NYPL Menu Dataset Analysis Report

Team ID: Team69
Name(Student ID):
Anthony Ravnic (aravni2@illinois.edu),
Kumji Park (kumjip2@illinois.edu),
Mark Synowiec (ms217@illinois.edu)
Date: 7/25/25
Course: CS513 Theory & Practice of Data Cleaning

Box folder: https://uofi.box.com/s/k8u68lz85e468ezj9dq39wxqpu7kd1ek

---

Introduction

Phase II of our Project centered around answering our U1 use case:

A historian may want to take a look at the vast collection of menus and see how prices have changed in the US over time in the 1800s and early 1900s. He may also want to see this broken out by state or city.

While the queries themselves seem somewhat easy, it does require a substantial amount of cleaning on several fronts

a. Key violations between menu items, menu pages, and menus
b. Incomplete/inconsistent place data, specifically US cities/states
c. Incomplete/inconsistent pricing data can lead to bad averages/medians
d. Potential Gap years where data wasn't available in a specific state or region
e. The date column contains dates far into the future or past that would be considered bad data

This cleaning of datasets menu, menupages, and menuitems (dishes were not used and thus left alone) was processed using several tools and steps. There were additional steps to both map the processes and load the data into SQLite for the queries necessary in our U1 use case.

The following is a list of the high-level steps used to fulfill our U1 requirement:

1. Use openrefine to clean/parse Menu and MenuItems datasets for bad data, outliers, and categorization of places
2. Load both raw and cleaned datasets as tables in a SQLite database

3. Remove all IC/foreign key violations in the underlying tables
4. Generate queries to quantify the changes made both on the cleaning portion and the removal of IC violations
5. Create queries to answer the U1 case. Use similar queries to compare raw results vs cleaned results
6. Create visualizations of U1 case
7. Use YesWorkflow to show data lineage/provenance of both the overall workflow and the inner data cleaning workflows.

After successfully completing these steps, we were able to get quantifiable results and comparisons between the cleaned and raw datasets.

The below sections give more details on the steps listed above.

1. Description of Data Cleaning Performed

The menu.csv file contained the columns necessary to clean to meet the data integrity needs of our use case, which was to look at how average menu prices changed year by year in the United States. To accomplish this, the place column was duplicated as place_cleaned and values were grouped into “RAILROAD”, “CITY, STATE”, and “UNKNOWN” values to standardize the data. The date column was duplicated into year, then to the year_cleaned column with outliers removed to contain only valid years. The data was also filtered on the dollar value in the currency column and saved to remove unrelated entries.

The menuitem.csv contained columns necessary to link price to the menu field dimensions, and we needed the price column to be accurate in order to fulfill our U1 case. In order to accomplish this, we removed numerous price values of $0.00 or blanks, as these were nonsensical values for the items.

Rationale:

Our use case was to clean a data set so a historian could start by looking at dates of the menus from the U.S. to see how dishes within them changed in price throughout the 1800s and 1900s. The cleaning described above was necessary to achieve our U1 use case for the following reasons:

- Foreign Key/IC violations were present between menus, menu pages, and menu items. Filtering out menus outside of the U.S. was performed to reduce the number of key mappings between these tables. Additional foreign key violations were removed to clean up menupages and menuitems. By removing key violations, orphan records are removed, and the underlying datasets become cleaner and more straightforward. Orphan records break the link needed to group and clean prices correctly. IC violation removal occurred at the end of this process, as we needed a cleaned menu dataset as

the parent table and the basis to remove all erroneous records from menupages and menuitems
- Place data was parsed/cleaned to get more accurate representations of cities and states. This allows us to break out multiple state categorizations and validate locations within the U.S., increasing the number of valid datapoints used to calculate average price for our U1 case.
- Place data was also categorized into “unknown” or “Railroad” categories where befitting. This data was less useful for our U1 case, but still helpful in calculating national menu price averages and comparisons in menu items that were “stateless”
- Currency data was cleaned/parsed to cross check that only locations within the U.S. were used and only prices in the correct currency were compared. This allowed for pricing to be accurate at the state level and removed non-comparable currencies/prices from affecting overall averages.
- Dates were simplified to years, and outliers were removed to ensure only known correct years would be included in the comparison data. Additionally, outliers, nulls, and bad formats in dates can cause misinterpretations of the data and lead to incorrect conclusions. Simplifying to years provided larger groupings of price data to get better averages. Blank, outlier, and implausible dates end up in the wrong time bins, creating false spikes or gaps in the trend lines.
- Cleaning of the currency and place columns was necessary to ensure prices were both of the same currency and within the United States. This data, coupled with the date column cleaned to represent the year, allows the prices in dishes to be consistent and represent the desired use case.
- Prices of $0.00 were removed in the final SQL queries as opposed to pre-cleaning steps before load. This was done to preserve measurements/counts of ic violations, but removed to preserve averages across the years (most of these were for unknown or railroads but some cities as well). Price values above $0.00 were kept as they could represent true prices in the early years of menus.

Overall, our phase II cleaning process aligned almost exactly with our phase I estimation. This includes: cleaning of broken foreign key relationships, unparsed/bad place data, implasible/outlier dates, missing/bad currency values, $0 or blank price data, and inconsistent location/place naming.

Two initially planned cleanings in Phase I were not needed.

1. Inconsistent dish naming - We didn’t need to use dishes to achieve our U1 case. This is also why we didn’t remove dish.id → menuitems.dish_id ic violations, as it wasn’t pertinent to our use case.
2. Null prices - There were no null prices tied to our cleaned menu dataset. As a secondary point, sqlite queries will ignore null values when calculating averages.

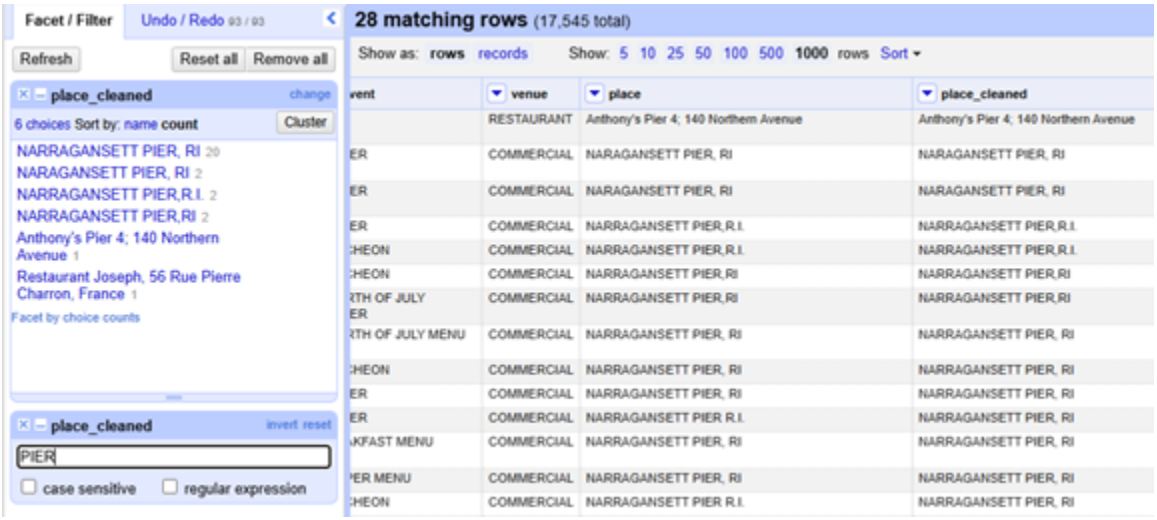
Detailed Description of Data Cleaning Performed

MENU

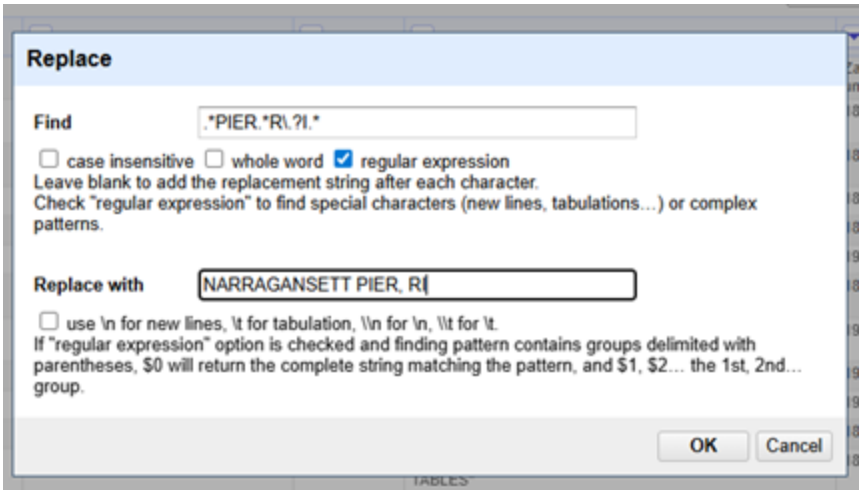
The place column contained the location data needed to determine the city and state associated with each entry. This column was duplicated as place_cleaned, and this second column was cleaned to standardize the data.

There were many entries that were for railroad lines instead of specific locations. For these entries, the places_cleaned column was changed to RAILROAD. Certain keywords were used to identify this category of entries: EN ROUTE, DINING CAR, and CAFE CAR in the place column; RAILWAY, RAILROAD, LINE, R. R., DINING CAR, and COMPANY in the location and sponsor columns; or in some cases, when an entry in the location or sponsor columns was identified as a railroad company but there weren't clear keywords in other columns other than the railroad company name then those other entries were classified as RAILROAD in place_cleaned. An example of this is for SOUTHERN PACIFIC, where the keyword DINING CAR identified the entry as RAILROAD. Other entries that contained SOUTHERN PACIFIC in all three of the sponsor, location, and place columns could be identified as RAILROAD as well.

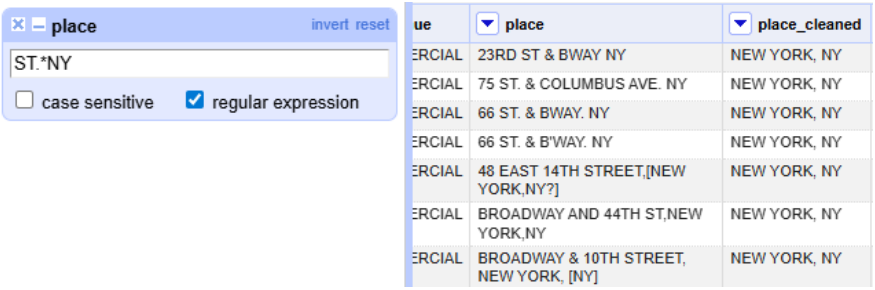
The place column also contained entries containing both city and state names. Frequently, these would have varied spellings, punctuation, and abbreviations. After text facets were used to group similar entries, a text filter was used to spot-check cities to ensure all versions of the city were grouped. The following image shows an example where more versions need to be grouped.

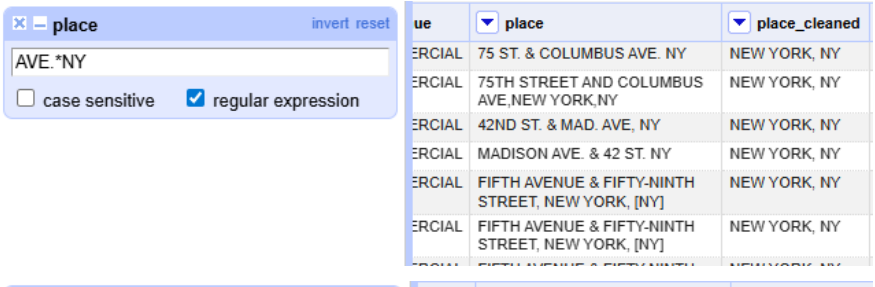


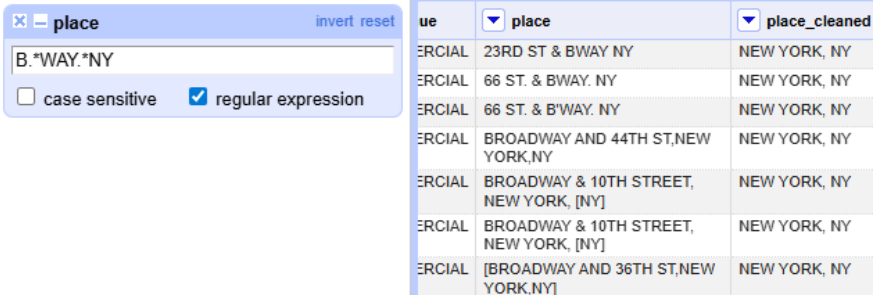
After excluding entries that do not match the desired city/state, a regular expression replacement could be used to manually perform a text facet grouping.



Instead of containing city and state, several entries contained a restaurant name or street address followed by a variation of NY, NYC, or NEW YORK. For these entries, it was assumed that the entry resided in NEW YORK, NY since it is more reasonable to assume the street or restaurant would be entered with the city instead of the state, since this would be more meaningful. In these case,s a regex text filter was used to quickly group and replace the place_cleaned column with NEW YORK, NY. Some examples of this are included below.







Prior to replacing the filtered entries, they were reviewed to ensure all entries matched the desired result. This approach was performed near the end of cleaning so it was uncommon to see entries that didn't match properly.

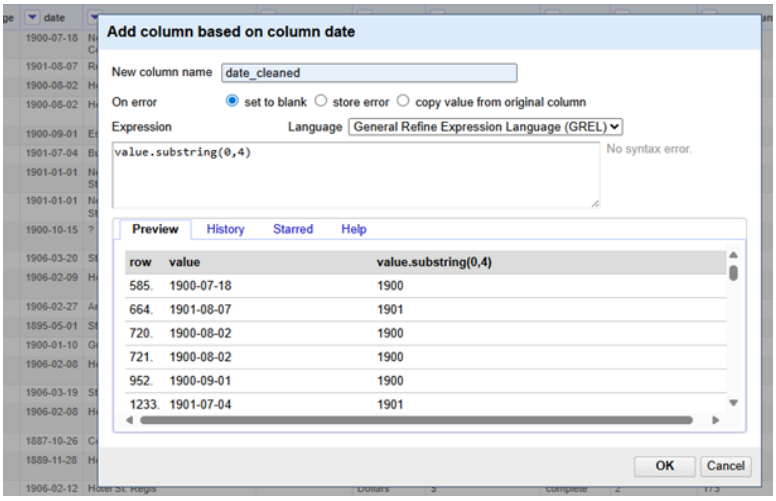
All entries that didn't fit into RAILROAD or CITY, STATE groups were then set to UNKNOWN in the place_cleaned column.

The currency column was used to filter data, we know to use the correct currency - Dollars. In this column, most of the Dollars data was correct, but a few rows that contained Dollars were clearly located outside the United States. There were two table entries in Yokohama, Japan, and three entries in England that contained the currency Dollars. For these entries, the currency was cleared to remove them from the set we needed for the use case. There were also entries from Canada that were labeled with Dollars, which were changed to Canadian Dollars. These entries were easy to find by filtering the list by Dollars in the currency column after the variability of the places_cleaned column was reduced.

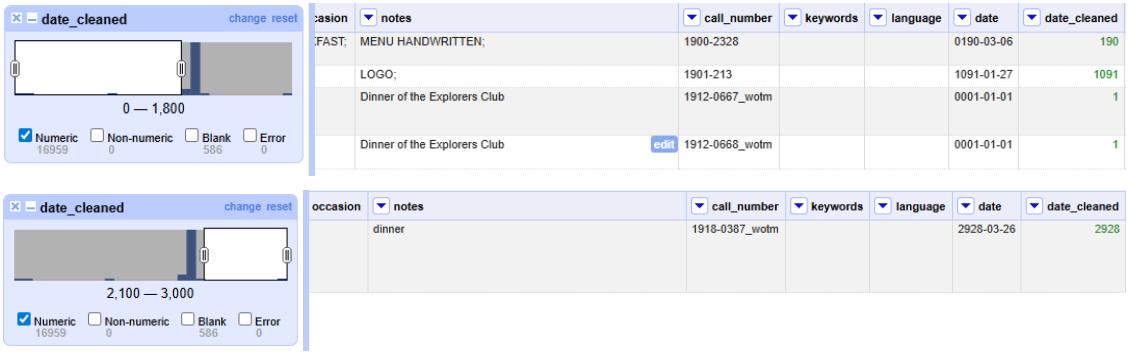
The file was saved with the text filter for currency set to "Dollars" to remove all entries that did not apply to our use case. At this point, this included all non-cleaned entries that did not contain a valid CITY, STATE, RAILROAD, or UNKNOWN entry.

After performing cleaning on the place and currency columns and saving a current version of the CSV and JSON files, OpenRefine struggled to accept commands and eventually halted. After restarting the application, I was able to continue cleaning by opening the cleaned CSV file and continuing with changes.

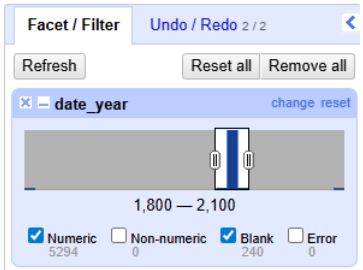
A date_cleaned column was created by parsing out the first four characters of the date.



Once the column was created, the only other step needed was to use a numeric facet to remove outliers. The following screenshots show the outliers that were cleared.



The following shows the valid values selected to remove the outliers using the numeric facet. This resulted in only years between 1800 and 2100 to be present in the cleaned table.



MENUITEMS

MenuItems contained blank and $0.00 price values that needed to be removed from the overall. Cleaning involved changing blanks to $0.00 and then filtering out $0.00 price values as shown in the recipe below:

Custom facet on column price

Expression Language General Refine Expression Language (GREL)

```
isBlank(value) or value <= 0
```

No syntax error.

Preview | History | Starred | Help

row	value	isBlank(value) or value <= 0
1.	0.4	false
2.	0.6	false
3.	0.4	false
4.	0.5	false
5.	0.5	false
6.	0.1	false


```
  "columnName": "price",
  "expression": "grel:if(isBlank(value), 0, value)",
  "onError": "keep-original",
  "repeat": false,
  "repeatCount": 10,
  "description": "Text transform on cells in column price using expression grel:if(isBlank(value), 0, value)"
}
```

```
{
  "op": "core/row-removal",
  "engineConfig": {
    "facets": [
      {
        "type": "list",
        "name": "price",
        "expression": "value",
        "columnName": "price",
        "invert": false,
        "omitBlank": false,
        "omitError": false,
        "selection": [
          {
            "v": {
              "v": 0,
              "l": "0"
            }
          }
        ],
        "selectBlank": false,
        "selectError": false
      }
    ],
    "mode": "row-based"
  },
  "description": "Remove rows filtered price value 0"
```

Removal of IC Violations

Once these cleaned datasets were loaded into the database, we could then begin removing IC violations. Initially we attempted to do this using foreign key constraints and ignoring any violations from coming into the database. However, SQLite kept crashing and removing extra rows, so we decided to remove IC violations by transforming the cleaned tables. An intermediate table cleaned_menuitems_w_ic was retained to calculate the changes due to cleaning and then due to ic violations. These can be seen in the below sql queries:

```
--create cleaned data structures from old (unformatted tables). Menu and menuitems were cleaned and thus need to be loaded manually
create table cleaned_MenuPages as  select mp.* from menupages mp inner join cleaned_menu cm on cm.id = mp.menu_id;



--MENUITEMS price column was scrubbed to bring total from 1335255 to 888707, saving these with ic violations to separate interim table for later ic violation counts
```

Then ic violations from cleaned_menuitems_w_IC were dropped and loaded into the final table cleaned_menuItems. This was done because we needed to preserve both the cleaning rows removed and the ic violations for diagnostics in the next sections

```
--remove IC violations from cleaned_menuitems_w_ic and create final cleaned_menuitems table
create table cleaned_menuitems as select mi.* from cleaned_menuItems_w_IC mi inner join cleaned_menuPages mp on mi.menu_page_id = mp.id inner join cleaned_menu m on m.id = mp.menu_id;
```

---

2. Documentation of Data Quality Changes

Diagnostic Comparison of RAW datasets to Cleaned:

After cleaning and refining the raw datasets via the processes above, we noted a marked improvement in the quality and accuracy of the analysis. First, we review the changes in each dataset.

Menu Changes

By first cleaning Menu, we saw the following changes from Raw to Clean:

original_menu_rowcount	cleaned_menu_rowcount	Num_rows_removed_via cleaning
17562	5532	12030

These high-level changes were due to filtering

Column Cleaned	Description	Number of Cells Cleaned	Resulting Format
Currency	Filtered out non-dollar currencies	12,013 removed	Currency = ‘Dollars’
** place copied to place_cleaned	Standardize formatting to location information	1,371 converted	RAILROAD, [CITY, STATE], UNKNOWN
currency	Modify or clear “Dollars” entries outside the U.S.	15 removed	“Dollars”
date copied first to year then to year_cleaned	Convert to year and remove outliers	5,531 converted 2 removed	4 Digit Year

****Note:** The place column was not removed/filtered only parsed and converted into a separate clean column.

Also of note, **12013** removed currencies +**15** removed currencies outside of us + **2** removed dates = **12030**, the number of removed rows in cleaned_menu.

No IC violations occurred as Menu was a Parent table so only cleaning was performed.

MenuPages Changes

Menupages acts as a sort of bridge table between menu and menuitems to connect place/year to price, the ids between them were only used to connect the tables for finding avg price for our U1 case. Thus, only IC violations were removed between the original raw menupages table and the cleaned_menupages table.

original_menupages_count	cleaned_menupages_rowcount	num_ic_violations_removed
66,937	16,704	50,233

MenuItems Changes

MenuItems had both cleaning removals (due to price issues) and also IC violation removals (due to it being a child table to menupages). Since cleaning occurred before IC violations were analyzed, we summarize these in the following tables first:

MenuItem.csv price	Remove rows that are zero and empty	**446,548** rows removed	Number format (unchanged)

Original menuitems rowcount	Cleaned menuitems rowcount (before IC Violations)	Num rows removed
1,335,255	888,707	446,548 **(33.44%)**

The **446,548** rows removed are broken out into their respective cleaning steps:

blank_price_rows_removed	zero_price_rows_removed	**Total Removed**
446,185	363	**446,548**

As you can see, the two scenarios (blank price, and $0.00 price) add to the total.

Afterwards, the table was loaded into SQLite as an interim table cleaned_menuitems_w_IC. This was then scrubbed of IC violations to create the final table cleaned_MenuItems. The row difference between these tables gives you the total IC violations removed.

Cleaned_menuitems_w_ic rowcount	Cleaned_menuitems rowcount	num_ic_violations_removed
888,707	796,289	**92,418**

Thus **1,335,255** initial rows - **446,548** removed rows - **92,418** IC violations = **796,289** Final Cleaned rows in cleaned_menuItems

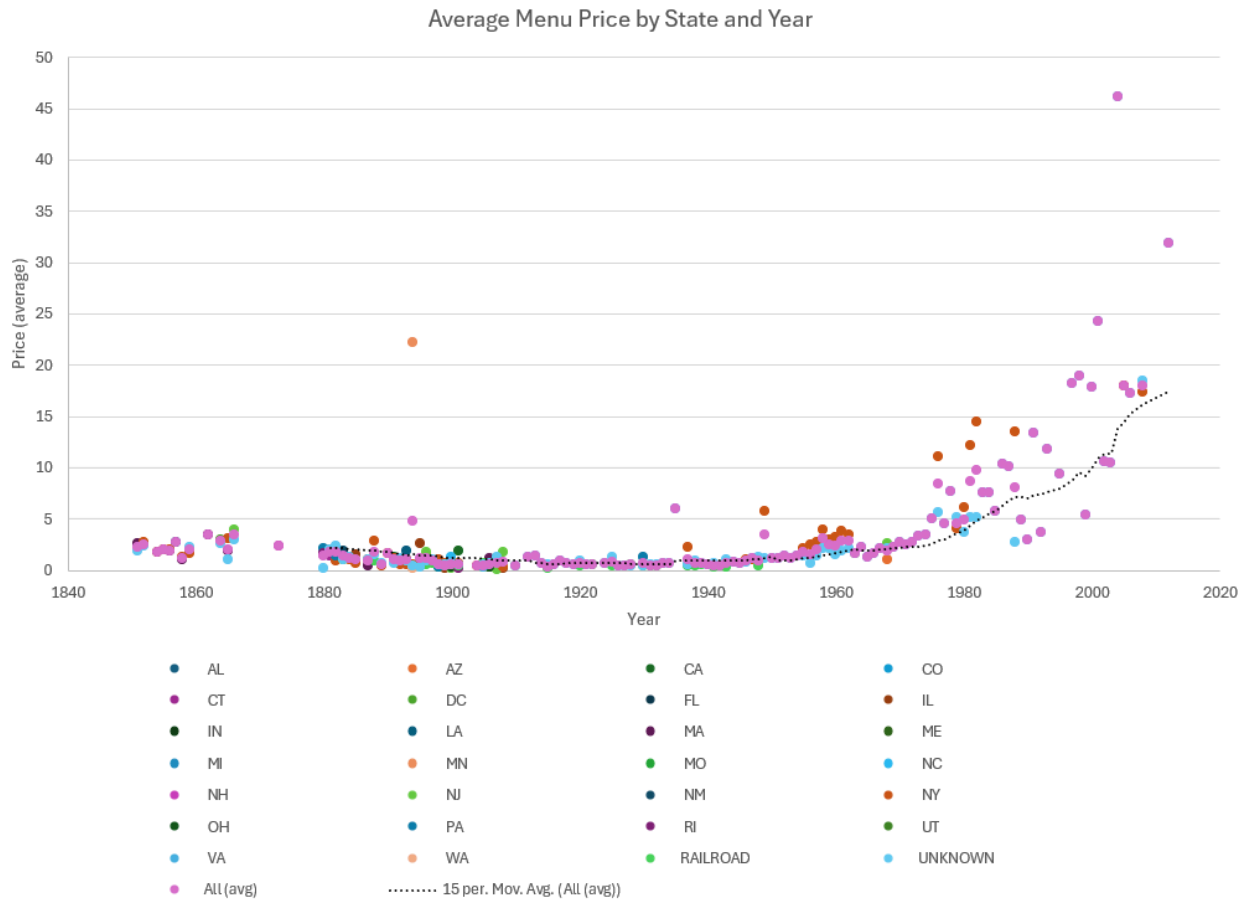
Dishes

Dishes were not used in our U1 case and thus, no cleaning or ic violations (of menuItems) were removed. However, it is mentioned that we could use Dishes for additional use cases in the “next steps” section below

Data Quality Improvements

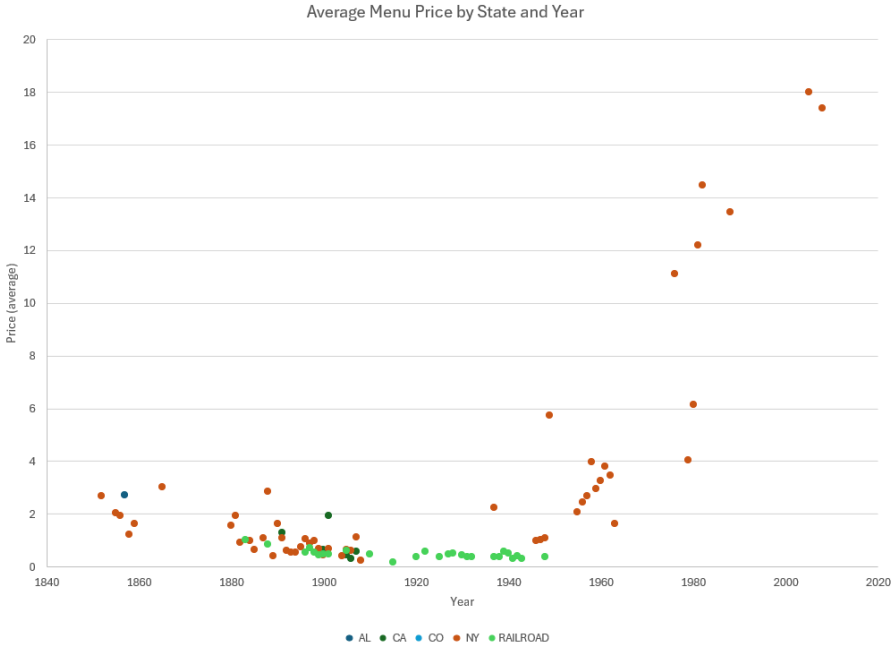
As mentioned, by including rigorous cleaning and parsing of the data, we were able to increase both data accuracy and data quality.

By parsing place data into the various States, Unknown, and Railroad categories (cities became too granular and sparse), we were able to get detailed visualizations of the data by category

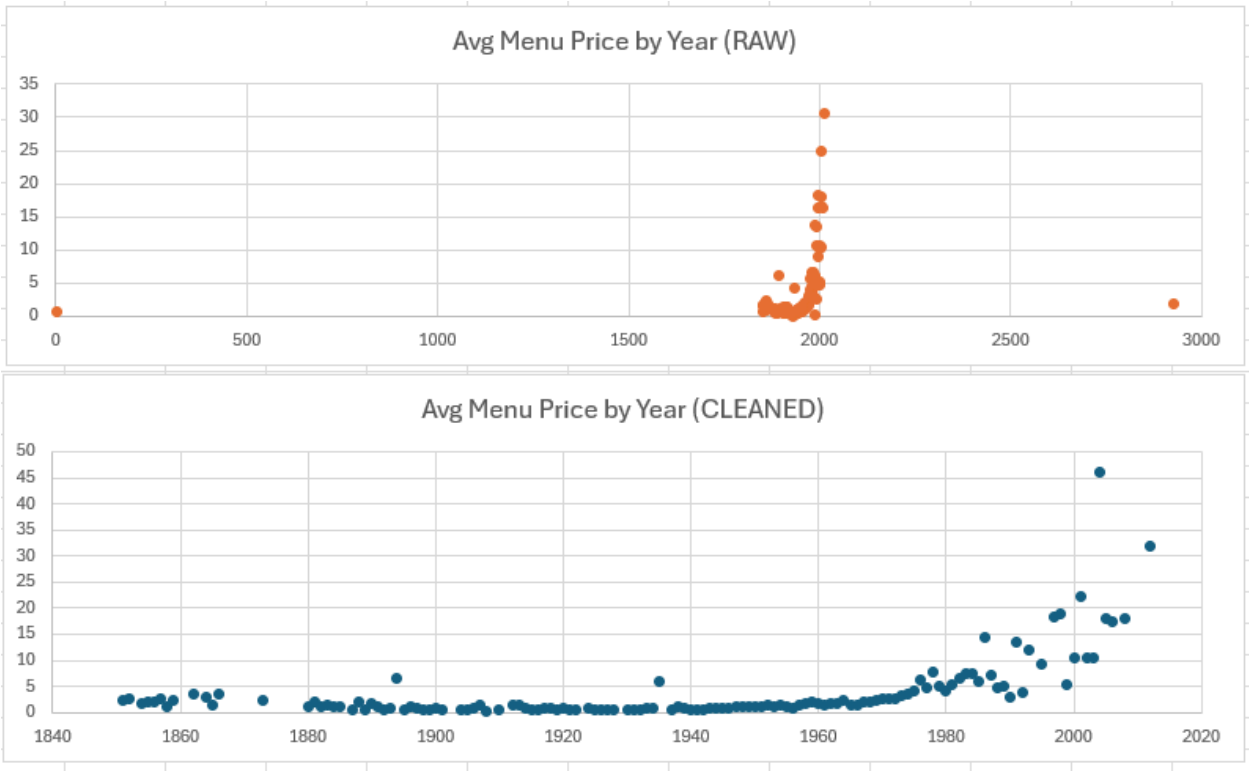


Looking at the above figure, we were able to parse out 29 different categories along with 26 states. Additionally, we were able to create moving averages on overall prices across the years (black dotted line).

We included an “All” average encompassing all categories combined to show the overall trend. We included this due to sparse data points across the years for most of the states, as seen below. Railroad and Unknown categories are also shown.

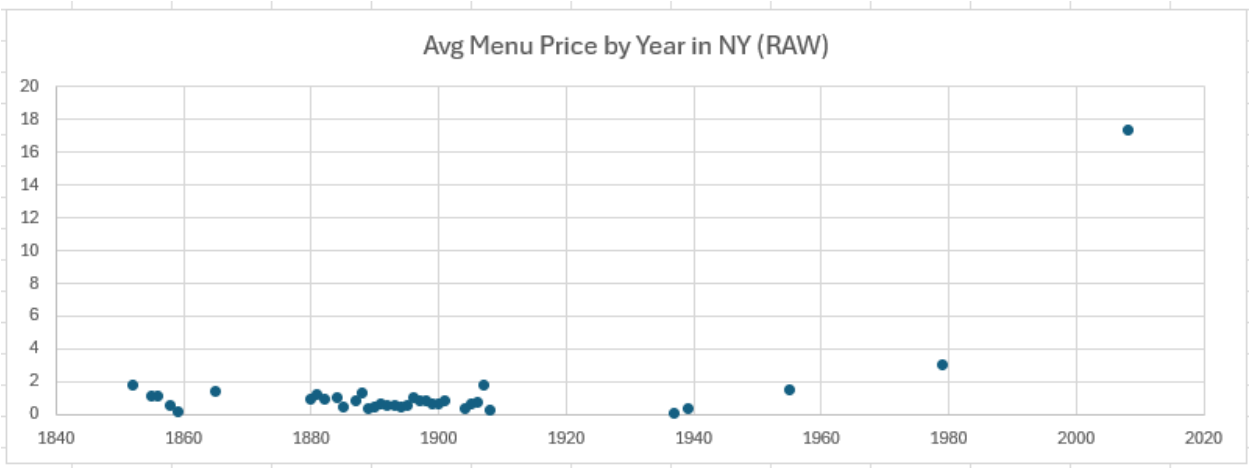


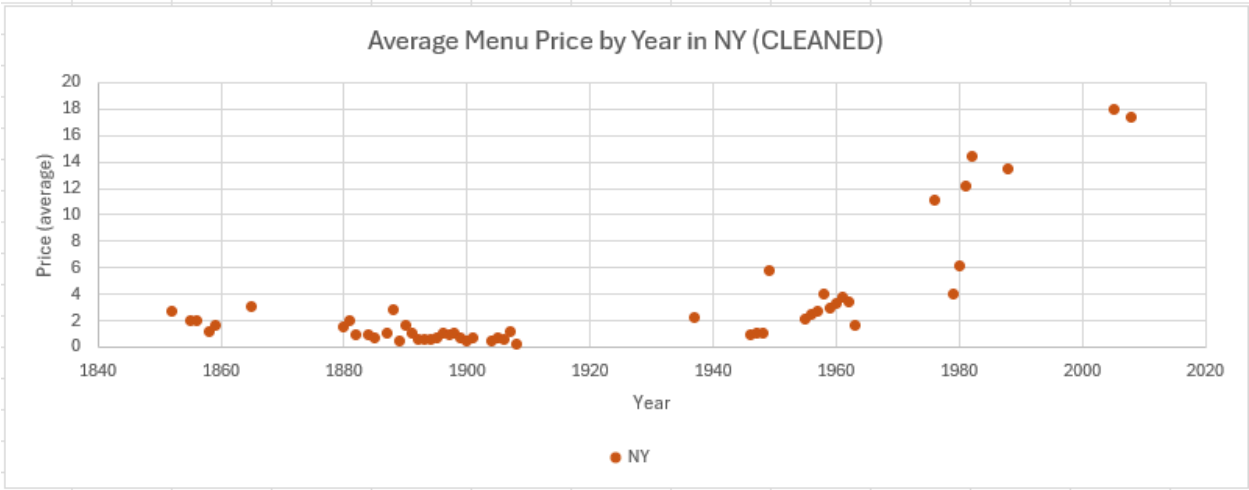
Comparing the average price by year (all categories) between the raw dataset and the cleaned you can see a marked improvement:



Without removing date outliers, the heart of the above graph gets squeezed between to improbable dates. Removing $0 and blank prices also causes the true average prices to be visualized in the below graph (~$31 vs $46 for the highest data points).

Due to unparsed place data in the raw menu dataset, this is as granular as you can get without using a “like” statement to categorize, as best you can, the various states. This means there is no way to represent states accurately without 100’s of like statements. For comparison, if we were to use a like ‘%NY%’ or ‘%NEW YORK%’ statement to compare raw to cleaned, we would see the following graphs.





Note that there are far more data points in the below cleaned dataset. This tells us there are many more versions of NY we would have to add to a like statement to ONLY get NY as a category. While the years in the raw set seem to be reasonable, the values of the data points in the cleaned set are more accurate (and higher) due to the removal of blank and $0 pricing.

There is also something to be said for removing IC violations, too. Although these orphan rows wouldn’t have made it into the final queries

As mentioned above, cleaning the original NYPL datasets allowed us to accurately visualize and fulfill our U1 use case. A summary of major improvements was found in the following:

1. Categorizing place data allowed us to break out menu prices into 29 unique categories and 26 states. It also allowed us to identify further areas of investigation (the “Unknown” place category can be further parsed).
2. Removing non-dollar currencies and converting currencies to dollars for places within the U.S. allowed us to focus more on U.S.-relevant records
3. Cleaning pricing data revealed a significant increase in average prices in the U.S over time. This correlates well with inflation.
4. Converting the Date column to year and removing outliers helped to group pricing throughout the years and give more accurate time range representations

---

3. Create a Workflow Model

To document the end-to-end data transformation and cleaning process, we created both outer and inner workflow diagrams using YesWorkflow and OR2YW. The outer workflow (W1) depicts the stages from raw ingestion to final query-ready datasets, while the inner workflow (W2) focuses on OpenRefine operations including cell transformations, filtering, and column derivation. The annotated workflow allows us to trace how each key output was derived, promoting reproducibility and transparency in the cleaning pipeline.

Outer Workflow (W1): End-to-End Data Cleaning Pipeline

The outer workflow captures the complete sequence of our cleaning project. It includes the following stages:

1. Cleaning Menu.csv in two steps:

Using OpenRefine, we cleaned the ‘place’ and ‘currency’ columns, created derived columns like ‘place_cleaned’ and ‘year_cleaned’, and standardized location naming across records.

2. Cleaning Menuitem.csv:

We removed price values that were zero or missing using a Python script. This allowed us to compare pre-/post-cleaning price distributions. However, the impact of this cleaning step was not visually distinct in the histogram plots. So we revisited OpenRefine later to explicitly transform blank values in the price column to 0, remove rows where price is 0, and then save the updated state to export a JSON recipe file for inner workflow modeling.

3. Loading into SQLite & Removing IC:

Cleaned datasets were loaded into SQLite, where further integrity constraint(IC) violations were addressed using SQL queries. Although ‘menuitems’ -> ‘menupages’ -> ‘menus’ form a clean

FK chain, we discovered indirect IC violations and numerous orphaned entries. These were removed to produce four cleaned tables.

4. Cleaning Diagnostic Summary:

Queries were constructed in sql to compare the number of rows dropped from the original datasets (menu and menupages) and their cleaned versions. The number of rows dropped

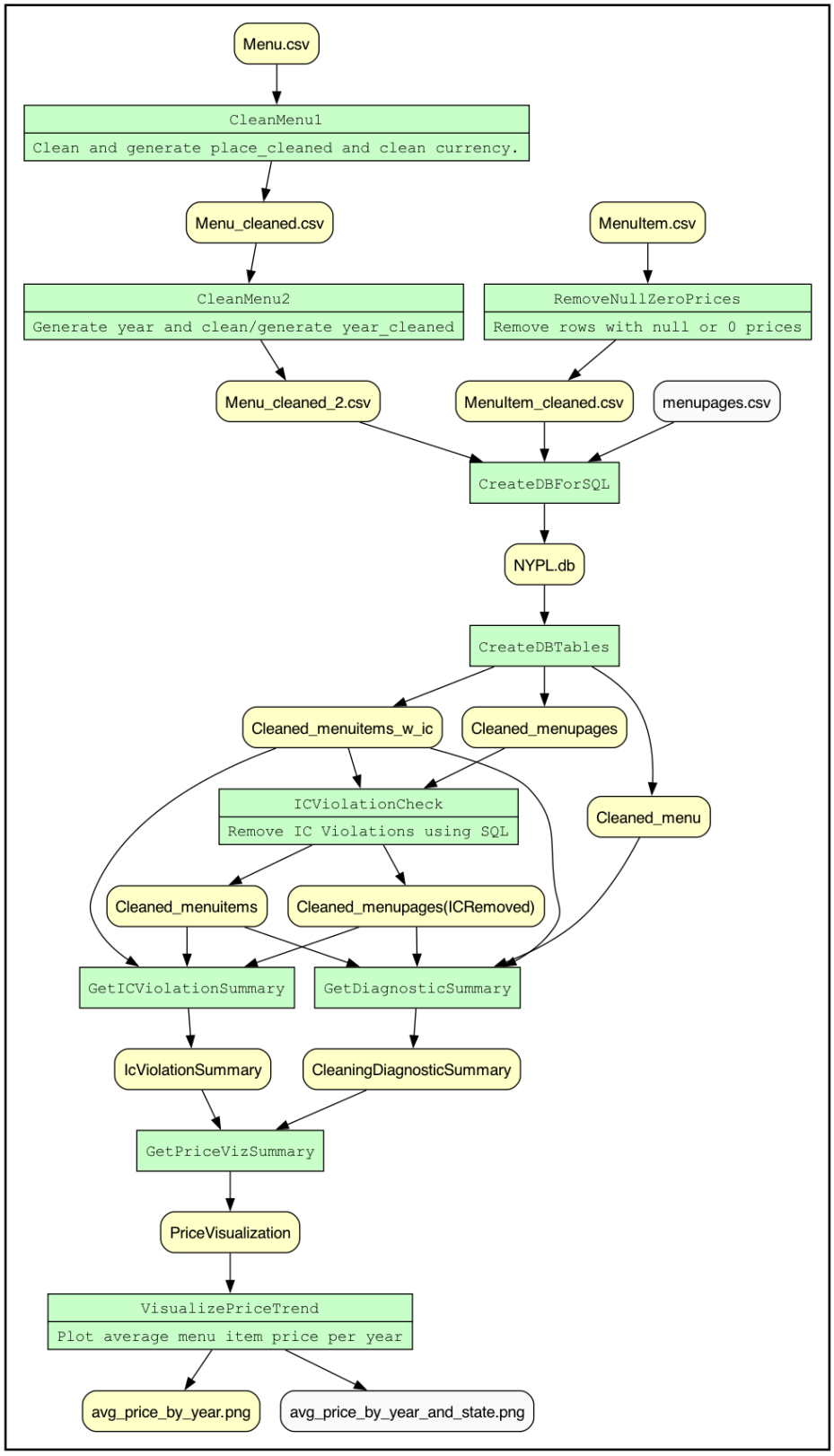
5. IC Violation Summary:

Queries were used to compare the number of rows dropped between the original menupages and menuitems datasets and their cleaned versions.

6. Final Visualization and U1 fulfillment:

Using the cleaned and validated database, we ran analytical queries to support our U1 use case. Two key visualizations were generated: ‘avg_price_by_year.png’, ‘avg_price_by_year_and_state.png’

W1_Outer_Workflow



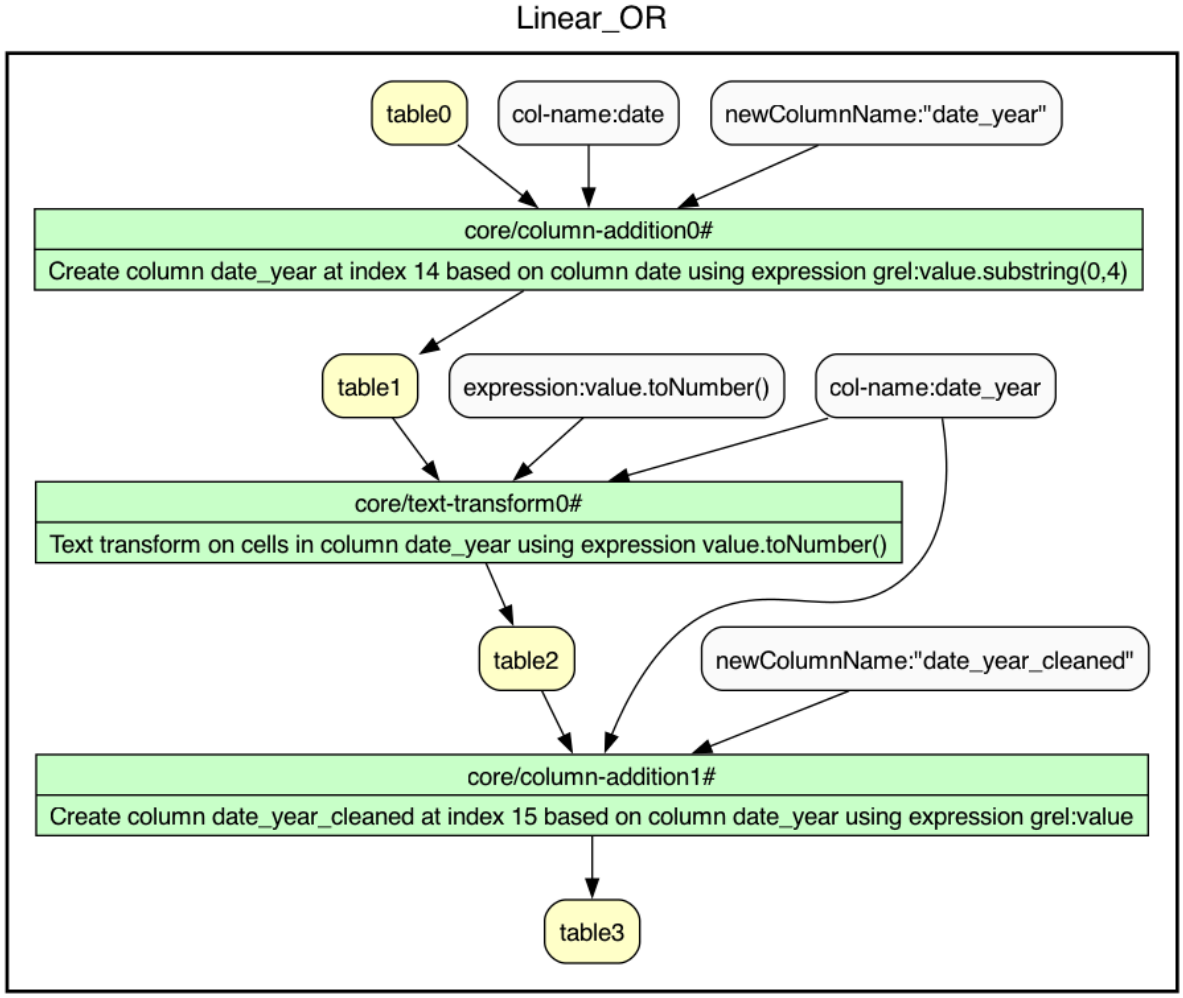
Inner Workflow (W2): OpenRefine Operations

The inner workflow focuses on the sequence of data transformations within OpenRefine. To represent this, we used OpenRefines' history and exported it as a JSON recipe file. And then the JSON file was converted into a diagram image(PNG format) using OR2YW.

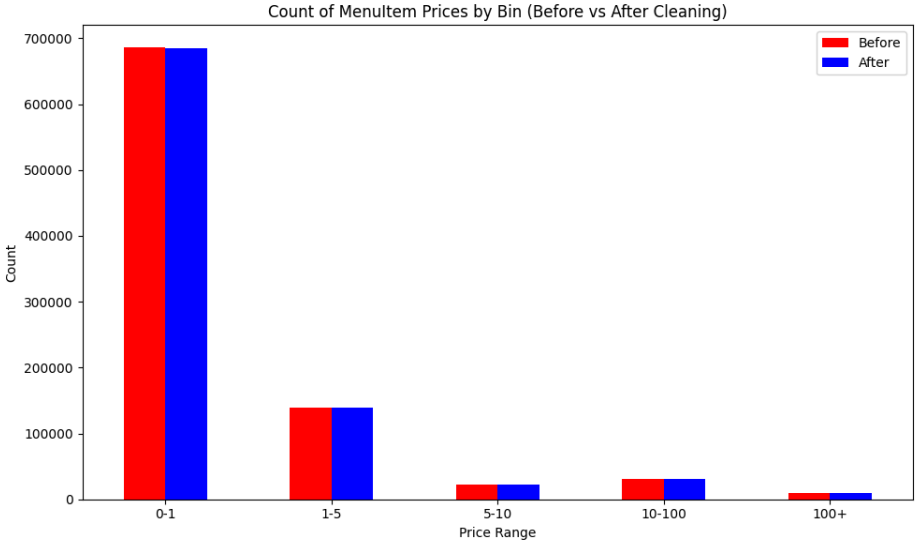
We have three different inner workflows:

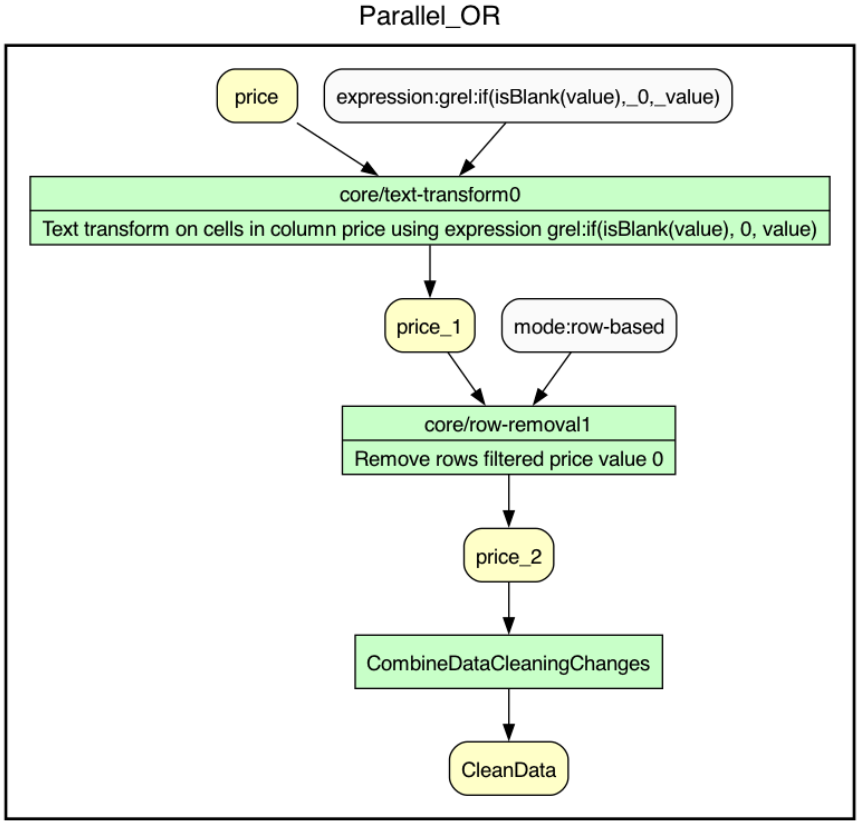
1. Cleaning Menu workflow(Clean and generate place_cleaned and clean currency)
   The OpenRefine recipe file contained over 300 transformation steps, making it too complex to visualize directly. To enable graph generation, we simplified the workflow and will submit the resulting png file separately. You can also see a picture of this in the appendix at the end of this report

2. Cleaning Menu 2 workflow(Generate year and clean/generate year_cleaned)



3. Cleaning MenuItem(Remove rows with null or 0 prices)
   As described in the outer workflow, we initially used Pandas to generate plots for before-or-after comparisons. However, when the visual difference was minimal, we revisited OpenRefine to refine the cleaning process and achieve clearer visual results.





4. Conclusions and Summary

Findings:

In addition to the cleaning improvements found in the “data quality improvements” section, the following general findings were also noticed:

1. For locations and prices in the U.S., data quality and accuracy were significantly improved via cleaning when comparing the raw datasets to the cleaned, in almost every dimension used. (see above sections)
2. The dataset was heavily focused on New York (for currencies within the US), and other states were sparse with data. Given the limits of collection, there isn’t much we can do to improve this
3. The ‘Unknown’ category still represents a significant amount of entries in the U.S., and most are not categorizable to state based on the information in the underlying datasets
4. Even with the larger categories like New York, ‘Unknown’, and ‘Railroads’, data across the years was sparse/incomplete, but still enough to arrive at valuable conclusions in price increases for our U1 case
5. For the historian mentioned in our U1 case, there is likely a need for further analysis/investigation into the datasets (see next steps)

Lessons Learned

In total, there were more than 200 steps performed in the cleaning process. Most of those steps were required to clean the place column. Improvements to the workflow were understood as cleaning was performed:

- The main filtering column was currency since price data couldn’t be correctly averaged between currencies. Filtering first on currency == “Dollars” would have reduced the number of entries cleaned in the process since those were the only entries relevant to the use case. Furthermore, in the future, when looking at datasets, it would be useful to identify whether subsets of a dataset are only relevant and first reduce to the subset prior to cleaning. This isn’t universally true if there were situations where cleaning could deduce data missing to properly identify the proper subset.
- Regular expressions were used to clean around irrelevant characters, and often a few different expressions were required to handle different combinations of formats. Since we didn’t need to include or match against certain characters, those characters could

have been removed to simplify matching and reduce cleaning steps. The following characters could have been removed first: [ ] ( ) ; . "

- Initially attempting to load datasets (ignoring rows with IC violations) caused SQLite to crash. It appears there were additional rows that were removed, but due to a crash, it was impossible to check the logs to see why. Ultimately, we decided to handle IC violations via SQL statements. While the menu was a parent table and there were no cleaning steps in menupages, these were relatively easy. However, because MenuItems contained both cleaning and IC violations, and we wanted to track cleaning/IC removals, a separate interim table was used to count cleaned vs IC rows removed.
- Using OR2YW(Yesworkflow) helped us reflect on the structure of our overall cleaning process and communicate it clearly through visual models. And it encouraged us to think of our cleaning tasks in modular steps with clear inputs and outputs. We discovered that not all JSON export structures were compatible out of the box. Manual formatting of .yw files was necessary in some cases to ensure that they could be parsed and visualize the model correctly.

Problems Encountered:

- Initially, trying to load datasets with foreign key constraints (even after ignoring error rows) caused significant problems. After loading the cleaned_menu dataset, SQLite had several issues loading and removing IC violations from menupages and menuitems. This not only caused it to crash, but also removed more rows than anticipated. Due to it crashing with no logs, our initial counts of IC violation removals didn't match what was expected.
- Using the SQLite database added its own challenges. After uploading cleaned_menu, the row size was reduced to ~5500 records. When loading menupages and menuitems datasets with foreign keys (removing ic violations), it caused SQLite to crash. Additional rows were also lost, but due to the crash, it was impossible to tell what was removed. We instead took an approach of using the reduced clean_menu to populate these new cleaned_menupages and cleaned_menuitems with only the foreign keys pertaining to cleaned_menu, thus removing IC violations in the process.
- We found that even after rigorous cleaning, there were still a handful of bad records in the menu that we didn't account for. These included some blank rows for city/state and Null year values. However, by doing a thorough cleaning first, we were able to remove these handful of rows using SQL in the final queries. These issues would have been much harder to catch and a "drop in the bucket" of problems had we just used raw datasets.
- One of the JSON files (menu_cleaned.json) was extremely large and complex, causing a performance issue that we had to manually simplify to improve the clarity of the inner workflow diagram successfully. This experience emphasized the importance of planning for model tractability, especially when working with tools that generate large operation histories.

- Initially, we attempted to fully automate the workflow visualization by converting OpenRefine JSON export files into YW(YesWorkflow markup), then into DOT and PNG files using a Python script. However, this process faced multiple challenges that the exported JSON structure from OpenRefine was too nested and inconsistent for direct parsing, and the conversion to YW markup required manual intervention to correct formatting, indentation, and grammar because YesWorkflow CLI was too sensitive to syntax issues.

Possible Next Steps:

While we did achieve our U1 use case, some refinement or "follow-up questions" still remain. If we had more time/resources, we would likely drill down into the following:

- **Clean and refine pricing averages based on course (breakfast, lunch, and dinner or combinations thereof)**. Often, pricing between these items can be vastly different, especially if certain restaurants are dinner exclusive with higher ticket items.
- **Parse additional columns "event" and "sponsor" for more city/state locations.** We found that even after parsing the place for city/state, columns sponsor and event in "menu" held cities with no states (and vice versa) that were missed. Additional location-based data is contained in these fields that were not included in our cleaning and ultimately were labelled "UNKNOWN". By parsing this data further, we could add more data points to specific cities/states and improve our accuracy on pricing averages.
- **Clean and utilize the "dish" dataset to drill down and categorize average pricing by dish category.** Grouping by various main ingredient categories like chicken, steak, vegetable etc… was discussed as a follow-up analysis.
- **Create a dollar-adjusted average of pricing based on inflation rates over the years.** An interesting follow-up would be to price-adjust these menu items to account for inflation over the years and see how much items really increased on a dollar-adjusted basis.

Reflections:

- **Mark** played a key role in cleaning the raw datasets using **OpenRefine**. He categorized complex place names, handled inconsistent currency formats, and refined date fields to remove outliers. He also created scripts to help reduce openrefine recipes to create our yes workflow diagrams and make them usable. Mark also contributed a lot to the technical documentation around data cleaning in the report

- **Kumji** was instrumental in creating our **Yes Workflow** diagrams (both inner and outer) to document the overall cleaning pipeline and dependencies. She also assisted in other areas like cleaning the menupages dataset using openrefine, creating scripts used to convert yes workflow files into diagrams, and documentation in the report.

- **Anthony** took the lead on our team and provided valuable technical direction, and maintained clear communication throughout the project. He developed and executed SQL scripts to identify and resolve foreign key violations, and structured the cleaned datasets for analysis and visualization. He also helped with documentation around data changes in the report

All three of us contributed to the report documentation, both in writing and reviewing/revising the various sections. All three of us did an equal and massive amount of work in putting together both phase I and phase II

APPENDIX

YesWorkflow menu cleaning steps