End-to-end Data Cleaning Workflow for NYPL Restaurant Data

Reshma Lal Jagadheesh, Nithish Kaviyan Dhayananda Ganesh

I. Introduction

DATA Cleaning is the area in which most data scientists spend their maximum time on. It is also the area less fun to work with and needs a lot of manual tasks. With modern data cleaning tools, this laborious manual task has been reduced to some extent but not completely automated. In this project, we provide step-by-step procedure we carried out to clean a transcribed restaurant menu data, managed by the New York Public Library (NYPL). We also provide workflow of the various data cleaning operations performed to automate the cleaning process.

II. Dataset

The Dataset used for this project is taken from the New York Public Library website. The data comes from the library's menu collection, housed in the Rare Book Division. It has a collection of approximately 45,000 menus, out of which over 25,000 were added by Miss Frank E. Buttolph (1850-1924). Of the 45,000 menus, a quarter of them have been digitized so far and made available in the NYPL Digital Gallery.

The menus have been transcribed by volunteers and the data in the NYPL website is being updated on the 1st and 16th of every month. The data used in this project is from 16th November, 2018.

The data consists of 4 csv files: *Menu*, *MenuPage*, *MenuItem*, and *Dish*. Description of the files are given below.

TABLE I
DESCRIPTION OF THE DATA

File	Number of attributes	Number of Records	File size
Menu	20	17545	3.08 MB
MenuPage	7	66937	4.45 MB
MenuItem	9	1333629	113 MB
Dish	9	426124	25.3 MB

A. Menu

Menu is the primary element of the dataset. It contains unique id for each menu, and other attributes including event for which the menu was prepared, location of the place in which the menu was used, occasion for which it was used, language in which the menu existed, total number of pages and other such fields. The metadata description of the table is given below:

- ***id*** : unique identifier for each instance of the table
- ***name*** : name of the menu (mostly blank)
- ***sponsor*** : sponsor of the menu (name of the restaurant)
- ***event*** : event for which the menu was prepared
- ***venue*** : venue where the menu was used
- ***place*** : description of the city, state of the place where the menu was used
- ***physical_description*** : physical description and dimensions of the menu (card or folder)
- ***occasion*** : occasion for which the menu was used
- ***notes*** : special comments on the menu
- ***call_number*** : menu's call number in the NYPL
- ***keywords*** : keywords for the menu (completely blank)
- ***language*** : language in which the menu was printed (completely blank)
- ***date*** : date in which the menu was collected (format YYYY-MM-DD)
- ***location*** : location where the menu was used
- ***location_type*** : type of the location (completely blank)
- ***currency*** : currency in which items in the menu were charged
- ***currency_symbol*** : symbol of the currency
- ***status*** : transcription status of the menu (complete or under review)
- ***page_count*** : number of pages in the menu
- ***dish_count*** : number of dishes in the menu

B. MenuPage

MenuPage describes about different pages in the menu. It contains an unique id for each record and it also reference to the Menu table through *menu_id*.The metadata description of the table is given below:

- ***id*** : unique id for each instance of the table
- ***menu_id*** : menu to which the menu page instance belongs
- ***page_number*** : page number in the menu
- ***image_id*** : id of the image of the menu page in NYPL directory
- ***full_height*** : height of the menu
- ***full_width*** : width of the menu
- ***uuid*** : unique identifier id for the menu

C. MenuItem

Each instance in MenuItem describes has an unique identifier and describes about the items in the menu. It also references to MenuPage and Dish tables through *menu_page_id* and *dish_id* respectively. The metadata description of the table is given below:

- ***id*** : unique id for each instance of MenuItem
- ***menu_page_id*** : menu page to which the item belongs (refers to the id in MenuPage table)

- ***price*** : price of the item
- ***high_price*** : price of the largest portion of the item
- ***dish_id*** : dish to which the record belongs (refers to the id in Dish table)
- ***created_at*** : date and time at which the record was created
- ***updated_at*** : date and time at which the record was last updated
- ***xpos*** : x-axis position of the item on the scanned
- ***name*** : name of the dish
- ***menus_appeared*** : image
- ***ypos*** : y-axis position of the item on the scanned image

D. Dish

Dish table contains details about different dishes appeared in the menus. It contains an unique identifier for each dish. The metadata description of the table is given below:

- ***id*** : unique id for each instance of Dish
- ***name*** : name of the dish
- ***menus_appeared*** : number of menus in which the dish appeared
- ***times_appeared*** : number of times the dish appeared
- ***first_appeared*** : year at which the dish appeared for the first time
- ***last_appeared*** : year at which the dish appeared for the last time
- ***lowest_price*** : lowest price this dish was sold for
- ***highest_price*** : highest price this dish was sold for

III. Use Case

Once the data is properly cleaned there are many possible potential use cases for the data.

1) Can be used to find out some of the most popular dishes for a particular region. Example: Italy - ”Pizza”
2) Estimate the price of the dishes based on the data available
3) Select a menu for a particular occasion. For example Thanksgiving- ”Turkey”.
4) To find out the location of restaurant for unique food. Example- sushi (not found in all restaurants)
5) Find out how the price of a dish has changed over time. Also if possible, find out the possible reason why the price has increased or decreased or remained the same.
6) Analyse the price of the dishes based on the region. Example: Is European food more expensive than Indian food?

IV. How dirty is the data?

The data is really messy for any actual use case. The Menu dataset has a lot of Special characters, trailing white spaces in between, similar words not being clustered together (example luncheon and lunch were treated as separate attributes). The dataset was primarily cleaned using OpenRefine, Python and SQL. There were many null values within the data. A basic visualisation of the null values in each dataset is shown below.

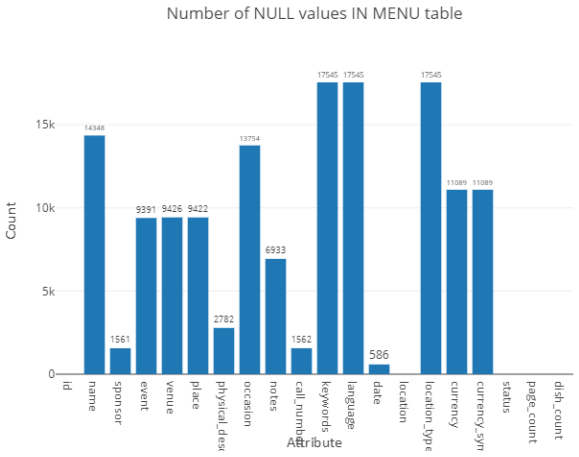

Fig. 1. Number of null values in each attribute of Menu table

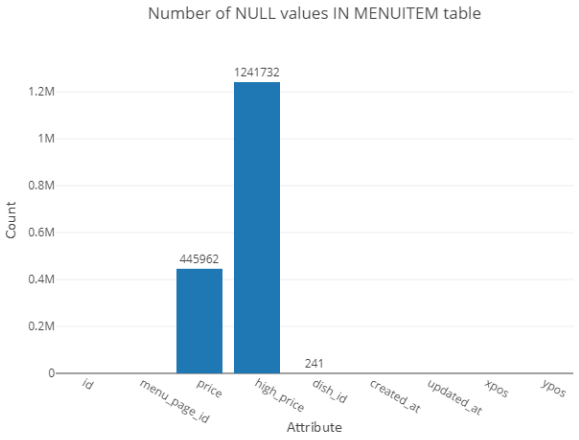

Fig. 2. Number of null values in each attribute of MenuItem table

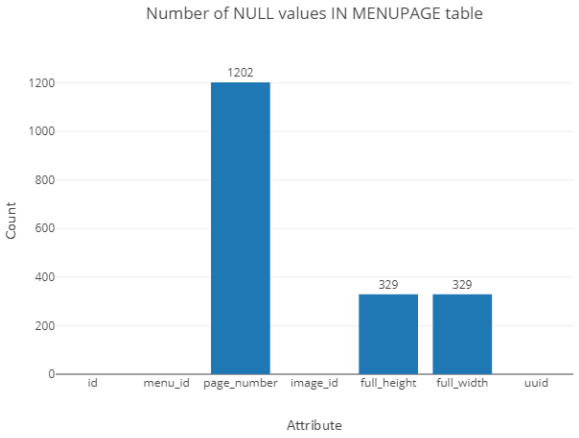

Fig. 3. Number of null values in each attribute of MenuPage table

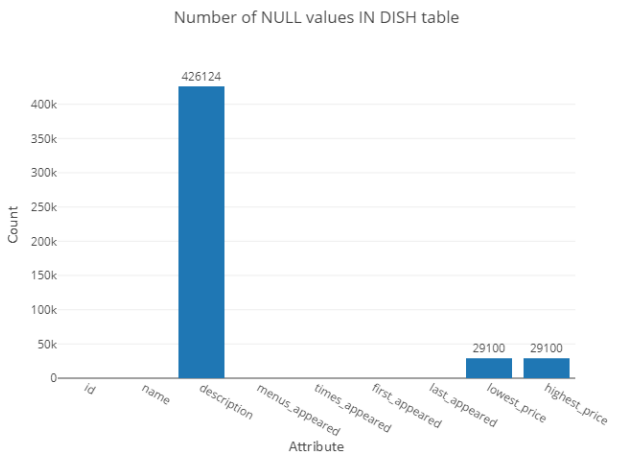

Fig. 4. Number of null values in each attributes of Dish table

V. Data Cleaning Using OpenRefine

The basic cleaning operations such as removing consecutive white spaces, trim leading and trailing white spaces and removing special characters were done using OpenRefine. One of the most important advantage of using OpenRefine is it's clustering feature using Text faucet. Normally in any other data cleaning tool or software this process will be very tedious and time consuming and also there is a chance of having many errors. Using OpenRefine's clustering feature all these problems were rectified.

Each dataset was loaded individually into OpenRefine and all the cleaning operations were performed step by step. The Menu dataset was the untidiest of them all, and it required a lot of cleaning to be done.

A. *Menu*

In the Menu dataset the following Columns were cleaned.

*1) **Sponsor**:* The following Data cleaning operations were performed on the Sponsor column :

- collapse consecutive white spaces
- Trim leading and trailing white spaces
- Use text facet and cluster using key collision methods such as fingerprint, ngram fingerprint and metaphone3
- Remove special characters such as ?,(,),",[,] and \

The number of cells that were affected due to the data cleaning operation shown above is displayed in Table II.

TABLE II
Data Cleaning on Sponsor Column

Data Cleaning operation	Number of cells changed
collapse consecutive white spaces	127
trim leading and trailing white spaces	15
Change values to Title case	8872
Cluster- Key Collision - fingerprint	4249
Cluster- Key Collision - ngram fingerprint	1842
Cluster- Key Collision - metaphone3	5463
Remove ?	61
Remove (and)	143
Remove "	329
Remove [and]	61
Remove \	137

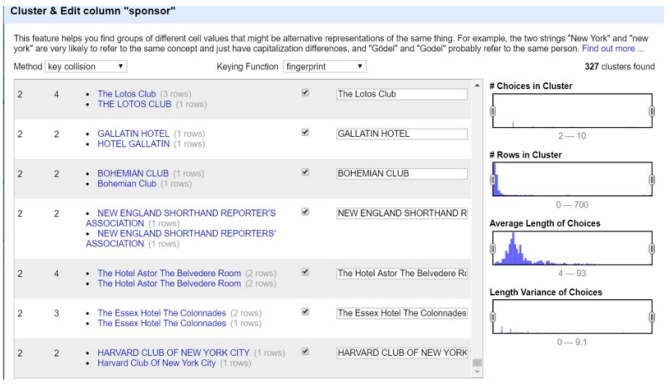

Fig. 5. Clustering Sponsor column using key collision- fingerprint

Fig. 6. Clustering Sponsor column using key collision- metaphone3

Fig. 7. Clustering Sponsor column using key collision- ngram fingerprint

*2) **Event**:* The following Data cleaning operations were performed on the Event column :

- collapse consecutive white spaces
- Trim leading and trailing white spaces
- Use text facet and cluster using key collision methods such as fingerprint, ngram fingerprint and metaphone3
- Remove special characters such as ?,(,),",[and]

The number of cells that were affected due to the data cleaning operation shown above is displayed in Table III.

TABLE III
DATA CLEANING ON EVENT COLUMN

Data Cleaning operation	Number of cells changed
collapse consecutive white spaces	6
trim leading and trailing white spaces	3
Change values to Title case	7829
Cluster- Key Collision - fingerprint	5314
Cluster- Key Collision - ngram fingerprint	2325
Cluster- Key Collision - metaphone3	1537
Remove ?	44
Remove (and)	30
Remove ”	29
Remove [and]	41

*3) **Venue**:* The following Data cleaning operations were performed on the Venue column :

- collapse consecutive white spaces
- Trim leading and trailing white spaces
- Use text facet and cluster using key collision methods such as fingerprint, ngram fingerprint and metaphone3
- Remove special characters such as ?,(,),:,[and]

The number of cells that were affected due to the data cleaning operation shown above is displayed in Table IV.

TABLE IV
DATA CLEANING ON VENUE COLUMN

Data Cleaning operation	Number of cells changed
collapse consecutive white spaces	0
trim leading and trailing white spaces	0
Change values to Upper case	9
Cluster- Key Collision - fingerprint	2251
Cluster- Key Collision - ngram fingerprint	32
Cluster- Key Collision - metaphone3	5052
Remove ?	28
Remove (and)	116
Remove ;	1972
Remove [and]	2

*4) **Place**:* The following Data cleaning operations were performed on the Place column :

- collapse consecutive white spaces
- Trim leading and trailing white spaces
- Use text facet and cluster using key collision methods such as fingerprint, ngram fingerprint and metaphone3
- Remove special characters such as ?,(,),:,[and]

The number of cells that were affected due to the data cleaning operation shown above is displayed in Table V.

TABLE V
DATA CLEANING ON PLACE COLUMN

Data Cleaning operation	Number of cells changed
collapse consecutive white spaces	0
trim leading and trailing white spaces	0
Change values to Upper case	899
Cluster- Key Collision - fingerprint	3251
Cluster- Key Collision - ngram fingerprint	2435
Cluster- Key Collision - metaphone3	3256
Remove ?	326
Remove (and)	164
Remove ”	522
Remove ;	284
Remove [and]	518

*5) **Occasion**:* The following Data cleaning operations were performed on the Occasion column :

- collapse consecutive white spaces
- Trim leading and trailing white spaces
- Use text facet and cluster using key collision methods such as fingerprint, ngram fingerprint and metaphone3
- Remove special characters such as ?,(,),:,[and]

The number of cells that were affected due to the data cleaning operation shown above is displayed in Table VI.

TABLE VI
DATA CLEANING ON OCCASION COLUMN

Data Cleaning operation	Number of cells changed
collapse consecutive white spaces	3
trim leading and trailing white spaces	0
Change values to Upper case	17
Cluster- Key Collision - fingerprint	2779
Cluster- Key Collision - ngram fingerprint	272
Cluster- Key Collision - metaphone3	1362
Remove ?	98
Remove (and)	395
Remove ”	5
Remove ;	2628
Remove [and]	53

*6) **Location**:* The following Data cleaning operations were performed on the Location column :

- collapse consecutive white spaces
- Trim leading and trailing white spaces
- Use text facet and cluster using key collision methods such as fingerprint, ngram fingerprint and metaphone3
- Remove special characters such as ?,(,),:,[and]

The number of cells that were affected due to the data cleaning operation shown above is displayed in Table VII.

TABLE VII
DATA CLEANING ON LOCATION COLUMN

Data Cleaning operation	Number of cells changed
collapse consecutive white spaces	555
trim leading and trailing white spaces	15
Change values to Title case	1475
Cluster- Key Collision - fingerprint	4109
Cluster- Key Collision - ngram fingerprint	6156
Remove ?	107
Remove (and)	67
Remove ”	418
Remove [and]	224

*7) **Left as is**:* The columns id, name, keyword, language, status, page_count and dish_count were left as it is without making any changes in Menu dataset. There were no changes made in MenuPage, MenuItem and Dish dataset as well.

VI. WORKFLOW DIAGRAM FOR OPENREFINE

The Workflow graph for the OpenRefine cleaning operation that was performed on the data is shown in Fig. 22,23 and 24.

VII. DATA CLEANING USING PYTHON

After cleaning Menu table using OpenRefine, it was saved as a *csv* file. The updated Menu file, along with other three files, were cleaned in Python to remove columns with all unkown values and also date columns were checked if they were in valid format (i.e. if the years, months and dates are valid). The *csv* files were read into python as *pandas dataframes*.

A. *Removal of unknown columns*

1) Menu table: Three columns in Menu, *language, keywords, location_type* has all unknown values. This was checked using a single line of code in python which is shown in the image below.

Fig. 8. Drop unknown columns in Menu

The code checks the number of rows which does not have an unknown value in the corresponding column. The above mentioned three tables had unknown values and were deleted from Menu table. Thus, the columns in the Menu table has been reduced to 17 from 20.

2) Dish table: Description column in Dish table was found to have all unknown values and was removed from the Dish table. This operation is shown in the following figure:

Fig. 9. Drop unknown columns in Dish

After removing the column, number of columns in Dish table was reduced to 8 from 9.

B. *Validity check of dates*

One task which was always tempting to perform was to check if the dates are in correct format and if they are valid. Pandas had an easy way to convert *object* types to *datetime* type by which the *created_at* and *updated_at* columns in MenuItem table were found to have valid date and time. But for the *date* column in Menu which only has the date, a custom function was written to check if each entry in that column is a valid date. The below figure shows the function and the result of validity check operation:

Fig. 10. Function to check if the date is valid

As with the output in the above figure, the dates in Menu table were found to be in correct format.

After the cleaning operations in OpenRefine and Python, the tables had following dimensions:

TABLE VIII
DIMENSIONS OF TABLE AFTER CLEANING

File	Number of attributes	Number of Records
Menu	17	17545
MenuPage	7	66937B
MenuItem	9	1333629
Dish	8	426124

VIII. ENTITY RELATIONSHIP DIAGRAM

After performing different cleaning operations, a relational database was created using the four tables to check for integrity constraint violations. ER diagram for the database was designed using SQLDBM online database design tool. The ER diagram is shown in the figure below:

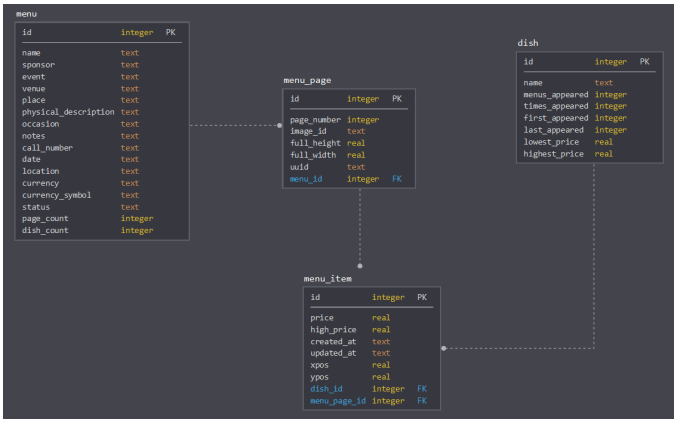

Fig. 11. ER Diagram for the relational database

IX. CHECKING INTEGRITY CONSTRAINTS USING SQLITE

Based on the relational schema as shown in the ER diagram, a database was created in SQLITE database using *sqlite3* package in Python. The database was checked integrity constraint violations all the four tables.

A. *Menu*

The following ICs were checked for Menu table:

- *id* column, being the primary key checked for uniqueness

  Query
  select id, count(id) as count_id
  from menu
  group by id
  having count(id) $>$ 1;

- Also, *id* column was checked for any null values

  Query
  select id

from menu
where id is **null**;

Fig. 12. Primary key check for Menu table

Both the queries above returned empty list, confirming that primary keys are unique and not null in Menu table.

- *page_count* column was checked if it has any value as '0'. It also returned an empty table, showing that page counts in the menu are also non-zero integers.

Query
select *
from menu
where page_count = 0;

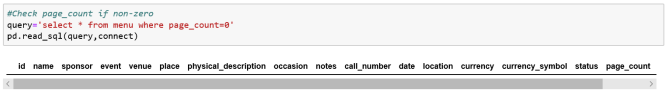

Fig. 13. Check if page_count is non-zero

B. *MenuPage*

The following ICs were checked for MenuPage table:

- *id* column, being the primary key was checked for uniqueness

Query
select id, count(id) as count_id
from menu_
group by id
having count(id) > 1;

- Also, *id* column was checked for any null values

Query
select id
from menu_
where id is **null**;

Fig. 14. Primary key check for MenuPage table

Both the queries above returned empty list, confirming that primary keys are unique and not null in MenuPage table.

- *page_number* column was checked for any value with '0'. It also returned an empty table.

Query
select *
from menu_page
where page_number = 0;

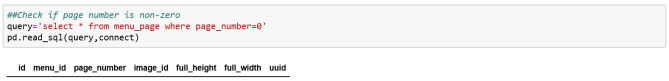

Fig. 15. Check if page_number is non-zero

- *page_number* column was checked if its values are less than or equal to the total page count of the menu (*page_number* column in Menu table). This check was done by INNER JOIN between Menu and MenuPage tables.

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

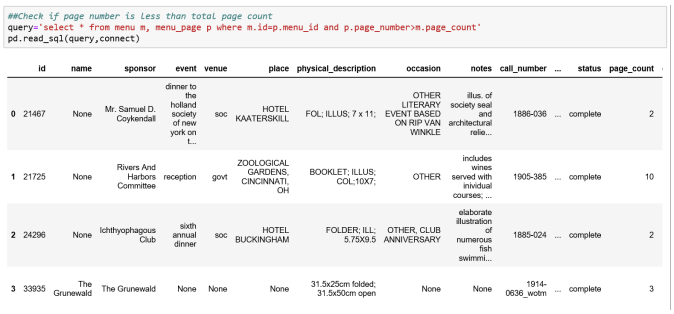

Fig. 16. Check if page_number is within total page count

The above figure shows screenshot of execution of the query. There were 6 rows which violated this constraint. However, the records were not deleted from the table since it also might cause removal of other important details from those records.

C. *MenuItem*

The following ICs were checked for MenuItem table:

- *id* column, being the primary key was checked for uniqueness

Query
select id, count(id) as count_id
from menu_item

group by id
having count(id) $>$ 1;

- Also, *id* column was checked for any null values

Query
select id
from menu_item
where id is **null**;

```
##Check uniqueness of primary key
query='select id, count(id) as count_id from menu_item group by id having count(id)>1'
pd.read_sql(query,connect)
```

id	count_id

```
##Check if primary key is null
query='select id from menu_item where id is null'
pd.read_sql(query,connect)
```

id

Fig. 17. Primary key check for MenuItem table

Both the queries above returned empty list, confirming that primary keys are unique and not null in Menu table.

- *dish_id* column is checked if its entries are present in *id* column of Dish table

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

```
##Check if dish_id in menu_item is present in dish table
query='select * from menu_item where dish_id not in (select id from dish)'
pd.read_sql(query,connect)
```

	id	menu_page_id	price	high_price	dish_id	created_at	updated_at	xpos	ypos
0	619133	51020	NaN	None	220797.0	2011-10-30 14:27:33 UTC	2011-10-30 14:27:33 UTC	0.605714	0.215599
1	837354	60235	0.20	None	329183.0	2012-03-08 23:28:32 UTC	2012-03-08 23:28:32 UTC	0.664286	0.173588
2	1047160	69117	0.45	None	395403.0	2012-08-14 09:52:01 UTC	2012-08-14 09:52:01 UTC	0.377333	0.469635

Fig. 18. Check if all entries in dish_id are present in id column of Dish table

As shown in the above figure, it was observed that three instances were found in *id* column of Dish table. These rows were removed from MenuItem table.

D. *Dish*

The following ICs were checked for Dish table:

- *id* column, the primary key was checked for uniqueness and for null values
  Similar to other tables, the primary key was unique and not null.

- *first_appeared* and *last_appeared* columns are checked if all of their entries are between 1851 and 2018.

Query
select *
from dish
where (first_appeared **not between** 1851 **and** 2018)
or (last_appeared **not between** 1851 **and** 2018);

```
In [43]: ##Check if first appeared and last appeared are within 1851 and 2018
query='select * from dish where (first_appeared not between 1851 and 2018) or (last_appeared not between 1851 and 2018)'
pd.read_sql(query,connect)
```

58389	518426	Cream Fritters à la Reine	1	1	0	0	0.00	0.00
58390	518427	Stuffed Green Peppers, Sauce Madère	1	1	0	0	0.00	0.00
58391	518428	Pork Chops Sauté, Sauce Maltaisé	1	0	0	0	0.00	0.00
58392	518429	Pork Chops Sauté, Sauce Maltaise	1	1	0	0	0.00	0.00
58393	518430	Gâteau Solferino	1	1	0	0	0.00	0.00
58394	518431	Gelee au rhum	1	1	0	0	0.00	0.00
58395	518432	riké à l'indienne chaud ou froid	1	1	0	0	0.00	0.00
58396	518433	Sandemann Tawney	1	1	0	0	0.00	0.00
58397	518434	Cocoa, Per Pot	1	1	0	0	0.00	0.00
58398	518435	Illuminated Ice-cream, Macaroon Cake, Ornament...	1	1	0	0	0.00	0.00
58399	518436	Roquefort, American and Pine-apple Cheese	1	1	0	0	0.00	0.00

58400 rows × 8 columns

Fig. 19. Check if first_appeared and last_appeared are between 1851 and 2018

58,400 rows had violated this constraint. However, entries in the other columns of these rows were valid. Hence, they were not deleted from the table.

- *first_appeared* and *last_appeared* columns are checked if *first_appeared* is less than *last_appeared*

Query
select *
from dish
where first_appeared $>$ last_appeared;

```
##Check if first_appeared is less than last_appeared
query='select * from dish where first_appeared > last_appeared'
pd.read_sql(query,connect)
```

	id	name	menus_appeared	times_appeared	first_appeared	last_appeared	lowest_price	highest_price
0	164029	Clear beef broth	0	1	1900	0	0.25	0.25
1	204888	Hot roast beef with gravy	0	1	1900	0	0.25	0.25
2	250693	SURI LEBERLI - Shredded Calf's Liver Flambe in...	0	1	1945	0	NaN	NaN
3	250699	SWISS MINCED VEAL, ROESTI	0	1	1945	0	NaN	NaN
4	301736	Cafe Glacee	0	2	1940	0	0.40	0.40
5	309629	Garlic Butter	0	1	1947	0	0.40	0.40

Fig. 20. Check if first_appeared is less than last_appeared

Five rows violated this constraint.

- checked if *lowest_price* is less than *highest_price*

Query
select *
from dish
where lowest_price $>$ highest_price;

```
##Check if lowest_price is less than highest_price
query='select * from dish where lowest_price > highest_price'
pd.read_sql(query, connect)
```

id	name	menus_appeared	times_appeared	first_appeared	last_appeared	lowest_price	highest_price

Fig. 21. Check if lowest_price is less than highest_price

When executed the above query, an empty table showed up. Thus lowest price is less than highest price for all entries.

X. Create Workflow Model using YesWorkflow

The complete workflow graph for all the cleaning operations that were performed in the data is shown in Fig. 25, 26 and

27.

XI. Conclusion

In this project, we presented the steps we followed to clean the NYPL restaurant menu data to prepare it for further analysis (as mentioned in the use-cases). Cleaning was initially done using OpenRefine and Python. The cleaned data were loaded into SQLITE database and checked for integrity constranit violations. All the operations performed were modelled as workflow graph using YesWorkflow tool.

With respect to the tools used, OpenRefine provided more convenient features to cluster similar texts and also the GUI interface, provided better visualization of the prospective changes. Python script provided a convenient way to load the data into SQLITE database via its *sqlite3* package, and enabled easy checking of the IC violations.

The entire cleaning process was time-consuming. However, the workflow provided can be used as a guideline to perform cleaning of similar datasets for similar use-cases.

XII. Acknowledgement

We would like to thank our Professor Bertram Ludaescher, Teaching Assistants Nikolaus Nova Parulian and Jessica Yi-Yun Cheng for their valuable insights and useful suggestions for implementing this project.

XIII. Reference

1) http://openrefine.org/documentation.html
2) https://github.com/nikolausn/OR2YW/blob/master/README.md
3) https://pypi.org/project/or2ywtool/
4) https://www.tableau.com/learn/training
5) http://menus.nypl.org/
6) https://thetakeout.com/these-historical-menus-show-how-drastically-the-way-we-1798258043
7) https://pandas.pydata.org/
8) https://www.w3schools.com/sql/

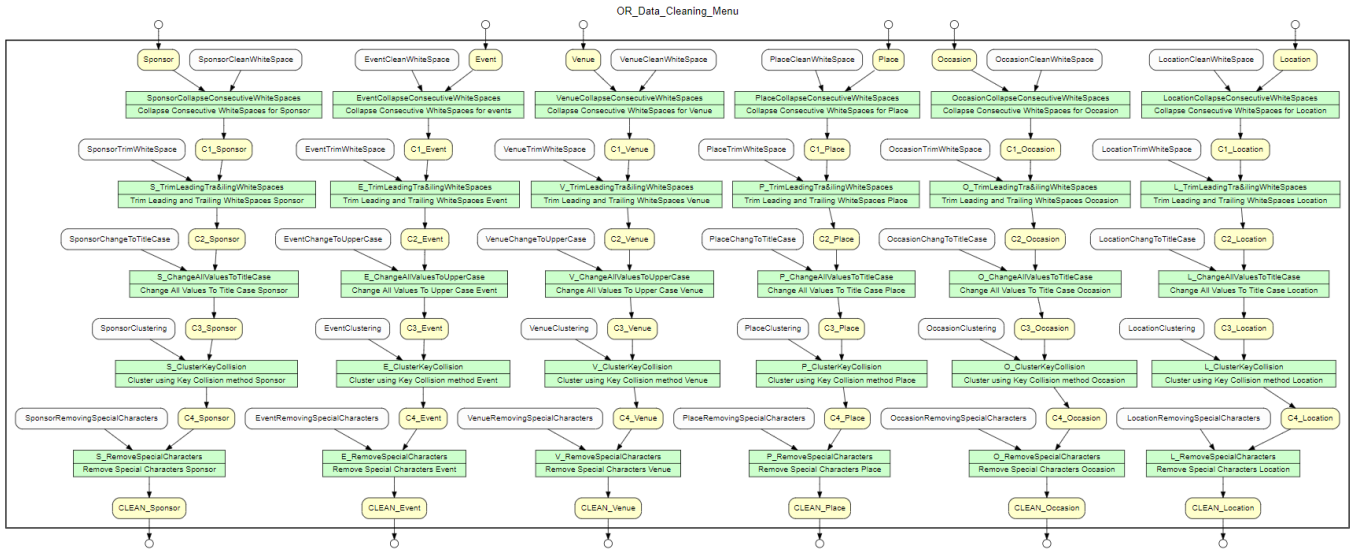

Fig. 22. Yes workflow graph for OpenRefine

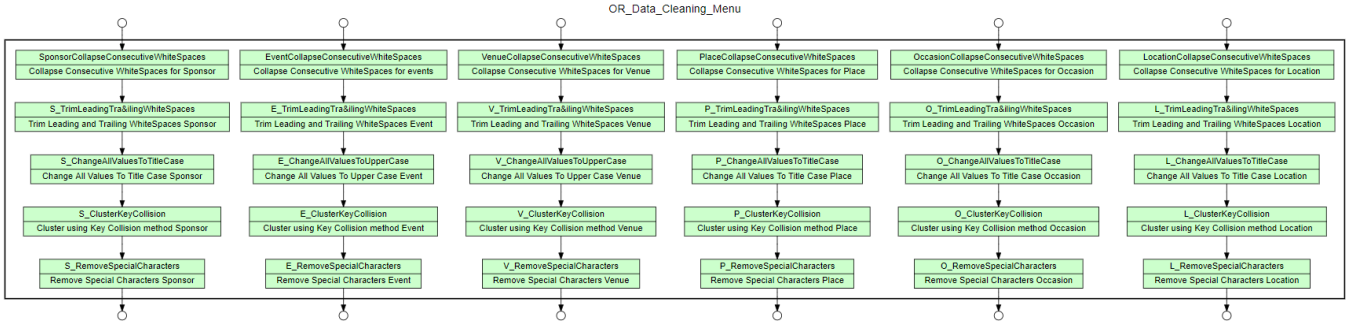

Fig. 23. Operations workflow graph for OpenRefine

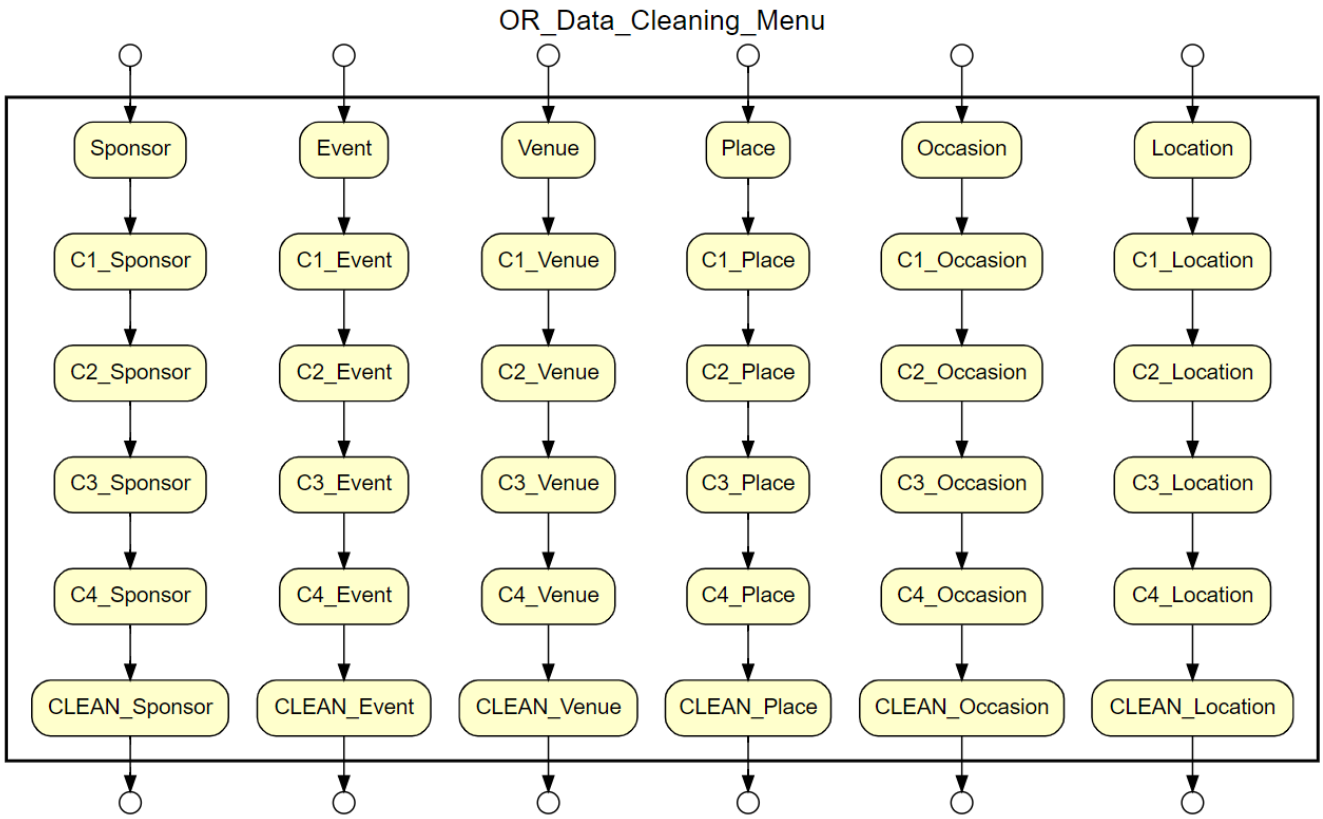

Fig. 24. Data workflow graph for OpenRefine

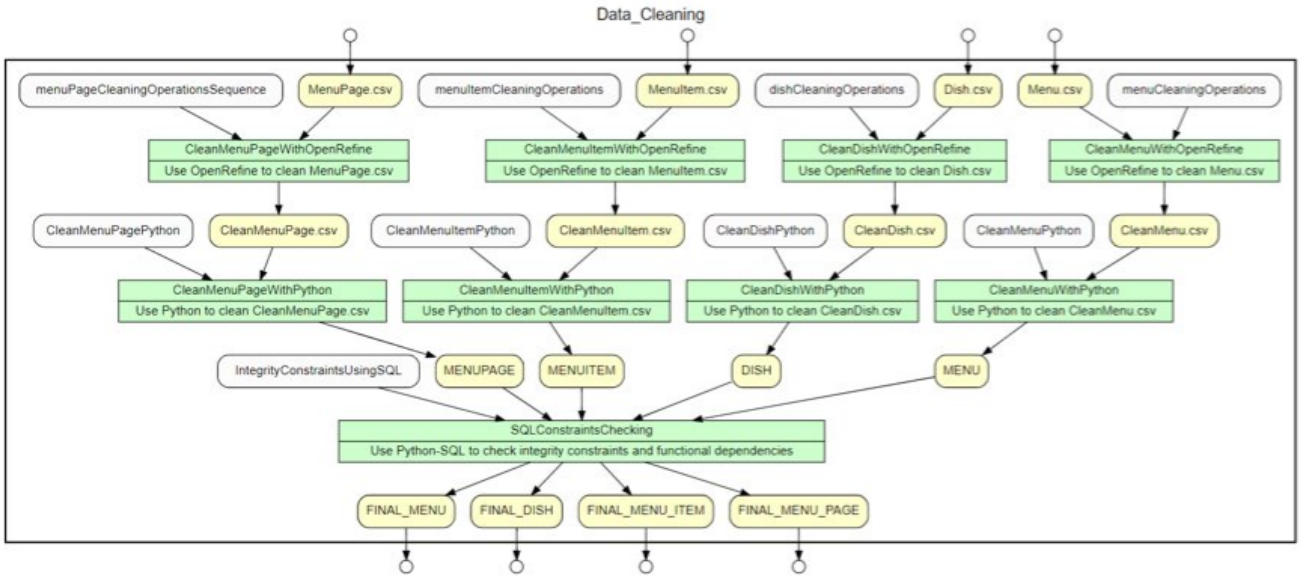

Fig. 25. Yes Workflow graph for all Data cleaning operations that were performed

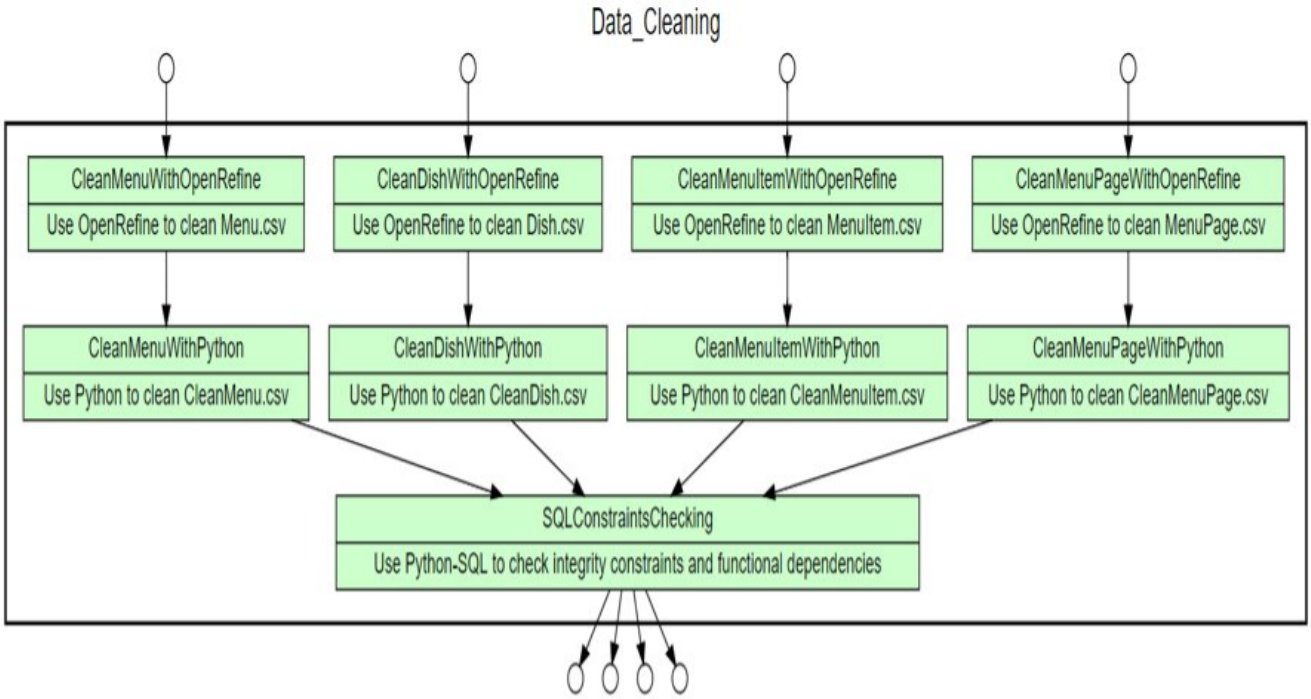

Fig. 26. Operations workflow graph for Data Cleaning

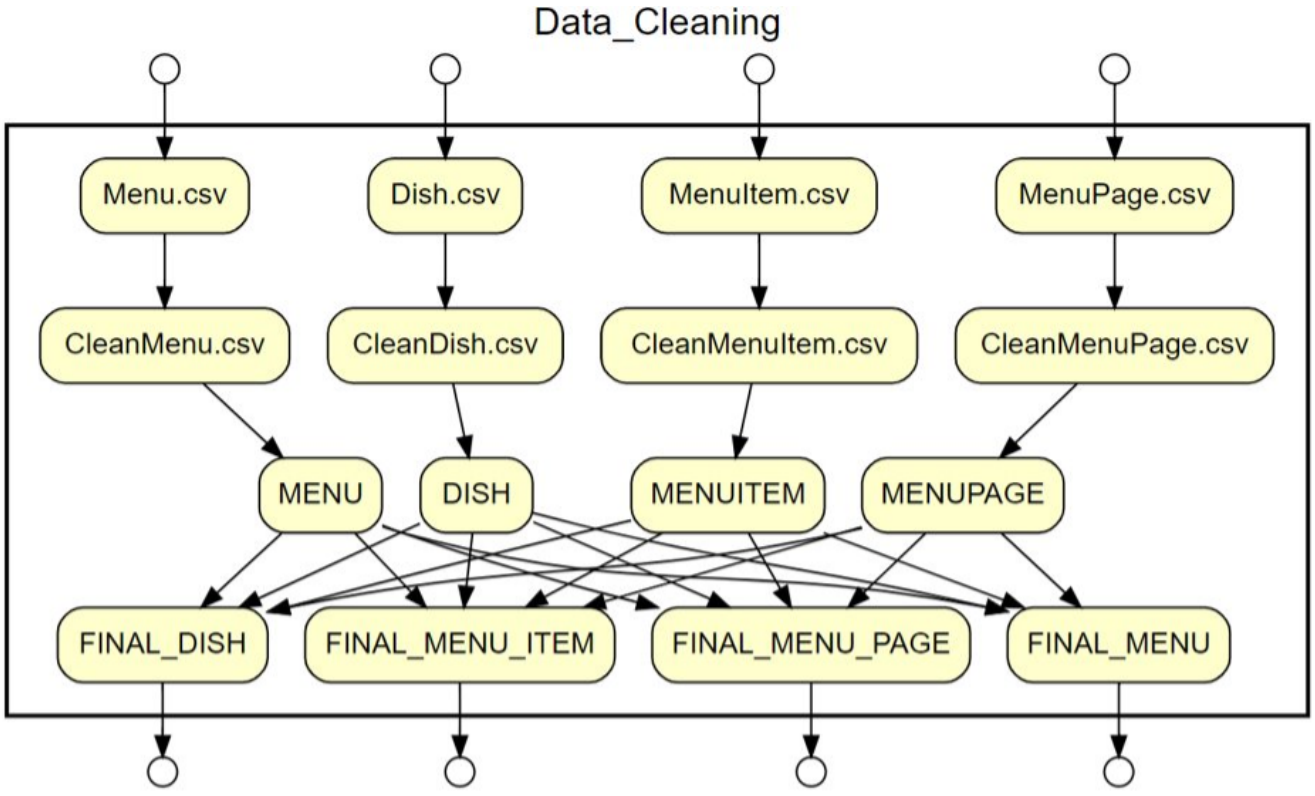

Fig. 27. Data workflow graph for all data cleaning operations