

--	--	--

# CS513: Theory & Practice of Data Cleaning

## **Final Project Report (Summer 2021)**

Professor: Bertram Ludäscher

Team: Marina Polupanova (marinap2@illinois.edu), Wei-Lun Tsai (wltsai2@illinois.edu)

University of Illinois, Urbana-Champaign

---

		1
--	--	---

--	--	--

# CS513: Theory & Practice of Data Cleaning

## **Final Project Phase I (Summer 2021)**

Professor: Bertram Ludäscher

Team: Marina Polupanova (marinap2@illinois.edu), Wei-Lun Tsai (wltsai2@illinois.edu)

University of Illinois, Urbana-Champaign

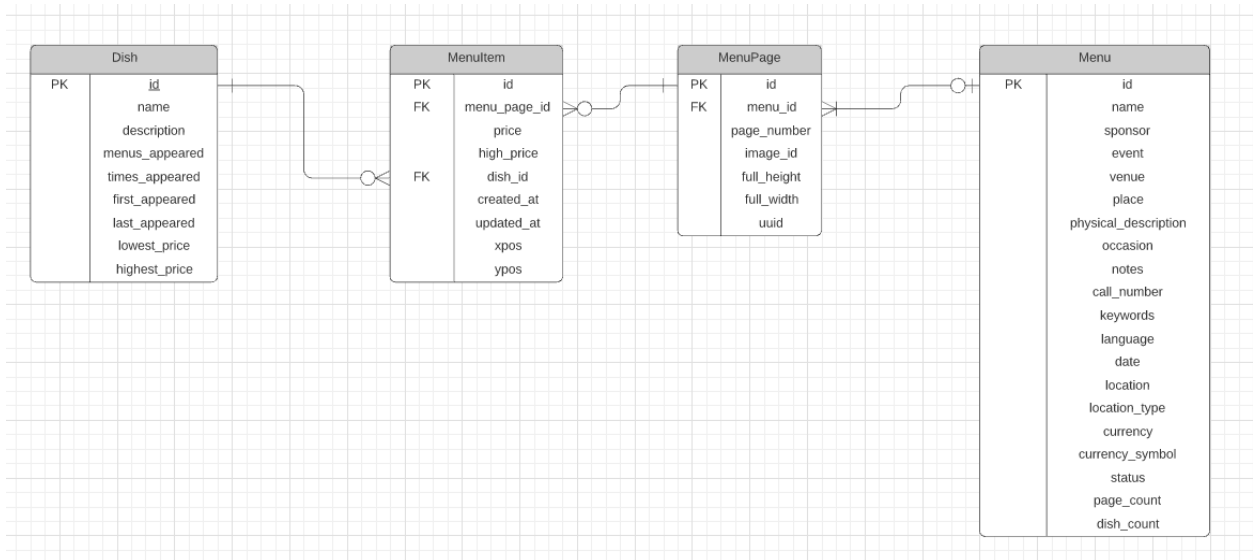
		2
--	--	---

## 1. Identify a Dataset

For the Data Cleaning Final Project, our Dataset (D) of interest will be the NYPL Menus data, available publicly via [their website](http://menus.nypl.org/about)<sup>1</sup>. Below is a more detailed description of the dataset.

## 2. Describe the Dataset

The dataset can be described by following ER diagram (created on non-cleaned dataset), and all the feature columns of all tables in the dataset have TEXT format, if checked with SQLite, or “string” if checked with OpenRefine.



The dataset is collected by The New York Public Library and contain in the currently analyzed database upload 17547 menu from around 45000 menus, dated from 1840 year till present time, collected by NYPL since 1900. The significant part of the menus has New York as origin, but the dataset contains menus also from other states of US, and even from all over the world. Information is collected in 4 tables: Dish table, which represent names, descriptions, number of menus where it appeared, time of first and last appearance of the meal, and its highest and cheapest price; MenuItem table, where we can see on which page this MenuItem appeared, its price, and if the price is high, dish which is used in this MenuItem, and the time of creation and updating this record, and xpos and ypos; MenuPage table, containing the id of the Menu, where it appeared and page number with related image\_id and sizes of the record (full\_height, full\_width), and identifier of this page; Menu table, where it is stored name, sponsor, event, venue type (commercial, government or other), place (information in which restaurant, or/and state of US or another country the event happened), physical description (if it is a booklet, broadside, or card, or other and its size), occasion (holiday or anniversary to which event was dedicated), notes (mostly containing physical appearance of menu), call number, keywords and language of menu, date, location (name of the organization, on which territory is the event) and location type, currency, currency symbol, status (if the record is under review or completed), and page and dish count.

<sup>1</sup> <http://menus.nypl.org/about>

--	--	--

### 3. Develop a Target Use Case

For our **target use case (U<sub>1</sub>)** we will aim to answer the following: “In the state of NY, what is the average price of each dish, and how many distinct locations offer each dish?” After joining the four data tables (Menu, MenuPage, MenuItem, Dish) provided by the NYPL, we will be able to build a dataset with relations between locations, dishes, and prices. Then, we will query this dataset to determine the average price of each available dish and number of distinct locations that offer that dish.

A **use case (U<sub>0</sub>) that requires zero data cleaning** can be calculated: “What is the number of menus collected within this NYPL dataset?” This can be accomplished by simply querying the count(id) from table Menu. This use case requires no data cleaning, as the column “id” contains distinct values, and every row in the Menu table has some meaningful record.

A **use case (U<sub>2</sub>) for which this NYPL dataset is never (good) enough** can be determined: “From which menu(s) does each dish in the Dish table originate?” Since we discovered in our initial investigation that there are entries in the Dish table (i.e. dishes) that are not present in at least one MenuItem entry, we cannot determine from which menu(s) every single dish originates using this exact dataset (i.e. there will always be some dishes for which we don’t have their corresponding listing in a menu).

### 4. List Obvious Data Quality Problems

There are a number of obvious data quality problems in the dataset. Some of the Menu.location and Dish.name values contain unnecessary special characters (e.g. ‘;’, ‘(?)’, ‘-’, etc.) which aren’t meaningful to the name and affect data analyses that require aggregation or sorting. There are also several values that are slight variants of each other (e.g. “The Waldorf Astoria” vs. “The Waldorf Astoria Hotel” or “Apricot” vs. “apricot”) and thus need to be clustered and cleaned since U<sub>1</sub> requires a deduplicated dataset for effective aggregation. Fields that indicate locations that are not available (e.g. variants of “Restaurant name and/or location not given”) need to be standardized so that we may filter them out when forming the queries for U<sub>1</sub>. We summarize our initial observations of obvious data quality problems below:

1. To start with, in some fields, there are “;” at the end of the value record, which brings confusion when reading the data not in specialized data cleaning SW like OpenRefine. This “;” does not bring any valuable information to the record itself. In addition, there are many excessive spaces both in the middle, and in the beginning/end of some records. Please see the example from Dish table:

```
sqlite> select * from Dish where name like "%;" limit 2;
62534,"CAVENDISH, Light, 4s. per lb.;",",",1,1,1900,1900,0.0,0.0
79948,"Mixed Salad;",",",1,1,1900,1900,0.0,0.0
```

Here is the example of extensive spaces from Dish table:

```
sqlite> select * from Dish where name like " Compote Assorted";
362107," Compote Assorted",",",1,1,1933,1933,0.0,0.0
```

2. In the table Dish, some of the entries of the meals are entered in “”, and some are not:

```
sqlite> select * from Dish limit 10;
```

		4
--	--	---

--	--	--

- 1, "Consomme printaniere royal", "", 8, 8, 1897, 1927, 0.2, 0.4
- 2, "Chicken gumbo", "", 111, 117, 1895, 1960, 0.1, 0.8
- 3, "Tomato aux croutons", "", 14, 14, 1893, 1917, 0.25, 0.4
- 4, "Onion au gratin", "", 41, 41, 1900, 1971, 0.25, 1.0
- 5, "St. Emilion", "", 66, 68, 1881, 1981, 0.0, 18.0
- 7, Radishes, "", 3262, 3346, 1854, 2928, 0.0, 25.0
- 8, "Chicken soup with rice", "", 48, 49, 1897, 1961, 0.1, 0.6
- 9, "Clam broth (cup)", "", 14, 16, 1899, 1962, 0.15, 0.4
- 10, "Cream of new asparagus, croutons", "", 2, 2, 1900, 1900, 0.0, 0.0
- 11, "Clear green turtle", "", 156, 156, 1893, 1937, 0.25, 60.0

3. There is an issue with the "location" field in Menu table, as same place is named differently there. A good example is "Waldorf Astoria", where it appeared 15 different variants of same location name:

```
sqlite> select location from Menu group by location having location
like "%Waldorf%" limit 15;
```

"Annual Dinner Of The New York University Alumni Association, Waldorf Astoria"

"Men's Bar Of The Waldorf Astoria"

"The Starlight Roof Of The Waldorf Astoria"

"The Waldorf"

"The Waldorf Astoria"

"The Waldorf Astoria Ballroom"

"The Waldorf Astoria Hotel"

"The Waldorf Astoria; Grand Ballroom"

"Waldorf Astoria"

"Waldorf Astoria"

"Waldorf Astoria Hotel"

"Waldorf Astoria Hotel (New York, N.Y.)"

"Waldorf Astoria Hotel, Astor Gallery"

"Waldorf Astoria Hotel, Stalight Roof"

"Waldorf Astoria?"

		5
--	--	---

--	--	--

4. Real locations are missing in the table Menu in the field “location” in some number of fields, and instead of it there are values like “Restaurant name and/or location not given”:

```
sqlite> select count(id) from Menu where location like "%Restaurant
name and/or location not given%";
```

166

5. All the records in all the tables of the dataset are in TEXT, so to operate with numeric values we will need to translate it to numbers. For example, please see the types of the Dish table:

```
sqlite> .schema
```

```
CREATE TABLE Dish(
```

```
  "id" TEXT,
```

```
  "name" TEXT,
```

```
  "description" TEXT,
```

```
  "menus_appeared" TEXT,
```

```
  "times_appeared" TEXT,
```

```
  "first_appeared" TEXT,
```

```
  "last_appeared" TEXT,
```

```
  "lowest_price" TEXT,
```

```
  "highest_price" TEXT
```

```
);
```

6. Many values in the column “price” of MenuItem table are absent, so it will be hard to judge on the prices of the dishes for the missing rows:

```
sqlite> select count(id) from MenuItem where price="";
```

446259

Same situation is with table Dish, where there are many missing values for the highest and lowest prices of meals:

```
sqlite> select count(id) from Dish where lowest_price="";
```

29101

```
sqlite> select count(id) from Dish where highest_price="";
```

29101

7. Some of the lowest and highest prices of dishes in the Dish table are equal to zero:

```
sqlite> select count(id) from Dish where highest_price="0.0";
```

		6
--	--	---

--	--	--

222698

```
sqlite> select count(id) from Dish where lowest_price="0.0";
```

227250

There is same kind of issue in MenuItem table for the price:

```
sqlite> select count(id) from MenuItem where price="0.0";
```

362

8. The field "name" of Dish table contains some irrelevant records like "&", which cannot be treated as a name of the dish so we can probably delete it:

```
sqlite> select * from Dish where name="&;
```

637,&,"",4,4,1901,1901,0.0,0.0

Also, there are some duplicated dishes names, which parts were recorded in different register:

```
sqlite> select name from Dish where name like "Apricot";
```

*Apricot*

*apricot*

9. Finally, according to the ER diagram, not for every dish we can find existing MenuItem, and so some of the dishes may never appear in any menu. To ensure we will have the dataset fit for our use case, we will need to ensure that in the final combined table we will have only the dishes, present in some menus.

## 5. Devise an Initial Plan

Below is the initial plan and tentative task assignments we will follow to clean the dataset to achieve our target use case ( $U_1$ ).

**S1: description of dataset D and matching use case  $U_1$ .** The dataset D is from the NYPL Menus data, and is comprised of four tables: Menu, MenuPage, MenuItem, and Dish. Each table includes primary and foreign keys that enable joining to at least one other table, thereby allowing us to build a dataset of relations between locations, dishes, and prices. Specifically, we will leverage these relationships between primary and foreign keys to build the combined dataset:

- Menu.id (PK) -> MenuPage.menu\_id (FK)
- MenuPage.id (PK) -> MenuItem.menu\_page\_id (FK)
- MenuItem.dish\_id (PK) -> Dish.id (FK)

Then, we will query this combined dataset to determine the average price of each available dish and number of distinct locations that offer that dish to fulfill our target use case( $U_1$ ).

**S2: profiling of D to identify the quality problems P that need to be addressed to support  $U_1$ .** Below is a list of quality problems (P) we will address via data cleaning:

- Menu.location contains messy and duplicate values (e.g. "[The Waldorf Astoria Hotel (New York, N.Y.)]"; "Adam's Restaurant" vs. "Adams' Restaurant"; "Waldorf Astoria" vs. "Waldorf Astoria"),

		7
--	--	---

--	--	--

so we propose to cluster that and come with the most relevant list of location names with unified names. These need to be addressed to allow for accurate aggregations in support of  $U_1$ .

- Menuitem.price is recorded as String data type, so we will transfer it to numeric, and contains blank values or zero values that need to be ignored in the analysis for  $U_1$ . This needs to be done to support our  $U_1$ , which performs computations (avg) based on Menuitem.price.
- Dish.name contains messy and invalid values that need to either be cleaned or ignored in the analysis for  $U_1$  (e.g. "\$4.50"; " Assorted Sausage"; ">> American Rye"). Presence of many special symbols prevent effective clustering and so effective aggregation of "Dish" and "location" names, needed support of  $U_1$ . Also, presence of any special characters in columns "sponsor" and "place" can prevent effectively filtering out rows with "New York" mentioning, which is required for  $U_1$

**S3: performing the data cleaning process using one or more tools to address the problems P.** We will potentially utilize the following tools to perform data cleaning:

- OpenRefine: initial cleaning of the messy raw data
- SQLite: checking integrity constraints to identify duplicative keys and data
- SQLite: joining data tables to build and query the combined dataset
- YESWorkflow: documenting the data cleaning provenance and workflow

**S4: checking that your new dataset  $D_0$  is an improved version of D.** By inspecting our new dataset  $D_0$ , we will check integrity constraints and document that the Menu.location, Menuitem.price, and Dish.name fields no longer contain messy, duplicate, or blank data. Additionally, we will check that the data obtained from our queries is aggregated by Menu.location and Dish.name to confirm that our target  $U_0$  is satisfied.

**S5: documenting the types and number of changes that have been executed on D to obtain  $D_0$ .** Using provenance metadata generated by OpenRefine and YesWorkflow, we will document all the data cleaning actions taken against D to arrive at  $D_0$ . Here is a tentative list of changes we expect to make on D to obtain  $D_0$ :

1. Using OpenRefine, remove any unrelated ";", and spaces from the columns used in our analysis. We will also need to check if we will have any quality issues due to absence or presence of quotes in the name and location field of Dish or Menu tables.
2. Using OpenRefine, correct any dirty data issues for the Dish.name and Menu.location fields, including but not limited to: removing unnecessary spaces, clustering to deduplicate, identifying invalid values for removal, and normalizing text case.
3. Using OpenRefine, we propose to delete the rows in the Menu table with invalid or missing location to query on real locations.
4. Using OpenRefine, we propose to delete the rows in the Dish table with invalid or missing names to query on real dish names.
5. Using OpenRefine, convert Menuitem.price, Dish.lowest\_price, Dish.highest\_price from String to numeric data type.
6. Using SQLite, join the Menu, MenuPage, Menuitem, and Dish tables (after data cleaning tasks completed with OpenRefine) to create a combined dataset.
7. Using SQLite, query the combined dataset described above to obtain location(s) associated with the max and min price values associated with each dish, aggregated by the Dish.name field. Data entries with invalid or null data will be filtered out.

		8
--	--	---



--	--	--

8. Using SQLite or Datalog, check integrity constraints against  $D_0$  to validate that our data cleaning has been sufficient for satisfying  $U_1$  and that our results for  $U_1$  are as expected.
9. Using YesWorkflow, we will document each of our steps above to generate our workflow model diagrams.

**For Phase 2, we propose to tentatively split the work as follows:**

- Initial data cleaning with OpenRefine – Marina Polupanova
- Datalog/Python/SQLite cleaning, integrity constraint checking, and querying – Wei-Lun Tsai
- YesWorkflow for documenting data provenance to produce a workflow diagram for the actual data cleaning steps that were performed – both
- Final report for Phase 2 – both

--	--	--

# CS513: Theory & Practice of Data Cleaning

## **Final Project Phase II (Summer 2021)**

Professor: Bertram Ludäscher

Team: Marina Polupanova (marinap2@illinois.edu), Wei-Lun Tsai (wltsai2@illinois.edu)

University of Illinois, Urbana-Champaign

		10
--	--	----

--	--	--

## 1. Data Cleaning Performed

### Data Cleaning Using OpenRefine

First, before trying any further queries, as we already identified a number of data issues to be addressed, we have loaded the tables “Dish.csv” and “Menu.csv” to OpenRefine, and started cleaning following columns with following target:

1. Dish.name - to support the U1 case, this column will need to contain the least number of distinct values and least number of special characters possible. To make sure that we will make clustering in a most effective way, we need to ensure to remove the most special symbols first.
2. Menu.location - same target as for the table “Dish.name”
3. Menu.place and Menu.sponsor we need just for filtering out the rows with different variants of “New York” mentioning, so the main target was to remove as much as possible special characters, which can increase the number of search variants of “New York” regex expression (example is presence of unexpected variant “N.Y.”), and cluster values to remove even more wrong or inconsistent records, which will be hard to remove with regex. It was not the target to clean those columns 100%, as they were used just for filtering. We also used those two columns to better explore the dataset and OpenRefine capabilities, to better prepare for cleaning “location” and “name” columns.
4. All required numeric columns from “Menu” and “Dish” like “price”, “highest\_price”, “lowest\_price” and “id”’s we just transformed to numeric, to avoid wrong decimal separators or any special symbols, to make efficient average calculation for U1. Rest we deleted.
5. Columns “MenuItem.id”, “MenuItem.dish\_id”, “MenuItem.menu\_page\_id”, “MenuPage.id”, “MenuPage.menu\_id”, which we used for our joins for U1, contained integer values (even if in text format), so SQLite used them without need for clean, so we did not modify it anyhow in OpenRefine. We did not need to convert it to a numeric type because per [SQLite documentation](https://www.sqlite.org/datatype3.html)<sup>2</sup>, it automatically converts to REAL or INTEGER values when mathematical or comparative operations are performed against the field.

### Detailed cleaning Steps: OpenRefine

#### Menu table:

1. Column “Menu.id”: we have attempted to trim whitespaces, but there were no any.
2. Column “Menu.sponsor”:
  - a. We have trimmed whitespaces at the end and the beginning of the line(14 changes done), and trimmed consecutive spaces (127 changes are done).
  - b. Did the clustering with key collision method, keying function=fingerprint (5440 changes were done), and unified the names of the places according to the clustering, to have it unified. The logic of choosing the name was:

---

<sup>2</sup> <https://www.sqlite.org/datatype3.html>

--	--	--

- i. When the hotel name was appearing, we have chosen to put “hotel” word at the end, so we had chosen the substitution for different variants like this even if this variant with “hotel” at the end was not the most popular one.

518

- Hotel Imperial (5 rows)
- IMPERIAL HOTEL (5 rows)
- Imperial Hotel (4 rows)
- HOTEL IMPERIAL (3 rows)
- Impérial Hotel (1 rows)

☒

Imperial Hotel

- ii. In case, if in some hotel’s name we have met “ ’ ” sign, we have chosen the variant with this sign even if it was not the most popular one. We mostly have chosen the variants with “ ’ ” at the end of the word, to indicate plurals, except for some names like “Child’s”, where we supposed that the apostrophe was indicating single name.

38

- Sherry's (6 rows)
- SHERRY'S (1 rows)
- Sherrys (1 rows)

☒

Sherry's

- iii. During clustering, it was identified some rows with the name “(CLUB)”, “? CLUB” which may mean that the source of the dataset were not sure which club it was, so we have changed those on “Some club” to leave us the opportunity to filter this rows afterwards having uncertainty of the real place.
- iv. During Clustering&Edit, we have removed “( )” or “[ ]”, dashes, “?” or any quotes around the names, which were same as some existing name. Those characters most probably meant that the creator of the dataset was unsure of those values, but for the purpose of our use case we assumed that he was correct and removed it.

23

- S.S. "Ile De France"" (2 rows)
- S.S. Ile De France (1 rows)

☒

S.S. Ile De France

25

- \The Manor\" (4 rows)
- The Manor (1 rows)

☒

The Manor

22

- U.S.S. New York (1 rows)
- U.S.S. New York (?) (1 rows)

☒

U.S.S. New York

22

- (FIFTH AVE. HOTEL) (1 rows)
- FIFTH AVE. HOTEL (1 rows)

☒

FIFTH AVE. HOTEL

- v. During Clustering &Edit, we have changed all the names like “ R[Restaurant name and/or location not given Restaurant name and/or location not given” or “[Not given]” to “Unknown” as we have identified, that there was already such field value, and due to big difference between them in terms of text content,

--	--	--

there were no way to unify it somehow else. This will help us to filter out and delete missing values in U1.

2	5	<ul style="list-style-type: none"> <li>• [Not Given] (3 rows)</li> <li>• [Not given] (2 rows)</li> </ul>	<input checked="" type="checkbox"/>	Unknown
---	---	--	-------------------------------------	---------

- vi. During Clustering&Edit, in case if there were several choices, we have chosen the variant where we have all words starting with the capital letter rather than first word with capital and next all with small.

2	34	<ul style="list-style-type: none"> <li>• Park Lane (33 rows)</li> <li>• Park lane (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Park Lane
---	----	---	-------------------------------------	-----------

- vii. During Clustering&Edit, we tried to choose the variant which will not contain any special symbols, like used in French or German - to be able to parse the text further using only English characters.

2	2	<ul style="list-style-type: none"> <li>• Cafe de Paris (1 rows)</li> <li>• Café de Paris (1 rows)</li> </ul>	<input type="checkbox"/>	Cafe de Paris
---	---	--	--------------------------	---------------

5	174	<ul style="list-style-type: none"> <li>• [Restaurant name and/or location not given] (118 rows)</li> <li>• [Restaurant Name And/Or Location Not Given] (41 rows)</li> <li>• Restaurant name and/or location not given] (13 rows)</li> <li>• Restaurant name and/or location not given (1 rows)</li> <li>• [Restaurant name and/or location not given (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Unknown
---	-----	--	-------------------------------------	---------

- c. Then we did reclustering, as we have mentioned during the first clustering that some of the clusters are in fact can be also merged together. We used method “nearest neighbour”, and the key “levenshtein” with radius=1 and Block chars=6. At this time, more of the orthographic errors were fixed. It was worth fixing those orthographic errors, as we needed to ensure that the words “New York” would not have any errors in that and would be able to easily filter out at the last step. Totally 3347 changes were done.
- i. In the example below, we can see different ways of naming “AMERICA” and “LINE”, also “GENERAL”, and “NORDDEUTSCHER”. When changing such records, we have used English language rules, or in case of some other language, some arbitrarily chosen variant.

--	--	--

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
4	23	<ul style="list-style-type: none"> <li>HAMBURG-AMERICA LINIE (12 rows)</li> <li>HAMBURG-AMERIKA LINE (7 rows)</li> <li>HAMBURG-AMERICAN LINE (3 rows)</li> <li>HAMBURG-AMERICA LINE (1 rows)</li> </ul>	<input type="checkbox"/>	HAMBURG-AMERICA LINIE
3	692	<ul style="list-style-type: none"> <li>NORDDEUTSCHER LLOYD BREMEN (667 rows)</li> <li>NORDDEUTSCHER LLOYD BREMEN (21 rows)</li> <li>NORDDEUTECHER LLOYD BREMEN (4 rows)</li> </ul>	<input type="checkbox"/>	NORDDEUTSCHER LLOYD BREMEN
3	31	<ul style="list-style-type: none"> <li>COMPAGNIE GENERAL TRANSATLANTIQUE (22 rows)</li> <li>COMPAGNIE GENERALE TRANSATLANTIQUE (8 rows)</li> <li>COMPNGNIE GENERAL TRANSATLANTIQUE (1 rows)</li> </ul>	<input type="checkbox"/>	COMPAGNIE GENERAL TRANSATLANTIQUE

Just in order not to make any other reclustering, in case if we have seen the similar clusters we have copied into “New Cell Value” the same name as we have chosen for already met similar clusters. For example, “NORDDEUTSCHER LLOYD BREMEN” and “HAMBURG-AMERICA LINE” had so many variations that they appeared as a cluster multiple times. We have assigned this unified name to every its variant.

- ii. Some of the clusters, where it was not possible to define which was the right variant - we left as it is. Example:

2	2	<ul style="list-style-type: none"> <li>Supper Menu, S. S. Vaderland, March 18th, 1910. (1 rows)</li> <li>Supper Menu, S. S. Vaderland, March 19th, 1910. (1 rows)</li> </ul>	<input type="checkbox"/>	Supper Menu, S. S. Vaderland, March 18th, 1910.
---	---	--	--------------------------	---

After re-clustering, only the items for which we did not know the right unified name have left in the choice fields. We did not change it further, because for our Use case 1 it did not matter: we did not identified anything, related to unique “sponsor” name, so only one thing we cared is that workds “NY” or “New York” were without orthographic errors and in minimal possible variants of cases (and in the rest of clusters there were no issue with that).

c. As during clustering we have noticed, that part of the names is given in Title case, and part in the upper case, we had transformed all the values in the column to the Title case, to be able further more effectively filter “NY” or “New York” values, giving less number of variants to the filter. Here 8212 transformations were done.

3. Column “Menu.location”. Following changes were done:
  - a. Trim heading and trailing whitespaces: 14 changes made;
  - b. Collapsing consecutive whitespaces - 555 changes done;
  - c. Transformed all values to Title case. 1281 transformation done.
  - d. Cluster&Edit on this column (183 clusters found) with key collision method, keying function=fingerprint. 4209 changes done. The choices were made to join the clusters:
    - i. We have replaced all the cells with the unknown content by the label “Unknown”:

--	--	--

4	166	<ul style="list-style-type: none"> <li>• [restaurant Name And/or Location Not Given] (151 rows)</li> <li>• Restaurant Name And/or Location Not Given] (13 rows)</li> <li>• Restaurant Name And/or Location Not Given (1 rows)</li> <li>• [restaurant Name And/or Location Not Given (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	<input type="text" value="Unknown"/>
2	36	<ul style="list-style-type: none"> <li>• [restaurant And/or Location Not Given.] (30 rows)</li> <li>• [restaurant And/or Location Not Given] (6 rows)</li> </ul>	<input checked="" type="checkbox"/>	<input type="text" value="Unknown"/>

- ii. All the cells, where we had the “()”, “{}” and “?”, which may mean that the person making this table was hesitating if the value is correct, we have deleted with the assumption that this person is correct and the values were like he supposed. Example:

4	33	<ul style="list-style-type: none"> <li>• Fifth Avenue Hotel (28 rows)</li> <li>• Fifth Avenue Hotel (?) (2 rows)</li> <li>• Fifth Avenue Hotel? (2 rows)</li> <li>• (fifth Avenue Hotel?) (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	<input type="text" value="Fifth Avenue Hotel"/>
---	----	---	-------------------------------------	---

- iii. We have chosen as in the previous column the values with “ ‘ ” even if they were not the most popular. Example:

3	67	<ul style="list-style-type: none"> <li>• Childs (58 rows)</li> <li>• Childs' (8 rows)</li> <li>• Child's (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	<input type="text" value="Child's"/>
---	----	--	-------------------------------------	--------------------------------------

- iv. We have chosen in the hotel names the ones, where the “hotel” was at the end. Example:

3	29	<ul style="list-style-type: none"> <li>• Gramercy Park Hotel (20 rows)</li> <li>• Hotel Gramercy Park (8 rows)</li> <li>• Gramercy Park Hotel; Hotel Gramercy Park (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	<input type="text" value="Gramercy Park Hotel"/>
---	----	--	-------------------------------------	--

- v. Substituted all the names with excessive quotes like:

2	4	<ul style="list-style-type: none"> <li>• S.S. American Shipper (3 rows)</li> <li>• S.S. "american Shipper"" (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	<input type="text" value="S.S. American Shipper"/>
---	---	---	-------------------------------------	--

		15
--	--	----

--	--	--

- vi. We had several clusters of similar names, which we have substituted over this pass to single unified name for all clusters (chosen the one with less errors, and without commas):

4	784	<ul style="list-style-type: none"> <li>Norddeutscher Lloyd Bremen (780 rows)</li> <li>Norddeutscher Lloyd, Bremen (2 rows)</li> <li>Bremen Norddeutscher Lloyd (1 rows)</li> <li>Norddeutscher Lloyd Bremen; (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Norddeutscher Lloyd Bremen
2	29	<ul style="list-style-type: none"> <li>Nordeutscher Lloyd Bremen (27 rows)</li> <li>Nordeutscher Lloyd, Bremen (2 rows)</li> </ul>	<input type="checkbox"/>	Norddeutscher Lloyd Bremen

- vii. Every time, when there was “Co.”, we have chosen the variant with the “.”:

2	2	<ul style="list-style-type: none"> <li>Ancient And Honorable Artillery Co Of Massachusetts (1 rows)</li> <li>Ancient And Honorable Artillery Co. Of Massachusetts (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Ancient And Honorable Artillery Co.
---	---	---	-------------------------------------	-------------------------------------

- viii. We have chosen the variant without special national symbols (German, French):

2	3	<ul style="list-style-type: none"> <li>Muhlenkamper Fahrhaus (2 rows)</li> <li>Mühlenkamper Fährhaus (1 rows)</li> </ul>	<input type="checkbox"/>	Muhlenkamper Fahrhaus
---	---	--	--------------------------	-----------------------

- ix. We have chosen the version with more correct grammar. Example:

2	3	<ul style="list-style-type: none"> <li>Michigan Society Of The Sons Of The American Revolution (2 rows)</li> <li>Michigan Society Sons Of The American Revolution (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Michigan Society Of The Sons Of Th
---	---	---	-------------------------------------	------------------------------------

- e. We have continued clustering with the method “nearest neighbour”, and the key “levenshtein” with radius=1 and Block chars=6 (158 clusters found). With this method, as with the column “sponsor” we had mostly orthographical errors, which we had to fix. 3449 changes were done.
- i. Orthographical errors we have fixed on the English language rules. Here also we can note many variants of names with missing or extra letters. Examples of the errors we have seen:



--	--	--

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
4	21	<ul style="list-style-type: none"> <li>Hamburg America Linie (12 rows)</li> <li>Hamburg Amerika Line (5 rows)</li> <li>Hamburg American Line (3 rows)</li> <li>Hamburg America Line (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Hamburg America Line

Example with extra letter:

2	7	<ul style="list-style-type: none"> <li>Pullman Dining Car Service The President (5 rows)</li> <li>Pullman Dining Car Serfvice The President (2 rows)</li> </ul>	<input checked="" type="checkbox"/>	Pullman Dining Car Service The Pre:
---	---	---	-------------------------------------	-------------------------------------

Example with missing letter:

2	5	<ul style="list-style-type: none"> <li>Palace Hotel Restaurant And Ladies Grill Room (3 rows)</li> <li>Palace Hotel Restaurat And Ladies Grill Room (2 rows)</li> </ul>	<input checked="" type="checkbox"/>	Palace Hotel Restaurant And Ladies
---	---	---	-------------------------------------	------------------------------------

- ii. Unified some names, where in one variant we had spaces, and in the other - have no.

2	791	<ul style="list-style-type: none"> <li>Waldorf Astoria (707 rows)</li> <li>Waldorf-astoria (84 rows)</li> </ul>	<input checked="" type="checkbox"/>	Waldorf Astoria
---	-----	---	-------------------------------------	-----------------

- iii. Chosen the variants, where after abbreviation there is a space, and where there is no space in-between abbreviation letters:

2	5	<ul style="list-style-type: none"> <li>Red Star Line S.S. Southwark (3 rows)</li> <li>Red Star Line S.S.Southwark (2 rows)</li> </ul>	<input checked="" type="checkbox"/>	Red Star Line S.S. Southwark
---	---	---	-------------------------------------	------------------------------

2	2	<ul style="list-style-type: none"> <li>S. S. President Wilson (1 rows)</li> <li>S.S. President Wilson (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	S.S. President Wilson
---	---	---	-------------------------------------	-----------------------

- f. After the operations above, 21 clusters were still found with same method. It was not clear what version of the name was the correct (even if there was evidence that there is some wrong and one potentially correct there. This could potentially influence the case, as we will need to define prices of the dish for the unique location. So, we have decided to merge all the clusters with the proposed by OpenRefine value. We did not changed only "The Cortland/The Portland" as those variants seemed different, and Bellini/Cellini". 111 cells were changed on this step.

		17
--	--	----

2	5	<ul style="list-style-type: none"> <li>Rathskeller (4 rows)</li> <li>Ratskeller (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Rathskeller
2	15	<ul style="list-style-type: none"> <li>Nippon Yusen Kaisha S.S.Kobe Maru (9 rows)</li> <li>Nippon Yusen Kaisha S.S. Kobe Maru (6 rows)</li> </ul>	<input checked="" type="checkbox"/>	Nippon Yusen Kaisha S.S. Kobe Maru
2	5	<ul style="list-style-type: none"> <li>Cortland (4 rows)</li> <li>Cortlandt (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Cortland
2	2	<ul style="list-style-type: none"> <li>Bellini (1 rows)</li> <li>Cellini (1 rows)</li> </ul>	<input type="checkbox"/>	Bellini
2	2	<ul style="list-style-type: none"> <li>Luncheon Menu, S. S. Vaderland, March 18th, 1910. (1 rows)</li> <li>Luncheon Menu, S. S. Vaderland, March 19th, 1910. (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Luncheon Menu, S. S. Vaderland, March 18th, 1910.
2	13	<ul style="list-style-type: none"> <li>Royal Poinciana (12 rows)</li> <li>Royal Ponciana (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Royal Poinciana

- g. After the previous step, only 2 clusters left, which as I have mentioned we left unmerged.

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
2	12	<ul style="list-style-type: none"> <li>The Cortland (7 rows)</li> <li>The Portland (5 rows)</li> </ul>	<input type="checkbox"/>	The Cortland
2	2	<ul style="list-style-type: none"> <li>Bellini (1 rows)</li> <li>Cellini (1 rows)</li> </ul>	<input type="checkbox"/>	Bellini

#### 4. Column "Menu.place":

- a. As we have checked, transforming values to Title case, create too many variants of "NY" cases, as there are many special symbols like "[", ";", in the strings so we make the following GREL expression to remove it first. 1167 changes were made.

```
value.replace(/;/, " ").replace(/[\[\]]/, " ")
```

--	--	--

Expression

Language General Refine Expression Language (GREL) ▾

value.replace(/;/," ").replace(/\[\{\}\]/," ")

Preview

History

Starred

Help

row	value	value.replace(/;/," ").replace ...
1.	HOT SPRINGS, AR	HOT SPRINGS, AR
2.	MILWAUKEE, [WI];	MILWAUKEE, WI
3.	DAMPFER KAISER WILHELM DER GROSSE;	DAMPFER KAISER WILHELM DER GROSSE
4.	DAMPFER KAISER WILHELM DER GROSSE;	DAMPFER KAISER WILHELM DER GROSSE
5.	DAMPFER KAISER WILHELM DER GROSSE;	DAMPFER KAISER WILHELM DER GROSSE
6.	R.M.S. EMPRESS OF CHINA	R.M.S. EMPRESS OF CHINA
7.	NEW YORK, NY	NEW YORK, NY

- Then we have made Trim heading and trailing whitespaces: 1004 changes made;
- Mande collapsing consecutive whitespaces - 713 changes done;
- Transformed all values to Title case. 7337 transformation done.
- Cluster&Edit on this column (253 clusters found) with key collision method, keying function=fingerprint. 2484 changes done. The same choices were made for changing the clusters as were made in the clustering&merging previous columns, with some additional:
  - Where possible, we have chosen “Ny” or “New York” name for NY and New York.

7	49	<ul style="list-style-type: none"> <li>Delmonico's, New York, Ny (41 rows)</li> <li>Delmonico's, New York, N.Y. (2 rows)</li> <li>Delmonicos, New York, N.Y. (2 rows)</li> <li>Delmonico's, (new York, Ny?) (1 rows)</li> <li>Delmonico's. New York, Ny (1 rows)</li> <li>Delmonicos, (new York, Ny?) (1 rows)</li> <li>Delmonicos, New York, Ny (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Delmonico's, New York, Ny
7	20	<ul style="list-style-type: none"> <li>En Route "friedrich Der Grosse" (11 rows)</li> <li>En Route - "friedrich Der Grosse" (4 rows)</li> <li>En Route " Friedrich Der Grosse" (1 rows)</li> <li>En Route "friedrich Der Grosse " (1 rows)</li> <li>En Route 'friedrich Der Grosse" (1 rows)</li> <li>En Route - 'friedrich Der Grosse " (1 rows)</li> <li>En Route - :friedrich Der Grosse" (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	En Route " Friedrich Der Grosse"
7	98	<ul style="list-style-type: none"> <li>New York (87 rows)</li> <li>(new York?) (4 rows)</li> <li>"new York" (3 rows)</li> <li>(new York) (1 rows)</li> <li>New York (?) (1 rows)</li> <li>New York(?) (1 rows)</li> <li>New York? (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	New York

- In case if there was a choice of the state separate by “,” , we used this variant.

--	--	--

3	3	<ul style="list-style-type: none"> <li>• The Hollenden Cleveland Oh? (1 rows)</li> <li>• The Hollenden, Cleveland Oh (1 rows)</li> <li>• The Hollenden, Cleveland, Oh (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	The Hollenden, Cleveland, Oh
---	---	---	-------------------------------------	------------------------------

- iii. In case, if I have noticed, that the state was starting from the small letter due to there was no space after the comma, or if there was some known state, like Massachusets, which was named “Ma” or “mass”, we decided to correct that:

2	13	<ul style="list-style-type: none"> <li>• U.S.M.S. New York (9 rows)</li> <li>• Usms New York (4 rows)</li> </ul>	<input checked="" type="checkbox"/>	U.S.M.S. New York
2	10	<ul style="list-style-type: none"> <li>• Washington,d.C. (7 rows)</li> <li>• Washington,dc (3 rows)</li> </ul>	<input checked="" type="checkbox"/>	Washington, D.C.
2	6	<ul style="list-style-type: none"> <li>• Boston,mass (5 rows)</li> <li>• (boston,mass) (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Boston, Ma
2	6	<ul style="list-style-type: none"> <li>• Copenhagen, Denmark (5 rows)</li> <li>• Copenhagen Denmark? (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Copenhagen, Denmark
2	2	<ul style="list-style-type: none"> <li>• Brunswick,the,boston Mass (1 rows)</li> <li>• Brunswick,the,boston, Mass (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Brunswick,the, Boston Ma

- f. After that, we have recluster and found out 4 more clusters to merge, so we did. 94 cells were changed based on our choice. Also, we did reclustering once more, and found 1 more cluster with similar values, and did 54 more cells changes.

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
3	4	<ul style="list-style-type: none"> <li>• Delmonico's, New York,ny? (2 rows)</li> <li>• Delmonico's New York,ny (1 rows)</li> <li>• Delmonico's, New York,n.Y. (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	Delmonico's New York, Ny
2	68	<ul style="list-style-type: none"> <li>• Delmonico's Ny (56 rows)</li> <li>• Delmonico's, Ny (12 rows)</li> </ul>	<input checked="" type="checkbox"/>	Delmonico's, Ny
2	14	<ul style="list-style-type: none"> <li>• Chicago, Ill (10 rows)</li> <li>• Chicago ,ill (4 rows)</li> </ul>	<input checked="" type="checkbox"/>	Chicago, Ill
2	8	<ul style="list-style-type: none"> <li>• The Arlington, Washington Dc (4 rows)</li> <li>• The Arlington, Washington, D.C. (4 rows)</li> </ul>	<input checked="" type="checkbox"/>	The Arlington, Washington, D.C.

- g. Clustering method “nearest neighbour”, and the key “levenshtein” with radius=1 and Block chars=6 found 178 clusters, so we did merging with the same principles, as for the previously chosen columns. 2019 cells were modified during these clusters merge.

--	--	--

- h. After reclustering, 14 more clusters found, and we have done merging with the variants chosen as per the earlier described principles. 482 cells were modified at this step. We have found one more issue, not met before: the states were mistakenly written for some cities, so we even had to recheck in the Internet, to make sure that NY cities will be appropriately represented for our Use case:

2	5	<ul style="list-style-type: none"> <li>Malone, Nj (4 rows)</li> <li>Malone, Ny (1 rows)</li> </ul>	<input checked="" type="checkbox"/>	<input type="text" value="Malone, Ny"/>
---	---	--	-------------------------------------	---

After cleaning column “place” we have decided to do some more changes with “location” and “sponsor”, which were not done yet, which can help to make better filtering on city/state :

1. Removing “;”, removing “[]”. 58 cells changed for “sponsor”, 59 cells changed for “location”
2. Trimming training and heading whitespaces. 40 cells changed for “sponsor”, 44 cells changed for “location”
3. Removing consecutive whitespaces after the procedures above. 26 cells changed for “sponsor”, 27 cells changed for “location”

As a next step, we will remove all the rest columns apart from “id”, “location”, “place”, “sponsor” from this table to speed up further SQL operations. We will add only 1 additional field: “dish\_count” to make further some integrity constraints check - to define, whether the number of dishes from the table “Dish” which we will find for the Menu.id will comply to this number.

We will transform the columns “id” and “dish\_count” to numbers to be able to process it further accordingly.

Now, as the text of the columns filtered and cleaned enough to clearly define “New York” locations, we will create a new column near all cleaned 3 columns with the names “place\_NY”, “location\_NY”, “sponsor\_NY”, where we will record “true/false” depending, whether in the respective main column we will see the words “New York”, “Ny” (which appeared as we have transformed all to Title case, so NY became Ny), and “ny”(which appeared for the same reason, but in the places where NY appeared after comma or point without a space after it). To make it, following GREL expression was used:

`value.contains(/\b(?:New York|Ny|ny)\b/).`

For column “sponsor” 15986 row filling were done, for “place” - 8123, for “location” - 17547.

		21
--	--	----

--	--	--

### Add column based on column place

New column name

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

Expression  Language General Refine Expression Language (GREL) ▾

No syntax error.

Preview	History	Starred	Help
3.	Dampfer Kaiser Wilhelm Der Grosse	false	
4.	Dampfer Kaiser Wilhelm Der Grosse	false	
5.	Dampfer Kaiser Wilhelm Der Grosse	false	
6.	R.M.S. Empress Of China	false	
7.	New York, Ny	true	
8.	Schnelldampfer Daiser Wilhelm Der Grosse	false	
9.	Dampfer Kaiser Wilhelm Der Grosse	false	
10.	Dampfer Kaiser Wilhelm Der Grosse	false	

Now, if we will observe the dataset, we will see that the expression had identified the rows, belonging to New York correctly, and now all what is needed, is to join all 3 created columns, to have only one column for corresponding rows filtering.

▼ All	▼ id	▼ sponsor	▼ sponsor_NY	▼ place	▼ place_NY	▼ location	▼ location_NY	▼ dish_count	
★🔊	39.	12502	Cunard Line	false	R.M.S. Lucania	false	Cunard Line	false	90
★🔊	40.	12503	Police Department Of The City Of New York	true	Delmonico's	false	Police Department Of The City Of New York	true	30
★🔊	41.	12504	U.S.M.S. New York	true	En Route	false	U.S.M.S. New York	true	99
★🔊	42.	12505	Bartholdi Hotel	false	Ny	true	Bartholdi Hotel	false	203
★🔊	43.	12506	Bartholdi Hotel	false	23rd St & Bway Ny	true	Bartholdi Hotel	false	336
★🔊	44.	12507	Haan's	false	75 St. & Columbus Ave. Ny	true	Haan's	false	392
★🔊	45.	12508	Marie Antoinette Hotel	false	66 St. & Bway, Ny	true	Marie Antoinette Hotel	false	82
★🔊	46.	12509	Red Star Line - S.S. Southwark	false	En Route	false	Red Star Line S.S. Southwark	false	56
★🔊	47.	12510	Marie Antoinette Hotel	false	66 St. & Bway, Ny	true	Marie Antoinette Hotel	false	128
★🔊	48.	12511	Third Panel Sheriff's Jury New York County	true	Hotel Savoy,new York,ny	true	Third Panel Sheriff's Jury New York County	true	35

Now we are joining 3 columns, to receive 1 column with results of all of them, separated with commas:

		22
--	--	----

--	--	--

### Join columns

Select and order columns to join

☒ sponsor\_NY
☐ id
☐ sponsor
☐ place
☒ place\_NY
☐ location
☒ location\_NY
☐ dish\_count

Select All De-select All

Select options

Separator between the content of each column:

Enter one or more characters, or keep blank to join the columns without separator.

☐ Replace nulls with...   
Enter one or more characters, or keep blank to replace nulls with blank strings.
☒ Skip nulls.

☐ In separator and nulls substitutes, use \n for new lines, \t for tabulation, \\n for \n, \\t for \t.

☐ Write result in selected column.
☒ Write result in new column named...

☐ Delete joined columns.

OK Cancel

Next, we are applying the following expression, to get the 1 column with “true/false”:

`value.contains(/\b(?:true)\b/).`

17547 row insertion is done in the column “if\_NY”. You can see on the example below, that the column “if\_NY” really contain the indicator that in one of the 3 initially identified column NY is mentioned in some of the forms:

▼ All	▼ id	▼ sponsor	▼ sponsor_NY	▼ if_NY	▼ place	▼ place_NY	▼ location	▼ location_NY	▼ dish_count
☆	21. 12483	Manhattan Hotel	false	true	New York, Ny	true	Manhattan Hotel	false	129
☆	22. 12484	Boston Boot & Shoe Club	false	false	Brunswick Hotel, Boston, Ma	false	Boston Boot & Shoe Club	false	24
☆	23. 12485	Canadian Pacific Railway	false	false	En Route Aboard R.M.S. Empress Of China	false	Canadian Pacific Railway	false	36
☆	24. 12486	Savoy Hotel	false	true	New York	true	Savoy Hotel	false	15
☆	25. 12487	Pacific Mail Steamship Co.	false	false	En Route Aboard Panama Line Steamship City Of Para	false	Pacific Mail Steamship Co.	false	13
☆	26. 12488	Chi Psi Fraternity	false	true	Holland House, New York City	true	Chi Psi Fraternity	false	19
☆	27. 12489	Colby Alumni Association Of New York	true	true	St. Denis Hotel, New York	true	Colby Alumni Association Of New York	true	21
☆	28. 12490	Imperial Hotel	false	false	(vienna, Austria ?)	false	Imperial Hotel	false	16
☆	29. 12491	Whitefriars Club	false	false	Anderton's Hotel, Fleet Street, E.C. (england?)	false	Whitefriars Club	false	13
☆	30. 12492	Wilson College Alumnae Association	false	false	Hotel Flanders	false	Wilson College Alumnae Association	false	14

During observation of the piece of the new dataset, we still can see some text fields issues, like the one below with Vienna, Austria in “place” column:

☆	25. 12487	Pacific Mail Steamship Co.	false	En Route Aboard Panama Line Steamship City Of Para	false	Pacific Mail Steamship Co.	false	13
☆	26. 12488	Chi Psi Fraternity	false	Holland House, New York City	true	Chi Psi Fraternity	false	19
☆	27. 12489	Colby Alumni Association Of New York	true	St. Denis Hotel, New York	true	Colby Alumni Association Of New York	true	21
☆	28. 12490	Imperial Hotel	false	(vienna, Austria ?)	false	Imperial Hotel	false	16

We have decided not to clean the brackets, as in some cases, they appear to look reasonable:

▼ location
National Verbandes Deutsch Amerikanischer Journalisten Und Schriftsteller (unusual)

But we will further clean the “?” signs as we will make the same assumption as before, that we believe the person who copied the value here, and assume he was correct, naming this place as it is named.

		23
--	--	----

--	--	--

We use following GREL expression:

`value.replace(/\?/, " ").`

For column “sponsor” 118 changes were done, for “place” - 253, for “location” - 107.

Here you can see the same row cleaned up:

27.	12489	Colby Alumni Association Of New York	true	true	St. Denis Hotel, New York	true	Colby Alumni Association Of New York	true	21
28.	12490	Imperial Hotel	false	false	(vienna, Austria )	false	Imperial Hotel	false	16
29.	12491	Whitefriars Club	false	false	Anderton's Hotel, Fleet Street, E.C. (england )	false	Whitefriars Club	false	13

And as a final step, we remove the columns “place\_NY”, “location\_NY”, “sponsor\_NY” as they are no longer needed and save the csv file and the json for further use.

### **“Dish” table.**

After cleaning Menu.csv file, we are coming to cleaning “Dish.csv” file. From this file, we will need just “name” and “id” columns for our case, and we will leave also (without cleaning) “highest\_price” and “lowest\_price”.

“id” column: only one change we do is we convert it to numeric format (428082 changes done).

“name” column we will clean with the similar operations, like we used for columns of Menu table:

1. Trimming leading and trailing whitespaces - 9288 changes done
2. First, as we can see from the data browsing, there are some cases, when at the end of the line there are misleading “.” and “;”. We are going to delete those at the end of the line, and also will attempt to remove “;” at the end of the line, if there are any:

546.	656	Fruit.		37	38	1889	1901		
------	-----	--------	--	----	----	------	------	--	--

1041.	1239	Cocoa,	
1042.	1240	Cocovena,	
1043.	1241	Biscuits	
1044.	1243	Frische Milch	
1045.	1244	Boiled turbot	
1046.	1245	Sahne	
1047.	1246	Sauce crevette	
1048.	1247	Crescents ,	

we used following expression to remove “.” and “;” at the end of the lines:

`value.replace(/[.,$]/, " ").replace(/[,,$]/, " ")`

At the end of this step, 11694 corresponding signs were removed.

Then, to remove “;” at the end of the strings (which can appear according to experience with “Menu” table) we used:

`value.replace(/[,,$]/, " ")`

and 15 cells were changed.

		24
--	--	----



--	--	--

- There appeared some irrelevant quotes at the beginning of some rows, followed by space and further words:

1161.	1375	" yellow
1162.	1376	Benedictine

and also, there were some irrelevant & signs at the beginning of the line, or even instead of the line:

530.	637	&		4	4	1901	1901	0.0	0.0
------	-----	---	--	---	---	------	------	-----	-----

We have cleaned it with following expression (creating somewhere blank rows):

```
value.replace(/^[&]$/, " ").replace(/^" /, " ")
```

280 cells were changed.

- There are some mentions "(to order)", which in fact do not correspond to the name of the meal. We have decided to remove it.

731.	889	Hamburger Steaks
732.	890	Broiled Cumberland Ham (to order)
733.	891	Veal Cutlets, Tomato Sauce
734.	892	Stewed rhubarb
735.	893	Frizzled Bacon (to order)
736.	894	Fried Potatoes
737.	895	Stewed Tripe, White Sauce

Following expression used:

```
value.replace(/\(to order\)$/, " ")
```

118 cells were changed.

- Some ":" and "-" found in the values at the beginning and end of line, so we will clean it as well with GREL.

428072.	520693	Coffee, Tea, or Demi Tasse
428073.	520694	Playing Cards :

```
value.replace(/[:]$/, " ").replace(/^[^-]$/, " ")
```

219 cells were changed based on this.

- It was defined, that at the end of some lines there were prices separated by some commas values. We will trim it down.

		25
--	--	----

--	--	--

427882.	520471	Served on a steak roll with our own BBQ sauce - includes choice of topping .....4.25
427883.	520472	Add .70 for each extra topping
427884.	520473	Freshly baked potato, buttered and stuffed with your choice of the following:
427885.	520474	Platters
427886.	520475	including choice of topping ..... 3.25
427887.	520476	Fresh potatoes, sliced and cooked golden brown ..... 1.25
427888.	520477	Toppings

```
value.replace(/\.+ ?\d\.\d\d$/, " ")
```

208 cells changed based on this

7. Some items are in brackets:







☆	🚩	427874.	520463	COFEE, TEA OR SNAKE
☆	🚩	427875.	520464	(GRATUITIES AND TAXES INCLUDED)
☆	🚩	427876.	520465	COFEE, TEA OR SANKA
☆	🚩	427877.	520466	Topless Burger
☆	🚩	427878.	520467	The Sirloin Buraer

We will leave it as it is, as in some cases the brackets can be reasonable, so will be hard to imagine all possible variants and filter it:

7.	8	Chicken soup with rice
8.	9	Clam broth (cup)
9.	10	Cream of new asparagus, croutons

8. Some items include dates, which will need to be replaced to null values, as they definitely do not represent a dish. It will be hard to filter out all such values, as in some cells as you will see below, the year means some part of the meal (year of wine for instance). So the regex for filtering will cover only part of the cases.

427951.	520540	19 JUILLET 1989
427952.	520541	CHILD'S PLATE Taco, tostada or tamale with beans or posole, beverage

		428064.	<a href="#">520685</a>	February 22, 1916
		428065.	<a href="#">520686</a>	Guacanayabo Bay, Cuba
		428066.	<a href="#">520687</a>	Surf Pot Luck

--	--	--

☆	🗨	427916.	520505	Menus 1984
☆	🗨	427917.	520506	Canned Beer and Drinks
☆	🗨	427918.	520507	Allegrini Valpolicella Classico
☆	🗨	427919.	520508	Calzone Ripieno mit Schinken- und Kasefüllung
☆	🗨	427920.	520509	Calzone Ripieno mit Schinken- und Käsefüllung
☆	🗨	427921.	520510	Wine: 1996 Allegrini Valpolicella Classico

value.replace(/.\*\d\d, \d\d\d\d \$/, " ").replace(/.\*\d\d [a-zA-z]+ \d\d\d\d \$/, " ")

35 cells were replaced by NULL values.

9. There will be some items, which do not make sense, so we filter it out:

☆	🗨	427969.	520590	10th
☆	🗨	427970.	520591	5th
☆	🗨	427971.	520592	Bottle

value.replace(/^\d?\d?th\$/, " ")

Fortunately, only those 2 were filtered.

10. In some items, it seems that the numbers of menu items were not filtered out:

☆	🗨	426804.	519268	58 Aigle Hospices Cantonaux (Etat de Vaud)..... 1968
☆	🗨	426805.	519269	59 Yvorne, Clos du Rocher (Obriest S.A.)..... 1968
☆	🗨	426806.	519270	59 Yvorne, Clos du Rocher (Obriest S.A.)..... 1968
☆	🗨	426807.	519271	61 Fendant pétillant, Les Murettes (R. Gilliard)..... 1969
☆	🗨	426808.	519272	61 Fendant pétillant, Les Murettes (R. Gilliard)..... 1969
☆	🗨	426809.	519273	62 Johannisberg Mont d'Or (Domaine du Mont d'Or)..... 1969
☆	🗨	426810.	519274	62 Johannisberg Mont d'Or (Domaine du Mont d'Or)..... 1969
☆	🗨	426811.	519275	53 Mont d'Or, Riesling-Auslese (Domaine du Mont d'Or)..... 1968

We will delete here the numbers at the beginning, but will leave the numbers at the end, as it contain the year of the wine.

value.replace(/^\d?\d/, " ")

6820 cells were filtered out.

11. Some items contained \* at the beginning of the line:

value.replace(/^ \*\/, " ")

1200 lines were changed.

12. Now, we will trim once more the head and trailing whitespaces, as we did many substitutions.  
20283 changes were made

13. Consecutive white spaces removed: 6481 changes done.

14. As some items were written in capital case, we will transform all in the Title case.

		27
--	--	----

--	--	--

☆	🗨	38.	41	Strawberries	
☆	🗨	39.	42	Preserved figs	
☆	🗨	40.	43	BLUE POINTS	
☆	🗨	41.	44	CONSOMME ANGLAISE	
☆	🗨	42.	45	CREAM OF CAULIFLOWER	

284628 cells were changed with this change.

- Next, we have done clustering with key collision method, keying function “fingerprint”. It found 29528 clusters. As there were so many clusters, and the result of clustering and proposed substitutions for merge looked pretty adequate, like below, we have chosen to accept all the changes.

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
23	35	<ul style="list-style-type: none"> <li>Imported Ginger Ale, C. &amp; C (7 rows)</li> <li>C &amp; C Imported Ginger Ale (3 rows)</li> <li>Ginger Ale, Imported C &amp; C (3 rows)</li> <li>Ginger Ale Imported, C. &amp; C (2 rows)</li> <li>Imported Ginger Ale C. &amp; C (2 rows)</li> <li>"c. &amp; C." Ginger Ale, Imported (1 rows)</li> <li>C. &amp; C. Ginger Ale, Imported (1 rows)</li> <li>C. &amp; C. Imported Ginger Ale (1 rows)</li> <li>Ginger Ale (c. &amp; C.), Imported (1 rows)</li> <li>Ginger Ale (imported) C. &amp; C (1 rows)</li> <li>Ginger Ale, C &amp; C (imported) (1 rows)</li> <li>Ginger Ale, C. &amp; C. (imported) (1 rows)</li> <li>Ginger Ale, C. &amp; C., Imported (1 rows)</li> <li>Ginger Ale, Imported C. &amp; C (1 rows)</li> <li>Ginger Ale, Imported, C &amp; C (1 rows)</li> <li>Ginger Ale, Imported, C. &amp; C (1 rows)</li> <li>Imported (c. &amp; C.) Ginger Ale (1 rows)</li> <li>Imported C. &amp; C. Ginger Ale (1 rows)</li> <li>Imported Ginger Ale (c &amp; C) (1 rows)</li> <li>Imported Ginger Ale (c. &amp; C.) (1 rows)</li> <li>Imported Ginger Ale C &amp; C (1 rows)</li> <li>Imported Ginger Ale, C &amp; C (1 rows)</li> <li>Imported Ginger Ale. C. &amp; C (1 rows)</li> </ul>	<input type="checkbox"/>	Imported Ginger Ale, C. & C

21	38	<ul style="list-style-type: none"> <li>Broiled Chicken (half) (8 rows)</li> <li>Half Broiled Chicken (4 rows)</li> <li>Broiled Chicken, Half (3 rows)</li> <li>Broiled Chicken Half (2 rows)</li> <li>Broiled Half Chicken (2 rows)</li> <li>Chicken (half), Broiled (2 rows)</li> <li>Chicken, Broiled, Half (2 rows)</li> <li>Half Chicken, Broiled (2 rows)</li> <li>(half) Broiled Chicken (1 rows)</li> <li>Broiled Chicken - (half) (1 rows)</li> <li>Broiled Chicken, (half) (1 rows)</li> <li>Chicken (half) (broiled) (1 rows)</li> <li>Chicken Broiled (half) (1 rows)</li> <li>Chicken Broiled Half (1 rows)</li> <li>Chicken Broiled, Half (1 rows)</li> <li>Chicken Half, Broiled (1 rows)</li> <li>Chicken, Broiled (half) (1 rows)</li> <li>Chicken, Half, Broiled (1 rows)</li> <li>Half (broiled Chicken) (1 rows)</li> <li>Half Chicken Broiled (1 rows)</li> <li>[broiled] Chicken (half) (1 rows)</li> </ul>	<input type="checkbox"/>	Broiled Chicken (half)
----	----	--	--------------------------	------------------------

--	--	--

As the number of clusters was huge, OpenRefine had issues with merging them all within one operation. So, we chosen multiple times the thresholds in the clustering options (the number of choices for instance) to make less the number of clusters merge per operations.

Method: key collision Keying Function: fingerprint 29039 clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
6	9	<ul style="list-style-type: none"> <li>Assorted Hors D'oeuvres (4 rows)</li> <li>Assorted Hors D'oeuvres (1 rows)</li> <li>Assorted Hors D'oeuvres (1 rows)</li> <li>Hors D'oeuvres (assorted) (1 rows)</li> </ul>	<input type="checkbox"/>	Assorted Hors D'oeuvres

# Choices in Cluster:

# Rows in Cluster:

First, we used the filter “Choices in Cluster”, and then - “Average Length of Choices”. When we have came to the last items in “Average length of choices, it turned out that on the range 2.5 - 4.5 some of the items were non-meaningful, so we turned it to null:

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
2	2	<ul style="list-style-type: none"> <li>.00 (1 rows)</li> <li>00 (1 rows)</li> </ul>	<input type="checkbox"/>	.00
2	2	<ul style="list-style-type: none"> <li>.85 (1 rows)</li> <li>8,5 (1 rows)</li> </ul>	<input type="checkbox"/>	.85
2	3	<ul style="list-style-type: none"> <li>Te (2 rows)</li> <li>Te' (1 rows)</li> </ul>	<input type="checkbox"/>	Te

So following number of cells was changed as a result of steps which were done during clustering:

16. Mass edit 1274 cells in column name
17. Mass edit 1428 cells in column name
18. Mass edit 4563 cells in column name
19. Mass edit 7746 cells in column name
20. Mass edit 8522 cells in column name
21. Mass edit 17780 cells in column name
22. Mass edit 90 cells in column name
23. Mass edit 1220 cells in column name
24. Mass edit 4425 cells in column name
25. Mass edit 5083 cells in column name
26. Mass edit 6774 cells in column name
27. Mass edit 9815 cells in column name
28. Mass edit 13245 cells in column name
29. Mass edit 9820 cells in column name
30. Mass edit 1268 cells in column name
31. Mass edit 637 cells in column name
32. Mass edit 98 cells in column name

--	--	--

So totally 93151 cells were changed after clustering step.

- At the last phase, we are deleting unnecessary columns (everything apart from “id”, “name”, “highest\_price” and “lowest\_price”, and convert later 2 columns to numeric format. (398981 cells were changes for each of two columns).

## Secondary cleaning Steps: OpenRefine

When we have done initial join of the table, and checked integrity constraints, we have identified that the joined table, which does not contain any missing values and contains only needed columns/rows for our Use case still has some issues with data in the text columns, which can potentially lead to issues with values aggregation.

We have found those issues in the columns “location”, “dish\_name”. Here are the found issues and changes which we did in OpenRefine to fix it:

“dish\_name”:

- Some entries in the “dish\_name” contained “\*” symbols, which were not meaningful. Example:

**The 400 \*\*\***

We have deleted it with GREL (42 changes done):

```
value.replace(/\*/, "\ ")
```

- Some entries still appeared to use “?” symbols, so we have removed it. Examples:

? Boiled Lobster

?? Fry?? Stew

We have removed it with expression (19 changes done):

```
value.replace(/\?/, " ")
```

- Some lines had “+” signs. We have removed 2 items:

```
value.replace(/\+/, " ")
```

- Some items had “-” at the beginning of the line, probably indicating beginning of line in some menus:

- Clicquot Brut Champagne

We have removed it with expression at the beginning of the line(31 changes were done):

```
value.replace(/^-/ , " ")
```

and at the end of the line (1 change done):

```
value.replace(/-$/, " ")
```

		30
--	--	----

--	--	--

5. To make sure, that we are not missing any wrong symbols at the beginning and end of the line, we have made trimming of trailing and head whitespaces. 57 changes were done.
6. As we have seen some “/” in the text, which served as brackets for some words, we have trimmed it in a way, not to disturb the fraction numbers, like “1/2”. 342 changes made.

`value.replace(/\D\D/, " ")`

7. We have noticed, that there are still non-meaningful brackets like “{}”, “[ ]”, which can be trimmed. Example:

`"Shanley's",Ny,"Shanley's","Blue Point Ronk[?] Roast On Toast",0.4,0.4,0.4,""`

We have trimmed it with following expression (196 changes done):

`value.replace(/\}/, " ").replace(/\{/, " ").replace(/\[/, " ").replace(/\]/, " ")`

8. We found out that there are still records with (to Order), which were missed at initial cleaning as they contained the word “order” in the Title case. Example:

`(to Order) Green Turtle Soup`

We have trimmed it with (4 changes done):

`value.replace(/\(to Order\)/, " ")`

9. Then, we have again for more effective cleaning trimmed head and trailing whitespaces (145 changes done), and consecutive (246 changes done) whitespaces.
10. We have defined that some starting single and double quotes, followed by space were meaningless, so we have trimmed it. Example:

`" A La Bearnaise`

We have used following expression (53 changes done):

`value.replace(/^[^" /, " ")`

`value.replace(/^[^' /, " ")`

11. We have found 1 case of round brackets with space inside, which was not trimmed at initial cleaning, one case with 3 dots at the beginning, and couple of cases of presence of arrows, which we trimmed as well. Examples:

`<green Turtle Au Vin De Champagne`

`( )d Rabbit`

`...Roma`

We have used following expressions:

`value.replace(/\( \)/, " ") - 1 case replaced`

`value.replace(/^[</, " ").replace(/^[>/, " ") - 2 cases replaced`

`value.replace(/^[\.\\.\\. /, " ") - 1 case replaced`

		31
--	--	----

--	--	--

12. We found out, that after all manipulations, we can do some more clustering with key collision method, keying function=fingerprint:

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
2	49	<ul style="list-style-type: none"> <li>Lobster Croquettes (48 rows)</li> <li>lobster:-- croquettes (1 rows)</li> </ul>	<input type="checkbox"/>	Lobster Croquettes
2	50	<ul style="list-style-type: none"> <li>Broiled Live Lobster (49 rows)</li> <li>lobster:-- broiled Live (1 rows)</li> </ul>	<input type="checkbox"/>	Broiled Live Lobster

So, we were able to change 96 more values.

“location” column:

1. We have removed only square and curly brackets with (273 changes done):

`value.replace(/\*/, " ").replace(/\+/, " ").replace(/\//, " ")`

Now after the changes we have made, we can proceed with exploring results of the Use case 1. The data in general still have some issues, for example some “( )” over some words, or some spelling errors, but it will not be a big issue for us as we will still be able to make the count of the distinct dishes and locations, and present the results even in case if in the dish or location will be some spelling error.

## Data Cleaning Using SQLite

To produce a cleaned and joint dataset to support our target use case  $U_1$ , we decided to use SQLite given that (1) the four NYPL Menus data tables we need to join have relations with each other through primary and foreign keys, (2) we can easily filter for the data we need for the joint dataset to support  $U_1$ , and (3) SQLite is lightweight and easy to user while performing adequately with the size of our dataset.

### a. Set Up Database

First, we set up the database in preparation for SQLite operations, starting with the initial step of importing the NYPL Menus data files (raw and cleaned versions of DishRaw.csv, MenuRaw.csv, DishClean.csv, MenuClean.csv, MenuItem.csv, MenuPage.csv), all stored as individual tables in the database using the .import command in SQLite. Using the .save command in SQLite, we save the data tables into a database to produce our NYPL\_Menus.db file. This step is necessary to create the working database for the subsequent queries we will use to not only create the joint dataset to support  $U_1$ , but also check for integrity constraint violations.



```

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Yitongs-MacBook-Pro:~ willtsai$ /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/sqlite-tools-osx-x8
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
[sqlite> .mode csv
[sqlite> .headers on
[sqlite> .import /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/DishClean.csv DishClean
[sqlite> .import /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/DishRaw.csv DishRaw
[sqlite> .import /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/MenuClean.csv MenuClean
[sqlite> .import /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/MenuRaw.csv MenuRaw
[sqlite> .import /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/MenuItem.csv MenuItem
[sqlite> .import /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/MenuPage.csv MenuPage
[sqlite> .save /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/NYPL_Menus.db
[sqlite> █

```

## b. Produce Joint Dataset

Subsequently, in SQLite we proceed to create the joint dataset combining data from all four NYPL Menus tables. This step is needed to produce the dataset required to support our target use case ( $U_1$ ). This was done by running a query to join the Dish, Menu, MenuItem, and MenuPage tables to create the joint dataset containing the data needed to support  $U_1$ ; using the following key relationships:

```

Menu.id (PK) -> MenuPage.menu_id (FK)
MenuPage.id (PK) -> MenuItem.menu_page_id (FK)
MenuItem.dish_id (PK) -> Dish.id (FK)

```

The joint dataset is stored as new files named MenuAggClean.csv and MenuAggRaw.csv using the .once command, and then the data tables are added into our DB using the .import and .save commands. Note that the MenuAggClean.csv file is created by querying the cleaned DishClean and MenuClean tables while MenuAggRaw.csv is created by querying the raw DishRaw and MenuRaw tables. Thus, the queries vary slightly since only MenuClean contains the 'if\_NY' column, which was created during the OpenRefine stage of our data cleaning. Since  $U_1$  will only require the dish\_name, location, and mi\_price columns, only include these in addition to place, sponsor, dish\_lowest\_price, dish\_highest\_price, and mi\_high\_price in our joint dataset for supporting both  $U_1$  and integrity constraint checks. We created both MenuAggClean and MenuAggRaw tables so that we may evaluate the clean and raw data side-by-side as a part of our integrity constraint violation checks. Note that the joint dataset that is needed to support  $U_1$  is stored in the MenuAggClean table.

```

[sqlite> .once /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/MenuAggClean.csv
sqlite> SELECT
...>     m.location
...>     , m.place
...>     , m.sponsor
...>     , agg.dish_name
...>     , agg.dish_lowest_price
...>     , agg.dish_highest_price
...>     , agg.mi_price
...>     , agg.mi_high_price
...> FROM MenuClean m
...> JOIN (
...>     SELECT
...>         mp.id AS mp_id
...>         , mp.menu_id
...>         , md.mi_id
...>         , md.mi_dish_id
...>         , md.mi_price
...>         , md.mi_high_price
...>         , md.dish_name
...>         , md.dish_lowest_price
...>         , md.dish_highest_price
...>     FROM MenuPage mp
...>     JOIN (
...>         SELECT
...>             mi.menu_page_id
...>             , mi.id AS mi_id
...>             , mi.dish_id AS mi_dish_id
...>             , mi.price AS mi_price
...>             , mi.high_price AS mi_high_price
...>             , d.name AS dish_name
...>             , d.lowest_price AS dish_lowest_price
...>             , d.highest_price AS dish_highest_price
...>         FROM MenuItem mi
...>         JOIN DishClean d
...>         ON mi.dish_id = d.id
...>     ) md
...>     ON mp.id = md.menu_page_id
...> ) agg
...> ON m.id = agg.menu_id
...> WHERE
...>     agg.mi_price > 0
...>     AND m.location IS NOT NULL
...>     AND m.location != 'Unknown'
...>     AND m.if_NY = 'true';
[sqlite> .import /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/MenuAggClean.csv MenuAggClean
[sqlite> .save /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/NYPL_Menus.db

```

```

[sqlite> .once /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/MenuAggRaw.csv
sqlite> SELECT
...>     m.location
...>     , m.place
...>     , m.sponsor
...>     , agg.dish_name
...>     , agg.dish_lowest_price
...>     , agg.dish_highest_price
...>     , agg.mi_price
...>     , agg.mi_high_price
...> FROM MenuRaw m
...> JOIN (
...>     SELECT
...>         mp.id AS mp_id
...>         , mp.menu_id
...>         , md.mi_id
...>         , md.mi_dish_id
...>         , md.mi_price
...>         , md.mi_high_price
...>         , md.dish_name
...>         , md.dish_lowest_price
...>         , md.dish_highest_price
...>     FROM MenuPage mp
...>     JOIN (
...>         SELECT
...>             mi.menu_page_id
...>             , mi.id AS mi_id
...>             , mi.dish_id AS mi_dish_id
...>             , mi.price AS mi_price
...>             , mi.high_price AS mi_high_price
...>             , d.name AS dish_name
...>             , d.lowest_price AS dish_lowest_price
...>             , d.highest_price AS dish_highest_price
...>         FROM MenuItem mi
...>         JOIN DishRaw d
...>         ON mi.dish_id = d.id
...>     ) md
...>     ON mp.id = md.menu_page_id
...> ) agg
...> ON m.id = agg.menu_id
...> WHERE
...>     agg.mi_price > 0
[ ...>     AND m.location IS NOT NULL;
[sqlite> .import /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/MenuAggRaw.csv MenuAggRaw
[sqlite> _save /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/NYPL_Menus/NYPL_Menus.db

```

## 2. Data Quality Changes

Please see a brief summary below of how many changes we have done to the cleaned columns:

Column Name	Number of Cells Changed	Brief Description of Changes
Menu.sponsor	207	Trimming leading and trailing whitespaces, consecutive spaces
Menu.sponsor	5440	Clustering of cells using key collision method
Menu.sponsor	3347	Clustering of cells using nearest neighbour method
Menu.sponsor	8212	Changing to the Title case
Menu.location	640	Trimming leading and trailing whitespaces, consecutive spaces
Menu.location	1281	Changing to the Title case
Menu.location	4209	Clustering of cells using key collision method
Menu.location	3560	Clustering of cells using nearest neighbour

		method
Menu.place	1420	Removing "[", ",", "?"
Menu.place	1717	Trimming leading and trailing whitespaces, consecutive spaces
Menu.place	7337	Changing to the Title case
Menu.place	2632	Clustering of cells using key collision method
Menu.place	2501	Clustering of cells using nearest neighbour method
Menu.location	166	Removing "[", ",", "?"
Menu.sponsor	176	Removing "[", ",", "?"
Menu.place	8123	Creating NY indicator column
Menu.location	17547	Creating NY indicator column
Menu.sponsor	15986	Creating NY indicator column
if_NY	17547	Creating "if_NY" indicator column
Dish.name	36052	Trimming leading and trailing whitespaces, consecutive spaces
Dish.name	284628	Changing to the Title case
Dish.name	93247	Clustering of cells using key collision method
Dish.name	13408	Removing ":", ";", and "," at the end of the line, removing single quotes "'", and "&" at the beginning of the line, ":", "-" at beginning and end of line
Dish.name	122	Removing (to order)
Dish.name	7028	Remove from text of the dish prices separated by commas and double digit meal numbers
Dish.name	37	Remove non-meaningful values (weird signs, dates)
Second-step Dish.name cleaning	1138	Removing different meaningless symbols and further trimming spaces.

Second-step "location" cleaning	273	Removing square and curly brackets

## Integrity Constraint Violation Checks

We devised a series of Integrity Constraint Violation (ICV) SQL queries which we have run against our database set up in SQLite. Checking our cleaned joint dataset against our ICV queries gives us confidence that the data cleaning we have performed is sufficient to support  $U_1$ . Below are detailed explanations of each ICV check we performed.

ICV\_1 -- We check to confirm the removal of non-value-added '[ ]', '{ }', '""', '()', ' ( )', or '?' characters in the location column and observe that cleaned values no longer contain these characters:

```
sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggClean
...> WHERE
...>     location LIKE '%[]%'
...>     OR location LIKE '%{}%'
...>     OR location LIKE '%""%'
...>     OR location LIKE '%?%'
...>     OR location LIKE '%()%'
[ ...>     OR location LIKE '%( )%';
COUNT(mi_price)
0
sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggRaw
...> WHERE
...>     location LIKE '%[]%'
...>     OR location LIKE '%{}%'
...>     OR location LIKE '%""%'
...>     OR location LIKE '%?%'
...>     OR location LIKE '%()%'
[ ...>     OR location LIKE '%( )%';
COUNT(mi_price)
2262
```

ICV\_2 -- We check to confirm the removal of non-value-added '[ ]', '{ }', '""', '()', ' ( )', or '?' characters in the dish\_name column and observe that cleaned values no longer contain these characters:

```
sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggClean
...> WHERE
...>     dish_name LIKE '%[]%'
...>     OR dish_name LIKE '%{}%'
...>     OR dish_name LIKE '%""%'
...>     OR dish_name LIKE '%?%'
...>     OR dish_name LIKE '%()%'
[ ...>     OR dish_name LIKE '%( )%';
COUNT(mi_price)
0
sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggRaw
...> WHERE
...>     dish_name LIKE '%[]%'
...>     OR dish_name LIKE '%{}%'
...>     OR dish_name LIKE '%""%'
...>     OR dish_name LIKE '%?%'
...>     OR dish_name LIKE '%()%'
[ ...>     OR dish_name LIKE '%( )%';
COUNT(mi_price)
239
```

ICV\_3 -- We check to confirm removal of leading, trailing, and double spaces in the location column and observe that cleaned values no longer contain unnecessary spaces:

```

sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggClean
...> WHERE
...>     location LIKE '% '
...>     OR location LIKE ' %'
[ ...>     OR location LIKE '% %';
COUNT(mi_price)
0
sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggRaw
...> WHERE
...>     location LIKE '% '
...>     OR location LIKE ' %'
[ ...>     OR location LIKE '% %';
COUNT(mi_price)
7416

```

ICV\_4 -- We check to confirm removal of leading, trailing, and double spaces in the dish\_name column and observe that cleaned values no longer contain unnecessary spaces:

```

sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggClean
...> WHERE
...>     dish_name LIKE '% '
...>     OR dish_name LIKE ' %'
[ ...>     OR dish_name LIKE '% %';
COUNT(mi_price)
0
sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggRaw
...> WHERE
...>     dish_name LIKE '% '
...>     OR dish_name LIKE ' %'
[ ...>     OR dish_name LIKE '% %';
COUNT(mi_price)
14467

```

ICV\_5 -- We check to confirm removal of leading ellipses, greater than, less than, as well as leading or trailing comma, colon, semi-colon, dash, asterisk, or ampersand characters in the location column and observe that cleaned values no longer contain these characters:

<pre> sqlite&gt; SELECT COUNT(mi_price) ...&gt; FROM MenuAggClean ...&gt; WHERE ...&gt;     location LIKE '...%' ...&gt;     OR location LIKE '&lt;%' ...&gt;     OR location LIKE '&gt;%' ...&gt;     OR location LIKE ',%' ...&gt;     OR location LIKE '%,' ...&gt;     OR location LIKE ':%' ...&gt;     OR location LIKE '%:' ...&gt;     OR location LIKE ';%' ...&gt;     OR location LIKE '%;' ...&gt;     OR location LIKE '-%' ...&gt;     OR location LIKE '%-' ...&gt;     OR location LIKE '*%' ...&gt;     OR location LIKE '%*' ...&gt;     OR location LIKE '&amp;%' [ ...&gt;     OR location LIKE '%&amp;'; COUNT(mi_price) 0 </pre>	<pre> sqlite&gt; SELECT COUNT(mi_price) ...&gt; FROM MenuAggRaw ...&gt; WHERE ...&gt;     location LIKE '...%' ...&gt;     OR location LIKE '&lt;%' ...&gt;     OR location LIKE '&gt;%' ...&gt;     OR location LIKE ',%' ...&gt;     OR location LIKE '%,' ...&gt;     OR location LIKE ':%' ...&gt;     OR location LIKE '%:' ...&gt;     OR location LIKE ';%' ...&gt;     OR location LIKE '%;' ...&gt;     OR location LIKE '-%' ...&gt;     OR location LIKE '%-' ...&gt;     OR location LIKE '*%' ...&gt;     OR location LIKE '%*' ...&gt;     OR location LIKE '&amp;%' [ ...&gt;     OR location LIKE '%&amp;'; COUNT(mi_price) 1734 </pre>
--	---

--	--	--

ICV\_6 -- We check to confirm removal of leading period, greater than, less than, as well as leading or trailing comma, colon, semi-colon, dash, asterisk, or ampersand characters in the dish\_name column and observe that cleaned values no longer contain these characters:

```
sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggClean
...> WHERE
...>   dish_name LIKE '...%'
...>   OR dish_name LIKE '<%'
...>   OR dish_name LIKE '>%'
...>   OR dish_name LIKE ',%'
...>   OR dish_name LIKE '%,'
...>   OR dish_name LIKE ':%'
...>   OR dish_name LIKE '%:'
...>   OR dish_name LIKE ';%'
...>   OR dish_name LIKE '%;'
...>   OR dish_name LIKE '-%'
...>   OR dish_name LIKE '%-'
...>   OR dish_name LIKE '*%'
...>   OR dish_name LIKE '%*'
...>   OR dish_name LIKE '&%'
...>   OR dish_name LIKE '%&';
COUNT(mi_price)
0

sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggRaw
...> WHERE
...>   dish_name LIKE '...%'
...>   OR dish_name LIKE '<%'
...>   OR dish_name LIKE '>%'
...>   OR dish_name LIKE ',%'
...>   OR dish_name LIKE '%,'
...>   OR dish_name LIKE ':%'
...>   OR dish_name LIKE '%:'
...>   OR dish_name LIKE ';%'
...>   OR dish_name LIKE '%;'
...>   OR dish_name LIKE '-%'
...>   OR dish_name LIKE '%-'
...>   OR dish_name LIKE '*%'
...>   OR dish_name LIKE '%*'
...>   OR dish_name LIKE '&%'
...>   OR dish_name LIKE '%&';
COUNT(mi_price)
2580
```

ICV\_7 -- We check to confirm that our clustering efforts in OpenRefine resulted in normalization of values in the location column to allow for accurate data aggregation and grouping to support  $U_1$ . We observe that our cleaned dataset contains 415 less distinct location values than the raw dataset, indicating that our clustering efforts yielded intended results:

```
sqlite> SELECT COUNT(DISTINCT location)
[ ...> FROM MenuClean;
"COUNT(DISTINCT location)"
5869
sqlite> SELECT COUNT(DISTINCT location)
[ ...> FROM MenuRaw;
"COUNT(DISTINCT location)"
6284
```

ICV\_8 -- We check to confirm that our clustering efforts in OpenRefine resulted in normalization of values in the dish\_name column to allow for accurate data aggregation and grouping to support  $U_1$ . We observe that our cleaned dataset contains 82,526 less distinct dish\_name values than the raw dataset, indicating that our clustering efforts yielded intended results:

```
sqlite> SELECT COUNT(DISTINCT name)
[ ...> FROM DishClean;
COUNT(DISTINCT name)
345526
sqlite> SELECT COUNT(DISTINCT name)
[ ...> FROM DishRaw;
COUNT(DISTINCT name)
428052
```

ICV\_9 -- We check to confirm removal of invalid values in the location column where location or restaurant name is indicated as not provided. We observe that our cleaned dataset does not contain entries where the location is not provided:

--	--	--

```

sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggClean
...> WHERE
...>     location LIKE '%Not Given%'
...>     OR location LIKE '%not given%'
...>     OR location LIKE '%Not given%'
...>     OR location LIKE '%not Given%'
[ ...>     OR location = 'Unknown';
COUNT(mi_price)
0
sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggRaw
...> WHERE
...>     location LIKE '%Not Given%'
...>     OR location LIKE '%not given%'
...>     OR location LIKE '%Not given%'
...>     OR location LIKE '%not Given%'
[ ...>     OR location = 'Unknown';
COUNT(mi_price)
4845

```

ICV\_10 -- We check to confirm removal of non-value added content in dish\_name values, specifically '(to order)' and variants. We observe that cleaned values no longer contain these characters:

```

sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggClean
...> WHERE
...>     dish_name LIKE '%(to order)%'
...>     OR dish_name LIKE '%(To Order)%'
...>     OR dish_name LIKE '%(To order)%'
...>     OR dish_name LIKE '%(to Order)%'
[ ...>     OR dish_name LIKE '%(TO ORDER)%';
COUNT(mi_price)
0
sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggRaw
...> WHERE
...>     dish_name LIKE '%(to order)%'
...>     OR dish_name LIKE '%(To Order)%'
...>     OR dish_name LIKE '%(To order)%'
...>     OR dish_name LIKE '%(to Order)%'
[ ...>     OR dish_name LIKE '%(TO ORDER)%';
COUNT(mi_price)
197

```

ICV\_11 -- We check to confirm that the newly added MenuClean.if\_NY column includes only boolean TRUE/FALSE values to ensure our joint dataset contains the necessary data needed for  $U_1$ , and observe that this is indeed the case:

```

sqlite> SELECT DISTINCT if_NY
[ ...> FROM MenuClean;
if_NY
false
true

```

ICV\_12 -- We check to confirm that entries with invalid or null data are filtered out of our cleaned joint dataset, and observe that this is indeed the case:



--	--	--

```

sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggClean
...> WHERE
...>     location IS NULL
...>     OR location = 'Unknown'
...>     OR dish_name IS NULL
...>     OR mi_price <= 0;
COUNT(mi_price)
0
sqlite> SELECT COUNT(mi_price)
...> FROM MenuAggRaw
...> WHERE
...>     location IS NULL
...>     OR location = 'Unknown'
...>     OR dish_name IS NULL
...>     OR mi_price <= 0;
COUNT(mi_price)
673

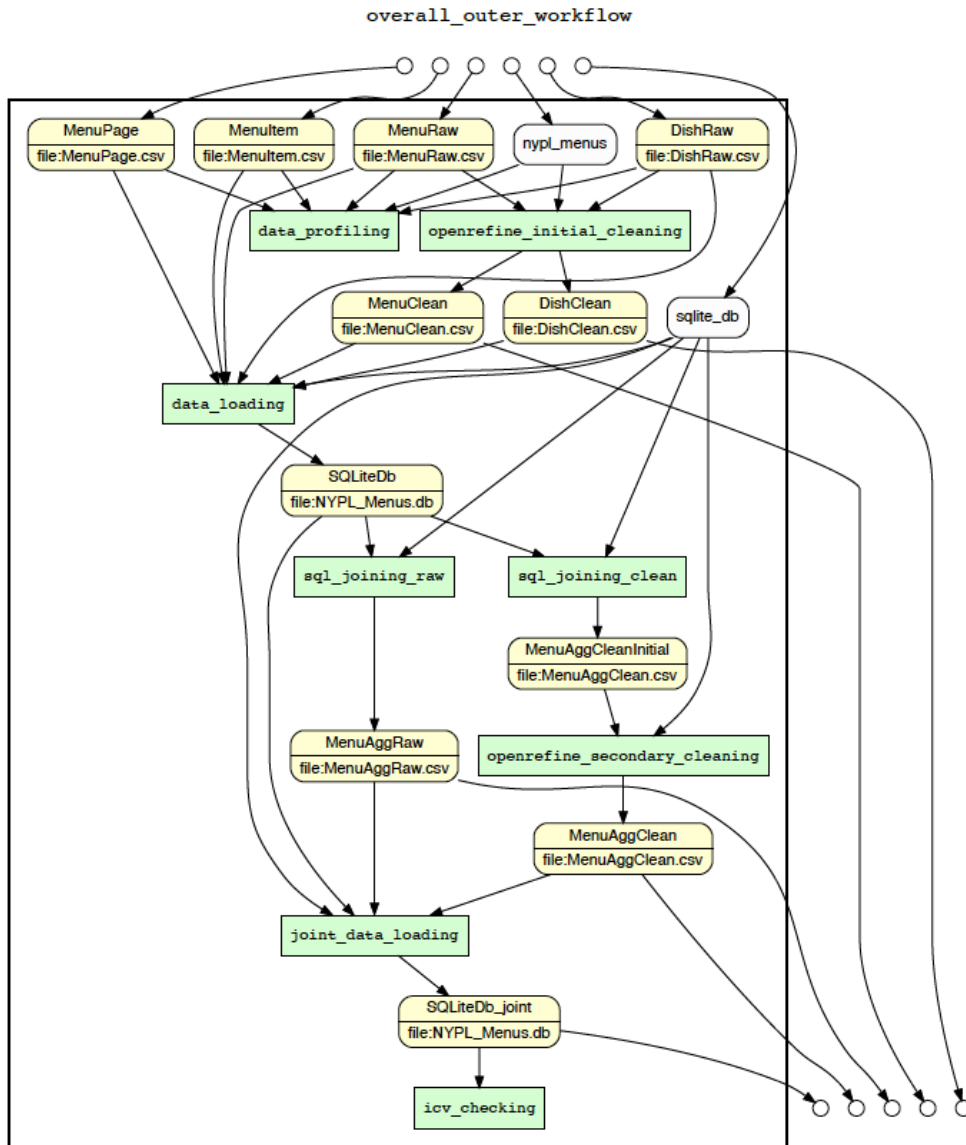
```

### 3. Workflow Models

#### Overall “Outer” Workflow ( $W_o$ )

Despite having not used Python or other scripting languages for our project, we decided to create our Overall “Outer” Workflow ( $W_o$ ) using the YesWorkflow tool, given the ease of use thanks to its automated graph rendering capabilities. We constructed a set of annotations stored in a .yw file, from which we generated the workflow diagrams utilizing the YesWorkflow command line tool. The overall workflow contains seven different processes, starting with data profiling, then initial OpenRefine cleaning before data uploading into a SQLite Db where data joining is completed to produce the joint dataset needed for  $U_1$ . A secondary round of OpenRefine cleaning is performed against the joint dataset to correct errors discovered after joining. The joint dataset is then loaded back into the SQLite Db and subsequently checked for integrity constraint violations to confirm that the data is sufficiently cleaned to support  $U_1$ . The Four original NYPL Menu data tables (Dish, Menu, MenuPage, MenuItem) are inputs to our overall workflow. Four cleaned new data tables are produced as a result of our overall workflow - these are all required for  $U_1$ : DishClean, MenuClean, MenuAggClean, and MenuAggRaw. NYPL\_Menus.db is also produced from our workflow - this is the working Db from which we run SQL queries for data cleaning, joining, and integrity constraint violation checking.

--	--	--



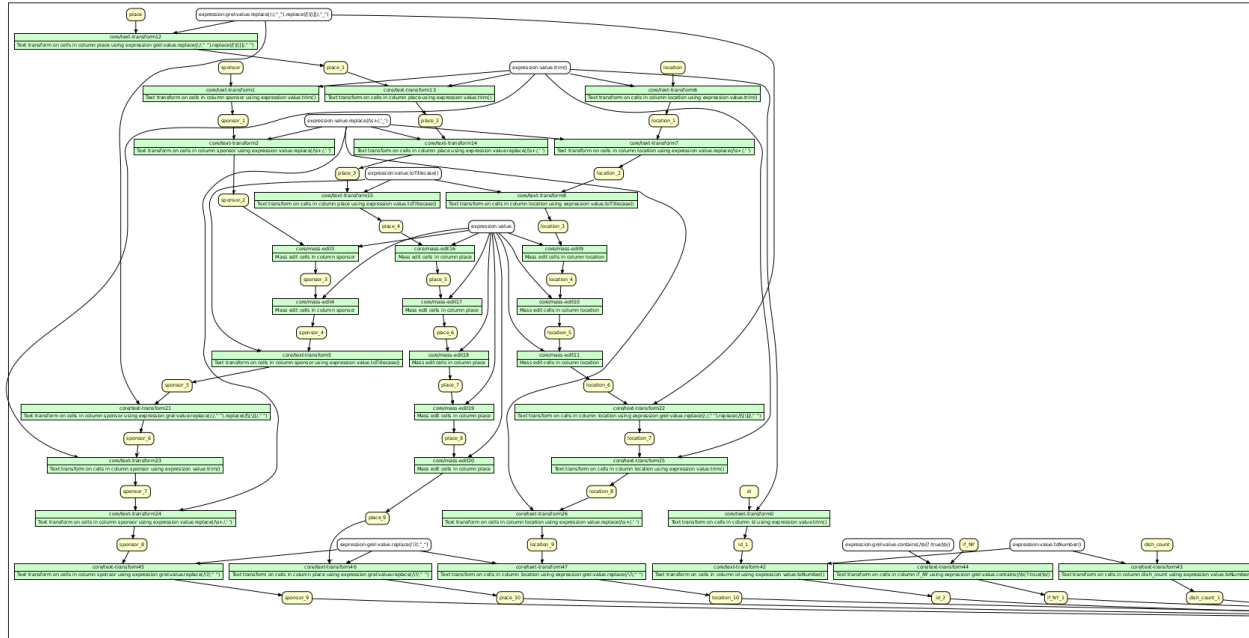
## Data Cleaning “Inner” Workflow (W<sub>i</sub>)

For inner workflow we have used or2yw tool, and it was used to combine the activities which we have done for cleaning of Dish and Menu tables in OpenRefine tool.

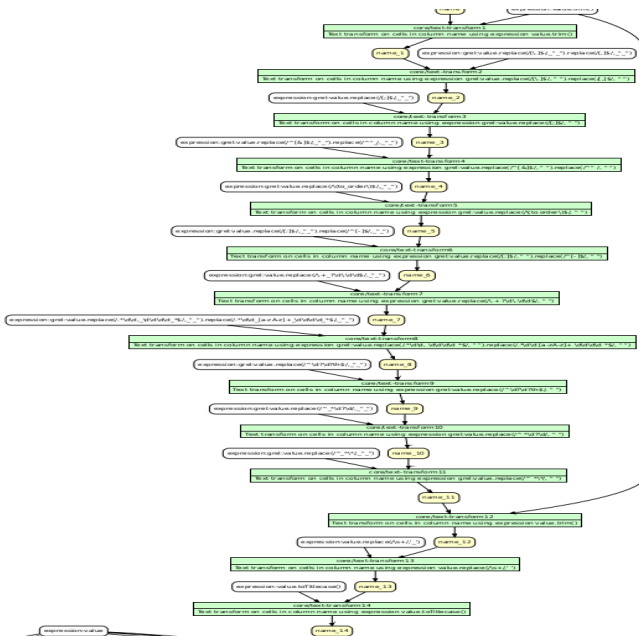
First, we have cleaned “Menu.csv” file, and the following workflow represent all the operations, used in it. It is depicted as “openrefine\_cleaning” square in the overall workflow. The workflow document is called “menu\_workflow.pdf” and stored in the project folder.

The document, which depicts the workflow of cleaning “Dish.csv” is called “dish\_workflow.pdf” and also stored in the project folder.

At the end, when we have completed the first integrity constraints check, and made secondary cleaning, we have made another workflow, called “joined\_file\_workflow.pdf”. Please see the example screenshot of the “menu\_workflow.pdf” workflow:



Here is the screenshot of part of “dish\_workflow.pdf”:



## 4. Conclusions and Summary

### Conclusion

We executed our data cleaning activities in OpenRefine and SQLite, which both proved sufficient and performant for tasks required to clean the data in preparation for  $U_1$  given the size of the dataset and operations needed. SQLite was also ideal for checking integrity constraint violations given that our cleaned datasets could be easily queried from the same database used to store the tables we created. Finally, we utilized the YesWorkflow tool to create our overall and data cleaning workflow artifacts, given the built-in graph generating capabilities of the tool making it easy to use. Overall, we cleaned five columns in the raw data, changing 520,434 cells while also adding an `if_NY` column to the Menu table with 17,547 incremental cells. We produced two cleaned versions of the data tables (MenuClean and DishClean) and also created the joint dataset (MenuAggClean and MenuAggRaw) with data from all four NYPL Menus tables needed to support  $U_1$ .

### Producing Data for the Target Use Case ( $U_1$ )

We also built a query against our joint dataset in SQLite to implement the target use case. The output of the query is stored in a .csv file which we used to visualize the data in MS-Excel.

```
[sqlite> .mode csv
[sqlite> .once /Users/willtsai/Documents/School/IllinoisMCS/CS513_DataCleaning/U1_data.csv
sqlite> SELECT
...>     dish_name
...>     , COUNT(DISTINCT location)
...>     , AVG(mi_price)
...> FROM MenuAggClean
...> GROUP BY dish_name;
```

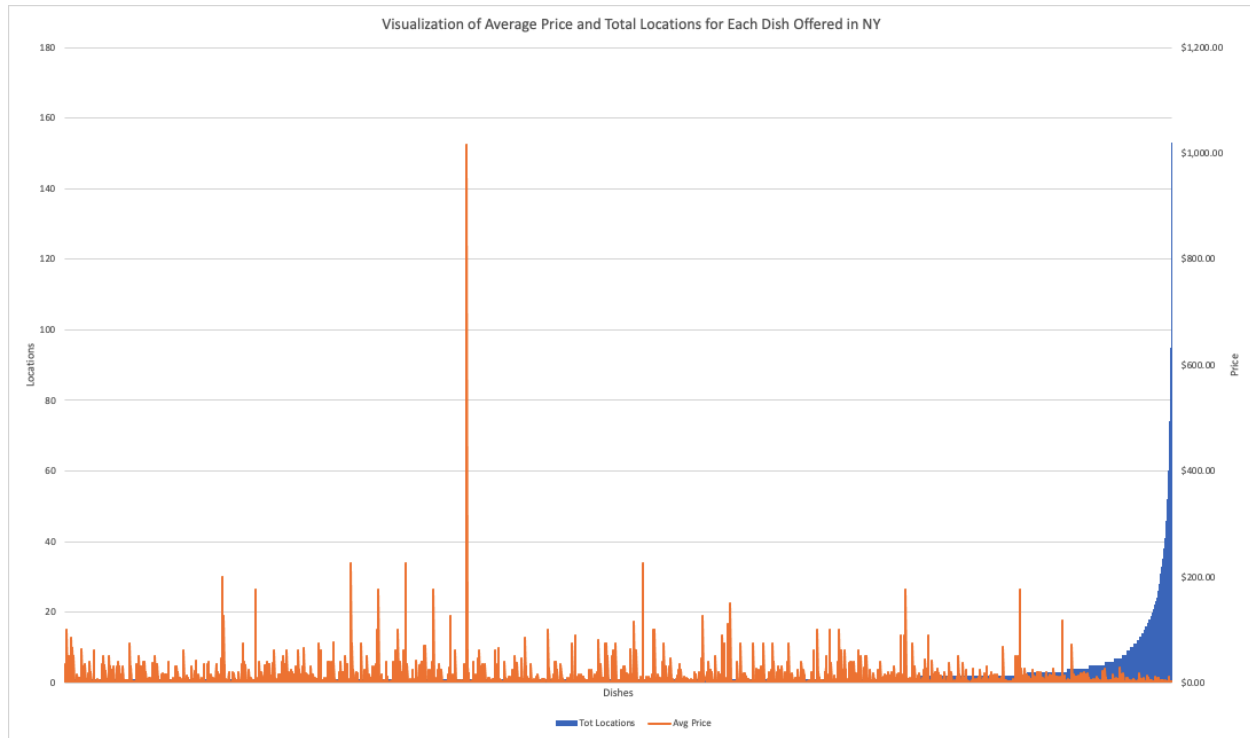
After loading the data into MS-Excel, we utilized the PivotChart functionality to visualize the data, which revealed some interesting findings:

1. Chicken Salad was the most offered dish by locations in the state of NY
2. There is a dish called “Cream Cheese With Bar-le-duc Jelly” that is clearly the outlier for highest average price at \$1,017. Since we did not have a way to verify whether this price is accurate (it would very well be a royal dish, we did not correct this high price value in our data cleaning.

#### Top 10 Most Offered Dishes by Locations in the state of NY:

Dish Name	Tot Locations	Avg Price
Chicken Salad	153	\$1.48
Lobster Salad	141	\$1.49
Stewed Tomatoes	140	\$0.48
Potato Salad	131	\$0.42
Stewed Prunes	129	\$0.34
French Peas	122	\$0.75
Celery	117	\$0.92
Milk Toast	116	\$0.52
Ham And Eggs	113	\$0.82
Potatoes Lyonnaise	112	\$0.28

### Visualization of Average Price and Total Locations for Each Dish Offered in NY:



## Lessons Learned

We had following Lessons Learned from this project:

1. We have figured out, that in data cleaning the sequence of actions plays a very big role. For example, if you are choosing to transform your text strings to Title case prior to trimming any special characters, you will get many wrongly cased words (due to any word after a special character goes from small letter). Or, if you will not trim the spaces before searching for the special characters at the beginning and end of the line, regex can miss it and leave the lines uncleaned. So, during the cleaning strategy development such details should be accounted for.
2. We have found out, that for the target use case it is not necessary to clean 100% of issues in the target columns and the cleaning should be really done in the sense of need to fit the use case. We have experienced the situation, when having lots of weird values in the columns "dish" and "location" after the first cleaning iteration we have received an almost clean table after joins. The reason why the values like "10th", "0846" etc were left in the Dish table after the first cleaning was that it was not clear how to filter it from legit values, like year (4 digit), or indication of content quantity in some dishes. But as we have seen in the final joined table - they were in fact not present in any menu, so it was good that we had economy time, not trimming it.
3. There can be several stages in the cleaning workflow, when sometimes after some integrity constraints checks we have to return to the same phase, which we have done already, as some items appeared to be missed out (like we did a secondary cleaning step with OpenRefine).

--	--	--

4. It is very important to document the process and results of the work, to make lessons learned at every next step and do improvement in the workflow already in the second and following steps, in case it will not be possible to revert or repeat the first step.

## Contributions from Each Team Member

The contributions from each team member is summarized in the table below:

Task(s)	Team Member(s)
Dataset Identification	Marina Polupanova & Wei-Lun Tsai
Data Profiling and Description	Marina Polupanova
Data Quality Problems Identification	Marina Polupanova
Use Case Development and Initial Plan	Wei-Lun Tsai
OpenRefine Data Cleaning	Marina Polupanova
SQLite Data Cleaning and Integrity Constraints	Wei-Lun Tsai
Outer Workflow Model Development and Documentation	Wei-Lun Tsai
Inner Workflow Model Development and Documentation	Marina Polupanova
Phase I & II Report Authoring	Marina Polupanova & Wei-Lun Tsai

## 5. Supplemental Materials

You may find our supplemental materials in this [ZIP file](#)<sup>3</sup>.

You may find our Raw and Cleaned Datasets in this [Box Folder Link](#)<sup>4</sup>.

<sup>3</sup> <https://uofi.box.com/s/8ibirxp1hqjwty2o073fjdi8gf1p4xt0>

<sup>4</sup> <https://uofi.box.com/s/88z01soh7s1ry058sazx6gz735uzylpi>