**Technical Report | ETL Project**

**Compiled By: Eric Meyer, Michelle Simek and Kristen Scaletta**

**Data Sources:**

Food inspection .csv: <https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5>

Sidewalk café permit (Chicago) .csv: https://data.cityofchicago.org/Community-Economic-Development/Sidewalk-Cafe-Permits-Current/qnjv-hj2q

**Type of Transformation Needed:**

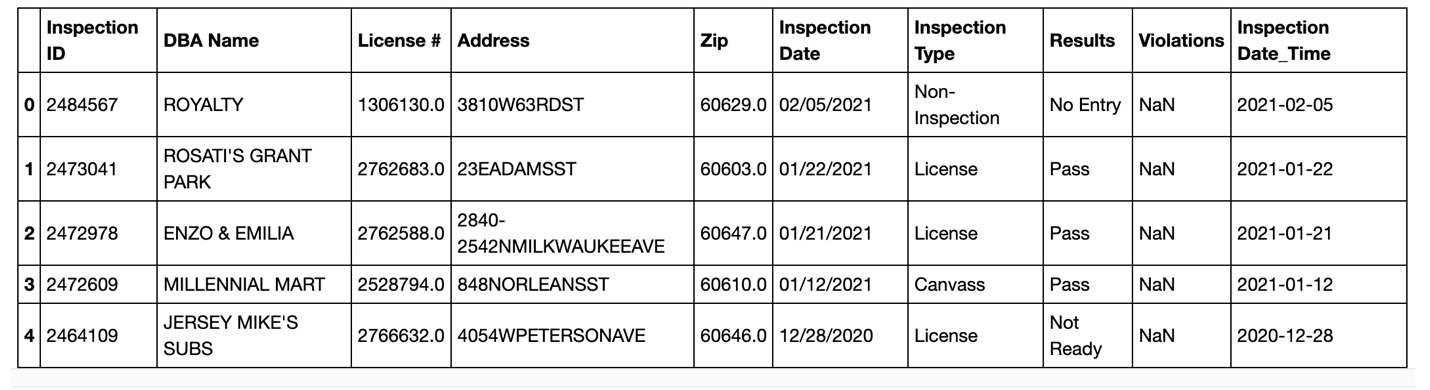
To use the data in the manner intended, we transformed it in the following manner (listed by data source):

*Food Inspection Data*

* Removed columns to only include those needed for our analysis, resulting in the following:

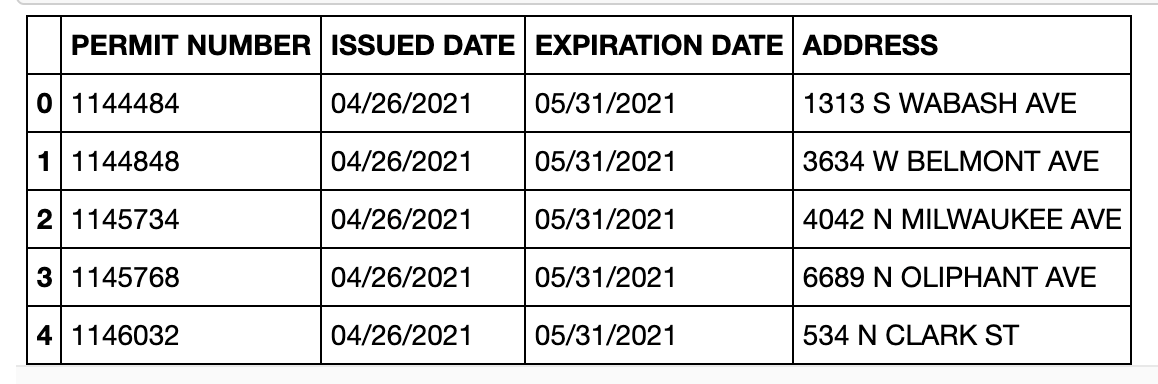
**

* When exploring the data, we noticed there was a space at the end of the address in one table we were working with and not the other. Therefore, we removed all spaces in the address column so that we could use that column to join on another table.
* We were only interested in data from 2020 or above so we added a new column based on the Inspection Date column that was to be in date/time format and filtered by the year 2020 or greater. This allowed us to match the same data range as provided in the other table we used. See screenshot below, which includes the new Inspection Date\_Time column.

****

*Sidewalk Café Data*

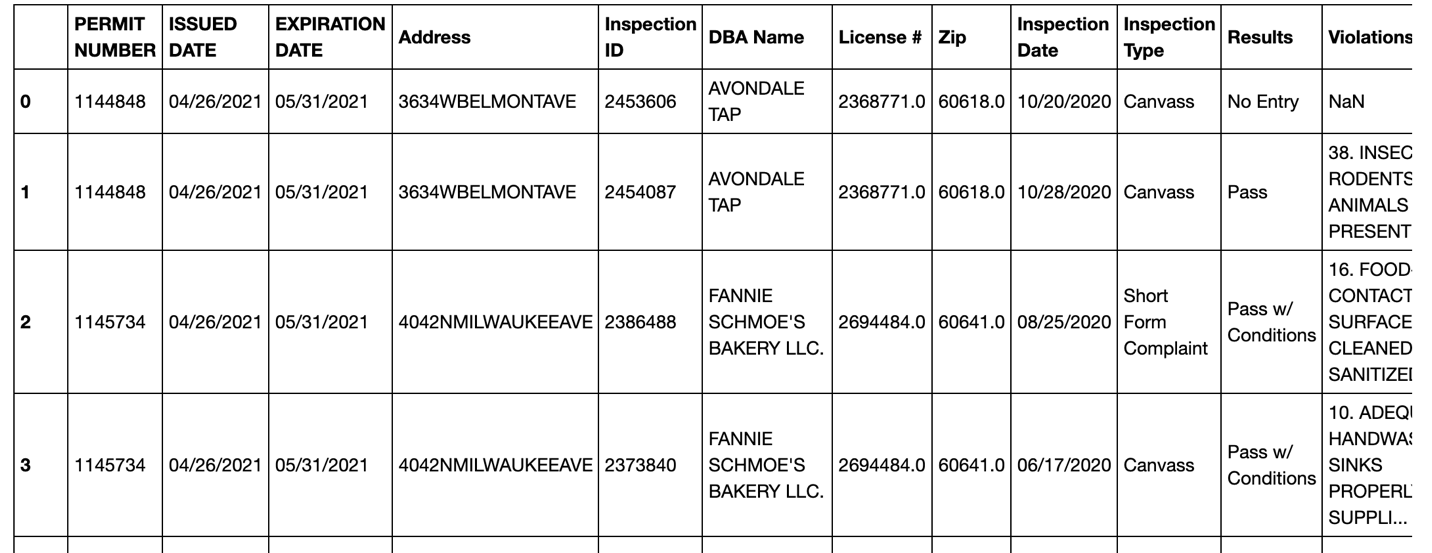
* Removed columns to only include those needed for our analysis, resulting in the following:

**

* Renamed the “ADDRESS” column to be “Address” in order to use it to join this data with that in the Food Inspection Data table described above.
* Removed all spaces in the address column so that we could use that column to join on another table.

*Combining Data Sources*

* Once the data sources were cleaned, we performed an inner join on the data using the “Address” column. This resulted in 1623 rows.



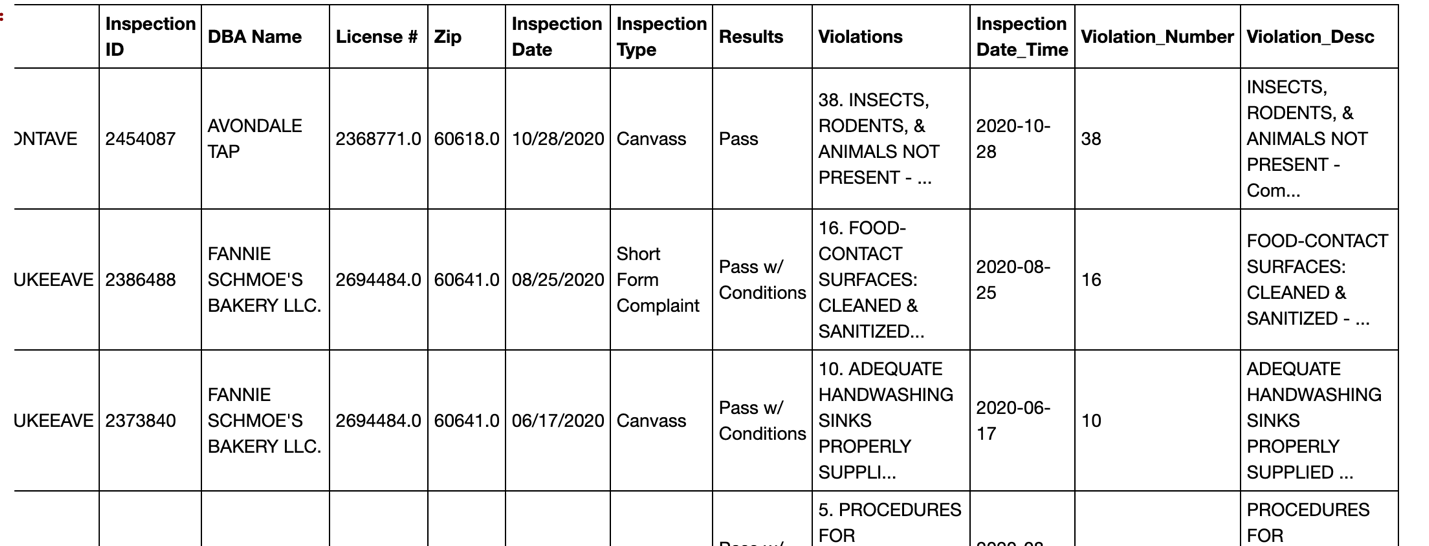
* We were only interested in rows that had each value populated so we then dropped all rows that had any null values. This resulted in 1189 rows.

*Preparing Dataframes for Database*

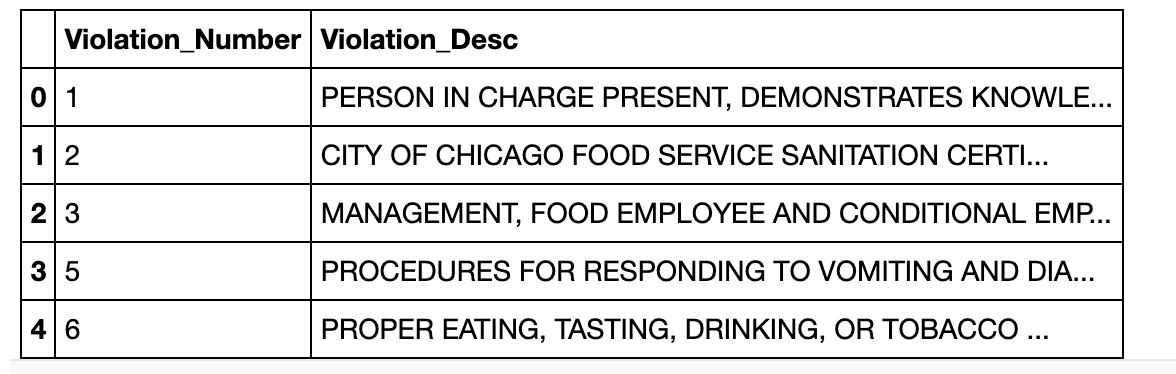
Our data had three columns that did not contain unique values: inspection type, results and violations. Rather than repeating text in this dataframe, we broke each of those columns into its own table and referenced the appropriate row in the joined dataframe above with a unique ID. Here is how we did that for each column referenced:

*Violations\_df*

* We created two new columns in the joined dataframe by splitting the Violations column on the “.” character.

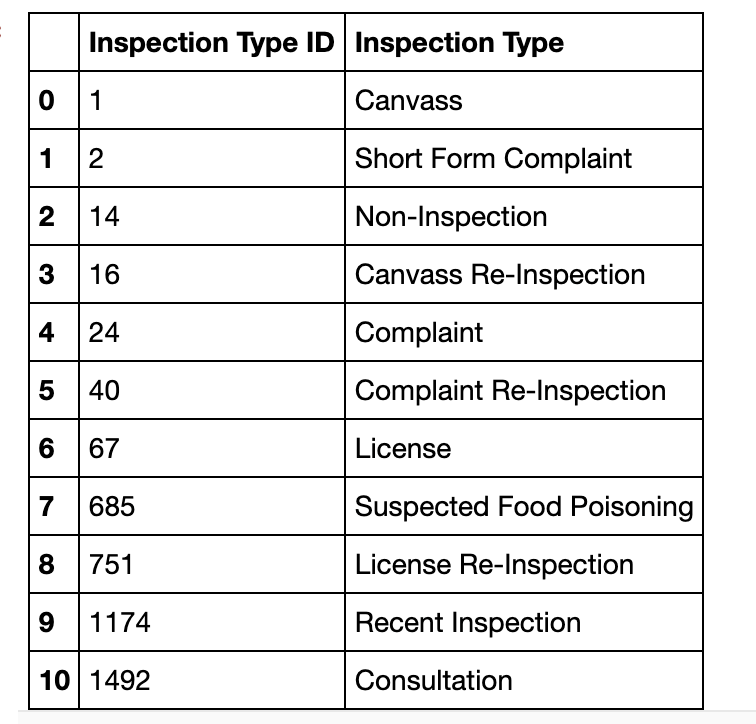


* We made a new dataframe, “Violations\_df”, with the two new columns, “Violation\_Number” and “Violation\_Desc”.
* In the new datafraame, we dropped duplicate rows.
* Next, we changed Violation\_Number from a string to an integer to sort by that column.
* We then reset the index and specified column names, resulting in this dataframe:



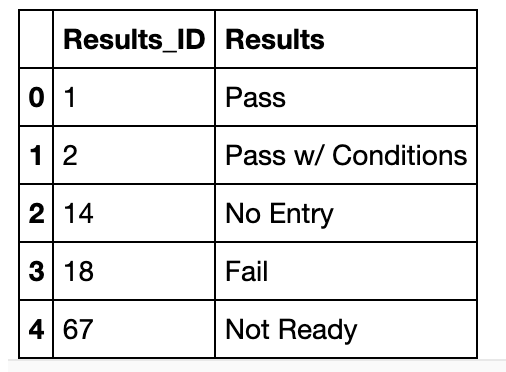
*Inspections\_df*

* We took the Inspection Type column from our joined dataframe as the basis for this new dataframe.
* We made a new dataframe, “Inspection\_df”, with the “Inspection Type” column from our joined dataframe.
* In the Inspection\_df datafraame, we dropped duplicate rows.
* We then reset the index and specified column names, resulting in this dataframe:

****

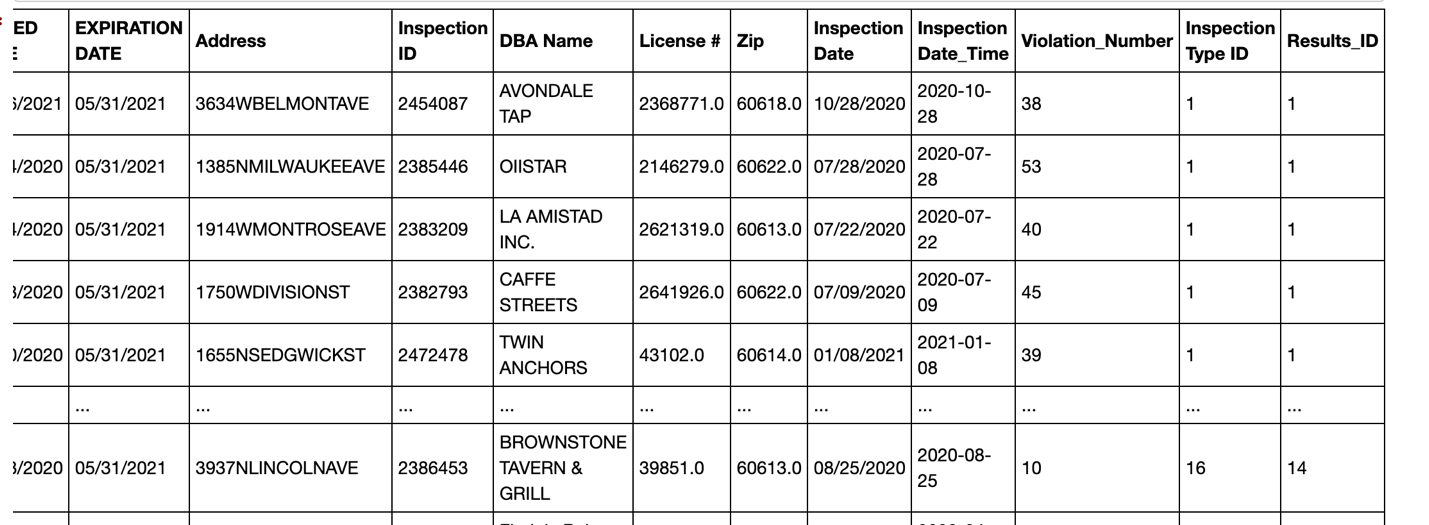
*Results\_df*

* We took the Results column from our joined dataframe as the basis for this new dataframe.
* We made a new dataframe, “Results\_df”, with the “Results” column from our joined dataframe.
* In the Results\_df datafraame, we dropped duplicate rows.
* We then reset the index and specified column names, resulting in this dataframe:

**

*Final Table*

* We updated our joined dataframe to be named final\_df and merged it on “Inspection Type” from the Inspection\_df.
* We then inner joined final\_df with Results\_df on the “Results” column.
* Lastly, we dropped any columns in final\_df that were duplicative (e.g. Results since we had Results\_ID). This resulted in our final dataframe:



**Type of Final Production Database:**

The four dataframes created above are stored in a relational database.

**Final Tables/Collections:**

We had four tables to load into our database, as follows:

* Final\_df
* Results
* Inspections
* Violations

In terms of connectivity, the following columns allow for table joins:

* The Results table connects to the Final table via the Results ID column.
* The Inspections table connects to the Final table via the Inspection Type ID column.
* The Violations table connects to the Final Table via the Violation\_Number column.

**Steps to Reproduce Process:**

By following this procedure, you will be able to recreate the technical process used to extract, transform and load this data.

1. Download the .csv files found by visiting the Sources listed above: <https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5> and <https://data.cityofchicago.org/Community-Economic-Development/Sidewalk-Cafe-Permits-Current/qnjv-hj2q>
2. Create a new Jupyter Notebook (.ipynb file) and save it in the same folder as the .csv files.
3. Import the appropriate dependencies (e.g. pandas, datetime and sqlalchemy) and the two .csv files into the .ipynb file.
4. Transform the data. See “Types of Transformation Needed” section above for a walk through of data transformation and the .ipynb file for the related code.
5. At the end of the transformation, you will have four dataframes as listed above. In PGAdmin, create four tables with the same column names and types as exist in the dataframes.
6. Using SQLAlchemy, load the four dataframes into tables. Once complete, you can perform SQL commands on these tables to get the data you would like related to City of Chicago restaurant inspections and sidewalk café permits.