

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 4.1e - Facility Type Clustering Examples

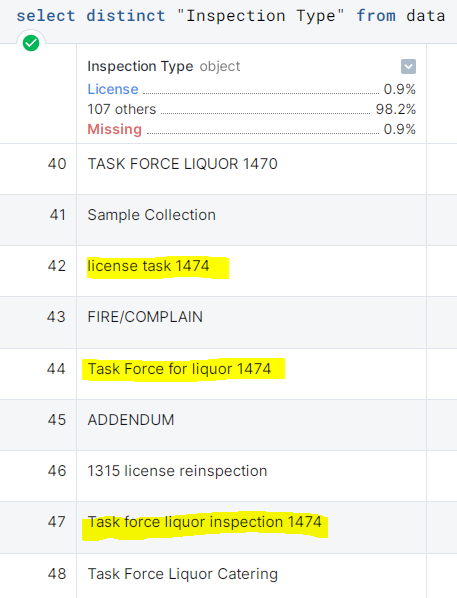
 

Figure 4.1g&h – Inspection Type & City Clustering Examples

1. Correct Datatypes - Zip needs to convert from float to string and trim to 5 characters. License should be converted to integer and convert NaN to 0, so can filter them out. Inspection Date needs to be parsed into date object.

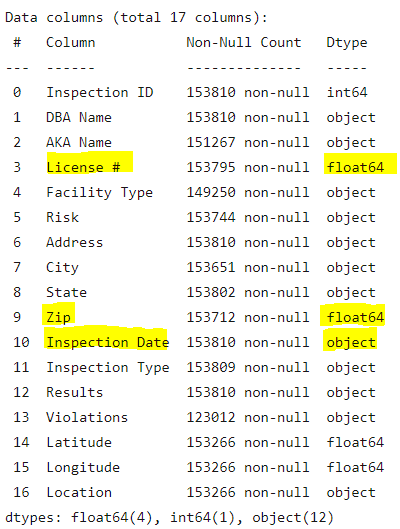


Figure 4.1i – Datatype Conversion Examples

1. The Violations column contains the code, description and comments all listed in a single column. These will need to be parsed using Regex in order for us to store it into the DB schema. In total there are 46 distinct violation codes, each having a unique description. This description should be stored with the code in a master table and should be made to have the same sentence case. The comments should be stored with each inspection violation code. Note that there can multiple lines of comments for each violation.

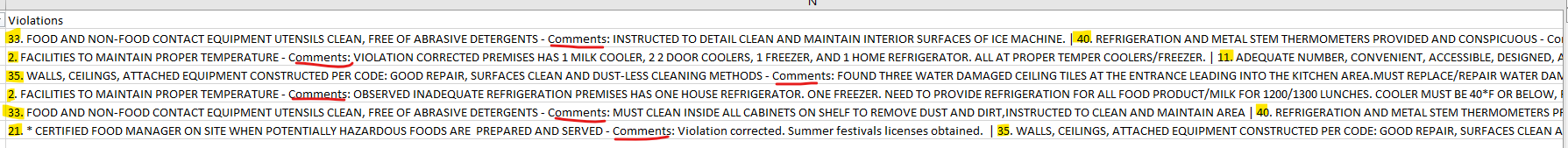
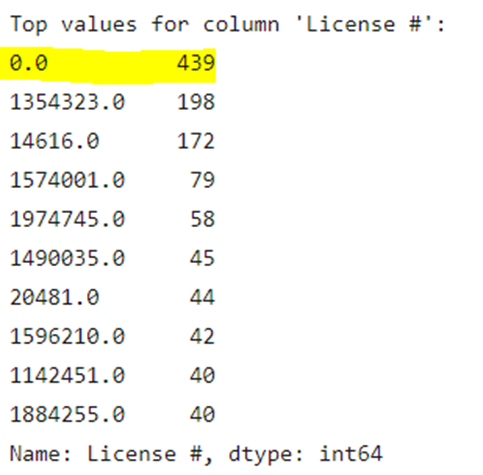


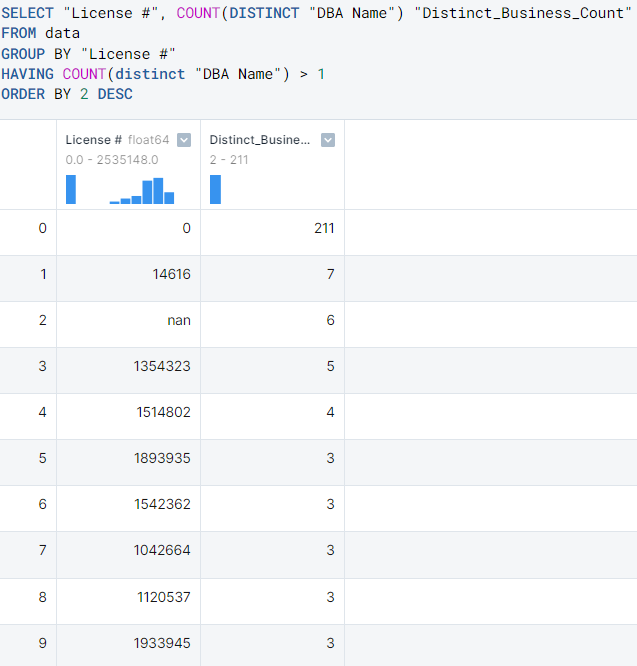
Figure 4.1k – Violations Format

1. License # as zero - there are 439 records with a license number of 0.

A screenshot of a computer

Description automatically generated with medium confidence Figure 4.1l&m– License # 0 Examples

1. License # is not enough to uniquely to a food establishment. As can see in screenshot below, there are cases where two different businesses share the same license #. Food establishments can also have different licenses either due to having multiple locations in case of franchises or a business might have had to be temporarily closed and then reassigned new license. In such case, we would need to correlate based on the inspection date to figure out most recent license # as valid one, but as of now that is out of scope for our use case. To uniquely identify a food establishment, we plan to create constraint on composite key index using following columns License #, DBA Name, AKA Name, and Location



Here is link our Python notebook used for this analysis:

[https://deepnote.com/@mcs-ds/CS513FinalProjectTeam59-5c00fda9-e408-](https://deepnote.com/@mcs-ds/CS513FinalProjectTeam59-5c00fda9-e408-45c6-ac1e-09c6fb54e7f2)4 5c6-ac1e-09c6fb54e7f2

## 4.2 Why data cleaning is necessary for main use case U1

Data cleaning is a crucial step to support the main use case of providing insights and suggestions based on food safety inspections. Currently the ‘Violation’ column contains various types of violations and their corresponding comments in a single column. To pave the way for a more detailed analysis and provide reliable insights and suggestions, data cleaning will be needed to extract these violations and comments to their own tables and columns. With cleaning, we plan to standardize the violation types and leverage it for analysis. Next to provide safety insights by facility types, we need a high level of consistency of the data. There are noticeable inconsistencies with the ‘Facility Types’ & ‘Inspection Type’ columns like spelling mistakes, inconsistent casing, and lack of standardization.

Without data cleaning of this dataset, we will not be able to ensure integrity and validity of the data, which will not achieve our goal of providing accurate and reliable insights for our consumers.

# **5. Phase-II Initial Plan**

## 5.1 Data Cleaning Plan

1. General data cleaning using OpenRefine, Python, SQLite, & YesWorkflow tools
   1. Check for missing values
   2. Check for and remove Duplicate records
   3. Fix & standardize inconsistent data
   4. Data type conversions
   5. Formatting columns to same case (uppercase/title case).
   6. Perform text cleaning to remove extraneous special characters, punctuations, and leading, trailing and consecutive whitespaces
   7. Check for and Remove outliers
2. Use clustering to group and clean categorical data column like Risk, Inspection Type, Results are inconsistencies or misspellings
3. Parse violation columns using Python and RegEx
   1. Extract each violation to its own table
   2. Extract comments out for each violation and store with Inspection Violation mapping
4. Fix Facility Type & Inpsection Type column using OpenRefine and python
   1. Standardized spelling and facility type name
   2. Cluster and merge similar names to use standardized name
5. Create SQL constraints to ensure unique locations & food establishments
6. Load data to SQLite database and perform data integrity checks via SQL.
7. Generate YesWorkflow model using OR2YW Tool
8. Demonstrate Data quality improvements
9. Document Data Cleaning procedure into final report

The cleaned data will be loaded to the SQLite database matching the ERD. It will be used for the data analysis steps further to answer the main use case questions and visualizations.

## 5.2 Who in the team will be responsible for which steps and Timeline

|  |  |  |
| --- | --- | --- |
| **Action** | **Who** | **Deadline** |
| S1 - General data cleaning (trailing spaces, missing values, duplicates, outliers) | Theara | Jul, 11 |
| S2 - Data cleaning using Clustering | Ashley | Jul, 11 |
| S3 - Clean violation column | Avinash | Jul, 11 |
| S4 - Fix facility type column | Avinash | Jul, 13 |
| S7- Create workflow model using YesWorkflow tool | Ashley | Jul, 14 |
| S6 - Create the python script to load the SQLite tables with the cleaned data | Avinash | Jul, 18 |
| S8 - Perform the U1 analysis and visualizations | Theara | Jul, 21 |
| S9 Write detailed description of data cleaning performed. Describe all data cleaning steps performed (high-level) and explain the Rationale. | All | Jul, 24 |
| S9 Document data quality changes, show how the data improved and quantify it. Which columns and how many cells affected? | All | Jul, 24 |
| S9 Work on Conclusions and Summary | All | Jul, 24 |
| S9 Finish Final Report | All | Jul, 24 |