**CS 513 - Theory & Practice of data cleaning**

**Summer 2023**

**Team Project: Phase 2 Report**

**(Team - 30)**

Prof: [Bertram Ludaescher](https://mediaspace.illinois.edu/createdby/eyJpdiI6IjRHTGRHVXZHUThUS2xVeUJka0RZWXc9PSIsInZhbHVlIjoiVjlmV3NkOTZ2eXpJcTJTeFI1emh3QT09IiwibWFjIjoiNTcyZjE3M2MzZWEzMDkzYTg1ZjRjZWFkZmNkNmY4ZDY1YjA2OTAwMGUzODZiMjcxNDA1NWE5YWFjNTM0NTEwMCJ9)

Project Members:

Xiaojing Xu (xx33@illinois.edu)

Xinyu Fu (xinyufu2@illinois.edu)

Jake Xu (cxu54@illinois.edu)

July,30,2023

**1. Description of Data Cleaning Performed**

**1.1 Data Cleaning Process with OpenRefine**

During the initial data cleaning process, we focused on improving text string columns using OpenRefine. Specifically, we targeted and refined several key columns critical to our use cases, including "DBA Name," "Facility Type," and "Inspection Date." We utilized OpenRefine's powerful clustering capabilities and regular expression support to efficiently group similar entries and apply standardized replacements based on patterns and phonetic similarities.

The main data cleaning steps performed on the related columns are as follows:

DBA Name:

- Collapsed consecutive white spaces to ensure consistent formatting.

- Transformed text to uppercase for uniformity and ease of comparison.

- Utilized several clustering techniques with Key Collision and Fingerprint, Metaphone3, and Beider-Morse to identify and merge similar entries. Through this rigorous clustering and merging process, the dataset has been streamlined, leading to a notable reduction in the number of variations for establishments such as MCDONALD'S, SUBWAY, DUNKIN DONUTS, 7-ELEVEN, and numerous others.

- Manually resolved additional discrepancies for the name "MCDONALD'S," including "MACDONALD" and "MCDONALD'S CORPORATION."

AKA Name:

- Collapsed consecutive white spaces to unify the entries.

- Transformed the text to uppercase for consistency.

Facility Type:

- Collapsed consecutive white spaces to ensure uniformity.

- Transformed the text to uppercase for consistency.

- Utilized several clustering techniques using Key Collision with Metaphone3 and Daitch-Mokotoff to merge similar Facility Type entries, resulting in more consistent and accurate groupings of similar types of establishments.

- Used Regular Expressions to merge specific Facility Type entries that represented the same category but had slight variations in their names. For example, categories like "DAYCARE" and "MOBILE FOOD" were consolidated using regular expressions to encompass various similar subcategories under a single category.

- Blank values in the Facility Type column were filled with default categories based on specific keywords found in the DBA Names. Regular expressions were used to identify keywords related to "RESTAURANT," "GROCERY STORE," "DAYCARE," "BAKERY," "COFFEE/CAFE," and others, which were then used to categorize records with missing Facility Type information. For instance, entries with keywords related to "RESTAURANT," "PIZZA," "SANDWICH," etc., were assigned "RESTAURANT" as their facility type.

Address:

- Trimmed leading and trailing spaces and Collapsed consecutive white spaces for consistent representation.

- Transformed the text to uppercase for uniform formatting.

- Merged similar entries using clustering techniques like Key Collision and Fingerprint and n-gram (size 3) Fingerprint to group variants of the same addresses.

City:

- Transformed the text to uppercase for consistency.

State:

- Manually verified and filled in blank cells with "IL" to indicate the state (Illinois).

Inspection Type:

- Transformed the text to title case to standardize the format.

- Merged similar entries using clustering techniques like Key Collision and Metaphone3 to group related inspection types.

Violations:

- Trimmed leading and trailing spaces to remove unnecessary gaps.

- Collapsed consecutive white spaces for consistent representation.

Inspection Date:

- Convert the date to ISO date format 'yyyy-MM-dd' for better date handling and analysis.

Inspection Year:

- Created a new column based on Inspection Date to extract the year for easy year-wise analysis.

By utilizing diverse clustering techniques and regular expression-based replacements, we effectively consolidated different representations or spellings of the same facility type into a standardized form. This data cleaning approach significantly reduced the number of distinct choices in the Facility Type column from 432 in the original dataset to a more manageable 195 in the cleaned data. The reduction in choices ensured that the proportions of risk categories for each facility type will be accurately and meaningfully calculated. Filling the null values in the Facility Type column using the information provided by the DBA Name column enabled us to include 1758 entries in the analysis. This increase in data coverage leads to more reliable insights into how different facility types are associated with specific risk levels.

By using clustering methods and manually resolving discrepancies, we merged similar entries for establishments in the DBA Name column like MCDONALD'S, SUBWAY, DUNKIN DONUTS, and 7-ELEVEN, which previously have appeared in different variations due to misspellings or inconsistent formatting. The number of entries for MCDONALD'S increased from 475 in the original data to 1763 in the cleaned data. Having a clean and standardized dataset for the DBA Name column helps avoid misclassifications or errors that could affect the interpretation of inspection trends for McDonald's.

By converting the Inspection Date to the ISO date format 'yyyy-MM-dd', we ensure that all inspection dates are consistently formatted and can be easily handled and analyzed in a standardized manner. The creation of a new column, Inspection Year allowed us to extract the year from each inspection record, which simplifies the process of grouping and aggregating inspection results on a yearly basis.

**1.2 Data Cleaning Process with Python**

After doing the first-step cleaning, we solved several textual quality problems in the dataset. However, it is not enough for us to utilize the data for analyzing in our use cases. Focused on the main problems we need to solve, here are the steps we dealt with the dataset in Python using pandas and other libraries:

**1.2.1 Dealing with missing data**

There are multiple critical columns containing empty or null values in the records, which will affect the results in our use cases. For different columns, we used various strategies to minimize the loss of original data, and improve the quality of the whole dataset.

1. Remove null value records in License #, Risk, Address, Zip, Inspection Type, Facility Type.

We define License # as the primary key for the establishments, so the records with a null value need to be removed. Since the missing value will interfere with the functions to find the typical records in U1b, U1c, we also removed the null value records in Risk, Address, Zip, Inspection Type, and Facility Type. Notice that these are all qualitative data that usually require complex modeling to auto-fill, so we hold that these missing columns are justified to be removed for our use cases.

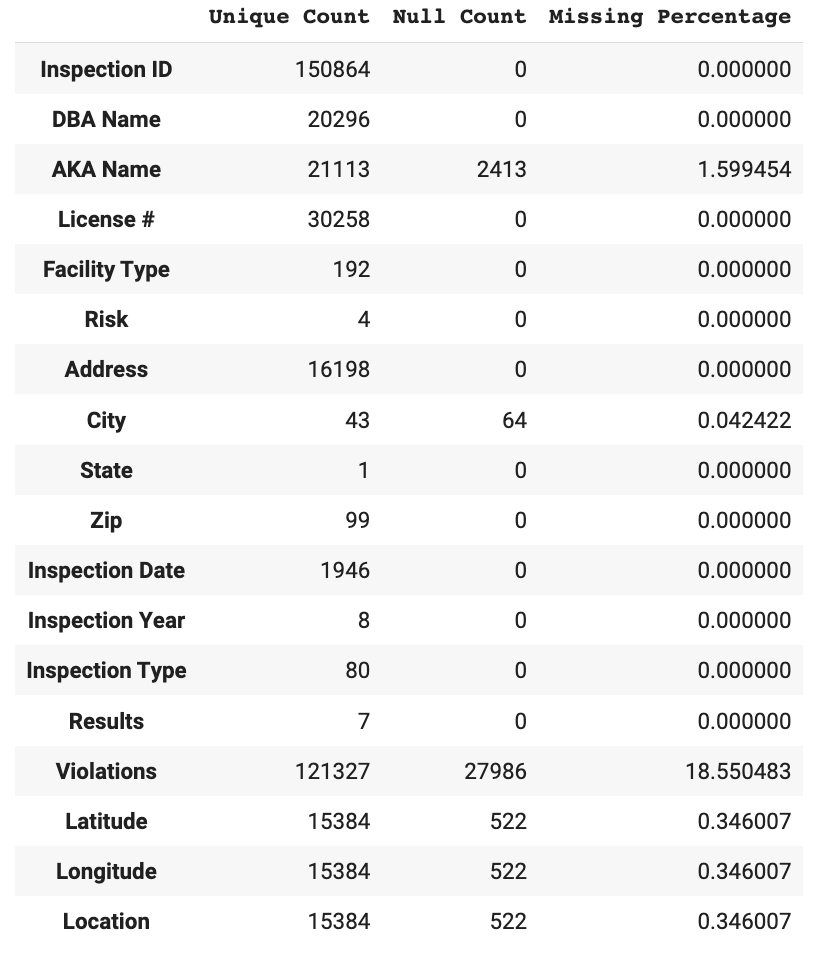


Figure 1: Unique and Null count after removing null from License #, Risk, Address, Zip, Inspection Type, Facility Type

1. For column City, we could fill the null based on the Zip column, by using the library ‘uszipcode’ to get the location of the place and find out the main city name.

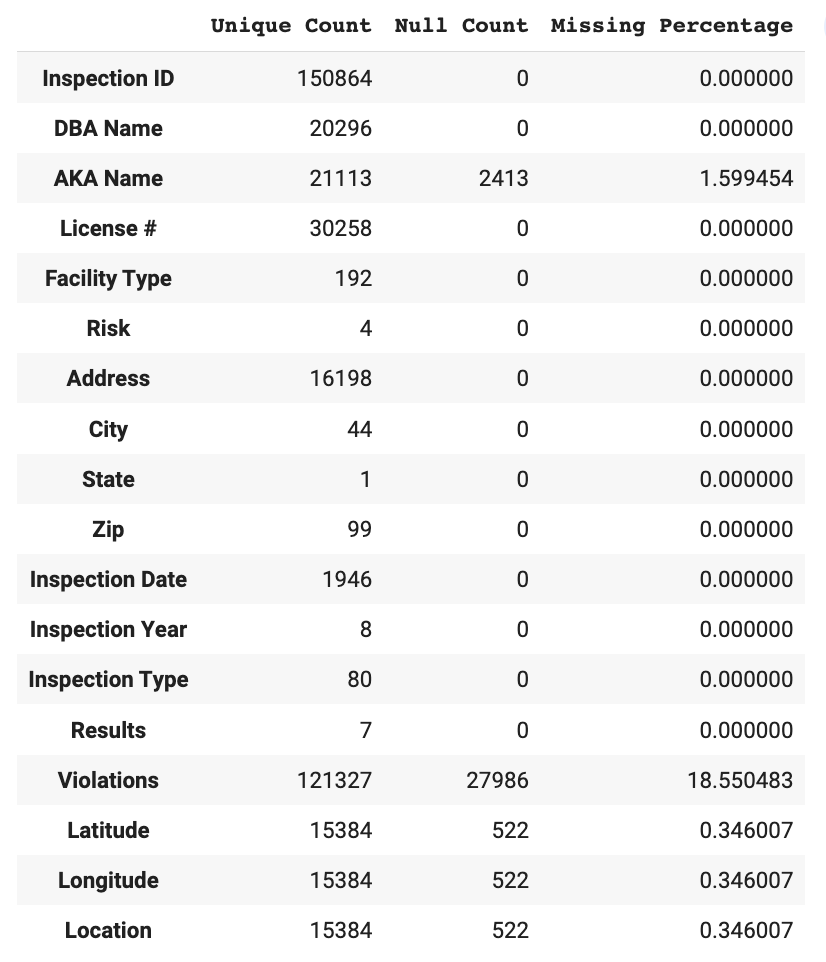


Figure 2: Unique and Null count after filling null for City

**1.2.2 Dealing with incorrect format**

In the original dataset we found that the type of ‘License #’ and ‘Zip’ are floats, and ‘Inspection Date’ is an object. To formalize the above columns, we changed their types to the following table. Correct data types ensure that the data is accurate and consistent. If the data is stored in the wrong data type, it may lead to incorrect calculations, aggregations, or analyses.

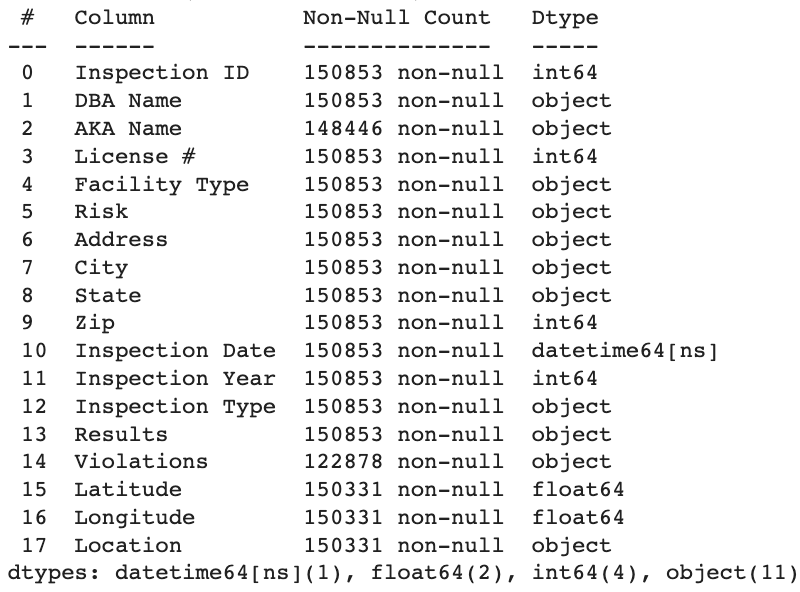


Figure 3: Data fields information after cleaning

**1.2.3 Dealing with data inconsistency**

From the data description document of Chicago Food Inspection, we found the criteria of recording the result:

Establishments receiving a ‘pass’ were found to have no critical or serious violations Establishments receiving a ‘pass with conditions’ were found to have critical or serious violations, but these were corrected during the inspection.

Establishments receiving a ‘fail’ were found to have critical or serious violations that were not correctable during the inspection.

In that case, ‘Results’ is determined by ‘Violations’, and we need to verify and correct mistakes in this column.

Basically we did the following two steps:

1. We replaced "null" Violations and "Pass with conditions" Results with "Pass".
2. We replaced “null” Violations and "Fail" Results with "Pass".

On the same ground, based on the documentation, Risk should only have 3 values. We need to remove all records with an "All''.

The rationale behind this is because we observed that these "All" in the Risk column are always tied with null violations + a non-pass inspection results such as "No Entry" or "Not Ready."

Yet we also discovered that some "All" are tied with null violations + "Fail" inspection results. While this combination itself is problematic enough that it deserves some cleaning operations described above, it probably shows that this "All" in the Risk column is not informative: the majority of "Fail" is associated with a 1-3 risk level in the dataset, instead of some vague ‘All’ risk levels.

Based on its relatively low proportion of presence in the Risk column, we treat this "All" as something obscure and should be deleted.

**1.2.4 Discovering Integrity Constraint Violations**

Integrity constraints are rules and conditions that data in the dataset must adhere to in order to maintain data accuracy, consistency, and reliability. It helps ensure the quality and accuracy of the data.

We defined the constraints as following:

1. The Inspection ID should be unique for each record.

We need to separate out the column Inspection ID to check if there exists these constraint violations.

First, we project this column and remove all duplicated records. Then, we check if Inspection is unique in the new dataset.

The rationale behind the projection is to make cleaning operations easier in the projected dataset, to test the cleaning effects more easily, and to later replace the uncleaned columns from the original dataset in a more organized and controllable way, provided that we indeed discovered some violations.

1. The License # should be unique for each establishment.

Like above, we need to separate out the columns License #, Address to check if these constraint violations exist.  
First, we project these two columns and remove all duplicated records. Then, we check if License # is unique in the new dataset.

During the cleaning and verifying process, we noticed that there are records with a license # equal to 0, which needs to be dropped.

The rationale behind this deletion is that either these establishments failed to get a license, or there were some administrative problems within the Chicago agencies regarding these establishments' assignments.

Based on the 7-year period for this dataset, we probably shouldn't include these establishments for our main use cases since this time span of having a weird license is not simply ignorable. Therefore, we hold that such deletion is reasonable.

1. The range of latitude and longitude should be limited.

It is general common sense that the ‘Latitude’ value should be between 0 and 90, and the ‘Longitude’ value should be in [-180,180].

1. Use DBA name to determine AKA name.

Having a consolidated view of data under the AKA name allows for more accurate and meaningful analysis and reporting.

**1.3 Data Cleaning Process with SQL**

We imported the dataset into SQL to use Integrity Constraint Violations checks on the columns to verify the data meet the constraints.

Then we compared the results from the dirty data, first-step cleaned data from OpenRefine and cleaned data from Python to see the raise of the data quality. This also serves as a final check on all previous cleaning operations to make sure they are bug-free without any syntax/logic errors in the previous code.

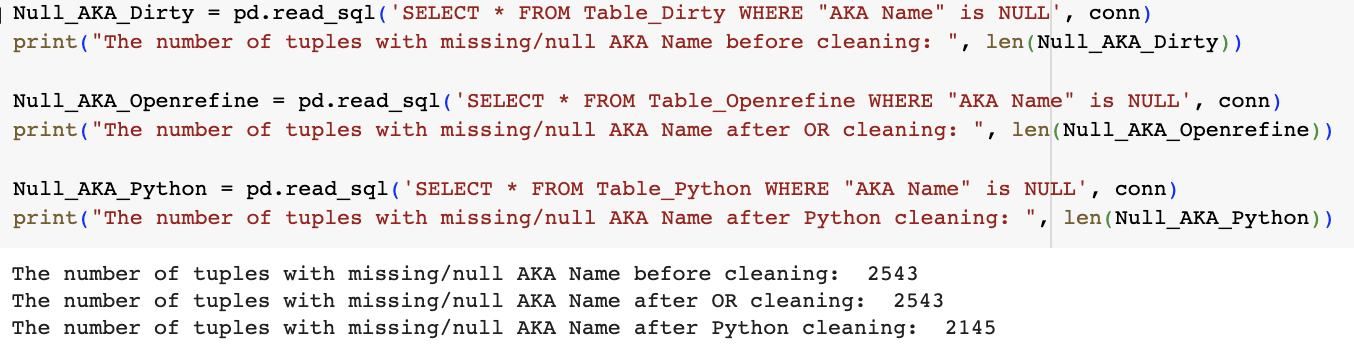


Figure 4: Data missing checks on AKA Name

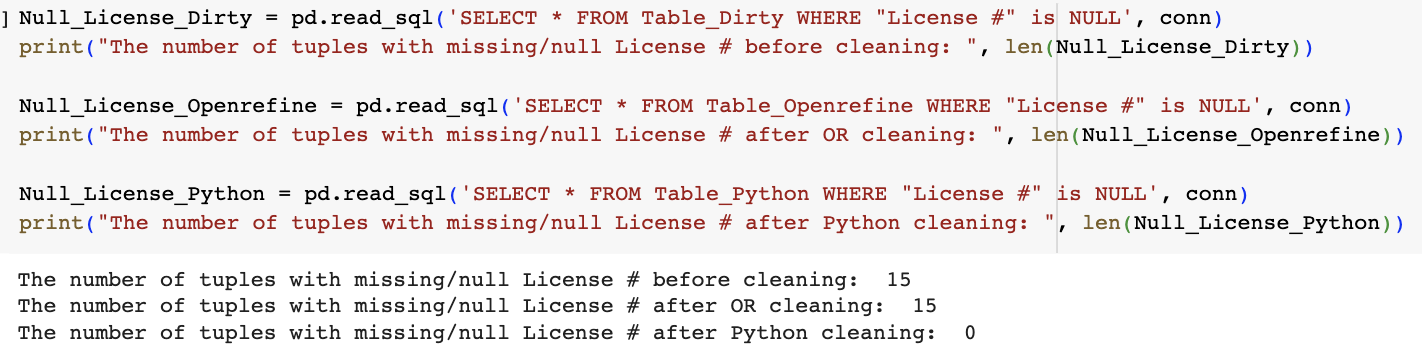


Figure 5: Data missing checks on License #

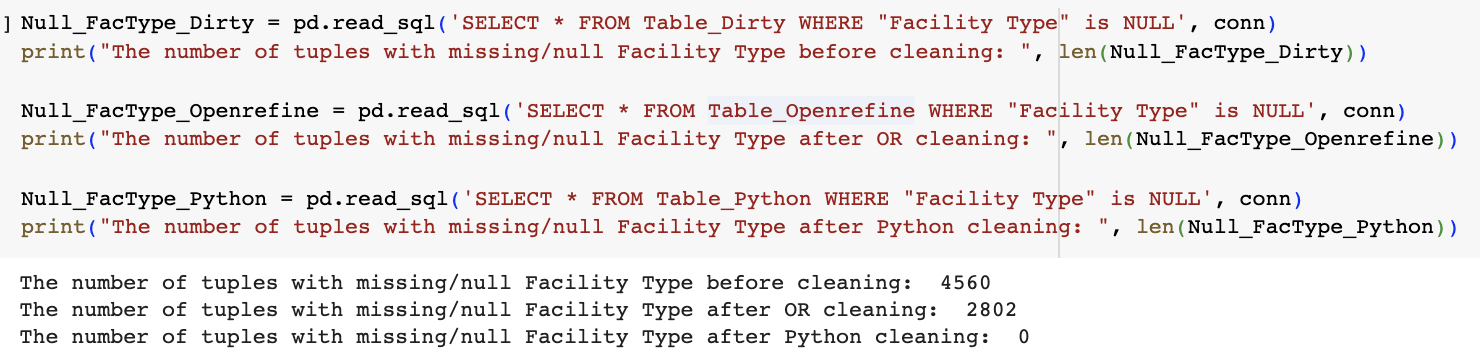


Figure 6: Data missing checks on Facility Type

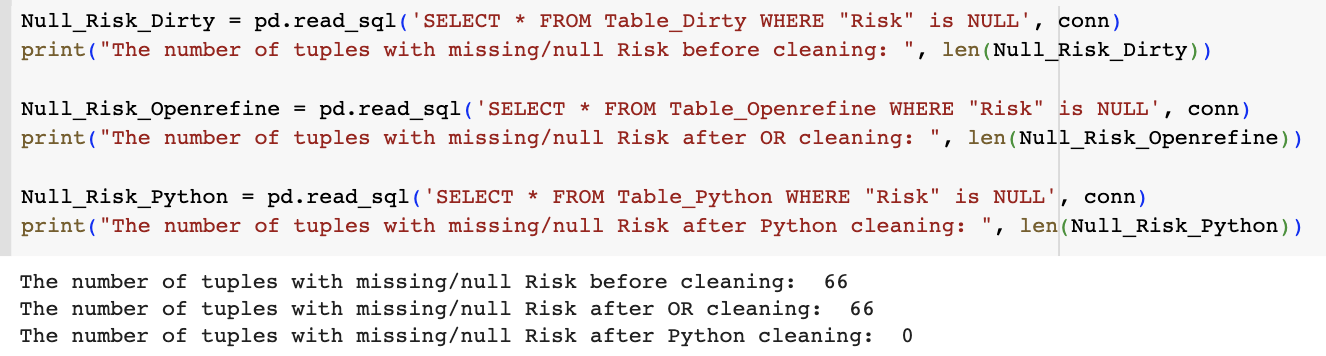


Figure 6: Data missing checks on Risk

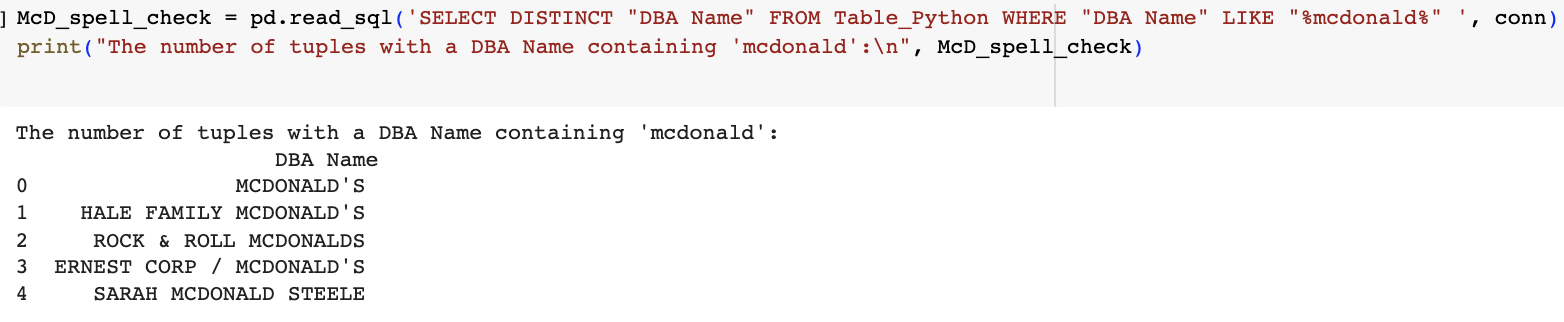


Figure 7: Field Spelling check on DBA Name “MCDONALD’S”

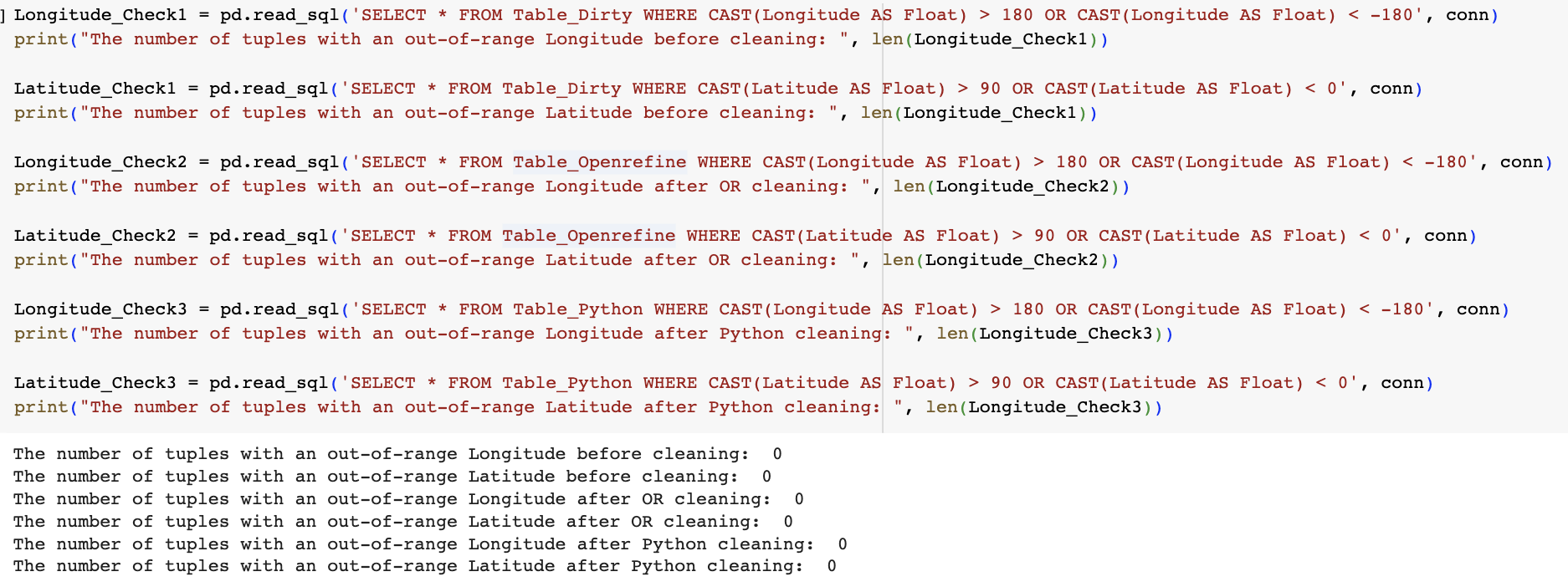


Figure 8: Range checks on Longitude and Latitude

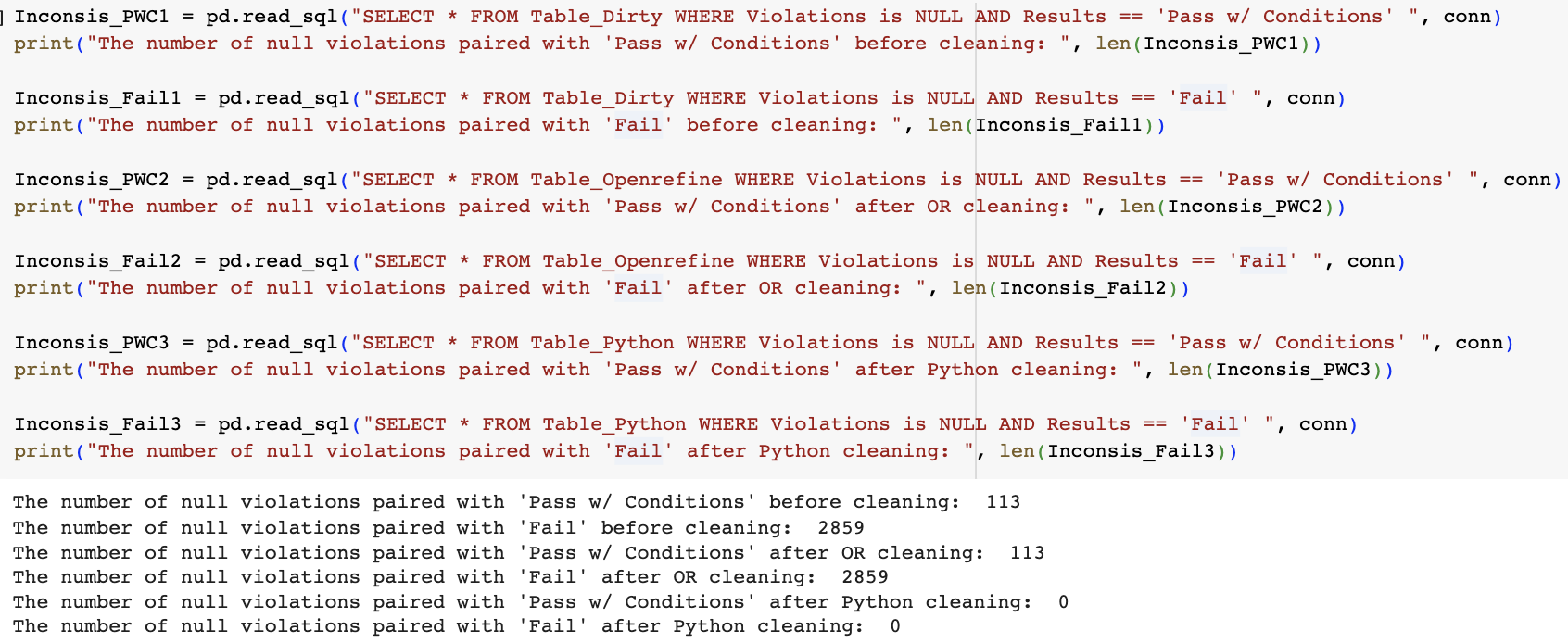


Figure 9: Pair Violations and Result to check the validity of data

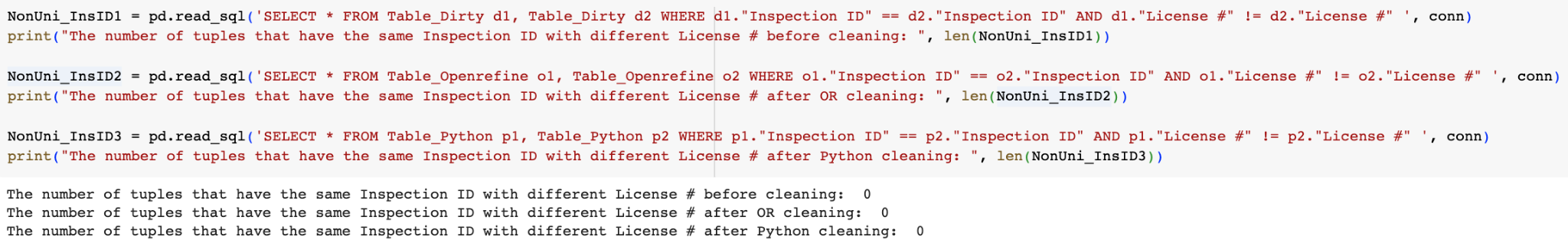


Figure 10: Integrity constraint violation check on Inspection ID

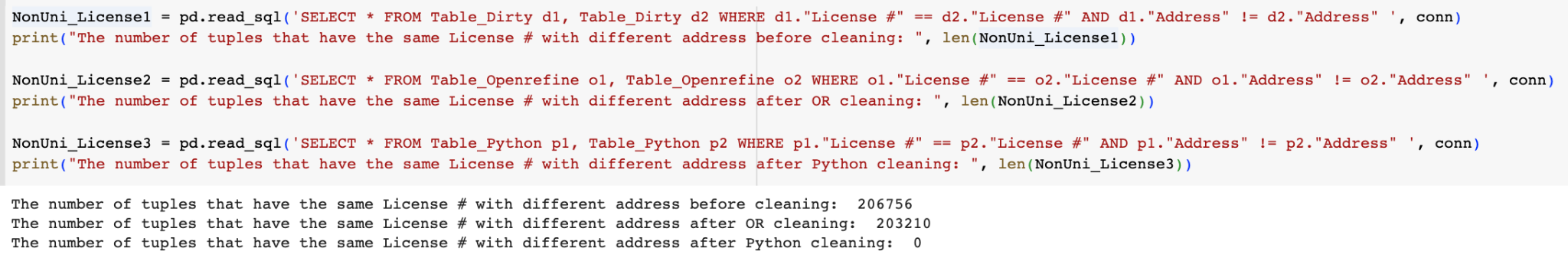


Figure 11: Integrity constraint violation check on License #

**1.4 Data Quality Improvements & Use Case Demonstration**

**1.4.1 Data Quality Improvements**

After the data cleaning process in OpenRefine, Python and SQL, the quality of the whole dataset has been improved in the following aspects:

1. **i. Spaces trimmed in columns,letter cases standardized, and similar inputs clustered.**

By removing spaces in textual data, standardizing formats, and aligning cases uniformly, we ensure data consistency and integrity. Additionally, the application of clustering techniques helped identify and eliminate duplicates, resulting in a more concise and coherent dataset. These data cleaning operations synergistically contributed to data quality enhancement, empowering researchers and analysts to extract valuable insights and make informed decisions with confidence, ultimately driving meaningful outcomes and facilitating progress in various domains.

We know that there are variations in capitalization and spacing for the DBA Name “McDonald’s.” As shown above, we check whether all similar entries have been merged into our set standard “MCDONALD'S” so that analysts won’t be dragged into unnecessary triviality and can focus on their desired analysis associated with the golden arches.

**ii. Missing data checked and dropped/filled according to different situations.**

We checked missing Data such as Longitude,Latitude, Facility Type, etc to make sure that there is no presence of null values. Here, we check the ones that can throw off our use cases’ results (e.g. Facility Type) and we ignore the ones that are either easy to fill in without real impacts (e.g. State) or simply have meaningful null values (e.g. Violations).

After dealing with missing values in the dataset, the null values are eliminated in all critically necessary columns, which will ensure the process of use cases afterwards.

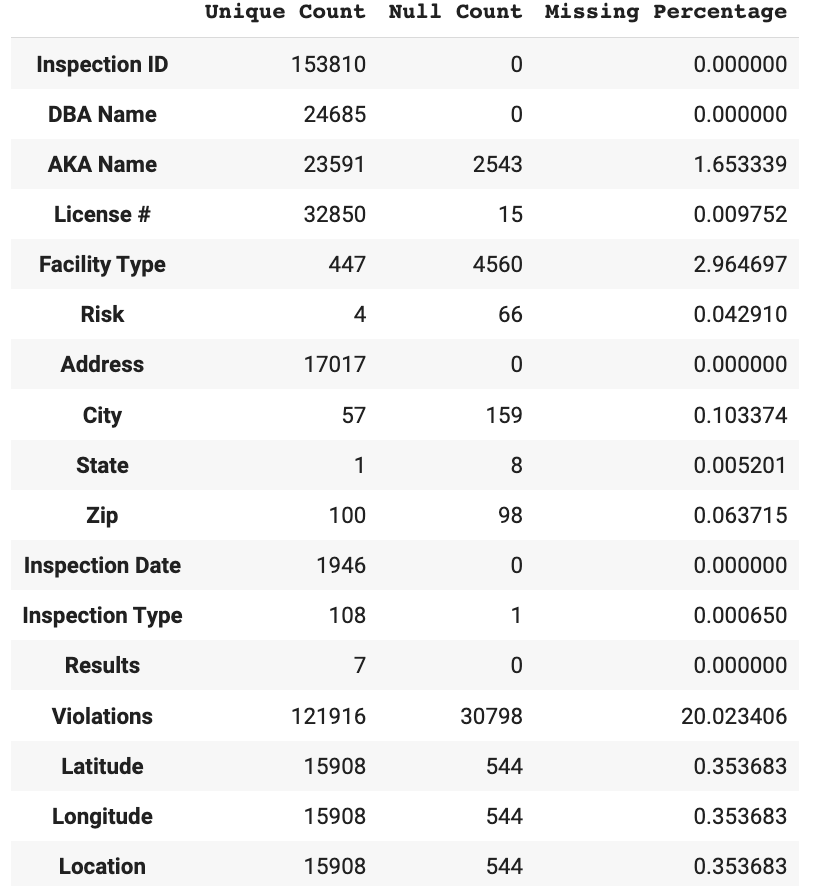


Figure 12: null value counts before and after dealing with missing values

1. **Data type checked and standardized**

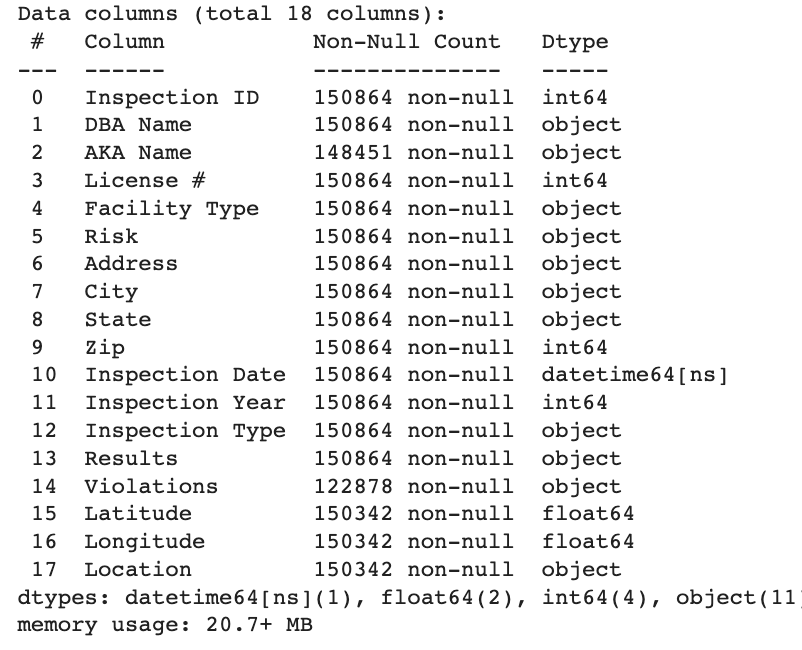
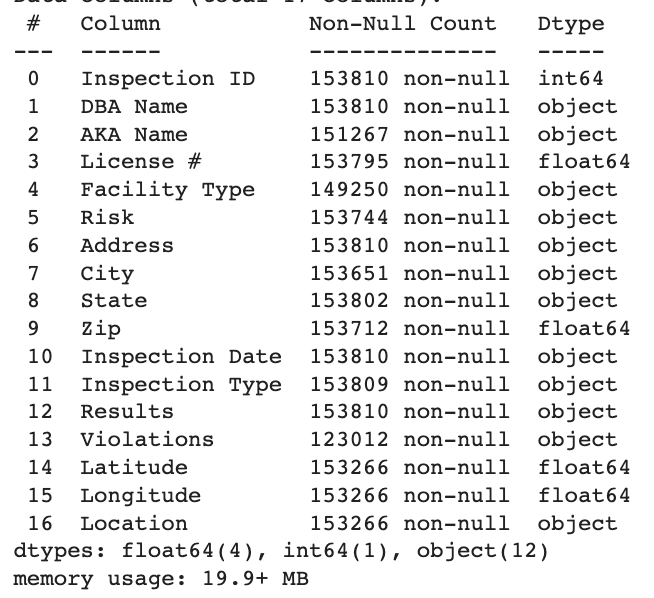
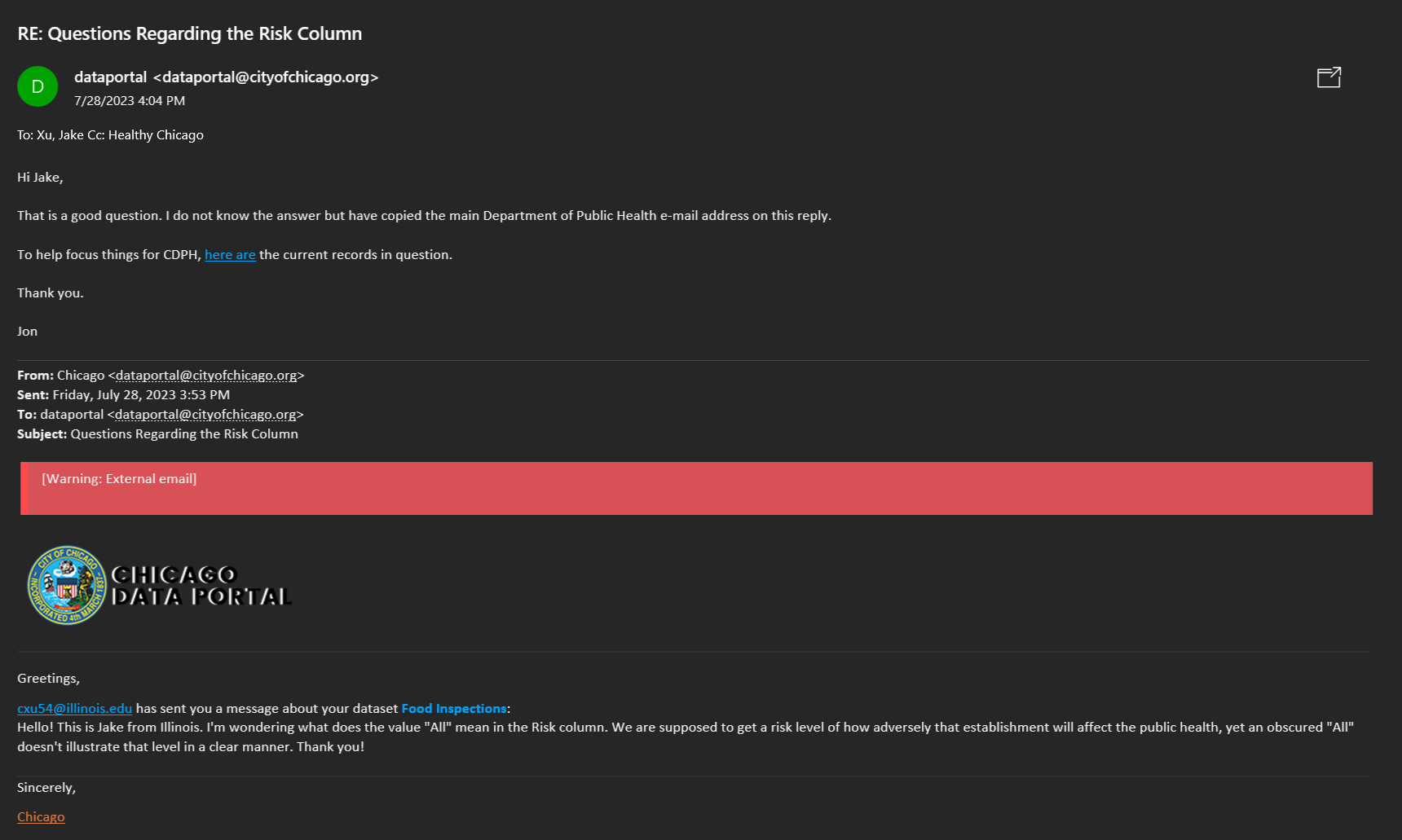


Figure 13: Data type before and after dealing with data formatting

1. **Data inconsistency checked and corrected**

Based on Figure 11 in the Phase I report, we know there exists a huge inconsistency between the Results and Violations columns, where some Violations do not indicate any type of violation yet the Results have “Pass with conditions.” This doesn’t make sense since there has been no violation whatsoever detected during the inspection.

Another important consistency check is via the official route: asking the data owners for confusing columns! For example, before making the assumption that we should delete ‘All’ in the ‘Risk’ column, we emailed City of Chicago Data Portal for its true meaning and make sure that we are not manipulating the dataset in a biased way (although they didn’t respond to date when we finished our report):



Likewise in real-world scenarios, we know that some inconsistency checks are conducted through non-technical ways but via direct communications between clients, so it is crucial to implement that practice before we make any major assumptions.

After applying the replacement of ‘Result’, the data is more reliable in the results of the inspection.

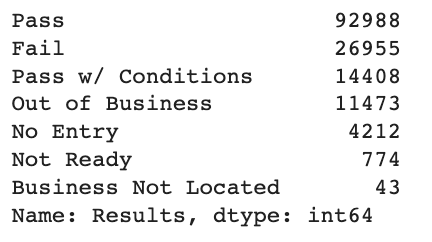
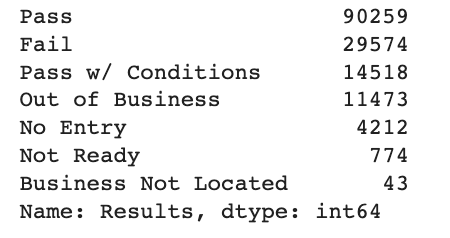


Figure 14: Value counts on ‘Results’ before and after cleaning

As stated, we removed the ‘All’ value in the ‘Risk’ column:’

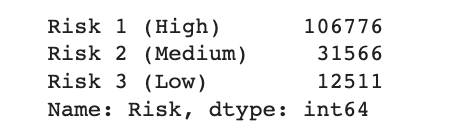
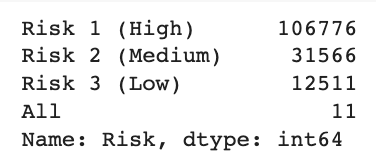
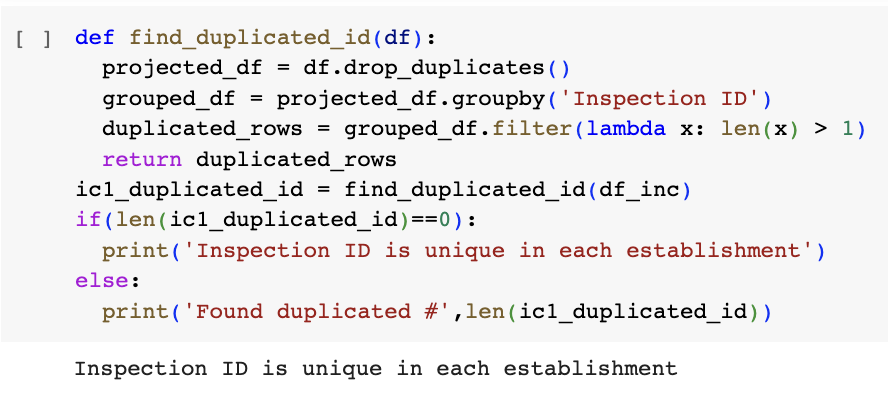


Figure 15: Value counts on ‘Risks’ before and after cleaning

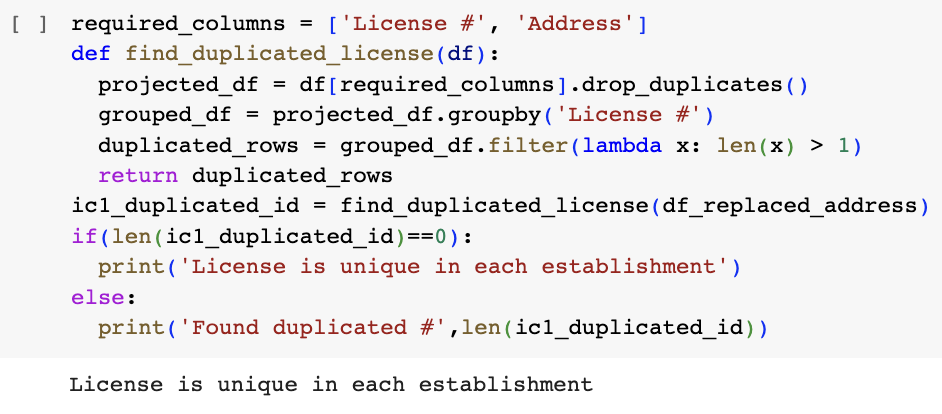
1. **Data verification through integrity constraint violations**

Based on the requirements both from our use cases and the official dataset documentation, we defined a couple of integrity constraint violations to check and improve the quality of data in certain columns. Moreover, these specifically defined functions also imply some ways for us to clean the data in Python.

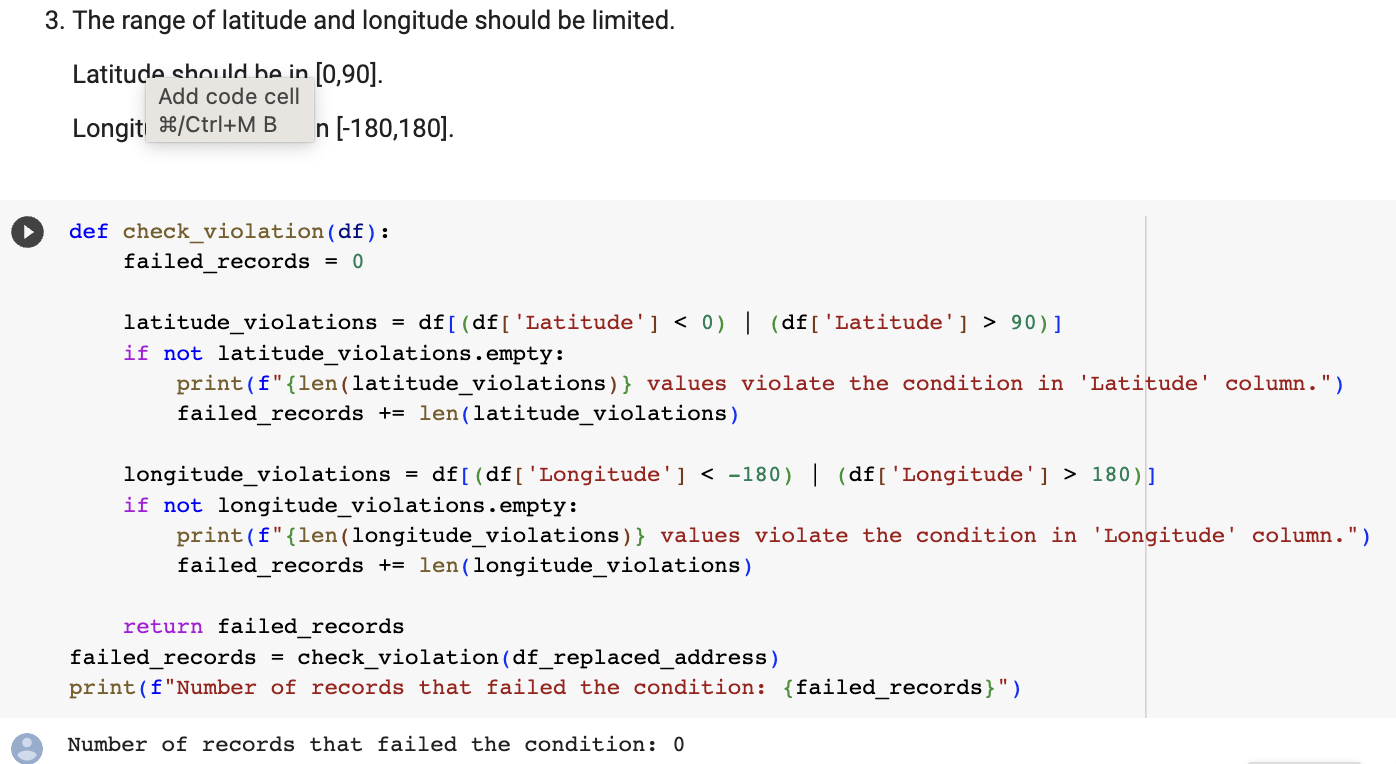
1. The Inspection ID should be unique for each record.

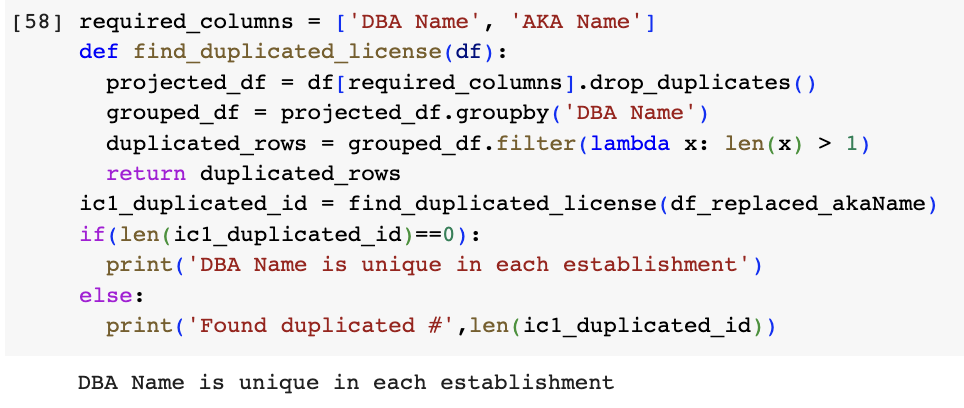


1. The License # should be unique for each establishment.



1. The range of latitude and longitude should be limited.



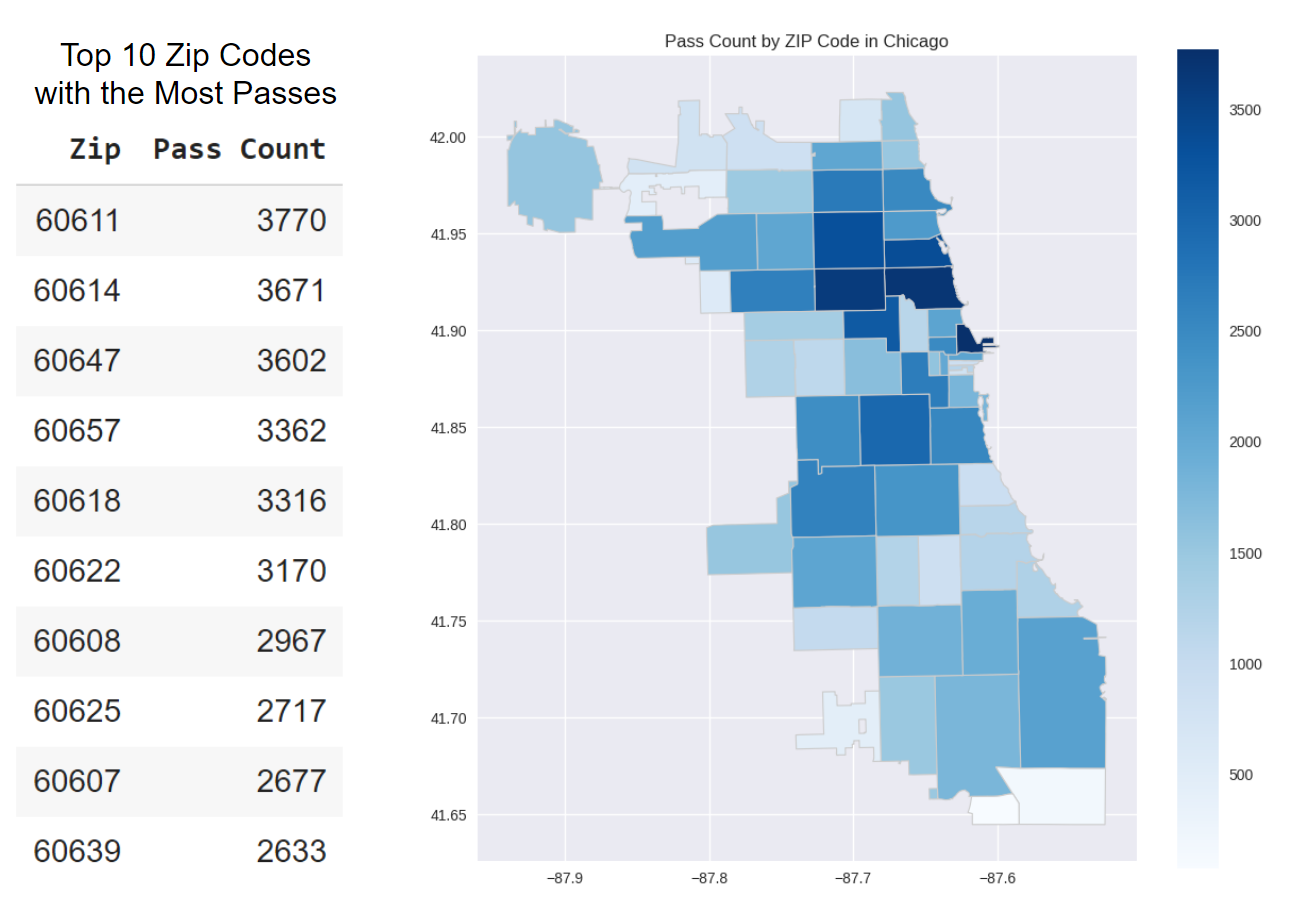
1. Use DBA name to determine AKA name.

By passing through these integrity constraint violations, we ensured that the data adheres to predefined rules, standards, and constraints, while also getting valuable insights for how to use Python packages and tools to conduct the actual cleaning (e.g. using dictionary objects, using projections, etc.) .

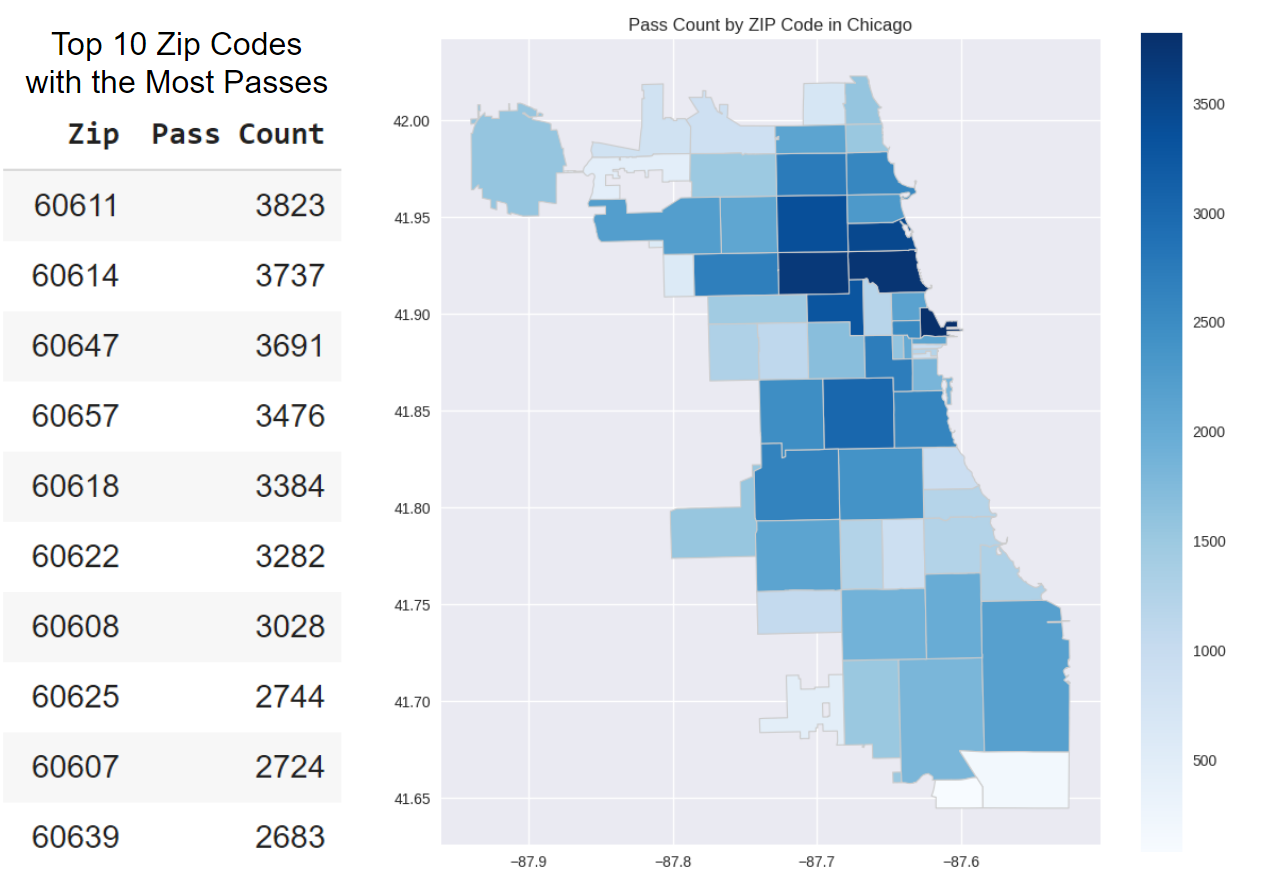
**1.4.2 Use Case Demonstration**

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

Before data cleaning:



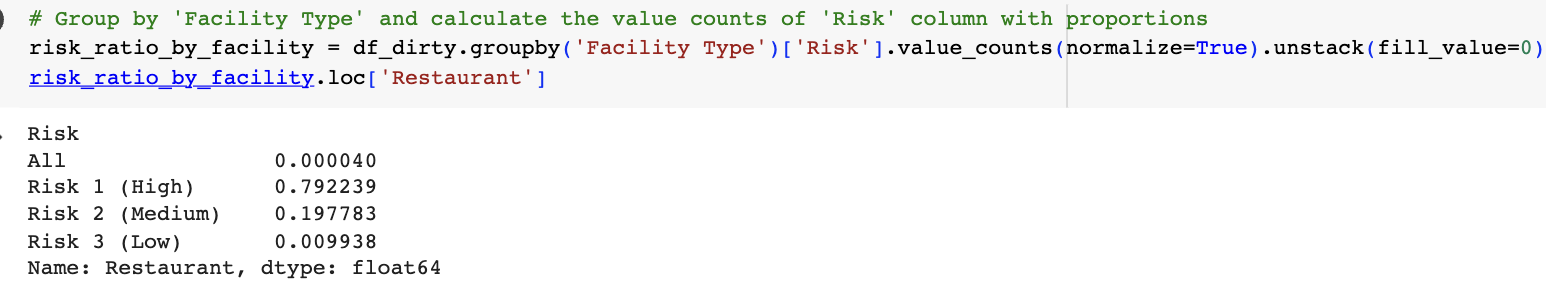
After data cleaning:



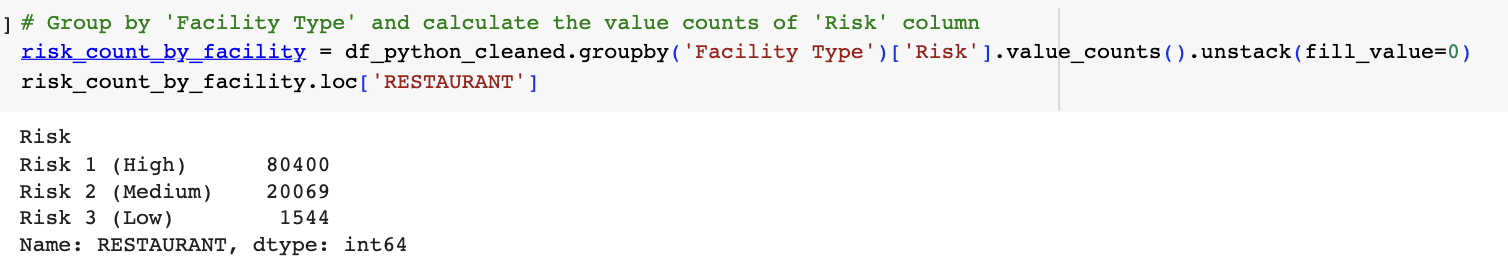
1. **Find out the proportions of Risk Category for Each Facility Type**

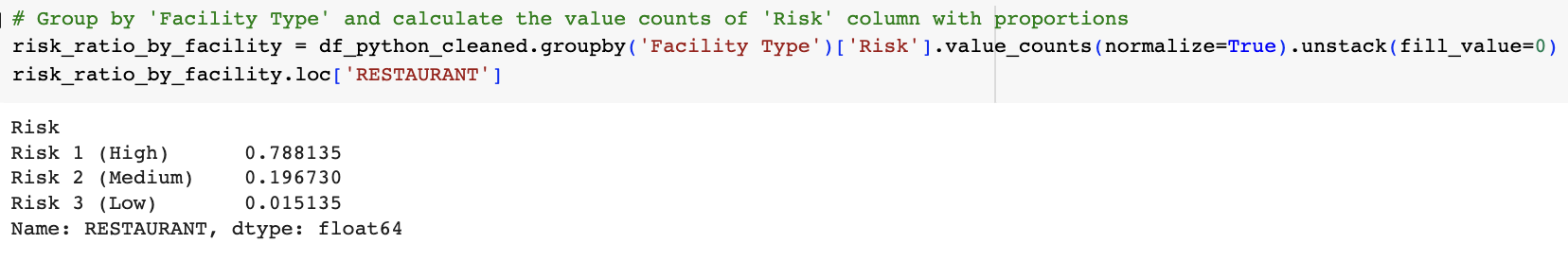
Before data cleaning:

****

****

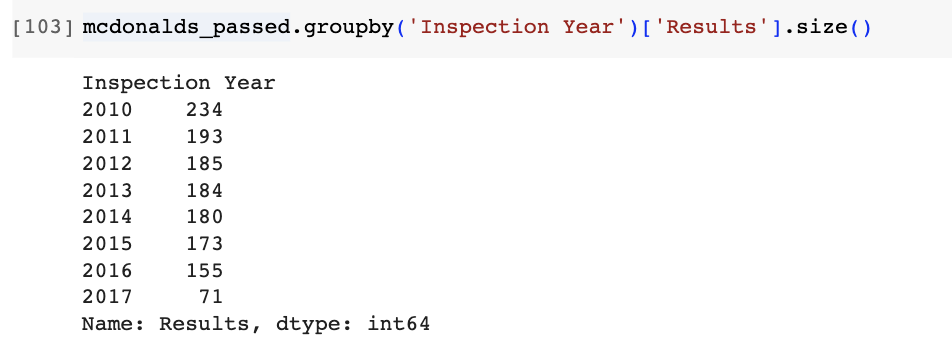
After data cleaning:



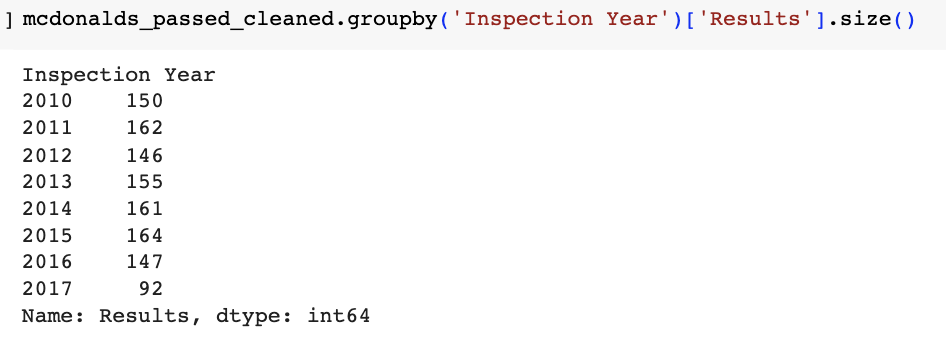


1. **Display Yearly Trends of Inspection Results of McDonald's.**

Before data cleaning:



After data cleaning:



**2. Document data quality changes**

**2.1 Quantify Changes using OpenRefine**

| **Column Cleaned** | **Process** | **# of cells modified** |
| --- | --- | --- |
| **DBA Name** | Collapse consecutive white spaces | 3,322 |
| Text transform to Uppercase | 10,443 |
| Merge similar entries by clustering with Key Collision and Fingerprint | 10,574 |
| Merge similar entries by clustering with Key Collision and Metaphone3 | 18,991 |
| Merge similar entries by clustering with Key Collision and Beider-Morse | 3,553 |
| Manually merge similar entries of “MCDONALD'S” (“MACDONALD” and “MCDONALD'S CORPORATION”) | 35 |
| Merge similar entries by clustering with Key Collision and Fingerprint | 39 |
| Merge similar entries by clustering with Key Collision and n-gram (size 2) Fingerprint | 924 |
| **AKA Name** | Collapse consecutive white spaces | 3,560 |
| Text transform to Uppercase | 9,585 |
| **Facility Type** | Collapse consecutive white spaces | 13 |
| Text transform to Uppercase | 146,729 |
| Merge similar entries by clustering with Key Collision and Metaphone3 | 6,881 |
| Merge similar entries by clustering with Key Collision and Daitch-Mokotoff | 828 |
| Use value.replace(/.\*(DAY\s\*CARE).\*/, 'DAYCARE') to merge similar entries of “DAYCARE”, including “1584-DAY CARE ABOVE 2 YEARS”, “DAYCARE (0-6 YEARS OLD), “DAYCARE (UNDER 2 YEARS)”, “DAYCARE 2 YRS TO 12 YRS”, “DAYCARE 2-6, UNDER 6”, “DAYCARE 6 WKS-5YRS”, “DAYCARE ABOVE AND UNDER 2 YEARS”, “DAYCARE NIGHT”. | 1,920 |
| Use value.replace(/.\*MOBIL.\*/, 'MOBILE FOOD') to merge similar entries of “MOBILE FOOD”, including “MOBIL FOOD 1315”, “MOBILE DESSERT CART”, “MOBILE FOOD DESSERTS VENDOR”, “MOBILE FOOD DISPENSER”, “MOBILE FOOD PREPARER”, “MOBILE FOOD TRUCK”, “MOBILE FROZEN DESSERT DISP/NON-MOTORIZED”, “MOBILE DESSERT VENDOR”, “MOBILE DESSERTS VENDOR”, “MOBILE FROZEN DESSERT DISPENSER\_NON MOTORIZED.”, “MOBILE FROZEN DESSERT VENDOR”. | 1301 |
| Use value.replace(/.\*BANQUET.\*/, 'BANQUET') to merge similar entries of “BANQUET”, including “BANQUET DINING”, “BANQUET FACILITY”, “BANQUET HALL”, “BANQUET ROOM”, “BANQUET ROOMS”, “BANQUET/KITCHEN”, “BANQUETS”, “BANQUETS/ROOM SERVICE”. | 200 |
| Use value.replace(/.\*GAS STATION.\*/, 'GAS STATION') to merge similar entries of “GAS STATATION”, including “GROCERY/GAS STATION”, “GROCERY/SERVICE GAS STATION”, “RESTAURANT/GAS STATION”, “GAS STATION /SUBWAY MINI MART.”, “GAS STATION /SUBWAY MINI MART”, “GAS STATION/RESTAURANT”. | 74 |
| Use value.replace(/.\*GROCERY.\*/, 'GROCERY STORE') to merge similar entries of “GROCERY STORE”, including “GROCERY”, “GROCERY AND BUTCHER”, “GROCERY STORE/BAKERY”, “GROCERY(SUSHI PREP)”, “GROCERY/BAKERY”, “GROCERY/BUTCHER”, “GROCERY/CAFE”, “GROCERY/DOLLAR STORE”, “GROCERY/DRUG STORE”, “GROCERY/LIQUOR”, “GROCERY/LIQUOR STORE”, “GROCERY/TAQUERIA”, “GROCERY/TAVERN”, “BAKERY/GROCERY”, “DELI/GROCERY STORE”, “DRUG STORE/GROCERY”, “DOLLAR & GROCERY STORE”, “DOLLAR & GROCERY STORE”, “DRUG STORE/GROCERY”, “DRUG/GROCERY STORE”. | 142 |
| Use value.replace(/.\*BAKERY.\*/, 'BAKERY') to merge similar entries of “BAKERY”, including “WHOLESALE BAKERY”, “”, “RESTAURANT/BAKERY”, “BAKERY/DELI”. | 21 |
| Use value.replace(/.\*LIQUOR.\*/, 'LIQUOR')to merge similar entries of “LIQUOR”, including “LIQUOR CONSUMPTION ON PREMISES.”, “LIQUOR STORE”, “LIQUORE STORE/BAR”, “RESTAURANT AND LIQUOR”. | 25 |
| Use value.replace(/.\*SHARED KITCHEN.\*/, 'SHARED KITCHEN') to merge similar entries of “SHARED KITCHEN'”, including “SHARED KITCHEN USER (LONG TREM)”, “SHARED KITCHEN USER (LONG TREM)”, “SHARED KITCHEN USER (SHORT TERM)”. | 216 |
| Use value.replace(/.\*CHURCH.\*/, 'CHURCH') to merge similar entries of “'CHURCH''”, including “CHURCH KITCHEN”, “CHURCH/AFTER SCHOOL PROGRAM”, “CHURCH/SPECIAL EVENT”, “CHURCH/SPECIAL EVENTS” | 42 |
| Use value.replace(/.\*HERBAL.\*/, 'HERBAL') to merge similar entries of “HERBAL'”, including “HERBAL DRINKS”, “HERBAL LIFE”, “HERBAL LIFE SHOP”, “HERBAL MEDICINE”, “HERBAL REMEDY”, “HERBAL STORE”, “HERBALIFE NUTRITION”, “HERBALIFE”. | 32 |
| Use value.replace(/.\*CONVENIEN.\*/, 'CONVENIENCE STORE') to merge similar entries of “CONVENIENCE STORE”, including “CONVENIENCE”, “CONVENIENCE/DRUG STORE”, “CONVENIENT STORE” | 65 |
| Use value.replace(/.\*KIOSK.\*/, 'KIOSK') to merge similar entries of “KIOSK”, including “O'HARE KIOSK”, “COFFEE KIOSK”, “NAVY PIER KIOSK”, “TEMPORARY KIOSK” | 66 |
| Use value.replace(/.\*CANDY.\*/, 'CANDY') to merge similar entries of “CANDY”, including “CANDY MAKER”, “CANDY SHOP”, “CANDY STORE”, “CANDY/GELATO” | 24 |
| Use value.replace(/.\*(JUICE|SMOOTHIE).\*/, 'JUICE BAR') to merge similar entries of “JUICE BAR”, including “HEALTH/ JUICE BAR”, “JUICE AND SALAD BAR”, “JUICE BAR” | 17 |
| Use value.replace(/.\*(COFFEE|CAFE).\*/, 'COFFEE/CAFE') to merge similar entries of “COFFEE/CAFE'”, including “ANIMAL SHELTER CAFE PERMIT”, “COFFEE ROASTER”, “COFFEE VENDING MACHINE”, “KIDS CAFE”, etc. | 144 |
| Use value.replace(/.\*(ICE CREAM|GELATO|FROZEN DESSERT).\*/, 'FROZEN DESSERT') to merge similar entries of “FROZEN DESSERT”, including “FROZEN DESSERT PUSHCARTS”, “ICE CREAM PARLOR”, “GELATO SHOP”, etc | 46 |
| Fill in the blank values with “RESTAURANT” using text filters to search for keywords in DBA Names with regular expression: RESTAURANT|PIZZA|SANDWICH|TACO|BBQ|BARBEQUE|BURGER|GYRO|GRILL|FAST FOOD|CHOP SUEY|WOK|MCDONALD'S|SUBWAY|STARBUCKS KING|DUNKIN DONUTS|HAROLD'S CHICKEN SHACK|KENTUCKY FRIED CHICKEN|QUIZNO'S SUB|TOKYO LUNCH|COLD STONE|COLD STONE|EUREST DINING SERVICES|GIORDANO'S|GOLD COAST DOGS|J & J FISH|PANDA EXPRESS|SHARKS FISH & CHICKEN|SOFITEL CHICAGO WATER TOWER | 903 |
| Fill in the blank values with “GROCERY STORE” using text filters to search for keywords in DBA Names with regular expression: GROCERY|MART|MARKET|7-ELEVEN|FAMILY DOLLAR|WALGREENS|WHITE HEN PANTRY|CVS|SAVE-A-LOT|FOOD STORE | 509 |
| Fill in the blank values with “DAYCARE” using text filters to search for keywords in DBA Names with regular expression: (DAY.\*CARE)|(PRE.\*SCHOOL) | 25 |
| Fill in the blank values with “BAKERY” using text filters to search for keywords in DBA Names with regular expression: BAKERY | 59 |
| Fill in the blank values with “COFFEE/CAFE” using text filters to search for keywords in DBA Names with regular expression: COFFEE|CAFE | 265 |
| **Address** | Trim leading and trailing spaces | 153,366 |
| Collapse consecutive white spaces | 618 |
| Text transform to Uppercase | 9488 |
| Merge similar entries by clustering with Key Collision and Fingerprint | 344 |
| Merge similar entries by clustering with Key Collision and n-gram (size 3) Fingerprint | 211 |
| **City** | Text transform to Uppercase | 348 |
| Merge similar entries by clustering with Key Collision and Daitch-Mokotoff | 153,490 |
| **State** | Manually confirm and fill the blank cells with “IL” | 8 |
| **Inspection Type** | Text transform to Titlecase | 29,926 |
| Merge similar entries by clustering with Key Collision and Metaphone3 | 90,126 |
| **Violations** | Trim leading and trailing spaces | 18,580 |
| Collapse consecutive white spaces | 84,606 |
| **Inspection Date** | toString(toDate(value),'yyyy-MM-dd') | 153,810 |
| **Inspection Year** | Add a column based on Inspection Date value.split("-")[0] | 153,810 |

**2.2 Quantify Changes using Python**

| **Column Cleaned** | **Process** | **# of cells modified** |
| --- | --- | --- |
| **AKA Name** | Value replaced & filled based on DBA Name | 2826 |
| **License #** | Remove null value | 15 |
| Change data type | 150461 |
| **Facility Type** | Remove null value | 2802 |
| **Risk** | Remove null value | 66 |
| Remove value = ‘all’ | 11 |
| **Address** | Remove null value | 3 |
| Value replaced based on License # | 814 |
| **Zip** | Remove null value | 98 |
| Change data type | 150461 |
| **City** | Null value filled based on Zip | 159 |
| **Results** | Replace value based on Violations:  df['Violations'].isnull()) & (df['Results'] == 'Pass w/ Conditions'  df['Violations'].isnull()) & (df['Results'] == 'Fail') | 2031 |
| **Inspection Date** | Change data type | 150461 |
| **Inspection Type** | Remove null value | 1 |

**3. Create a workflow model**

**3.1 Outer data cleaning workflow**

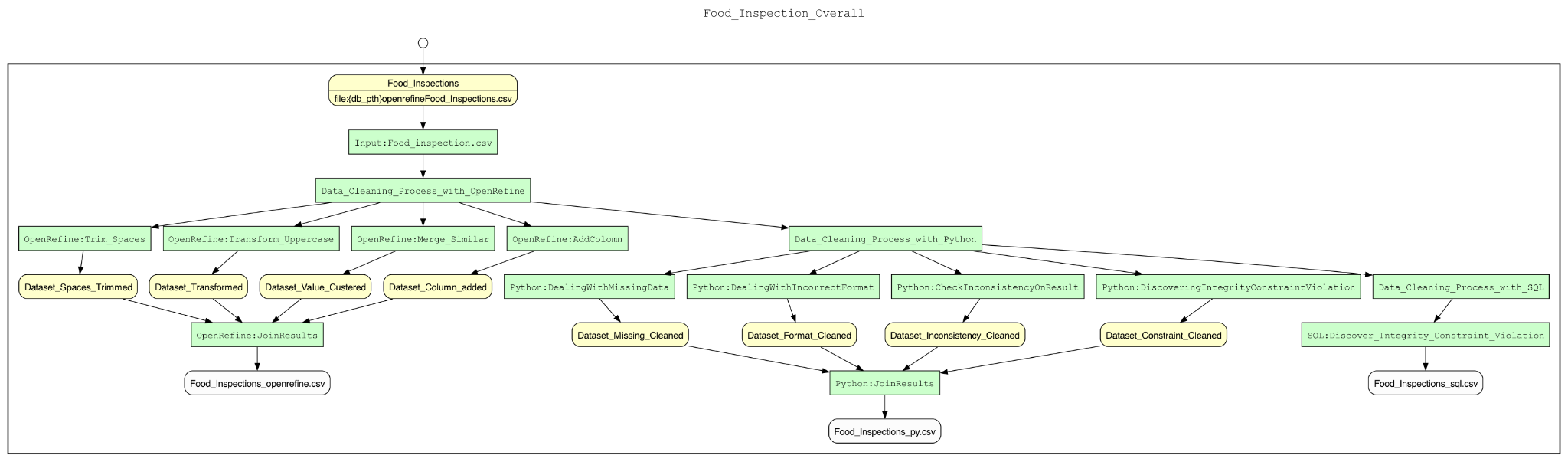
****

Figure 16: YesWorkFlow Diagram for the whole project

**3.2 Inner data cleaning workflow using OpenRefine**

Figure 17 presents the YesWorkFlow model for OpenRefine, which outlines each stage of our data cleaning procedure on targeted columns. The YesWorkFlow diagram is derived from OpenRefine's Json history file using the OR2YWTool.

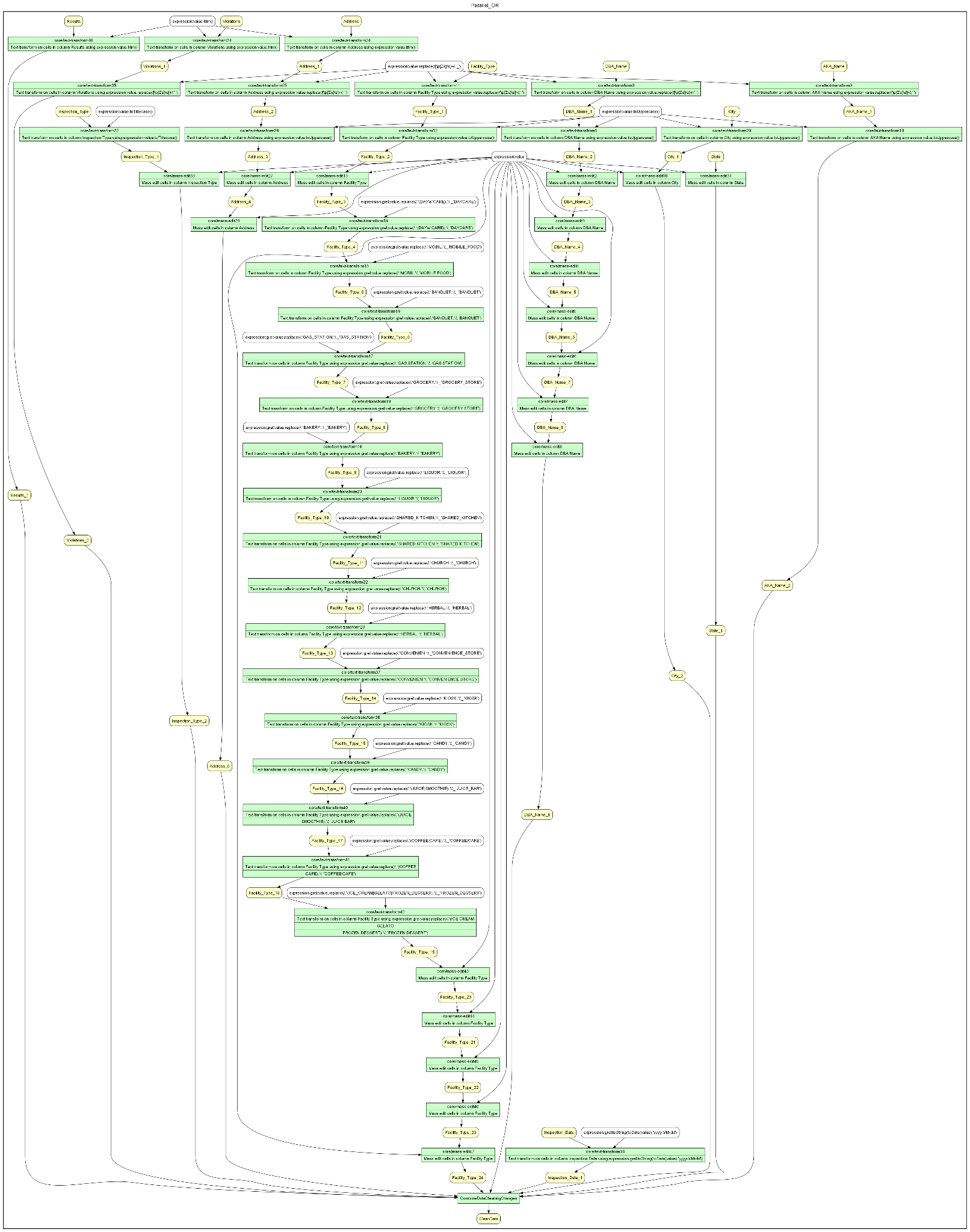
****

Figure 17: YesWorkFlow Diagram for OpenRefine Cleaning History

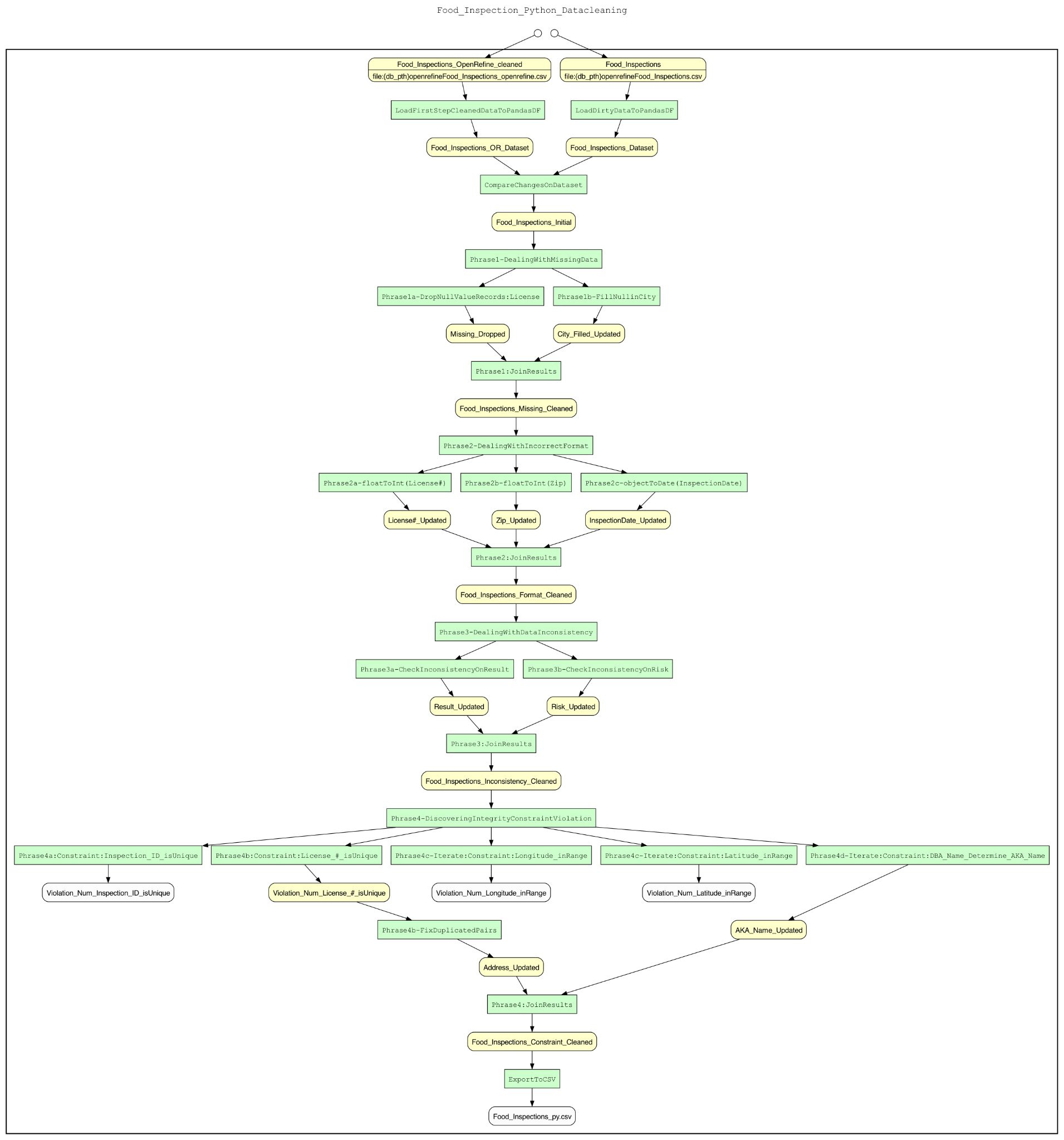


Figure 18: YesWorkFlow Diagram for Python Cleaning History

**4. Conclusions & Summary**

This project presented an invaluable opportunity to gain a comprehensive understanding of the intricacies involved in data cleaning. While some preliminary analysis could be performed without modification, a more profound exploration necessitated meticulous cleaning to ensure consistency and adherence to constraints across all columns. Addressing outliers and other salient data points was imperative to enhance data integrity. In our endeavor, we adopted a multi-faceted approach, leveraging the combined prowess of OpenRefine, Python, and SQL.

Xinyu contributed to the part regarding OpenRefine, which served as a foundational tool laying the groundwork for essential cleaning procedures. The sequential cleansing of individual columns formed a basis for subsequent processing with SQL. However, it became evident that OpenRefine's capabilities alone were insufficient, prompting us to supplement the process with Python for further refinement.

Xiaojing contributed most of the Python and related tools, which proved instrumental in identifying specific outliers within columns and discerning distinct characteristics within entries. For instance, it enabled us to detect instances where a single license number was associated with multiple addresses. Complementing this, the application of SQL queries facilitated the identification of data constraints and the assurance of data distinctiveness.

Jake contributed to the SQL tool and helped with writing Python cleaning rationales, which played instrumental roles in ensuring data integrity and enforcing constraints during the data cleaning and analysis phases. SQL queries also serve as a “double check” to detect any potentially insufficient cleaning in the previous steps. Leveraging the structured querying capabilities of SQL, we effectively established and implemented a range of Integrity Constraint Violations (ICVs) on the dataset.

Another important feature for this group project is the frequent communications between members regarding functionalities and integrity standards. These “double-check” works also make sure that the previous cleaning did work as a result and bug-free for the respective codes. Various communications and compromises are typical in real-world projects, so it is critical to have some experience here.

Consequently, this project reinforced the significance of employing a diverse toolkit for optimal data cleaning outcomes. By judiciously selecting the dataset and delineating the targeted use cases, we meticulously outlined the requisite steps to culminate in a dataset that epitomizes accuracy, cleanliness, and conciseness—qualities vital for illuminating our use case with user experience.

**5. Supplements**

The 3 versions of datasets can be found here:

Origin Data:

<https://drive.google.com/file/d/1W40Ceow3UTD0xtrOv4lHalKR6QKyILkA/view?usp=drive_link>

OpenRefine Cleaned Data:

<https://drive.google.com/file/d/1Dl7VTTlyNDzfBTe49c896PuVauzW08hL/view?usp=drive_link>

Python Cleaned Data:

<https://drive.google.com/file/d/1amuxel87fr43wUb3r_6-YIqdLARGgSS4/view?usp=drive_link>

We also created a Jupyter Notebook for all codes related to cleaning operations. We believe that collecting all the works on one sheet keeps track of our project and plays important roles in group discussions/collaborations. It can be found here: <https://drive.google.com/file/d/1KQHTXny7OhDnnzIVFoHNcxpq_vibS5xy/view?usp=drive_link>