CS513: Theory & Practice of Data Cleaning

Final Project Phase 2 Report

Team 59: Data Mavericks

Avinash Baldeo, Ashley George, Sotheara Chea

abaldeo2@illinois.edu, ageorge8@illinois.edu, chea3@illinois.edu

Table of Contents

1. Description of Data Cleaning Performed 3

1.1 High Level Data Cleaning Steps 3

1.2 Rationale for Data Cleaning Performed 3

1.3 Comparision to Phase I Plan 6

2. Document data quality changes 6

2.1 Summary of Data Cleaning Changes 6

2.2 Data Quality Improvement 11

3. Workflow Model 15

3.1 Outer Workflow 15

3.2 Inner Workflow 16

4. Conclusions & Summary 20

4.1 Summary and Conclusions with Lessons Learned 20

4.2 Team Member Contributions 21

5. Supplementary Materials 22

1. Description of Data Cleaning Performed

1.1 High Level Data Cleaning Steps

(Identify and describe all (high-level) data cleaning steps you have performed.)

For phase II of this project, our group has performed several major data cleaning operations aimed at improving the Chicago Food Inspections dataset data quality issues that were identified through our phase 1 analysis work. These data cleaning steps were necessary to help support our main use case, U1, which was to provide local food consumers with an interactive visualization dashboard of Chicago's different food establishments, giving insights into their overall health risk based on past the inspection results. The following are the key high-level data cleaning steps that we took:

1. General data cleaning using OpenRefine to handle datatype conversions, case formatting, and trimming/collapsing extra whitespace characters
2. Clustering columns such as *Inspection Type, Facility Type, DBA name, Address* and City *in OpenRefine* to handle inconsistent data naming and misspellings/typos.
3. Backfilling missing Geolocation data for Establishment Locations using a Geoencoding API in Python. This included *Latitude*, *Longitude*, Location, *Zip*, *City* and *State* columns.
4. Creating SQL constraints on the FoodEstablishment and EstablishmentLocation tables to ensure duplicates are eliminated when loading data into database.
5. Using Python regex to parse the *Violations* column and extract each Food Inspection violation and the respective comments to be stored in the normalized database tables. This enabled detailed analysis of individual violations.
6. Using Python to handle missing data & eliminate duplicate rows before loading the cleaned dataset int our SQLite database created using the database models we have defined in phase 1.
7. After loading the data, we performed data integrity checks using SQL queries then exported the normalized database to a CSV flat file for use in the visualization dashboard.

1.2 Rationale for Data Cleaning Performed

(For each high-level data cleaning step, you have performed, explain its rationale. Was the step really required to support use case U1? Explain. If not, explain why those steps were still useful.)

The rationale for data cleaning performed are as follows:

1. General Data cleaning
    i. Missing values: Having missing values can lead to inaccurate data and biased analysis. By fixing the missing values, we ensure that the data used for our Chicago Food Establishment Insight Tool, U1 case, is complete and representative. This is important as we need complete data to analyze the risk level and violations as it compares to its facility type.
    ii. Inconsistent data: inconsistent formats and spelling variations will hinder our data analysis and visualizations. This step creates uniformity and facilitates easier grouping and filtering for our consumers. This step is essential to our U1 case and allows us to accurately correlate risk level with specific facility types and DBA names. This helps our consumers make informed decisions based on consistent data.
    iii. Data type conversion: Correct data type is important for appropriate calculations and comparisons. Converting our data to the appropriate types enables accurate analysis. This step was not absolutely required to support our U1, but it was still important if we want our consumers to filter the insights by date range. It also was needed to ensure the data gets loaded to the SQLlite database with appropriate data types.
    iv. Formatting: Consistent casing makes it easier to compare data, group similar values, and perform search/filtering. It also increases the readability of our data. This step supports our U1 as it made it easier to perform our cluster cleaning step and make it easier for our consumers to read our insights and visualizations.
    v. Removing whitespaces: Extra whitespace characters at the beginning or end of strings may go unnoticed but can affect data matching and filtering. Removing this unnecessary noise from the data leads to consistent and accurate analysis. This step supports our U1 by ensuring our textual data like the violations and addresses can by properly filtered to focus on inspection results of interest.
2. Clustering Columns and Data Standardization
    i. Multiple columns in the Food establishments' inspection data have inconsistent data naming and misspellings. For example, the "Facility Type" column has variations like "Rest," "Restaurant," or "Restaurant/Grocery." Inconsistent data naming can lead to fragmented or misleading visualizations. By clustering similar

values, the visualization dashboard can present a clear and coherent picture of food establishments' characteristics and inspection results. This ensures that the insights presented to consumers are meaningful and easily interpretable, fostering informed decision-making based on the data-driven findings.

3. Backfilling missing Geolocation data for Establishment Locations using Geocodio API in Python:
    i. Geolocation data, such as latitude, longitude, and address information, is crucial for visualizing and analyzing the distribution of food establishments on a map. Backfilling missing geolocation data enhances the accuracy and completeness of the dataset, enabling a more precise representation of each food establishment's location. This information supports the main use case U1 by allowing consumers to easily identify the geographic distribution of food establishments with respect to their inspection results and risk levels.
4. Creating SQL constraints on the FoodEstablishment and EstablishmentLocation tables:
    i. SQL constraints ensure data integrity and consistency in the database. By enforcing uniqueness on certain columns (e.g., address, city, zip, state for Locations), duplicates can be prevented during data loading. This constraint ensures that each food establishment is uniquely identified in the database, which is essential for the accuracy of analysis and visualization. For use case U1, having clean and consistent data prevents any confusion caused by multiple entries representing the same food establishment. This allows consumers to access reliable information about each establishment and gain trust in the insights provided by the visualization dashboard.
5. Using Python regex to parse the Violations column and extract each Food Inspection violation and the respective comments:
    i. The Violations column often contains multiple violations and comments bundled together. Extracting each violation and its corresponding comment into separate normalized database tables allows for a detailed analysis of individual violations. This granularity enables a deeper understanding of the types and frequencies of violations across different food establishments. In use case U1, we can use this information to see what the most common violation codes are based on the type of inspection & type of facility,

6. Using Python to filter/handle missing data & eliminate duplicate rows before loading the cleaned dataset into our SQLite database:
    i. Handling missing data and eliminating duplicate rows are essential data cleaning steps that improve data quality. Missing data can lead to inaccurate analysis, while duplicates can skew results and misrepresent numbers shown on a dashboard. By filtering out missing data and duplicates, we ensure that the cleaned dataset is reliable and consistent. For use case U1, a clean dataset is essential for generating accurate insights that represent the food establishments' inspection history.
7. Performing data integrity checks using SQL queries and exporting the normalized database to a CSV flat file:
    i. Data integrity checks verify the correctness and consistency of the data after cleaning and preparation. Ensuring data integrity is crucial for avoiding errors and inaccuracies in the visualization dashboard. Exporting the normalized database to a CSV flat file format allows for seamless integration with the visualization tools, such as Tableau.

1.3 Comparision to Phase I Plan

The main difference between our actual phase II workflow versus our phase 1 plan is that we went ahead with backfilling the location data using Geocoding technique. While the data quality issue was identified and mentioned in 4.1 of our phase 1 report, we did not have an explicit step in our phase 1 plan to address this as we were not confident at that point if the missing locations were valid or not. Upon closer inspection we did find they were valid and could be fixed with a Python library called PyGeocodio. Another minor difference is that we did not have to cluster columns like Risk & Results as there were no outliers in these columns. Overall, we found that we did stick closely to our phase 1 plan in terms of data cleaning, although the order might have been slightly different. For example, the step to parse violations columns using Python and Regex was done as part of the script to populate the database tables.

2. Document data quality changes

2.1 Summary of Data Cleaning Changes

OpenRefine Data Cleaning Performed			
Steps	Description	Rationale	Summary
1	Text transform- trim leading and trailing whitespaces	Required for data consistency and accuracy. One of our aims with U1 is providing risk severity by geographic locations, so having a	Address: 153,366 cells Violations: 18,580 cells

		consistent address will help with this.	
2	Text transform to Number	Required to perform numeric comparisons and statistical models.	Inspection ID: 153,795 cells License #: 153,795 cells
3	Dealing with Null/Blank values	Required for data integrity and quality. For our U1 case, there were 4560 locations in the dataset with null facility types. To get the correlation between the risk level and facility type, each location must have a facility type. We change all null cells in the facility type to "OTHER/UNKNOWN"	Facility Type: 4,560 cells
4	Text transform to Upper Case	To increase readability and consistency in the dataset. This will also help with facilitating data matching and clustering the data.	DBA Name: 10,443 cells AKA Name: 9,459 cells City: 348 cells
5	Cluster "like" names into a single value	Required to data consistency and reduce the spread of different naming convention and input error. For example, in the city column, there were misspelling of 'CHICAGO' as 'CCHICAGO' and 'CHCHICAGO'	DBA Name: 16,860 cells City: 153,485 cells Inspection Type: 101,910 cells
6	Text transform to Title Case	To increase readability and consistency in the dataset. In our U1 case, our inspection type was transformed to title case to create a consistent and readable format	Inspection Type: 29,910 cells Facility Type: 9,544 cells Address: 152,667 cells
7	Text transform to Date	To provide filtering and sorting of date range. This will also support our data analysis to provide trends and patterns to create our U1 insights	Inspection Date: 153,810 cells
8	Text transform to collapse consecutive whitespaces	To remove consecutive whitespaces with only one	Facility Type: 13 cells Address: 618 cells

		space. This will help avoid the reduce mismatches due to extra spaces.	DBA Name: 2,920 cells AKA Name: 3/560 cells
9	Cluster "like" names into a single value	Required to data consistency and reduce the spread of different naming convention and input error. Like combining like terms like: "Daycare" and "Day Care"	Facility Type: 5,191 cells Inspection Type: 16,478 cells
10	Text transform to Title Case	To increase readability and consistency in the dataset. In our U1 case, our inspection type was transformed to title case to create a consistent and readable format	Results: 28,324 cells City: 153,651
11	Text transform to collapse consecutive whitespaces on Violations column	Clean multiple consecutive spaces with one. This will yield better results when we later extract violation to a separate table.	Violations: 84,606 cells
12	Text transform to remove some Inspection Types with extraneous digits at the end	Removed extraneous digits at the end of Inspection type column	Inspection Type: 46 cells
13	Text transform- trim leading and trailing whitespaces	Required for data consistency and accuracy. One of our aims with U1 is providing risk severity by geographic locations, so having a consistent address will help with this.	Inspection Type: 46 cells
14	Text transform to String	Required for data consistency and textual data analysis.	Risk: 66 cells Violations: 30,798 AKA Name: 2,543 cells City: 159 cells Zip: 98 cells Location: 544 cells Inspection Type: 1 cell
15	Clustering fixes to categorical fields	To cluster and standardize categories in the fields so that we can limit the number or types. Performed multiple	Inspection Type 7846 cells DBA Name AKA Name

		rounds of clustering on same column to standardize.	Facility Type City 2 cells

Python Data Cleaning Performed			
Steps	Description	Rationale	Summary
1	Repair Missing Location Data	This was done to enhance the accuracy of map-based visualizations on the dashboard	Used PyGeocodio API to Geocode Latitude, Longitude using Address. Used Reverse Geocoding to fill Zip, City, & State
2	Normalize Columns Names	This was done to map columns from the CSV to the Database schema	Replaced spaces in column names with underscore and made all column names lowercase

3	Derive Risk Level & Risk Category Columns from Risk	This was done to split/extract Risk Category (High/Medium/Low) and Risk Level (1,2,3) into separate columns to show & filter on the dashboard	Used Regex for risk_level
			`str.extract('(\d+)')`
			Used Regex for risk_category
			`.str.split(' \(', expand=True).strip(')')`

4	Convert Data Types	This was done to map the data types from python to match the database schema	Parse Inspection Date and convert to Datetime
			`pd.to_datetime(df['inspection_date'], format='%Y-%m-%dT%H:%M:%SZ')`
			Convert License# to integer
			`pd.to_numeric(df['license_number'], errors='coerce').astype('Int64')`

5	Replace Invalid values (Nan/Inf) with None	This was done to avoid loading incompatible data in the database which would result in an error in our python process	City, Location, Latitude, Longitude, State, AKA Name `.astype(str).replace({'nan': None, 'inf': None})`

6	Drop Blank/Null Rows	This was done as we have not null constraint for address field in our database schema	Address: 3 rows `df.drop(null_rows)`
7	Format & Trim Zipcode	This was done to convert Zipcode from float to string to fit into database column	Convert Zip to string and trim to first 5 characters `.astype(str).apply(lambda x: x[:5]).`
8	Split Violations column and extract Codes, Descriptions and Comments	This was done to store inspection violations, health code and their description in normalized database tables	Used below regex `re.match(r'(\d+)\.\s+(.+?)\s+-\s+Comments:\s+(.+)', line.strip())`

9	Trim trailing/leading & consecutive whitespace in Violation Description & Comment	This was done to avoid formatting/display issues for these fields as a general best practice	`violation_df['description'].str.replace(r'\s+', ' ', regex=True)`
			
			`inspection_violation_df['comment'].str.replace(r'\s+', ' ', regex=True)`

10	Strip & replace special character in Violation Comment and Description	This was done to remove asterisk character which might cause display issues	`violation_df['description'].str.replace('*', '')`
11	Make Violation Comment uppercase	This was done to make filtering consistent on this field	`inspection_violation_df['comment'].str.upper()`
12	Drop duplicated Inspection Violation rows	This was done to avoid primary key constraint that says combination of inspection ID	`inspection_violation_df.drop_duplicates()`

		and violation code is unique	
13	Fill Nulls in AKA Name with DBA Name	This was done as we use this field as part of composite key to identify unique establishment	`food_inspection_df['aka_name'].fillna(df['dba_name'])`

14	Manually Fix Location Data for 2 addresses	After backfilling location, found that these 2 same addresses had different Location coordinates, so we manually corrected in code	Used pandas df assignment to set location for below addresses
			`3901 S Dr Martin Luther King Jr Dr`
			`4628 N Cumberland Ave`

15	Drop Duplicated Food Establishment rows	This was done to satisfy integrity constraint added that Establishments are unique	`df.drop_duplicates(subset=['license_number', 'dba_name', 'aka_name'])`
16	Drop Duplicated Establishment Location rows	This was done to satisfy integrity constraint added that Locations are unique	`df.drop_duplicates(subset=['location'])`)
17	DB Extraction to CSV Flat File	This was done to export inspections that had violations to be shown on our tableau dashboard	Ran sql query in export_db_flat_file. Resulted in 31481 rows without violations filtered. 586,563 -> 555,082

2.2 Data Quality Improvement

Query #	Query Statement	Qu(D) Results	Qu(D') Results	ΔD Difference	Comments

1	SELECT count(*) - count(state) AS Nulls_in_state FROM EstablishmentLocation;	8	0	-8	Missing State Data Repaired
2	SELECT count(*) - count(city) AS Nulls_in_city FROM EstablishmentLocation;	157	0	-157	Missing City Data Repaired
3	SELECT count(*) - count(zip) AS Nulls_in_zip FROM EstablishmentLocation;	96	0	-96	Missing Zip Data Repaired
4	SELECT count(*) - count(latitude) AS Nulls_in_latitud FROM EstablishmentLocation;	541	0	-541	Missing Latitude Data Repaired
5	SELECT count(*) - count(longitude) AS Nulls_in_longitude FROM EstablishmentLocation;	541	0	-541	Missing Longitude Data Repaired
6	SELECT count(*) - count(location) AS Nulls_in_locatio FROM EstablishmentLocation;	541	0	-541	Missing Location Data Repaired
7	SELECT count(1) FROM EstablishmentLocation	153807	16017	-137790	Duplicate Locations Removed
8	SELECT address, city, STATE, zip, count(DISTINCT id) FROM EstablishmentLocation GROUP BY address, city, STATE, zip HAVING count(DISTINCT id) > 1;	15073	0	-15073	Unique Locations constraint enforced
9	SELECT COUNT(*) - COUNT(aka_name) AS NullsInAkaName FROM FoodEstablishment;	2543	0	-2543	Nulls filled with DBA Name
10	SELECT COUNT(*) - COUNT(facility_type) AS NullsInFacilityType FROM FoodEstablishment;	4559	0	-4559	Missing Facility Type Removed
11	SELECT count(1) FROM FoodEstablishment;	153807	33916	-119891	Duplicate Establishments Removed
12	SELECT license_number, dba_name, aka_name,	24194	0	-24194	Unique Establishments constraint enforced

	`count(DISTINCT id) FROM FoodEstablishmen GROUP BY license_number, dba_name, aka_name HAVING count(DISTINCT id) > 1;`				
13	`SELECT count(1) Violations_Count FROM InspectionViolation`	123012	555082	432070	*Split Violations Column into separate rows, Qu(D) is single column row count
14	`SELECT count(DISTINCT dba_name) FROM FoodEstablishment;`	24683	23136	-1547	Clustered DBA Name Result
15	`SELECT count(DISTINCT facility_type) FROM FoodEstablishment;`	447	256	-191	Clustered Facility Type Result
16	`SELECT count(DISTINCT inspection_type) FROM FoodInspection;`	108	68	-40	Clustered Inspection Type Result
17	`SELECT count(DISTINCT address) FROM EstablishmentLocation;`	17016	16017	-999	Clustered Address Result
18	`SELECT count(DISTINCT city) FROM EstablishmentLocation;`	57	42	-15	Clustered City Result

Number of null values in each column:

	Column Name	Null Values	Percentage of Null Values	Non-Null Values
0	DBA Name	0	0.00	555082
1	AKA Name	6465	1.16	548617
2	License #	36	0.01	555046
3	Facility Type	336	0.06	554746
4	Address	0	0.00	555082
5	City	362	0.07	554720
6	State	19	0.00	555063
7	Zip	182	0.03	554900
8	Latitude	1600	0.29	553482
9	Longitude	1600	0.29	553482
10	Location	1600	0.29	553482
11	Inspection ID	0	0.00	555082
12	Inspection Type	2	0.00	555080
13	Inspection Date	0	0.00	555082
14	Results	0	0.00	555082
15	Risk	3	0.00	555079
16	Risk Category	3	0.00	555079
17	Risk Level	3	0.00	555079
18	Violation Code	0	0.00	555082
19	Description	0	0.00	555082
20	Comment	0	0.00	555082

Figure 2.1 - Counts of Null Values in DB Extract loaded from Raw Dataset

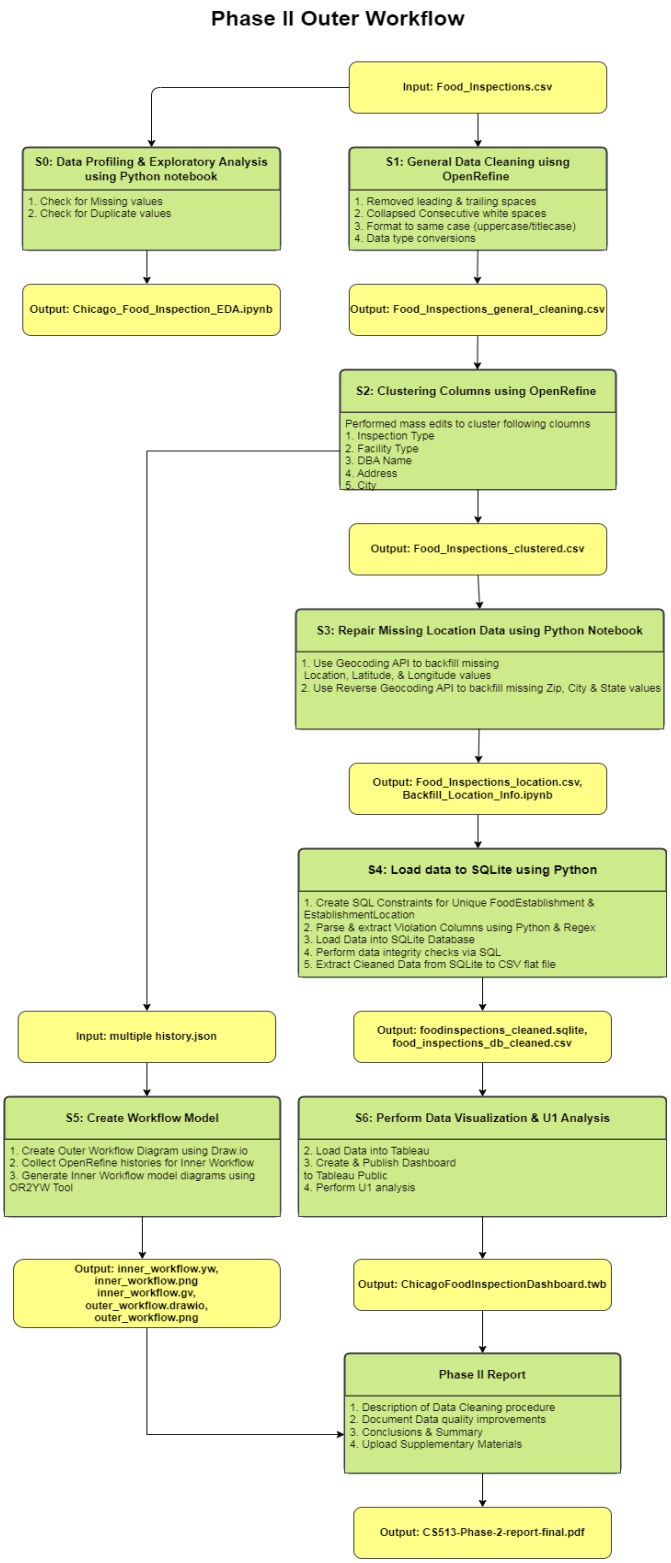
Number of null values in each column:

	Column Name	Null Values	Percentage of Null Values	Non-Null Values
0	DBA Name	0	0.00	555082
1	AKA Name	0	0.00	555082
2	License #	36	0.01	555046
3	Facility Type	0	0.00	555082
4	Address	0	0.00	555082
5	City	0	0.00	555082
6	State	0	0.00	555082
7	Zip	0	0.00	555082
8	Latitude	0	0.00	555082
9	Longitude	0	0.00	555082
10	Location	0	0.00	555082
11	Inspection ID	0	0.00	555082
12	Inspection Type	2	0.00	555080
13	Inspection Date	0	0.00	555082
14	Results	0	0.00	555082
15	Risk	3	0.00	555079
16	Risk Category	3	0.00	555079
17	Risk Level	3	0.00	555079
18	Violation Code	0	0.00	555082
19	Description	0	0.00	555082
20	Comment	0	0.00	555082

Figure 2.2 - Counts of Null Values in DB Extract loaded from Cleaned Dataset

3. Workflow Model

3.1 Outer Workflow



Design & Tools Used Overview

OpenRefine – We used OpenRefine for General Data cleaning because it has prebuilt functionality and allows us to visually see the changes as we were cleaning the data. It also has better Clustering algorithms than we could implement on our own in python.

Python Notebooks & Scripts – We used Python notebook scripts to backfill missing latitude and longitude data. We also used pandas and ponyorm libraries for loading the cleaned CSV file into our SQLite database. We used the web-based notebook platform Deepnote which is like Google Collab for the interactive data profiling in phase 1 and for experimenting with the Geocoding API for the first time.

SQLite – We used sqlite as our database since it is the simplest to use and can handle a dataset of this size (<600k rows) with no issue. We used it to load the data then perform data integrity checks to ensure data quality has improved.

Draw.io – We used this online diagram tool to generate our outer workflow diagrams

OR2YWTool – We used to generate a YesWorkflow model from OpenRefine history

Tableau – We used Tableau public for creating and hosting our interactive dashboard since it's free & current industry standard.

3.2 Inner Workflow

The combined inner workflow history is made available under the Supplementary materials. As it is too large to see visible on this document, we have broken it into sub snippets to show each phase of the OpenRefine cleaning process.

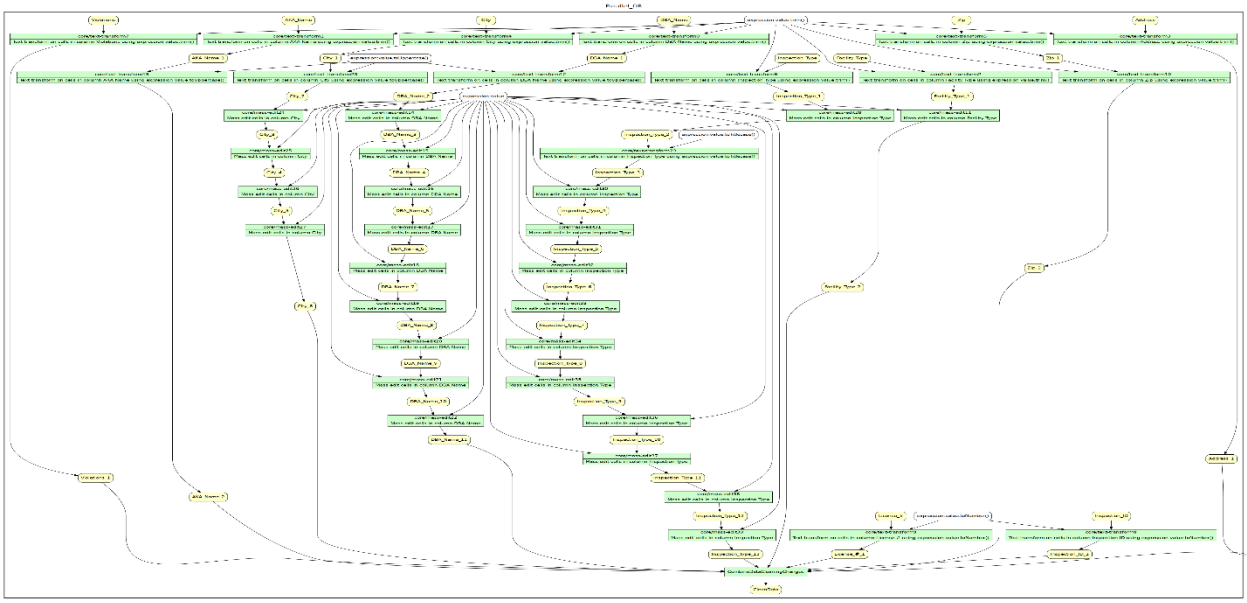


Figure 3.1 - General Data Cleaning YesWorkflow Diagram

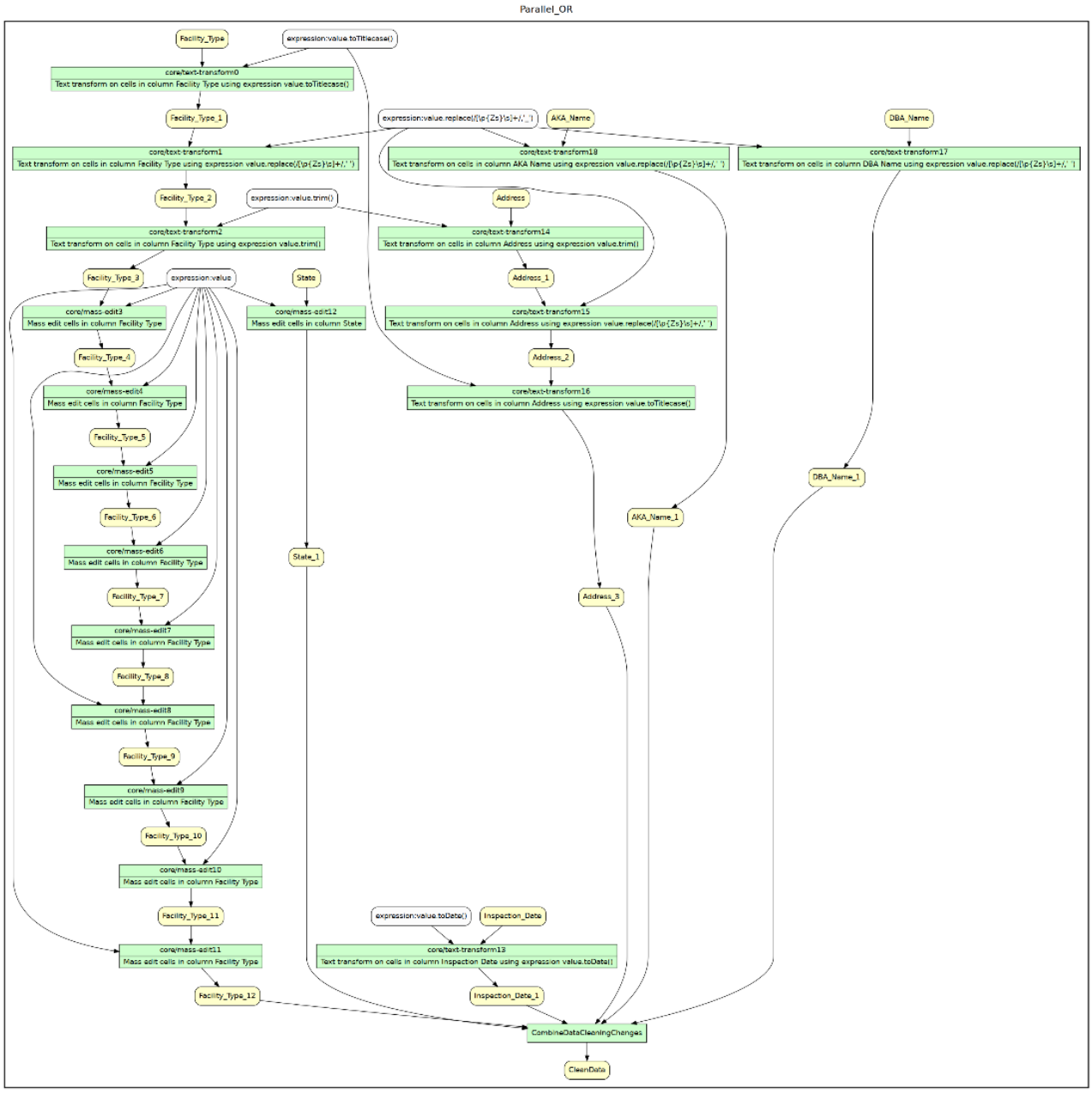


Figure 3.2 - Initial Clustering YesWorkflow Diagram

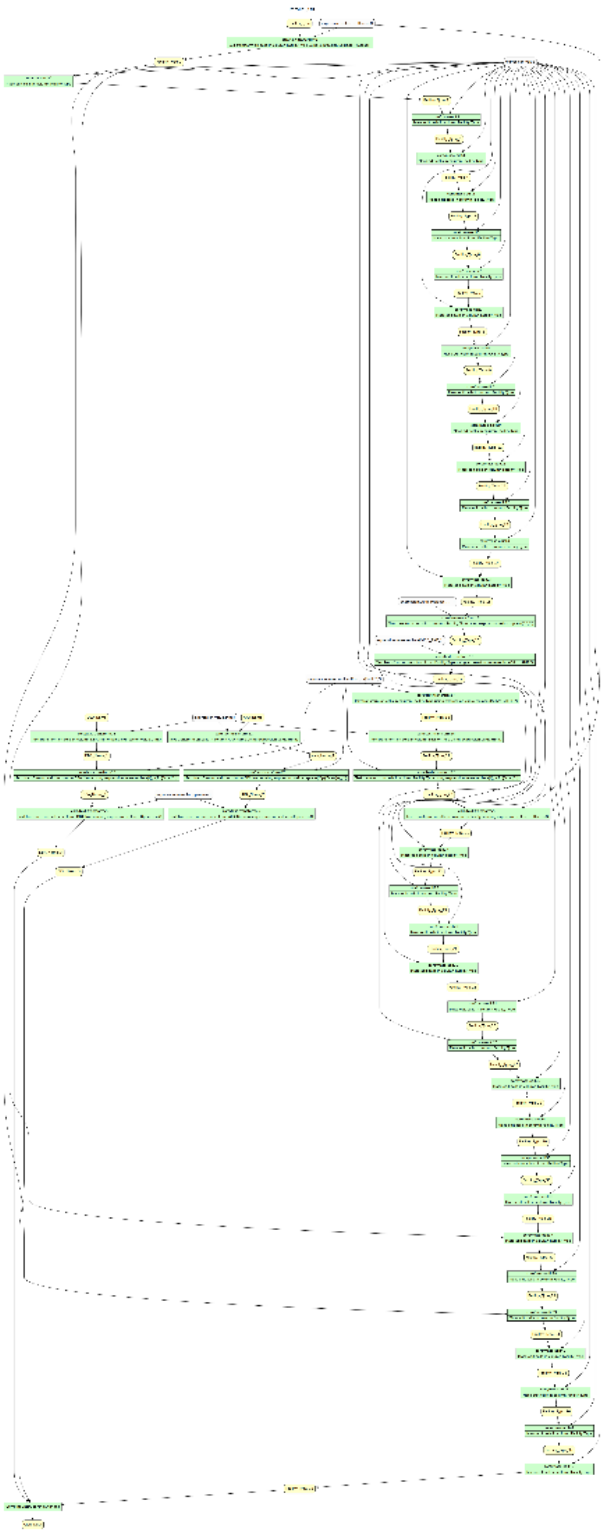


Figure 3.3 - Facility Type Clustering YesWorkflow Diagram

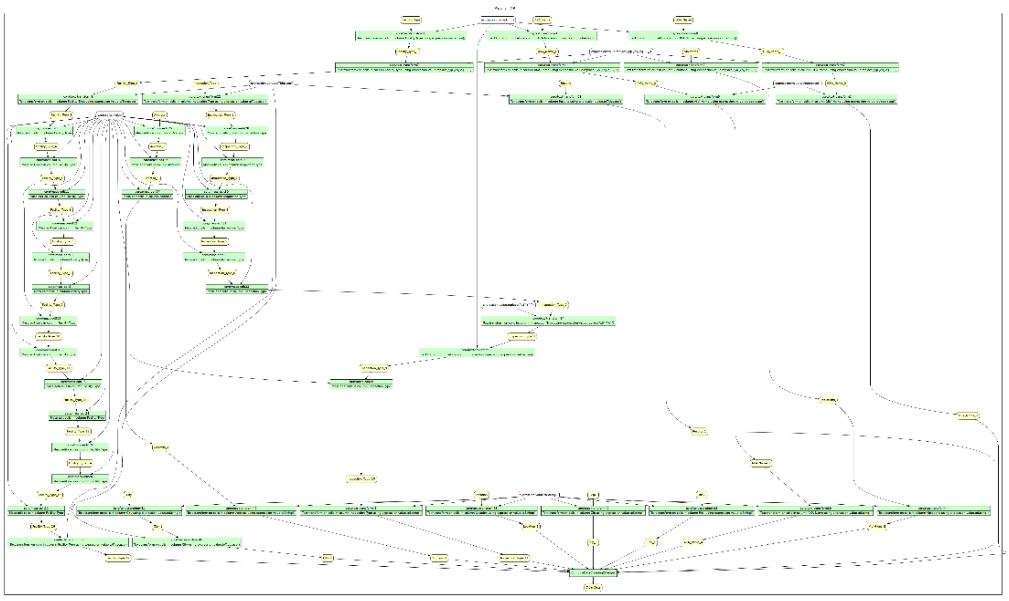


Figure 3.4 - More Clustering & Cleaning YesWorkflow Diagram

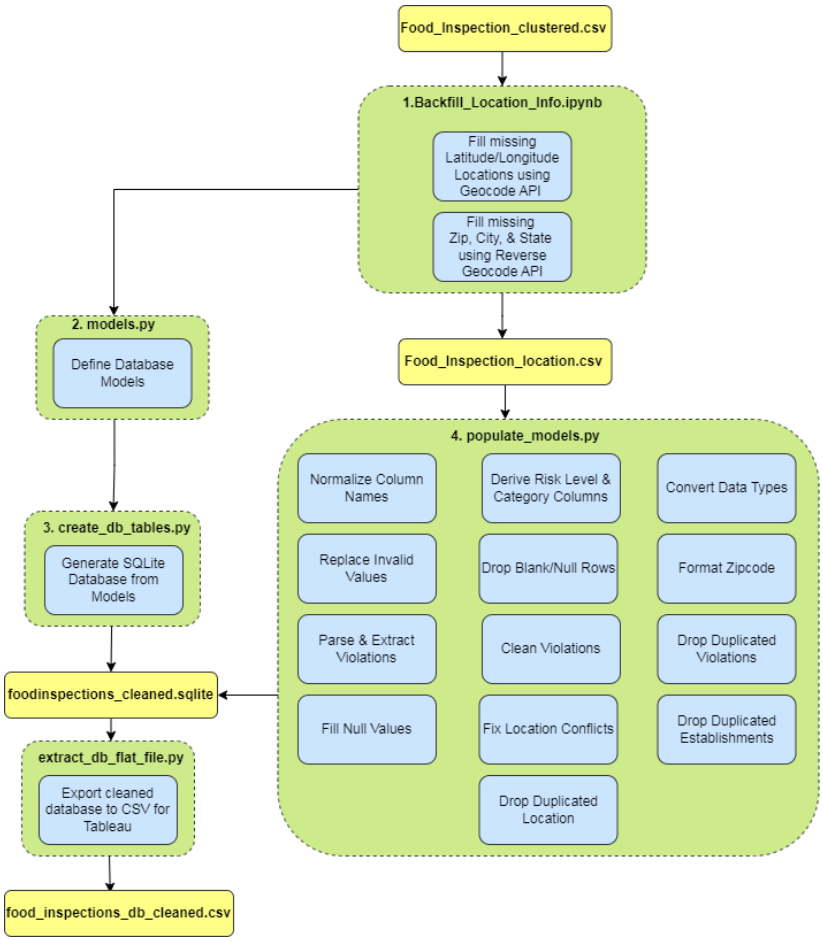


Figure 3.5 - Python Inner Workflow Diagram

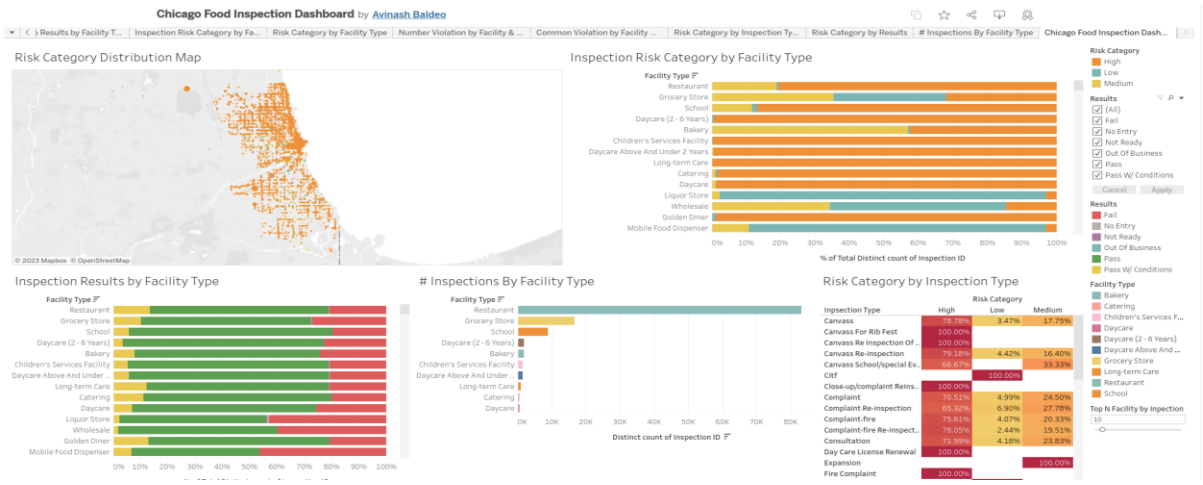
4. Conclusions & Summary

4.1 Summary and Conclusions with Lessons Learned

In this project we aim to transform our chosen dataset, Chicago Food Inspection, into a clean and high-quality data to support our main use case, U1. Our main use case is to build an insightful visualization tool that will provide information on the safety of Chicago food establishments based on their inspection violation result, frequency, and risk levels. To achieve our goal in the main use case, various data cleaning steps were performed to get our given dataset to its highest-level quality. This will make the new dataset suitable for analysis and visualization for our insight tool. We intended to address key questions related to food safety such as:

- Correlation between facility type and risk level from the inspections
- Variation in inspection results among different types of establishments
- Most common violation codes by inspection type and facility type
- Distribution of inspection results and risk severity by geographic locations

With the cleaned data, we built a visualization dashboard using Tableau to provide the 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.



The detailed interactive dashboard is hosted on the tableau public link below:

https://public.tableau.com/app/profile/avinash.baldeo/viz/ChicagoFoodInspectionDashboard/ChicagoFoodInspectionDashboard

The current dataset covers the period from 2011 through 2017. As the city of Chicago updates the data each year, the data cleaning steps we have performed could be automated and maintained as a pipeline to keep the visualization dashboard up to date as future work.

Lessons Learned:

- Data cleaning is crucial to data visualization: To achieve our main use case, it was crucial that our given dataset was properly cleaned to ensure high data integrity and quality. Otherwise, it would greatly decrease accuracy and reliability of the insights that we are providing to our end users who are Chicago Food consumers. The original dataset would not have provided misleading/incomplete results on the dashboard.
- Importance of understanding use case before cleaning: Understanding the intended use case for the dataset is essential in identifying the appropriate data cleaning steps. For example, if our main use case for this dataset was to just provide the results of an inspection based on a given inspection ID, then it be unnecessary to parse out individual comments in the *Violations* column or perform any clustering work on the *Facility Type* column.
- Data clustering for normalization: By clustering our various categorical data like DBA Names, Facility Type, and Inspection type, this makes our data consistent, fixing spelling errors and standardizing different naming conventions. Clustering provided a powerful mechanism to standardize inconsistent and wrongly formatted data collected over a long period into a uniform format. This greatly helped us in providing consistent and concise reporting dashboards.
- Power of different data cleaning tools: Using different tools that were available to use, such as OpenRefine, Python, SQLite, and Regex, allowed us to perform our essential data cleaning steps quickly and efficiently. For Clustering and all basic textual cleaning, OpenRefine was very effective. The SQL constraints and queries helped us validate the integrity of the cleaned data. Python and regex proved very useful in doing specialized cleaning after the OpenRefine cleaning.
- Data cleaning is an iterative process and will need to be performed using multiple relevant tools best suited for each step for the best results.

In conclusion, this project successfully demonstrated the importance of data cleaning to achieve one's main use case for a dataset. The data cleaning was necessary and sufficient to provide the meaningful insights we planned for our U1. Various data cleaning steps were performed to help us ensure that the insights for our visualization tool are consistent and accurate. We have learned many lessons throughout this project.

4.2 Team Member Contributions

Ashley	Creating dataset use cases Complete data clustering steps

	Document & quantify data quality changes in summary table Write description of data cleaning steps performed and rationale Inner YesWorkflow models from OpenRefine history Work on the summary, conclusion and lesson learned Add Supplementary materials (Workflow model & OpenRefine History)
Avinash Baldeo	Exploratory data analysis and profiling Creating database ERD diagrams and table schemas Python notebook to backfill missing Location data Python scripts for parsing violations column & populating database model Shells scripts for creating database tables, loading raw and cleaned datasets, and exporting final database schema Data quality integrity checks Building interactive dashboard using Tableau Outer workflow diagram Inner python workflow diagram Write description of data cleaning steps performed and rationale Document Data quality improvements Documenting Python data quality changes Work on the summary, conclusion and lesson learned Add Supplementary materials (Queries, Other History)
Sotheara	List dataset quality problems Create data cleaning plan and timeline Perform general data cleaning steps Document & quantify data quality changes in summary table Work on the summary, conclusion and lesson learned Add Supplementary materials (DataLinks)

5. Supplementary Materials

Please see attached ZIP folder with the following supplementary materials:

Workflow Model	• outer_workflow.drawio • outer_workflow.pdf • inner_workflow.yw • inner_workflow.gv • inner_workflow.pdf • python_inner_workflow.drawio • python_inner_workflow.pdf
OpenRefine Operation History	• OpenRefineHistory.json
Other History	• Chicago_Food_Inspection_EDA.ipynb • Backfill_Location_Info.ipynb • create_db_tables.py

	• models.py • populate_models.py • export_db_flat_file.py
Queries	• Queries.txt
Original and Cleaned Datasets	DataLinks.txt points to Box folder containing following: • Food_inspections.csv (Original Raw Dataset) • foodinspections_raw.sqlite (Phase 1 Database) • foodinspections_cleaned.sqlite (Phase 2 Database) • food_inspections_db_cleaned.csv (Cleaned Phase 2 DB extract) • food_inspections_db_raw.csv (Phase 1 DB Extract)

Note: The full project files are available on our GitHub repository URL