NYPL dataset

8-Bit Cleaners

Nitin Tangellamudi (tangell2) Blake Ordway (bordway2)

University of Illinois Urbana-Champaign

CS 513 Data Cleaning Final Project

Professor Ludaescher

August 3, 2020

8-Bit Cleaners

CS 513 Data Cleaning Final Project

NYPL dataset

The NYPL dataset contains 4 different files: *Dish.csv, Menu.csv, MenuItem.csv*, and *MenuPage.csv*. In our project, our two-person team took the first 2 files and performed the prescribed steps on the dataset. The following sections describe the approach we took to analyze and clean the data.

1. Analysis

The NYPL menu dataset contains 4 comma-separated CSV files with sizes ranging from 3.1MB to 115.6MB. Each file contains some unique ID to identify the record. Some of the files are clean enough to use right now for some purposes. In general, all files would need *some* cleaning such as white space cleaning.

After looking through the dataset, it may be impossible to use the dataset, for example, to determine the average height and width of the menus using all of the records since in some places it seems that the height and width columns' values are swapped. Cleaning that would be next to impossible, at least for automatic cleaning. (Although a good data scientist/statistician could just trim the bottom 2.5% and top 2.5% values from the dataset, that doesn't relate to cleaning.)

Very simple use cases such as finding the average number of dishes per menu should be pretty simple and not require any type of cleaning of the data. The number of dishes could just be counted and divided by the number of menus.

To be used for a normal use case, such as just looking up the most/least popular dishes by year, data should be cleaned in the most general way possible. This includes trimming unnecessary whitespace, removing superfluous columns, combining fingerprinted string clusters, and making sure some degree of capitalization is used.

1.1 Dish.csv

The file contains the name of dishes, the number of menus it appeared in, the number of times it appeared (possibly different since it could appear multiple times in a single menu), the first time it appeared, the last time it appeared, the lowest recorded price, and the highest recorded price. There is a place for a description of the dish, but this field is blank in all records.

1.2 Menu.csv

This file contains information regarding the menus that have been transformed into these CSV files. There is a field for the menu name but is generally empty until later records. Also included are the sponsor, event, venue, place, physical description of the menu, the occasion for the menu, any notes about the menu, the call number, any keywords, the language it was printed in, the date, location, and location type. There are also other fields such as currency used, the currency's symbol, the status of the parsing/transformation (in progress, complete), the number of pages, and the number of dishes.

2. Cleaning the files

To clean the files, we used OpenRefine. The max memory, heap size, and form content size were increased so that OpenRefine could handle the larger file sizes and transformations required to clean the data.

2.1. Dish.csv

1. **DELETE EXTRANEOUS COLUMN**. The **description** field is empty for all records. This column can be safely removed from the dataset.

2. **REORDER ID FIELD.** Some ids are occasionally skipped. The ids should be in numerical order with no ids skipped, starting at id 1.
    a. Transform the **id** by using the following GREL: row.index + 1
    b. There should be no missing IDs now in the records

3. **STRING GENERAL CLEANING**. The **name** column has a mix of all caps, initial caps, or title case. **Name** records also have multiple subsequent whitespaces and trailing whitespace.
    a. Use the common transformations on the **name** column to do the following: remove leading and trailing white space (applies to *9045 cells*), collapse consecutive white space (applies to *6415 cells*), and transform to title case (applies to *284606 cells*)
    b. The name records are now generally cleaned. The steps to clean white space may need to be performed again after cleaning characters or clustering the names.

4. **FIX BLANK FIELDS.** The **lowest_price** and **highest_price** are sometimes empty. Let's go ahead and fix that using some interesting transforms.
    a. First, make a numeric facet for the **highest_price**. Filter by any blank values
    b. Next, transform the remaining rows' **highest_price** with the following GREL statement: 0. (Putting simply 0 will transform the blank values to 0) (applies to *29100 cells*).
    c. Clear the current facet, and do steps ***a*** and ***b*** but for the **lowest_price** (applies to *29100 cells*).

5. **FIX BLATANT INCORRECT DATES**. Some dates in **first_appeared** and **last_appeared** have their dates in 2800 or 2900 instead of the 18-1900s. It also seems that some of the **first_appeared** dates are after the **last_appeared** dates which doesn't make much sense.
    a. Fix the incorrect dates for both the **first_appeared** and **last_appeared** columns. The following GREL statement will look for any dates starting with a number greater than 1, and change it to 1. This makes sense since the timeframe for the menus seems to be the 1800s to 1900s, and not into the 2000s
    b. GREL: if (value.toNumber() > cells['last_appeared'].value.toNumber(), cells['last_appeared'].value.toNumber(), value) (applies to *2228 cells for **first_appeared***) (applies to *2652 cells for **last_appeared***)
    c. Do another GREL transform on the **first_appeared** column: if (value.toNumber() > cells['last_appeared'].value.toNumber(), cells['last_appeared'].value.toNumber(), value) (applies to *259 cells*)

6. **FILL IN BLANK DATES*.*** Some dates **first_appeared** are 0 or 1. These can be changed pretty easily by setting it equal to the **last_appeared** if this is the case.
    a. Use the following GREL statement: if(value.toNumber() < 2, cells['last_appeared'].value, value) (applies to *177 cells*)
    b. Now any **first_appeared** dates that were 0 or 1 are the same value as **last_appeared**

At this point, the file is quite clean. Surprisingly, the names didn't need to be cleaned and much of the cleaning required was making sure the data had some integrity and made sense.

2.2. Menu.csv

Because many of the strings needed cleaning, the number of cells that were cleaned may be more or less based on the order in which the cleaning is done.

1. **REMOVE EXTRANEOUS COLUMNS**. A few columns contain little to no data or are not relevant.

    a. Remove the following columns: **keywords**, **language**, and **location_type**.
    b. Whenever there is a menu **name**, the same string is repeated in the **location** and **sponsor** columns.
        i. Remove the **name** and **location** columns
        ii. Rename the **sponsor** column to be **name**
2. **STRING GENERAL CLEANING**. Columns with strings should be cleaned by ensuring consistent capitalization and removing any trailing, leading, or subsequent white spaces
    a. Consistent capitalization across multiple columns is important.
        i. Transform the **name** (applies to *8872 cells*) and **place** (applies to *7338 cells*) columns to title case.
        ii. Transform the **event** (applies to *7777 cells*) and **venue** (applies to *8110 cells*) to lower case.
    b. Any trailing or leading whitespaces should be removed from the following columns: **name** (applies to *15 cells*), **event** (applies to 3 *cells*), **venue** (applies to *0 cells*), **place** (applies to *8 cells*), **physical_description** (applies to *384 cells*), **occasion** (applies to *0 cells*), and **notes** (applies to *125 cells*)
    c. Any subsequent whitespace should be removed from the following columns: **name** (applies to *127 cells*), **event** (applies to *6 cells*), **venue** (applies to *0 cells*), **place** (applies to *45 cells*), **physical_description** (applies to *38 cells*), **occasion** (applies to *3 cells*), and **notes** (applies to *195 cells*)

3. **FIX EVENTS.**
    a. Use a text facet to cluster the **event** column. Combine similar events into similar clusters to make the data look more pretty.
    b. On the **event** column use the following GREL expressions:
        i. value.replace(‘?’,’’) (applies to *195 cells*)
        ii. value.replace(‘\”’,’’) (applies to *19 cells*)
        iii. value.replace(/\[(.+)\]/,'$1') (applies to *50 cells*)
        iv. value.replace(/\((.+)\)/,'$1') (applies to *26 cells*)
4. **FIX NAME.**
    a. Use a text facet to cluster the **name** column. Combine similar names into similar clusters to make the data look more pretty.
    b. On the **event** column use the following GREL expressions:
        i. value.replace(‘?’,’’) (applies to *118 cells*)
        ii. value.replace(‘\”’,’’) (applies to *369 cells*)
        iii. value.replace(/\[(.+)\]/,'$1') (applies to *27 cells*)
        iv. value.replace(/\((.+)\)/,'$1') (applies to *144 cells*)
    c. After the clustering/removing of bad characters, some of the names need to be updated to title case again. Transform the column to title case

5. **FIX PLACE**.
    a. On the **place** column, use the following GREL expressions:
        i. value.replace(‘\”’,’’) (applies to *499 cells*)

ii. value.replace(/(,(?=\S))/, ', ') (applies to *919 cells*)
iii. value.replace(/\[(.+)\]/,'$1') (applies to *847 cells*)
iv. value.replace(/\((.+)\)/,'$1') (applies to *225 cells*)
v. value.replace('?', '') (applies to *380 cells*)
vi. value.replace(/^(.+);/, '$1') (applies to *377 cells*)

6. **REMOVE TRAILING SEMICOLONS FROM COLUMNS**. Transform **venue** (applies to *1898cells*), **occasion** (applies to *2331 cells*), **physical_description** (applies to *12334 cells*), and **notes** (applies to *7076 cells*) using the following GREL to remove trailing semicolons: value.replace(/^(.+);/,'$1')

At this point, the menu data is clean enough to work with and pass on to the next step. There are a few integrity constraints that will need to be checked later on.

3. Developing a relational schema

id	name	description	menus_appeared	times_appeared	first_appeared	last_appeared	lowest_price	highest_price

Dish (
id int,
name varchar(50),
menus_appeared int,
times_appeared int,
first_appeared int,
last_appeared int,
lowest_price double,
highest_price double
)

id	name	sponsor	event	venue	place	physical_description	occasion	notes	call_number	date	location	currency	currency_symbol	status	page_count	dish_count

Menu (
id int,
name varchar(50),
event varchar(50),
venue varchar(50),
place varchar(50),
physical_description varchar(50),
occasion varchar(50),

```
    notes varchar(50),
    call_number varchar(20),
    date text,
    currency varchar(50),
    currency_symbol varchar(50),
    status varchar(50),
    page_count int,
    dish_count int
)
```

Loading the Data

The data was loaded directly from the SQL command line. First. the table was created using the schema described above, after which the data was then loaded into the schema and ready to be validated according to the ICs present in this section. A row that violates an integrity constraint will be removed.

Menu

```
sqlite> create table Menu (id int,name varchar(50),event varchar(50),venue varchar(50),place varchar(50),physical_description varchar(50),occasion varchar(50),notes varchar(50),call_number varchar(20),date text,currency varchar(50),currency_symbol varchar(50),status varchar(50),page_count int,dish_count int);
sqlite> .mode csv
sqlite> .import C:/Users/Nitin/Documents/git/DataCleaning-FinalProject/part3/Menu_IC_1.csv Menu
```

Dish

```
sqlite> create table Dish (id int,name varchar(50),menus_appeared int,times_appeared int,first_appeared int,last_appeared int,lowest_price double,highest_price double);
sqlite> .mode csv
sqlite> .import C:/Users/Nitin/Documents/git/DataCleaning-FinalProject/part3/Dish_IC_3.csv Dish
```

Dish

1. Have a name	SELECT * from Dish d WHERE d.name IS NULL OR d.name =’’;
2. Should appear on at least 1 menu	SELECT * from Dish d WHERE d.menus_appeared<1;
3. Should menus_appeared < times_appeared	SELECT * from Dish d WHERE d.menus_appeared > d.times_appeared;

4. Should last_appear > first_appear	SELECT * from Dish d WHERE d.last_appeared < d.first_appeared;
5. Should highest_price >= lowest_price	SELECT * from Dish d WHERE d.highest_price < d.lowest_price;

Menu

1. Should have name	SELECT * from Menu m WHERE m.name IS NULL OR m.name='';
2. Page Count > 0	SELECT * from Menu m WHERE m.name IS NULL OR m.page_count='';
3. Dish Count > 0	SELECT * from Menu m WHERE m.name IS NULL OR m.dish_count='';

*** We wanted to enforce the constraint that Dish Count >= Page Count > 0, but found that there were lots of pages that violated this. We reasoned that a very creative menu could indeed have more pages than menu items, and relaxed this integrity constraint**

Files

Dish_IC_1.csv - Dish table removed of entries failing IC 1

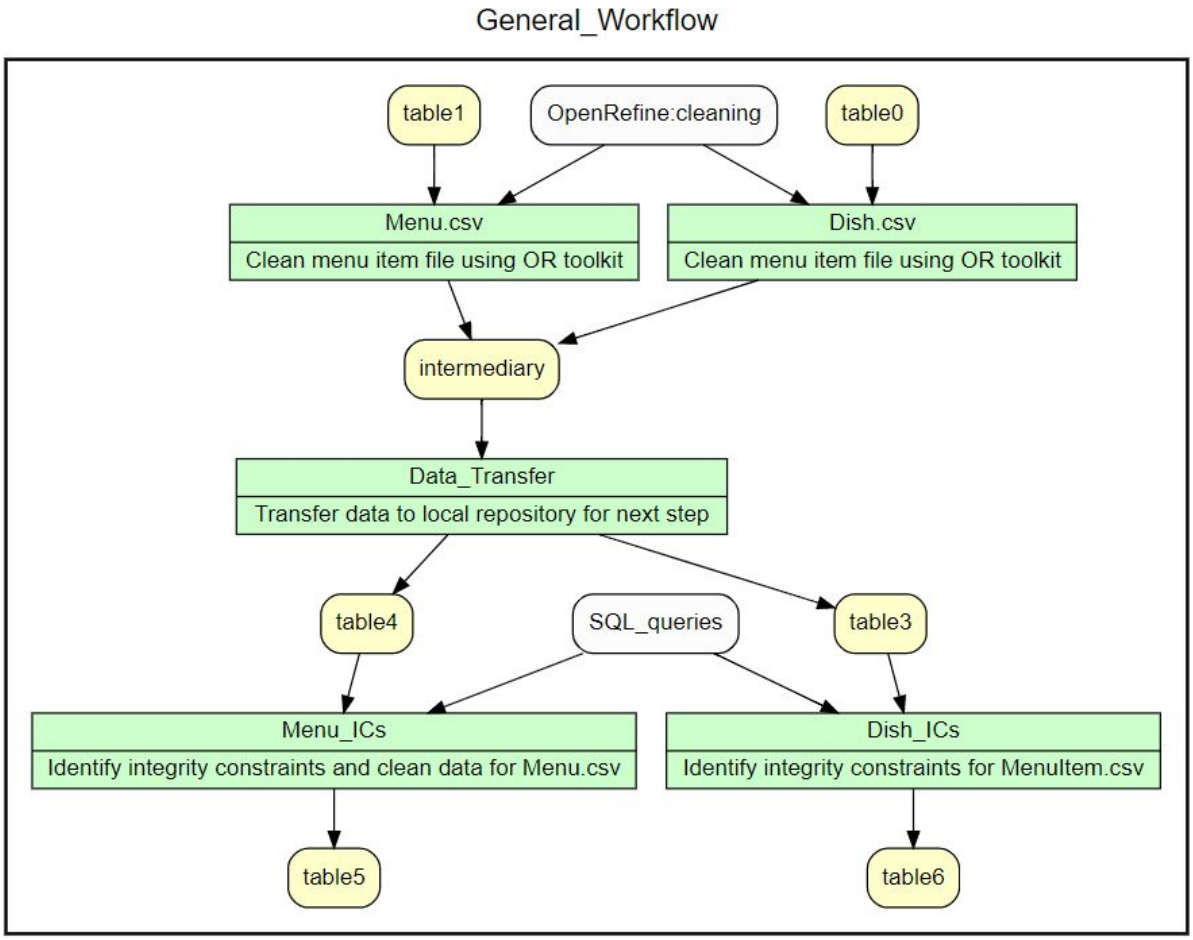
Menu_IC_2.csv - Menu table removed of entries failing IC 2

Menu_IC_3.csv - Menu table removed of entries failing IC 2 and IC 3

4. Creating a workflow model

Our general workflow including the cleaning and creating integrity constraints is shown below. The workflow was generated using YesWorkflow. The OpenRefine history workflows (and the one below) will be included in a separate deliverables folder for better resolution and the ability to zoom in. The two images of the OpenRefine workflows could not be added to the document

due to their large size. Decreasing resolution or cropping would have led to unreadable images or images with missing text. Using our prerogative, we decided to only include those in the deliverables.



5. Takeaways and Challenges

One of the biggest challenges to cleaning the data set was remembering to check that we couldn't just look at the first 50 records as examples of bad data. The data for the menus spanned across probably years of data collection. For example, in some of the earliest dish records, one of the columns was always blank. But for the last records, they were almost always filled. This meant the column shouldn't be removed. This challenge provided us with a takeaway for this project: dealing with a large dataset that spans over a long period of time means that you have to check for differences in the data at different spots in the data.

A challenge in enforcing ICs is that Menu and Dish are linked by MenuItems. Our team consists of only two members so we didn't get to clean MenuItems. This means that we couldn't add more interesting ICs that check if the Dishes show up on the Menus present.

Contribution

bordway2 - Initial Assessment, Open Refine, YesWorkflow, Challenges and Takeaway
tangell2 - Schema, Integrity Constraints, Challenges and Takeaway