

CS513 MaybeWorkFlow Final Project Report

By: Abhilash Jacob (Jacob11), Ishan Shah (ishanbs2), Nathaniel Rupsis (nrupsis2)

Project Repo: <https://github.com/MaybeWorkFlow/CS513>

Introduction and Overview

For the final project, the MaybeWorkFlow team chose to work with the New York Public Library's *What's on the Menu* dataset. The biggest reason behind choosing this dataset is to ensure "sufficient dirtiness". This data set allows us to focus on the tools and techniques taught throughout the semester.

It's worth noting that the dataset is updated on the 1st and 16th of each month. For our report, we're using a version updated on 07/01/20.

When starting this project, our team met up and decided on a couple sample questions we'd like to answer from this data set. A few questions we had were:

- Which Dishes are considered "Featured"?
- Do the average menu sizes increase, or decrease over time?
- What sort of metrics over time can we obtain? Average price of a dish over time, highest / lowest price, etc

We felt that having these questions in mind would not only help us shape the way we view the data set, but also direct us on the cleaning path. Once we developed our questions, we decided upon a divide and conquer approach for this project. Each individual was tasked with taking on 2 of the workflow phases, then we'd aggregating the results in a final deliverable document.

Initial Data assessment

The New York Public Library's crowd-sourced historical menus comprises four data tables with more than 1.9 million records. Each table has a primary key to uniquely identify elements. Menu, MenuPage, MenuItem and Dish are the four tables in the data sets. Menu table is a standalone table that describes various locations. Event and occasion in which the menu is served. It also provides sponsor's information and its

physical location. The MenuPage table contains the information about the menu page's image and its uuid. It also provides information about the layout dimensions. It has foreign key of the menu id where the menu table is connected. The Dish table is another standalone table that encompasses the dish names and its popularity based on the number of times it appeared in the menu. It also provides more vital data such as price and its first and last appearance in the menu along with times it appeared in the menu. MenuItem contains two relational keys for MenuPage and Dish tables. Other information in the table seems to be additional information for the MenuPage.

Based on the initial assessment, it is evident the data has an old history and is not cleaned. It has substantial integrity issues where many data elements reference to an incorrect data set. For example, many records for last and first appearance in the dish tables are out of range. Similarly, the lowest and highest price for dishes in record are zero. The loss of data is another found issue in the given data sets, many columns are completely unfilled or partially filled. For example, the description column in the dish table is completely unfilled and the occasion column in the Menu table is partially filled. Other issues found in the tables pertain to the stagnant or isolated data which does not make any relation to overall datasets. For example, the status column in the menu table and xpos and ypos columns in the MenuItem table.

The Data cleaning typically is use case driven. In overall, this data set requires substantial cleaning, we identified that finding the expensive dish and which dish is "featured" (displayed in first page) can be derived without having any data cleaning task as these data elements are well defined without having noticeable quality issues from an initial assessment perspective.

There are also certain use cases that cannot be fulfilled even after significant data cleaning. One of them is to compute the highest or lowest pricey dishes, because many data are zero in both the highest and lowest column entry. Either data might be incorrectly entered or lack precision while entering the data or lost over time. The other use case might be aggregated metrics (Mean / median Price, Avg price per year, prices for dishes year over year, etc) from menu prices would be biased. There is a lot of missing data, and the results wouldn't be all that accurate.

The main use case we are trying to solve this project is to provide analytics data points for popular menus in a specific event. This requires substantial cleaning effort in Dish and Menu tables. Other tables (MenuItem and MenuPage) also may require a moderate level of cleaning since it has the relational information about use cases. We acknowledge that even with substantial cleaning, we may not be able to fulfill perfect use case scenarios. Based on the initial analysis, we can achieve this use case with

multi levels clustering operation in the event column in Menu and name column in Dish table along with standard cleaning process and standard cleaning process for MenuItem and MenuPage.

Data Cleaning Methods and Process

One of the best approaches for large data sets is use case-based cleaning. In use case-based cleaning, the data set is cleaned in accordance with use cases, where the data cleaning is performed to specific columns or subset in a large data set. New York Public Library's crowd-sourced historical menus is a larger data set. It is not practical to perform a deep data cleaning in all elements in the tables. Hence, our approach is to go with case-based cleaning. The use case defined for this project is "popular menus in a specific event". First, we identified tables and columns/elements relevant to the use case that defined and then identified appropriate tools based on size of the tables. In these data sets we identified that Dish tables and MenuItems are huge and cannot be performed with OpenRefine intuitively without multiple partitioning, we decide to go with cloud platform to perform the data cleaning for such tables. However we identified that MenuItems does not provide any meaningful data for our use case other than link to other tables, hence we decided to remove MenuItems table from further cleaning process. The Menu and MenuPage tables are comparably smaller in size than the Dish and data cleaning is performed in the OpenRefine tool without any partitioning.

We established the data cleaning process started with standard cleaning (white space, proper format, etc..), removing the unnecessary special characters, clustering the values appropriate for the specified use case and finally performed the integrity constraints (IC) for the selected columns based on the use cases defined. We identified and resolved a few ICs in the Dish table.

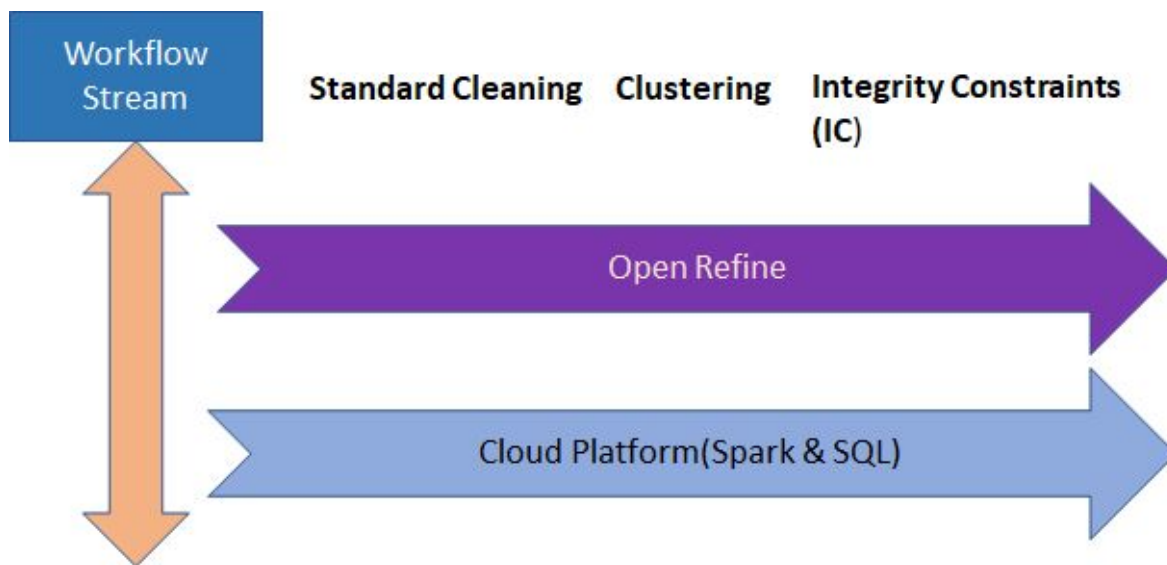


Figure 1: Data Cleaning Snapshot

OpenRefine Data cleaning

In OpenRefine, we started with standard cleaning for all the data sets of Menu and MenuPage, where leading and trailing and consecutive whitespace are trimmed, converted to an appropriate case, convert number to number format and convert date to ISO format without time formation. Later stage we have performed the advanced cleaning that includes multilevel clustering, facet edition and removal of special characters from the selected columns based on use case by using GREL scripting. To keep the provenance of the data while performing the multi clustering, separate columns are created before a deep level of multi clustering operations is performed. None of the columns are deleted to keep provenance as well as future work purpose even if data is not relevant to use case or completely unfilled. None of IC's identified in Menu or MenuPage tables. Following are the detailed descriptions of various operations performed in each column or column level operations.

Standard cleaning process in OpenRefine:

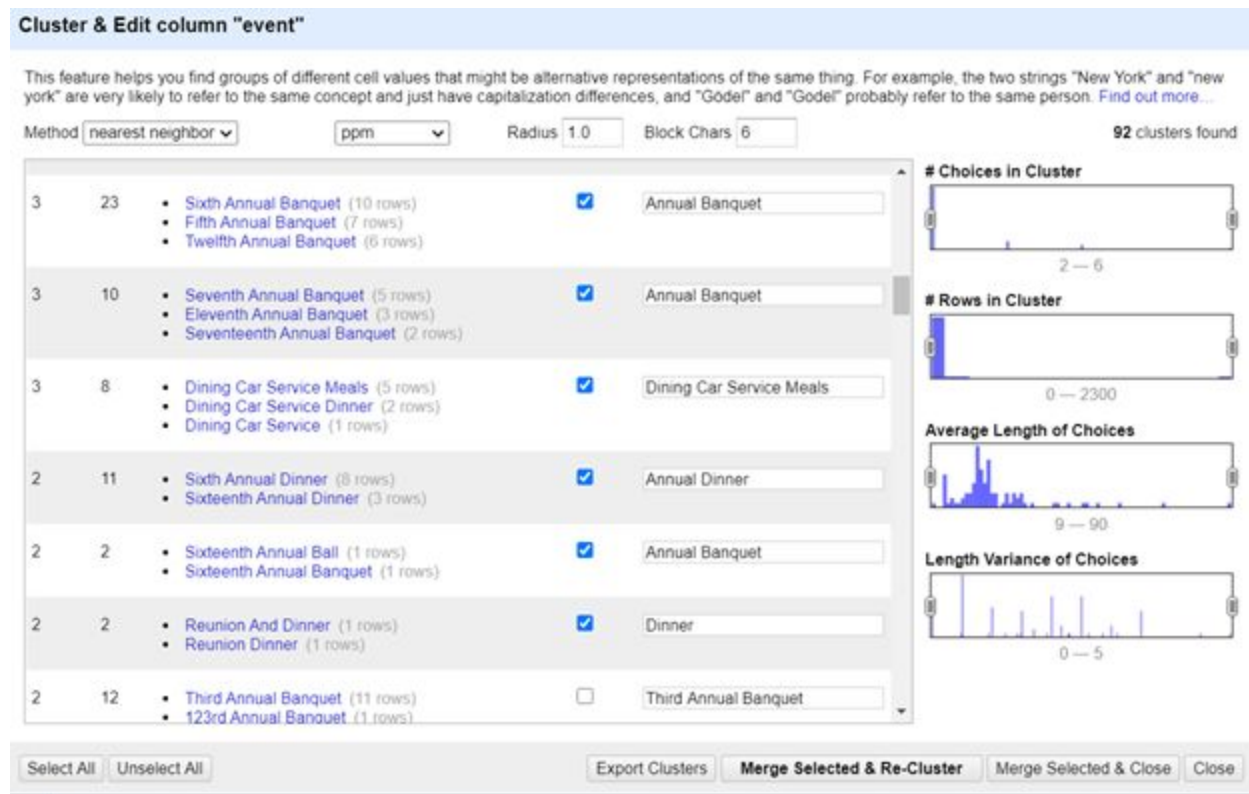
Operations	Columns	Table Name
	<ul style="list-style-type: none"> • Id • menu_id, • page_number 	Menu Page

Convert to Numbers	<ul style="list-style-type: none"> • image_id • full_height • full_width 	
	<ul style="list-style-type: none"> • Id • page_count • dish_count 	Menu
Trim and collapse white spaces	<ul style="list-style-type: none"> • Sponsor • Event • Event_grouped • Venu • Place • Physical_descriptions • Occasion • Notes • Call_number • Location • Status 	Menu
Convert to Title case	<ul style="list-style-type: none"> • Sponsor • Event • Event_grouped • Venu • Place • Physical_descriptions • Occasion • Notes • Call_number • Location • Status 	Menu
Convert to ISO Date format date	<ul style="list-style-type: none"> • Date 	Menu

Advanced Cleaning:

In the advance cleaning we executed the GREL command to remove the special characters in the event and occasion columns for the Menu table and repeated the

standard cleaning process again. Also, the event column underwent various facet edition operations as many record values does not imply useful meaning. Later stage we performed multiple clustering with all the permutation and combination of Method and Keying functions in text facet. We also performed the same set of operations in the occasion column as well.



We discovered that to have a useful event category we must further cluster the event records to a uniform category. By considering to keep provenance information, we created another column namely event_group in the Menu table based on multi clustered events and performed further multi clustering to create smaller categories under the event column. For example, many variance of values for cluster Banquet in Honor regrouped to 'Banquet In Honor' cell value. This enables us to have clean data sets for events and define robust use case definitions for 'popular dishes for an event'.

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 **metaphone3** 107 clusters found


21 22

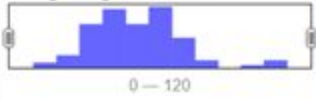
- Complimentary Supper To The Ladies Of The Hotel Men S Benefit Association (1 rows)
- Complimentary To B.W. Rowell, Illustrious Imperial Recorder Of Imperial Council Of North America (1 rows)
- Complimentary To Mr. W.H. Metson, prior To His Departure For Cape Nome, alaska (1 rows)
- Complimentary Banquet To Mr. N.C. Goodwin (1 rows)
- Complimentary Banquet To Edward M. Morgan, pres. (1 rows)


☐ **Banquet In Honor**

- Banquet In Honor Of The Third Annual Convention Of Above (2 rows)
- Banquet In Honor Of Above And Othr Members Of The Board Of Health (1 rows)
- Banquet In Honor Of Charles W. Fairbanks, Vice President Of U.S. (1 rows)
- Banquet In Honor Of Congressmen From Greater New York (1 rows)
- Banquet In Honor Of Count Ferdinand De Lessups (1 rows)
- Banquet In Honor Of Ferdinand W. Peck (1 rows)
- Banquet In Honor Of Grand Lodge Officers (1 rows)
- Banquet In Honor Of Hoo Hoo (1 rows)
- Banquet In Honor Of Judge Peter S. Grosscup & Judge Christian C. Kohlsaat (1 rows)
- Banquet In Honor Of K. Ishii (1 rows)
- Banquet In Honor Of Kaiser Wilhelm II (1 rows)

Choices in Cluster

2 — 40

Rows in Cluster

0 — 160

Average Length of Choices

0 — 120

Length Variance of Choices

0 — 58

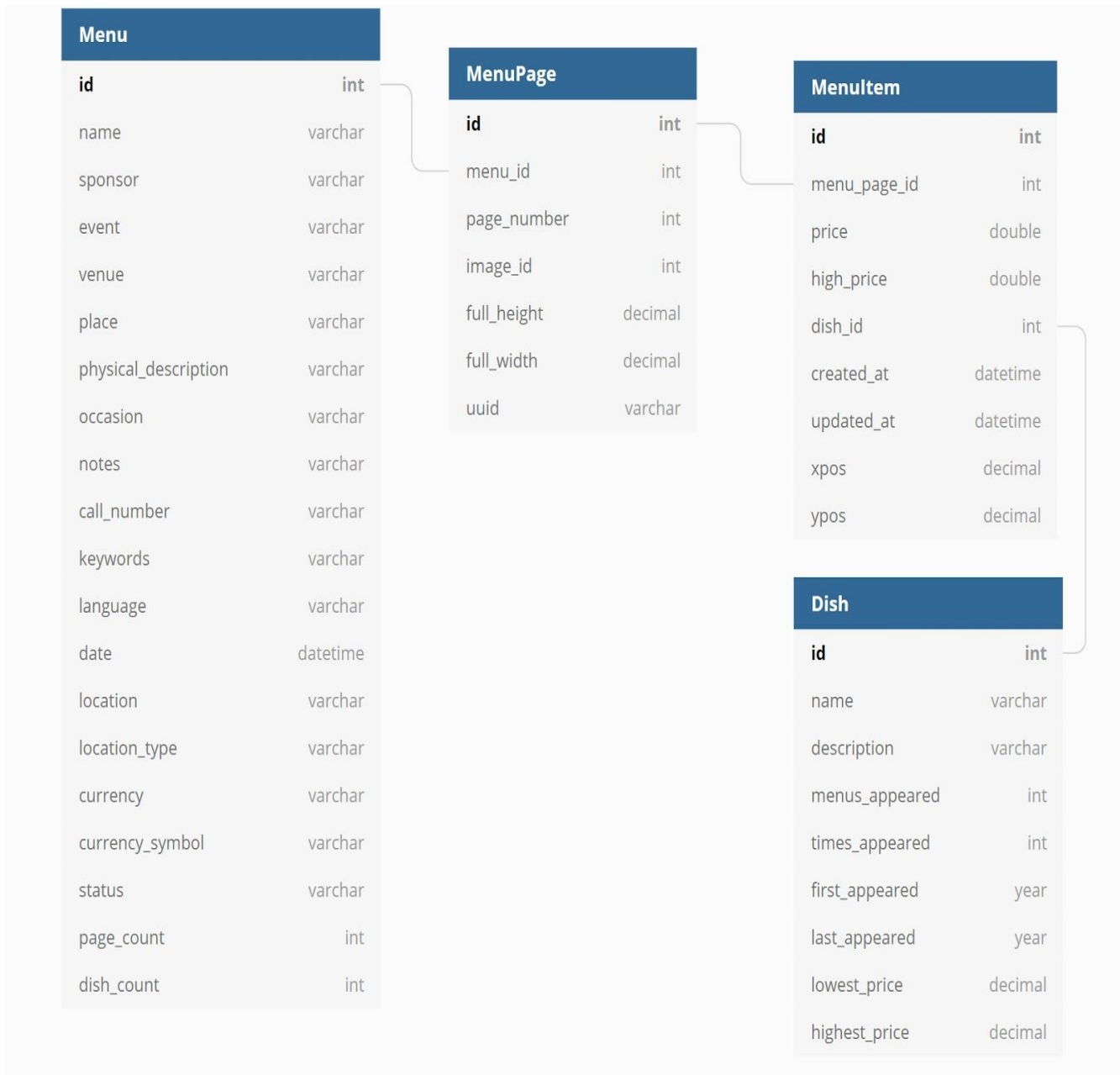
Select All Unselect All Export Clusters **Merge Selected & Re-Cluster** Merge Selected & Close Close

Additional data cleaning (Using SPARK)

One of the major issues faced during the OpenRefine data cleaning exercise was scaling. It just didn't work with larger datasets. Data had to be manually partitioned and cleaned separately per partition. It wasn't just inefficient but it prevented us from doing entire dataset wide clustering and other operations that needed the entire set together for data cleaning activities. Due to this limitation, we considered large scale distributed solutions such as Apache Spark which allows us to scale as much as we want and provide easy interfaces (python, java, SQL etc.) to interact and do operations on data. In addition to that, Spark takes care of partitioning on the back-end without the need of manual partitioning. We deployed our spark cluster on a Microsoft Azure VM in cloud and loaded the Dish.csv dataset. We used pyspark shell interface to do operations on the data (pyspark shell code is included in the zip folder submitted separately). Following cleaning operations were done using Spark.

- Convert text fields to lower case
- Remove trailing white spaces at the beginning and end
- Remove multiple consecutive white spaces and keep only one space
- Delete the unpopulated description column
- Cluster using MLlib's K-means clustering capability and combine similar names together for manual cleaning after that

Relational Schema



Schema definition for SQLite:

<pre>CREATE TABLE Menu (id INTEGER PRIMARY KEY, name TEXT, sponsor TEXT, event TEXT, venue TEXT, place TEXT, physical_description TEXT, occasion TEXT, notes TEXT, call_number TEXT, keywords TEXT, language TEXT, date datetime, location TEXT, location_type TEXT, currency TEXT, currency_symbol TEXT, status TEXT, page_count INTEGER, dish_count INTEGER);</pre>	<pre>CREATE TABLE MenuPage (id INTEGER PRIMARY KEY, menu_id INTEGER, page_number INTEGER, image_id INTEGER, full_height REAL, full_width REAL, uuid TEXT, FOREIGN KEY(menu_id) REFERENCES Menu(id));</pre>
<pre>CREATE TABLE MenuItem (id INTEGER PRIMARY KEY, menu_page_id INTEGER, price REAL, high_price REAL, dish_id INTEGER, created_at DATETIME, updated_at DATETIME, xpos REAL, ypos REAL, FOREIGN KEY(menu_page_id) REFERENCES MenuPage(id), FOREIGN KEY(dish_id) REFERENCES Dish(id));</pre>	<pre>CREATE TABLE Dish (id INTEGER PRIMARY KEY, name TEXT, description TEXT, menus_appeared INTEGER, times_appeared INTEGER, first_appeared DATETIME, last_appeared DATETIME, lowest_price REAL, highest_price REAL);</pre>

Steps to load data into SQLite (via prompt):

```
.mode csv Menu
.import Menu.csv Menu
.mode csv MenuItem
.import MenuItem.csv MenuItem
.mode csv MenuPage
.import MenuPage.csv MenuPage
.mode csv Dish
.import Menu.csv Dish
```

Integrity Constraints: *(Queries are included in a separate Queries.txt file)*

- Uniqueness constraint for primary key fields (id columns) of all tables
- Foreign Key/Inclusion constraints. Example, menu_id field values in MenuPage table should be present in id field in Menu table (same applies to manu_page_id and dish_id fields in MenuItem table)
- Value ranges constraints
 - Checks for non negative values
 - First appeared date value should be smaller than last appeared
 - Highest price value should be higher than all other price columns including price, lowest_price and other price column from other tables as well
 - Created_at date should be less than equal to the first_appeared field
- Total count of pages referencing same menu_id in the MenuPage table should be same as the value of page_count field for that menu id in the Menu table

Workflow

Menu

Since we were able to use OpenRefine as the primary cleaning tool for the Menu.csv data set, we were able to easily generate the yesWorkFlow model utilizing the or2ywtool. Here are the steps taken to generate the workflow model for Menu.csv

```
# Generate YW file
$ or2yw -i openRefine/Menu.json -o yesWorkFlow/Menu.py -t parallel
```

```
# To get workflow png
$ or2yw -i openRefine/Menu.json -o yesWorkFlow/Menu_workflow.png -ot png -t parallel
```

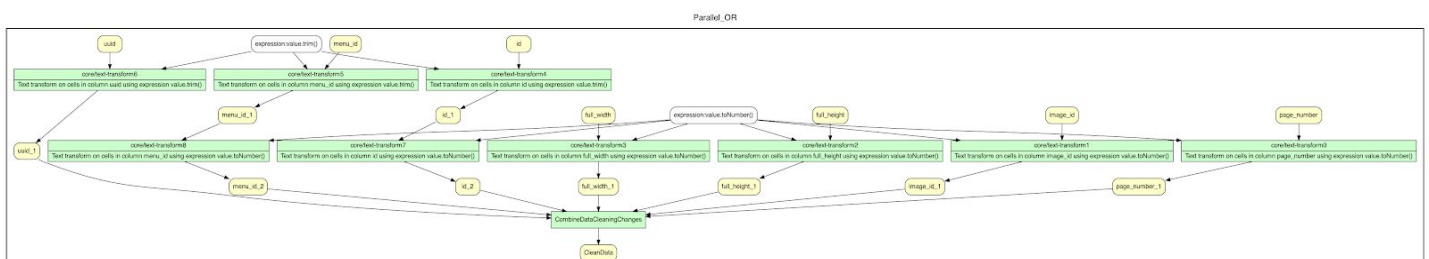
The Menu Workflow Diagram generated is too large to be placed inside the document. Instead, the Menu Workflow Diagram can be found here in our source files here: https://github.com/MaybeWorkFlow/CS513/blob/master/yesWorkFlow/Menu_workflow.png

Menu Page

The Menu Page workflow creation was almost Identical to the Menu workflow:

```
# generate YW file
$ or2yw -i openRefine/MenuPage.json -o yesWorkFlow/MenuPage.py -t parallel

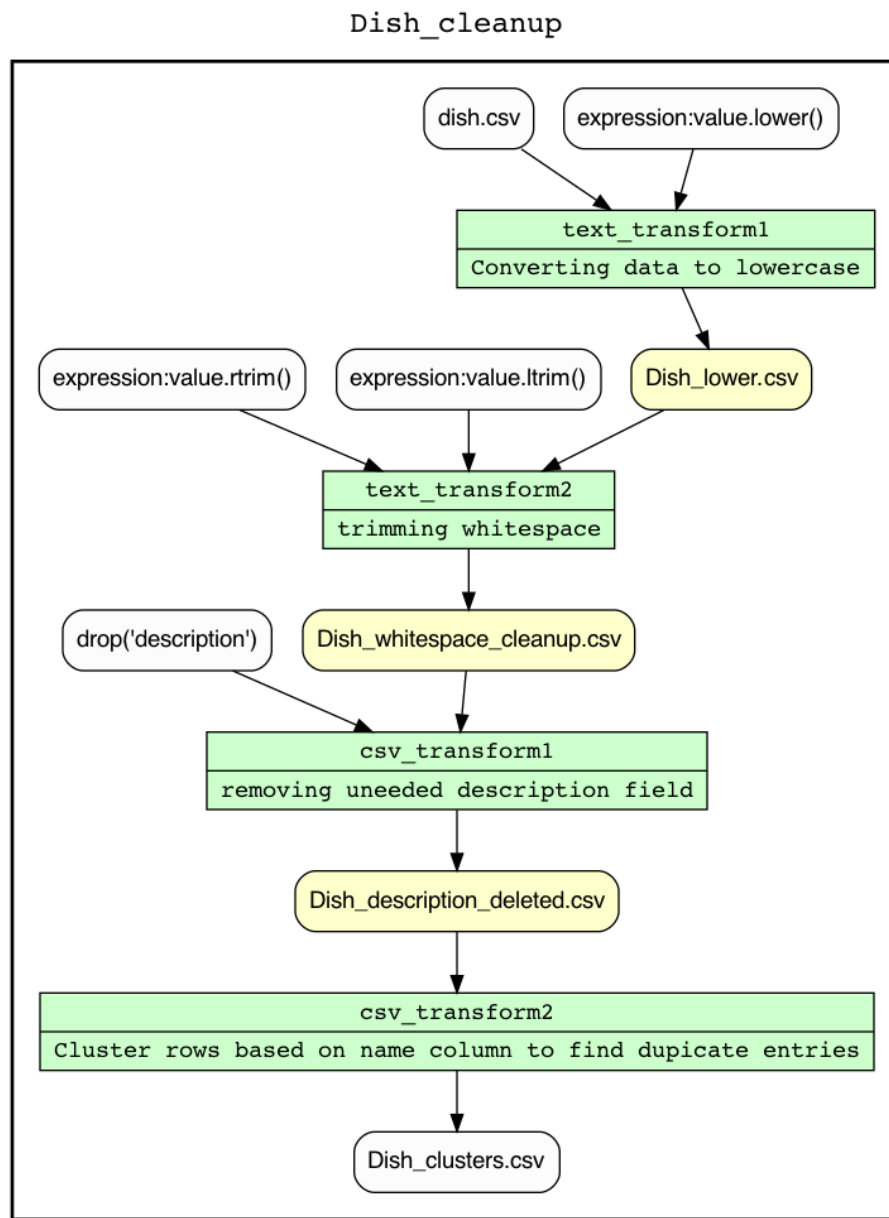
# To get workflow png
$ or2yw -i openRefine/MenuPage.json -o yesWorkFlow/MenuPage_workflow.png -ot png -t parallel
```



Dish

The Dish.csv dataset was too large to work with OpenRefine in a convenient manner, so we've opted to use a different tool to clean the data. The tool of choice to clean this data set was Spark. Since we used PySpark, it was easy for us to annotate and generate a yes workflow model on top of the source code.

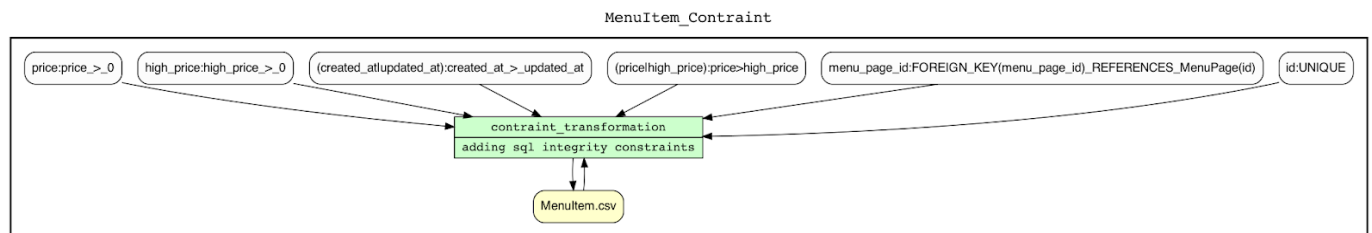
```
$ yw graph yesWorkFlow/dish.py | dot -Tpng -o yesWorkFlow/Dish_workflow.png
```



Menu Item

Like the Dish dataset, MenuItem.csv is too large to be modified with OpenRefine. Additionally, MenuItem.csv doesn't contain any text fields, so the Spark cleaning steps aren't applicable. For this data set, we've added SQL constraints, and have generated a yesWorkflow model to showcase them (even tho they aren't *directly* applied to the data).

```
# MenuItem
yw graph yesWorkflow/menuItem.py | dot -Tpng -o yesWorkflow/MenuItem_workflow.png
```



Provenance

Creating the data provenance was pretty trivial thanks to the yesWorkflow tools. To create the data provenance for each file, we extracted the graph edges from the work flow model using grep. With the edges extracted, there was just a little text replacement and the data provenance was basically arealy created.

```
$ yw graph yesWorkflow/dish.py | grep -e "->" > provenance/dish_edges.lp
```

Once the provenance model was created, it was just a matter of creating a little provenance query method, and then we could quickly look of the data lineage using clingo.

Ex:

```
% Dish Data Provenance
dish(text_transform1,"Dish_lower.csv").
dish("expression:value.lower()", text_transform1).
dish("dish.csv",text_transform1).
dish(text_transform2, "Dish_whitespace_cleanup.csv").
```

```

dish("expression:value.ltrim()", text_transform2).
dish("expression:value.rtrim()", text_transform2).
dish("Dish_lower.csv", text_transform2).
dish(csv_transform1, "Dish_description_deleted.csv").
dish("drop('description')", csv_transform1).
dish("Dish_whitespace_cleanup.csv", csv_transform1).
dish(csv_transform2, "Dish_clusters.csv").
dish("Dish_description_deleted.csv", csv_transform2).

dish_provenance(X,Y) :- dish(X,Y).
dish_provenance(X,Y) :- dish(X,Z), dish(Z,Y).

ans(X) :- dish_provenance(X, csv_transform2).

#show ans/1.

```

Conclusion and Future work

We believe that we have accomplished significant improvements in the overall data quality of New York Public Library's crowd-sourced historical menus dataset. With this, we can perform the data analytics for the use case 'popular menus in an event'.

Although our strategy was to clean the dataset based on use cases, we have applied a general foundational cleaning to each datasets for future use cases. Because of this, these data sets can be used for many additional use cases with only needing small additional cleaning efforts. We have observed that many data sets are grouped in an incorrect way across different tables. One of the future steps would be to group these together to simplify the workflow and expand to serve for more use cases. Although we have done tremendous improvement in quality of the data, we also think that there would be many areas of improvement. For example, to identify the best clustering and integrity constraints to simplify the overall workflow.

One of the biggest challenges in the project is to perform the data cleaning on large dataset. As OpenRefine is limited to small datasets, we had to find additional tools for

scaling. Clustering and defining Integrity Constraints (IC) are also equally challenged throughout the project. We got an opportunity to explore different tools to perform the cleaning other than OpenRefine, which helps to understand industry widely used tools sets for cleaning and defining workflows. After this significant effort in the project, it also helps to understand the significance of data cleaning in the data science domain.

Contributions:

- Abhilash Jacob: Data assessment, OpenRefine, Project report
- Ishan Shah - SQL Integrity/schema, additional Cleaning In Spark, Project Report
- Nathaniel Rupsis - WorkFlow models, Data provenance, Project report