INTRODUCTION

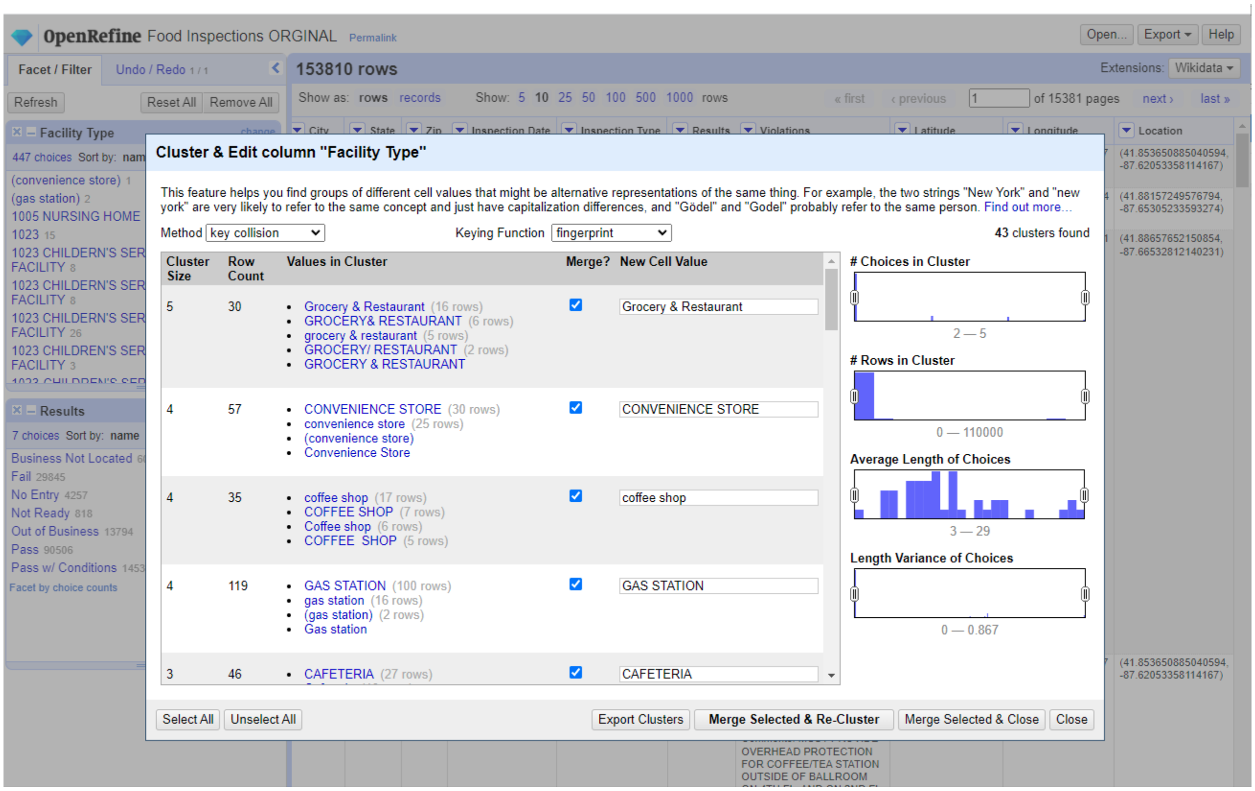
The main use case for our data cleaning project is for the Chicago launch of a fictional app called Helph that combines Yelp-like crowdsourcing reviews with food inspection information. It will allow users to look up recent health code violations. The team of Anthony Petrotte, Issac Aruldas and Paul Bromen cleaned the “Food Inspections.CSV” dataset to provide the raw material to accomplish this goal.

DATING CLEANING PERFORMED

Following our phase 1 plan we set, we took the data into **OpenRefine for general cleaning** and pruning unnecessary data items. These items included entries that were not restaurants, didn’t reflect actual inspections, or were not in Chicago.

First thing we did was eliminate the “Archive” column from the dataset. This showed that all of this data came from ‘Food_Inspections.csv.zip and wasn’t relevant to our project. Eliminating this was not relevant to our case, but makes the resulting data easier to work with.

Second, following our plan we analyzed the “Facility Type” Column to prune away facility types that weren’t relevant to our U1 case. Namely, we eliminated all institutions that couldn’t be determined to be restaurants. Using a text facet in OpenRefine revealed 447 types with many near overlaps mudding the clarity. So we clustered the facet to merge categorization errors. On the first cut we merged 43 clusters using the key collision method and fingerprint keying function.



Cluster & Edit column "Facility Type"

This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same person. Find out more…

Method: key collision

Keying Function: fingerprint

43 clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
5	30	Grocery & Restaurant (16 rows) GROCERY& RESTAURANT (6 rows) grocery & restaurant (5 rows) GROCERY/ RESTAURANT (2 rows) GROCERY & RESTAURANT	☑	Grocery & Restaurant
4	57	CONVENIENCE STORE (30 rows) convenience store (25 rows) (convenience store) Convenience Store	☑	CONVENIENCE STORE
4	35	coffee shop (17 rows) COFFEE SHOP (7 rows) Coffee shop (6 rows) COFFEE SHOP (5 rows)	☑	coffee shop
4	119	GAS STATION (100 rows) gas station (16 rows) (gas station) (2 rows) Gas station	☑	GAS STATION
3	46	CAFETERIA (27 rows)	☑	CAFETERIA

These mostly had different cases or spare characters, but were clearly intended to be the same category. Reclustering we found only 1 additional cluster where a two word category had interposed elements. Then we repeated clustering for Ngram-fingerprint and merging 23 additional clusters. Experimenting with NGRAM size we quickly learned that going above 3 resulted in false positives clusters. Going up to 10, collected nearly 95% of our entries! Switching to the metaphone3 Keying Function yielded 15 more potential clusters. 14 of these were deemed similar enough. The Dutch Mokotoff method yielded 20 clusters, but this time we had to use our human intuition on each one to investigate whether we were losing information that we wanted in order to consolidate the category numbers. If there was restaurant type information we generally left them unmerged as that could be handy for a future data scientist at our company, while we applied a heavy hand to categories that were certain to be tossed out in the final prune. Finally we came to the Berider-Morse key function of the key collision method. After much churning of the computer no clusters were found.

Moving to the nearest neighbor method I was sure that we would have exhausted all the possible clusters, but even on the lowest level we found an obvious cluster. Playing around with the parameters we hit on several potential clusters. Some of these were not intuitive and needed further investigation (Googling). For example Cafeteria was matched with Paleteria. Maybe that was an entry error? However there were 14 entries and it was a very specific error to occur so many times in our dataset. Looking up the definition, Paleteria turned out to mean a Latin American style popsicle shop. Combining these items would have reduced the richness of our data.

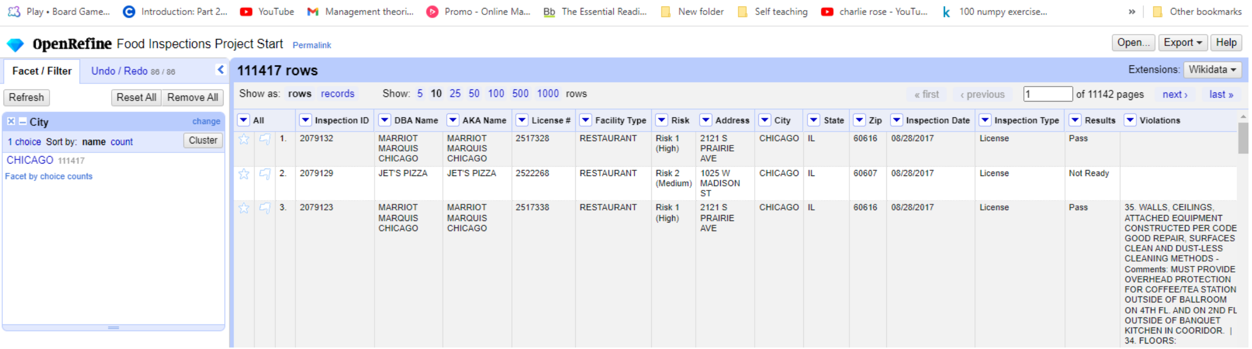
After all of this clustering we had reduced categories from 447 to 262. Still a lot to comb through but manageable to handle manually. This was a time consuming process that perhaps could have been handled programmatically, but would likely have required a rather advanced program to determine the merits of including a mobile food Preparer, a Mobile Frozen Desserts Vendor or a Commissary for Soft Serve Ice Cream Trucks. In our case the first to fit our use case and that last one doesn’t. Why? Because the first two serve customers while the third is a place for other vendors to prepare treats and unfortunately we have no way to tie it back to individual commissary members with the current data. The most head scratching category found was something labeled as “Watermelon House" facility type. If this were a movie this part of the project would be time stop motion montage, in case you are wondering what music, Janet Jackson’s 1997 classic “Go Deep” was playing on repeat for the 40 minutes of this process, like a linear regression it seemed fitting. We also made the whole column uppercase, but most of these redundancies had been caught by clustering so we only removed 1 category

The biggest question of what to do during this phase was how to handle the 4560 blanks. It would be too much time to go in and label each one, unless there was an intern handy. We opted to leave them in the dataset as a cursory glance revealed several restaurants without a facility type. In the real world we would use another dataset to try to match these entries to a category using Name or Location as a foreign key to merge the table.

At the end of this process we had 118 facility types and 111,532 entries representing a 74% reduction in restaurant types and eliminating 27% of the data items that were irrelevant to our primary use case.

As part of general cleanup we trimmed whitespace from each column. Additionally, we removed the outer parentheses from the “Location” column. Will this matter? Not for our U1 use case, but we figured it would be nice for the next data scientist down the line when they are using this data.

Then all entries without “Chicago” listed as the city column. As before we used a text facet and clustered the entries. Thankfully there were only 39 choices this time, and we were able to cluster that down to 31 choices. We manually eliminated all the non-Chicago choices. For the 100 blank entries we were able to determine that they were in Chicago based on their zip code. After this process we had 111.417 rows of restaurants located in Chicago.



OpenRefine Food Inspections Project Start

City

1 choice Sort by: name count

CHICAGO 111417

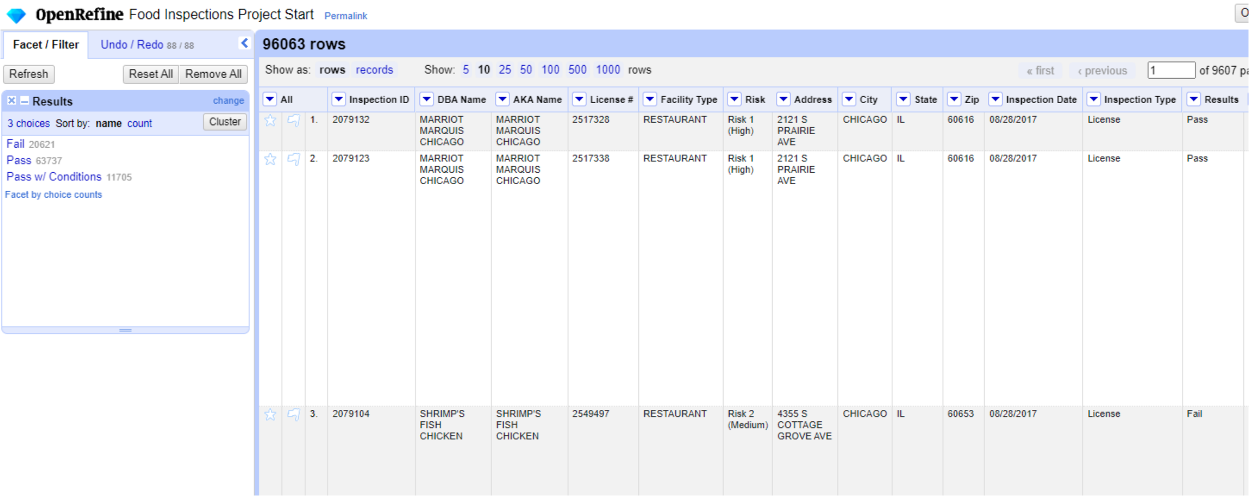
111417 rows

Show as: rows records   Show: 5 10 25 50 100 500 1000 rows

1 of 11142 pages

All			Inspection ID	DBA Name	AKA Name	License #	Facility Type	Risk	Address	City	State	Zip	Inspection Date	Inspection Type	Results	Violations
☆		1.	2079132	MARRIOT MARQUIS CHICAGO	MARRIOT MARQUIS CHICAGO	2517328	RESTAURANT	Risk 1 (High)	2121 S PRAIRIE AVE	CHICAGO	IL	60616	08/28/2017	License	Pass	
☆		2.	2079129	JET'S PIZZA	JET'S PIZZA	2522268	RESTAURANT	Risk 2 (Medium)	1025 W MADISON ST	CHICAGO	IL	60607	08/28/2017	License	Not Ready	
☆		3.	2079123	MARRIOT MARQUIS CHICAGO	MARRIOT MARQUIS CHICAGO	2517338	RESTAURANT	Risk 1 (High)	2121 S PRAIRIE AVE	CHICAGO	IL	60616	08/28/2017	License	Pass	35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Comments: MUST PROVIDE OVERHEAD PROTECTION FOR COFFEE/TEA STATION OUTSIDE OF BALLROOM ON 4TH FL. AND ON 2ND FL OUTSIDE OF BANQUET KITCHEN IN COORIDOR. | 34. FLOORS:

The last step of work in OpenRefine was using the “Results” Column to ensure that an inspection actually took place at the restaurant. Once again using a Text facet, we were able to get category counts there were only 7. We eliminated all inspections with “Business Not Located”, “No Entry” and “Not Ready”. We also removed “Out of Business,” but in real life would keep these in a separate CSV of these as this data would be important to our user experience (We wouldn’t want to clutter our UI by showing them irrelevant data on restaurants that were out of business). After eliminating these we are left with only “Fail”, “Pass”, and “Pass w/ Conditions”. These entries total 96,063 and cut the data that needs working through on the next steps in half compared to the original dirty data without losing richness or inspections that we would be interested in.



OpenRefine Food Inspections Project Start

Results

3 choices Sort by: name count

Fail 20621
Pass 63737
Pass w/ Conditions 11705

96063 rows

Show as: rows records   Show: 5 10 25 50 100 500 1000 rows

1 of 9607 pa

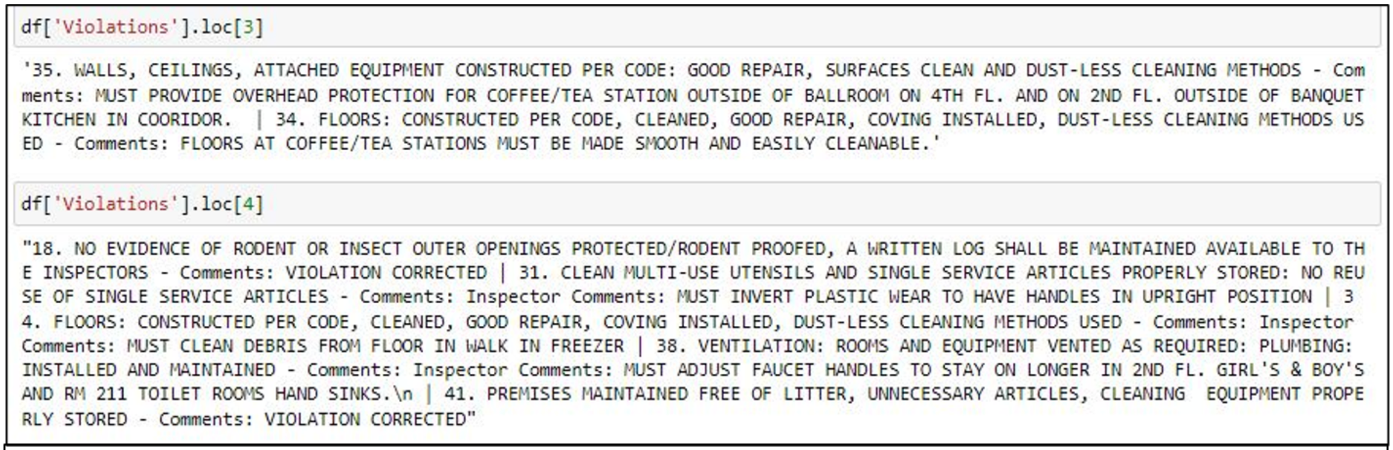
All			Inspection ID	DBA Name	AKA Name	License #	Facility Type	Risk	Address	City	State	Zip	Inspection Date	Inspection Type	Results
☆		1.	2079132	MARRIOT MARQUIS CHICAGO	MARRIOT MARQUIS CHICAGO	2517328	RESTAURANT	Risk 1 (High)	2121 S PRAIRIE AVE	CHICAGO	IL	60616	08/28/2017	License	Pass
☆		2.	2079123	MARRIOT MARQUIS CHICAGO	MARRIOT MARQUIS CHICAGO	2517338	RESTAURANT	Risk 1 (High)	2121 S PRAIRIE AVE	CHICAGO	IL	60616	08/28/2017	License	Pass
☆		3.	2079104	SHRIMP'S FISH CHICKEN	SHRIMP'S FISH CHICKEN	2549497	RESTAURANT	Risk 2 (Medium)	4355 S COTTAGE GROVE AVE	CHICAGO	IL	60653	08/28/2017	License	Fail

A final data transformation we undertook was converting the “Inspection Date” which was in American format to an ISO format so that it could be more easily manipulated. This is important for our U1 use case, because we want more recent inspections to take precedence in our UI and.

The OpenRefine work to prune and standardize the data was required to meet our use case. Without the time spent clustering and resolving restaurant type issues we would have lost a substantial part of the useful data. This goes the same for location, if we had not taken the time to comb through merging entries we would have lost significant entries. While a quick change, transforming “Inspection Date” to a standard format directly covers our use case, as we only want to surface the most recent inspection. In a sense, pruning the data and making sure only inspections that actually took place was not necessary (because we could have simply not included them in queries), but leaving in this chaffe would have made all subsequent steps less computationally efficient and increased the risk that these times slipped through.

At this point we transitioned to using Python for the third phase of our data cleaning plan, which included restructuring the data by splitting the “Violations” column and dropping outliers or entries that failed to meet our integrity constraints.

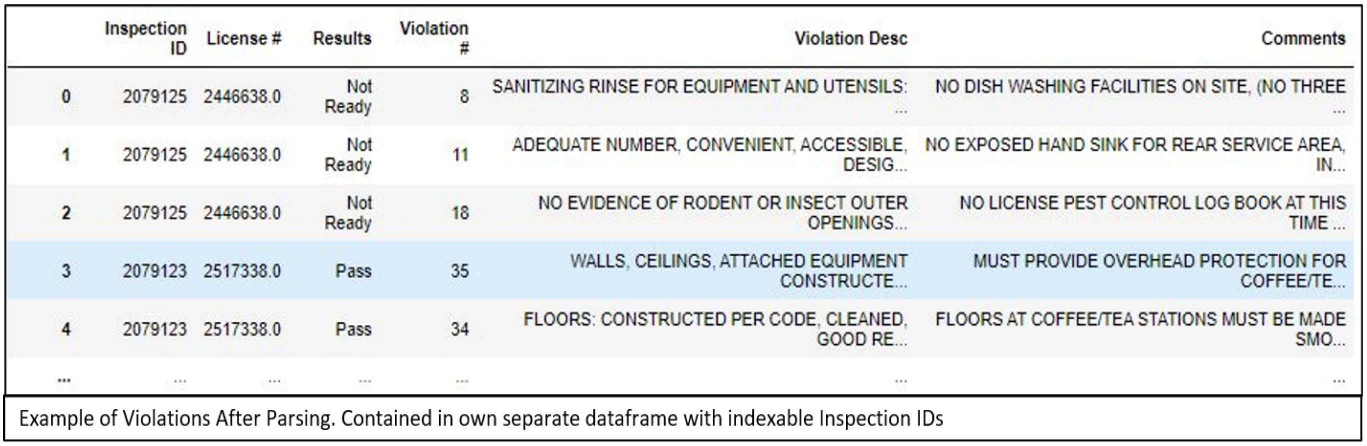
In order to parse the Violation column into a usable format, each entry needed to be split twice. The Violations themselves were Pipe delimited making them easy to split. The comments per Violation were also equally easy to split, due to the start of the comments section saying “-Comments:” consistently. However, each inspection was given a unique Inspection ID, so in order to keep the data indexable, each Violation needed to also have the unique Inspection ID and the License Number associated with the entry present. This meant that the parsing function used to separate out the violations needed to strip each Violation per Inspection and ensure the Inspection ID was present in that row.



```
df['Violations'].loc[3]

'35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Com
ments: MUST PROVIDE OVERHEAD PROTECTION FOR COFFEE/TEA STATION OUTSIDE OF BALLROOM ON 4TH FL. AND ON 2ND FL. OUTSIDE OF BANQUET
KITCHEN IN COORIDOR.  | 34. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUST-LESS CLEANING METHODS US
ED - Comments: FLOORS AT COFFEE/TEA STATIONS MUST BE MADE SMOOTH AND EASILY CLEANABLE.'

df['Violations'].loc[4]

"18. NO EVIDENCE OF RODENT OR INSECT OUTER OPENINGS PROTECTED/RODENT PROOFED, A WRITTEN LOG SHALL BE MAINTAINED AVAILABLE TO TH
E INSPECTORS - Comments: VIOLATION CORRECTED | 31. CLEAN MULTI-USE UTENSILS AND SINGLE SERVICE ARTICLES PROPERLY STORED: NO REU
SE OF SINGLE SERVICE ARTICLES - Comments: Inspector Comments: MUST INVERT PLASTIC WEAR TO HAVE HANDLES IN UPRIGHT POSITION | 3
4. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUST-LESS CLEANING METHODS USED - Comments: Inspector
Comments: MUST CLEAN DEBRIS FROM FLOOR IN WALK IN FREEZER | 38. VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING:
INSTALLED AND MAINTAINED - Comments: Inspector Comments: MUST ADJUST FAUCET HANDLES TO STAY ON LONGER IN 2ND FL. GIRL'S & BOY'S
AND RM 211 TOILET ROOMS HAND SINKS.\n | 41. PREMISES MAINTAINED FREE OF LITTER, UNNECESSARY ARTICLES, CLEANING  EQUIPMENT PROPE
RLY STORED - Comments: VIOLATION CORRECTED"
```

Example of Violations Before Parsing. Example Circled in Red: Pipe Delimit and Comment delimiter used.

From there, splitting the Violation Number from the description was very straight forward. Having the Violation Number separated allows us to create a more standard database structure by placing the Violation Number and its corresponding description in its own indexable table so as not to over encumber the Inspection_Details table with repeated Descriptions of the Violation. At the end of the parsing, we are left with a dataframe of the indexable Inspection IDs and the Inspection and Violation information needed to create our database tables.

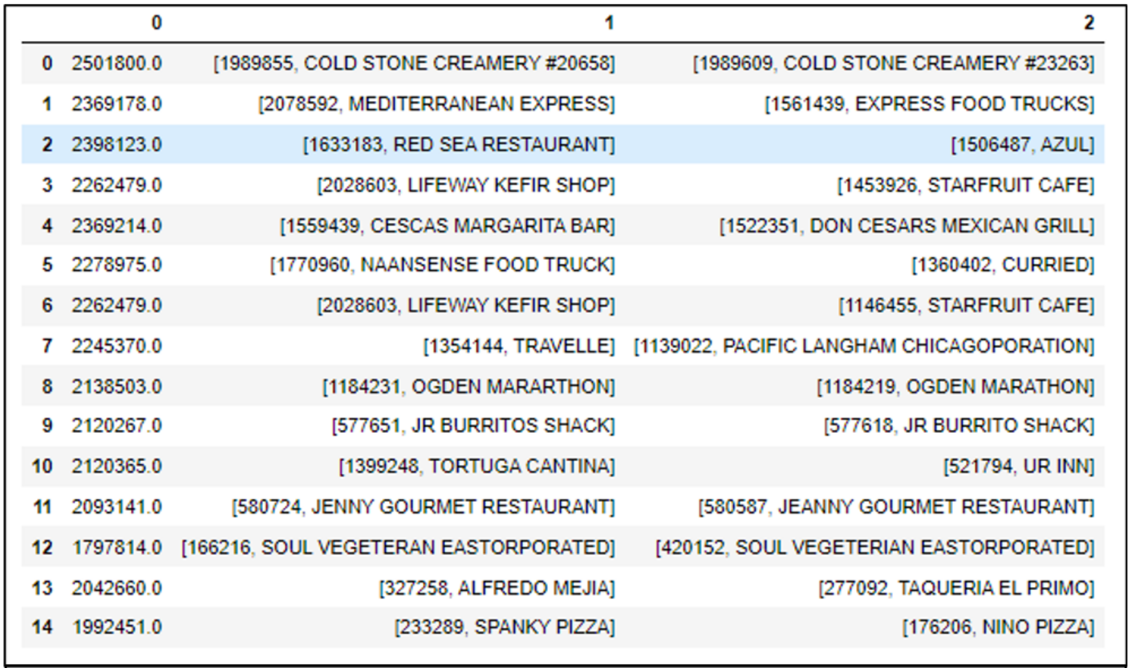


	Inspection ID	License #	Results	Violation #	Violation Desc	Comments
0	2079125	2446638.0	Not Ready	8	SANITIZING RINSE FOR EQUIPMENT AND UTENSILS: ...	NO DISH WASHING FACILITIES ON SITE, (NO THREE ...
1	2079125	2446638.0	Not Ready	11	ADEQUATE NUMBER, CONVENIENT, ACCESSIBLE, DESIG...	NO EXPOSED HAND SINK FOR REAR SERVICE AREA, IN...
2	2079125	2446638.0	Not Ready	18	NO EVIDENCE OF RODENT OR INSECT OUTER OPENINGS...	NO LICENSE PEST CONTROL LOG BOOK AT THIS TIME ...
3	2079123	2517338.0	Pass	35	WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTE...	MUST PROVIDE OVERHEAD PROTECTION FOR COFFEE/TE...
4	2079123	2517338.0	Pass	34	FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD RE...	FLOORS AT COFFEE/TEA STATIONS MUST BE MADE SMO...
...	...	...	...	...	...	...

Example of Violations After Parsing. Contained in own separate dataframe with indexable Inspection IDs

Before going to insert the data into our database, we first want to check the main indexable keys that we will be using as the Primary Keys for uniqueness. **This is our main integrity test** for potential violations so we can be sure we are not attempting to insert erroneous data into our new database. We do this by using Python to scan the dataframe.

Firstly, we want to check to see if the License # given in the data is recorded under two different business names. The License # should theoretically be unique to the business itself, so if more than one Business has the same License #, we should consider this to be an integrity violation. When checking this on the original unaltered dataset, we get 78 integrity violations. Many of these are the same business spelled in various ways. After running our OpenRefine cleaning, we get this reduced down to 15, with only a few of which being unfixable outliers that will need to be removed to fit the integrity constraints of our proposed database. Below you can see the remaining integrity violations that we will consider outliers.

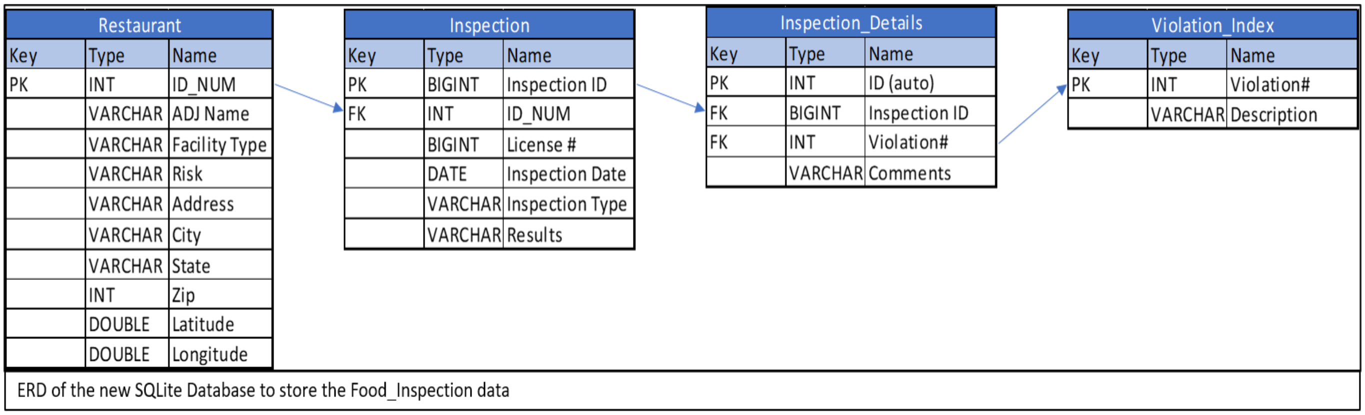


	0	1	2
0	2501800.0	[1989855, COLD STONE CREAMERY #20658]	[1989609, COLD STONE CREAMERY #23263]
1	2369178.0	[2078592, MEDITERRANEAN EXPRESS]	[1561439, EXPRESS FOOD TRUCKS]
2	2398123.0	[1633183, RED SEA RESTAURANT]	[1506487, AZUL]
3	2262479.0	[2028603, LIFEWAY KEFIR SHOP]	[1453926, STARFRUIT CAFE]
4	2369214.0	[1559439, CESCAS MARGARITA BAR]	[1522351, DON CESARS MEXICAN GRILL]
5	2278975.0	[1770960, NAANSENSE FOOD TRUCK]	[1360402, CURRIED]
6	2262479.0	[2028603, LIFEWAY KEFIR SHOP]	[1146455, STARFRUIT CAFE]
7	2245370.0	[1354144, TRAVELLE]	[1139022, PACIFIC LANGHAM CHICAGOPORATION]
8	2138503.0	[1184231, OGDEN MARARTHON]	[1184219, OGDEN MARATHON]
9	2120267.0	[577651, JR BURRITOS SHACK]	[577618, JR BURRITO SHACK]
10	2120365.0	[1399248, TORTUGA CANTINA]	[521794, UR INN]
11	2093141.0	[580724, JENNY GOURMET RESTAURANT]	[580587, JEANNY GOURMET RESTAURANT]
12	1797814.0	[166216, SOUL VEGETERAN EASTORPORATED]	[420152, SOUL VEGETERIAN EASTORPORATED]
13	2042660.0	[327258, ALFREDO MEJIA]	[277092, TAQUERIA EL PRIMO]
14	1992451.0	[233289, SPANKY PIZZA]	[176206, NINO PIZZA]

Screen Capture of the remaining Integrity Violations. Key for Reference:
Licence # | [Inspection ID, DBA NAME] | [Inspection ID, DBA NAME]

So after cleaning up these last names and ensuring that the Inspection ID is in fact unique, we are good to begin **prepping the data to insert into the SQLite database**.

Inserting the data into a database is in itself its own integrity test. If the keys are off, or the structure of the insert is wrong, the insertion should error and fixes will need to be made. This makes the insertion one of the largest tests for data integrity in our workflow. Below you can see an ERD of the structure we are using for our food inspection database.



Restaurant

Key	Type	Name
PK	INT	ID_NUM
	VARCHAR	ADJ Name
	VARCHAR	Facility Type
	VARCHAR	Risk
	VARCHAR	Address
	VARCHAR	City
	VARCHAR	State
	INT	Zip
	DOUBLE	Latitude
	DOUBLE	Longitude

Inspection

Key	Type	Name
PK	BIGINT	Inspection ID
FK	INT	ID_NUM
	BIGINT	License #
	DATE	Inspection Date
	VARCHAR	Inspection Type
	VARCHAR	Results

Inspection_Details

Key	Type	Name
PK	INT	ID (auto)
FK	BIGINT	Inspection ID
FK	INT	Violation#
	VARCHAR	Comments

Violation_Index

Key	Type	Name
PK	INT	Violation#
	VARCHAR	Description

ERD of the new SQLite Database to store the Food_Inspection data

We begin by adding data to the violation table, which is a very simple table of the unique Violation Numbers and their Descriptions so as to be indexable for the inspections. After the violations comes the Restaurants, which include all of the unique details of each individual restaurant, with its name, location, type and risk level. This required little alteration aside from removing the duplicates and adding in a unique identification number and went very easily.

Next is the insertion of the Inspections into the Inspection table. This is the hardest insertion because in order to keep the database indexable, the unique id_number assigned in the last step to each restaurant in the Restaurant table needs to be merged so you can index the inspection to the restaurant. In order to accomplish this, a subset was used from the cleaned dataset and the new Restaurant data table was then merged to match the Inspection IDs to their corresponding restaurant id_nums (you can see this relation in our outer workflow model). After that, simply using the violation data to complete the Inspection_Detail table data insert is very straightforward as all the needed data is already present.

Now that the data is in our table, we can do some due diligence and query both the original dataset and our new database to see what the effects of changes we made!

To query the original dataset as well as the new database, we used Python to run the queries so we could easily compare the outputs of the queries. To query the original dataset, we used the pandasql library, which is an extension of the pandas dataframe library, so we could run SQL queries directly from the dataframe. This allows us to keep the queries the same despite the structural differences between the original data csv file and the new SQLite database structure.

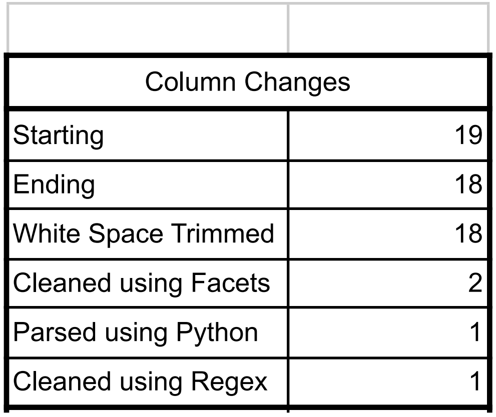
The first query tests were run on the string fields to test for count changes. A simple check of the count of distinct names from the original dataset showed that there were 24685 unique restaurant names found. After our cleaning procedure, this number was reduced down to 12332 unique names in the restaurant table of the database. We can reasonably conclude that this is from not just the removal of dataset rows irrelevant to our use case, but mainly due to the standardization process and fixing of erroneous spellings. We can also show using the WHERE clause to capture only restaurants in the city of Chicago that the original dataset still had 24579 distinct restaurant names with this stipulation. Checking the Facility types in a similar manner showed a reduction of 447 distinct facility types in the original dataset down to 113 in our cleaned data. Interestingly enough, despite this data being for the city of Chicago, when doing this on the City column, the original data had 57 unique values. This was obviously reduced to 1 for our final cleaned data since our use case is only for the city of Chicago.

Further checking the data with SQL queries, we next looked at the Inspection ID, which is arguably the most important data point in our dataset. We see that the original data had 153,810 unique Inspection IDs and our cleaned data has 96,063. It is again reasonable to infer that this reduction is due to the reduction in the entries unneeded for our use case. This reduction is also the primary metric to indicate the amount of data that was removed from the original dataset, as each row corresponded to a unique Inspection ID.

This means all-in-all, we reduced the dataset from that original 153,810 inspections down to 96,063 relevant to our use case, and indexed the restaurants and violation information related to those 96,063 inspections. This method allows us to easily query our database for the inspection information based on either the inspection itself, or the restaurant in question by indexing its id_num.

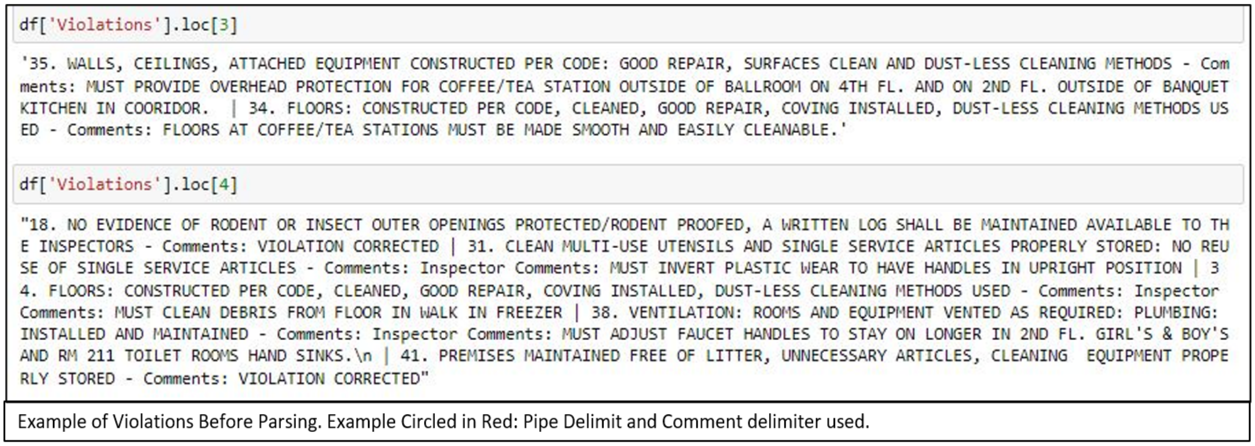
DOCUMENT DATA QUALITY CHANGES

The tables below along with the before and after queries demonstrate. Data quality was significantly improved for our main use case. We fixed all integrity constraints and pruned the data of irrelevant entries. All told, we eliminated 37.5% of the entries in the database that were either the wrong facility type, not in Chicago or represented entries where an inspection did not occur. More detail on the impact as well as the reasoning behind these changes can be found in the narrative above.



Column Changes	
Starting	19
Ending	18
White Space Trimmed	18
Cleaned using Facets	2
Parsed using Python	1
Cleaned using Regex	1

Entries	
Starting	153810
Ending	96063



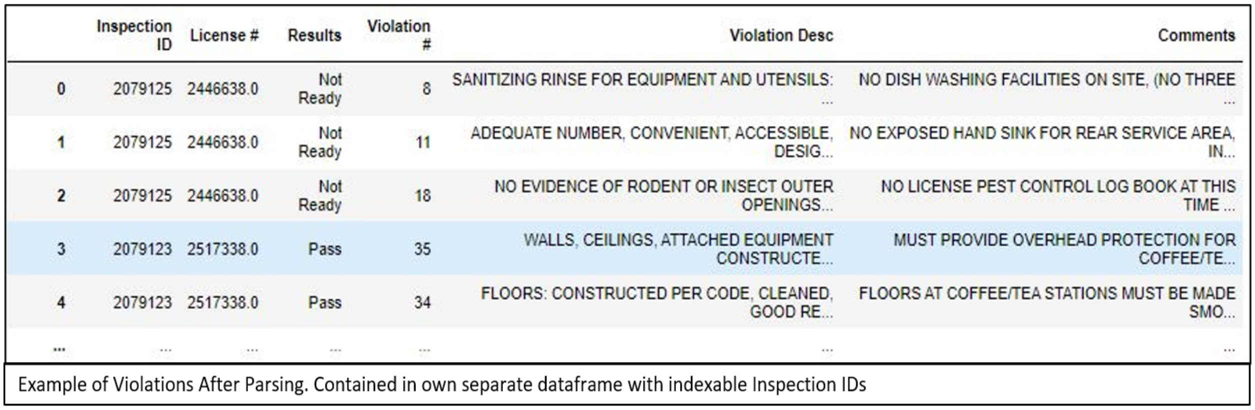
```
df['Violations'].loc[3]
```

```
'35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Com
ments: MUST PROVIDE OVERHEAD PROTECTION FOR COFFEE/TEA STATION OUTSIDE OF BALLROOM ON 4TH FL. AND ON 2ND FL. OUTSIDE OF BANQUET
KITCHEN IN COORIDOR.  | 34. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUST-LESS CLEANING METHODS US
ED - Comments: FLOORS AT COFFEE/TEA STATIONS MUST BE MADE SMOOTH AND EASILY CLEANABLE.'
```

```
df['Violations'].loc[4]
```

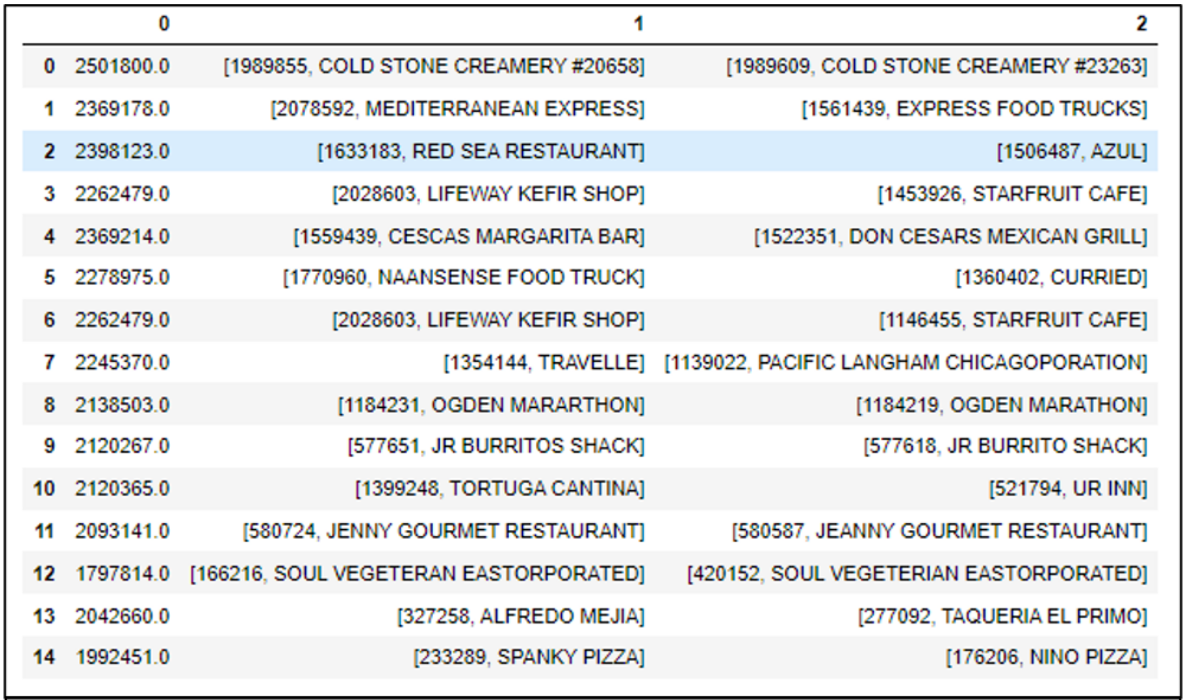
```
"18. NO EVIDENCE OF RODENT OR INSECT OUTER OPENINGS PROTECTED/RODENT PROOFED, A WRITTEN LOG SHALL BE MAINTAINED AVAILABLE TO TH
E INSPECTORS - Comments: VIOLATION CORRECTED | 31. CLEAN MULTI-USE UTENSILS AND SINGLE SERVICE ARTICLES PROPERLY STORED: NO REU
SE OF SINGLE SERVICE ARTICLES - Comments: Inspector Comments: MUST INVERT PLASTIC WEAR TO HAVE HANDLES IN UPRIGHT POSITION | 3
4. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUST-LESS CLEANING METHODS USED - Comments: Inspector
Comments: MUST CLEAN DEBRIS FROM FLOOR IN WALK IN FREEZER | 38. VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING:
INSTALLED AND MAINTAINED - Comments: Inspector Comments: MUST ADJUST FAUCET HANDLES TO STAY ON LONGER IN 2ND FL. GIRL'S & BOY'S
AND RM 211 TOILET ROOMS HAND SINKS.\n | 41. PREMISES MAINTAINED FREE OF LITTER, UNNECESSARY ARTICLES, CLEANING  EQUIPMENT PROPE
RLY STORED - Comments: VIOLATION CORRECTED"
```

Example of Violations Before Parsing. Example Circled in Red: Pipe Delimit and Comment delimiter used.



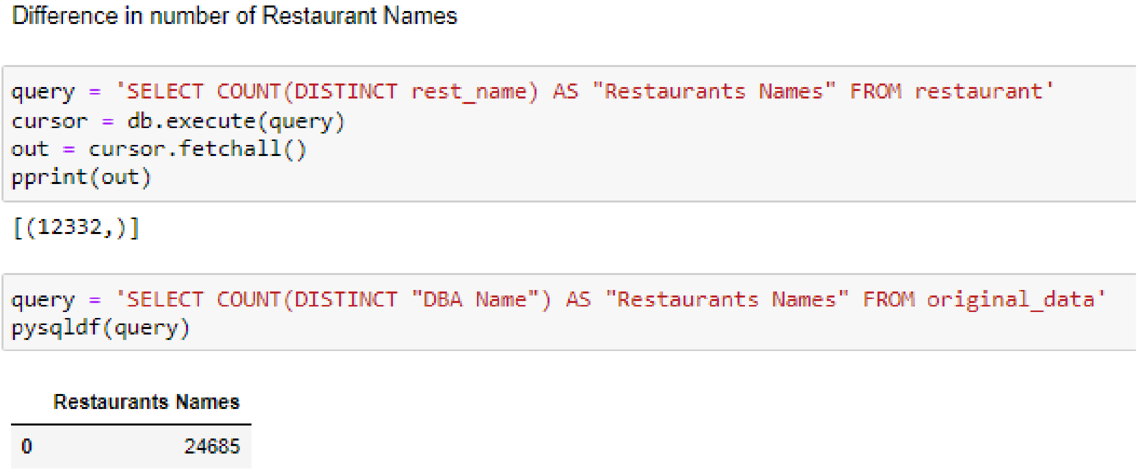
	Inspection ID	License #	Results	Violation #	Violation Desc	Comments
0	2079125	2446638.0	Not Ready	8	SANITIZING RINSE FOR EQUIPMENT AND UTENSILS: ...	NO DISH WASHING FACILITIES ON SITE, (NO THREE ...
1	2079125	2446638.0	Not Ready	11	ADEQUATE NUMBER, CONVENIENT, ACCESSIBLE, DESIG...	NO EXPOSED HAND SINK FOR REAR SERVICE AREA, IN...
2	2079125	2446638.0	Not Ready	18	NO EVIDENCE OF RODENT OR INSECT OUTER OPENINGS...	NO LICENSE PEST CONTROL LOG BOOK AT THIS TIME ...
3	2079123	2517338.0	Pass	35	WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTE...	MUST PROVIDE OVERHEAD PROTECTION FOR COFFEE/TE...
4	2079123	2517338.0	Pass	34	FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD RE...	FLOORS AT COFFEE/TEA STATIONS MUST BE MADE SMO...
...	...	...	...	...	...	...

Example of Violations After Parsing. Contained in own separate dataframe with indexable Inspection IDs



	0	1	2
0	2501800.0	[1989855, COLD STONE CREAMERY #20658]	[1989609, COLD STONE CREAMERY #23263]
1	2369178.0	[2078592, MEDITERRANEAN EXPRESS]	[1561439, EXPRESS FOOD TRUCKS]
2	2398123.0	[1633183, RED SEA RESTAURANT]	[1506487, AZUL]
3	2262479.0	[2028603, LIFEWAY KEFIR SHOP]	[1453926, STARFRUIT CAFE]
4	2369214.0	[1559439, CESCAS MARGARITA BAR]	[1522351, DON CESARS MEXICAN GRILL]
5	2278975.0	[1770960, NAANSENSE FOOD TRUCK]	[1360402, CURRIED]
6	2262479.0	[2028603, LIFEWAY KEFIR SHOP]	[1146455, STARFRUIT CAFE]
7	2245370.0	[1354144, TRAVELLE]	[1139022, PACIFIC LANGHAM CHICAGOPORATION]
8	2138503.0	[1184231, OGDEN MARARTHON]	[1184219, OGDEN MARATHON]
9	2120267.0	[577651, JR BURRITOS SHACK]	[577618, JR BURRITO SHACK]
10	2120365.0	[1399248, TORTUGA CANTINA]	[521794, UR INN]
11	2093141.0	[580724, JENNY GOURMET RESTAURANT]	[580587, JEANNY GOURMET RESTAURANT]
12	1797814.0	[166216, SOUL VEGETERAN EASTORPORATED]	[420152, SOUL VEGETERIAN EASTORPORATED]
13	2042660.0	[327258, ALFREDO MEJIA]	[277092, TAQUERIA EL PRIMO]
14	1992451.0	[233289, SPANKY PIZZA]	[176206, NINO PIZZA]

Screen Capture of the remaining Integrity Violations. Key for Reference:
Licence # | [Inspection ID, DBA NAME] | [Inspection ID, DBA NAME]



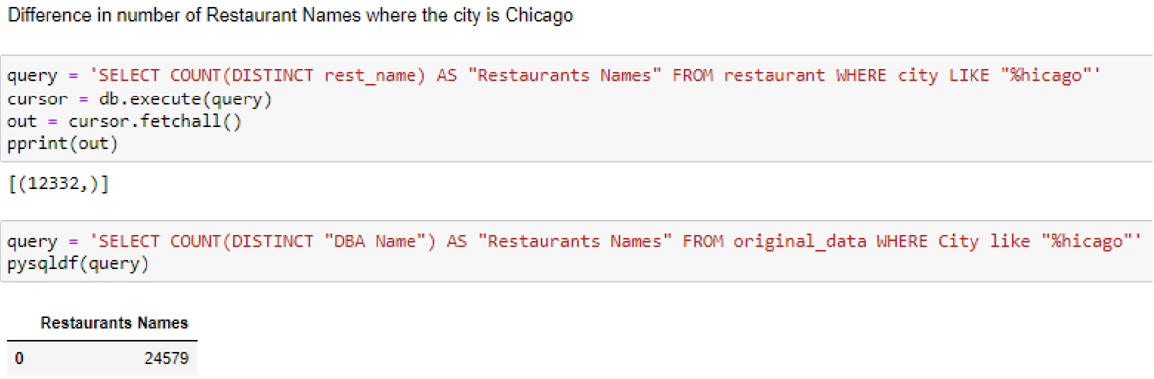
Difference in number of Restaurant Names

```
query = 'SELECT COUNT(DISTINCT rest_name) AS "Restaurants Names" FROM restaurant'
cursor = db.execute(query)
out = cursor.fetchall()
pprint(out)
```

```
[(12332,)]
```

```
query = 'SELECT COUNT(DISTINCT "DBA Name") AS "Restaurants Names" FROM original_data'
pysqldf(query)
```

	Restaurants Names
0	24685



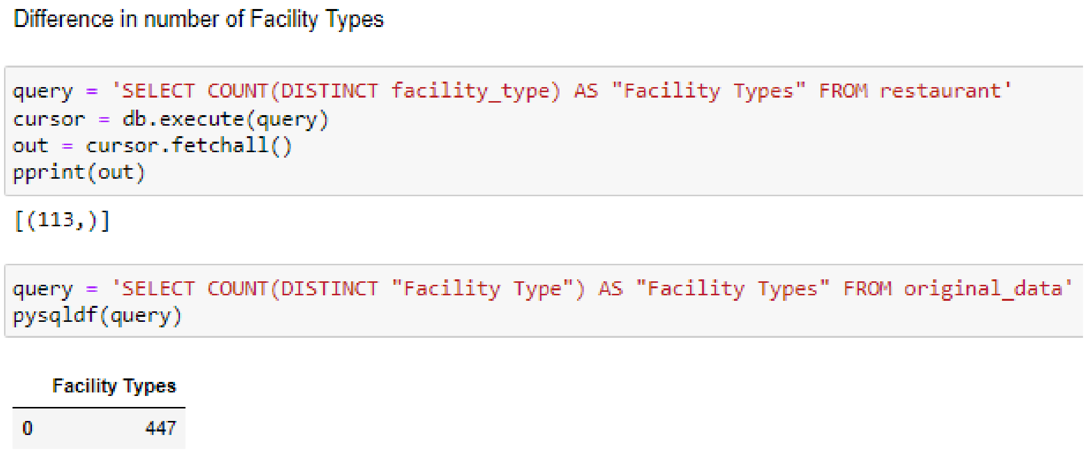
Difference in number of Restaurant Names where the city is Chicago

```
query = 'SELECT COUNT(DISTINCT rest_name) AS "Restaurants Names" FROM restaurant WHERE city LIKE "%hicago"'
cursor = db.execute(query)
out = cursor.fetchall()
pprint(out)
```

```
[(12332,)]
```

```
query = 'SELECT COUNT(DISTINCT "DBA Name") AS "Restaurants Names" FROM original_data WHERE City like "%hicago"'
pysqldf(query)
```

	Restaurants Names
0	24579



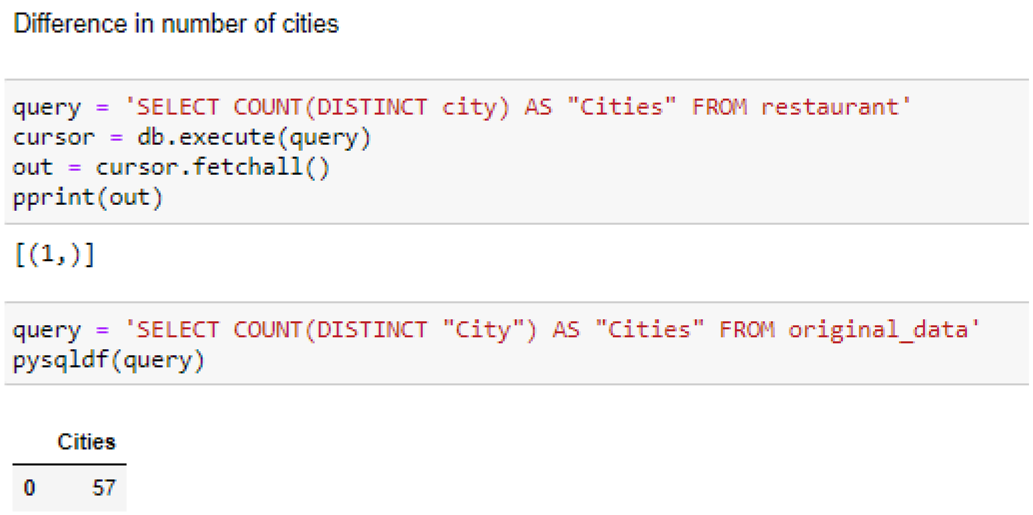
Difference in number of Facility Types

```
query = 'SELECT COUNT(DISTINCT facility_type) AS "Facility Types" FROM restaurant'
cursor = db.execute(query)
out = cursor.fetchall()
pprint(out)
```

```
[(113,)]
```

```
query = 'SELECT COUNT(DISTINCT "Facility Type") AS "Facility Types" FROM original_data'
pysqldf(query)
```

	Facility Types
0	447



Difference in number of cities

```
query = 'SELECT COUNT(DISTINCT city) AS "Cities" FROM restaurant'
cursor = db.execute(query)
out = cursor.fetchall()
pprint(out)
```

```
[(1,)]
```

```
query = 'SELECT COUNT(DISTINCT "City") AS "Cities" FROM original_data'
pysqldf(query)
```

	Cities
0	57

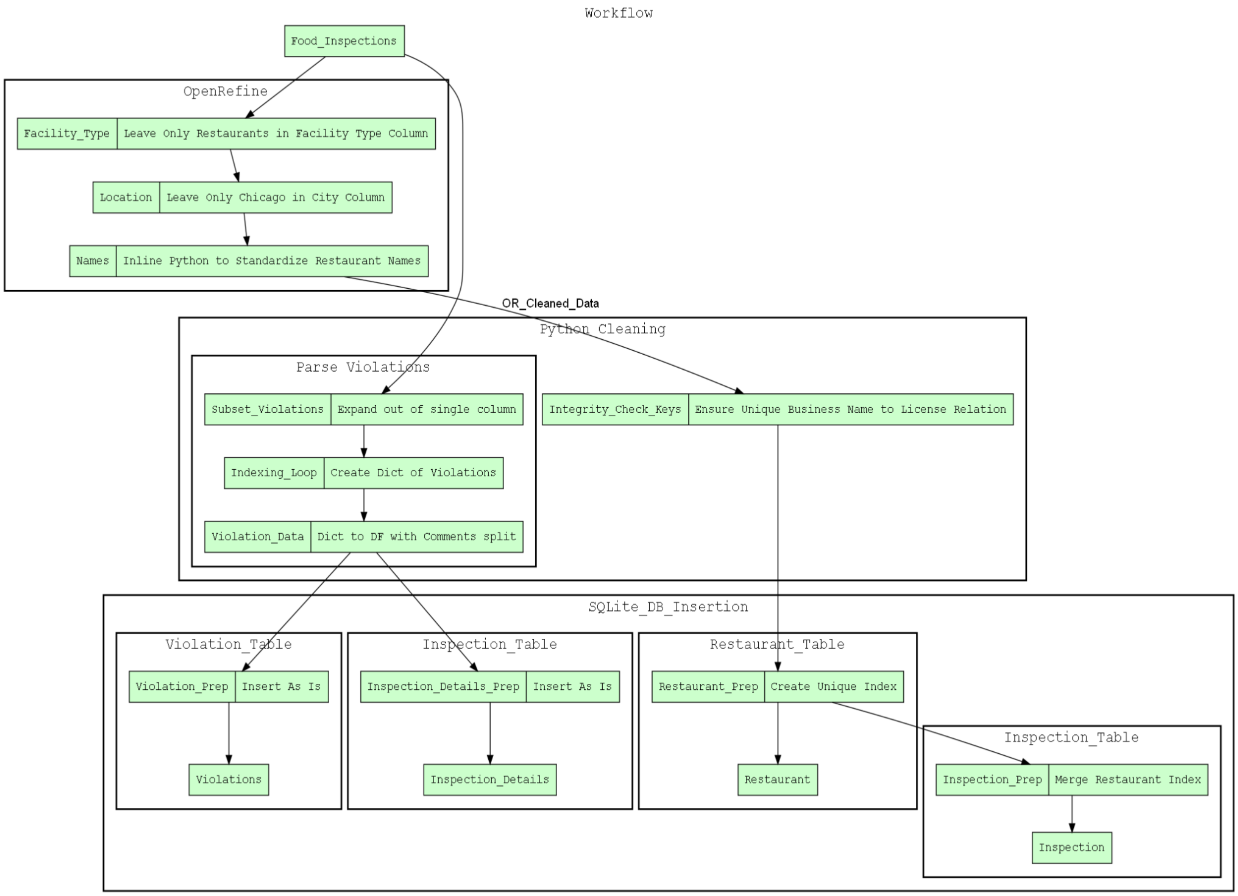
WORKFLOW MODEL

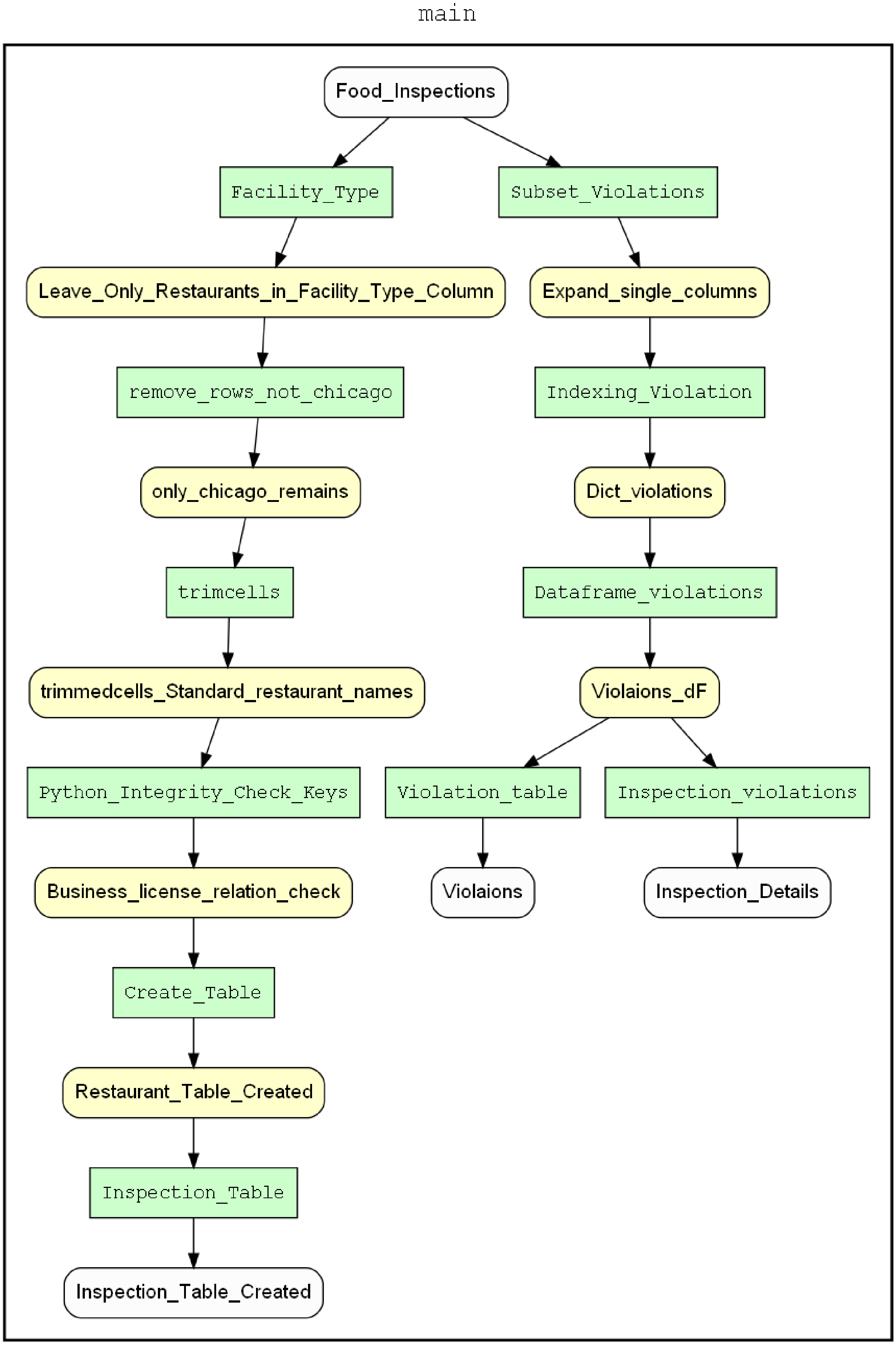
We have provided outer and inner workflow graphs. The inner workflow was generated by annotations output from the OpenRefine history and generated with the or2yw tool and the outer workflows were generated using YesWorkFlow and GraphViz.

The detailed granular Inner workflow was not included as part of this report because it was too large to render properly, it can be found in the supplementary material in pdf format.

Yesworkflow was chosen because it was part of the class, giving us the opportunity to experiment with its capabilities for inline code graphical representations.

These are the outer (primary) workflow




main
Food_Inspections
Facility_Type
Subset_Violations
Leave_Only_Restaurants_in_Facility_Type_Column
Expand_single_columns
remove_rows_not_chicago
Indexing_Violation
only_chicago_remains
Dict_violations
trimcells
Dataframe_violations
trimmedcells_Standard_restaurant_names
Violaions_dF
Python_Integrity_Check_Keys
Violation_table
Inspection_violations
Business_license_relation_check
Violaions
Inspection_Details
Create_Table
Restaurant_Table_Created
Inspection_Table
Inspection_Table_Created

SUMMARY

At the end of this project we successfully cleaned the Chicago Department Public Health's food inspection data for our use case of providing health inspection lookup data for our app. We used OpenRefine to fix formatting, find errors in many columns, transform temporal information and eliminate over a third of the entries (57,717) that were not useful for our goal use case. Then we used Python to perform further integrity constraint cleaning and created a function that would parse the free text entry "Violations" column so it was easy to query. These were put into a SQLite database that will form the backbone of our app's health inspection lookup functionality.

Along the way we learned many lessons. It furthered our belief OpenRefine is a powerful and flexible data cleaning tool. Compared to using Microsoft Excel it likely saved 20 hours. We also learned that we have to be careful using facets to cluster. Once when experimenting with an NGRAM size of 10 we got 95% of our entries as false positives!

We also learned the power of SQLite and how easy it can be to create a local database for a multitude of purposes. We also learned a few valuable ways to create workflow models quickly that could aid in showing and explaining complex data manipulation processes to outside parties.

We also learned the advantages of visualization to help understand data processing and help understand the changes in data. We understood Yesworkflow's strength in documenting data visually and openrefine in automatically performing this step.These enable quickly sharing the changes to a broad audience.

A potential next step for this project would be to set up a pipeline so that we could get up to date data, as part of this we use more automation in the cleaning workflow for inserting information into the database and building error reports to enhance review. Additionally we could use the data cleaning recipe we created here to clean food inspection data sets for other cities as Helph expands to more cities. Lastly, we could look at feature engineering and data augmentation for ML applications, a future use case here could be predicting what restaurants would have health violations soon.

This project's contributions: Paul Bromen performed the initial cleaning in OpenRefine, Tony Petrotte then performed the data cleaning steps using Python and put the final results into the SQLite database, and Issac contributed the workflow model. Documenting data quality changes was a team effort although, being most knowledgeable on this particular database setup, Tony took charge of the integrity constraint implementation. For the write up we each wrote our respective sections and collaborated on creating a cohesive document.

Supplementary Materials Included in Zip file:

- **Workflow Model**
    - **Workflow Images**

        - InnerWorkFlow.pdf
        - OuterWorkFlow1.pdf
        - OuterWorkFlow2.pdf
        - Parse_Violation_YW.pdf
    - **Files to produce Workflow Images**
        - OuterWorkFlow1.gv
        - OuterWorkFlow2.py
        - OuterWorkFlow2.gv
        - parse_violation_yw.py
- **OpenRefine History**
    - **JSON file containing OpenRefine History**
        - OpenRefineHistory.json
- **Other History**
    - **Notebooks used for Python alterations**
        - db_setup.ipynb
        - Key_Integrity_Check.ipynb
        - Parse_Violations.ipynb
        - Query_Checking.ipynb
    - **Scripts**
        - parse_violations.py
        - parse_violation_yw.py
        - transform_name.py
- **Queries**
    - **File Containing Queries ran**
        - queries.txt
- **Datasets**
    - **File Containing link to Box Drive with Original data and new SQLite database file**
        - DataLinks.txt