CS513: Data Cleaning Project

Final Report

Shreya Reddy Peesary, Gokul Krishnamoorthy and Ameet Deulgaonkar
peesary2@illinois.edu, gokulk2@illinois.edu, ameetd2@illinois.edu
University of Illinois, Urbana-Champaign

Note: Regarding revaluation of Phase-1 report
Points received for initial Phase-1 submission : 98 (out of 100)
Our team chooses option: (A) No change to Phase-1 report

1. Overview

As part of the CS513 Theory and Practise of Data Cleaning final project in the summer of 2021, we performed a data cleaning exercise on a real world dataset, the New York Public Library's restaurant menu collection[1] to put into practise the theory and tools we learnt in the course. Our work was divided into 2 phases, in the first phase we did a detailed data quality assessment and reported our findings, in the second phase we performed the actual data cleaning exercise. This final report combines our findings from both the phases.

The document is organized on the following lines:

- Sections 2-5 are related to the work done in phase-1 of the project (they are the same as in the phase-1 report).
- Sections 6-9 are related to our work in the phase-2 of the project.

Specifically, Section 2 describes the dataset we will be working with and the data dictionary, and Section 3 describes the primary use case(s) that we want the dataset to address (i.e our main motivation for data cleaning). We also describe use cases that do not require data cleaning and also provide use cases that are impossible to achieve with any amount of cleaning. We then make a strong case for data cleaning by demonstrating the unsatisfactory results on the dirty data. The section 5 then provides details on the major data quality issues we have discovered and the remediations to achieve our use case.

Section 6 describes the actual data cleaning operations performed in phase-2 and their rationale and section 7 quantifies and articulates the resulting improvements to the quality of the data and subsequently to the results of our SQL queries. Section 8 describes our workflow model and finally we provide our conclusions and key learnings in section 9.

2. Dataset and Data Dictionary

For this project, we will be using the New York Public Library's restaurant menu collection. The collection is one of largest menu collections in the world. The origins of the collection date back to 1900 by Miss Frank E. Buttolph and as of today the total number stands at 45000 menus. There is an ongoing effort to manually transcribe the contents of menu images by volunteers through the "What's on the Menu?" project to facilitate various kinds of academic (like this project) and commercial research.

The dataset is made available as four different files in the `csv` format. The data objects contained in each of these files are described below. Note: we provide only a brief description of the fields below, for a more detailed description of each field please refer to the ER diagram and the Appendix A section.

Menu.csv

- Size: 3.1 MB
- Records: 17550
- Description: Each entry in this file represents a physical menu object. Fields contain information about the sponsor of the menu (e.g. restaurant, company/organization), event, occasion, number of items in the menu, date and physical size (length x Width) of the menu.

MenuPage.csv

- Size: 4.5 MB
- Records: 66938
- Description: Each row represents a physical page in the menu (to the image of the menu). Fields contain information about the digital image of the page along with page number and reference to the menu identifier it belongs to.

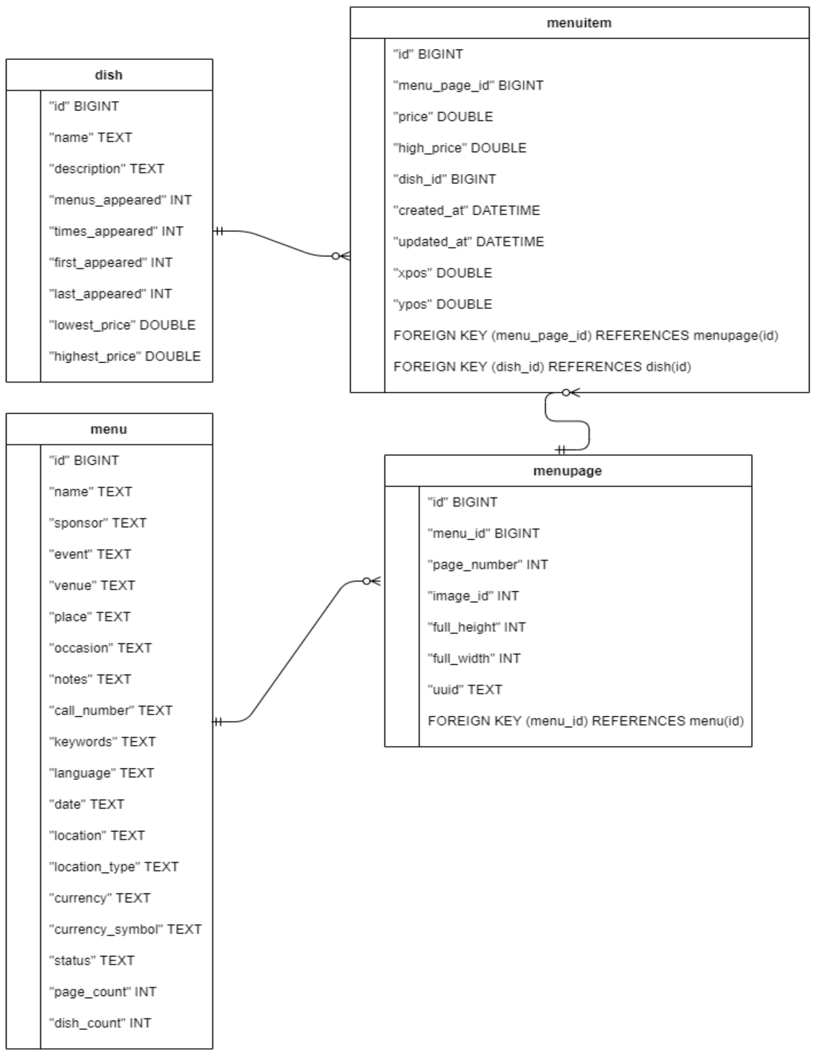
MenuItem.csv

- Size: 113 MB
- Records: 1334780
- Description: Each row represents an entry on a menu page. Fields contain price, position where it appears in the menu (xpos, ypos). Each row also references entries in MenuPage and Dish.

Dish.csv

- Size: 26M
- Records: 428086
- Description: Collection of all dishes across all menus. Each entry is assigned a unique identifier which does not necessarily represent a unique dish across all menus. Fields include name of the disk, number of menus it appeared in, number of times it appeared, maximum and minimum prices for the dish.

ER diagram for the Menu dataset



3. Use Case(s)

3.1. Primary use case(s):

a. Find popular dishes (i.e. top N dishes) by various dimensions. For e.g. find top 5 popular dishes served in various events (i.e. dinners, luncheons, etc.). Similarly, find top dishes served on various occasions (anniversaries, birthdays, etc.).

The SQL query that helps answer the above question (for “event” dimension) is as follows. We can find answers to other questions by replacing “event” with other dimension names.

```
SELECT *
```

```
FROM   (SELECT menu.event,
               dish.NAME,
               Count(dish.NAME)AS "count",
               Row_number()OVER (
               partition BY menu.event
               ORDER BY Count(dish.NAME) DESC) AS
dish_rank
        FROM   menu,
               menupage,
               menuitem,
               dish
        WHERE  menu.id = menupage.menu_id
               AND menupage.id =
menuitem.menu_page_id
               AND menuitem.dish_id = dish.id
        GROUP  BY menu.event,
                  dish.NAME
        HAVING Count(dish.NAME) > 50) AS event_dishes
WHERE  dish_rank <= 5
```

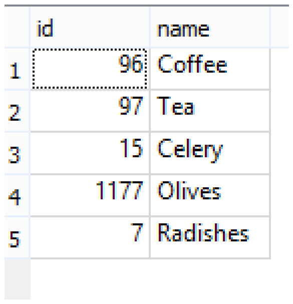
b. Find popular dishes (i.e. top N dishes) in each decade.

```
SELECT *
FROM   (SELECT Substr(Strftime("%y", date), 1, 3)
               || '0s' AS decade,
               dish.NAME,
               Count(dish.NAME) AS count,
               Row_number()
                 OVER (
       partition BY Substr(Strftime("%y", date), 1,
3)
       ORDER BY Count(dish.NAME) DESC) AS dish_rank
        FROM   menu,
               menupage,
               menuitem,
               dish
        WHERE  menu.id = menupage.menu_id
               AND menupage.id =
menuitem.menu_page_id
               AND menuitem.dish_id = dish.id
        GROUP  BY decade,
                  dish.id
        ORDER  BY decade,
                  dish.id,
                  count)
WHERE  dish_rank <= 5
```

3.2. Use case(s) that do not require data cleaning?

a. Find all time top 5 dishes (i.e across menus without any grouping by dimension).

```
SELECT id,
       name
FROM   dish
ORDER  BY times_appeared DESC
LIMIT  5;
```



	id	name
1	96	Coffee
2	97	Tea
3	15	Celery
4	1177	Olives
5	7	Radishes

b. Find top #5 largest menus by dish count or by number of menu pages.

```
SELECT *
FROM   menu
ORDER  BY dish_count DESC
LIMIT  5;

SELECT *
FROM   menu
ORDER  BY page_count DESC
LIMIT  5;
```

3.3. Impossible use case(s)?

a. Find popular dishes by city. Not enough information in the dataset.

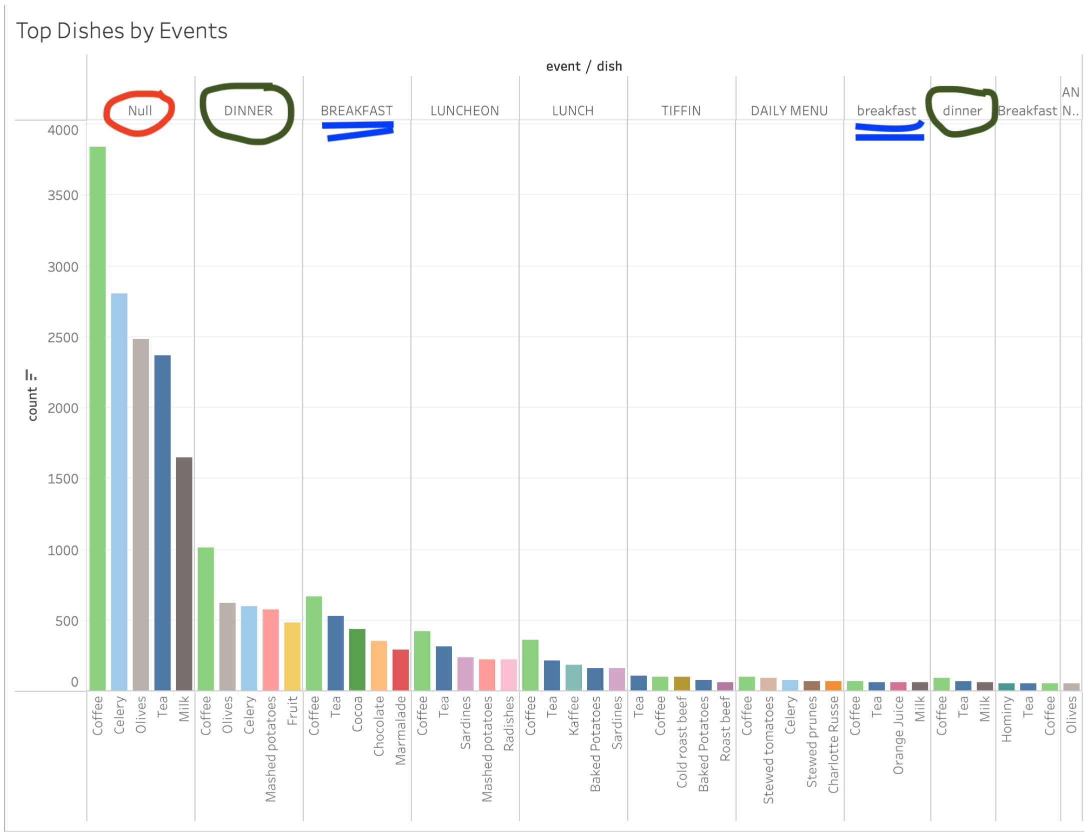
The information about the city where the menu is hosted appears to be somewhat haphazardly captured in the `place` attribute in `Menu.csv`. But, no amount of standardization or transformation of this attribute will lead us to meaningful or conclusive results about the popular dishes in a city.

b. There is no way to extract information about the reviews about the menus, so we cannot know the popularity of the menu by using techniques like sentiment analysis.

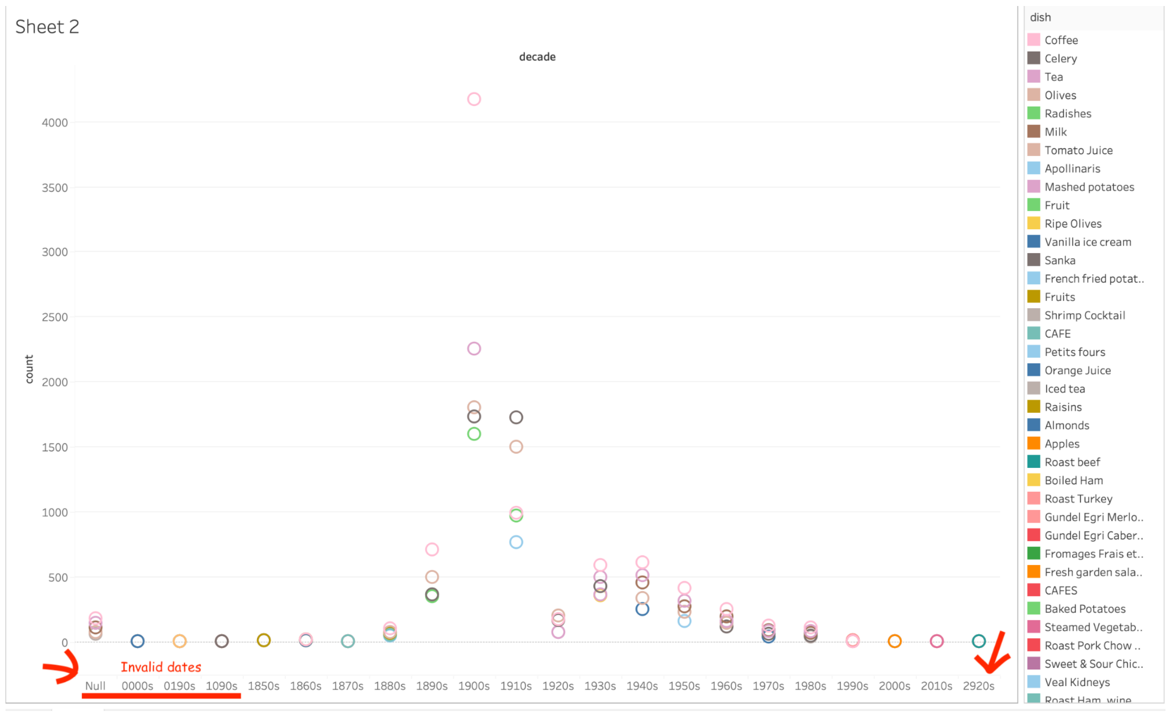
4. Motivation

Following is a Tableau charts for the SQL query results for use case #1 and use case #2 on the "dirty" data. We can see that the results are not very intuitive and do not provide a clear answer to our question. For e.g. for the use case #1, we see events with `null` value, we also see the multiple entries for "breakfast" and "lunch" etc, suggesting that this dataset needs curation before we can put it to use. Similar for use case #2 We see `null` values for dates and invalid dates.

Use Case #1 with dirty data (popular dishes by event)



Use Case #2 with dirty data (popular dishes by decades)



5. Data Profiling Results

5.1. Syntactic Issues

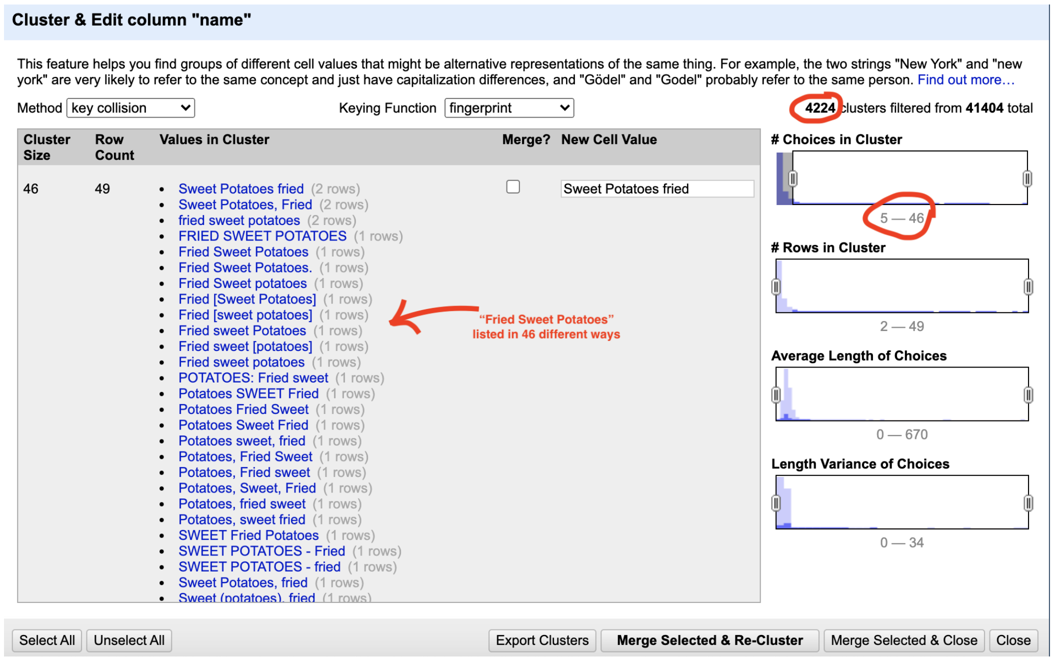
Dish.csv

Tool Used: Open Refine

Column: `name`

Issues:

1. Lots of inconsistencies in naming dishes. Clustering on the `name` column showed that there are potentially <u>4224</u> dishes that are listed in at least <u>5 different</u> ways. This will result in improper aggregations leading to wrong conclusions about popular dishes as seen in the motivation section.



Cluster & Edit column "name"

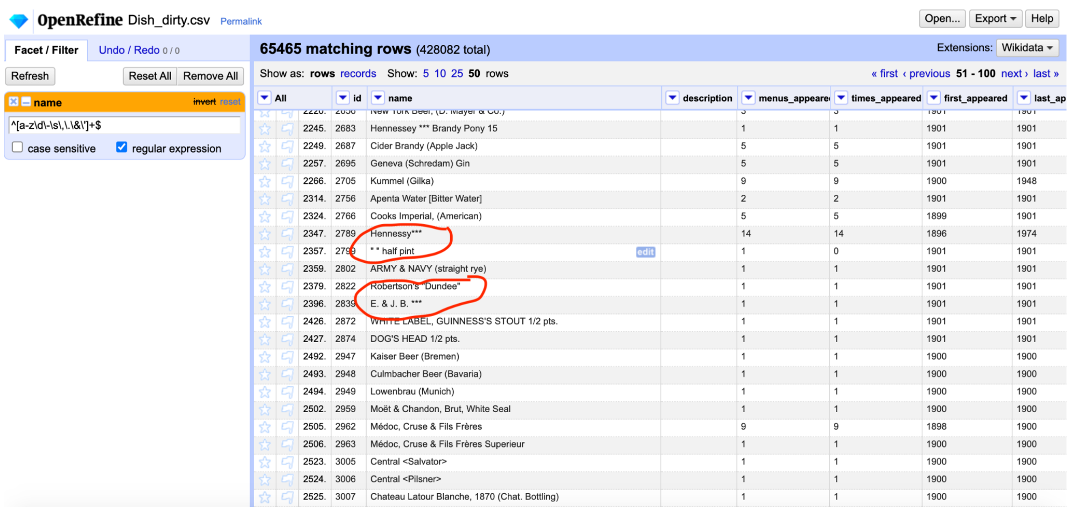
This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same person. Find out more…

Method: key collision — Keying Function: fingerprint — 4224 clusters filtered from 41404 total

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
46	49	Sweet Potatoes fried (2 rows) Sweet Potatoes, Fried (2 rows) fried sweet potatoes (2 rows) FRIED SWEET POTATOES (1 rows) Fried Sweet Potatoes (1 rows) Fried Sweet Potatoes. (1 rows) Fried Sweet potatoes (1 rows) Fried [Sweet Potatoes] (1 rows) Fried [sweet potatoes] (1 rows) Fried sweet Potatoes (1 rows) Fried sweet [potatoes] (1 rows) Fried sweet potatoes (1 rows) POTATOES: Fried sweet (1 rows) Potatoes SWEET Fried (1 rows) Potatoes Fried Sweet (1 rows) Potatoes Sweet Fried (1 rows) Potatoes sweet, fried (1 rows) Potatoes, Fried Sweet (1 rows) Potatoes, Fried sweet (1 rows) Potatoes, Sweet, Fried (1 rows) Potatoes, fried sweet (1 rows) Potatoes, sweet fried (1 rows) SWEET Fried Potatoes (1 rows) SWEET POTATOES - Fried (1 rows) SWEET POTATOES - fried (1 rows) Sweet Potatoes, fried (1 rows) Sweet (potatoes), fried (1 rows)	☐	Sweet Potatoes fried

"Fried Sweet Potatoes" listed in 46 different ways

2. Dish names contains special characters ([*\?\#\$\:\^\"\>]).



OpenRefine Dish_dirty.csv

Facet / Filter: name — `^[a-z\d\-\s\,\.\&\']+$` — ☐ case sensitive ☑ regular expression

65465 matching rows (428082 total)

Show as: rows records Show: 5 10 25 50 rows — « first ‹ previous 51 - 100 next › last »

All	id	name	description	menus_appeare	times_appeared	first_appeared	last_ap
2245.	2683	Hennessey *** Brandy Pony 15		1	1	1901	1901
2249.	2687	Cider Brandy (Apple Jack)		5	5	1901	1901
2257.	2695	Geneva (Schredam) Gin		5	5	1901	1901
2266.	2705	Kummel (Gilka)		9	9	1900	1948
2314.	2756	Apenta Water [Bitter Water]		2	2	1901	1901
2324.	2766	Cooks Imperial, (American)		5	5	1899	1901
2347.	2789	Hennessy***		14	14	1896	1974
2357.	2799	" " half pint		1	0	1901	1901
2359.	2802	ARMY & NAVY (straight rye)		1	1	1901	1901
2379.	2822	Robertson's "Dundee"		1	1	1901	1901
2396.	2839	E. & J. B. ***		1	1	1901	1901
2426.	2872	WHITE LABEL, GUINNESS'S STOUT 1/2 pts.		1	1	1901	1901
2427.	2874	DOG'S HEAD 1/2 pts.		1	1	1901	1901
2492.	2947	Kaiser Beer (Bremen)		1	1	1900	1900
2493.	2948	Culmbacher Beer (Bavaria)		1	1	1900	1900
2494.	2949	Lowenbrau (Munich)		1	1	1900	1900
2502.	2959	Moët & Chandon, Brut, White Seal		1	1	1900	1900
2505.	2962	Médoc, Cruse & Fils Frères		9	9	1898	1900
2506.	2963	Médoc, Cruse & Fils Frères Superieur		1	1	1900	1900
2523.	3005	Central <Salvator>		1	1	1900	1900
2524.	3006	Central <Pilsner>		1	1	1900	1900
2525.	3007	Chateau Latour Blanche, 1870 (Chat. Bottling)		1	1	1900	1900

3. Corrupted dish names like "???" or names that contain only numerals. Aggregations on these do not provide any real value.

Recipe (tentative):

1. Perform <u>standardization</u> of dish names using the OpenRefine's "cluster and edit" feature. Apply custom text transformations using regular expressions to <u>normalize</u> all dish names.
   a. Remove special characters from dish names [*\?\#\$\:\^\"\>]
   b. Remove "[ ]" and the enclosing text.
   c. Remove "()" and the enclosing text.
   d. Change to title case.

e. Remove leading, trailing and consecutive white spaces.

Challenges:

1. OpenRefine performance is really poor in clustering such a large number of records in `Dish.csv`. The clustering somehow works, but the mass edit does not complete in one go. Need to find a solution to edit and merge the clusters (either using a beefy server or performing merges in batches).
2. It is impossible to inspect and verify "standardized" names for all the clusters (at least with given time), so we will blindly accept OpenRefine's recommendation.

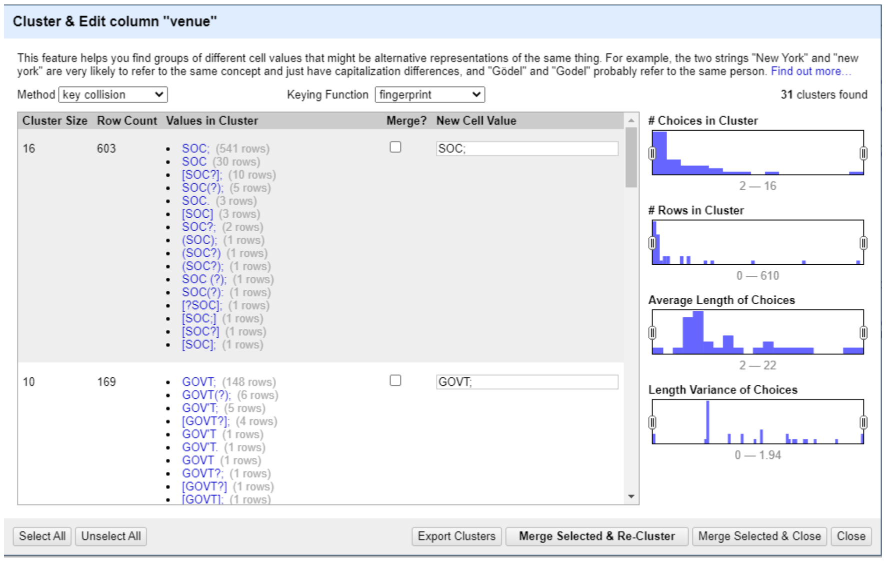
Menu.csv

Tool Used: OpenRefine

Column(s): `name, event, occasion`

Issues: Similar to Dish.csv, there are inconsistencies in the way the data is Entered in name, event, and occasion columns, from misspellings to extra spaces and other data entry related errors, that makes this data difficult to analyze later for our use case. The Major issue related to this csv is unsafe data, i.e., at certain fields the users were not sure about the values entered (for ex. (LUNCH?), (?DINNER?) (SUPPER?)). In order to fix these unreliable data we have made assumptions, by removing ?, that these values are correct and not unsafe data.

1. Inconsistent Values



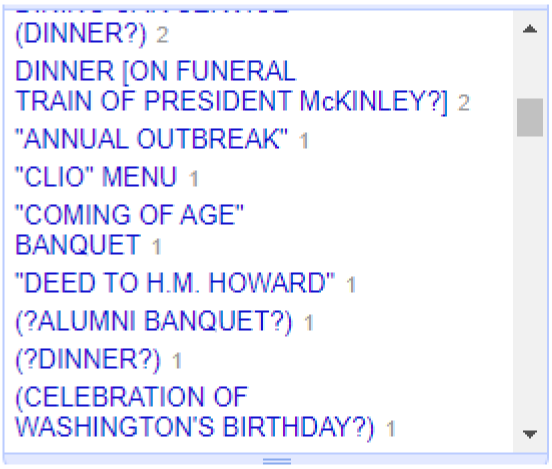
Cluster & Edit column "venue"

This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same person. Find out more...

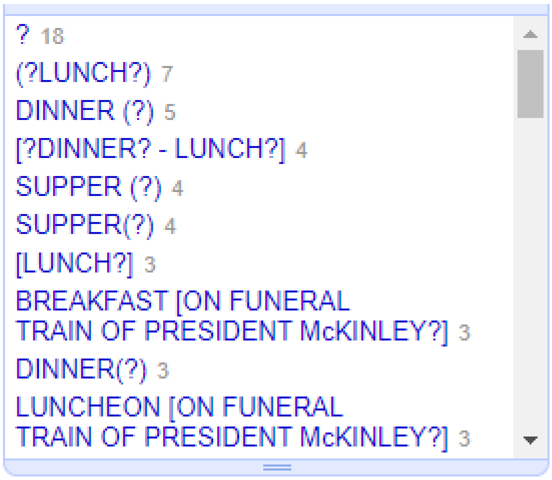
Method: key collision   Keying Function: fingerprint   **31** clusters found

Cluster Size	Row Count	Values in Cluster	Merge?	New Cell Value
16	603	SOC; (541 rows) SOC (30 rows) [SOC?]; (10 rows) SOC(?); (5 rows) SOC. (3 rows) [SOC] (3 rows) SOC?; (2 rows) (SOC); (1 rows) (SOC?) (1 rows) (SOC?); (1 rows) SOC (?); (1 rows) SOC(?): (1 rows) [?SOC]; (1 rows) [SOC;] (1 rows) [SOC?] (1 rows) [SOC]; (1 rows)	☐	SOC;
10	169	GOVT; (148 rows) GOVT(?); (6 rows) GOV'T; (5 rows) [GOVT?]; (4 rows) GOV'T (1 rows) GOV'T. (1 rows) GOVT (1 rows) GOVT?; (1 rows) [GOVT?] (1 rows) [GOVT]; (1 rows)	☐	GOVT;

2. Special Characters


(DINNER?) 2
DINNER [ON FUNERAL TRAIN OF PRESIDENT McKINLEY?] 2
"ANNUAL OUTBREAK" 1
"CLIO" MENU 1
"COMING OF AGE" BANQUET 1
"DEED TO H.M. HOWARD" 1
(?ALUMNI BANQUET?) 1
(?DINNER?) 1
(CELEBRATION OF WASHINGTON'S BIRTHDAY?) 1

3. Unsafe date


? 18
(?LUNCH?) 7
DINNER (?) 5
[?DINNER? - LUNCH?] 4
SUPPER (?) 4
SUPPER(?) 4
[LUNCH?] 3
BREAKFAST [ON FUNERAL TRAIN OF PRESIDENT McKINLEY?] 3
DINNER(?) 3
LUNCHEON [ON FUNERAL TRAIN OF PRESIDENT McKINLEY?] 3

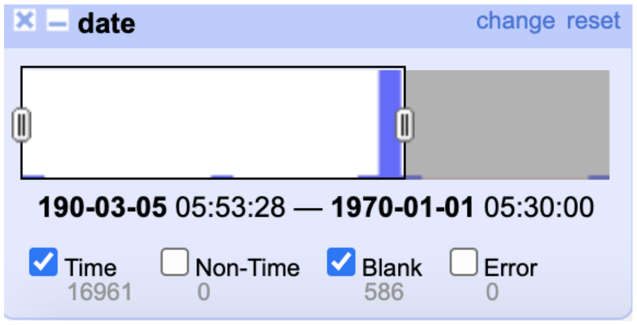
Recipe: (the below steps are not in the order they are executed)

1. Perform common transformations like “Trim leading and Trailing white spaces”, “collapse consecutive white space” (for name, event, and occasion columns), “to date” (for date column) and “to number” for page_count and dish_count columns)
2. Remove special characters from name, event, venue and occasion columns using REGEX via Edit Cell -> Replace option
3. Convert the unsafe data to safe data using the assumptions mentioned above
4. Replace the inconsistent values with a consistent value using OpenRefine’s “Cluster and Edit column” option
5. For values that require manual cleaning use the edit option in the text facet window
6. Use regex to replace field values that are separated by ‘,’, ‘:’ & ’;’, (for ex., “ANNIVERSARY; 13TH BURNS ANNIVERSARY;” ,”OTHER,Banquet”) to a common form (for ex., “ANNIVERSARY (13TH BURNS ANNIVERSARY)” ,”OTHER (Banquet)”) and then remove “()” and the enclosing text (for ex., “ANNIVERSARY” ,”OTHER”).

Column: `date`

Issues:

1. Found blank entries.



Recipe:

1. Keep the rows with blank entries but filter out the rows in the SQL query for use case #2 where this field will be used.

MenuPage.csv

Tool Used: Open Refine

Issues:

1. No significant data quality issues found in this dataset.

Recipe:

2. Transform the fields `id, menu_id, page_number, image_id, full_height, full_width` to number.

MenuItem.csv

Tool Used: Open Refine

Issues:

1. No significant data quality issues found in this dataset all well.

Recipe:

1. Transform the fields `id, menu_page_id, price, high_price, dish_id, xpos, ypos` to number
2. Transformed the columns created_at and updated_at to date using value.toDate('y-M-d H:m:s')

5.2. Data Integrity Issues

We created a database schema in SQLite and imported the dirty data.
The following sections capture our initial assessment on the data integrity issues in the dataset.

1. Primary key constraints.

Issues:

1. No primary key constraint violations found.

   We ran the following SQL queries to check if any of the tables violates the primary key constraint.

   ```
   SELECT id, count(id)
   FROM menu
   GROUP BY id
   HAVING count(id) > 1;
   ```

   Same for other tables, `menuitem`, `menupage` and `dish.`

2. Foreign Key constraint check (referential integrity).

   Issues:

   1. Records in `menupage` not having corresponding records in the `menu` table.

   ```
   sqlite> SELECT count(menu_id)
           FROM menupage
           WHERE menu_id NOT IN (SELECT id FROM menu);
   5799
   ```

   2. Records in `menuitem` not having corresponding records in `dish` table.

   ```
   sqlite> SELECT count(A.id)
   FROM menuitem A
   WHERE A. dish_id NOT IN (SELECT B.id FROM dish B);
   244
   ```

   Recipe:

   1. Delete records that violate referential integrity checks.

3. NULL constraint checks

   Issues:

   1. No NULL constraint check violations for primary keys found.

   ```
   SELECT Count(id)
   FROM   menu
   WHERE  id IS NULL

   SELECT Count(id)
   ```

```
FROM   menuitem
WHERE  id IS NULL

SELECT Count(id)
FROM   dish
WHERE  id IS NULL

SELECT Count(id)
FROM   menupage
WHERE  id IS NULL
```

4. Other data integrity checks

   The following are a list of data integrity checks we plan to run during the second phase of the project. These checks are there to ensure we are not working with corrupt data.

   a. Invalid values for `date` column in `menu` table.

   ```
   sqlite> SELECT date
   FROM   menu
   WHERE  ( date < 1700
             OR date > 2021 )
          AND date != "";
   ```

   b. Created date in menuitem table greater than updated date.

   ```
   SELECT * FROM menuitem
   WHERE CREATED_AT > UPDATED_AT;
   ```

   c. price in menuitem greater than the higher price.

   ```
   SELECT * FROM menuitem
   WHERE price > IFNULL(HIGH_price,0);
   ```

   d. Last appeared in dish table greater than current year

   ```
   SELECT * FROM dish
   WHERE LAST_APPEARED > 2021;
   ```

   e. Incorrect year in the dish table.

   ```
   SELECT * FROM dish
   WHERE IFNULL(FIRST_APPEARED,0) = 0 ;
   ```

   f. Incorrect year in the dish table.

```
SELECT * FROM dish
WHERE IFNULL(LAST_APPEARED,0) = 0;
```

g. First_appeared greater than last_appeared in the dish table.

```
SELECT * FROM dish
WHERE IFNULL(FIRST_APPEARED,0)  >
IFNULL(LAST_APPEARED,0);
```

h. Lowest price greater than highest price in dish table.

```
SELECT * FROM dish
WHERE IFNULL(lowest_price,0) >
IFNULL(highest_price,0);
```

6. Data Cleaning Operations

This section describes the actual data cleaning operations performed on each of the files in the dataset, the tool used and the rationale for performing the operation in the context of supporting our primary use case.

6.1. File: `Dish.csv`

6.1.1. Field: `name`

Cleaning the values of this field was the major focus of our work for `Dish.csv`. The following data cleaning operations were performed on this field (along with the rationale) using `OpenRefine`.

1. `Cluster and edit`

   As mentioned in the phase-1 report, we found a lot of instances where the same dish was specified in slightly different ways. Clustering provides a way to group these values and assign a standard name. This improved the accuracy of the aggregations performed using this field. We used the `key-collision` method using `fingerprint` and `ngram-fingerprint` (because many dishes were multi-word) functions to generate keys. We did not use the `phonetic` functions because we noticed that sometimes unrelated dishes were grouped into the same cluster. We were not able to get the nearest neighbour clustering scheme working on this field (the operation just wouldn't finish). Due to the sheer volume of clusters we did not inspect each and every cluster and instead just went by the recommendation provided by `OpenRefine`.

2. `Removed special characters`:

   Using the replace (with regex) feature in OpenRefine, we removed special characters (e.g. `[\"\*\-\>\{\}?\!]`) because they did not make sense in the names of dishes and also helped in duplicate detection and elimination using cluster and edit.

3. `Removed prices in dish names`:

   Many entries had prices of the dish (e.g. `$0.25`) in the name. We removed them using the replace feature (with regex) in OpenRefine as it further helped in detecting and eliminating duplicate dishes.

4. `Removed special characters`:

   Many entries had portions of the food in the dish name (e.g. `¼ lb.`). We removed them using the replace feature (with regex) in OpenRefine again to improve the detection and elimination of duplicate dishes.

5. `Removed spaces`:

   We repeatedly removed leading, trailing and consecutive white spaces to standardize the names.

6. `Removed the entries with empty dish names:`

   As they do not yield anything insightful in our query results.

6.1.2. Other columns

Most of the other fields in `Dish.csv` were left unchanged because they did need cleaning for our target use case.

6.1.3. Integrity Constraint Violations

There were no primary key (column: `id`) or foreign key constraint violations in `Dish.csv`.

[Note: We earlier reported integrity constraint violations in other fields in `Dish.csv` (e.g. `year`) in the phase-1 report but later realized that these did not need fixing for our target use case.]

6.2. File: `Menu.csv`

6.2.1. Field: `event`

This field was one of the major focus areas for this file/table. The following data cleaning operations were performed on this field using OpenRefine.

1. `Removed special characters`:

   Removed certain special characters `[\"\*\-\>\{\}?\!]` that did not make sense in the name of an event and also for standardization and normalization of event names for more accurate aggregations.

2. `Cluster and Edit:`

   Used OpenRefine "`Cluster and Edit`" feature to group similar event names and assign a standard label. We used both `key-collision` (all options) and `nearest-neighbour` (all options) methods to cluster entries repeated till we found no clusters.

3. `Custom transformation`

   Using GREL we performed a custom transformation on the event names, so that each [read as much as possible] entry maps to a standard name for the event. Our goal was to map as many entries as possible to one of `[Dinner, Breakfast, Lunch, Luncheon, Banquet, Meeting, Tiffin, Supper]`.

```
if (value.contains(/dinner/i), "Dinner",
  if (value.contains(/breakfast/i),
"Breakfast",
    if (value.contains(/lunch/i), "Lunch",
      if (value.contains(/luncheon/i),
"Luncheon",
        if (value.contains(/banquet/i),
"Banquet",
          if (value.contains(/supper/i),
"Supper", value,
            if (value.contains(/tiffin/i),
"Supper", value,
              if (value.contains(/meeting/i),
"Supper", value)))))))))
```

6.2.2. Field: `occasion`

The following data cleaning operations were performed on this field using OpenRefine.

1. `Removed special characters`:

   Removed certain special characters that did not make sense in the name of an event and performed common transformation steps like "Trim leading and Trailing white spaces" and "collapse consecutive white space".

2. `Cluster and Edit:`

   Used OpenRefine "`Cluster and Edit`" feature to group similar events and assign a standard label. We used both `key-collision` (all options) and `nearest-neighbour` (all options) methods to cluster entries repeated till we found no clusters.

3. `Custom transformation`

   Using GREL we performed a custom transformation on the occasion names, to replace field values that are separated by '**,**', '**:**' & '**;**', (for ex., "ANNIVERSARY; 13TH BURNS ANNIVERSARY;" ,"OTHER,Banquet") to a common form (for ex., "ANNIVERSARY (13TH BURNS ANNIVERSARY)" ,"OTHER (Banquet)") and then remove "()" or '[]' and the enclosing text (for ex., "ANNIVERSARY" ,"OTHER"), so that each entry maps to a standard name for the event.

4. `Manual Cleaning`

   Used Manual Cleaning Steps, using the edit option in Text Facet Window, to clean certain values for example ("à la carte" to "a la carte)

6.2.3. Integrity Constraint Violations

1. No primary key or foreign key constraint violations in `Menu.csv.`

2. Deleted rows in `Menu.csv` with invalid dates (i.e. `date < 1700 OR date > 2021`) . This was done to ensure that our results are not biased by the possible corrupted data.

6.3. File: `MenuPage.csv`

6.3.1. All Fields:

Our primary use case did not require any syntactic transformations to the fields in `MenuPage.csv` using OpenRefine. However we fixed some integrity constraints violations (described below).

6.3.2. Integrity Constraint Violations

1. No primary key in `MenuPage.csv.`
2. Detected foreign key constraint violations in `MenuPage.csv`. Deleted rows in `ManuPage.csv` where `menu_id` was not found in `Menu.csv` (i.e. a dangling or zombie menu page). This is important for our use case as our SQL query joins Menu.csv and `MenuPage.csv` on `menu_id.`

6.4. File: `MenuItem.csv`

6.4.1. All Fields:

Our primary use case did not require any syntactic transformations to the fields in `MenuItem.csv` using OpenRefine. However just like `MenuPage.csv` we fixed some integrity constraints violations (described below).

6.4.2. Integrity Constraint Violations

1. No primary key constraint violations found in `MenuItem.csv.`
2. Deleted rows in `MenuItem.csv` where `dish_id` was not found in `Dish.csv` (i.e. dangling or zombie menu items). This is important for our use case because our SQL query joins `Dish.csv` and `MenuItem.csv` on `dish_id`.

7. Data Quality Improvements

The following table summarizes the changes to the original dataset in quantitative terms as a result of our data cleaning operations described in section 6 and the corresponding improvements to the quality of data.

1. Dish.csv

Field/Column	Cells changed
name	~300K (out of ~400K) cells were changed.

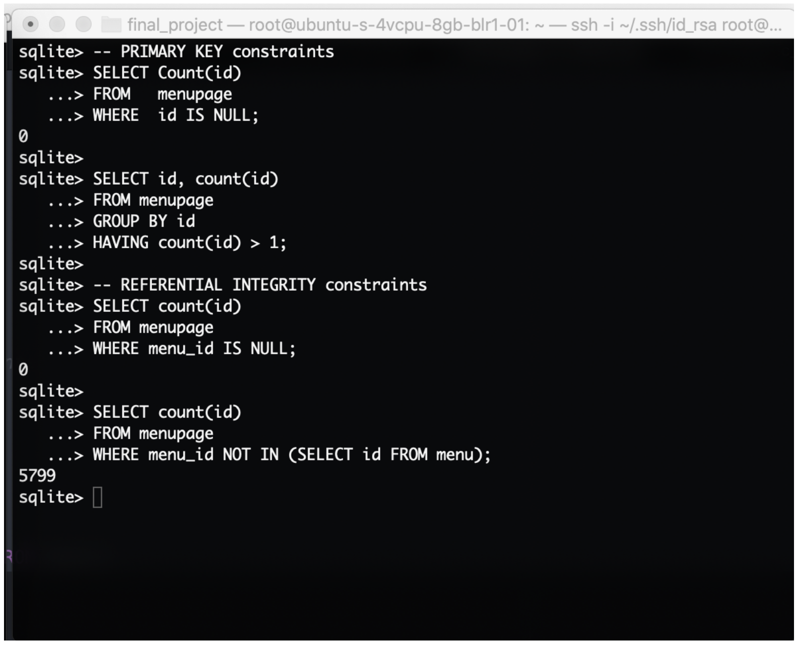
2. Menu.csv

Field/Column	Cells changed
event	2,634 (out of 17547) cells were changed
occasion	2,634 (out of 17547) cells were changed
date	16,961 (out of 17547) cells were changed

3. MenuPage.csv

Field/Column	Cells changed
menu_id	Fixed 5799 referential integrity constraint violations.

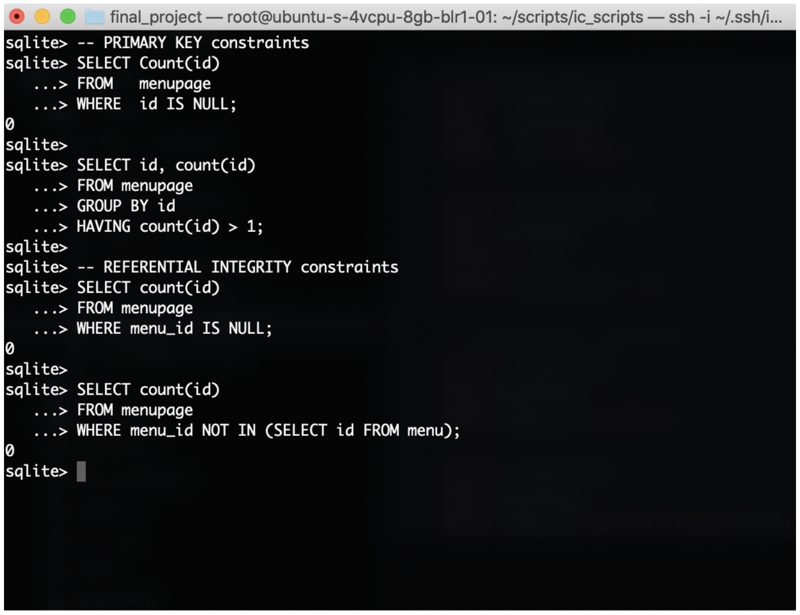
a. Before cleaning (menupage)



final_project — root@ubuntu-s-4vcpu-8gb-blr1-01: ~ — ssh -i ~/.ssh/id_rsa root@...

```
sqlite> -- PRIMARY KEY constraints
sqlite> SELECT Count(id)
   ...> FROM   menupage
   ...> WHERE  id IS NULL;
0
sqlite>
sqlite> SELECT id, count(id)
   ...> FROM menupage
   ...> GROUP BY id
   ...> HAVING count(id) > 1;
sqlite>
sqlite> -- REFERENTIAL INTEGRITY constraints
sqlite> SELECT count(id)
   ...> FROM menupage
   ...> WHERE menu_id IS NULL;
0
sqlite>
sqlite> SELECT count(id)
   ...> FROM menupage
   ...> WHERE menu_id NOT IN (SELECT id FROM menu);
5799
sqlite> 
```

b. After cleaning (menupage)



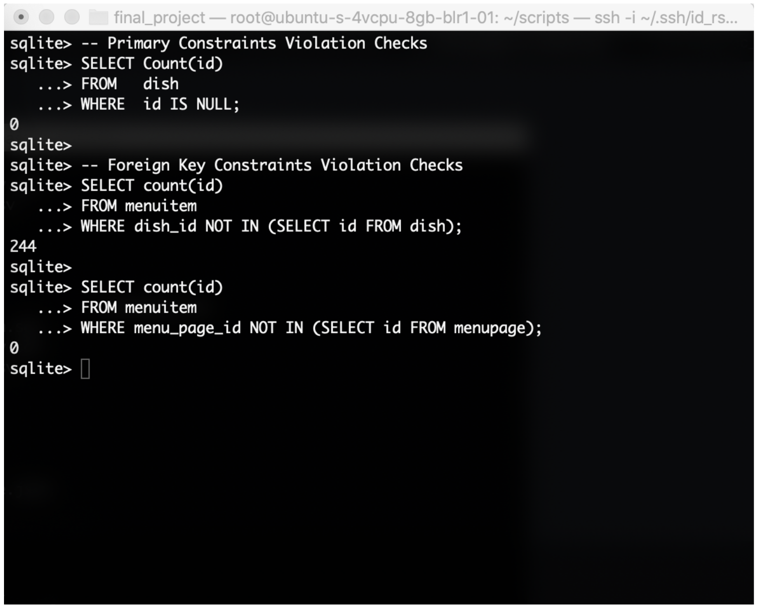
final_project — root@ubuntu-s-4vcpu-8gb-blr1-01: ~/scripts/ic_scripts — ssh -i ~/.ssh/i...

```
sqlite> -- PRIMARY KEY constraints
sqlite> SELECT Count(id)
   ...> FROM   menupage
   ...> WHERE  id IS NULL;
0
sqlite>
sqlite> SELECT id, count(id)
   ...> FROM menupage
   ...> GROUP BY id
   ...> HAVING count(id) > 1;
sqlite>
sqlite> -- REFERENTIAL INTEGRITY constraints
sqlite> SELECT count(id)
   ...> FROM menupage
   ...> WHERE menu_id IS NULL;
0
sqlite>
sqlite> SELECT count(id)
   ...> FROM menupage
   ...> WHERE menu_id NOT IN (SELECT id FROM menu);
0
sqlite> 
```

4. MenuItem.csv

Field/Column	Cells changed
dish_id	Fixed 244 referential integrity constraint violations.

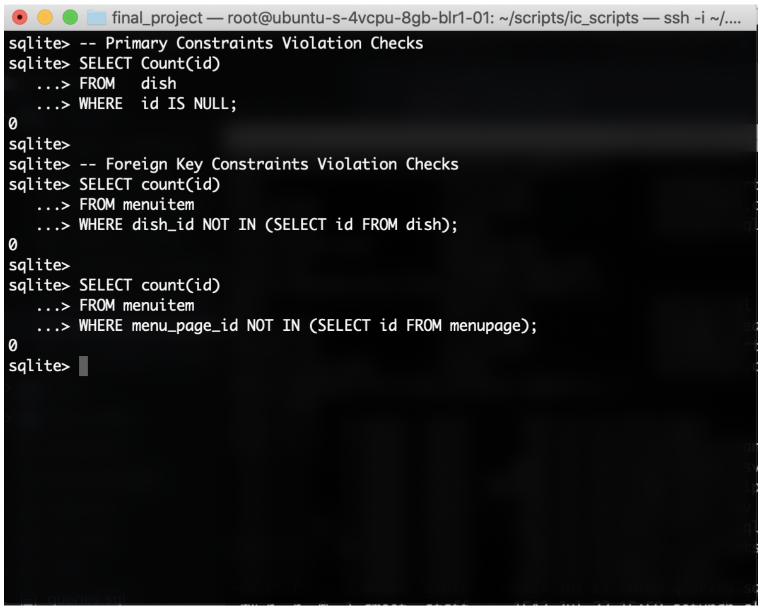
a. Before cleaning (menuitem)



final_project — root@ubuntu-s-4vcpu-8gb-blr1-01: ~/scripts — ssh -i ~/.ssh/id_rs...

```
sqlite> -- Primary Constraints Violation Checks
sqlite> SELECT Count(id)
   ...> FROM   dish
   ...> WHERE  id IS NULL;
0
sqlite>
sqlite> -- Foreign Key Constraints Violation Checks
sqlite> SELECT count(id)
   ...> FROM menuitem
   ...> WHERE dish_id NOT IN (SELECT id FROM dish);
244
sqlite>
sqlite> SELECT count(id)
   ...> FROM menuitem
   ...> WHERE menu_page_id NOT IN (SELECT id FROM menupage);
0
sqlite>
```

b. After cleaning (menuitem)

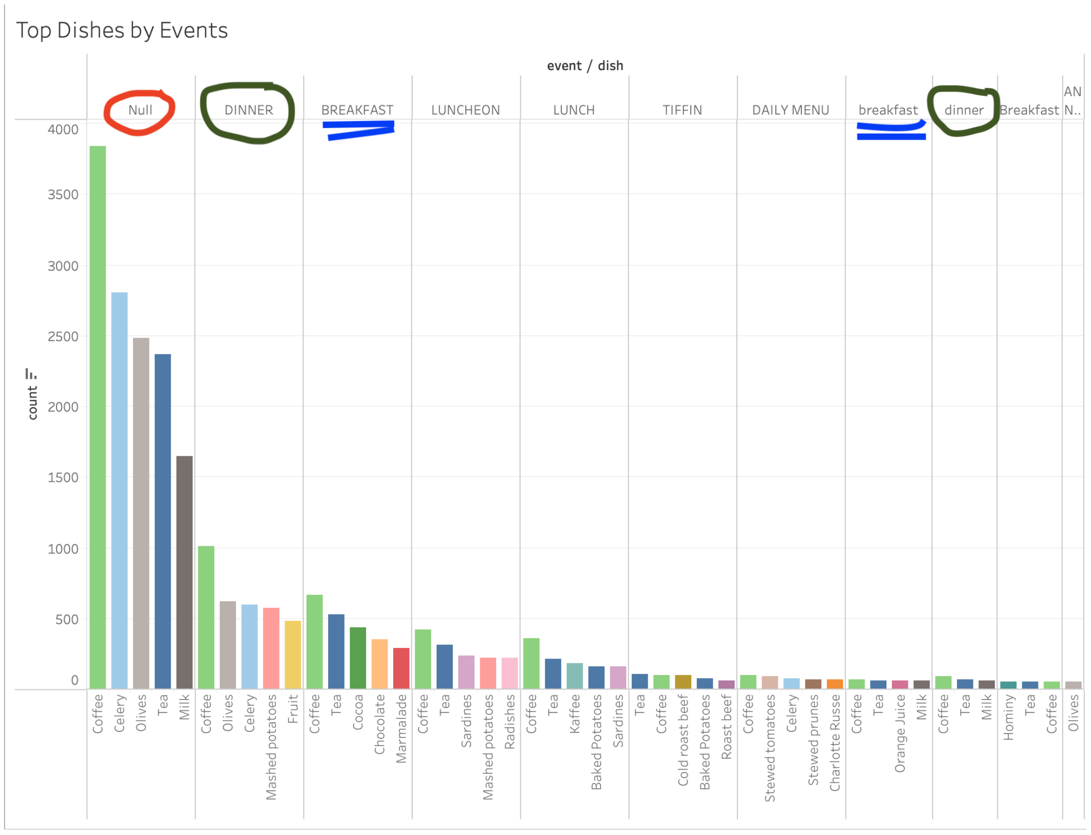


final_project — root@ubuntu-s-4vcpu-8gb-blr1-01: ~/scripts/ic_scripts — ssh -i ~/....

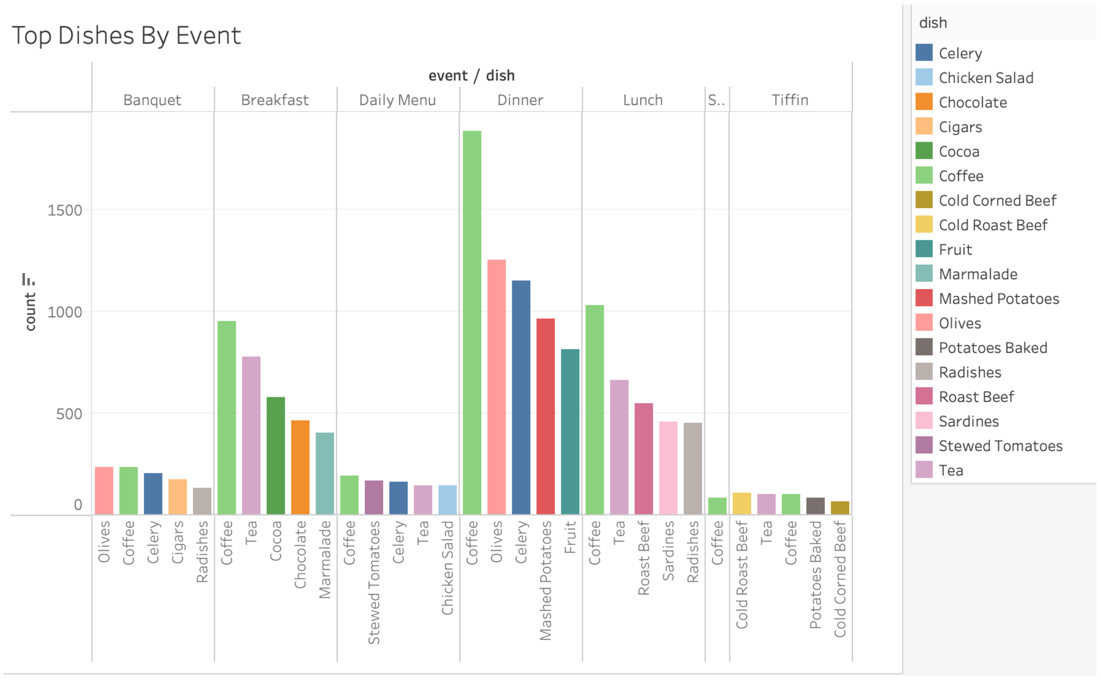
```
sqlite> -- Primary Constraints Violation Checks
sqlite> SELECT Count(id)
   ...> FROM   dish
   ...> WHERE  id IS NULL;
0
sqlite>
sqlite> -- Foreign Key Constraints Violation Checks
sqlite> SELECT count(id)
   ...> FROM menuitem
   ...> WHERE dish_id NOT IN (SELECT id FROM dish);
0
sqlite>
sqlite> SELECT count(id)
   ...> FROM menuitem
   ...> WHERE menu_page_id NOT IN (SELECT id FROM menupage);
0
sqlite>
```

The following charts provide visualizations of the results of our SQL queries before and after the data cleaning. We can clearly see the improvement in quality of the results. In both the use cases the charts generated from the clean dataset are far more interpretable.

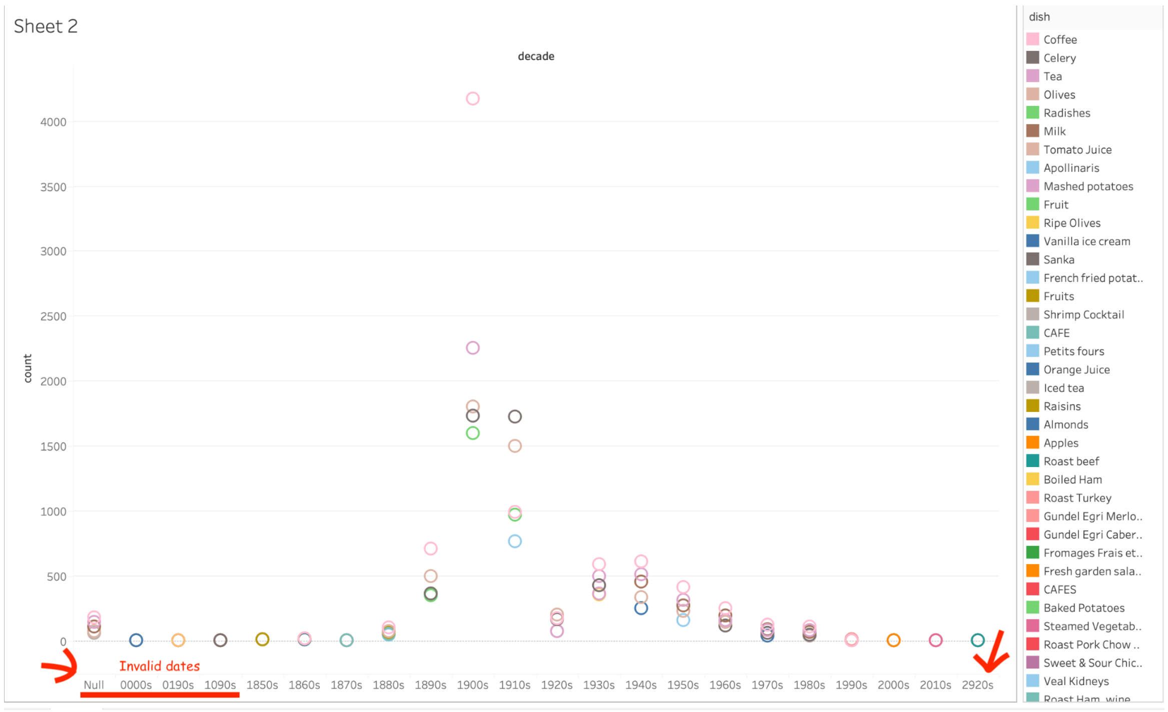
Use case #1.a: Top 5 dishes in events before data cleaning.



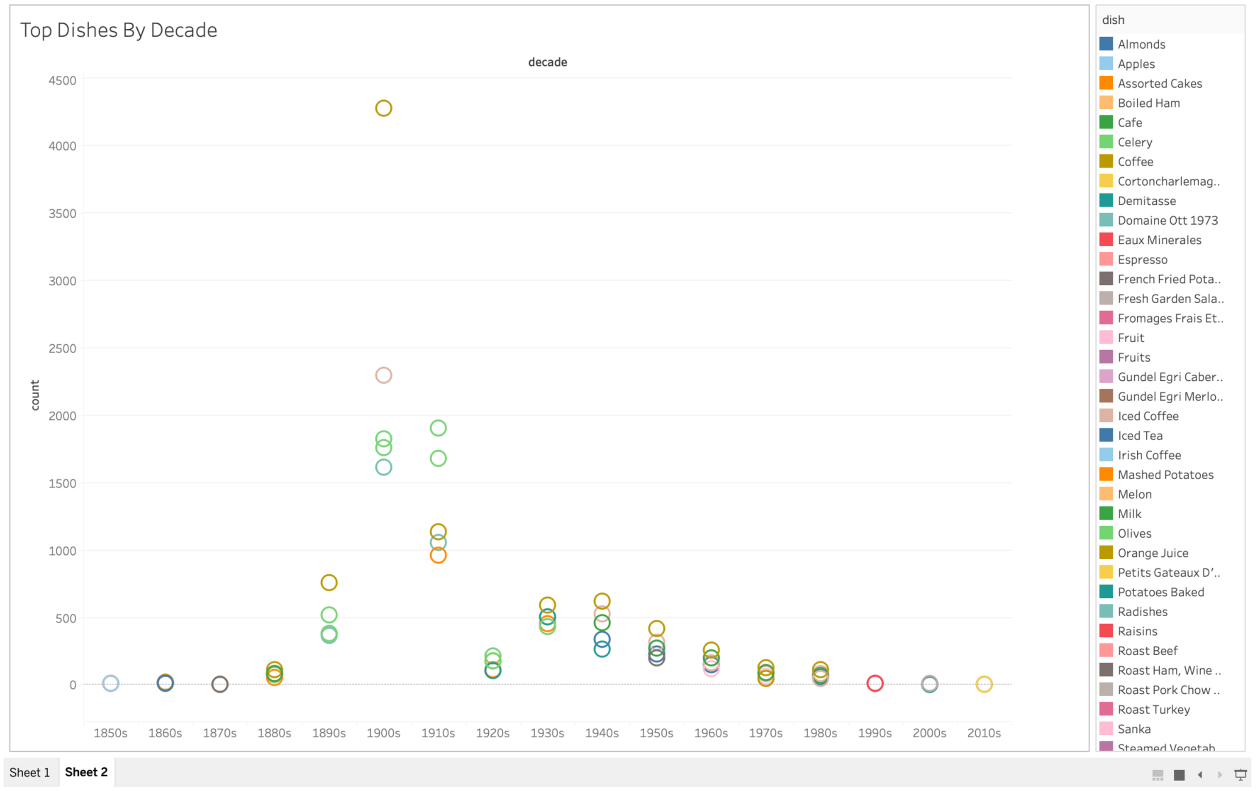
Use case #1.a: Top 5 dishes by events after data cleaning.



Use case #1.b: Top 5 dishes in a decade before data cleaning

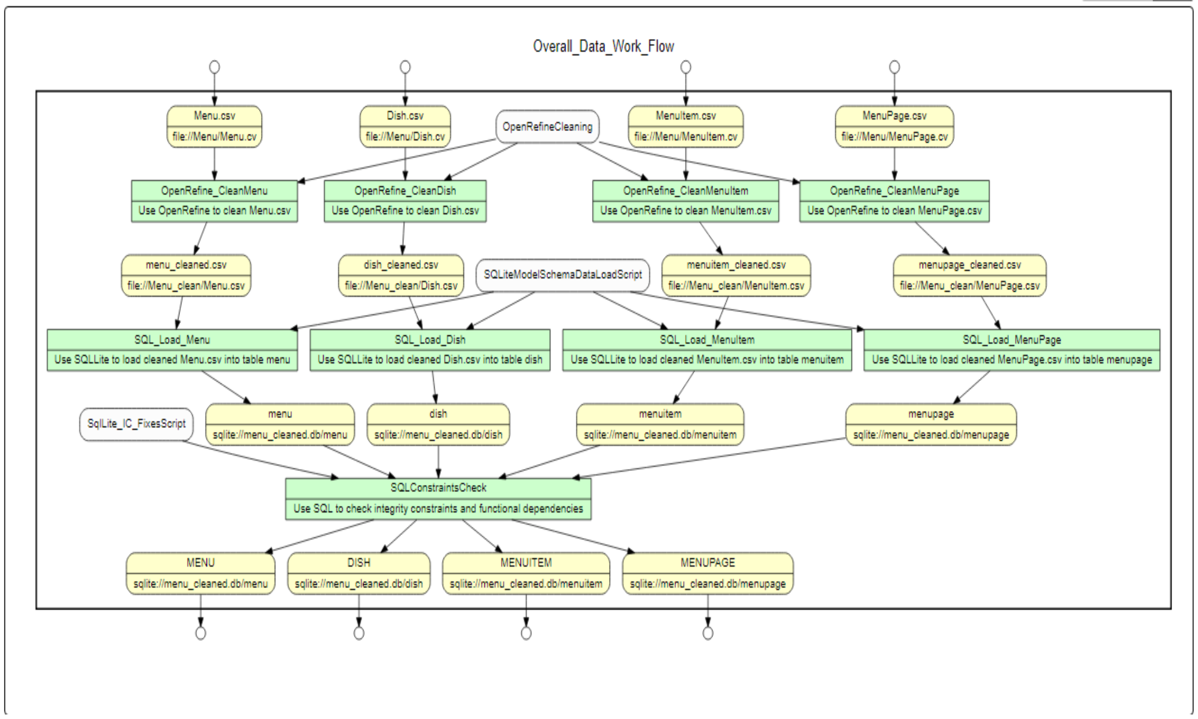


Use case #1.b: Top 5 dishes in a decade after data cleaning



8. Workflow Model

The following diagram was created using `YesWorkflow` webapp. It provides a high level view of our overall data cleaning workflow and also highlights the important computational steps with key inputs and outputs at each step.


Overall_Data_Work_Flow
Menu.csv
file://Menu/Menu.cv
Dish.csv
file://Menu/Dish.cv
OpenRefineCleaning
MenuItem.csv
file://Menu/MenuItem.cv
MenuPage.csv
file://Menu/MenuPage.cv
OpenRefine_CleanMenu
Use OpenRefine to clean Menu.csv
OpenRefine_CleanDish
Use OpenRefine to clean Dish.csv
OpenRefine_CleanMenuItem
Use OpenRefine to clean MenuItem.csv
OpenRefine_CleanMenuPage
Use OpenRefine to clean MenuPage.csv
menu_cleaned.csv
file://Menu_clean/Menu.csv
dish_cleaned.csv
file://Menu_clean/Dish.csv
SQLiteModelSchemaDataLoadScript
menuitem_cleaned.csv
file://Menu_clean/MenuItem.csv
menupage_cleaned.csv
file://Menu_clean/MenuPage.csv
SQL_Load_Menu
Use SQLLite to load cleaned Menu.csv into table menu
SQL_Load_Dish
Use SQLLite to load cleaned Dish.csv into table dish
SQL_Load_MenuItem
Use SQLLite to load cleaned MenuItem.csv into table menuitem
SQL_Load_MenuPage
Use SQLLite to load cleaned MenuPage.csv into table menupage
SqlLite_IC_FixesScript
menu
sqlite://menu_cleaned.db/menu
dish
sqlite://menu_cleaned.db/dish
menuitem
sqlite://menu_cleaned.db/menuitem
menupage
sqlite://menu_cleaned.db/menupage
SQLConstraintsCheck
Use SQL to check integrity constraints and functional dependencies
MENU
sqlite://menu_cleaned.db/menu
DISH
sqlite://menu_cleaned.db/dish
MENUITEM
sqlite://menu_cleaned.db/menuitem
MENUPAGE
sqlite://menu_cleaned.db/menupage

The major computational steps in our workflow and the tool used to perform each of them are as follows:

1. `OpenRefine_Clean*` (input: `raw_csv_file`, output: `curated_csv_file`)

   This step captures the data profiling, data cleaning and curation process (i.e. formatting issues, standardization and normalization of data). We used OpenRefine to perform this step because of its powerful features like creating facets, text clustering and custom transformation capabilities using GREL.

2. `SQL_Load_*` (input: `curated_csv_files`, output: `sqllite_db`)

   This step captures the loading of curated data from `OpenRefine` to `SQLite` database (preceded by schema definition step). We wrote a custom shell script in combination with `SQL` to perform this step. We scripted this step so the data loading process is completely automated so it can be repeated any number of times in a day with no manual effort (for e.g. say after someone performs an additional cleaning step in `OpenRefine`).

3. `SQL_Constraint_Check`

   This step captures the process of resolving all data integrity issues (primary key constraints, foreign key constraints and semantic issues associated with data). We used SQL because it was far easier to express the violation logic in SQL as compared to using GREL or regex in OpenRefine. We did not see the need for using datalog as we did not have to deal with recursive expressions.

8.1. Detailed Workflow

We used the `OR2YW` tool to generate YesWorkflow diagrams from the `OpenRefine` recipe files. As the diagrams are quite busy and difficult to follow without zooming in, we have included them in the zip file accompanying this report. Please refer to the following files in the zip file for detailed inner workflow diagrams for each of the files in the dataset.

```
1. InnerWorkflows/Dish.yw [and .png]
2. InnerWorkflows/MenuItem.yw [and .png]
3. InnerWorkflows/MenuPage_Parallel.yw [and .png]
4. InnerWorkflows/Menu_Parallel.pd.yw [and .png]
```

9. Conclusions

As part of this project we performed an end to end data cleaning exercise on the NYPL's Menu dataset [1]. At first, we used OpenRefine to explore and profile the dataset to estimate how dirty the dataset was. We then framed our target use case (a question) that we wanted the dataset to help answer. We then defined and executed a data processing workflow and performed the steps of data curation (using OpenRefine), database schema definition, data loading (using SQLite) and data integrity checks (using SQL). We also exported the results of our SQL query on both dirty and clean databases and performed visualizations using Tableau[2] charts to get a first hand feel of the improvement in quality of results as a result of our data cleaning exercise.

The following are some of our key conclusions and learnings from this exercise.

1. Most importantly, this exercise has made us realize [first hand] why they say the bulk of the work in a data analytics project happens in cleaning and curating the dataset. We also appreciate the meaning of the statement “garbage-in and garbage-out”.

2. We learnt the importance of clearly defining the target use case(s) before starting on data cleaning, otherwise the data cleaning may seem like a bottomless pit and we wouldn't know when enough is enough.

3. We now have a better understanding of the spectrum of tools (OpenRefine, SQL, datalog) available for data cleaning and their role in each step of the data cleaning workflow. We also realized there are limitations with some of the tools we used (e.g. scalability of OpenRefine) and also the fact this segment of the data analytics market is still evolving with availability of commercial tools like Trifecta[3].

4. We realized the importance of data provenance, where one should be able to trace the complete data lineage in order to fix issues like bad data seeping into the data lakes or warehouses. We also learnt how the `YesWorkflow` tool can be used for the purposes of data provenance.

The Team

Our project team consisted of 3 people. The following table lists the key contributions made by each member of the team.

Member Name	Primary Contributions	Secondary contributions
Ameet Deulgaonkar	1. Data cleaning and curation `Dish.csv` 2. Tableau Charts. 3. Scripts to load data from OpenRefine to SQLite. 4. Project Report.	1. SQL for use cases. 2. Review data cleaning work on `Menu.csv` and IC scripts. 3. Review scripts for fixing integrity constraints.
Gokul Krishnamurthy	1. Data cleaning and curation of `Menu.csv` 2. Workflow diagrams using YesWorkflow.	1. Review work on `Dish.csv.` 2. Project Report. 3. Review scripts for fixing integrity constraints.
Shreya Reddy Peesary	1. ER diagram 2. SQLite DB schema definition. 3. Scripts for fixing integrity constraints on all tables.	1. YesWorkflow diagrams for overall workflow. 2. Review data cleaning work on `Menu.csv` and `Dish.csv.` 3. Project Report.

References

[1] NYPL Lab’s menu dataset, http://menus.nypl.org/data
[2] Tableau Public, https://www.tableau.com/products/cloud-bi
[3] Trifecta, https://www.trifacta.com/data-cleansing/
[4] OpenRefine, https://openrefine.org/
[5] SQLite, https://www.sqlite.org/index.html

Appendix A

Brief description of the fields in each of the tables/csv files.

1. Menu.csv

Field Name	Description
`id`	Unique identifier for the menu
`name`	Unclear what this field is. Mostly blank, but in some places filled with the name of the restaurant.
`sponsor`	Name of the restaurant, organization or person sponsoring the menu
`event`	Break, lunch, dinner or anniversary etc.
`venue`	Indicates the type of the setting where the menu was served e.g. commercial, government, social etc.
`place`	Location of the event. No structure or format, some entries have an address others have the name of the restaurant etc.
`physical_description`	physical dimension of the menu (length x width)
`notes`	
`call_number`	Contact number
`keywords`	
`Language`	The language of the menu, mostly empty
`date`	Date the menu appeared
`location`	There is no clear difference between this and the place. Most like the name of the restaurant or organization.
`currency`	Currency in which the price is mentioned

`currency_symbol`	Symbol of the currency
`status`	Menu transcription status
`page_count`	Number of pages in the menu
`dish_count`	Number of dishes in the menu

2. MenuPage.csv

Field Name	Description
`id`	Unique identifier for menu page
`menu_id`	Reference to identifier or menu the page belongs to
`image_id`	Identifier of the digital image from which the transcription was done.
`full_height`	Height of the menu page in pixels
`full_width`	Width of the menu page in pixels
`uuid`	Unclear what this field is.

3. MenuItem.csv

Field Name	Description
`id`	Identifier of the menu item
`menu_page_id`	Identifier of the menu page that contains this menu item
`price`	Price of the menu item
`highest_price`	If the item appears more than once, then this field contains the highest price.
`dish_id`	Identifier of the dish that the menu item references.
`xpos and ypos`	Pixel coordinates at which the menu item appears in the menu page.
`created_at and updated_at`	Dates at which the menu item entry was created or updated in the database.

4. Dish.csv

Field Name	Description
`id`	Unique identifier for the dish
`name`	Name of the dish.
`description`	
`menus_appeared`	Number of menus in which the dish has appeared
`times_appeared`	Number of times the dish has appeared in a menu
`first_appeared`	Year in which the dish first appeared
`last_appeared`	Yeah in which the dish last appeared
`lowest_price`	The lowest ever quoted price for the dish
`highest_price`	The highest ever quoted price for the dish.