CS 513 Theory & Practice of Data Cleaning

Final Project Report: What's On The Menu?

Artsiom Strok
NetID: astrok2
astrok2@illinois.edu

Ramin Melikov
NetID: melikov2
melikov2@illinois.edu

Jonah Willis
NetID: jonahww2
jonahww2@illinois.edu

***Abstract*— Data cleansing is one of the most important and most time-consuming steps in data science projects. The final project aims to use various tools and techniques covered in this course, together in an end-to-end data cleaning workflow. All knowledge obtained in this course will be applied for New York Public Library's crowd-sourced historical menus dataset.**

Keywords—NYPL, menus, data cleansing, OpenRefine, YesWorkflow, SQL, provenance.

I. Overview and Initial Assessment of the Dataset

The New York Public Library is digitizing and transcribing its collection of historical menus. The collection includes about 45,000 menus from the 1840s to the present, and the goal of the digitization project is to transcribe each page of each menu, creating an enormous database of dishes, prices, locations, and so on. As of early July, 2020, the transcribed database contains 1,334,431 dishes from 17,545 menus.

This dataset is split into four files to minimize the amount of redundant information contained in each (and thus, the size of each file). The four data files are *Menu, MenuPage, MenuItem*, and *Dish*. These four files are described briefly here, and in detail in their individual file descriptions below.

A. Menu

The core element of the dataset. Each Menu has a unique identifier and associated data, including data on the venue and/or event that the menu was created for; the location that the menu was used; the currency in use on the menu; and various other fields.

- id – The unique identifier of the menu
- name – The name of the restaurant
- sponsor – Who sponsored the meal (organizations, people, name of restaurant)
- event – The category (e.g. lunch, annual dinner)
- venue – The type of place (e.g. commercial, social, professional)
- place – Where the meal took place (often a geographic location)
- physical_description – The dimension and material description of the menu
- occasion – The occasion of the meal (holidays, anniversaries, daily)
- notes – The notes by librarians about the original material
- call_number – The call number of the menu
- keywords – The keywords of the menu
- language – The language of the menu
- date – The date of the menu
- location – The organization or business who produced the menu
- location_type – The type of the location
- currency – The system of money the menu uses (dollars, etc.)
- currency_symbol – The symbol for the currency ($, etc.)
- status – The completeness of the menu transcription (transcribed, under review, etc.)
- page_count – How many pages the menu has
- dish_count – How many dishes the menu has

Each menu is associated with some number of MenuPage values.

The first inspection of the data shows us that this file has 17545 entries and 20 columns.

Three columns '*keywords*', '*language'*, '*location*_type' do not have any values and can be deleted.

The 'id' column has all unique numeric values. Thus we can assume no issues with this column.

The '*name*' column has only 3197 non-empty values. There also placeholders for missing value, e.g., *'[Restaurant name and/or location not given]*' or *'[Not given]*'. There are a lot of names that exactly the same but due to extra spaces, punctuations, different order of words, typos, diacritical mars they don't match exactly.

The '*sponsor*' column has 15984 non-empty values, and these values have similar issues as '*name*' column. Also, some of the values are just question marks.

The '*event*' column has 8154 non-empty values. The values for this column can be grouped into different buckets such as '*breakfast*', '*lunch*', '*dinner*' etc. Also, some of these values are written in different language e.g., French or German, and depends on the use case can be grouped together. The values such as '*107th, 108th ... anniversary dinner*' can be grouped together as just '*anniversary dinner'*. Each value can have multiple categories e.g., '*lunch and dinner'*, which also can be post-processed based on the use case.

The '*venue*' column has 8119 non-empty values. The values in this column have the most of common issues, including question marks, extra punctuations, etc., and new unique issues with abbreviations e.g., '*SOC*' and '*SOCIAL*', '*COM*' and '*COMMERCIAL*'. In addition, this column can also have multiple categories within one value. The '*place*' column has 8123 non-empty values. And again, besides common issues, this column has an issue with partial values. The value can represent just the name of the place or place and city or address line, city and state, etc.

The '*physical_description*' column 14763 non-empty values. There as some *'#N/A*' values. Each value in this column has multiple sub-values such as type of menu e.g. '*booklet*', '*card*', '*folder*' and physical dimensions of the menu e.g. '*5.75 X 7.25*', '*5 X 8*' and some unique features of the menu e.g. with or without illustration, regular or column layout, folded or open. And this column can have multiple variations of such properties within one value.

The '*occasion*' column has 3791 non-empty values. The values of this column also can be grouped into multiple buckets.

The '*notes*' column has 10613 non-empty values. The values in this column mostly represented by paragraphs of free text, mostly unstructured. Depends on the use case, additional features can be derived for this column.

The '*call_number*' column has 15983 non-empty values. The majority of values in this column are numeric with some OCR-like issue e.g. we see '*o*' instead of '*0*', or '*l*' instead '*1*'. Some of them have postfixes such as '*item*', '*_wotm*', '*copy*'. And some of them starting from the word and continuing with a number, e.g. '*Zander 645*', '*Soete 162*', '*Baratta 35*'.

The '*date*' column has 16959 non-empty values. And only three values where there are some issues with the year and can be easily detected using timeline facet from OpenRefine. The 'location' column does not have empty values. However, there are values such as question mark. The issues are similar to the issues with '*name*' or 'sponsor' columns.

The columns '*currency*' and '*currency_symbol*' both have 6456 non-empty values, and they look good. Some preprocessing can be done for cents because it can be cents of different currency.

The '*status*' column has all values available and does not have any issues.

The '*page_count*' and '*dish_count*' columns also have all values available. There are some extreme values that need to be analyzed.

B. MenuPage

Each MenuPage refers to the Menu it comes from, via the menu_id variable (corresponding to _Menu_id). Each MenuPage also has a unique identifier of its own. Associated MenuPage data includes the page number of this MenuPage, an identifier for the scanned image of the page, and the dimensions of the page.

- id – The unique identifier of the menu page
- menu_id – The unique identifier of the menu, corresponds to Menu id
- page_number – The number representing sequence of page in the menu
- image_id – The unique identifier of the page image
- full_height – The height of the page image in pixels
- full_width – The width of the page image in pixels
- uuid – The universally unique identifier for the highest resolution version of the image

Each MenuPage is associated with some number of MenuItem values.

The first inspection of the data shows us that this file has 66937 entries and seven columns. The *id* and *menu_id* columns seem clean on first inspection with no missing entries and a relatively uniform distribution.

The *page_number*, *full_height*, and *full_width* columns all have missing entries (1202, 329, and 329, respectively) but seem to be otherwise clean. Both *full_height* and *full_width* are missing entries in the exact same rows.

The *image_id* column presents the most issues. The values in this column are using three different formats. About half of the entries are using 7-digit numeric IDs, another half are using 10-digit numeric IDs, and a few (23) of the values are using alpha-numeric IDs.

Finally, the *uuid* column was almost entirely clean, only one entry needed to be updated to use lower-case letters. It is worth noting that some uuids are duplicated.

C. MenuItem

Each MenuItem refers to both the MenuPage it is found on -- via the menupageid variable -- and the Dish that it represents -- via the dish_id variable. Each MenuItem also has a unique identifier of its own. Other associated data includes the price of the item and the dates when the item was created or modified in the database.

- id – The unique identifier of the menu item
- menu_page_id – The unique identifier of the page the menu item is on, corresponds to MenuPage id
- price – The first price of menu item
- high_price – If the item has more than on price on a single menu, the highest price. If there are more than two values for price, the web application instructs volunteers to enter the lowest and highest prices rather than all values.
- dish_id – The unique identifier of the dish, corresponds to Dish id
- created_at – The date/time of the first transcription
- updated_at – The date/time of the last edit to the value
- xpos – The horizontal coordinate on the page for the upper left point where menu item is on the page
- ypos – The vertical coordinate on the page for the upper left point where the menu item is on the page

The first inspection of the data shows us that this file has 1332726 entries and nine columns. The *id*, *menu_page_id*, *dish_id*, *created_at*, *updated_at, xpos,* and *ypos* columns seem clean on first inspection.

The *price* column 445916 blank rows. It is also worth noting that there are 130 rows with extremely high (over $10000) prices.

The *high_price* column has 1240821 blank rows, which means that the vast majority of the rows are blank. It may be worth excluding this column.

D. Dish

A Dish is a broad category that covers some number of MenuItems. Each dish has a unique id, to which it is referred by its affiliated MenuItems. Each dish also has a name, a description, a number of menus it appears on, and both date and price ranges.

- id – The unique identifier of the dish
- name – The name of dish
- description – The description of the dish
- menus_appeared – The total count of menus on which dish with this id appears
- times_appeared – The total count of appearances of the dish with this id across all menus
- first_appeared – The earliest year of a menu on which a dish with this id appears
- last_appeared – The latest year of a menu on which a dish with this id appears
- lowest_price – The lowest price associated with a dish with a given id
- highest_price – The highest price associated with a dish with a given id
-

http://menus.nypl.org/about

https://www.kaggle.com/nypl/whats-on-the-menu

The '*Dish.csv*' file has data about dishes that have appeared or didn't appear in menus since 1851. In particular, it contains data about the dish's name, its description, how many times and in how many menus it appeared, when it appeared first and last, and its lowest and highest price. This dataset contains 423,397 observations of 9 variables. The variables are as follows:

- id – The unique identifier
- name – The name of the dish
- description – The description of the dish
- menus_appeared – The number of menus in which this dish appeared
- times_appeared – The number of times this dish appeared
- first_appeared – The year when this dish appeared first
- last_appeared – The year when this dish appeared last
- lowest_price – The lowest known price for this dish
- highest_price – The highest known price for this dish

Upon initial analysis, some data quality issues have been discovered. First, the 'description' column has no values in it at all. And at first thought one would think that perhaps it should be removed. However, when we analyzed the 'name' column, we have discovered that there are 9,125 rows where the name column most likely contains the description of the dish because the length of the text is over 100 characters and most of the names are under 100 characters. This is an issue because if we move the text of these rows from the 'name' column into the 'description' column, then how do we fill the 'name' column? One way to do it would be to go through all of the descriptions and then try to deduce a name for the dish from the description but this would require quite a bit of work. One possible solution would be to move the text into the description column and fill the 'name' column with the '*Unknown*' value. However, when we think of it in parts of the dataset, 9,125 represents just 2% of the data. That is, only 2% of the data has a description. In that case, does a description have value if it only 2% of the data has it? Maybe this could be normalized further and perhaps there is a substructure there, but it is hard to tell. Furthermore, there are 48,311 duplicates in the '*name*' column. Should they be excluded and if so, what about the statistics they contain?

The '*times_appeared*' column has some negative values, suggesting that some dishes appeared '*-2*' times in some menus while others '*-6*', etc. There are also some '*0*'s in there as well. One recommendation would be to lump all of the values less than or equal to 0 into the '*0*' group. However, there is another problem with this column. For many of the '*0*' values, there are values of '*0*', '*1*', '*2*', and '*3*' in the '*menus_appeared*' column. This doesn't make sense. How could a dish appear 0 times but appear 1 time in a menu? Maybe these 2 columns could be merged into 1?

There are also issues between the '*first_appeared*' and '*last_appeared*' columns. First, in the '*first_appeared*' column we have some values that don't fit in. They are '*0*', '*1*', and '*2928*'. All other values in this column fall in the range between 1851 and 2012. There is a same problem in the '*last_appeared*' column. Furthermore, upon some testing of the values, we've discovered that some values in the '*first_appeared*' column are greater than those in the '*last_appeared*' column. Granted, there aren't that many that comprise this violation.

And finally, there are some issues in the '*lowest_price*' and '*highest_price*' columns. Mainly, we have some quite a bit of blank rows there for each column. However, we've test to see if there are any violations in the data like '*lowest_price*' greater than '*highest_price*' or if there are values in another while there are blanks in one but there weren't such violations here.

Some use cases for this data would be in the space of restaurant entrepreneurs. Before a restaurant offers a dish, it could look at this dataset to see if a similar dish has been in the menus and how popular it is, as well as the price at which it's been offered throughout the years.

Other use cases would be in the space of journalists or other researchers who are doing research on some dishes. It would be interesting to see if there are any forgotten dishes that perhaps could be revived.

II. Data Cleaning with OpenRefine

We use OpenRefine version 3.1 to clean the dataset. Each file was cleaned separately.

A. *Menu file*

The '*id*' column was converted to the number using common transformation '*To Number*'. See FIGURE 1 All 17545 were converted without any issues as expected.

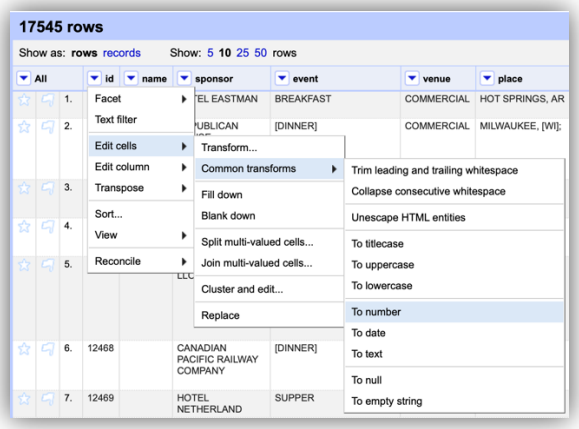

FIGURE 1 COMMON TRANSFORM FOR NUMBER

Common transformations such as '*Trim leading and trailing whitespace'*, '*Collapse consecutive whitespace*', '*To titlecase*' and in some cases '*To uppercase*' were applied for each text column. The same transformations were applied repeatedly when other transformations may cause the issues fixed before.

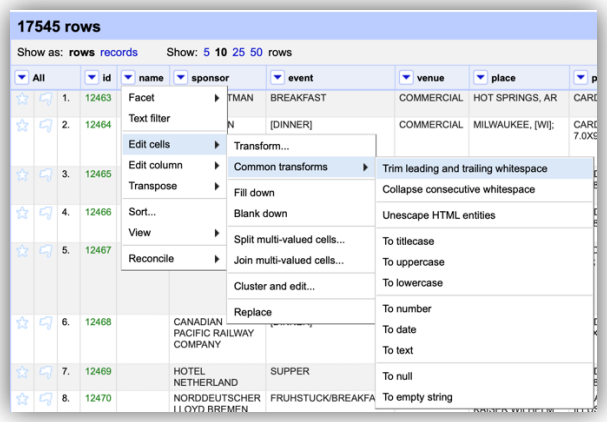

FIGURE 2 COMMON TRANSFORMS FOR TEXT

In the next step, we applied the text filter '*[^\w\s]*' with '*regular expression*' checkbox checked for each column to identify values containing the non-word or special characters '*"&[]()}{?%#!/*'. Later all special characters were removed by GREL expression '*value.replace(/["&()}{?%#!\/\[\]]/, "")*'.

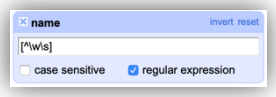

FIGURE 3 TEXT FILTER TO KEEP VALUES WITH SPACIAL CHARACTERS

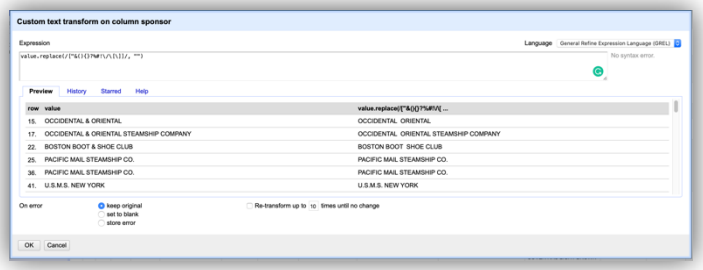

FIGURE 4 CUSTOM TEXT TRANSFORM USING GREL TO REMOVE SPECIAL CHARACTERS

The rest of the punctuations were replaced by space. Common transformations to remove leading and trailing spaces, and collapse consecutive spaces, were applied.

The values *'[Restaurant name and/or location not given]'* or *'[Not given]'* were replaced by empty value.

All diacritic characters were mapped to ASCII characters using standard mapping e.g., '*á*' mapped to '*a*', '*é*' to '*e*'.

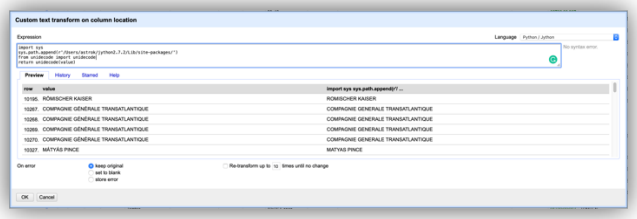

FIGURE 5 JYTHON SCRIPT TO REMOVE DIACRITIC MARKS

The most of transformations were done using the text facet and cluster feature of OpenRefine. The *'key collision'* method with all four key functions '*fingerprint*', *'ngram fingerprint*', '*metaphone3*', *'cologne-phonetic*', and *'nearest neighbor*' method with '*levenshtein*' and '*PMM*' distance functions with default parameters were used to find and group values.

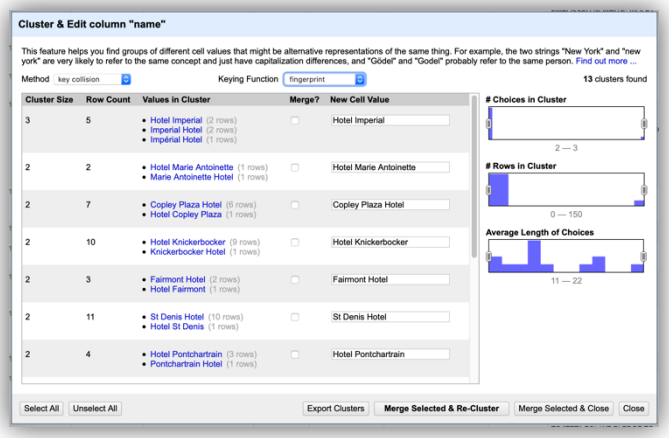

FIGURE 6 CLUSTERING OF TEXT VALUES

The '*date'* column was converted to date format. Two outliers were identified using timeline facet and corrected manually by looking at original images.

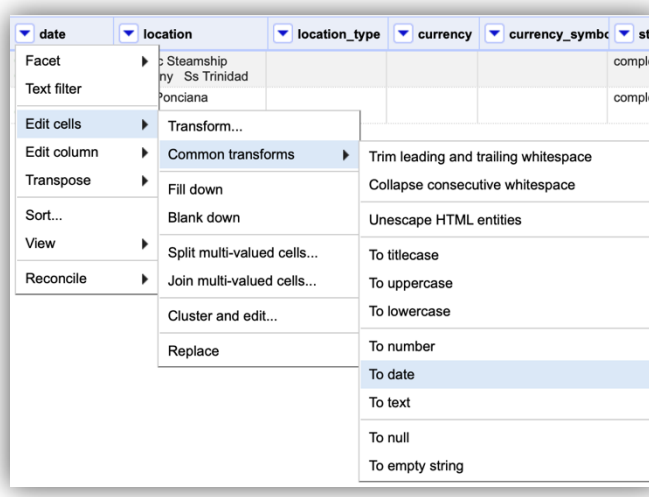

FIGURE 7 COMMON TRASFORM FOR DATE

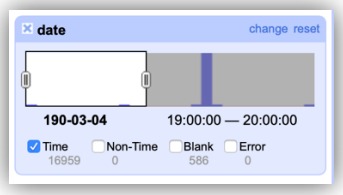

FIGURE 8 TIMELINE FACET TO DETECT OUTLIERS

Column	Cells Affected
name	1481
sponsor	11375
event	1448
venue	2551
place	6189
physical_description	14763
occasion	2616
notes	10613
call_number	15981
keywords	0
language	0
date	16959
location	17544
location_type	0
currency	0
currency_symbol	0
status	0
page_count	0
dish_count	0

The '*physical_description*' was split into multiple columns

- *physical_description_type*
- *physical_description_emblem*
- *physical_description_folded*
- *physical_description_lamindated*
- *physical_description_color*
- *physical_description_us*
- *column_structure*
- *has_illustration*
- *physical_size*

to generate more features and create the structure for the menu's physical properties.

For more details see '*Open_Refine_History-Menu.json' file.*

B. Menu Page

The '*id*', *'menu_id'*, *'page_number'*, '*full_height'*, and '*full_width'* columns were converted to numbers using common transformation '*To Number*'. Refer to FIGURE 1. All 66937 rows were converted without any issues for the '*id*' and *'menu_id' columns.*

The *page_number*, *full_height*, and *full_width* columns all have some blank rows (1202, 329, and 329, respectively) but otherwise all entries in these columns are valid numeric values. No further cleaning besides the initial *'To Number'* transform was done for these columns.

As mentioned in Section I, the *'image_id'* column has IDs in three different formats, 7-digit numeric, 10-digit numeric, and alpha-numeric. Although it would be ideal to have all IDs in a consistent format, fortunately the IDs were all unique. Therefore, no cleaning was done for this column.

The *'uuid'* column was cleaned using the *'To Lowercase'* transformation which affected one row. See FIGURE 9.

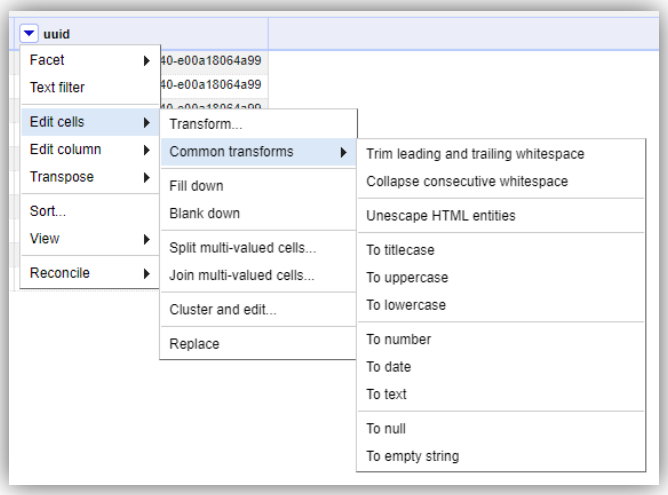

FIGURE 9 COMMON TRANSFORM TO LOWERCASE

Column	Cells Affected
id	66937
menu_id	66937
page_number	65735
image_id	66914
full_height	66608
full_width	66608
uuid	1

FIGURE 10 SUMMARY OF AFFECTED CELLS FOR MENU PAGE

C. Menu Item

The '*id'*, *'menu_page_id'*, *'price'*, *'high_price'*, *'dish_id'*, *'created_at'*, *'updated_at'*, *'xpos'*, and *'ypos'* were converted

to numbers using the common transformation *'To Number.'* Refer to FIGURE 1. All rows in the '*id*', '*menu_page_id*', '*dish_id*', '*created_at*', '*updated_at*', *'xpos'*, and '*ypos'* columns were converted to numeric values without issue. The *'price'* and *'high_price'* columns each had a significant number of blank rows (445916 and 1240821, respectively). No further cleaning besides the *'To Number'* transform was done for these columns.

Column	Cells Affected
id	1332726
menu_page_id	1332726
price	886810
high_price	91905
dish_id	1332485
created_at	0
updated_at	0
xpos	1332726
ypos	1332726

FIGURE 11 SUMMARY OF AFFECTED CELLS FOR MENU PAGE

III. DEVELOPING A RELATIONAL SCHEMA

We used SQLite (ver. 3.31.1) as DBMS and DB Browser for SQLite (ver. 3.12.0) as a visual editor. First, we created a DB schema with 4 tables and then imported cleaned CSV files.

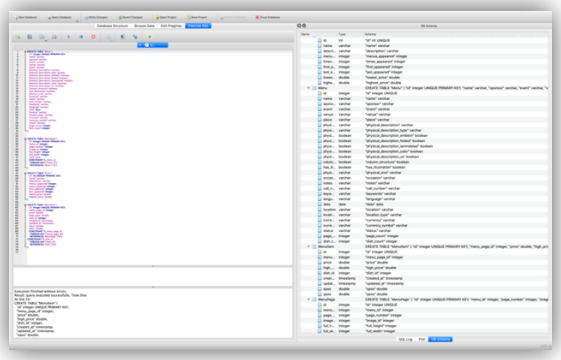

FIGURE 12 EXECUTION OF THE DDL SCRIPT IN DB BROWSER

The DDL script contains three additional constrains, foreign keys linking '*Menu*' with '*MenuPage*' table, '*MenuPage*' with '*MenuItem*', and finally '*MenuItem*' to '*Dish*'.

- ALTER TABLE "MenuPage" ADD FOREIGN KEY ("menu_id") REFERENCES "Menu" ("id");
- ALTER TABLE "MenuItem" ADD FOREIGN KEY ("menu_page_id") REFERENCES "MenuPage" ("id");
- ALTER TABLE "MenuItem" ADD FOREIGN KEY ("dish_id") REFERENCES "Dish" ("id");

Please, for additional details, see the '*database-schema.sql*' file.

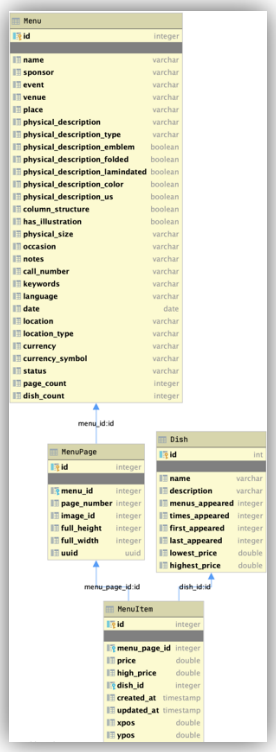

FIGURE 103 UML DIAGRAM OF ER SCHEMA

We created additional SQL queries to profile and check the integrity constraints of the data.

A. Dish table

The first query to check how many dishes we have in the database, which never appeared on any of the menus.

select count(*) from Dish as d where d.menus_appeared == 0;

Another similar query to the previous, but this time we check the total count of appearances.

select count(*) from Dish as d where d.times_appeared == 0;

The next query to check how many dishes do not appear on the menu, but the total number of times they appeared is greater than zero.

select count(*) from Dish as d where d.menus_appeared == 0 and d.times_appeared > 0;

The next query to check how many dishes do not appear on the menu, but we have the record of either the first time or the last time it appeared.

select count(*) from Dish as d where d.times_appeared == 0 and (d.first_appeared > 0 or d.last_appeared > 0);

The next query checks whether we have records where the highest prices is greater than the lower price.

select count(*) from Dish as d where d.highest_price < d.lowest_price;

The next query checks whether we negative prices in the database.

select count(*) from Dish as d where d.lowest_price < 0 or d.highest_price < 0;

The next query checks how many records we have where the dish's first year of appearance is out of range when these menus possibly collected.

select count(*) from Dish as d where d.first_appeared < 1850 or d.first_appeared > 2020;

The next query checks inconsistences in tracking of the year when dish is appeared.

select d.first_appeared, d.last_appeared from Dish as d where (d.first_appeared == 0 and d.last_appeared != 0) or (d.first_appeared != 0 and d.last_appeared == 0);

The next query shows us any dishes that not linked to the menu.

select count(*) from Dish as d where d.id not in (select mi.dish_id from MenuItem as mi);

B. Menu table

The query checks that status values in the list of valid options.

select m. from Menu as m where m.status not in ('complete', 'under review');*

The next query show us the number of menus outside of range of the possible date of the menu.

select m. from Menu as m where m.date not between '1850-01-01' and date('now');*

The next query counts number of menus with zero number of dishes.

select m. from Menu as m where m.dish_count == 0;*

The next query is soft check to manually review possible outliers in terms of dish count.

select m. from Menu as m where m.dish_count > 1000;*

The next query counts number of menus with zero number of pages.

select m. from Menu as m where m.page_count == 0;*

The next query is soft check to manually review possible outliers in terms of page count.

select m. from Menu as m where m.page_count > 50;*

The next query checks the menus not linked to menu pages.

select m. from Menu as m where m.id not in (select mp.menu_id from MenuPage as mp);*

C. Menu item

The query to check how many menu items updated even before they were created.

select count(*) from MenuItem as mi where mi.created_at > mi.updated_at;

The next query shows how many menu items have negative price.

select count(*) from MenuItem as mi where mi.price < 0;

The next query counts how many menu items have the price greater than the highest price.

select count(*) from MenuItem as mi where mi.price > mi.high_price and mi.high_price is not null;

We apply the next query for both 'created_at' and 'updated_at' columns to see how many items fall out of the range of dates when data was possibly created.

select mi. from MenuItem as mi where mi.created_at not between '1850-01-01' and date('now');*

We apply the next query for both x position and y position of the item to see if we have any records with negative coordinates.

select count(*) from MenuItem as mi where mi.xpos < 0;

We apply the next query for both coordinates to see if they out of the physical resolution of the image they appeared.

*select * from MenuItem as mi inner join MenuPage mp on mi.menu_page_id = mp.id where mi.xpos > mp.full_width;*

D. Menu page

The query to see if we have any pages with negative page number.

select count(*) from MenuPage as mp where mp.page_number < 0;

The next query shows us menu pages that have page number higher than actual number of pages on the menu.

select m.page_count, mp.page_number from MenuPage as mp inner join Menu as m on mp.menu_id = m.id where mp.page_number > m.page_count;

The next two queries check the pages with negative width and height respectively.

select count(*) from MenuPage as mp where mp.full_height < 0;

select count(*) from MenuPage as mp where mp.full_width < 0;

The next query checks whether we have pages not linked to menu items.

select count(*) from MenuPage as mp where mp.id not in (select mi.menu_page_id from MenuItem as mi);

Please see '*queries.sql*' for additional details and queries.

IV. Creating a Workflow Model

The key inputs for the overall workflow in FIGURE 14 (see *overall_workflow.png* for full size picture) are the four CSV files that comprise the NYPL Menu dataset – *Menu.csv, MenuPage.csv, MenuItem.csv, and Dish.csv.* Within these files there are a few foreign key dependencies to be aware of that serve to link the files to each other. Refer to FIGURE 13 to see the linkages among the tables. The key outputs of the workflow are the cleaned copies of these files which are generated after going through the various steps in the workflow. The overall workflow consists of two main phases, data cleaning with OpenRefine and integrity constraint verification in SQL.

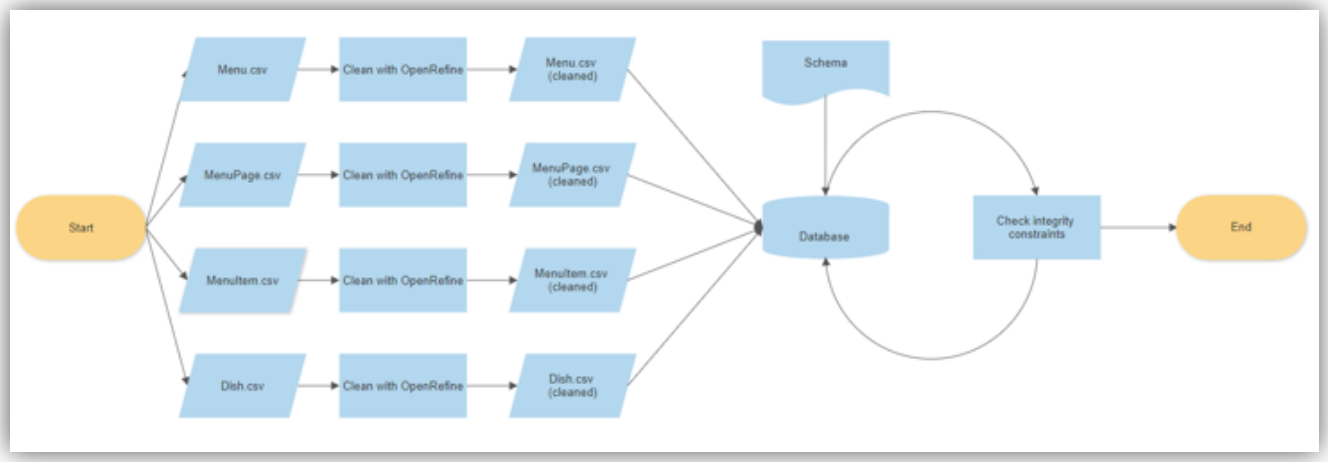

Figure 14 Overall Workflow

For the OpenRefine workflows, the key inputs are the columns in each table and the key outputs are the resulting cleaned columns and any new columns that were created from the existing data. Refer to Section II of this report for more specific details.

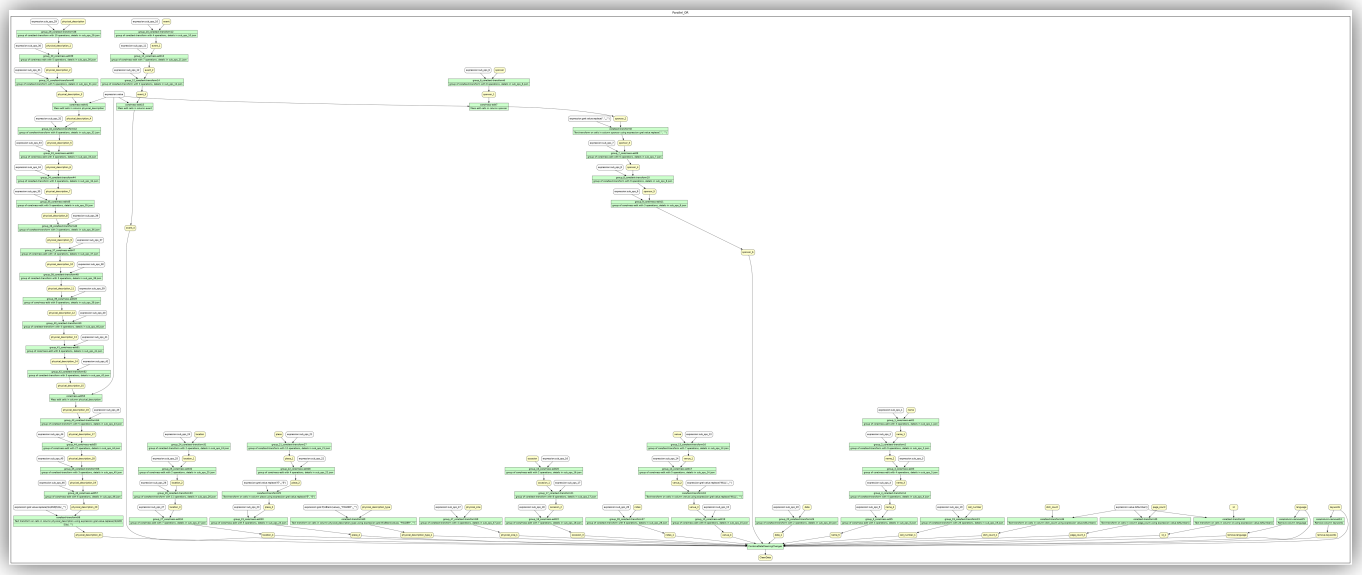

Figure 15 OpenRefine Menu Workflow

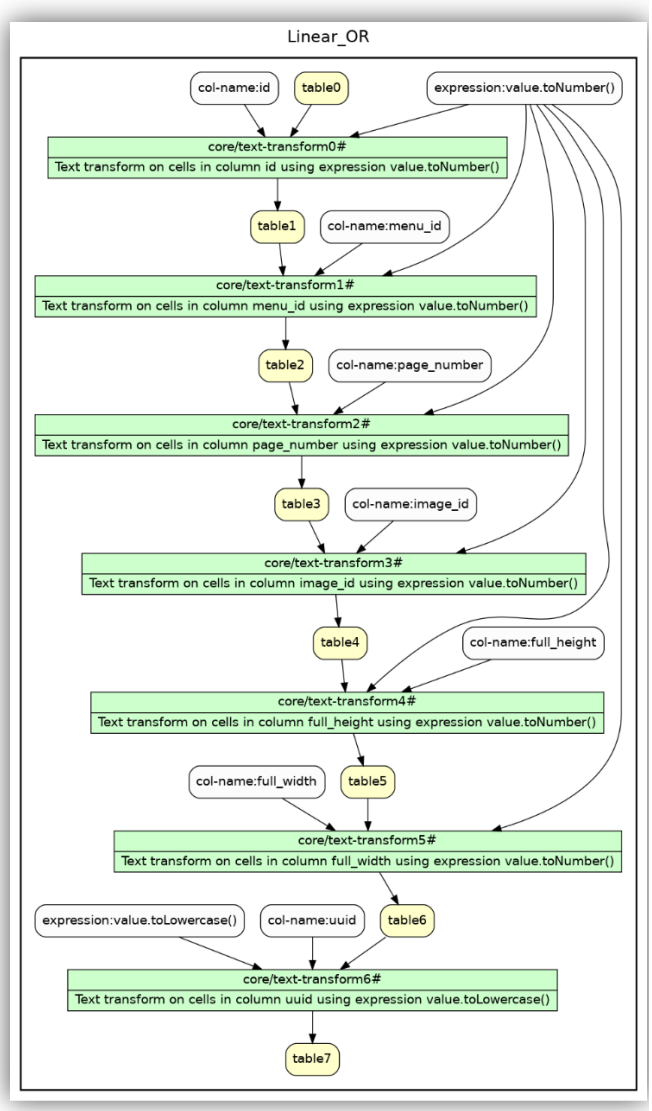

FIGURE 16 OPENREFINE MENUPAGE WORKFLOW

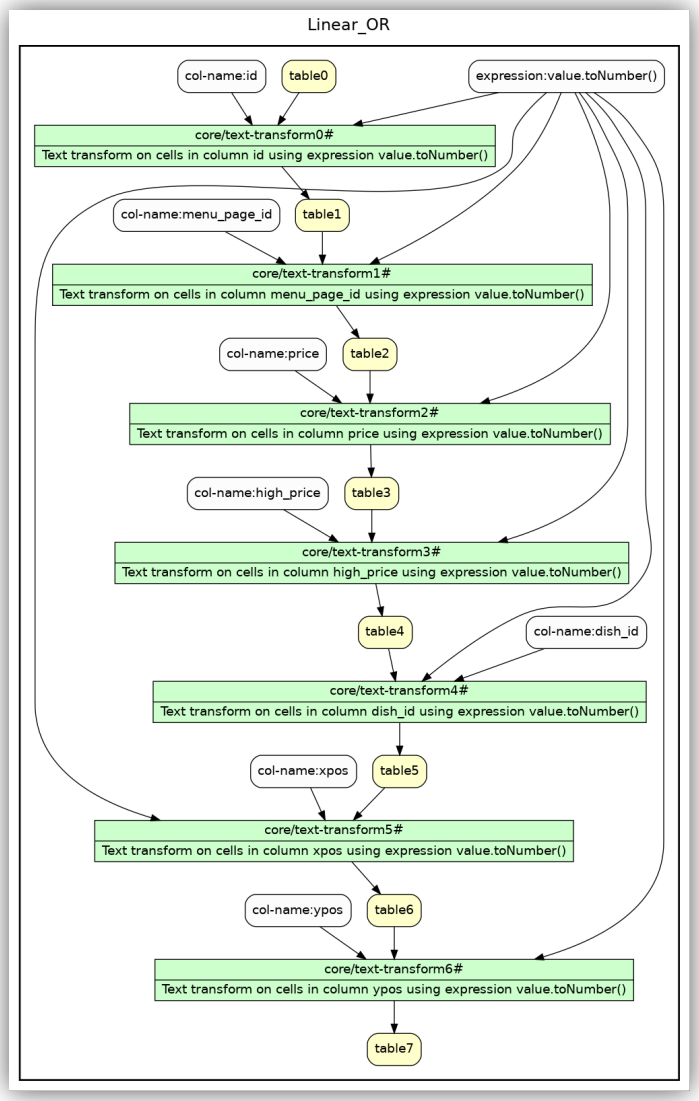

FIGURE 17 OPENREFINE MENUITEM WORKFLOW

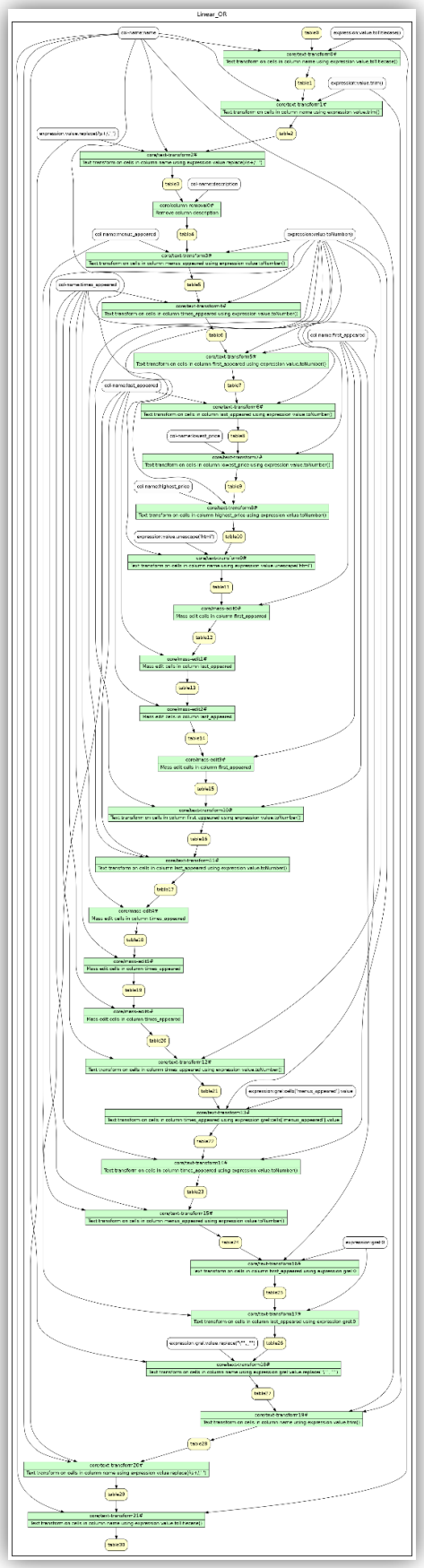

FIGURE 18
OPENREFINE DISH WORKFLOW

Refer to *Open_Refine_History_Menu.png*, *Open_Refine_History_MenuPage.png*, *Open_Refine_History_MenuItem.png*, and *Open_Refine_History_Dish.png* for full size images of these workflows.

V. DEVELOPING PROVENANCE

We did generate several `.lp` files (orhd.lp, orhm.lp, orhmi.lp, orhmp.lp) which serve as data representations of provided workflows graphs. Here is a sample of what those files look like:

```
% The edge facts edge(A, B)
edge('core/text-transform0#', 'table1').
edge('col-name:name', 'core/text-transform0#').
edge('expression:value.toTitlecase()', 'core/text-transform0#').
edge('table0', 'core/text-transform0#').
edge('core/text-transform1#', 'table2').
edge('col-name:name', 'core/text-transform1#').
edge('expression:value.trim()', 'core/text-transform1#').
edge('table1', 'core/text-transform1#').
edge('core/text-transform2#', 'table3').
edge('col-name:name', 'core/text-transform2#').
edge('expression:value.replace(/\\s+/,'_')', 'core/text-transform2#').
```

The files tell us how each node in the workflow is connected and we could query these files using Datalog queries. For Figures 16-18, the queries would be as simple as:

```
anc(X,Y) :- edge(X,Y).
answer(X) :- anc(X, node_name).
```

And, for Figure 15 the queries would be a little more complex like the following to find “ancestors” of a node:

```
anc(X,Y) :- edge(X,Y).
anc(X,Y) :- edge(X,Z), anc(Z,Y).
answer(X) :- anc(node_name,X).
```

To find common “ancestors” of a node

```
anc(X,Y) :- edge(X,Y).
anc(X,Y) :- edge(X,Z), anc(Z,Y).
ca(X,A,A) :- anc(X,A).
ca(A,X,A) :- anc(X,A).
ca(X,Y,A) :- anc(X,A), anc(Y,A), X != Y.
answer(X) :- ca(node1, node2, X).
```

However, we ran into a problem here. We discuss this problem in the “Challenges” section.

VI. CHALLENGES

There are two types of challenges we faced during the work on the project. The first one is related to ambiguity in the data and the second one is related to technical issues with software or libraries we used in the project.

We did discover a lot more data quality issues as we worked on the data. In the Dish.csv file, we found Greek, Chinese, Sanskrit, and other symbols, and quite a bit of them. All records with names written in these languages had to be removed. Furthermore, we found a lot of other issues here that could only be solved over a very long period and after a lot of tedious work. We simply couldn’t address every issue in this column within the allotted time.

When we were working on developing provenance queries in Datalog/DLV. As you can see, the node names have lots of different characters. And even though they are properly enclosed in quotes, we still had a problem with Clingo. We had 2 options here: either we redo the workflow, renaming each node so that it doesn't have all of these illegal characters, or we use use REGEX and replace each illegal character with something that would properly represent it and would maintain the workflow structure. However, this was going to be a lot of work in either case and would require a lot of time and we were running out of time to address this issue. If it were the case where Clingo didn't have these issues with characters, it would work as intended.

VII. Contribution

All three of us were involved in the whole data cleaning workflow. We divided tasks between us and cross-checked others' work to complete the project in time successfully.

Name	Major Contribution
Artsiom Strok	Project overview, initial assessment of Menu.csv file, data cleanising Menu.csv using OpenRefine, Section 3 – Developing a relational schema
Ramin Melikov	Initial assessment of Dish.csv file, data cleansing Dish.csv using OpenRefine, Section 5 – Developing provenance
Jonah Willis	Initial assessment of MenuPage.csv and MenuItem.csv files, data cleansing MenuPage.csv and MenuItem.csv using OpenRefine, Section 4 – Creating of workflow model