CS513 Data Cleaning Final Project

Mike Linfoot (linfoot2@illinois.edu) and Ahmad Al-Dhalaan (ahmada4@illinois.edu)

Summer 2020

1.0 Overview and Initial Assessment of the Dataset

The dataset is provided by the New York Public Library as part of their efforts in collecting and transcribing menu dish items in New York restaurants. The dataset covers approximately 45,000 menus from the 1840's to the present and the version used was dated 2020-06-16.

1.1 Structure and Content of the Dataset

The dataset consists of four comma-separated value (CSV) files: *Menu, MenuPage, MenuItem, and Dish.*

Menu

This is the main file in the dataset. Each row in this file represents a menu with an ID number and relevant data concerning the menu, such as the venue, location, number of pages and number of dishes. There are 17,546 rows in this file.

Each *Menu* observation is related to one or more *MenuPage* observation(s).

MenuPage

This file consists of the image data for each menu page and relates the *Menu* file with the *MenuItem* file. There are 66,938 rows in this file.

Each *MenuPage* observation is related to one or more *MenuItem* observation(s).

MenuItem

This file consists of the price of each item and the highest price recorded for that item. There are 1,048,575 rows in this file.

Some *MenuItem* observations are related to a *Dish* observation.

Dish

This file consists of the name of the item, number of times it has appeared and date and price ranges. There are 426,986 rows in this file.

1.2 Data Quality Issues

The following data quality issues were identified with the initial dataset:

- Menu:
    - It is not clear what *name*, *sponsor* and *location* fields are used for (some are the same as each other, some are blank). Some inconsistent spellings within these fields.
    - A few invalid values in *date* field: years 0001 (x2), 0190, 1091, 2928.
    - Many blank entries in *date* field.
    - Lots of inconsistencies in *occasion* and *event* fields. Sometimes these are the same, sometimes different.
    - Inconsistencies in *venue* and *place* fields.
    - *dish_count* = 0 for some *Menu* records even though there are associated *MenuPage* records with *MenuItem* records attached.
- MenuPage:
    - Many blank *page_number* fields (1202 records).
    - Many blanks in *full_height* and *full_width* fields.
- MenuItem:
    - Many blank *dish_id* fields.
    - Some items have a *high_price* lower than *price*.
    - *price* is 0 or blank for many records.
- Dish:
    - Different variations of *name* field values for similar dishes.
    - *description* field is always blank.
    - For dishes in USD, some menus use dollars, some use cents.
    - Some invalid *first_appeared* and *last_appeared* field values (e.g. 1).

1.3 Use Cases

The dataset holds a wide variety of possible use cases that an analyst might find interesting. Some of these use cases require cleaning of the dataset before analysis can begin. Some use cases can be started with the current state of the dataset, while other use cases do not seem to be achievable even with a rigorous data cleaning process.

1.3.1 Feasible Use Case

A feasible use case that can be achieved in an end-to-end data cleaning workflow relates to the price evolution of dishes over time. *Dish* has the dish name and price ranges over the range of dates that the item was on a menu. *MenuItem* then fills in the range with various “snapshots” of the price of the item over years. *MenuItem* only includes the price of the item, so *Menu* will be needed to get the associated date for that price.

MenuPage is used to relate *Menu* to the other two files. A scatter plot can be generated from this clean dataset plotting price versus date for each item.

In this use case, the messy and inconsistent values in the text columns in *Menu* will not be necessary. We will mainly need to ensure that the “date” column is clean and consistent with the date range in *Dish*. The same consistency will need to be ensured in *MenuItem*, ensuring that item prices (“price” column) are in the price range in *Dish*. The date ranges (“first_appeared”/“last_appeared” columns) and price ranges (“lowest_price”/“highest_price” columns) in *Dish* will also need to be consistent, with the second column in each range being greater than the first column. In addition, the dish names in the *Dish* “name” column will need to be cleaned and clustered to ensure that each dish name is unique.

Finally, we need to ensure that integrity between the tables is maintained, i.e. that foreign keys do not contain NULL values or values that do not exist as a primary key in the associated table.

1.3.2 Other Use Cases

There are some use cases where the dataset is already clean enough, especially when null values are ignored. Summary statistics, such as mean, median, and range, can be achieved from *Menu* and *MenuItem*. In *Menu*, we can get the mean page count for all menus and the mean dish count in all menus. Similarly, in *MenuItem* we can calculate mean price and mean high prices from available prices.

On the other hand, use cases that we could not envision the dataset being clean enough for are use cases dealing with the restaurant name and its associated sponsor, event, location, venue and place in the *Menu* file. The cells in these columns are very messy and do not follow an obvious consistent method. It will be very difficult to simultaneously clean these columns while maintaining their accuracy.

2.0 OpenRefine

We begin the data cleaning workflow with the use of OpenRefine. OpenRefine is a data wrangling desktop application with powerful tools in data cleaning. In this project, we use OpenRefine to perform cleaning operations, such as faceting, clustering and transformations. We use built-in tools and write our own regular expressions and python scripts to accomplish these data cleaning tasks.

We upload the four CSV files in OpenRefine with UTF-8 encoding, which preserves the characteristics of the text columns. We then separately clean each file and produce clean CSV files and JSON recipes that document the cleaning process for each file, all attached in the supplementary materials ZIP file.

2.1 Dish.csv:

We begin by performing basic cleaning and transformation tasks before we further investigate the table:

- trim and collapse white space for the text column:
  - *name* column
- transform all numeric columns to number:
  - *id, menus_appeared, times_appeared, first_appeared, last_appeared, lowest_price, highest_price* columns
- delete irrelevant empty column:
  - *description* column

Faceting the *name* column, we see dishes with similar names that can be grouped together. Clustering the *name* column using the "key collision" method and "fingerprint" keying function we are presented with 40,778 clusters. Attempting to perform this large cluster in one step was computationally not possible for us. Therefore, we split the merging process of the clusters into 13 steps as follows:

Number of Merges	Cluster Size (rows)	Number of Clusters	Cells Edited
1	6-45	2,585	24,022
1	5	1,523	7,972
1	4	2,940	12,248
3	3	6,850	21,142
7	2	26,880	54,659

Finally, we wrote a jython if/else statement to transform the *first_appeared* and *last_appeared* columns, making "null" all values outside the range of 1850-2020:

```
if (value < 1850 or value > 2020):
  return None
else:
  return value
```

2.2 Menu.csv:

We begin by performing basic cleaning and transformation tasks before we further investigate the table:

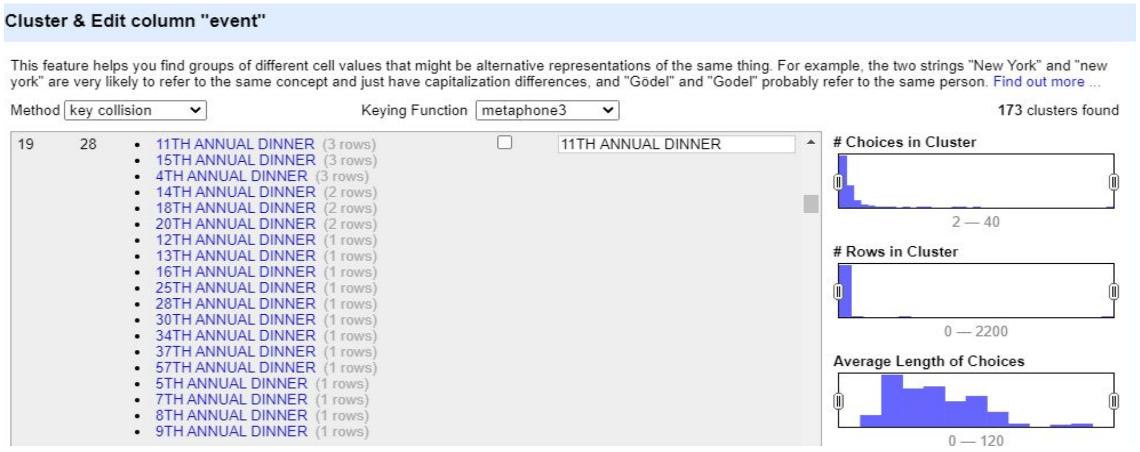
- trim and collapse white space for the text columns:
    - *name, sponsor, event, venue, place, physical_description, occasion, notes, location, currency, currency_symbol, status, call_number* columns
- transform all numeric columns to number:
    - *page_count, dish_count* columns
- transform *date* column to date and remove unnecessary time information using:
    - *toString(toDate(value),"yyyy-MM-dd")*
- delete irrelevant empty columns:
    - *keyword, language, location_type* columns

Faceting the *name, sponsor, event, venue, place, occasion and location* columns, we see texts with unwanted special characters. We remove the special characters, *% # ! “ \ ? ( ) [ ]*, from these columns using the following GREL expression:

```
value.replace(/%/,").replace(/#/,"). replace(/!/,").replace(/\"/,")
.replace(/\\\\/,").replace(/\\?/,").replace(/\\(/,")
.replace(/\\)/,").replace(/\\[/,").replace(/\\]/,")
```

Faceting also shows cells with similar texts that can be grouped together in the *name, sponsor, event, venue, place, occasion and location* columns. We cluster and merge the text in these cells using the fingerprint, ngram-fingerprint (N=2) and metaphone3 keying functions of the key collision clustering method.

For all columns, the fingerprint, and ngram-fingerprint keying functions were appropriate. However, the metaphone3 keying function was not appropriate for some columns as it made many inaccurate clusters, especially when there were numbers in the text. In the *event* column, for example, the metaphone3 keying function inaccurately clustered 19 choices into one cluster as shown in the following screenshot from OpenRefine:



Cluster & Edit column "event"

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: metaphone3
173 clusters found

19 28
- 11TH ANNUAL DINNER (3 rows)
- 15TH ANNUAL DINNER (3 rows)
- 4TH ANNUAL DINNER (3 rows)
- 14TH ANNUAL DINNER (2 rows)
- 18TH ANNUAL DINNER (2 rows)
- 20TH ANNUAL DINNER (2 rows)
- 12TH ANNUAL DINNER (1 rows)
- 13TH ANNUAL DINNER (1 rows)
- 16TH ANNUAL DINNER (1 rows)
- 25TH ANNUAL DINNER (1 rows)
- 28TH ANNUAL DINNER (1 rows)
- 30TH ANNUAL DINNER (1 rows)
- 34TH ANNUAL DINNER (1 rows)
- 37TH ANNUAL DINNER (1 rows)
- 57TH ANNUAL DINNER (1 rows)
- 5TH ANNUAL DINNER (1 rows)
- 7TH ANNUAL DINNER (1 rows)
- 8TH ANNUAL DINNER (1 rows)
- 9TH ANNUAL DINNER (1 rows)

11TH ANNUAL DINNER

Therefore, we excluded the metaphone3 keying function from some column clustering.

The table below shows the results of the clustering and merging of these columns:

Column	**Keying Function**	**Number of Clusters**	**Cells Edited**
name	fingerprint	31	706
	ngram-fingerprint	13	622
	metaphone3	36	780
sponsor	fingerprint	321	5440
	ngram-fingerprint	75	1844
	metaphone3	400	5177
event	fingerprint	68	5597
	ngram-fingerprint	15	2329
venue	fingerprint	31	2251
	ngram-fingerprint	7	32
	metaphone3	8	5052
place	fingerprint	320	3235
	ngram-fingerprint	211	2435
physical_description	fingerprint	231	1514
	ngram-fingerprint	549	4575
occasion	fingerprint	47	2779
	ngram-fingerprint	9	272
notes	fingerprint	151	1548
	ngram-fingerprint	24	320
location	fingerprint	183	4440
	ngram-fingerprint	68	48
currency_symbol	fingerprint	4	216

Finally, faceting the *date* column showed three cells having a year of 0190, 1091 and 2928. These were manually transformed to "null".

2.3 MenuItem.csv & MenuPage.csv:

Numeric columns were transformed to number. No other transformations or cleaning tasks are done and the files are ready for further data analysis as is.

3.0 Relational Schema

The output files from OpenRefine were loaded into a MySQL database hosted on Amazon RDS, in order to check cross-table integrity and perform further cleaning.

3.1 Schema and Table Creation

The schema and tables were created as below:

```
/* Create the schema */
CREATE SCHEMA `menus` DEFAULT CHARACTER SET utf8 ;

/* Create Dish table */
CREATE TABLE `menus`.`Dish` (
  `id` INT NOT NULL,
  `name` VARCHAR(2000) NULL,
  `description` VARCHAR(2000) NULL,
  `menus_appeared` INT NULL,
  `times_appeared` INT NULL,
  `first_appeared` INT NULL,
  `last_appeared` INT NULL,
  `lowest_price` DECIMAL(10,2) NULL,
  `highest_price` DECIMAL(10,2) NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC));

/* Create MenuItem table */
CREATE TABLE `menus`.`MenuItem` (
  `id` INT NOT NULL,
  `menu_page_id` INT NULL,
  `price` DECIMAL(10,2) NULL,
  `high_price` DECIMAL(10,2) NULL,
  `dish_id` INT NULL,
  `created_at` DATETIME NULL,
  `updated_at` DATETIME NULL,
  `xpos` DECIMAL(7,6) NULL,
  `ypos` DECIMAL(7,6) NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC));

/* Create MenuPage table */
CREATE TABLE `menus`.`MenuPage` (
  `id` INT NOT NULL,
  `menu_id` INT NULL,
  `page_number` INT NULL,
  `image_id` INT NULL,
  `full_height` INT NULL,
  `full_width` INT NULL,
  `uuid` VARCHAR(45) NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC));

/* Create Menu table */
CREATE TABLE `menus`.`Menu` (
  `id` INT NOT NULL,
  `name` VARCHAR(100) NULL,
  `sponsor` VARCHAR(100) NULL,
  `event` VARCHAR(100) NULL,
  `venue` VARCHAR(100) NULL,
  `place` VARCHAR(100) NULL,
  `physical_description` VARCHAR(100) NULL,
  `occasion` VARCHAR(100) NULL,
  `notes` VARCHAR(2000) NULL,
  `call_number` VARCHAR(45) NULL,
  `keywords` VARCHAR(100) NULL,
  `language` VARCHAR(45) NULL,
  `date` DATE NULL,
  `location` VARCHAR(100) NULL,
  `location_type` VARCHAR(45) NULL,
  `currency` VARCHAR(45) NULL,
  `currency_symbol` VARCHAR(45) NULL,
  `status` VARCHAR(45) NULL,
  `page_count` INT NULL,
  `dish_count` INT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `id_UNIQUE` (`id` ASC));
```

No foreign key constraints were used as the goal was to load all data into the tables (even if invalid) and then check data integrity constraints using scripts.

3.2 Loading the Data

The cleaned output data from OpenRefine was then loaded into the tables using Python scripts. These scripts “load_Menu.py”, “load_MenuPage”, “load_MenuItem.py” and “load_Dish.py”, can be found in the supplementary materials ZIP file.

The scripts inserted the data into the tables in blocks of 10,000 rows to avoid memory/time-out issues with the larger tables. They also performed minimal pre-processing on the data, replacing blank integer or data values with the value NULL, to avoid them being set to 0 by MySQL.

3.3 Integrity Constraints

The following integrity constraints were identified and checked:

1) Each *MenuPage* record has a corresponding *Menu* record
    - 0 *MenuPage* records have a NULL *menu_id* field
    - 5803 *MenuPage* records have a menu_id field not present in *Menu* table

2) Each *MenuItem* record has a corresponding *MenuPage* record
    - 0 *MenuItem* records have a NULL *menu_page_id* field
    - 0 *MenuItem* records have a *menu_page_id* field not present in *MenuPage* table

3) Each *MenuItem* record has a corresponding *Dish* record
    - 241 *MenuItem* records have a NULL *dish_id* field
    - 3 *MenuItem* records have a *dish_id* field not present in *Dish* table

4) Number of *MenuPage* records for each *Menu* matches the *page_count* field in the *Menu* record
    - 0 *MenuPage* records have mismatch in *page_count* field

5) Total number of *MenuItem* records across each *MenuPage* for a *Menu* matches the *dish_count* field in the *Menu* record
    - 211 *Menu* records have mismatch in *dish_count* field

6) *first_appeared* and *last_appeared* fields in *Dish* table match earliest and latest *date* fields in corresponding *Menu* records
    - 5847 *Dish* records have mismatch in *first_appeared* or *last_appeared* field

7) *price* field in *MenuItem* record is in range of *lowest_price* / *highest_price* for corresponding *Dish* record
    - 151,833 *MenuItem* records have value in *price* field out of range

8) *menus_appeared* field in *Dish* record corresponds to the actual number of *Menu* records with which the *Dish* is associated (via *MenuItem* and *MenuPage* records)
    - 10,783 *Dish* records have mismatch in *menus_appeared* field

9) *times_appeared* field in *Dish* record corresponds to the actual number of times the *Dish* is referenced by a *MenuItem* record
    - 136 *Dish* records have mismatch in *times_appeared* field

The SQL queries used to check each of these constraints (and their results) can be found in the file “Queries.txt” in the supplementary materials ZIP file.

3.4 Fixing Integrity Issues

For the chosen use case, the most important cleaning identified at this stage was to resolve the foreign key integrity issues (constraints 1 and 3 above) because it is essential to be able to link *Dish* records through to their associated *Menu* via the *MenuItem* and *MenuPage* records in order to see the year of the menu on which they appeared.

For the NULL foreign keys and values pointing to non-existent primary keys, without having access to the source data and fixing manually (which would be a huge task out of scope of this project), the only solution available was to delete the affected records.

This was performed using DELETE queries, provided in the file "DB_Cleaning.txt" in the supplementary materials ZIP file. The steps followed were as below:

1) Delete *MenuPage* records linked to non-existent *Menu* records. This removed 5803 *MenuPage* records.
2) Due to the deletion of these *MenuPage* records, there were a number of *MenuItem* records linking to *MenuPage* records that no longer existed. These *MenuItem* were therefore deleted in turn. This removed 5373 *MenuItem* records.
3) Delete *MenuItem* records linked to non-existent *Dish* records. This removed 3 *MenuItem* records.
4) Delete *MenuItem* records with NULL *dish_id* value. This removed 241 *MenuItem* records.

"Orphaned" *Dish* records (i.e. those not referenced by any *MenuItem*) record were not deleted as these could still be used in the future if the dishes appear on new menus, to avoid new records having to be created for these. These do not represent an integrity issue even if they could be considered redundant.

The results of the DB cleaning are as below:

Table name	**Rows before DB cleaning**	**Rows after DB cleaning**	**Difference**
Menu	17,545	17,545	0
MenuPage	66,937	61,134	-5,803
MenuItem	1,334,419	1,328,802	-5,617
Dish	426,985	426,985	0

3.4 Exporting Tables

Once cleaned, the tables were exported into CSV files using the Table Data Export Wizard in MySQL Workbench.

4.0 Workflow Model

The key inputs to the data cleaning workflow are the 4 CSVs files “Menu.csv”, “MenuPage.csv”, “MenuItem.csv” and “Dish.csv”.

These are cleaned in OpenRefine to generate intermediate “cleaned” output files which are a dependency for the next stage, the Python table loaders. Also output at this stage are the JSON recipe files.

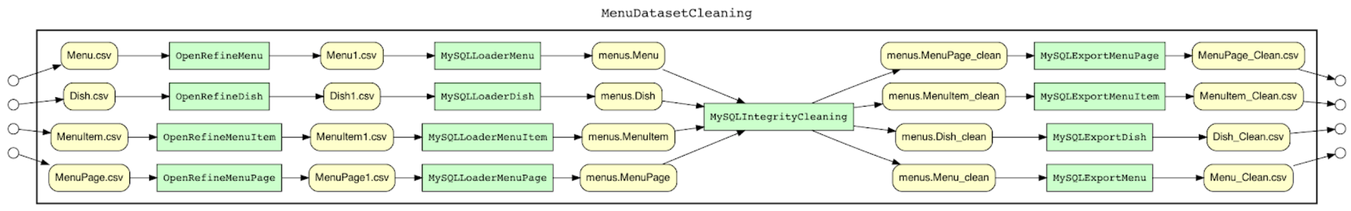
The cleaned output files are input to the Python table loaders with the output being the files loaded into MySQL tables, which is a dependency for the next stage.

The MySQL tables are then checked for integrity constraints, some of which are fixed using SQL. The output from this stage is the cleaned tables in the database, which are a dependency for the final stage.

This stage, which generates the final output from the workflow, takes the database tables as input and exports the tables into 4 CSV files, “Menu_Clean.csv”, “MenuPage_Clean.csv”, “MenuItem_Clean.csv” and “Dish_Clean.csv”.

4.1 Overall Workflow

The overall workflow as generated by YesWorkflow is as below:

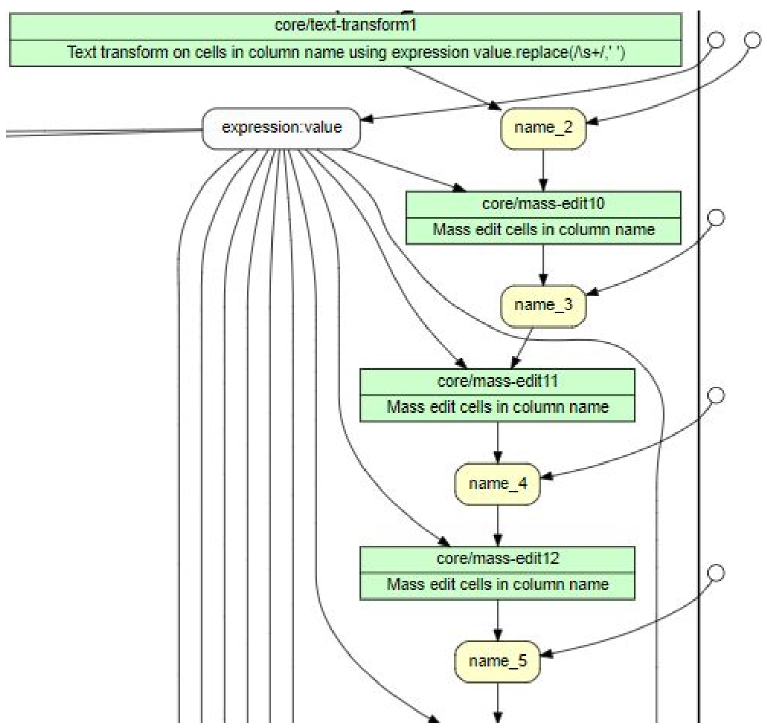


The YesWorkflow annotations file (“Overall_Workflow.py”) and the generated Graphviz file (“Overall_Workflow.gv”) can be found in the supplementary materials ZIP file.

4.2 OpenRefine Workflows

Each table is represented by an OpenRefine YesWorkflow, demonstrating the cleaning steps undertaken. In each workflow, the input is the CSV file from NYPL and the output are the clean files and JSON files. While the linear nature of the workflow might indicate the presence of many dependencies, there are in fact no true dependencies and the steps can be taken in any order.

Below is a section of the *Dish* cleaning workflow, showing the beginning of the 13-step clustering process on the “name” column mentioned in section 2.1:


core/text-transform1
Text transform on cells in column name using expression value.replace(/\s+/,' ')
expression:value
name_2
core/mass-edit10
Mass edit cells in column name
name_3
core/mass-edit11
Mass edit cells in column name
name_4
core/mass-edit12
Mass edit cells in column name
name_5

The full workflows are too large to show in this report. We have provided the generated Graphviz files for each table along with their visual representations (in PNG format) in the supplementary materials ZIP file.

5.0 Conclusion

By first cleaning individual tables using OpenRefine, then loading the cleaned tables into MySQL and fixing a number of integrity constraint issues, we were able to generate clean data fit for purpose for the chosen use case.

The whole process was documented using JSON recipes from OpenRefine, Python scripts, SQL queries and YesWorkflow workflows, thus enabling provenance information to be captured and the process to be easily repeated.

6.0 Split of Work

The work was evenly split between the two group members with the main focus areas as below:

Ahmad Al-Dhalaan - initial data quality assessment, use case development, OpenRefine data cleaning, OpenRefine workflows, documentation

Mike Linfoot - initial data quality assessment, use case development, SQL data loaders, SQL data integrity checking and cleaning, overall workflow, documentation