

Casey Blair: NetID: cdblair2; email: cdblair2@illinois.edu

Omar Boffil: NetID: oboffil2; email: oboffil2@illinois.edu

Team: Game of Threads

CS 513 Final Project

New York Library's Crowd Sourced Historical Menu Dataset Cleaning

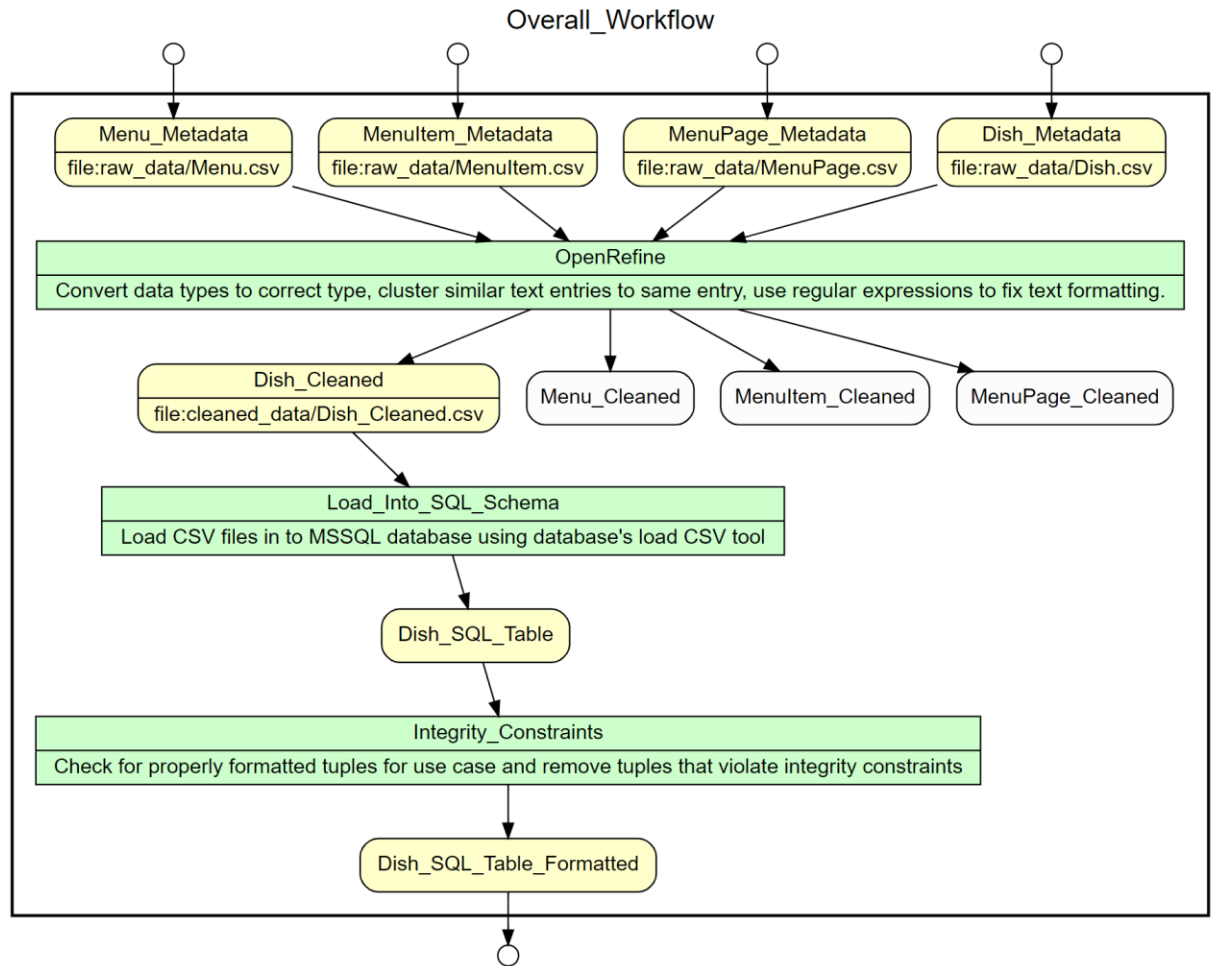
1. Initial Assessment of the Dataset and Use Case Discussion

The New York Library's Crowd Sourced Historical Menu dataset contains data gathered from over a hundred years of menus from restaurants across the United States of America. The names of restaurants the menus came from, what year the menus were written, the meal type (breakfast, lunch, dinner, special event, or other) the menu was for, images of the menu itself, and the prices of the food items (dishes) that were on the menu all can be found in this dataset. This dataset has text entries from many different authors: restaurants wrote their own menus and not all menus are completely in English. This leads to challenges in cleaning this dataset for analysis as the same meal may have slightly different names due to the multitude of authors of this dataset, different accent marks and umlauts used, and misspelled words. And there are incomplete entries in the dataset that we will detect with our SQL integrity constraints if the blank entries need to be removed from our dataset or can be included and ignored as the blank values will not affect our calculations. And we will need to focus our data cleaning on subsets of the dataset relevant to our goal of calculating average dish prices over time: we will focus our cleaning efforts mostly on the Dish.csv file since the other files do not contain data relevant to calculating average prices of dishes.

Our real-world use case of this dataset is calculating the average price of individual dishes over time. This will require that the name of the dishes be consistent: that multiple entries for the same dish have the same text format so our SQL aggregation queries could correctly calculate the average value of the dish from multiple entries. The price of the dish must be included, so we must catch entries that do

not have a price in our integrity constraints. If the low price is higher than the recorded high price, then we must flag these as errors and exclude them as there seems to be an error in data entry, and that the price, both low and high, are in numeric format so our SQL queries can correctly calculate an average price. Also, any dates we use to group our dishes into time windows must be in ISO format so our SQL database can correctly use the ISO format date to create a database date object correctly.

Our overall workflow ended up focusing on the Dish.csv file as the name and prices of the dishes were both located in the Dish.csv file. Therefore, even though we had performed data cleaning operations in OpenRefine on the Menu.csv, MenuItem.csv, and MenuPage.csv file, it was only necessary to use the cleaned Dish CSV file in the next step of our pipeline: the SQL integrity checks. Our overall workflow took the form of the following diagram.



2. Identifying use cases for which the dataset is already clean and use cases for which it will never be clean enough or usable

The dataset would already be clean enough for someone to browse the images associated with the image_id column in the MenuPage.csv file if they were using the MenuPage.csv file and had access to the folder with the PNG or JPEG images of the menu pages. Since the image_id field is technically a number, before trying to access a file using this data entry the number would be converted back to a string to find the file name on the file system, and the image_id column was already in string format. This would please users that simply want to browse the dataset looking for images of menus.

The dataset would never be clean enough to perform a historical study of the most popular food types based for breakfast, lunch, and dinner over the decades that span the entries in this dataset.

There are many entries missing in the “event” column of the Menu.csv file, however, in the non-blank entries in this column is recorded the event such as “Breakfast, Lunch, Dinner, etc.” Also, since not every entry is entered as breakfast, lunch, or dinner: some are special one-time occasions such as weddings or fundraiser meals, this column does not have a narrow set of entries that would allow for a truly accurate study of the above mentioned categories to be conducted. Also, some entries are not in English: some entries for Breakfast are written in German as “Frühstück” which I attempted to partially translate to English during the clustering step we performed on this column, however, there may have been entries in other languages that were missed since this file was not really needed for our use case. Because of the different language entries in this column, the lack of entries, and the multitude of possible entries that make up the entries in this column determining which time of the day each of these meals were eaten would not be possible.

3. Description of the Dataset’s Structure, Quality Issues, and Data Cleaning Steps

This dataset consists of four files: Menu.csv, Dish.csv, MenuItem.csv, and MenuPage.csv. The Menu.csv file has metadata about the menu itself: which restaurant it was from, what date the menu was recorded in the dataset, what event (if any) the menu was from such as breakfast, lunch, or a banquet. Also, it includes the type of venue (commercial, etc.), location (hotel, restaurant name, ship name, or banquet or other event name), what language the menu is in, the occasion of the event (ex: Easter), the city and state of the event/restaurant, a free form text description of the menu appearance, a free form text entry of the notes about the menu from the person who recorded the menu into the

dataset, the status of the menu record entry (complete, etc.), how many pages the menu was, and how many dishes (meals) were on the menu.

The MenuPage.csv file has an ID column which is a unique identifier of that menu page in the file along with a foreign key to the ID of the menu that page was from (the menu_id column). The page number of this MenuPage is also recorded, as well as an image_id which must link this menu page entry to an image in a file system somewhere. The height and width of the menu page are also recorded along with a UUID value which is a unique identifier for this menu.

Data cleaning issues with the MenuPage.csv file was that the columns with numeric columns, the ID, menu_id, page_number, image_id, full_height, and full_width columns all were strings instead of numbers. The first step taken in OpenRefine on this page was to convert the columns that need to be numeric data types in to numeric data types. Then, we made sure to strip the leading and trailing white space from the uuid column, and then we stripped consecutive white space entries to ensure duplicate white space entries were not present.

The MenuItem.csv file has a menu_page_id column which is a foreign key to the MenuPage.csv file's ID column. There is a price column with the recorded price of this specific menu item, along with a high_price column that has many missing entries. The dish_id column is a foreign key to the Dish.csv file's ID column. The created_at and updated_at columns record when the entry was first entered into the dataset and the last time the record was updated respectively. The xpos and ypos columns appear to be a x and y coordinate for where the menu item was located on the page of the menu.

The MenuItem.csv file also had many columns that needed to be converted from text data types to numeric data types. The ID, menu_page_id, price, high_price, dish_id, xpos, and ypos columns were all in text format but needed to be converted to numeric format, which was our first step in this file. Then, we used regular expressions to format the created_at and updated_at columns with ISO

standardized data formats so that if we decided these dates were necessary for our use case, we would be able to import them in to our database by creating database native date objects using an ISO standardized string format. After using regular expressions to change the “UTC” string at the end of the string to a “Z”, then placing a “T” character between the date and time so the string format matches the ISO standard, we trimmed the leading and trailing white space in case either our regular expression introduced extra white space in to the string or the white space existed before our text manipulation. We then collapsed the extra white space to ensure unnecessary white space was not in our text.

The Dish.csv file has data about individual dishes or meals you could order from restaurants that the menus were collected from. There is a name of the dish in the name column, a description of the dish in the description which is blank for most of the entries, the number of menus the dish appeared in in the menus_appeared column, the number of times it appeared in all the menus in the times_appeared column, the year when the dish first appeared and last appeared in the first_appeared and last_appeared columns, respectively. And the lowest price and the highest price of the dish in the lowest_price and highest_price columns, respectively.

The Dish.csv file also had many columns that were in text format but needed to be converted to numeric format. The ID, menus_appeared, times_appeared, first_appeared, last_appeared, lowest_price, and highest_price columns all needed to be converted to the numeric data type. Then, the name column, which has the text of the name of the dish, needed leading and trailing white space stripped from its entries. Also, some entries of the name column were in all capital letters, so we decided to standardize the casing of the entries using title case so the same name with different capitalization schemas would not appear as different (distinct) entries when our SQL queries aggregated names to calculate average price of dishes. So, we applied the title case transform to this column. And the description column was blank for all entries, so we decided to remove it from the dataset as it would not be useful in our analysis.

Dish.csv's name column also had many non-alphanumeric characters such as parenthesis, square brackets, asterisks, octothorpes (#), question marks, and quotation marks. Since we will need to aggregate entries with the same name in our SQL queries, it was important to remove these non-alphanumeric characters so our SQL queries would not mistake the same entry in different rows as distinct values because of an extra character. We also discovered that there were names with the same meaning but used an ampersand in some rows, and the word "and" in other rows. We decided to standardize the term "and" as the text entry "and" instead of the "&" character, so we used a regular expression to replace all "&" characters with the word "and" which worked nicely.

Since we had already run the title case on the name column, but we just introduced new word "and" in to several rows, it was necessary to run the title case transform again on this column. This ensured that the names were truly in title case format.

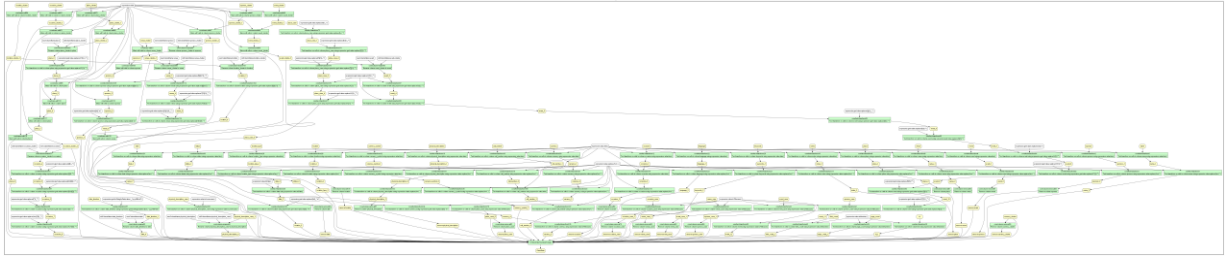
We noticed that some entries had non-English characters such as vowels with umlauts, and e and "a" characters with accent marks. However, other rows had the same text without the umlaut or accent marks. We decided to standardize the name column to English characters without umlauts or accent marks for our analysis goals as it was more important for our SQL queries to aggregate name entries with the same entry than the grammatically correct spelling in the native language (German, French, or other) that the dish came from. Since different authors included the accents, while others excluded them, we needed to standardize the entries for our SQL analysis step to work correctly.

Then, we use the clustering tool in OpenRefine to find similar entries in the name column to standardize these entries into the same text. We had to edit our refine.bat file in our Windows 10 installation of OpenRefine to allow OpenRefine the ability to process over 2.5 million entries in the cluster tool as the default limit was about one million. Also, we had to update the maximum available RAM allowed for the program from 1.5 GB to 5 GB to allow the OpenRefine clustering operation to

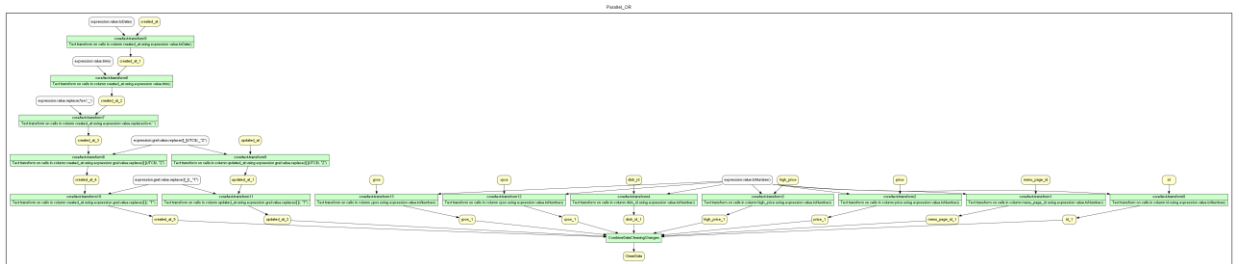
complete in a reasonable amount of time. After making these adjustments to the OpenRefine settings we were able to run the clustering operation and cluster over 2 million entries.

The MenuPage.csv file does not have many missing entries. This table has data about which menu the page came from, the page number of the menu the page is, an image_id value to identify an image filename or ID of a picture of the actual menu, the height and width of the page, and a UUID value that is a unique identifier for this page.

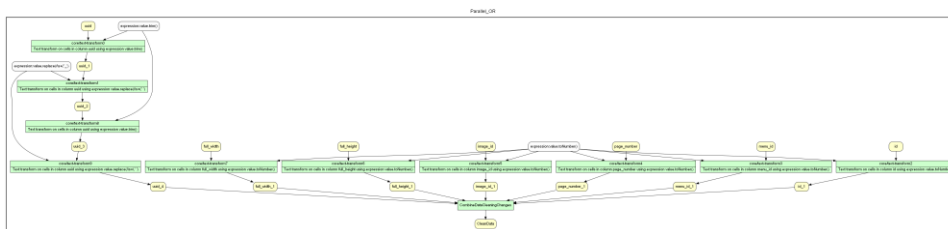
The following diagram represents the OpenRefine operations performed on the Dish.csv file. This file can also be found in the zip file named Dish_Workflow.yw for the YesWorkflow definition file and the Dish_Workflow.png for the image.



The following diagram represents the OpenRefine operations performed on the MenuItem.csv file. This file can also be found in the zip file named MenuItem_Workflow.yw for the YesWorkflow definition file and the MenuItem_Workflow.png for the image.



Finally, the following diagram represents the OpenRefine operations performed on the MenuPage.csv file. This file can also be found in the zip file named MenuPage_Workflow.yw for the YesWorkflow definition file and the MenuPage_Workflow.png for the image.



Since after we performed the OpenRefine stage of the cleaning pipeline we discovered the data we needed was in the Dish_Cleaned.csv file, we decided to only quantify the OpenRefine data cleaning operations performed on this file. They are as follows:

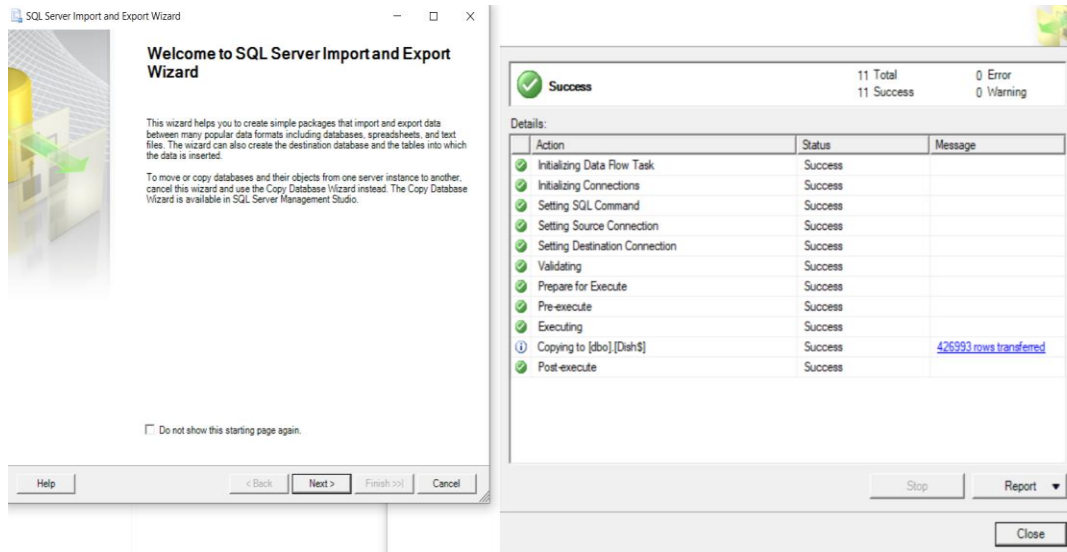
Column Name	Operation	Cells Affected
id	Change data type to number	426993

menus_appeared	Change data type to number	426993
times_appeared	Change data type to number	426993
first_appeared	Change data type to number	426993
last_appeared	Change data type to number	426993
lowest_price	Change data type to number	397892
highest_price	Change data type to number	397892
name	Trim leading and trailing whitespace	9226
name	Collapse concurrent whitespace	6554
name	Change to Title Case	284180
name	Regular expression: remove [], (), or " chars	35815
name	Remove period char (".")	34629
name	Change to Title Case	33471
name	Remove <> chars	186
name	Replace "ä" char with "a" char	1508
name	Replace "é" with "e" char	5834
name	Replace "ü" with "u" char	1899
name	Replace "+" with "&" char	126
name	Replace double hyphen (--) with single hyphen (-)	1373
name	Replace & with "and"	11945
name	Change to Title Case	11736
name	Replace " - " (space, hyphen, then space) sequence with single space	10381
name	Remove * chars	1800
name	Remove #, ?, and " : " chars	520
name	Cluster similar entries in to standardized value using OpenRefine's Cluster functionality	61435

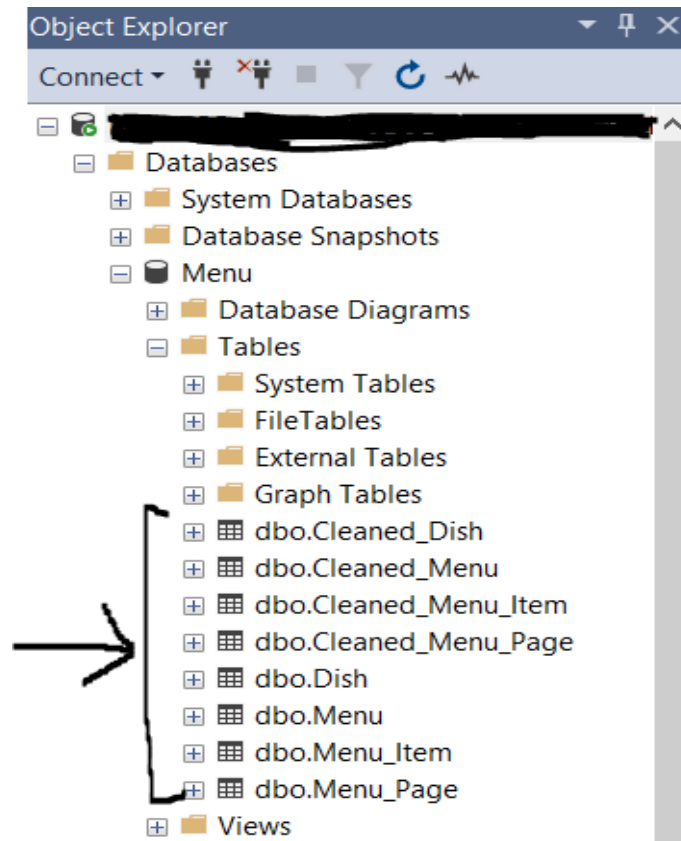
The operation history of our OpenRefine steps can be found in the Open_Refine_History_Filename.json files in the ZIP file. Since we processed multiple files, and each of the input CSV files had their own OpenRefine JSON history file, we kept the files separated for clarity's sake.

4. Load data into a database

To continue with futures inspection in the original dataset, we created a database called Menu in SQL Server. We load the data using the SQL Server Import and export Wizard GUI.



We created tables for each file and each of the cleaned data from OpenRefine. This demonstrates how the integrity constraints don't exist in the original dataset, but it exists in the new files that were modified in OpenRefine.



We want to apply some further modification to the Cleaned_Dish table to help us get the results that we are trying to achieve first.

1.1 - Because we want to know the average price of each Dish, we don't want to work with Dish that doesn't have an amount in both columns we want the ones with prices in one column or both, so we create a new table in our database that has the right information:

- `select * into Dish_price from [dbo].[Cleaned_Dish] where lowest_price <> 0 and highest_price <> 0`

1.2 - Now that we have all the Dish with price, we will drop the columns that are not necessary for our purpose which are menus_appeared and times_appeared

- `alter table [dbo].[Dish_price] drop column menus_appeared, times_appeared`

1.3 - Then, we rename the table [dbo].[Dish_price] to [dbo].[Cleaned_Dish]

Now that we applied the last modification to the data, we can continue with the relational schema, where we will create rules to ensure that the information is useful for our calculations.

5. Developing a relational schema and Identifying the appropriate integrity constraint

The Menu file and the Dish file are the principal tables that will help us to calculate the average price for each different Dish, the price increase in time, and the most offered Dish in the country. We will create logical integrity constraints that ensure that our file is clean enough to trust in the data.

We created and ran a query to show the preview of the original Dish table and the cleaned version of it:

Running the queries:

- `select * from [dbo].[Dish]`

```
--Original Dish Table
select * from [dbo].[Dish]
```

	id	name	description	menus_appeared	times_appeared	first_appeared	last_appeared	lowest_price	highest_price
1	1	Consomme printaniere royal	NULL	8	8	1897	1927	0.2	0.4
2	2	Chicken gumbo	NULL	111	117	1895	1960	0.1	0.8
3	3	Tomato aux croutons	NULL	14	14	1893	1917	0.25	0.4
4	4	Onion au gratin	NULL	41	41	1900	1971	0.25	1
5	5	St. Emilion	NULL	66	68	1881	1981	0	18
6	7	Radishes	NULL	3263	3347	1854	2928	0	25
7	8	Chicken soup with rice	NULL	48	49	1897	1961	0.1	0.6
8	9	Clam broth (cup)	NULL	14	16	1899	1962	0.15	0.4
9	10	Cream of new asparagus, croutons	NULL	2	2	1900	1900	0	0
10	11	Clear green turtle	NULL	156	156	1893	1937	0.25	60
11	12	Striped bass saute, meuniere	NULL	2	2	1900	1900	0	0

- `select * from [dbo].[Cleaned_Dish]`

```
select *
-- Cleaned Dish Table
Select * from [dbo].[Cleaned_Dish]
```

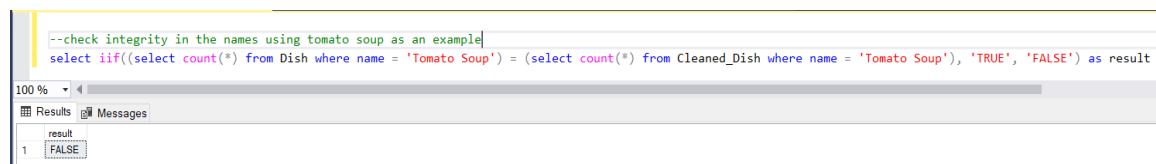
	id	name	first_appeared	last_appeared	lowest_price	highest_price
1	121136	Carrots And Peas In Cream	1900	1933	15	15
2	121139	Chicken Squab Whole	1896	1896	1.5	1.5
3	121140	Mignons Of Filet Of Beef With Mushrooms	1896	1896	1.5	1.5
4	121141	Waldorf Potatoes	1896	1896	0.3	0.3
5	121142	Chicken Broth A La Bellevue Per Cup	1896	1900	0.3	0.4
6	121143	Sandwich A La Rex	1896	1914	0.3	0.3
7	121144	Filet Chateaubriand A La Jardiniere	1900	1900	150	150
8	121145	Demidoff Salad	1896	1915	0.65	1.5
9	121147	Demi Deuill Salad	1896	1896	1	1
10	121148	Salad A La Rex	1896	1896	1	1
11	121149	Porterhouse Steak Single With Bacon Or Onions	1900	1900	110	110

We can appreciate the result of this query, one of the modifications that were done in the data cleaning process, where we have a description column at the original file and not in the cleaned file, and a preview of the different columns and their values that we help us to create the rules.

Making the comparison and creating rules to make sure integrity on the Dish file, we create a rule to make sure that the name of the dishes on the data has all the same format and that they are the same amount of meals for each section or plate.

As you can find in the query below:

```
select iif((select count(*) from Dish where name = 'Tomato Soup') =  
(select count(*) from Cleaned_Dish where name = 'Tomato Soup'),  
'TRUE', 'FALSE') as result
```



When we compare the original dish file with the cleaned data using this rule, we don't pass it. This is because the original records have a different format to type the same name for each Dish. This could have a severe problem when you work with the data because we will find duplicates values that will affect our calculation. But after we cleaned the file, we can create different rules to make sure that the Dish cleaned files are correct for our purpose.

This file must follow:

i. Count of duplicate id, results = 0

- `select distinct id, count(id) as '#id' from Cleaned_Dish group by
id having count(id) > 1`


```
select distinct id, count(id) as '#id'
from Cleaned_Dish group by id having count(id) > 1
```

100 %

Results Messages

id	#id
----	-----

The Primary key must be unique for each case; that is why it doesn't show any results. If we check for the id modifying, we get the unique identifiers and how many times they repeat.

- `select distinct id, count(id) as '#id' from Cleaned_Dish group by id having count(id) = 1`

```
select distinct id, count(id) as '#id'
from Cleaned_Dish group by id having count(id) = 1
order by id
```

100 %

Results Messages

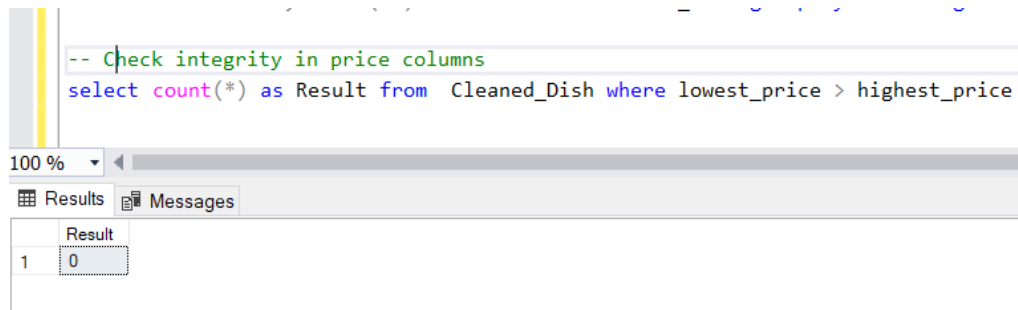
	id	#id
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	7	1
7	8	1
8	9	1
9	10	1
10	11	1
11	12	1

We need to make sure we don't have prices in the lower price columns higher than the highest price.

This is essential if we want to calculate the average costs for each Dish and its price rise.

ii. `lowest_price > highest_price`, results = 0:

- `select count(*) as Result from Cleaned_Dish where lowest_price > highest_price`



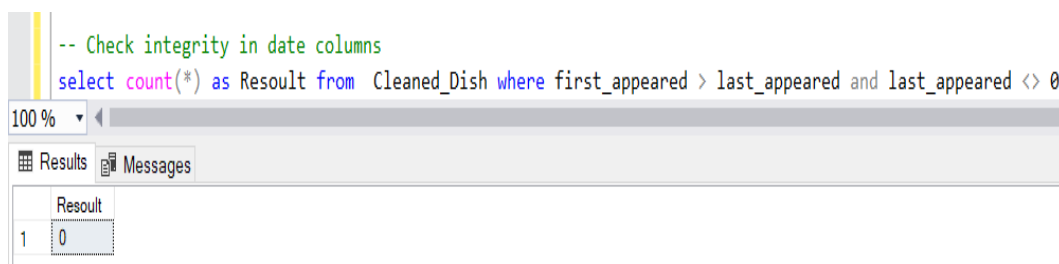
```
-- Check integrity in price columns
select count(*) as Result from Cleaned_Dish where lowest_price > highest_price
```

Result
0

The same happens with the appeared days the `first_appeared` column must be years greater than `last_appeared`, but because some dishes do not have the last appeared day, but we still want to work with their price, we make sure we select them too.

iii. `first_appeared > last_appeared and last_appeared <> 0`, results = 0

- `select count(*) from Cleaned_Dish where first_appeared > last_appeared and last_appeared <> 0`



```
-- Check integrity in date columns
select count(*) as Result from Cleaned_Dish where first_appeared > last_appeared and last_appeared <> 0
```

Result
0

After the Dish file was clean and passed the above rules, we can do some calculations that will help us understand the average price per Dish. This information is valuable because it will help people to know what dishes have, on average, the more affordable price.

iv. Average price per dish

- `select name, ROUND(AVG(lowest_price) + AVG(highest_price) / 2, 2)`
`AS 'AVERAGE DISH PLATE' from Cleaned_Dish group by name`

```
-- Calculate average price per dish
select name, ROUND(AVG(lowest_price) + AVG(highest_price) / 2, 2) AS 'AVERAGE DISH PLATE'
from Cleaned_Dish
group by name
```

name	AVERAGE DISH PLATE
A Broiled Pork Tenderloin	0.55
A Broiled Pork Tenderloin Breaded With Tomato Sa...	0.56
A Changing Louisiana Soup Specialty	4.13
A Cup Of Chocolate	0.15
A Fresh Vegetable Dinner Served With Poached Egg	2.7
A Fresh Vegetable Dinner With Poached Egg	2.31
A Genuine Boiled New England Dinner	0.52
A Glass Of Pure Cream	0.38
A Hupfel's Sons Special	0.1
A La Herbert Salad	0.7

v. Last time of the Dish in menu

- `select distinct name, max(last_appeared) as 'Lats year in menu'`
`from [dbo].[Cleaned_Dish] group by name`

```
-- Last time a dish appears in the menu
select distinct name, max(last_appeared) as 'Lats year in menu'
from [dbo].[Cleaned_Dish]
group by name
```

name	Lats year in menu
Field Salad	1968
Field Son And Co Sloe	1918
Field Son And Co Sloe Gin	1917
Fiesta Fried Cheesecake	1997
Fig Pudding Spice Sauce	1937
Filbert Ice Cream	1963
Filet Chateaubriand A La Bordelais	1887
Filet De Boeuf Pique Jardiniere	1906
Filet De Chevreuil Tyrolienne	1917
Filet De Kingfish Bella	1917
Filet De Porc Milanaise	1913

vi. First time data was stored

- `select min(first_appeared) as 'Oldest year in the menu' from [dbo].[Cleaned_Dish]`

```
-- Oldest year in the menu
select min(first_appeared) as 'Oldest year in the menu' from [dbo].[Cleaned_Dish]
```

Oldest year in the menu
1851

As we can see now that the data is cleaned, we can create different queries that help us to understand the data more and to produce analysis that will help us with predictions or other things.

6. Challenges Encountered

The biggest challenges encountered while working with this dataset was how to standardize entries in different languages. If the dataset was going to be used in a web interface, it would have been

important to preserve non-English characters with accents and umlauts to preserve the original language to display to the user. However, for our analysis use case where we needed to standardize entries from multiple authors where some used non-English characters while others did not when describing the same dish, we went back and forth on how we should process this part of the data and ultimately decided that using English chars would simplify other processing issues that our DB system might encounter too. Some database systems have issues with non-English text encodings, and adding the extra work of ensuring database system compatibility for characters that were not necessary for the SQL aggregation functions we were planning on using for the real world use case we decided to use English characters. This helped us not need to look up all the possible dishes during the Dish.csv's OpenRefine clustering stage as there were as many entries with accents or umlauts as there were not, and not being a native speaker in either German or French it would require an external dataset to truly correct every dish name in to it's true native language's spelling.

7. Contributions

Casey Blair (NetID: cdblair2, email: cdblair2@illinois.edu) worked on the OpenRefine cleaning steps and YesWorkflow diagrams. Omar Boffil (NetID: oboffil2, email: oboffil2@illinois.edu) worked on the relational schema and integrity constraints in SQL.