

**Theory and Practice of Data Cleaning**

**Food Application using NYPL Menu Dataset**

**By**

**Garima Garg & Pallavi Tyagi**

**Instructor: Bertram Ludäscher**

**12-18-2017**

## **OUTLINE TITLE**

I. Initial Assessment

II. Goal of the Project

III. Cleaning the datasets

A. Cleaning with Open Refine

B. Cleaning with Python

C. Comparing OpenRefine and Python

IV. Retrieving Data Using Python

V. Creating WorkFlow using YesWorkFlow

VI. Conclusion

### Initial Assessment

The dataset that we used for our project is New York Public Library's (NYPL) Menu dataset. The goal of our project is to clean and retrieve the data that can be further used for the purpose of creating food application (web or mobile). The dataset contains four files, namely, 'Dish.csv', 'Menu.csv', 'MenuItems.csv', and 'MenuPages.csv' which are described in the table below.

File name	Number of rows	Number of columns
Dish.csv	424509	9
Menu.csv	17545	20
MenuItems.csv	1333287	9
MenuPages.csv	66937	7

For the purpose of creating a food application not all the columns present in the dataset are important. Hence we remove some of the columns and then start its cleaning process using Open Refine. The resultant files (after eliminating the unnecessary columns) in the dataset are as described in the tables below.

#### 'Dish.csv'

Column Name	Description
'dish_id'	Id associated with each dish.
'dish_name'	Name of each dish.

'menus_appeared'	Total number of menus the dish appeared in.
'times_appeared'	Number of times the dish appeared on the menus in total.
'first_appeared'	The year a dish appeared for the first time.
'last_appeared'	The year a dish appeared last time.
'lowest_price'	The lowest price that a dish was sold for.
'highest_price'	The highest price that a dish was sold for.
'dish_name_code'	Code associated with each dish name.

**‘Menu.csv’**

Column Name	Description
'menu_id'	Id associated with each menu.
'restaurant_name'	Name of the restaurant which the menu is created for.
'call_number'	Phone number of the restaurant mentioned on the menu.
'date'	Date on which the menu is created.
'location'	Location of the restaurant.
'currency'	Currency for the location where the restaurant is located at.

'currency_symbol'	Symbol of the currency.
-------------------	-------------------------

**‘MenuItems.csv’**

Column Name	Description
'created_at'	Date at which the menu was created.
'high_price'	Highest price for an item on the menu.
'id'	Id associated with a menu item.
'menu_page_id'	Id associated with the menu page.
'price'	Price for an item present on the menu.
'updated_at'	The date at which a menu item was updated.
'xpos'	X position of an item on the menu.
'ypos'	Y position of an item on the menu.
'dish_id'	Id associated with each dish.

**‘MenuPages.csv’**

Column Name	Description
'menu_page_id'	Id associated with each page of a menu.

'menu_id'	Id associated with each menu.
'page_number'	Page number of a particular page on the menu.
'image_id'	Id associated with image on the menu.
'full_height'	Height of the menu page.
'full_width'	Width of the menu page.
'uuid'	Uuid is attributed to NYPL Image Repository

## GOAL OF THE PROJECT

Data Cleaning forms a very important part in data analytics. It is the key to correct and proper analysis on the dataset. Without a clean and well defined dataset, any analytics on the data would render completely useless and a utter waste of time and resources. But before we go into the process of cleaning and preprocessing our dataset, we should know what analytics we want to perform on the dataset. How are we going to analyse the dataset and what all components are most required by us in this process?

In this project we have four datasets as described above. Our aim is to clean the dataset in such a manner that we can produce an efficient food application. This food application would be ordered on the basis of location, restaurant name, menu and then dishes. The Dishes dataset hence proves to be very useful with the information of the dish name, the highest price of the dish, the lowest price of the dish, the year in which the dish was introduced in the menu and the

year it was last seen in the menu. It also has a dish id which helps us to merge it with the Menu Items dataset, which in turn gives us the menu id (which will help us to in turn merge it with the Menu dataset), highest price of a dish on the menu, price of an item on the menu, the date on which the it was last updated and so on. The Menu dataset gives us information about the restaurant name, its location, their call number and the type of currency it uses. The Menu page dataset is not so relevant here and basically gives us insights on how the menu in a particular restaurant looks.

So in whole we aim to :

- 1) Create a dataset for a successful food application.
- 2) This dataset should be able to have information about the location of a restaurant and the dishes associated with it.
- 3) It should be able to tell the price of a particular dish and should contain the most recent data.

### **Cleaning the datasets**

For cleaning we used OpenRefine and Python. The basic cleaning was performed on OpenRefine. The places where the dataset was incorrect needed additional attention and hence Python was used to replace such incorrect data.

#### **Cleaning with OpenRefine**

OpenRefine was used to preliminary clean the dataset. As the dataset, Dishes and MenuItems are the one of the largest dataset, they needed the most cleaning. Menu dataset and Menu Pages dataset were mostly clean and didn't need much preliminary cleaning.

The Dishes dataset was huge, having 424509 rows. This huge dataset when loaded into OpenRefine rendered it useless and hence preprocessing was not possible. This called for breaking the original dataset into small parts and then cleaning it on OpenRefine. This dataset required a major cleaning on the dishes name, as most of the restaurants had a different name for the same dish for example “3 boiled eggs” and “boiled eggs (3)”. This called for the use of OpenRefine “Cluster and Edit” option which would cluster these similar words together and provide us with a window with checkboxes to choose amongst the options that are correct. This clustering was performed on, first the individual broken datasets and then finally on the entire dataset once all the datasets were merged. Moreover many columns were not required in this dataset for the food application. These columns which were “menus\_appeared” and “times\_appeared” were removed from the dataset. The remaining columns were basically numerical values that were assigned as numerical in the dataset.

The Menu Items dataset was huge too and hence the same technique (as used with Dishes dataset) was applied here. Here the basic aim was to change the format of ‘created\_at’ and ‘updated\_at’ columns in the dataset to a time format. This was a huge task as OpenRefine would not recognize them as time columns and needed us to use the GREL language to make the changes. The GREL language in OpenRefine helps us to apply changes over multiple columns. Initially the two columns are in string format as shown : 2011-03-28 15:00:44 UTC. This value was first separated using the split function. (Figure 3.1)



### Split column created\_at into several columns

**How to Split Column**

☒ by separator

Separator  ☐ regular expression

Split into  columns at most (leave blank for no limit)

☐ by field lengths

List of integers separated by commas, e.g., 5, 7, 15

**After Splitting**

☒ Guess cell type

☒ Remove this column

*Figure 3.1 Splitting a column in OpenRefine*

Space is given as the separator and then we get three columns one for 2011-03-28 and other for 15:00:44 and another for UTC. Removing the UTC column, we combined the remaining two columns using the GREL command as shown below in Figure 3.2.

### Custom text transform on column updated\_at 1

Expression Language General Refine Expression Language (GREL)

value+" "+cells["updated\_at 2"].value

No syntax error.

[Preview](#)
[History](#)
[Starred](#)
[Help](#)

row	value	value+" "+cells["updated_at 2" ...
1.	2012-09-24	2012-09-24 20:46:42
2.	2012-09-24	2012-09-24 20:46:53
3.	2012-09-24	2012-09-24 20:47:05
4.	2012-09-24	2012-09-24 20:47:14
5.	2012-09-24	2012-09-24 20:47:26
6.	2012-09-24	2012-09-24 20:47:35
7.	2012-09-24	2012-09-24 20:47:50

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

*Figure 3.2 Merging two columns in OpenRefine*

This would then give us the column in the desired format, where we could apply the todate feature in OpenRefine as shown below in Figure 3.3. The new data is in the format

2012-04-01T15:34:03Z.

« first < previous 1 - 10

id	dish_id	created_at 1	updated_at 1	xpos	ypos
0.1	375011	2012-09-24T20:46:42Z	Facet	0.56	0.618402
	414794	2012-09-24T20:46:53Z	Text filter	0.56	0.650511
	378057	2012-09-24T20:47:05Z	Edit cells	Transform... Common transforms Fill down Blank down Split multi-valued cells... Join multi-valued cells... Cluster and edit...	
	386146		Trim leading and trailing whitespace		
	907		Collapse consecutive whitespace		
	380037		Unescape HTML entities		
	414795		To titlecase		
	414796		To uppercase		
	414797		To lowercase		
	414798		To number		
			To date		
			To text		
			Blank out cells		

*Figure 3.3 Transformations in OpenRefine*

The Menu and the Menu Pages dataset were mostly clean and required basic cleaning that was implemented in the Dishes and the Menu Items dataset as well. These basic cleaning techniques are summarized below :

- 1) **Trim leading and Trailing whitespaces** (as can be seen in Figure 3.3) : This helps to remove the leading and trailing whitespaces in the dataset.

- 2) **Collapse Consecutive whitespaces** (Figure 3.3) : Helps to remove whitespaces in between words that are more than one. This removes the unnecessary spaces between the words.
- 3) **To lowercase** (Figure 3.3) : Wherever string values were present, they were converted into lowercase characters so that all of them have the same manner of display.
- 4) **To number** (Figure 3.3) : This converts all the values in the columns to numerical value if they can be converted. This was used heavily in the preprocessing of the dataset.
- 5) **Blank down** (Figure 3.3) : This was used to remove all blank cells from the dataset.
- 6) **Cluster and Edit** : This was used to cluster the dataset and assign a common name to similar rows.

After these basic data cleaning operations, we moved to Python to correct the incorrect values in the dataset.

### Cleaning with Python

The Menu Items dataset had a lot of ambiguity like the ‘high price’ was empty at most positions as shown in the Figure 3.4.

266657 rows										
Show as: rows records		Show: 5 10 25 50 rows				« first ‹ previous 1 -				
All		id	menu_page_id	price	high_price	dish_id	created_at	updated_at	xpos	ypos
☆	1.	1113750	67675	0.05	0.1	375011	2012-09-24T20:46:42Z	2012-09-24T20:46:42Z	0.56	0.618402
☆	2.	1113751	67675	0.05		414794	2012-09-24T20:46:53Z	2012-09-24T20:46:53Z	0.56	0.650511
☆	3.	1113752	67675	0.1		378057	2012-09-24T20:47:05Z	2012-09-24T20:47:05Z	0.56	0.695702
☆	4.	1113753	67675	0.1		386146	2012-09-24T20:47:14Z	2012-09-24T20:47:14Z	0.666667	0.694513
☆	5.	1113754	67675	0.2		907	2012-09-24T20:47:26Z	2012-09-24T20:47:26Z	0.558667	0.7123510000000001
☆	6.	1113755	67675	0.2		380037	2012-09-24T20:47:35Z	2012-09-24T20:47:35Z	0.557333	0.7290000000000001
☆	7.	1113756	67675	0.1		414795	2012-09-24T20:47:50Z	2012-09-24T20:47:50Z	0.558667	0.663593
☆	8.	1113757	67675	0.1		414796	2012-09-24T20:48:15Z	2012-09-24T20:48:15Z	0.605333	0.680242
☆	9.	1113758	67675	0.1		414797	2012-09-24T20:48:33Z	2012-09-24T20:48:33Z	0.688	0.680242
☆	10.	1113759	67675	0.1		414798	2012-09-24T20:48:48Z	2012-09-24T20:48:48Z	0.898667	0.679053

Figure 3.4 high price column in Menu Items dataset

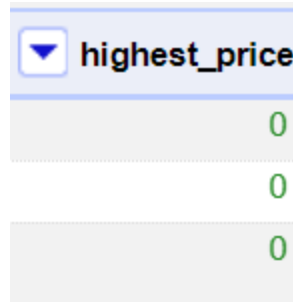
Also the Dishes dataset, which had columns 'last\_appeared' and 'first\_appeared' had issues cause in most places the year in 'first\_appeared' was greater than the year in 'last\_appeared'.

Also the 'last\_appeared' and 'first\_appeared' ad values like 0,1 an 2928 which are totally incorrect years. This can be shown in Figure 3.5.

▼ first_appeared	▼ last_appeared
1897	1927
1895	1960
1893	1917
1900	1971
1881	1981
1854	2928
1897	1961
1899	1962
1900	1900
1893	1937
1900	1900
1858	1987
1899	1900
1	2928
1897	1918
1880	1987
1856	2928
1865	1901
1852	1987
1851	1948
1898	1901
1880	1950
1900	1900
1900	1901

*Figure 3.5 last appeared and first appeared in Dish dataset*

Also at places in the ‘Dishes’ dataset the ‘highest\_price’ and values like 0 which were totally not valid and incorrect as can be seen in Figure 3.6.



highest_price
0
0
0

*Figure 3.6 highest price in Dish dataset*

All these ambiguities need to be addressed for a cleaned dataset. For this we make certain assumptions and hence correct the dataset likewise.

First looking at the “Dishes” dataset we need to make clusters on dishes name that would help us to group similar dishes together. As clustering does not work on strings in Python each dish was assigned a unique code present in dish name code as can be seen in Figure 3.7.

Hence now clustering can be performed on the dataset. In this project we divide the entire dataset into 5 clusters as can be seen in Figure 3.8. These clusters have the names that are most likely connected together or have similar attributes.

The aim behind making these clusters is that we assume that the contents in each cluster is closely connected to each other.

```
In [5]: obj_df["dish_name_code"] = obj_df["dish_name"].cat.codes
obj_df
```

Out[5]:

	dish_name	dish_name_code
0	consomme printaniere royal	85320
1	chicken gumbo	67776
2	tomato aux croutons	302885
3	onion au gratin	213947
4	st. emilion	286792
5	radishes	242512
6	chicken soup with rice	69540
7	cup clam broth	96452
8	cream of new asparagus croutons	92101
9	clear green turtle	75178
10	striped bass saute meuniere	291108
11	anchovies	11667
12	fresh lobsters in every style	125367
13	celery	57639
14	pimolas	227059
15	caviar	57162

*Figure 3.7 Dish name codes for Dish dataset*

In other words, each cluster has data values that are similar to each other. Hence we remove the ambiguities in dataset by making the following assumptions :

- 1) The highest value is assigned as the average value of all the other highest value average (excluding 0).
- 2) The values like 0,1,2928 are replaced by the highest count of years in the 'last\_appeared' and 'first\_appeared' respectively.

This helps to give us a dataset that is cleaned and with correct values and hence we can assume that the Dish dataset is very much cleaned for analysis as can be verified in Figure 3.9.

```
dish["Clusters"] = labels
```

```
print(dish.head())
```

	dish_id	dish_name	menus_appeared	times_appeared	\
0	1	consomme printaniere royal	8	8	
1	2	chicken gumbo	111	117	
2	3	tomato aux croutons	13	13	
3	4	onion au gratin	41	41	
4	5	st. emilion	66	68	

	first_appeared	last_appeared	lowest_price	highest_price	dish_name_code	\
0	1897	1927	0.20	0.4	85320	
1	1895	1960	0.10	0.8	67776	
2	1893	1917	0.25	0.4	302885	
3	1900	1971	0.25	1.0	213947	
4	1881	1981	0.00	18.0	286792	

	Clusters
0	1
1	1
2	4
3	4
4	4

*Figure 3.8 Clusters in Dish dataset*

```
In [45]: final_dish['first_appeared'] = np.where((final_dish['first_appeared'] > final_dish['last_appeared']),final_dish['last_appeared'],
```

```
In [46]: icd_data = np.where((final_dish['lowest_price'] > final_dish['highest_price']), 'Violation', 'No Violation')
```

```
for i in icd_data :
```

```
    if(i=='Violation'):
```

```
        print (True)
```

*Figure 3.9 Checking for Violations*

For Menu Items dataset we merge the cleaned Dish dataset with the Menu Items dataset on dish id as shown in Figure 3.10.

Once merged we assign the highest value that corresponds to the highest value of a dish to the high price column in Menu Items dataset wherever the high price column is null. As both these

columns represent the same characteristics of the highest price of a dish this seemed like a valid assumption. The code for this can be seen in Figure 3.11.

```
test = pd.merge(dish,menuitem, on= "dish_id",how="outer")
test.head()
```

Unnamed: 0	dish_id	dish_name	menus_appeared	times_appeared	first_appeared	last_appeared	lowest_price	highest_price	dish_name_code	Clusters	
0	265857.0	339839.0	"b. p. l."	1.0	1.0	1937.0	1937.0	0.0	2.551514	211.0	0.0
1	266465.0	340671.0	abricot	1.0	1.0	1937.0	1937.0	0.0	2.551514	8281.0	0.0

*Figure 3.10 Merging two datasets*

```
In [10]: test["high_price"] = np.where(test["high_price"].isnull(),test['highest_price'],test['high_price'])
test["high_price"] = np.where(test["high_price"] == 0,test['highest_price'],test['high_price'])
```

*Figure 3.11 Correcting high price in Menu Dataset*

This hence helps to remove any ambiguities in the Menu Items dataset and makes it clean for analysis.

## Comparing OpenRefine and Python

OpenRefine is a data cleaning tool and hence helps to achieve the basic data cleaning aspects.

GREL is powerful and can be used to make heavy data changes in OpenRefine. It is also very simple and easy to understand and can be learnt ‘on the go’.

On the other hand, Python is a powerful coding language. It comes with numerous packages and can incorporate the aspects of data cleaning. But it is not as easy to use as OpenRefine because not everyone can code. Although Python is powerful for not only data cleaning but also data manipulation, it lacks the ease with which OpenRefine can perform data cleaning especially for non coders.



For coders, Python (especially the library pandas) becomes a saviour and provides a great platform for both data cleaning and manipulation.

### Retrieving Data Using Python

Once the data is cleaned we retrieved the desired rows using python. The cleaned data files are obtained in 'csv' format. We then load the csv files into pandas dataframe using the pandas library of python. The python code for the above description is as given below:

```
In [1]: import pandas as pd
```

```
In [2]: # Loading the cleaned files into dataframe
Final_Menu_Item = pd.read_csv('Final_Menu_Item-2.csv')
Final_Dish = pd.read_csv('Final_Dish-1.csv')
Final_Menu = pd.read_csv('Final_Menu-1.csv')
Final_Menu_Page = pd.read_csv('Final_Menu_Page-1.csv')
```

```
In [18]: # Removing the Unnamed column
Final_Menu_Item = Final_Menu_Item.loc[:, ~Final_Menu_Item.columns.str.contains('^Unnamed')]
Final_Dish = Final_Dish.loc[:, ~Final_Dish.columns.str.contains('^Unnamed')]
Final_Menu = Final_Menu.loc[:, ~Final_Menu.columns.str.contains('^Unnamed')]
Final_Menu_Page = Final_Menu_Page.loc[:, ~Final_Menu_Page.columns.str.contains('^Unnamed')]
#Final_Menu_Item.columns
#Final_Dish.columns
#Final_Menu.columns
Final_Menu_Page.columns
```

The tables created from the csv files look as given below:

	menu_page_id	menu_id	page_number	image_id	full_height	full_width	uuid
0	119	12460	1.0	1603595	7230	5428	510D47E4-2955-A3D9-E040-E00A18064A99
1	120	12460	2.0	1603596	5428	7230	510D47E4-2956-A3D9-E040-E00A18064A99
2	121	12460	3.0	1603597	7230	5428	510D47E4-2957-A3D9-E040-E00A18064A99
3	122	12460	4.0	1603598	7230	5428	510D47E4-2958-A3D9-E040-E00A18064A99
4	123	12461	1.0	1603591	7230	5428	510D47E4-2959-A3D9-E040-E00A18064A99
5	124	12461	2.0	1603592	7230	5428	510D47E4-295A-A3D9-E040-E00A18064A99
6	125	12461	3.0	1603593	7230	5428	510D47E4-295B-A3D9-E040-E00A18064A99
7	126	12461	4.0	1603594	7230	5428	510D47E4-295C-A3D9-E040-E00A18064A99
8	129	12463	2.0	4000009170	3074	2046	510D47DB-491E-A3D9-E040-E00A18064A99
9	130	12463	1.0	466928	3049	2004	510D47DB-491F-A3D9-E040-E00A18064A99
10	131	12464	2.0	4000009171	3690	2888	510D47DB-4920-A3D9-E040-E00A18064A99

After the csv files are converted into data-frame we start retrieving the data for the scenario where the food application is created in 2017 and the application is designed to serve users based in the United States of America. To retrieve data for such application, the rows should be retrieved for the time between the year 2000 and 2017 so that relevant data for the food application to be created in 2017 is retrieved. To achieve the above scenario we try to run four SQL queries which are described as follows.

### Query 1

First query is designed to fetch all the rows from the dish table that have the first appeared date in the year '2000' and last appeared date less than '2017'. The code and the outcome of the query is as follows:

```
Final_Dish.loc[(Final_Dish['first_appeared'] == 2000) & (Final_Dish['last_appeared'] < 2017)].head(5)
```

	dish_id	dish_name	menus_appeared	times_appeared	first_appeared	last_appeared	lowest_price	highest_price	dish_name_code
18195	393888	claypot roasted salmon langostino tomato sauce...	1	0	2000	2000	20.95	20.95	75135
18196	393889	diver sea scallops sauteed vegetables and lent...	1	0	2000	2000	23.75	23.75	101130
18197	393895	grilled veal chop sauteed mushrooms and rosemary	1	0	2000	2000	27.75	27.75	148368
21922	401726	bruschetta toscana house specialty grilled cro...	1	0	2000	2000	7.95	7.95	48286
21923	401737	filet mignon portobellini mushrooms new potato...	1	0	2000	2000	24.95	24.95	114764

## Query 2

Once we get the dish data that is between the time period of 2000 and 2017, we now join the Dish table and Menu\_item table using inner join to fetch the rows that are between the period of 2000 and 2017 and where price of the dish is equivalent to mode price of the dishes in the table. Mode of the dishes price is found to fetch the rows that have the dishes with the most common price of the dish. The output of the above mentioned query is as given below.

```
Final_Dish_Menu_Item = pd.merge(Final_Dish, Final_Menu_Item, on='dish_id', how='inner')
```

```
Final_Dish_Menu_Item['created_at'] = pd.to_datetime(Final_Dish_Menu_Item['created_at'])
```

```
Final_Dish_Menu_Item['created_year'] = Final_Dish_Menu_Item['created_at'].apply(lambda x: x.year)
Final_Dish_Menu_Item
```

```
Final_Dish_Menu_Item.loc[(Final_Dish_Menu_Item['created_year'] >= 2000)&(Final_Dish_Menu_Item['created_year'] < 2017)]
```

```
x = Final_Dish_Menu_Item['price'].mode()
z = (x.values)
Final_Dish_Menu_Item_3=Final_Dish_Menu_Item.loc[(Final_Dish_Menu_Item['price']==z[0])
&(Final_Dish_Menu_Item['created_year'] >= 2000)
& (Final_Dish_Menu_Item['created_year'] < 2017)]
```

The data retrieved after running the above query is as follows:

	dish_id	dish_name	menus_appeared	times_appeared	first_appeared	last_appeared	lowest_price	highest_price	dish_name_code
480126	32	wheat vitos	3	3	1900	1900	0.0	5.146847	316045
480127	32	wheat vitos	3	3	1900	1900	0.0	5.146847	316045
480128	32	wheat vitos	3	3	1900	1900	0.0	5.146847	316045
482898	58	roast easter lamb mint sauce	4	4	1899	1900	0.0	5.146847	249161
482899	58	roast easter lamb mint sauce	4	4	1899	1900	0.0	5.146847	249161

### Query 3

Now lastly we want to retrieve the rows that have the currency as Dollars. The currency information in the dataset is given in the Menu data table. Hence we fetch the rows which have currency equivalent to Dollar only. This is achieved by running the code given in the following figure and the final data retrieved is also shown below it.

```
Final_Menu.loc[(Final_Menu['currency'] == 'Dollars')]
```

	menu_id	restaurant_name	call_number	date	location	currency	currency_symbol	page_count
0	20966	The Modern	+1 200 8-000	2008-03-03T00:00:00Z	The Modern	Dollars	\$	6
3	30875	Archons of Colophone	+1 193 8-0038	1938-03-14T00:00:00Z	Restaurant name and/or location not given	Dollars	\$	1
4	31952	Chalfonte	+1 191 9-0001	1919-01-01T00:00:00Z	Chalfonte	Dollars	\$	3
5	31953	The Aldine Club	+1 191 9-0002	1919-01-20T00:00:00Z	The Aldine Club	Dollars	\$	2
6	31954	Hotel Majestic	+1 191 9-0003	1919-01-16T00:00:00Z	Hotel Majestic	Dollars	\$	8
7	31955	Hotel Pennsylvania	+1 191 9-0004	1919-01-25T00:00:00Z	Hotel Pennsylvania	Dollars	\$	3

### Creating WorkFlow using YesWorkFlow

Workflows have been used since time immemorial and helps to give a clear cut flow of the entire process that was used in the dataset. YesWorkFlow is a tool which meets those needs and nothing much more is needed than a little coding in the comments section of the code. For this dataset we have divided the YesWorkFlow for each task.

- 1) Data cleaning YesWorkFlow for Dish dataset : This represents the entire cleaning of the dish dataset in Python shown in Figure 5.1.
- 2) Data cleaning YesWorkFlow for Menu Items dataset: This represents the data cleaning steps used in Python shown in Figure 5.2.
- 3) Data cleaning YesWorkFlow for Menu dataset: This shows the data cleaning tasks applied on Menu dataset in Python in Figure 5.3.
- 4) Data cleaning YesWorkFlow for Menu Pages dataset : This shows the process applied for the Menu Pages dataset.

These two datasets were mostly cleaned and did not require much processing.

The last YesWorkflow is the Data Retrieving using Python pathway that has been followed throughout the project in Figure 5.5.

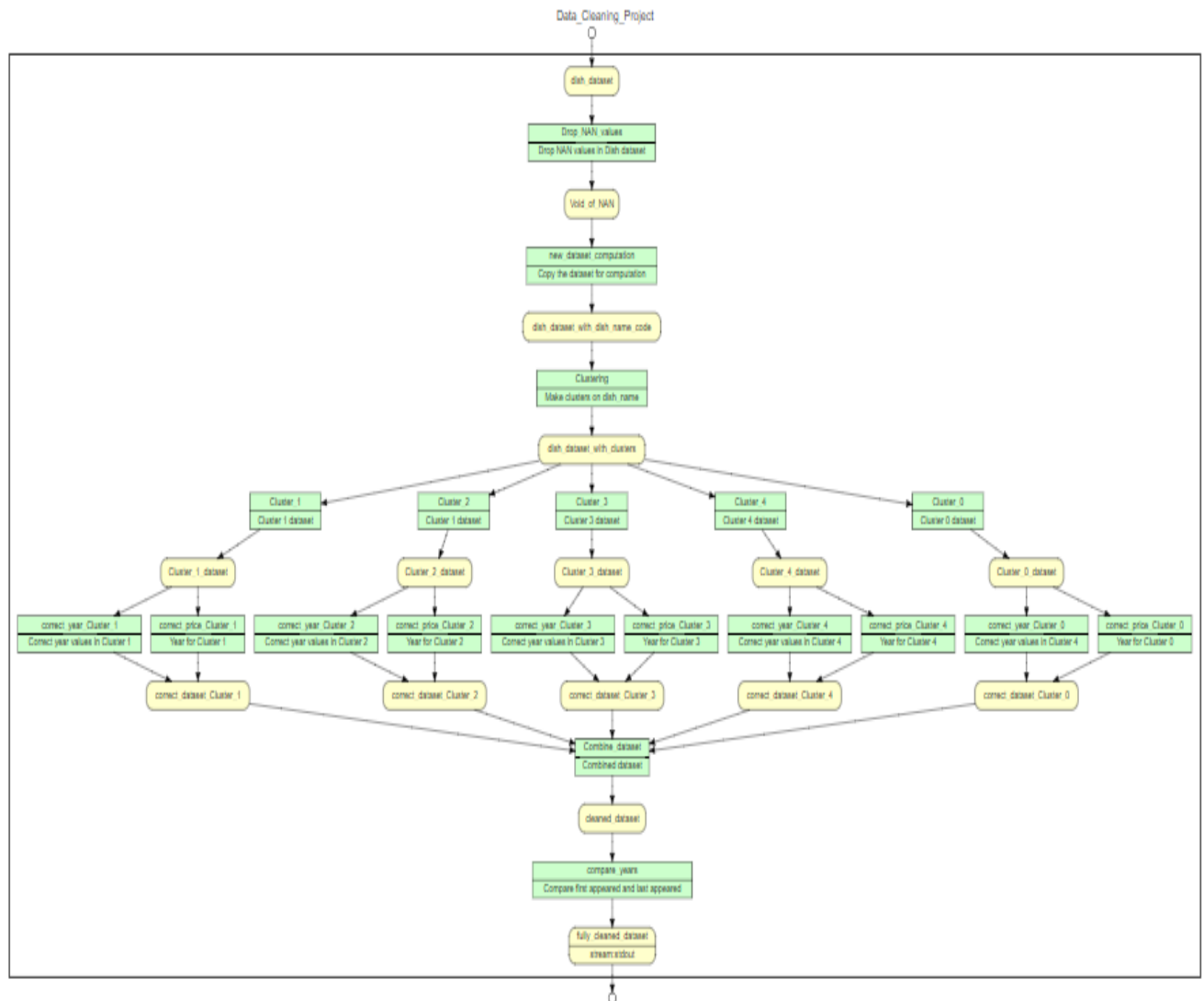


Figure 5.1 Dish dataset YesWorkflow

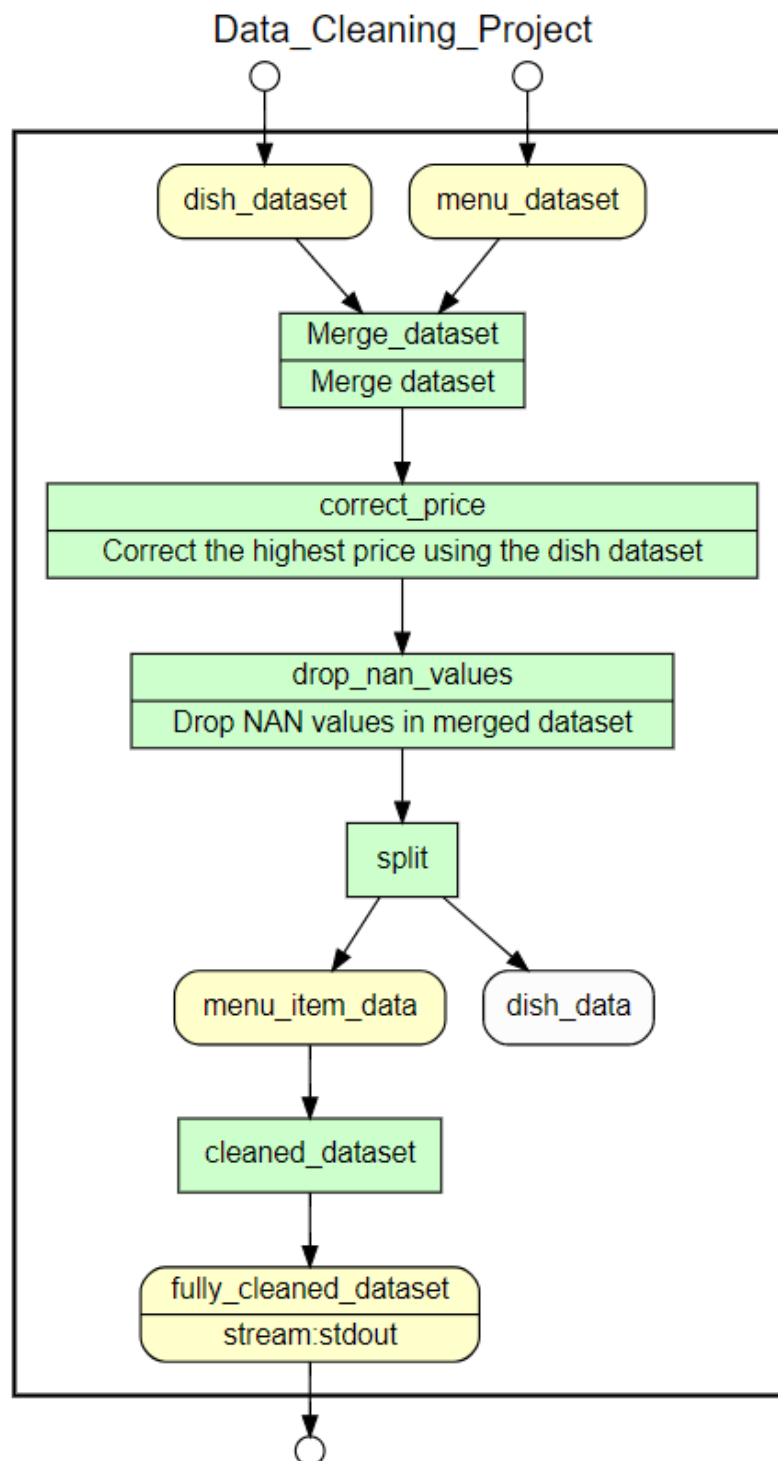


Figure 5.2 Menu Items dataset YesWorkflow

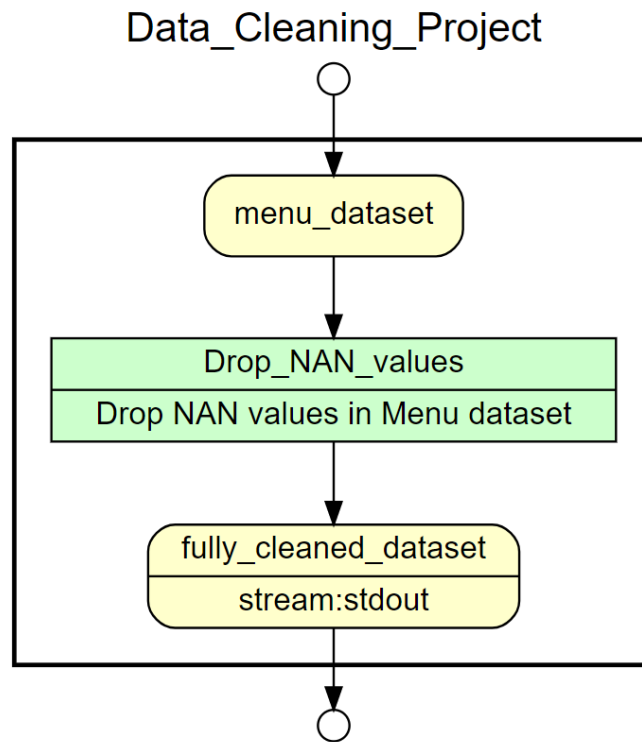


Figure 5.3 Menu dataset YesWorkflow

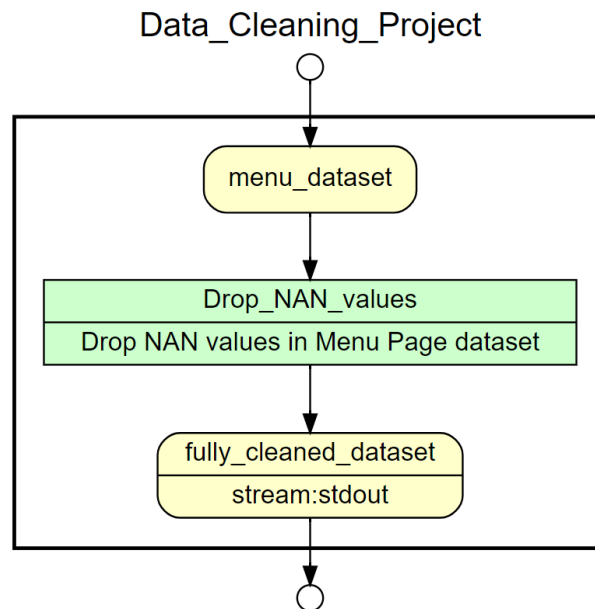
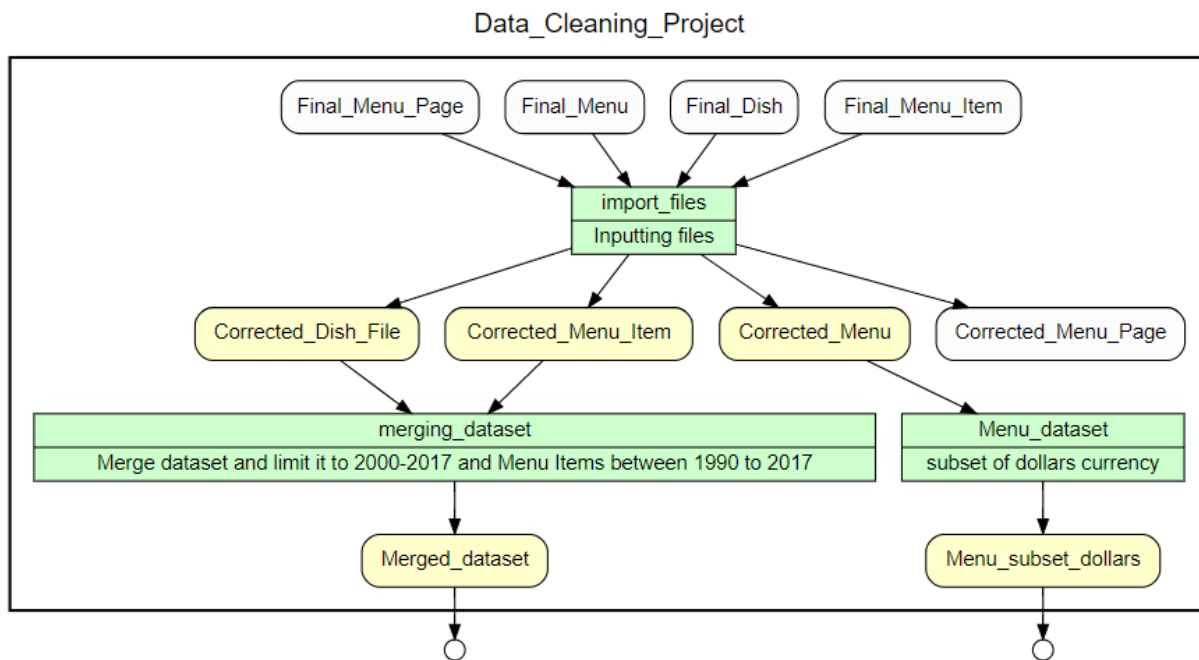


Figure 5.4 Menu Page dataset YesWorkflow





*Figure 5.5 Retrieve relevant Data From Python YesWorkflow*

### Conclusion

This project was started with an intention to clean and retrieve data for creating a food web-application or a food mobile application. The dataset was first loaded in excel and studied and then the columns that were not relevant for the purpose of creating a food application were eliminated from the dataset. After the removal of unnecessary columns the datasets were loaded into OpenRefine and cleaned and clustered on the basis of the requirement as explained in section III. After the cleaned csv files of data are retrieved the data is loaded into pandas dataframe to create tables and fetch the desired rows. This data can be further used for creating a food application. Once we get the final data, we then create the YesWorkflow diagrams for the all the processes that were involved in the entire data preparation processes.

## References

- [1] <http://menus.nypl.org/menus/>
- [2] <https://github.com/NYPL/menus-api>
- [3] <http://nypl.github.io/menus-api/>
- [4] <http://absflow.westus.cloudapp.azure.com/>
- [5] <https://pandas.pydata.org/pandas-docs/stable/>
- [6] <http://www.datacarpentry.org/OpenRefine-ecology-lesson/>