

Restaurant Menu Data New York Public Library (Data Cleaning Case Study)

Alpas, Michael
Department of Computer Science,
UIUC
(309)533-8994
malpas2@illinois.edu

Singh, Mohit
Department of Computer
Science, UIUC
(309)750-7110
Mohits3@illinois.edu

Yadav, Upendra Singh
Department of Computer Science,
UIUC
(215)439-4579
Usyadav2@illinois.edu

This data contains below 4 files which are related to each other.
High level description of these files is given below –

1. INTRODUCTION

Our team once again embarked to a new journey as part for CS513 Theory and Practice of Data Cleaning class. This is our third project as a team and always excited for the learning and camaraderie.

We understand that data cleaning is a very important process in providing Data Science or Machine Learning solutions. Most of the time, engineers need to spend a vast majority of their time in cleaning the data to be able to provide even the simplest data visualization to business partners.

For this project, we selected the historical restaurant menu offered by NY Public Library, we think the dataset is challenging for us to apply the data cleaning techniques we learned from the class. The focus of this project is to identify data quality issues from the data set and provide data cleaning solutions to address the problem.

In this project, we have used tools and techniques learnt in CS513: Theory and Practice of Data Cleaning subject to clean historical NYPL restaurant menu data. We have also documented our whole data cleaning process workflow and steps by using YesWorkflow.

2. DATASET

For this project we have used open source publicly available restaurant Menu dataset from New York Public Library. This dataset is one of the largest restaurant menu collections in the world which is used by historians, nutritional scientists, chefs, novelists and food enthusiasts. The main credit of this dataset goes to Miss Frank E. Buttolph, who collected more than 25,000 menus between 1900 to 1924 on behalf of NYPL library. This collection of data contains approximately 45,000 menus dating from 1840s to present in which about quarter of menus are digitized and made available in NYPL digital library. [\[1\]](#)

This data can be downloaded from below location –

<http://menus.nypl.org/data>

| File Name | Description | No. of Fields |
|--------------|--|---------------|
| Menu.csv | <ul style="list-style-type: none">Contains Information about Menus of different restaurants. | 20 |
| MenuPage.csv | <ul style="list-style-type: none">Information of Menu Pages referenced to Menu.Each Menu can have multiple Menu Pages. | 7 |
| MenuItem.csv | <ul style="list-style-type: none">Information about Menu Items referenced to each Menu Page.Each Menu Page can contain multiple Menu Items. | 9 |
| Dish.csv | <ul style="list-style-type: none">Contains information about dishes mapped to each Menu Item.Referenced to MenuItem.csv | 9 |

2.1 Menu.csv

This is one of the primary files among all the 4 files, which has menu information associated with different restaurants. Each menu is uniquely identified by Menu Id. Menu.csv file has relationship with MenuPage.csv file using Menu Page Id. One menu item can have multiple Menu Pages.

Following fields are present in the Menu.csv.

- Id - unique id for each Menu.
- Name – name of Menu
- Sponsor – name of the restaurant
- Event – name of the event for which menu was created.
- Venue – type of place where food was served from Menu, Ex – Educational, Private etc.
- Place – address where the menu was used. (Includes city and state name)
- Physical_Description – size of menu card.

- Occasion – type of event such as anniversary, birthday etc.
- Notes – any comments about Menu.
- Call_Number – menu’s call number.
- Keywords – keywords for Menu.
- Language – language in which Menu was printed.
- Date – date in which menu was collected.
- Location – place where Menu was used.
- Location_Type – type of location.
- Currency – currency used in Menu.
- Currency_Symbol – symbol of the currency used in Menu.
- Status – transcription status of the Menu such as complete or under review.
- Page_Count – total number of pages in the Menu.
- Dish_Count – total Number of dishes in the Menu.

2.2 MenuPage.csv

Following fields are present in MenuPage.csv.

- Id - unique Identifier of MenuPage.csv
- Image_Id – id of the image of Menu page.
- Menu_Id – foreign key from Menu.csv
- Page_Number – page number of Menu.
- Full_Height – height of menu page
- Full_Width – width of menu page
- UUID – unique Identifier

2.3 MenuItem.csv

Following fields are present in MenuItem.csv.

- Id – unique identifier for each row in Menu Item.
- Menu Page Id – menu item referenced to Menu Page.
- Price – price of the menu item.
- High_Price – price of the costliest portion of the item,
- Dish_Id – relationship between menu item and dish.
- Created_At – record creation date.
- XPos – X axis coordinate of the menu item in the scanned menu page.
- Ypos – Y axis coordinate of the menu item in the scanned menu page.
- Updated At – record last updated date.

2.4 Dish.csv

Following fields are present in Dish.csv.

- Id – Unique Identifier
- Name – dish name
- Description – description of the dish

- Menus_Appeared – number of menus where dish appeared.
- Times_Appeared – how many times dish appeared in a menu.
- First_Appeared – when dish appeared at first time.
- Last_Appeared – Year when dish appeared last time in menu.
- Lowest_Price – lowest price of the dish in menu.
- Highest_Price – highest price of the dish in menu

3. TARGET USE CASE

Our team explored the dataset and below are the possible use cases –

3.1 Main Use case (U1)- Data Cleaning is necessary

- Most popular dishes as per different location and most popular dishes as per different occasions.

After cleaning this dataset, we can easily find out which dish was served most in which occasion? Such as which dish was served most in Easter, thanksgiving or wedding anniversaries? We can also find out which dish was more popular in which location or place?

To work on above use case, we will have to Join Menu, Menu Page, Menu Item and Dish tables and we will have to clean Dish name, menu appeared fields from Dish table, Location, Place and occasion fields from Menu table.

3.2 Corner Use case (U0) – Zero Data Cleaning

- What was the costliest (highest price) dish sold in last 100 years?

There are some use cases where we can easily do some analysis in dataset without doing any data cleaning. One of the examples of such use case should be to find out the dish name which was sold in highest price in last 100 years of data.

To obtain this information, we just need two columns of Dish table (Dish Name and Highest Price), we can sort dishes based on prices and can identify dish name which was sold in highest price.

- When (In which particular year) a new dish was introduced and in which year it was removed from menu.

Getting above information will help to many food enthusiasts to understand when a particular dish was introduced in restaurants or when it was removed from their menu completely? This data we can easily get by analyzing Dish table and using two columns such as – Dish name, first appeared and last appeared.

3.3 Corner Use case (U2) – No amount of Data Cleaning

- Price trend (highest and lowest price) of the dishes over the years (timeline) in different events, occasions and locations.

Usually most of the relevant use cases we can get from this dataset by vigorously cleaning the dataset using the data cleaning technique taught in CS513 class.

However, in some use cases such as finding out highest and lowest price of dishes over the years in different locations, events and occasions. In such use cases where multiple columns are involved which needs cleaning of multiple

columns, it becomes quite challenging to give detail of every costliest and cheapest dish in every year as per different location and event.

One of the major issue, we will get here that while cleaning multiple columns (such as dish name, occasions, event, location, sponsor names), we will encounter multiple blank data and also bad data across multiple columns, due to that it would be very difficult to provide correct metric for this analytic query.

Along with above mentioned use cases, we can also find out multiple supplement use cases as mentioned below -

- Cheapest and costliest restaurants according to location over the timeline
By this dataset, we can identify which restaurants served costliest and which served cheapest dishes. We can also categorize costliest and cheapest restaurants according to location. This use case will help us to understand which locations had costliest restaurants in last 100 years.
- Trend analysis of a dish price fluctuation over the years
By analyzing this dataset, we can try to analyze how price of a dish changed over years in last century. Are there any dishes which became costlier over the time?
- Restaurants serving unique dishes.
We can also find out, are there any unique dishes which were served by some unique restaurants. Such as we can find out restaurants which are serving Japanese, Chinese or Filipino dishes.
- Trend of dishes over the timeline
We can also find out an interesting trend about which dishes were more popular in which decade.

4. DATA QUALITY ISSUES

During our dataset exploration, it seems the NY Public Library dataset will be challenging for the team – it will build our confidence and technical acumen by applying our knowledge learned from CS 513 Theory and Practice of Data Cleaning. For implementing the use case U1 the common data quality issues we observed are trailing white spaces and special characters. We also discussed clustering similar words to fix possible duplicate data. There are also several data columns having blank values that we might disregard during the process.

Another consideration we discussed was adding another dimensionality in some of the dataset. Basically, extracting a new column depending on the final use case of the data. In relation to this, removing some of the columns that might not be needed for the use case is another data cleaning opportunity.

Since for implementation of U1 use case, we will have to join all 4 tables (Menu, MenuPage, MenuItem, Dish), so we analyzed all the columns given in all 4 tables.

Below is the snippet of data columns describing quality issue:

4.1.1 Menu.csv

| Column | Description | Sample Value | Quality Issue |
|---------|---------------------------------------|------------------|---|
| name | Menu name | "Victoria Luise" | - Contains special character. - Some of the rows has blank value. |
| sponsor | Menu sponsor | ? HOTEL | - Contains special character. - Some of the rows has blank value. |
| event | Time of menu (i.e., breakfast, lunch) | ? | - Some rows have question mark as a value. - Some of the rows has blank value. |

4.1.2 Dish.csv

| Column | Description | Sample Value | Quality Issue |
|----------------|--------------------------|--------------|--|
| name | Dish name | " sautees | - Contains special character. - Some has numeric value. |
| description | Dish description | | - Blank description |
| first appeared | Year dish was introduced | 1 | - Invalid year format |

4.1.3 MenuItem.csv

| Column | Description | Sample Value | Quality Issue |
|--------|-------------|--------------|-------------------------------------|
| price | Item price | | - Some of the rows has blank value. |

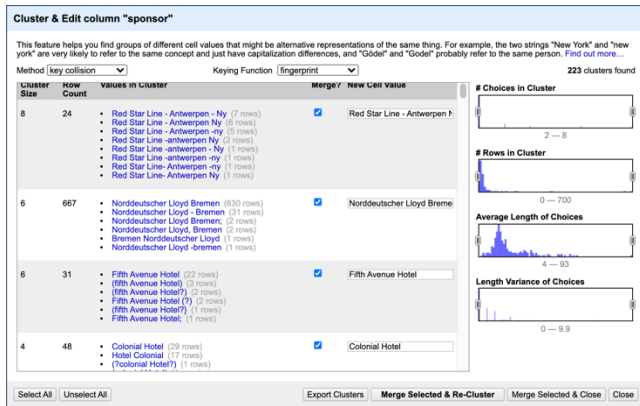


Figure 3: Clustering “Sponsor”

We also clustered the “Sponsor” values as you can see from the above figure which resulted to 4367 cells in column was updated.

Following steps were performed while cleaning Sponsor column –

- Common Steps such as trimming leading and trailing whitespaces, collapse conjugative white spaces, removing special character using GREL function.
- Created a facet and performed the cluster operation using the key-collision method and fingerprint function and after that merged the relevant clusters.
- Created a facet and performed the cluster operation using n-gram fingerprint, meta-phone3, cologne-phonetic methods and after that merged the relevant clusters.
- Created a facet and performed the cluster operation using the nearest neighbor method and levstein distance function and after that merged the relevant clusters.
- Created a facet and performed the cluster operation using the nearest neighbor method and PPM distance function and after that merged the relevant clusters.

5.1.1.2 Cleaning Event Column

Same here, we applied our cleanup workflow from the “Sponsor” column -- trimming leading and trailing whitespaces, followed by collapsing consecutive whitespace. Below was the regex pattern we used all over our process to remove special characters and 1128 cells in column was changed applying it.

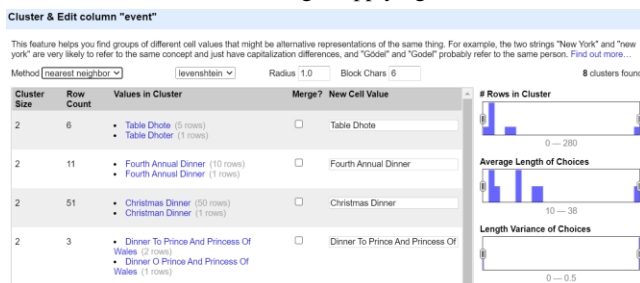


Figure 4: Clustering “Event” using nearest-neighbor.

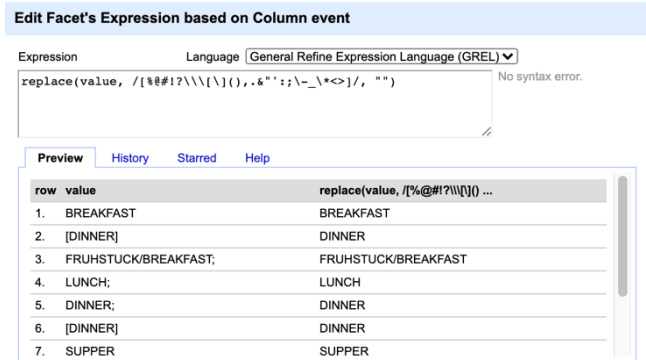


Figure 5: RegEx expression to remove special characters

5.1.1.3 Cleaning Physical Description Column

We performed additional steps in this column compared to the above cleanup workflow. We observed that the value for this column contains the “type” of the physical menu (i.e., card, folder, etc.) and the “size” (i.e., 4x5). We decided to create separate columns by splitting those values, leveraging our knowledge with Python programming language. Below are the images and Python scripts:

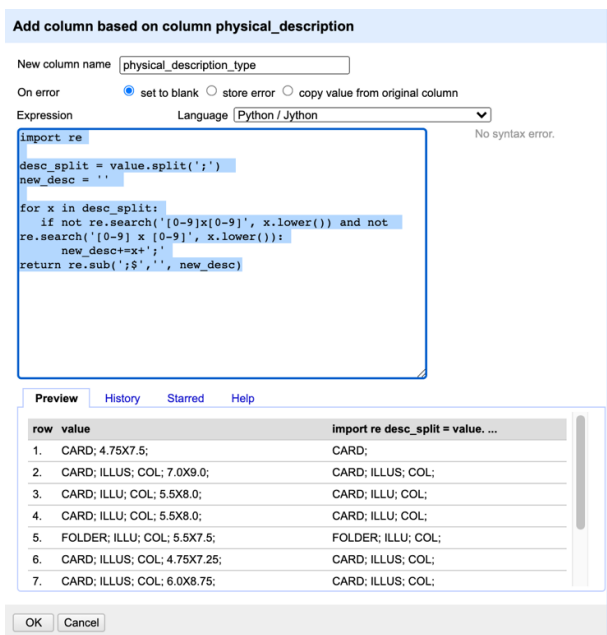


Figure 6: Creating new column *physical_description_type*

Add column based on column physical_description

New column name:

On error: ☒ set to blank ☐ store error ☐ copy value from original column

Expression:

Language:

No syntax error.

Preview History Starred Help

| row | value | import re menu_size = value.s ... |
|-----|------------------------------|-----------------------------------|
| 1. | CARD; 4.75X7.5; | 4.75X7.5 |
| 2. | CARD; ILLUS; COL; 7.0X9.0; | 7.0X9.0 |
| 3. | CARD; ILLU; COL; 5.5X8.0; | 5.5X8.0 |
| 4. | CARD; ILLU; COL; 5.5X8.0; | 5.5X8.0 |
| 5. | FOLDER; ILLU; COL; 5.5X7.5; | 5.5X7.5 |
| 6. | CARD; ILLUS; COL; 4.75X7.25; | 4.75X7.25 |
| 7. | CARD; ILLUS; COL; 6.0X8.75; | 6.0X8.75 |

OK Cancel

Figure 7: Creating new column *physical_menu_size*

And for our final step for this column, we clustered the values –

Cluster & Edit column "physical_description_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 "Göde" and "Gode" probably refer to the same person. [Find out more...](#)

Method: Keying Function: 73 clusters found

| Cluster Size | Row Count | Values in Cluster | Merge? | New Cell Value |
|--------------|-----------|---|-------------------------------------|----------------------|
| 8 | 921 | <ul style="list-style-type: none"> Card; Illus; Col; (773 rows) Card; Col; Illus; (15 rows) Card; Illus; Col; (35 rows) Card; Illus; Col; (7 rows) Card; Col; Illus; (2 rows) Card; Illus; Col; (1 rows) | <input checked="" type="checkbox"/> | Card; Illus; Col; |
| 7 | 354 | <ul style="list-style-type: none"> Folder; Illus; (229 rows) Folder; Illus; (15 rows) Illus; Folder; (5 rows) Folder; Illus; (3 rows) Folder; Illus; (1 rows) Folder; Illus; (1 rows) | <input checked="" type="checkbox"/> | Folder; Illus; |
| 6 | 301 | <ul style="list-style-type: none"> Booklet; Illus; Col; (250 rows) Booklet; Col; Illus; (39 rows) Booklet; Illus; Col; (5 rows) Booklet; Illus; Col; (2 rows) Booklet; Illus; Col; (1 rows) Booklet; Col; Illus; (1 rows) | <input checked="" type="checkbox"/> | Booklet; Illus; Col; |
| 6 | 615 | <ul style="list-style-type: none"> Folder; Illus; Col; (540 rows) Folder; Col; Illus; (138 rows) | <input checked="" type="checkbox"/> | Folder; Illus; Col; |

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

Figure 8: Clustering for *physical_description_type*

Cluster & Edit column "physical_menu_size"

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 "Göde" and "Gode" probably refer to the same person. [Find out more...](#)

Method: Keying Function: 269 clusters found

| Cluster Size | Row Count | Values in Cluster | Merge? | New Cell Value |
|--------------|-----------|---|-------------------------------------|----------------|
| 5 | 59 | <ul style="list-style-type: none"> 5x7.25 (30 rows) 5x7.25 (1 rows) 5x7.25 (1 rows) 5x7.25 (1 rows) | <input checked="" type="checkbox"/> | 5x7.25 |
| 4 | 295 | <ul style="list-style-type: none"> 4.5x7 (151 rows) 4.5x7 (62 rows) 4.5x7 (1 rows) 4.5x7 (1 rows) | <input checked="" type="checkbox"/> | 4.5x7 |
| 3 | 17 | <ul style="list-style-type: none"> 5x8.25 (11 rows) 5x8.25 (5 rows) 5x8.25 (1 rows) | <input checked="" type="checkbox"/> | 5x8.25 |
| 3 | 110 | <ul style="list-style-type: none"> 4.25x8.5 (72 rows) 4.25x8.5 (37 rows) 4.25x8.5 (1 rows) | <input checked="" type="checkbox"/> | 4.25x8.5 |
| 3 | 3 | <ul style="list-style-type: none"> 7x11.5 (1 rows) 7x11.5 (1 rows) 7x11.5 (1 rows) | <input checked="" type="checkbox"/> | 7x11.5 |
| 3 | 49 | <ul style="list-style-type: none"> 4.25x7 (26 rows) 4.25x7 (23 rows) | <input checked="" type="checkbox"/> | 4.25x7 |

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

Figure 9: Clustering for *physical_menu_size*

5.1.1.4 Cleaning Date Column

This column was interesting, though it is a date value, there seems to be an outlier – some has 2928 and 0001. We applied the below Python script to remove the outliers.

Custom text transform on column date

Expression:

Language:

Preview History Starred Help

| row | value | import re year = int(re.sub(' ... |
|-----|------------|-----------------------------------|
| 1. | 1900-04-15 | 1900-04-15 |
| 2. | 1900-04-15 | 1900-04-15 |
| 3. | 1900-04-16 | 1900-04-16 |
| 4. | 1900-04-16 | 1900-04-16 |
| 5. | 1900-04-16 | 1900-04-16 |
| 6. | 1900-04-16 | 1900-04-16 |
| 7. | 1900-04-16 | 1900-04-16 |

On error: ☒ keep original ☐ set to blank ☐ store error ☐ Re-transform up to 10 times until no change

Figure 10: Removing outliers

5.1.1.5 Cleaning Location Column

At first, we did cluster and we found 216 clusters, then we reviewed every clusters and after reviewing the name of each cluster did Merge Selected & Re-Cluster”.

Below is our clustering snapshot from Location column.

Cluster & Edit column "location"

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 "Göde" and "Gode" probably refer to the same person. [Find out more...](#)

Method: Keying Function: 192 clusters found

| Cluster Size | Row Count | Values in Cluster | Merge? | New Cell Value |
|--------------|-----------|--|-------------------------------------|---------------------------|
| 2 | 25 | <ul style="list-style-type: none"> Hotel Westminster (14 rows) Westminster Hotel (11 rows) | <input checked="" type="checkbox"/> | Hotel Westminster |
| 2 | 13 | <ul style="list-style-type: none"> Hotel Statler (10 rows) Statler Hotel (3 rows) | <input checked="" type="checkbox"/> | Hotel Statler |
| 2 | 2 | <ul style="list-style-type: none"> Hotel Vendome And Profile House (1 rows) Profile House And Hotel Vendome (1 rows) | <input checked="" type="checkbox"/> | Hotel Vendome And Profile |
| 2 | 2 | <ul style="list-style-type: none"> U.S.S. New York (1 rows) U.S.S. New York (?) (1 rows) | <input checked="" type="checkbox"/> | U.S.S. New York |
| 2 | 2 | <ul style="list-style-type: none"> Gallatin Hotel (1 rows) Hotel Gallatin (1 rows) | <input checked="" type="checkbox"/> | Gallatin Hotel |
| 2 | 5 | <ul style="list-style-type: none"> Hofbrau Haus (3 rows) Hofbrau Haus (2 rows) | <input checked="" type="checkbox"/> | Hofbrau Haus |
| 2 | 3 | <ul style="list-style-type: none"> New England Shorthand Reporters' Association (2 rows) New England Shorthand Reporter's Association (1 rows) | <input checked="" type="checkbox"/> | New England Shorthand Re |

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

Figure 11: Clustering “Location”

5.1.1.6 Cleaning call_number Column

Below steps were performed to clean call_number column –

- Trimmed leading and Trim trailing white spaces
- Collapse consecutive white spaces.

5.1.1.7 Cleaning notes Column

Below steps were performed to clean notes column –

- Trimmed leading and Trim trailing white spaces
- Collapse consecutive white spaces.

For other remaining columns we just trimmed leading and trailing white spaces and collapsed consecutive white spaces, apart from that did not perform any further cleaning we will not use them in our targeted usecase.

5.1.2 Dish.csv (size: 26.8 MB, rows: 428086)

Dish.CSV file was quite huge, and it was very challenging for us to clean this using key-collision method of OpenRefine. We changed memory heap size to 16 GB, after that we could cleaned this file.


Similarly Menu.csv while cleaning Dish.csv file our cleanup workflow started with trimming leading and trailing whitespaces, followed by collapsing consecutive whitespace.

Below common steps are performed in every column which we cleaned –

- Trim leading and trailing white spaces.
- Collapse consecutive white spaces.
- Remove the special characters using GREL functionality if existing in that column.
- Make a facet and perform clustering operation using key collisions and select and merge similar cluster (This method was very expensive to use in this file.)

5.1.2.1 Cleaning Name Column

Removing leading and trailing white spaces.


OpenRefine
Dish.csv
[Permalink](#)

Facet / Filter
Undo / Redo 1 / 2

0. Create project

1. Text transform on 9290 cells in column name: value.trim()

2. Text transform on 6582 cells in column name: value.replace(/s+/, "")

428132 rows

Show as: rows records
Show: 5 10 25 50 rows

| ▼ All | ▼ Id | ▼ name |
|-------|--------|----------------------------------|
| ☆ ↗ | 1. 1 | Consomme printaniere royal |
| ☆ ↗ | 2. 2 | Chicken gumbo |
| ☆ ↗ | 3. 3 | Tomato aux croutons |
| ☆ ↗ | 4. 4 | Onion au gratin |
| ☆ ↗ | 5. 5 | St. Emilion |
| ☆ ↗ | 6. 7 | Radishes |
| ☆ ↗ | 7. 8 | Chicken soup with rice |
| ☆ ↗ | 8. 9 | Clam broth (cup) |
| ☆ ↗ | 9. 10 | Cream of new asparagus, croutons |
| ☆ ↗ | 10. 11 | Clear green turtle |

Figure 12: Result in applying trim and removing special characters.

We removed special characters by using GREL functionality.

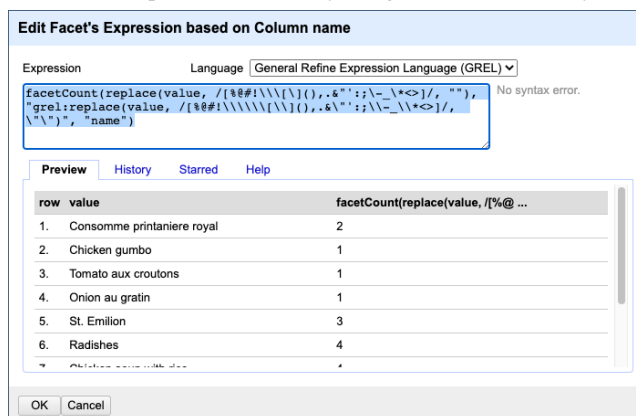


Figure 13: GREL functionality with the count

Clustering Operation using key collisions –

As stated above, we had memory issues while performing clustering key-collision operation in Dish.csv file.

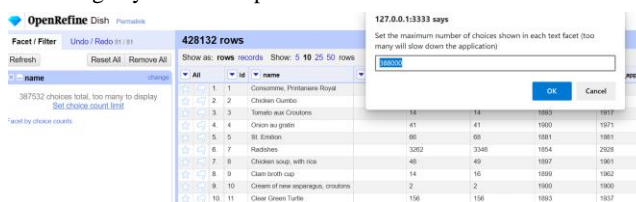


Figure 14: Issue in cleaning name column in Dish.csv due to memory size

Apart from name column, all other columns in Dish.csv file are numeric. They look pretty clean so we just trimmed leading and trailing white spaces and collapsed consecutive whitespaces of remaining columns.

5.1.3 MenuItem.csv (size: 118.6 MB, rows: 1048575)

No immediate concern on the data set when we initially loaded it through OpenRefine. All the columns are numeric, and they are almost cleaned, however we just trimmed leading and trailing white spaces and collapsed consecutive whitespaces.

5.1.4 MenuPage.csv (size: 4.7 MB, rows: 66937)

No immediate concern on the data set when we initially loaded it through OpenRefine. All the columns except dish column are numeric and they are almost cleaned, however we just trimmed leading and trailing white spaces and collapsed consecutive whitespaces.

5.1.5 Configuring Open Refine

Dish dataset is very large so by default OpenRefine takes very long time to clean data and get hanged. To overcome with this problem, we increased heap allocation (REFINE_MEMORY) to 16 GB which helped us to clean Dish dataset. These settings can be changed in OpenRefine/Contents/Info.plist file in Mac or refine.ini in windows OS.

Along with this, we have also updated REFINE MAX FORM CONTENT SIZE to 999999999.

```
# Memory and max form size allocations
#REFINE_MAX_FORM_CONTENT_SIZE=1048576
REFINE_MEMORY=1400M

# Set initial java heap space (default: 256M) for better performance with large datasets
REFINE_MIN_MEMORY=1400M
```

Figure 15: Increasing OpenRefine Heap size

6. DEVELOPING RELATIONAL DATABASE SCHEMA:

We used SQLite database (DB browser for SQLite and SQLite Studio) to create relational database and schema.

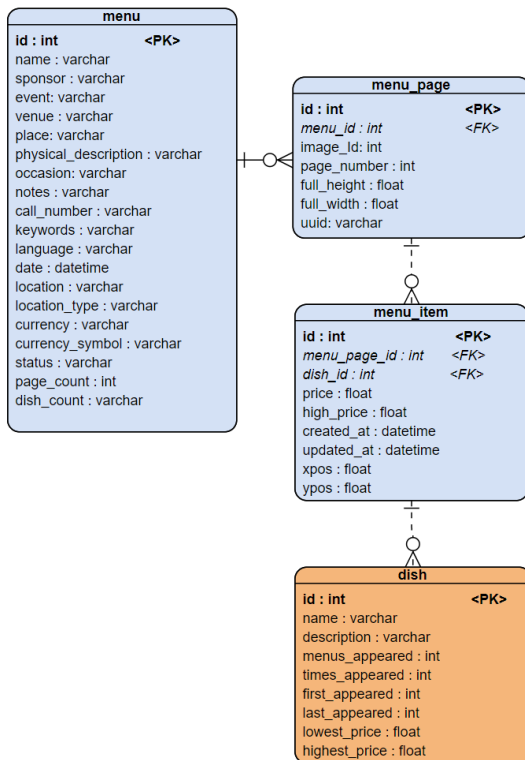
All relational DB schema files can be obtained from this [location](#).

At first, we created an empty database **nypl-db.db** using DB browser for SQLite tool and then loaded each cleaned Menu.csv, Dish.csv, Menupage.csv, MenuItem.csv file.

After importing all the CSV files, we checked integrity constraints (Primary Key and referential integrity constraints) in all the four tables.

6.1 Entity Relationship Diagram

In this below ER diagram, we have explained above dataset by creating a relation database and also by creating entity relationship diagram by depicting different tables as entities.[\[7\]](#)



NYPL dataset ER Diagram

Figure 16: ER diagram of NYPL dataset

6.2 Checking Integrity Constraints

6.2.1 Checking Integrity Constraints in Menu

1. **id** column in Menu table will act as primary key and it has to be unique, below is the query and screenshot from DB browser from SQLite.

Query –

```
select id, count(id) as count
from menu group by id having count(id) > 1;
```

```
2 select id, count(id) as count
3 from menu
4 group by id
5 having count(id) > 1;
```

```
Execution finished without errors.
Result: query executed successfully. Took 10ms
```

Figure 17: Checking Primary key is Unique

2. Since it is a primary key, so this column should also be checked for null values. (Query and screenshot from SQLite is below)

Query –

```
select id from menu where id is NULL;
```

```
7 select id
8 from menu
9 where id is NULL
10
```

```
Execution finished without errors.
Result: 0 rows returned in 22ms
At line 7:
select id
from menu
where id is NULL.
```

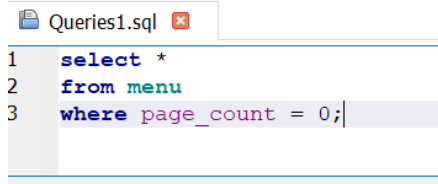
Figure 18: Checking Primary key constraint does not have null values

Result -These both above queries returned as empty, hence confirmed that id of Menu table is unique and does not contain any empty or null value.

3. **Page_count** column was checked for empty and null values, every menu must have some pages. We need to delete, if there are any records which have page_count as 0. (Query and screenshot from SQLite is below)

Query –

*select * from menu where page_count = 0;*



```
1 select *
2 from menu
3 where page_count = 0;
```

```
Execution finished without error
Result: 0 rows returned in 27ms
At line 1:
select *
from menu
where page_count = 0;
```

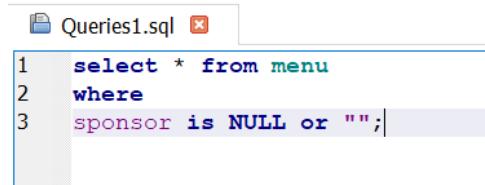
Figure 19: page_count does not have 0 value

Result -page_count is returned as empty, so no record violated this constraint.

4. **Sponsor** can't have empty or NULL values, because menu must belong to some restaurant.

Query –

*select * from menu where sponsor is NULL or "";*



```
1 select * from menu
2 where
3 sponsor is NULL or "";
```

```
Execution finished without errors
Result: 0 rows returned in 31ms
At line 1:
select * from menu
where
sponsor is NULL or "";
```

Figure 20: sponsor column does not have empty value

Result -sponsor column did not return any row, so no record violated this constraint.

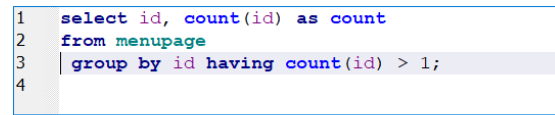
6.2.2 Checking Integrity Constraints in MenuPage

We checked below Integrity Constraints for MenuPage table

1. **id column** in MenuPage table will act as primary key and it has to be unique, below is the query and screenshot from DB browser from SQLite.

Query –

*select id, count(id) as count
from menupage group by id having count(id) > 1;*



```
1 select id, count(id) as count
2 from menupage
3 group by id having count(id) > 1;
4
```

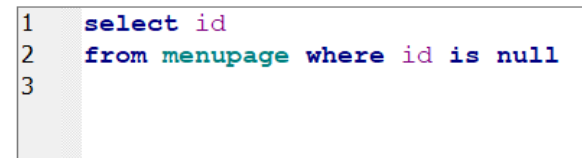
```
Result: 0 rows returned in 178ms
At line 1:
select id, count(id) as count
from menupage group by id having count(id) > 1;
```

Figure 21: Checking Primary key constraint is Unique

2. Since it is a primary key, so this id column should also be checked for null values. (Query and screenshot from SQLite is below)

Query –

select id from menupage where id is null



```
1 select id
2 from menupage where id is null
3
```

```
Result: 0 rows returned in 49ms
At line 1:
select id
from menupage
where id is null
```

Figure 22: Checking Primary key constraint does not have null values

Result - These both above queries returned as empty, hence confirmed that id of MenuPage table is unique and does not contain any empty or null value.

2. **Page_number** column was checked for empty and null values, every menupage must have some pages. We need to delete, if there are any records which have page_number as 0. (Query and screenshot from SQLite are below)

Query –

*select * from menupage where page_number = 0;*

```
1 select * from menupage
2 where page_number = 0;
3
```

Result: 0 rows returned in 69ms
At line 1:
select * from menupage
where page_number = 0;

Figure 23: page_number does not have 0 value

Result -page_number column did not return any row, so no record violated this constraint.

3. **Inner Join with Menu Table** – we checked page_number column was checked if its values are less than or equal to the total page count of the menu. This check was done by INNER JOIN between Menu and MenuPage tables.

Query –

*select * from menu m, menupage p where m.id = p.menu_id and p.page_number > m.page_count;*

```
1 select *
2 from menu m, menupage p
3 where m.id = p.menu_id and
4 p.page_number > m.page_count;
5
```

| | id | sponsor | event |
|---|-------|------------------------------|----------------------------------|
| 1 | 21467 | Mr Samuel D Coykendall | Dinner To The Holland Society... |
| 2 | 21725 | Rivers And Harbors Committee | Reception |
| 3 | 24296 | Ichthyophagous Club | 11th Annual Dinner |
| 4 | 33935 | The Grunewald | NULL |
| 5 | 34021 | Waldorf Astoria | NULL |
| 6 | 34967 | Hotel Astor | NULL |

Execution finished without errors.
Result: 6 rows returned in 302ms
At line 1:
select *
from menu m, menupage p
where m.id = p.menu_id and
p.page_number > m.page_count;

Figure 24: Inner Join with Menu Table

Result -This figure shows that there are 6 rows which are violating this constraint, it means in Menupage table, records are more than corresponding menu table. However, we are not deleting these rows as it may result in some important data loss.

6.2.3 Checking Integrity Constraints in MenuItem

We checked below Integrity Constraints for MenuItem table

1. **id column** in MenuItem table will act as primary key and it has to be unique, below is the query and screenshot from DB browser from SQLite.

Query –

*select id, count(id) as count
from menuItem group by id having count(id) > 1;*

```
1 select id, count(id) as count
2 from menuItem group by id having count(id) > 1;
3
4
```

Result: 0 rows returned in 3870ms
At line 1:
select id, count(id) as count
from menuItem group by id having count(id) > 1;

Figure 25: Checking Primary key constraint is Unique

2. Since it is a primary key, so this id column should also be checked for null values. (Query and screenshot from SQLite is below)

Query –

select id from menuItem where id is null

```
1 select id from menuItem where id is null
2
3
```

Execution finished without errors.
Result: 0 rows returned in 913ms
At line 1:
select id from menuItem where id is null

Figure 26: Checking Primary key constraint does not have null values

Result - These both above queries returned as empty, hence confirmed that id of MenuItem table is unique and does not contain any empty or null value.

2. **dish_id** column was checked for empty and null values, every menuitem must have some dishes. We need to delete, if there are any records which have dish_id as 0.

We also need to check, whether for each dish_id, corresponding dish is present in the dish table.

(Query and screenshot from SQLite is below)

Query –

*select * from menuitem where dish_id = 0;*

*select * from menuitem where dish_id not in (select id from dish) ;*

```
1 select * from menuitem where dish_id = 0;
2
3
```

Result: 0 rows returned in 960ms
At line 1:
select * from menuitem where dish_id = 0;

```
1 select * from menuitem where dish_id not in (select id from dish) ;
2
3
```

| | id | menu_page_id | price | high_price | dish_id | created_at | updated_at | xpos | ypos |
|---|---------|--------------|-------|------------|---------|-------------------------|-------------------------|----------|----------|
| 1 | 619133 | 51020 | NULL | NULL | 220797 | 2011-10-30 14:27:33 UTC | 2011-10-30 14:27:33 UTC | 0.605714 | 0.215599 |
| 2 | 837354 | 60235 | 0.2 | NULL | 329183 | 2012-03-08 23:28:32 UTC | 2012-03-08 23:28:32 UTC | 0.664286 | 0.173588 |
| 3 | 1047160 | 69117 | 0.45 | NULL | 395403 | 2012-08-14 09:52:01 UTC | 2012-08-14 09:52:01 UTC | 0.377333 | 0.469635 |

Result: 0 rows returned in 960ms
At line 1:
select * from menuitem where dish_id = 0;

Figure 27: dish_id does not have 0 value

Result – first query did not return any rows hence so no record violated for dish_id, however second query returned 3 rows, which we deleted as spurious records. After deleting, above constraint was satisfied.

6.2.4 Checking Integrity Constraints in Dish

We checked below Integrity Constraints for MenuItem table

1. **id column** in Dish table will act as primary key and it has to be unique, below is the query and screenshot from DB browser from SQLite.

Query –

select id, count(id) as count

from dish group by id having count(id) > 1;

```
1 select id, count(id) as count
2 from dish group by id having count(id) > 1;
3
4
5
```

Result: 0 rows returned in 1205ms
At line 1:
select id, count(id) as count
from dish group by id having count(id) > 1;

Figure 28: Checking Primary key constraint is Unique

2. Since it is a primary key, so this id column should also be checked for null values. (Query and screenshot from SQLite is below)

Query –

select id from dish where id is null

```
1 select id from dish where id is null
2
3
4
```

Result: 0 rows returned in 232ms
At line 1:
select id from dish where id is null

Figure 29: Checking Primary key constraint does not have null values

Result - These both above queries returned as empty, hence confirmed that id of dish table is unique and does not contain any empty or null value.

3. We checked **first appeared** and **last appeared** columns to make sure all of their entries are between 1851 and 2018.

Query –

*select * from dish where (first_appeared not between 1851 and 2018)*

```
1 select *
2 from dish
3 where (first_appeared not between 1851 and 2018)
4
5
```

| | id | name | description | menus_appeared | times_appeared |
|---|-----|-----------------|-------------|----------------|----------------|
| 1 | 15 | Celery | NULL | 4248 | 4693 |
| 2 | 38 | Apple sauce | NULL | 721 | 829 |
| 3 | 96 | Coffee | NULL | 7750 | 8497 |
| 4 | 103 | Kipperd Herring | NULL | 231 | 236 |

Execution finished without errors.
Result: 60238 rows returned in 239ms
At line 1:
select *
from dish
where (first_appeared not between 1851 and 2018)

Figure 30: dish appeared not between 1851 and 2018

Result- 60,238 rows violated this constraint. Since, entries in the other columns of these rows and corresponding other table were valid. Hence, we did not delete these from the table.

4. **lowest_price and highest_price** – check for any particular dish if lowest_price is higher than highest_price.

Query –

*select * from dish where lowest_price > highest_price*

```

Queries1.sql
1 select * from dish where lowest_price > highest_price
2
3
4

Result: 0 rows returned in 396ms
At line 1:
select * from dish where lowest_price > highest_price

```

Figure 31: particular dish if lowest_price is higher than highest_price

Result -This query did not return any row, so no record violated this constraint.

5. Last appeared should not be 0 or null.

Query-

*select * from dish where last_appeared = '0';*

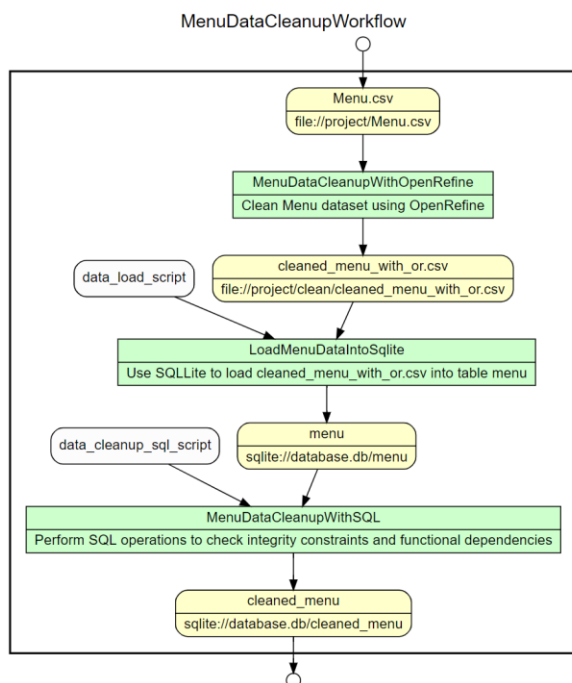
Result -We ran above query to obtain rows where last appeared was 0. We did not delete such rows as these dishes contained important data.

7. WORKFLOW MODEL

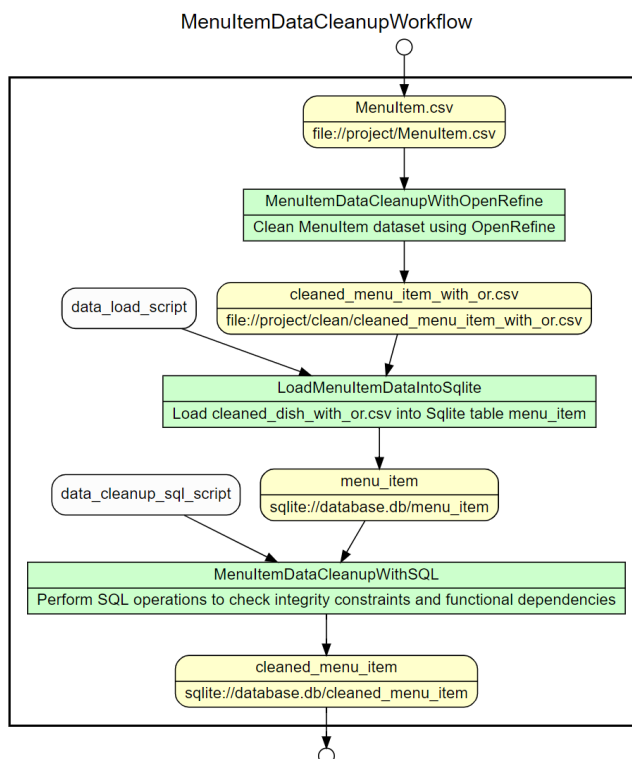
YesWorkflow model was used to create complete workflow graph. The complete work-flow graph which was used to clean whole data is shown below –

1. Complete Workflow

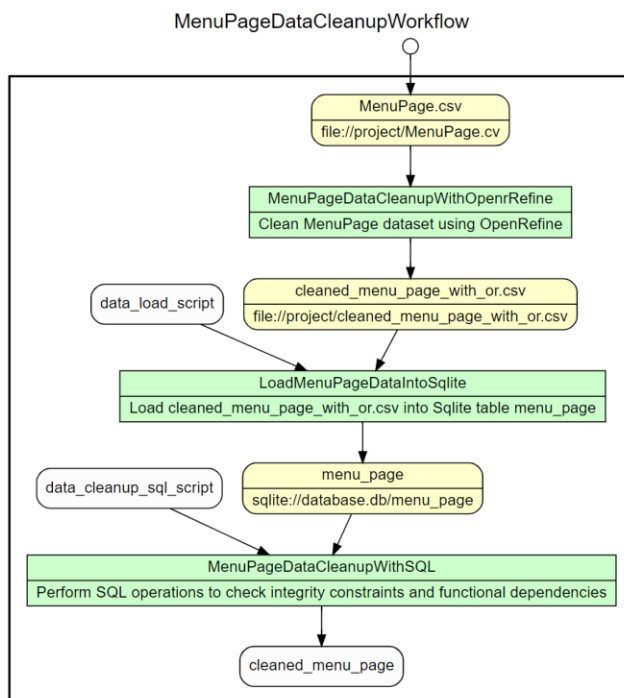
A. Menu dataset Cleanup -



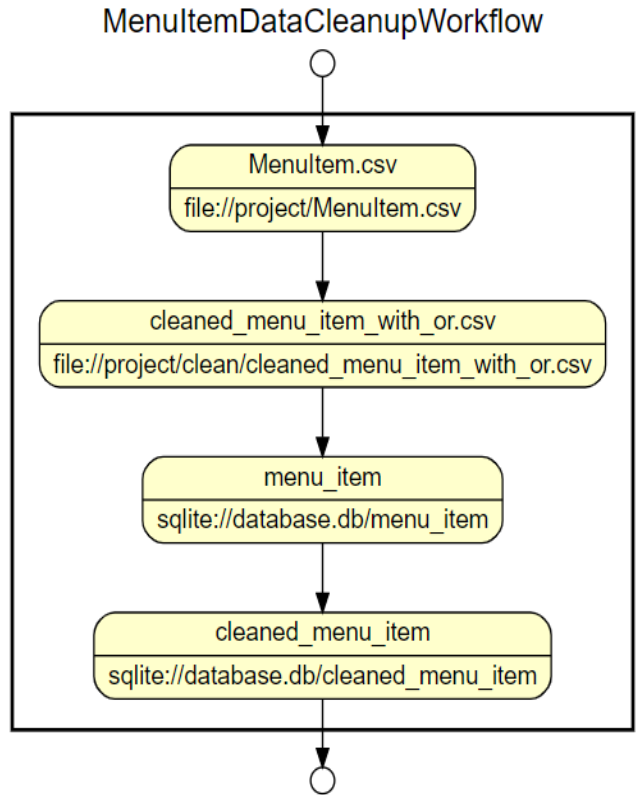
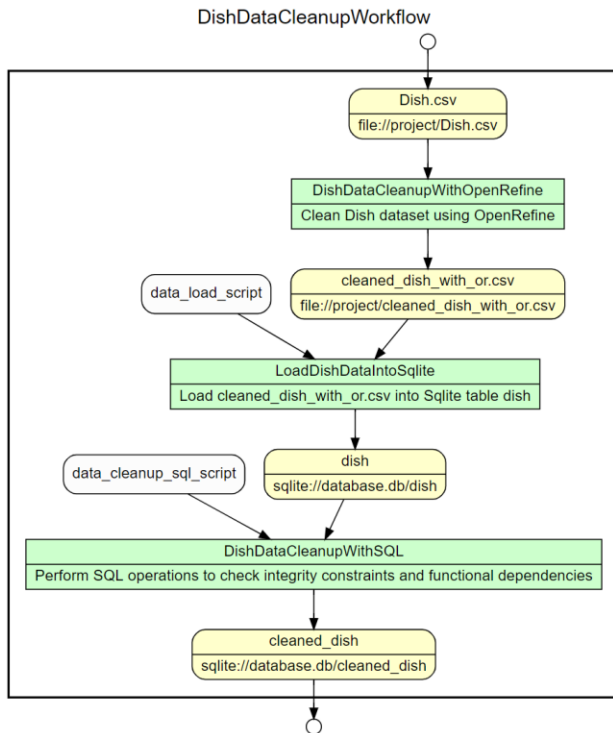
B. MenuItem Dataset Cleanup –



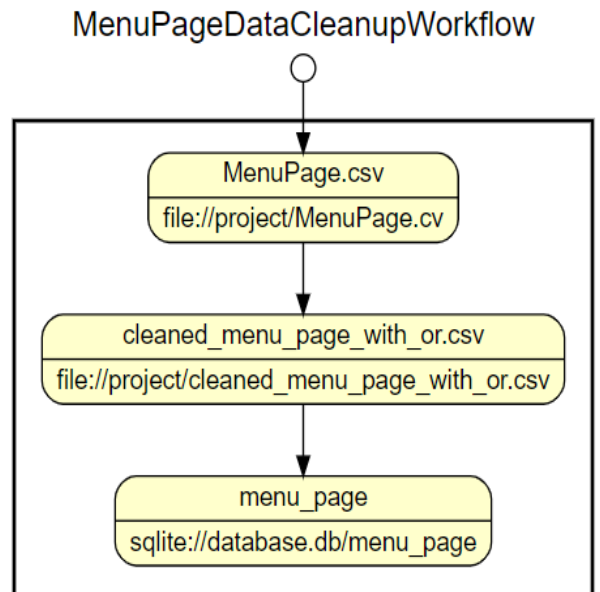
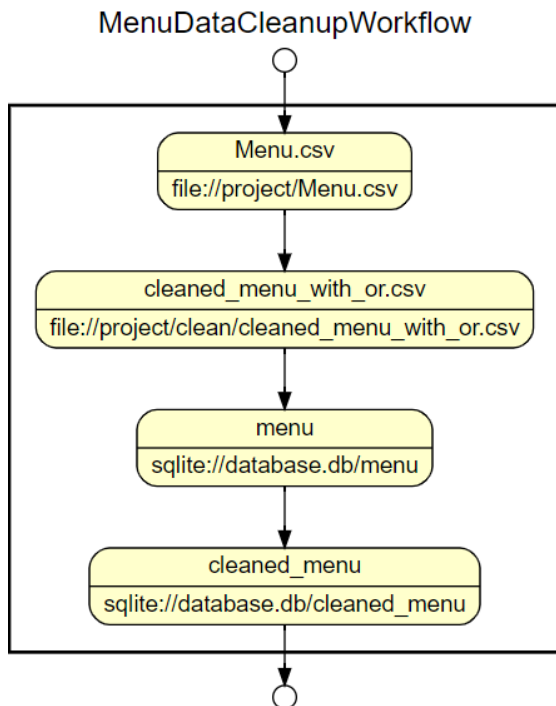
C. MenuPage dataset cleanup



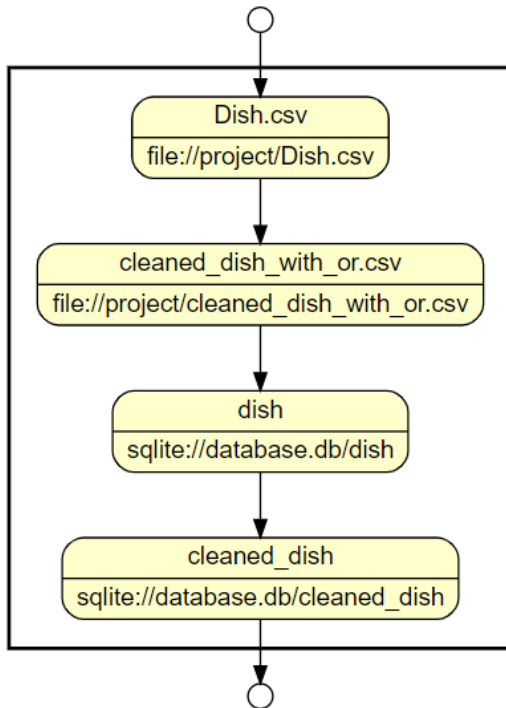
D. Dish dataset cleanup



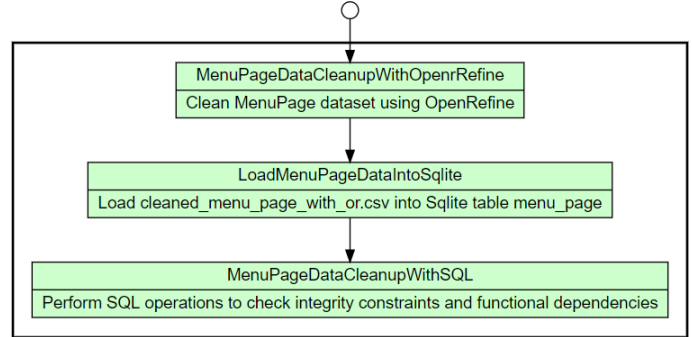
2. Workflow



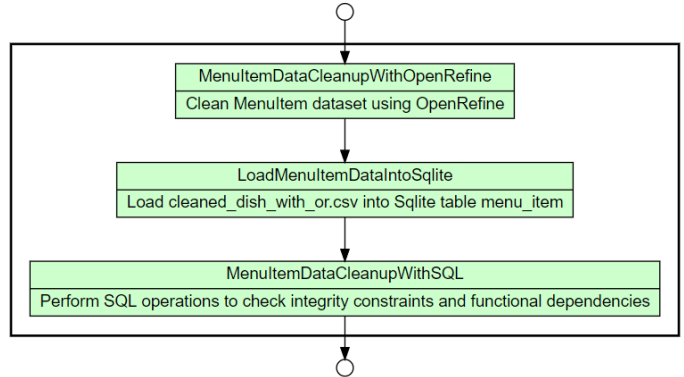
DishDataCleanupWorkflow



MenuPageDataCleanupWorkflow

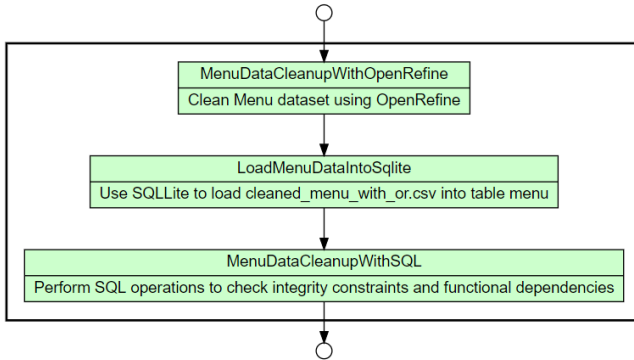


MenuItemDataCleanupWorkflow

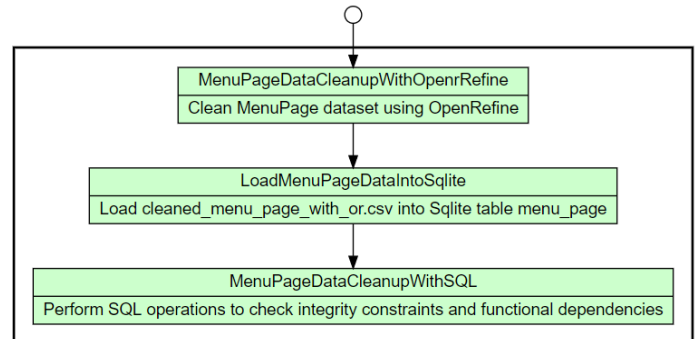


3. Dataflow

MenuDataCleanupWorkflow



MenuPageDataCleanupWorkflow



8. IMPROVEMENT AFTER DATA CLEANING

To verify, how data cleaning helped us in analyzing our use cases, we checked the performance of our targeted use case with raw dirty data and then we checked using cleaned data.

We have got some interesting finding, which we have listed below-

8.1 Target use case–

- A. What is the most popular dish?

Query-

```
select * from dish order by times_appeared desc
```

- B. What is the most popular dish according to location?

Query –

```
select distinct tmp1.name, tmp1.times_appeared,
m.location from menu m, (select tmp.name,
tmp.times_appeared, tmp.id, mp.menu_id from
MenuPage mp,(select d.name, d.times_appeared, mi.id
from dish d, MenuItem mi where d.id = mi.dish_id) tmp
where mp.menu_id = tmp.id) tmp1 where m.id =
tmp1.menu_id order by times_appeared DESC
```

8.2 Target use case execution with Dirty data –

- A. what is the most popular dish?

Performance Metric -

Execution Time – 1661 ms

Scanned no of rows – 428132

- B. What is the most popular dish according to location?

Performance Metric -

Execution Time – 7391 ms

Total no of rows in result – 15833

Screenshot -

```
Queries1.sql
1 --select * from dish order by times_appeared desc
2
3 select tmp1.name, tmp1.times_appeared, m.location from MenuRaw m,
4 (select tmp.name, tmp.times_appeared, tmp.id, mp.menu_id from MenuPageRaw mp,
5 (select d.name, d.times_appeared, mi.id from DishRaw d, MenuItemRaw mi
6 where d.id = mi.dish_id) tmp
7 where mp.menu_id = tmp.id) tmp1
8 where m.id = tmp1.menu_id order by times_appeared DESC
9
10
```

| | name | times_appeared | location |
|---|--------|----------------|---------------|
| 1 | Coffee | 8497 | Hotel Jermyn |
| 2 | Coffee | 8497 | Hotel Jermyn |
| 3 | Coffee | 8497 | Hotel Colombo |
| 4 | Coffee | 8497 | Hotel Colombo |
| 5 | Coffee | 8497 | Putnam House |

Execution finished without errors.
Result: 56094 rows returned in 7193ms
At line 1:
--select * from dish order by times_appeared desc
select tmp1.name, tmp1.times_appeared, m.location from MenuRaw m,
(select tmp.name, tmp.times_appeared, tmp.id, mp.menu_id from MenuPageRaw mp,
(select d.name, d.times_appeared, mi.id from DishRaw d, MenuItemRaw mi
where d.id = mi.dish_id) tmp
where mp.menu_id = tmp.id) tmp1
where m.id = tmp1.menu_id order by times_appeared DESC

Figure 32: Executing target use case with Dirty data set

8.3 Target use case execution with cleaned data –

- A. what is the most popular dish?

Performance Metric -

Execution Time – 1200 ms

Scanned no of rows – 428036

- B. What is the most popular dish according to location?

Performance Metric -

Execution Time – 7212ms

Total no of rows in result– 14355

Screenshot –

```
1 --select * from dish order by times_appeared desc
2
3 select distinct tmp1.name, tmp1.times_appeared, m.location from Menu m,
4 (select tmp.name, tmp.times_appeared, tmp.id, mp.menu_id from MenuPage mp,
5 (select d.name, d.times_appeared, mi.id from Dish d, MenuItem mi
6 where d.id = mi.dish_id) tmp
7 where mp.menu_id = tmp.id) tmp1
8 where m.id = tmp1.menu_id order by times_appeared DESC
9
10
```

| | name | times_appeared | location |
|---|--------|----------------|----------------------------|
| 1 | Coffee | 8497 | Hotel Jermyn |
| 2 | Coffee | 8497 | Hotel Colombo |
| 3 | Coffee | 8497 | Putnam House |
| 4 | Coffee | 8497 | Norddeutscher Lloyd Bremen |
| 5 | Coffee | 8497 | Hamburg Amerika Linie |

Execution finished without errors.
Result: 14355 rows returned in 7212ms
At line 3:
select distinct tmp1.name, tmp1.times_appeared, m.location from Menu m,
(select tmp.name, tmp.times_appeared, tmp.id, mp.menu_id from MenuPage mp,
(select d.name, d.times_appeared, mi.id from Dish d, MenuItem mi
where d.id = mi.dish_id) tmp
where mp.menu_id = tmp.id) tmp1
where m.id = tmp1.menu_id order by times_appeared DESC

Figure 33: Executing target use case with Clean data set

Result – By analyzing, above performance metrics we can safely conclude that –

- Retrieval time of target use case data reduced significantly.
- Quality of data also improved after data cleaning process.

9. CHALLENGES

The tool such as OpenRefine is a very important part of completing this project. However, it also has its challenges. We started cleaning up the biggest dataset -- Dish.csv and we immediately observed the limitations. Clustering took a lot of time and presented a vast number of selections as shown in the image below.

We started trusting the default value – merging and re-clustering it took more than 18 hours and didn't complete the process because we don't know if it was still running or not.

Blindly accepting default clustering value is another possible problem, it is very tedious validating it individually. We are eager to learn new tools that will eliminate these challenges.

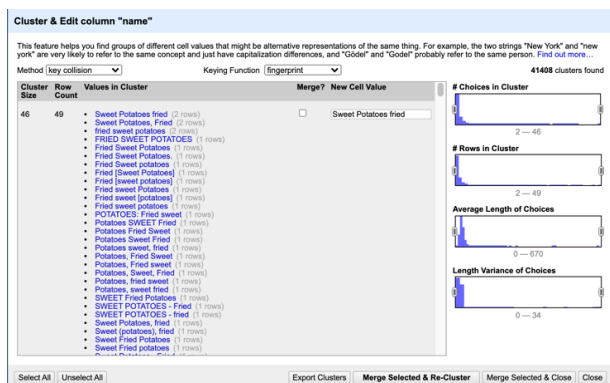


Figure 14: Cluster Selection

After trying hard, we found out few solutions which helped us to clean Dish.csv. At first, we increased heap allocation to 16 GB which helped us to clean Dish dataset. These settings can be changed in OpenRefine/Contents/Info.plist file.

Along with this, we have also updated REFINE MAX FORM CONTENT SIZE to 999999.

9. Work Assignments

This was one of the most interesting work we completed in last 1 month, it helped us in understand the whole data cleaning workflow step by step. We discussed as a team to give individual opportunity to clean the data, since there are 4 datasets available. While performing data cleaning activities, each team member captured their own cleaning step and at last, we selected one cleaned data set.

We also divided work among us as mentioned in below table.

| Member | Responsibility |
|---------------|--|
| Michael/Mohit | Clean NYPL dataset using Open Refine. Develop relational database schema. Formulate SQL code.[6] |
| Upendra | Clean NYPL dataset using OpenRefine. Create workflow models. [4] [5] Possible Python code to complete other data cleaning activities. [8] |

10. SUMMARY

We really enjoyed accomplishing this project and this is the type of project that we can contribute to our product teams back in the office immediately.

By this project, we can easily demonstrate that Data Cleaning is a very important step in any data analytic project. It helps in retrieving data faster and also good quality data.

11. NEXT STEPS

We also took Data Visualization in parallel in this semester and concepts learnt in this class immensely helped us in completion of our Data Visualization project.

In this project, we mainly used OpenRefine, SQLite and Python to clean data, in future, we want to explore some more tools for data cleaning.

12. REFERENCES

- [1] <http://menus.nypl.org/about>
- [2] <https://regexr.com/>
- [3] Data Munging: String Manipulation, Regular Expressions, and Data Cleaning DOI - 10.1002/9781119092919.ch4
- [4] https://www.researchgate.net/publication/335136628_Towards_Automated_Data_Cleaning_Workflows
- [5] Model based approach for developing Data Cleaning Solutions <https://dl.acm.org/doi/10.1145/2641575>
- [6] <https://www.sqlite.org/index.html>
- [7] ER Diagram - <https://online.visual-paradigm.com/drive/#diagramlist:proj=0&new=ERDiagram>
- [8] <https://realpython.com/python-data-cleaning-numpy-pandas/>
- [9] Clean data using Open Refine: <https://openrefine.org/>