Theory & Practice of Data Cleaning
July 30, 2021

Team:
Julia Paladino: juliav4@illinois.edu
Miles Grogger: milesjg2@illinois.edu
Vidya Anjur: anjur2@illinois.edu

Phase 2 Report

1. Points received for initial Phase-1 submission : 93 (out of 100)
2. Our team chooses the following Phase-1 option:
    - [X] (A) No change to Phase-1 report
    - [ ] (B) Improved (or extended) Phase-1 report

Data Cleaning Performed

[Vidya]

I used OpenRefine to clean data for our group's final project. I began by opening the Menu.csv file into a separate project. Total number of rows imported was 17547.

1. I first cleaned up the "event" column to consolidate meal names (Breakfast, Lunch, Dinner, etc). Before any edits, this column was clearly very inconsistent: there were multiple representations of the same meal in different languages (Breakfast & Fruhstuck, or Dinner & Supper), as well as extra special characters (#, [,], ;, ', &, etc), inconsistent capitalization, and blank spaces. I conducted the below steps to improve data quality in the "event" column:
    a. I started by creating a duplicate column, "event_edited", and applied GREL to it to remove some special characters ("grel:value.replace('\\[','').replace('\\]','').replace(';','').replace('\\(','').replace('\\)','',''). This step removed the special characters '[ ] ; ( )' which were the characters that I noticed most frequently upon first glance at the column. Creating a duplicate column also allowed me to keep the original data in case of error or if I needed it for comparison purposes later. This updated 8154 rows.
    b. Next, I trimmed leading and trailing whitespace from the new column (all consecutive steps here will be on the "event_edited" column). This step should be performed multiple times throughout cleaning. This updated 4 rows.
    c. I removed consecutive whitespace from the new column. This step should also be performed multiple times throughout cleaning. This updated 6 rows.
    d. I converted all of the rows to Uppercase capitalization since initial events used uppercase, title case and lowercase, and having a consistent capitalization made it a lot easier to group and read the data. This updated 729 rows.
    e. I also converted all of the rows into Text data type to make it easier to group and read the data. The data were various data types and transforming them further may not work if they are not all a consistent type. This updated 9393 rows.
    f. Next, I noticed I had missed a special character, '?'. I removed this character using GREL ("grel:value.replace('\\?','')"). This updated 93 rows.
    g. I proceeded by beginning to cluster events that had similar names and to remove duplicates or words in other languages. I opened a text facet and clustered using all of the available methods, starting with the key collision method and cycling through each of the functions until they were either empty or only displayed clusters that didn't belong together. I also used the nearest neighbor method with both levenshtein and ppm, changing the radius and block chars to see if it provided me with any new clusters. This updated the following numbers of rows

as I clustered: 4145, 34, 2463, 3, 1344, 2, 2673, 1012, 7, 489, 451, 3296, 2, 679, 974, 976, 3286, 4, 2, 421, 2, 4, 137, 8, 2 rows were updated, respectively.

h. At this point, I realized I had missed two more special characters, “ and \, so I used GREL to remove that from the data ("grel:value.replace('\\'','').replace('\\\"','')"). This updated 89 rows.
i. I then continued to cluster as described above in g: 11, 6, 28 rows were updated, respectively.
j. Next, I was unable to find any more clusters using the provided methods so I looked at groups/names within the text facet. I updated the outliers to the corresponding meal (for example, "COMPLIMENTARY DINNER TO THE REPRESENTATIVES OF THE PRESS OF NEW YORK" became “DINNER”). I had previously attempted to use GREL to update these but I wasn’t able to get it to work because some of the categories included multiple meals (breakfast and lunch or lunch and dinner, for example). By changing these names, I updated: 29 rows (individually listed as 1 row updates).
k. I found a few events that didn’t count as meals as we would use them for our use case (for example, ‘ANNIVERSARY’. What would that correspond to? It could be anything), so I changed this value to NULL instead since we wouldn’t be sure which meal they were talking about in the data. This affected 33 rows.
l. I continued to update outliers or invalid meals as in j and k above: this updated 14, 14, 2, 1, 3, 7, 2, 5, 18 , 45, 14, 23, 3, 1, 2, 3, 2, 1, 2, 2, 2, 3, 2, 1, 2, 15, 4, 4, 2, 6, 5, 11, 4, 4, 2, 14, 3, 9, 1, 10, 1, 55, 1, 28, 2, 2, 9, 2, 2, 3, 3, 10, 7, 3, 32, 3, 10, 3, 16, 8, 17, 4, 2, 2, 2, 3, 15, 12, 3, 9, 2, 1, 3, 8, 4 rows.
m. Next, I changed all of the blank values to NULL in order to consolidate null or unknown values. This updated 9411 rows.
n. I continued to update events within the text facet that did not count as meals as we would use them for our use case by using the strategies in j and k above, as well as changing meals listed in other languages (mostly german, french, and italian) to english (for example, “CARTE DU JOUR” became “DAILY MENU”): this updated 8, 3, 18, 6, 7, 24, 6, 5 rows.
o. I tried to use GREL to change instances of “LUNCHEON” to “LUNCH” and remove the word “COMPLIMENTARY” within the data to make it easier to group but was unsuccessful.
p. I continued updating rows using strategies from n: 8, 9, 8 rows were updated.
q. Here, I decided that I had made enough changes that more groupings should be able to show up if I used the clustering tool. So I began to re-cluster as in g above, as well as update cells as in n: 3652, 1, 5, 3, 2, 1, 33, 17, 11, 10, 10, 9, 6, 6, 6, 5, 5, 5, 4, 4, 4, 3, 6, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 94, 9, 98 rows updated.

r. I changed all of the null values to “UNKNOWN” in order to make things more organized for later data cleaning steps by my teammates, so that they wouldn’t have to deal with null values in Python. This updated 9698 rows.
s. Next, I decided to change “BANQUET” to “DINNER”, as banquets are very typically done during dinner time. This updated 477 rows.
t. Finally, I updated a few more event names to consolidate them with bigger groups (“SPECIAL MENU” to “MENU”, “THANKSGIVING MENU” to “MENU”, “RAMADHAN SNACK MENU” TO “SNACK MENU”, etc): updating 3, 3, 2 rows.
u. Finally, I repeated some of the initial cleaning steps. I trimmed excess whitespace, removed repeating whitespace, converted the column to Uppercase again, and converted the column to Text data type again. This updated 0 rows.

2. Next, I cleaned up the “place” column to consolidate the locations of the menus for our use case. In Menu.csv, the “place” column listed restaurants, ships, street addresses, cities, states, countries, etc. of restaurants or wherever the menus were from. Similarly to the “events”, I could initially spot a lot of dirty data with special characters, very similar places, etc. I performed the following cleaning steps to improve data quality in the “place” column:
    a. As in 1a, I started by creating a duplicate column, “place_edited”, and applied GREL to it to remove some special characters ("grel:value.replace('\\[','').replace('\\]','').replace('\\(','').replace(')','').replace(';',' ').replace('\\'','').replace('\\\"','')"). This step removed the special characters ‘[ ] ; ( ) \ “’ which were the characters that I noticed most frequently upon first glance at the column, and from my experience with 1. Creating a duplicate column also allowed me to keep the original data in case of error or if I needed it for comparison purposes later. This updated 8123 rows.
    b. Next, I trimmed leading and trailing whitespace from the new column (all consecutive steps here will be on the “place_edited” column). This step should be performed multiple times throughout cleaning. This updated 10 rows.
    c. I removed consecutive whitespace from the new column. This step should also be performed multiple times throughout cleaning. This updated 50 rows.
    d. I converted all of the rows to Uppercase capitalization since initial events used uppercase, title case and lowercase, and having a consistent capitalization made it a lot easier to group and read the data. This updated 899 rows.
    e. I also converted all of the rows into Text data type to make it easier to group and read the data. The data were various data types and transforming them further may not work if they are not all a consistent type. This updated 9424 rows.
    f. Next, I noticed I had missed a special character, ‘?’. I removed this character using GREL ("grel:value.replace('\\?','')"). This updated 380 rows.

g. I opened a text facet for the column and changed all of the blank values to UNKNOWN in order to consolidate null or unknown values. This updated 9509 rows.
h. I proceeded by beginning to cluster events that had similar names and to remove duplicates. I clustered using all of the available methods, starting with the key collision method and cycling through each of the functions until they were either empty or only displayed clusters that didn't belong together. I also used the nearest neighbor method with both levenshtein and ppm, changing the radius and block chars to see if it provided me with any new clusters. This updated the following numbers of rows as I clustered: 764, 1370, 58, 1007, 1174, 39, 1682, 322, 54, 790, 54, 665, 800, 127, 6, 51 rows, respectively.
i. Once I had gone through all the clustering options, I began to look more closely at the text facet. I sorted the categories by name and noticed there were a lot of similar places that weren't getting grouped by the clustering algorithms. I began to individually change these to group them together (for example, I noticed there were "203-205 SIXTH AVENUE,NEW YORK,NY" and "203-205 SIXTH AVE,NEW YORK,NY", so I changed the first one to match up with the second one). This updated: 2, 3, 2, 10, 2, 8, 2, 2, 2, 2, 2, 2, 3, 4, 6, 4, 3, 6, 3, 2, 1, 3, 2, 5, 2, 3, 4, 5, 2, 6, 2, 2, 2, 2, 3, 7, 4, 1, 11, 3, 4, 3 rows.
j. Next, I noticed I had missed the special characters { and }, so I removed these characters from the column using GREL ("grel:value.replace('\\{','').replace('\\}','')"). This updated 1 row.
k. I used clustering algorithms again as in step h and found more options for grouping since I had removed enough special characters and consolidated enough of the places. This updated: 4, 37, 29, 429 rows.
l. I used GREL to replace the character '&' with 'and' to remove the special character while still preserving the meaning of the place ("grel:value.replace('\\&','and')"). This updated 104 rows.
m. I used the clustering to edit one more row. This updated 1 row.
n. Finally, I repeated some of the initial cleaning steps. I trimmed excess whitespace, removed repeating whitespace, converted the column to Uppercase again, and converted the column to Text data type again. This updated 8, 0, 103, and 0 rows, respectively.

3. The third task was to clean up the "location" column to consolidate the locations of the menus for our use case. In Menu.csv, the "location" column listed the restaurants, ships, hotels, organizations, or wherever the menus were from. Similarly to the "place" column, I could initially spot a lot of dirty data with special characters, very similar

places, etc. I performed the following cleaning steps to improve data quality in the "location" column:

a. As in 1a and 2a, I started by creating a duplicate column, "location_edited", and applied GREL to it to remove some special characters "grel:value.replace('\\[','').replace('\\]','').replace(';','').replace(' & ',' and ').replace('\\(','').replace('\\)','').replace('\\{','').replace('\\}','')". This step removed the special characters '[ ] ; ( ) { } ' and replaced ' & ' with " and " which were the characters that I noticed most frequently upon first glance at the column, and from my experience with 1 and 2. Creating a duplicate column also allowed me to keep the original data in case of error or if I needed it for comparison purposes later. This updated 691 rows.
b. Next, I trimmed leading and trailing whitespace from the new column (all consecutive steps here will be on the "location_edited" column). This step should be performed multiple times throughout cleaning. This updated 0 rows.
c. I removed consecutive whitespace from the new column. This step should also be performed multiple times throughout cleaning. This updated 555 rows.
d. I also converted all of the rows into Text data type to make it easier to group and read the data. The data were various data types and transforming them further may not work if they are not all a consistent type. This updated 0 rows.
e. I noticed I had missed a special character, '?'. I removed this character using GREL ("grel:value.replace('\\?','')"). This updated 124 rows.
f. I trimmed the leading and trailing whitespace again. This step should be performed multiple times throughout cleaning. This updated 24 rows.
g. Next, I noticed the special characters ', ", and \ occurred a lot, so I removed these characters from the column using GREL ("grel:value.replace('\\'','').replace('\\\"','')" and grel:value.replace('\\\\','')"). This updated 1834 and 167 rows.
h. I proceeded by beginning to cluster events that had similar names and to remove duplicates. I opened a text facet and clustered using all of the available methods, starting with the key collision method and cycling through each of the functions until they were either empty or only displayed clusters that didn't belong together. I also used the nearest neighbor method with both levenshtein and ppm, changing the radius and block chars to see if it provided me with any new clusters. This updated the following numbers of rows as I clustered: 3388, 70, 1328, 16, 2823, 11, 4662, 1216, 1187, 5 rows.
i. Here, I noticed that my earlier GREL transformation on the ' & ' with the spaces did not catch all instances of the ampersand, so I used another GREL expression

to catch the ones without spaces ("grel:value.replace('\\&',' and ')"). This updated 19 rows.

j. I trimmed the leading and trailing whitespace again, and removed consecutive whitespace, updating 0 and 5 rows, respectively.
k. Next, I ran the clustering algorithms again as in h: 1, 5, 23, 32 rows were updated.
l. I changed all of the blank values to UNKNOWN in order to consolidate null or unknown values. This updated 48 rows.
m. I also changed the value "Restaurtant Name And/Or Location Not Given" to “UNKNOWN” to consolidate unknown values. This updated 1 row.
n. I noticed one more special character that I had missed, the ‘/’, and used GREL to remove it from the data ("grel:value.replace('\\/','')"). This updated 56 rows.
o. Finally, I repeated some of the initial cleaning steps. I trimmed excess whitespace, removed repeating whitespace, converted the column to Titlecase, and converted the column to Text data type again. This updated 0, 0, 1901, and 0 rows, respectively.

4. Finally, I had to extract the years from the “date” column so that we could use the year for each menu for our use case analysis later on. I performed the following steps:
    a. I start by changing the column “date” to be data type String to make it easier to extract the year from it. This affects 586 rows.
    b. I then create a duplicate column based on “date” named “year”, taking only the first four characters from each value in the column “date”. I had noticed that the “year” column was organized in the format yyyy-mm-dd, yyyy-xxxx or yyyy-xxxx_wotm, so the year was always displayed in the first four characters. I used GREL ("grel:value.substring(0,4)") . This updated 17547 rows.
    c. I trimmed the leading and trailing whitespace and updated 0 rows.
    d. Finally, I opened the text facet and updated all blank values to “UNKNOWN”, updating 586 rows.

[Julia]
I used OpenRefine to clean data for our final group project. I started with creating a project by importing the dish.csv file. Total number of rows imported was 428086.

1. First task was to clean up the column “name**”** to consolidate dish names. By simply looking at data and sorting it alphabetically and reversing it, I could spot several data issues, such as duplicate names, very similar names or names that could mean the same thing such as “Two Eggs” vs “Eggs (2)”, names with special characters (#, %, &, -, etc), inconsistent capitalization style and blank names. I performed the following cleaning steps to improve data quality in the “name” column:

a. I started with applying basic cleanup to remove consecutive whitespace. As a result, 6582 cells were updated. This step should be performed iteratively from time to time after other changes. I ran it several times during the data cleaning process.
b. Next, I applied a consistent uppercase capitalization style since initial names used a combination of uppercase, sentence case and title case. This step made it easier to read data by the human eye.
c. I proceeded with using the clustering function to identify and capture names that might represent the same thing. Once I tried using clustering with a method set to “key collision” and keying function to “fingerprint” on the “name” column, 41405 clusters were found. I ran into performance issues due to a large dataset size: OpenRefine simply couldn’t process clustering in one batch. As a workaround, I used filters on the right side of the edit dialog to reduce the number of choices in cluster, rows in cluster, average length of choices and length variance of choices (iteratively, one filter at a time). I played with filters to perform clustering in smaller batches in fourteen steps until the clustering edit dialog was blank. Each round still took a few minutes or so. As a result, the following number of cells in the “name” column were affected by mass edit: 16772, 16973, 1443, 2393, 2434, 5545, 5581, 2686, 5045, 6605, 5115, 5377, 10041, 12153.
d. Next, I removed duplicates by sorting names in alphabetical order and then selecting the “blank down” option for cell editing. This option detected cases with two or more following rows having the same names and if they did, the following rows values got removed. Next I used the “Facet by blank” to identify blank cells for the column “name” and edited rows to remove all blank rows. As a result, 82080 rows were removed.
e. Next, I iteratively searched for and removed special characters in names. I used the General Refine Expression Language (GREL) in combination with text transform to accomplish this step. For example, to get rid of question marks in names, I used the following expression: `value.replace(/\?/,"")`. As a part of the special characters clean up, I also removed all names that were written in other languages but english/latin by applying the following expression: `value.replace(/[^\u0020-\u007F]/,"")`. As a result, the following number of cells in the “name” column were affected by text transform: 187, 6086, 9550, 113953, 1826, 683, 271, 52, 6470, 75, 22487, 22339, 109, 29659, 245, 21873, 5, 946, 2, 944, 29011, 12348, 1095, 96, 131, 143, 11, 38, 30, 14, 18, 7, 5, 10, 7, 6, 74, 81. During this iterative process, I trimmed leading and trailing whitespaces several times.

    f. By observing data in the “names” column, I noticed that some names seem to represent the same idea. I tried clustering again with different keying function such as ngram-fingerprint and I was able to group together more similar names.
2. Second task was to make sure items don’t appear fewer times than the menus they appear on.
    a. I started with transforming text values of the “menus_appeared” and “times_apepared” columns to number values which can be done by using GUI and selecting: edit cells > common transforms to number or by applying the following expression: `value.toNumber()`. This way values in both columns are treated as numbers as opposed to strings to do any digit calculations or comparisons.
    b. Next, I created a new column named “appeared_check” based on the column “times_appeared” and passed the following expression: `cells.times_appeared.value - cells.menus_appeared.value` to see if returned value is zero, positive or negative. New column was filled with zeros, positive and negative values. Positive and zero values are valid combinations of “menus_appeared” and “times_appeared” values pairs, meaning, items don’t appear fewer times than the menus they appear on. Negative values are invalid because the “menus_appeared” value cannot be larger than the “times_appeared” value.
    c. I proceeded with converting zeros and positive values to text “Valid” and negative to “Invalid” by using the following expression: `if(value >= 0, "Valid", "Invalid")`
3. Third task was to check that years are within the correct bounds:
    a. I started with converting “first_appeared” and “last_appeared” columns from text values to date values by using date common transform.
    b. Next, I created a new column “date_bound_check” by using the following expression: `if(value.first_appeared <= value.last _appeared)`. This step was performed to make sure that the first year value appeared earlier than last year.
    c. Next, I attempted to use the TimeLine facet to filter out all dates that appeared before 1848 and after 2021, but this method eliminated all dates that are out of bound which was not a part of our initial plan. I tried reworking this step to keep both, in-bound and out-of-bound dates, but I ran into some implementation issues with using GREL expressions to recognize the correct date ranges. For example, I attempted to use the following expressions, but neither gave expected results:
        i. `if(or(value.first_appeared < 1848, value.last_appeared > 2021) , 1, 0)`

ii. if(or(value.first_appeared < ‘1848-01-01’, value.last_appeared > ‘2021-01-01’) , 1, 0)
Decision was made with other team members that Miles will try to create a flag for correct bounds by using Python outside of OpenRefine.

[Miles]

In order to support the use case of a geography-based analysis of menu items over time, Miles took the processed Menu table that Vidya had cleaned using OpenRefine, and used the GoogleMaps and Geopy APIs to assign regions based on the cleaned locations and places columns from Vidya’s work. The first step was to create a column concatenating locations and places into a single string (called ‘loc_place’). Although not used in the final use case, this was a necessary pre-processing step in order to improve results from the GoogleMaps places API. Once the loc_place column was created, unique values where both location AND place had not been classified as unknown were passed into the GoogleMaps API. For places that returned results, Latitude and Longitude from the result was passed into the Geopy API to retrieve regions. Provided a non-Null result was returned, loc_place - region pair was passed into a dictionary and applied to a new ‘region’ column in the Menu table. Where a region was unable to be assigned, the ‘region’ column took on a value of “Unknown” rather than removing the entry from the data set. Entries were not removed in this case since they could still be useful for non-geographic analysis.

In order to improve the internal validity of the ‘first_appeared’ and ‘last_appeared’ columns in the Dish table for time-specific analysis, Miles used Python to identify and, when possible, correct dates with either invalid or incorrect first_appeared and last_appeared dates. The first step was to create a column in the Menu table called ‘year_edited’ that converted the value in the ‘year’ column to a datetime object where possible, or Null where not. This was necessary in order to impute the correct values for first_appeared and last_appeared in the Dish table. Next, two columns ‘first_appeared_unknown’ and ‘last_appeared_unknown’ were created in the Dishes table which respectively converted the values of ‘first_appeared’ or ‘last_appeared’ to datetime, or, in the case of invalid time entries (‘0’,’1’, or values in the future), Null. Once completed, the Menu, MenuItem, MenuPage, and Dish tables were merged together on respective unique IDs. Finally, for each Dish that could be merged to the other tables, the minimum and maximum non-null dates from the ‘year_edited’ column for Menu entries was stored in a dictionary as its first_appeared and last_appeared dates respectively. Finally, this dictionary was used to create the ‘first_appeared_corrected’ and ‘last_appeared_corrected’ columns in the original Dishes table. For Dishes that either were not mapped to a Menu, or failed to determine correct dates, Null values were kept.

The final cleaning step using Python was to ensure the Internal Validity of entries in each of the 4 tables. This was done by removing any entries in the separate tables that was not able to get merged with the other tables, implying stand-alone Dishes or Menus. These were removed because they would not be able to be used for analysis at all. Additionally, we ensured that only one entry per id existed in each table.

Changes in Data Quality

[Vidya]

Data in the Menu.csv file was improved as follows:

- Columns changed: “event” (duplicated and edited as “event_edited”), “place” (duplicated and edited as “place_edited”), “location” (duplicated and edited as “location_edited”), and “date” (duplicated and edited as “year”)
-

	Cells edited in “event_edited”	**Cells edited in “place_edited”**	**Cells edited in “location_edited”**	**Cells edited in “year”**
Removing consecutive whitespace	6	50	560	0
Removing leading or trailing whitespace	4	18	24	0
Removing special characters	8336	8608	2891	0
Updating Title Case and Data type	10122	10426	1901	586
Updating blank/null values	9698	9509	49	586
Clustering	26493	9463	14767	0

Methods				
Consolidating group or individual names	1204	147	1	0

[Julia]

Data in the Dish.csv file was improved as follows:

- Number of cells changed by clustering to identify items that might mean same things: 16772, 16973, 1443, 2393, 2434, 5545, 5581, 2686, 5045, 6605, 5115, 5377, 10041, 12153 - each number shows number of cells changed after each clustering iteration.
- Number of cells changed by basic clean up such as removing consecutive whitespaces and trimmed leading and trailing whitespaces: 6582, 213, 24, 2, 66, 35, 34, 313, 76, 96, 57, 81
- Number of cells changed by clustering: 16772, 16973, 1443, 2393, 2434, 5545, 5581, 2686, 5045, 6605, 5115, 5377, 10041, 12153.
- Removed duplicates: 82080 rows were removed.
- Number of cells changed by spotting special characters: 187, 6086, 9550, 113953, 1826, 683, 271, 52, 6470, 75, 22487, 22339, 109, 29659, 245, 21873, 5, 946, 2, 944, 29011, 12348, 1095, 96, 131, 143, 11, 38, 30, 14, 18, 7, 5, 10, 7, 6, 74, 81.

[Miles]

The supplemental document ‘ApplyRegionsAndCorrectYears.html’ delivered along with this report gives more in-depth demonstrations of data-quality enhancements and internal validity checks for all Python related cleaning steps, but in summary:

The process used to assign regions to Menus in the ‘region’ column assigned regions to 4,801 of the 17,547 Menu entries and ‘UNKNOWN’ to the remaining 12,746 entries. Converting the ‘year’ column in the Menu table to datetime resulted in assigning 591 Menu entries a Null value for the year and a properly formatted datetime object for the remaining 16,956 entries.

Of the 343,698 entries in the Dish table, the process to correct first_appeared and last_appeared dates reduced the number of invalid dates by 11,381 and 11,375 entries respectively. Additionally, it corrected 37,698 first_appeared dates and 37,645 last_appeared entries where the original date did not match up with the first and last menus the dish appeared on. In the end, 36,109 first_appeared and 36,108 last_appeared entries were still Null after cleaning.

Finally, the Internal validity checks to ensure each entry in each of the Menu, MenuItem, MenuPage and Dish tables could me matched to each other removed 40,494 entries from the MenuPage table, 305,028 entries from the MenuItem table, 43 entries from the Menu table, and 12,162 entries from the Dish table.

Workflow Model

The Workflow model was created in diagrams.net. Outer workflow shows all higher-level steps we performed as a team during our final Data Cleaning project. Inner workflow shows detailed steps we performed during the data cleaning process by manually going through OpenRefine and Jupyter Notebooks logs.

Link to shared PDF:
https://drive.google.com/file/d/194s2cGqi7ajf5z7tGmawRdJFqF9ncGSp/view?usp=sharing

Conclusions and Summary

In conclusion, much of the cleaning process was fairly straightforward with no real surprises. One of the major takeaways we learned was about the power, but also the limitations of OpenRefine. For both Vidya and Julia, OpenRefine proved a powerful tool for clustering and cleaning text columns in the Menu and Dish tables respectively. However, unsurprisingly with messy datasets such as this one, there's a balance between functional and perfect results. While OpenRefine is a powerful tool, some of the additional flexibility offered by Python was better suited to tasks that provided supplemental data, or imputations of invalid/incorrect data across multiple tables. That being said, the ability to use built-in functions to apply unsupervised clustering techniques to more quickly group data was an invaluable advantage for this Project.