CS 513 - Theory And Practice of Data Cleaning Final Project - Phase 2 Report

Team ID - 150
Sushma Ponna - ponna2@illinois.edu
Michael Inoue - mwinoue2@illinois.edu
Sanjeev Datta - sanjeev6@illinois.edu

Changes from the Phase 1 plan -

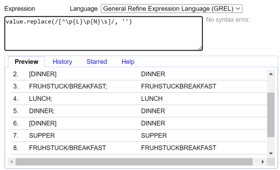
1. We refined our Use case U1 to be a more complex query spanning over all the 4 files. Our current use case **U1** is "***To find the most popular dish and then find out which venues serve the most expensive version of the most popular dish***"
2. We used OpenRefine, R, Python, SQL (SQLite), and YesWorkflow to complete the project.

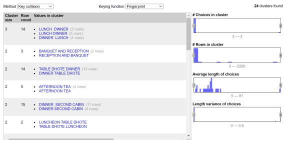
1. Description of Data Cleaning Performed

Data Cleaning Steps performed

a. Menu.csv

The below data cleaning steps have been performed on *'menu.csv'*

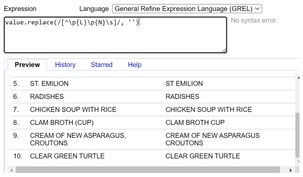
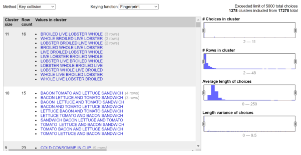
Data Cleaning Step	Tool Used to Perform the Data Cleaning Task	Columns Applied to	Reason for application	Example Case
Trim leading and trailing whitespace	OpenRefine	'name', 'sponsor', 'event', 'venue', 'place', 'occasion', 'notes', 'location', 'currency', 'status'	Normalizes multiple spaces between words to one single space. All the 3 steps make querying for the Use Case U1 easy and meaningful. The columns 'sponsor', 'event', 'venue', 'currency' are used in our use case U1 but it is good to have clean data in other columns so we can generate some insights in the future	' dinner ' → 'DINNER' 'Dinner ' → 'DINNER'
Collapse consecutive whitespace				
Convert text to uppercase				
Regular Expression "value.replace(/[^\p{L}\p{N}\s]/, '')"	OpenRefine	'sponsor', 'event', 'venue', 'place', 'occasion', 'location'	Deletes all special characters except space. Normalizes varied entries and makes searching for an entry retrieve a single result instead of multiple variants. The columns 'event', 'venue' are used in our use case U1 but it is good to have clean data in other columns so we can generate some insights in the future	 Expression Language General Refine Expression Language (GREL) `value.replace(/[^\p{L}\p{N}\s]/, '')` No syntax error. Preview History Starred Help 2. [DINNER] DINNER 3. FRUHSTUCK/BREAKFAST; FRUHSTUCKBREAKFAST 4. LUNCH; LUNCH 5. DINNER; DINNER 6. [DINNER] DINNER 7. SUPPER SUPPER 8. FRUHSTUCK/BREAKFAST FRUHSTUCKBREAKFAST

Cluster and Edit - (Method: Key collision)	OpenRefine	‘sponsor’, ‘event’, ‘venue’	The *‘Key collision’* method is the most appropriate for our project because we want to find an alternative representation of the key that contains the most meaningful part of the string. Majority of the clusters were grouped together by the *‘Fingerprint’* keying function followed by the *‘n-gram’* and *‘Metaphone3’* The *‘nearest-neighbor’* is time taking and computationally expensive The use case relies on the data from ‘event’, and ‘venue’ and it is important to normalize and cluster entries in these fields	 Method: Key collision Keying function: Fingerprint **24** clusters found Cluster size / Row count / Values in cluster 3 / 14 / LUNCH DINNER (5 rows); LUNCH DINNER (5 rows); DINNER LUNCH (4 rows) 2 / 3 / BANQUET AND RECEPTION (2 rows); RECEPTION AND BANQUET 2 / 14 / TABLE DHOTE DINNER (13 rows); DINNER TABLE DHOTE 2 / 5 / AFTERNOON TEA (4 rows); AFTERNOON TEA 2 / 15 / DINNER SECOND CABIN (11 rows); DINNER SECOND CABIN (4 rows) 2 / 2 / LUNCHEON TABLE DHOTE; TABLE DHOTE LUNCHEON # Choices in cluster 2 — 3 # Rows in cluster 0 — 2200 Average length of choices 5 — 81 Length variance of choices 0 — 0.5
Drop Column	OpenRefine	‘Keywords’, ‘language’, ‘location_type’	These 3 columns do not have any data populated and hence they are dropped	
Drop rows with ‘venue’ = NULL	R	‘venue’	These rows with missing ‘venue’ details do not add any information to our Use Case U1 results and are rather misleading and hence they are dropped.	
Drop rows with ‘event’= NULL	R	‘event’	These rows with missing ‘event’ details do not add any information to our Use Case U1 results and are rather misleading and hence they are dropped.	
Keeping rows with only ‘currency’ = ‘DOLLARS’	R	‘currency’	63% of the data are missing currency values and the currency conversion rates are not included in the data but it is important to find out which is the most expensive dish for our use case and hence we decided to keep this column and consider only records with ‘USD’ and drop the rest of the rows for our project	
Keeping rows with ‘event’ = ‘DINNER’	R	‘event’	It is important for our Use case to identify rows with ‘event’=’DINNER’ and hence we drop the rest of the records.	

b. **Dish.csv**

The below data cleaning steps have been performed on *‘dish.csv’*

Data Cleaning Step	**Tool Used to Perform the Data Cleaning Task**	**Columns Applied to**	**Reason for application**	**Example Case**
Trim leading and trailing whitespace	OpenRefine	‘name’	Normalizes multiple spaces between words to one single space. All the 3 steps make querying for the Use Case U1 easy and meaningful. The column ‘name’ is used in our use case U1	‘ clam broth (cup) ’ → ‘CLAM BROTH (CUP)’ ‘CLAM Broth (cup)’ → ‘CLAM BROTH (CUP)’
Collapse consecutive whitespace				
Convert text to uppercase				

Regular Expression “value.replace(/[^\p{L}\p{N}\s]/, '')"	OpenRefine	‘name’	Deletes all special characters except space. Normalizes varied entries and makes search for an entry retrieve a single result instead of multiple variants The column ‘name’ is used in our use case query to look for dish names	 Expression Language General Refine Expression Language (GREL) `value.replace(/[^\p{L}\p{N}\s]/, '')` No syntax error. 5. ST. EMILION — ST EMILION 6. RADISHES — RADISHES 7. CHICKEN SOUP WITH RICE — CHICKEN SOUP WITH RICE 8. CLAM BROTH (CUP) — CLAM BROTH CUP 9. CREAM OF NEW ASPARAGUS, CROUTONS — CREAM OF NEW ASPARAGUS CROUTONS 10. CLEAR GREEN TURTLE — CLEAR GREEN TURTLE
Cluster and Edit - (Method: Key collision)	OpenRefine	‘name’	The *‘Key collision’* method is the most appropriate for our project because we want to find an alternative representation of the key that contains the most meaningful part of the string. Majority of the clusters were grouped together by the *‘Fingerprint’* keying function followed by the *‘n-gram’* and *‘Metaphone3’* The *‘nearest-neighbor’* is time taking and computationally expensive The use case relies on the data from ‘name’ and it is important to normalize and cluster entries in this field	 Method Key collision Keying function Fingerprint Exceeded limit of 5000 total choices **1378** clusters included from **17278** total Cluster size / Row count / Values in cluster 11 / 16 / BROILED LIVE LOBSTER WHOLE (3 rows), WHOLE BROILED LIVE LOBSTER (3 rows), LOBSTER BROILED LIVE WHOLE (2 rows), BROILED WHOLE LIVE LOBSTER, LIVE BROILED LOBSTER WHOLE, LIVE LOBSTER BROILED WHOLE, LOBSTER LIVE BROILED WHOLE, LOBSTER WHOLE BROILED LIVE, LOBSTER WHOLE LIVE BROILED, WHOLE LIVE BROILED LOBSTER, WHOLE LIVE LOBSTER BROILED 10 / 15 / BACON TOMATO AND LETTUCE SANDWICH (4 rows), BACON LETTUCE AND TOMATO SANDWICH (3 rows), BACON LETTUCE AND TOMATO SANDWICH, BACON AND TOMATO LETTUCE SANDWICH, LETTUCE BACON AND TOMATO SANDWICH, LETTUCE TOMATO AND BACON SANDWICH, SANDWICH BACON LETTUCE AND TOMATO, TOMATO LETTUCE AND BACON SANDWICH, TOMATO BACON AND LETTUCE SANDWICH, TOMATO LETTUCE AND BACON SANDWICH 9 / 23 / COLD CONSOMME IN CUP (9 rows) # Choices in cluster 2 — 11 # Rows in cluster 2 — 48 Average length of choices 0 — 250 Length variance of choices 0 — 9.5
Drop Column	OpenRefine	‘description’	This column does not have any data populated and hence it is dropped	
Drop rows with ‘name’ = NULL	R	‘name’	The dish name needs to be a non null value in order for our use case query to work. These rows with missing ‘name’ details do not add any information to our Use Case U1 results and are rather misleading and hence they are dropped.	
Impute ‘times_appeared’ to non negative Number dish_cleaned[(dish_cleaned$times_appeared <= 0),"times_appeared"] = 1	R	‘times_appeared’	There cannot be negative values in the ‘times_appeared’ column. It can either be 1(since a dish has already made it to a particular menu) or a finite number greater than 1. Hence imputed all the negative values to 1. This column is not used in our use case query but it would helpful to render meaningful insights later if we plan to use this column	
Impute ‘first_appeared’ & ‘last_appeared’ dish_cleaned[dish_cleaned$first_appeared < 1840,"first_appeared"] = 1840 dish_cleaned[dish_cleaned$first_appeared > 2023,"first_appeared"] = 2023 dish_cleaned[dish_cleaned$last_appeared < 1840,"last_appeared"]	R	‘first_appeared’, ‘last_appeared’	These 2 columns are not used in our Use case query but would be helpful to render meaningful insights later if we plan to use this column. These are columns that denote the year in which a dish has first/last appeared on a menu. There are some values with 0, 1 and 2928 which have been imputed.	

ned[dish_cleaned$last_appeared > 2023,"last_appeared"]				

c. **Menu_item.csv**

Data Cleaning Step	**Tool Used to Perform the Data Cleaning Task**	**Columns Applied to**	**Reason for application**
Create a ‘high_price_new’ variable menu_item$high_price_new = ifelse(!is.na(menu_item$high_price), menu_item$high_price, ifelse(!is.na(menu_item$price), menu_item$price, NA))	R	‘price’, ‘high_price’	It is important to identify the most expensive dish for the use case and hence we need a variable which has the highest price of the dish at any given time. If the ‘high_price’ was missing, we take the value from ‘price’ to be the highest price and if both ‘price’ and ‘high_price’ are missing, we set this variable to be NA
Drop the rows which have ‘high_price_new’ = NULL	R	‘high_price_new’	The rows with missing price values do not contribute to the our use case U1 and hence we drop all the records were ‘high_price_new’ = NULL
Outlier Removal: Keep the rows which have ‘high_price_new’ < $500	R	‘high_price_new’	The rationale behind this step is that it would make our use case query result skewed and thus the result will no longer be representative of the true insight. There were 125 records out of the 56241 (< 0.2%) of the data records which had price > 500 and were dropped from the query

d. **Menu_page.csv**
No Data Cleanup required for menu_page.csv. It is left as is

Our use case U1 is “***To find the most popular dish and then find out which venues serve the most expensive version of the most popular dish***”

For this we use the cleaned data (the process of which is captured in the workflow section), perform an inner-join on ‘menu_cleaned’ and ‘menu_page’ and get the ‘menu_page_id’ column which is one the key required to join with the other dataframes. Then we perform an inner-join on ‘menu_item’ and ‘dish’ to get the ‘menu_page_id’ and the ‘dish_id’. In the next step we perform an inner-join on the above 2 inner-join dataframes by ‘menu_page_id’ to get all the columns into 1 single dataframe. Duplicate records if any are eliminated and we filter the ‘currency’ = ‘DOLLARS’ and then apply ‘event’ = ‘DINNER’ filter. The highest price of the dish is recorded. Some dishes had exorbitant prices like $2500, which were, on careful consideration, considered anomalies and filtered by applying a price filter which essentially took the records which had the dish price equals less than $500. We sort the resulting dataframe by ‘menus_appeared’ and ‘price’ in descending order and the result would be the Most popular dishes in descending order. We take the first entry which happens to be ‘dish’ = ‘COFFEE’, and then take the venues which serve the most expensive version of Coffee. The results are shown on the bar charts shown at the start of section #2.

2. Document Data Quality changes

Quantifying Row / Column Changes between D and D’

The table below depicts the size differences between the D and D’ datasets for the tables. The notation in the table is [row count, column count]. We can see that Menu had the greatest size reduction losing about 97% of its total rows and overall cell count. We’ve discussed how the data has been cleaned, but the data below shows the D’ datasets will have higher query throughput given the overall reduction in records for the relevant table.

Table Name	D shape	D' shape	Row Count Reduction [%]	Overall Cell Count Reduction [%]
Menu	[17545, 20]	[446, 17]	97.46	97.84
Dish	[423397, 9]	[423386, 8]	0.00	11.11
Menu Item	[1332726, 9]	[886868, 5]	33.45	63.03

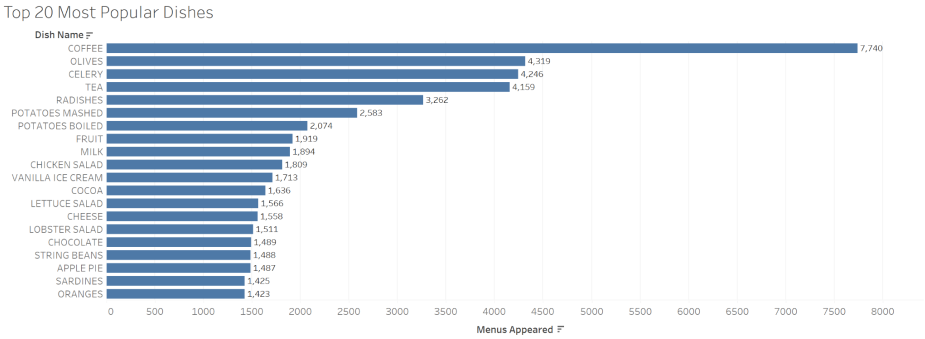
Data quality has indeed improved drastically between the original dataset D and the newly cleaned dataset D’. This is demonstrated below by querying a set of integrity constraints and documenting their corresponding violations (ICVs). In total, D had 21,088 rows with reported ICVs, whereas D’ had 0.

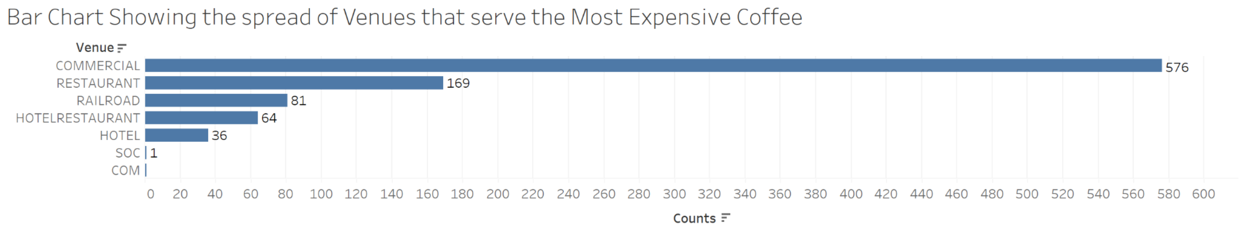
Data Cleaning Visualizations

The barcharts below help visualize some of the differences between the D and D’ datasets for our U1 use case.

U1 Query Results on the Cleaned Dataset (D’)

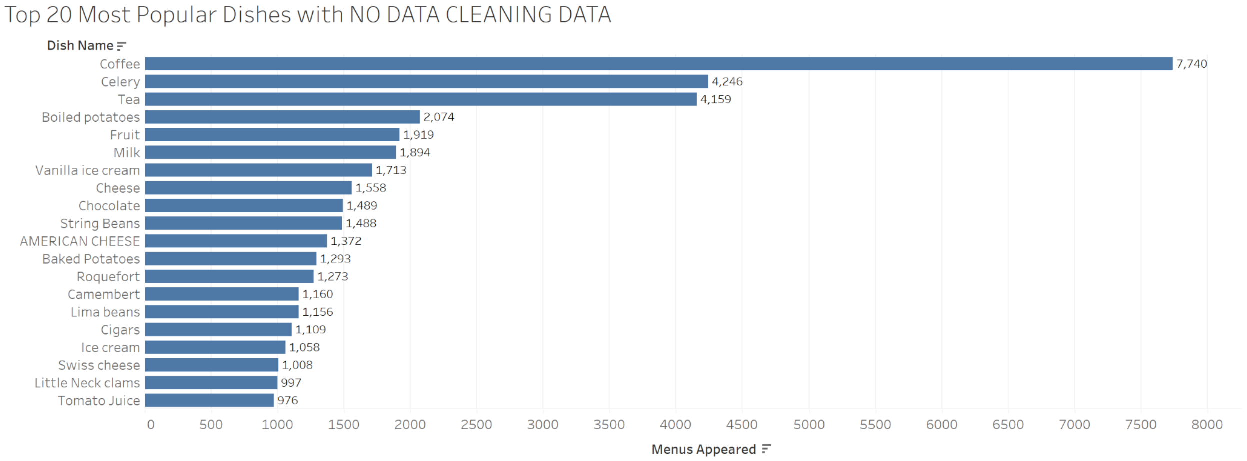
The bar chart below is derived using ‘clean’ data which shows the 20 most popular dishes with the most popular on the top. The second bar chart below is derived using ‘clean’ data which shows the spread of venues that serve the most expensive coffee (the most popular dish).

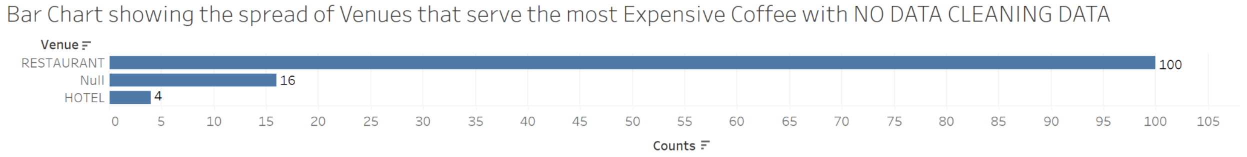




U1 Query Results on the Original Dataset (D)

The bar charts below are the same queries, but on the original dataset. Although, the most popular item happens to be "Coffee" in the below list, it is evident that the list of the most popular dishes is very different from the popular dishes list generated by using 'clean' data. It's also worth highlighting the delta between the spread of venues between the original and 'clean' datasets in the second chart. The chart also shows one of the items to be NULL which is misleading and cannot take away any meaningful insight from it. The charts below add evidence to the point that data cleaning is not only important but essential.



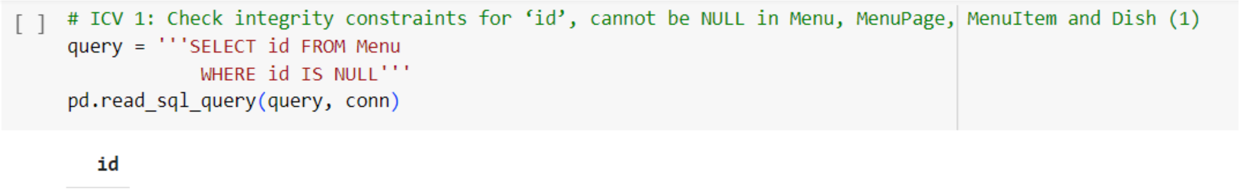


Integrity Constraint Violation Report

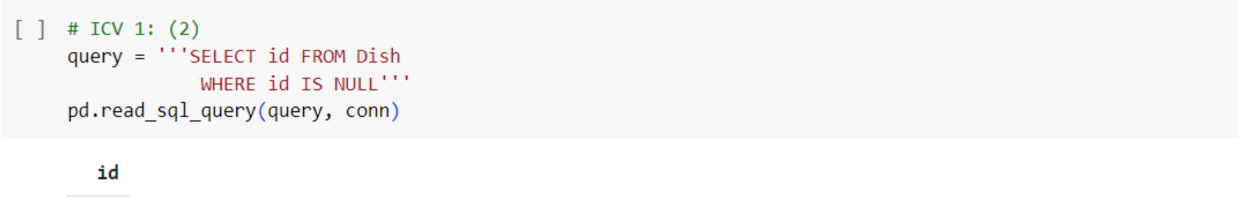
For the ICV queries below, we note that we will run the same query but for a new table generated from the new dataset (e.g. Dish and DishCleaned, Menu and MenuCleaned, etc.). The only exception for naming is MenuJoinMenuPage, as this is the result of joining the Menu and MenuPage dataset when cleaning the tables in R. As MenuPage didn't require any cleaning by itself, and the ICVs concerned were taken care of by a simple merge, this table will be more helpful to analyze. As such, some fields will be necessarily different (e.g. instead of checking 'id' we will check 'menu_page_id', etc.)

1. **<u>“Check integrity constraints for ‘id’, cannot be NULL in Menu, MenuPage, MenuItem and Dish.”</u>** This ICV report intended to expose a very basic assumption of the data, namely that the primary key ‘id’ must be non-NULL. Any such row with a NULL would need to otherwise be removed, as it violates a fundamental assumption we have for the data, especially for our use case U1. As demonstrated in the images below, the original dataset already did not have any ICVs, and we ensured that this remained true in our cleaned dataset (i.e. for this particular constraint, D and D’ are equally clean).

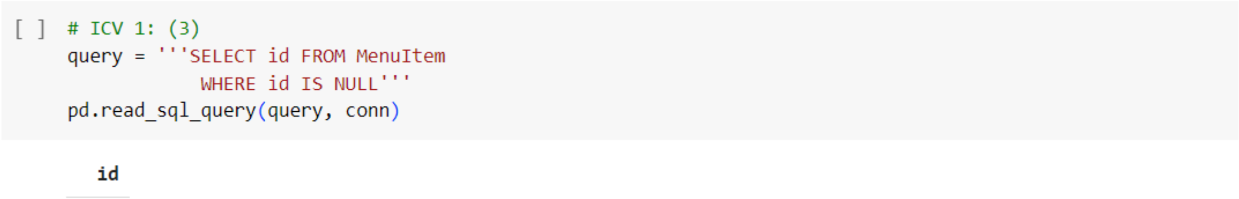
Original Dataset



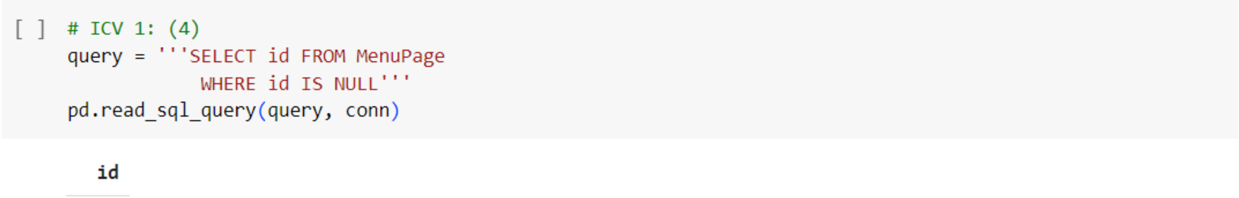
```
[ ]  # ICV 1: Check integrity constraints for ‘id’, cannot be NULL in Menu, MenuPage, MenuItem and Dish (1)
     query = '''SELECT id FROM Menu
                WHERE id IS NULL'''
     pd.read_sql_query(query, conn)

        id
```



```
[ ]  # ICV 1: (2)
     query = '''SELECT id FROM Dish
                WHERE id IS NULL'''
     pd.read_sql_query(query, conn)

        id
```

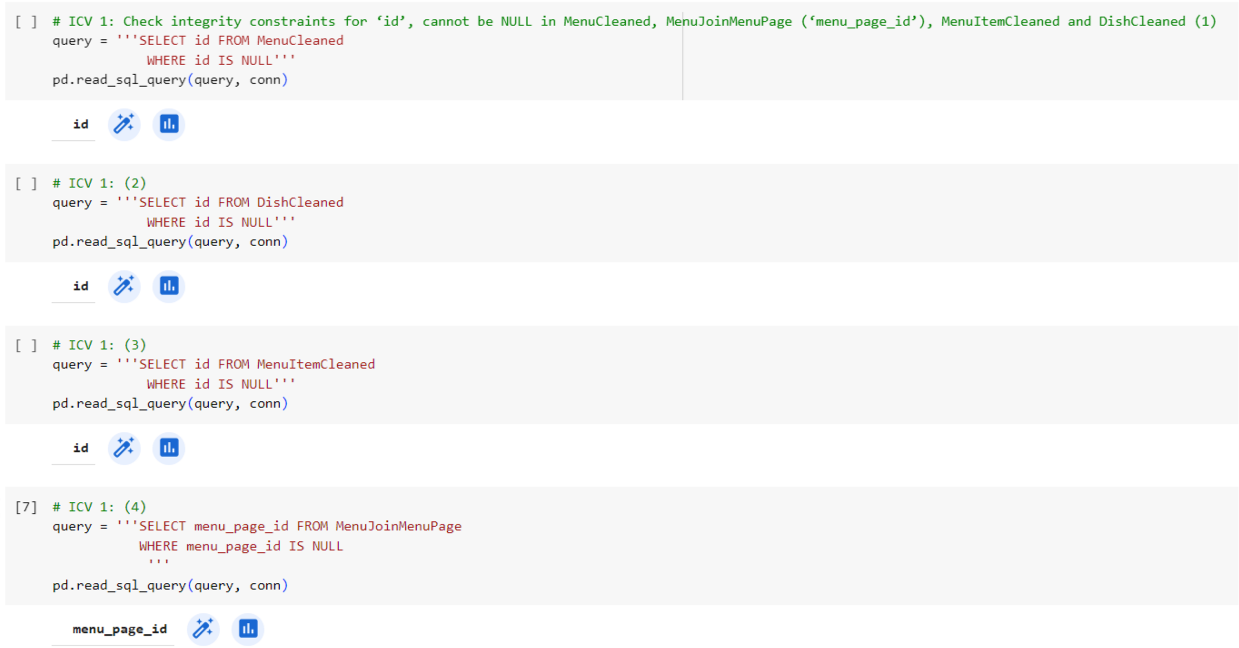


```
[ ]  # ICV 1: (3)
     query = '''SELECT id FROM MenuItem
                WHERE id IS NULL'''
     pd.read_sql_query(query, conn)

        id
```



```
[ ]  # ICV 1: (4)
     query = '''SELECT id FROM MenuPage
                WHERE id IS NULL'''
     pd.read_sql_query(query, conn)

        id
```

Cleaned Dataset



```
[ ] # ICV 1: Check integrity constraints for ‘id’, cannot be NULL in MenuCleaned, MenuJoinMenuPage (‘menu_page_id’), MenuItemCleaned and DishCleaned (1)
    query = '''SELECT id FROM MenuCleaned
                WHERE id IS NULL'''
    pd.read_sql_query(query, conn)
```

```
id
```

```
[ ] # ICV 1: (2)
    query = '''SELECT id FROM DishCleaned
                WHERE id IS NULL'''
    pd.read_sql_query(query, conn)
```

```
id
```

```
[ ] # ICV 1: (3)
    query = '''SELECT id FROM MenuItemCleaned
                WHERE id IS NULL'''
    pd.read_sql_query(query, conn)
```

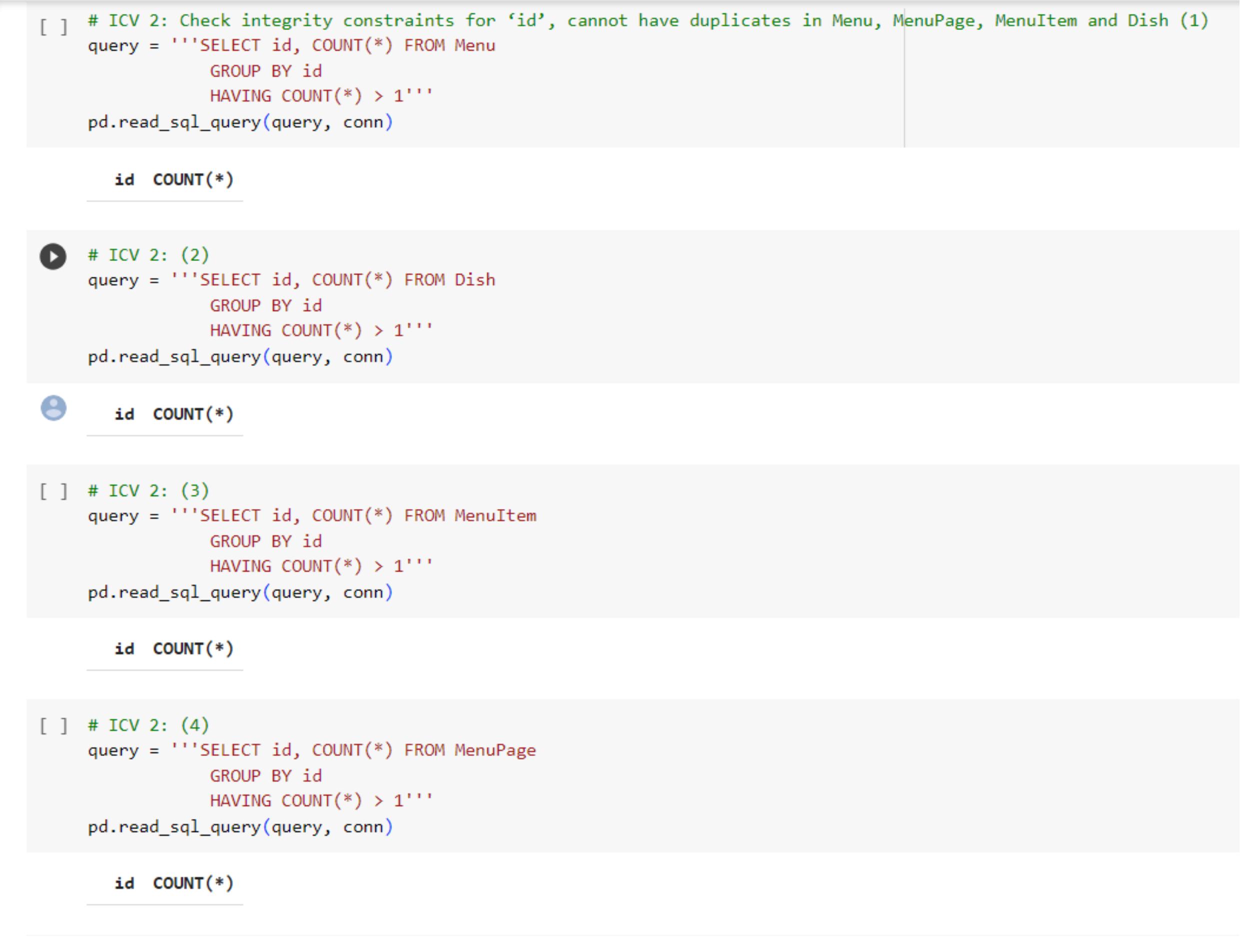
```
id
```

```
[7] # ICV 1: (4)
    query = '''SELECT menu_page_id FROM MenuJoinMenuPage
               WHERE menu_page_id IS NULL
                '''
    pd.read_sql_query(query, conn)
```

```
menu_page_id
```

2. **“Check integrity constraints for ‘id’, cannot have duplicates in Menu, MenuPage, MenuItem and Dish.”** This ICV report intended to expose another assumption of the data, that the primary key ‘id’ must not have duplicates, as this would violate id uniqueness. Any such row with duplicate ‘id’ would need to otherwise be removed, as it violates another assumption we have for the data, especially for our use case U1. As demonstrated in the images below, the original dataset already did not have any ICVs, and we ensured that this remained true in our cleaned dataset (i.e. for this particular constraint, D and D’ are equally clean).

Original Dataset



```
# ICV 2: Check integrity constraints for ‘id’, cannot have duplicates in Menu, MenuPage, MenuItem and Dish (1)
query = '''SELECT id, COUNT(*) FROM Menu
            GROUP BY id
            HAVING COUNT(*) > 1'''
pd.read_sql_query(query, conn)
```

```
id  COUNT(*)
```

```
# ICV 2: (2)
query = '''SELECT id, COUNT(*) FROM Dish
            GROUP BY id
            HAVING COUNT(*) > 1'''
pd.read_sql_query(query, conn)
```

```
id  COUNT(*)
```

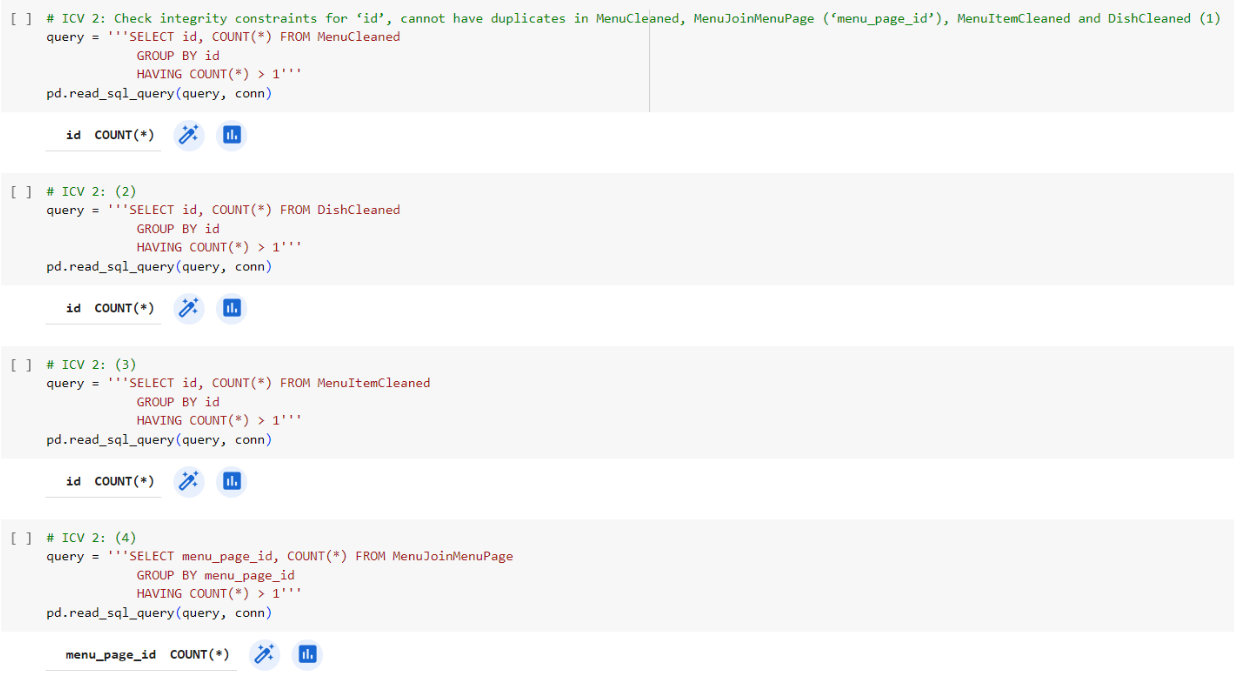
```
# ICV 2: (3)
query = '''SELECT id, COUNT(*) FROM MenuItem
            GROUP BY id
            HAVING COUNT(*) > 1'''
pd.read_sql_query(query, conn)
```

```
id  COUNT(*)
```

```
# ICV 2: (4)
query = '''SELECT id, COUNT(*) FROM MenuPage
            GROUP BY id
            HAVING COUNT(*) > 1'''
pd.read_sql_query(query, conn)
```

```
id  COUNT(*)
```

Cleaned Dataset



```
# ICV 2: Check integrity constraints for ‘id’, cannot have duplicates in MenuCleaned, MenuJoinMenuPage (‘menu_page_id’), MenuItemCleaned and DishCleaned (1)
query = '''SELECT id, COUNT(*) FROM MenuCleaned
            GROUP BY id
            HAVING COUNT(*) > 1'''
pd.read_sql_query(query, conn)
```

```
id  COUNT(*)
```

```
# ICV 2: (2)
query = '''SELECT id, COUNT(*) FROM DishCleaned
            GROUP BY id
            HAVING COUNT(*) > 1'''
pd.read_sql_query(query, conn)
```

```
id  COUNT(*)
```

```
# ICV 2: (3)
query = '''SELECT id, COUNT(*) FROM MenuItemCleaned
            GROUP BY id
            HAVING COUNT(*) > 1'''
pd.read_sql_query(query, conn)
```

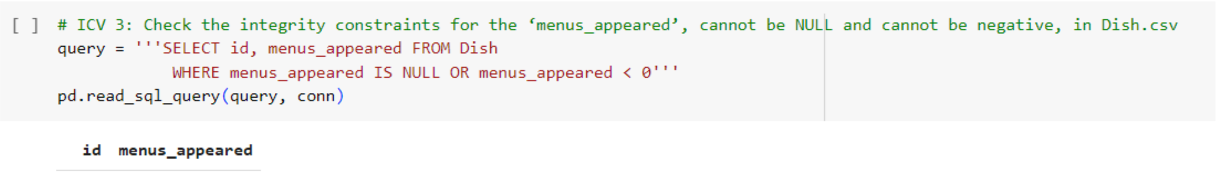
```
id  COUNT(*)
```

```
# ICV 2: (4)
query = '''SELECT menu_page_id, COUNT(*) FROM MenuJoinMenuPage
            GROUP BY menu_page_id
            HAVING COUNT(*) > 1'''
pd.read_sql_query(query, conn)
```

```
menu_page_id  COUNT(*)
```

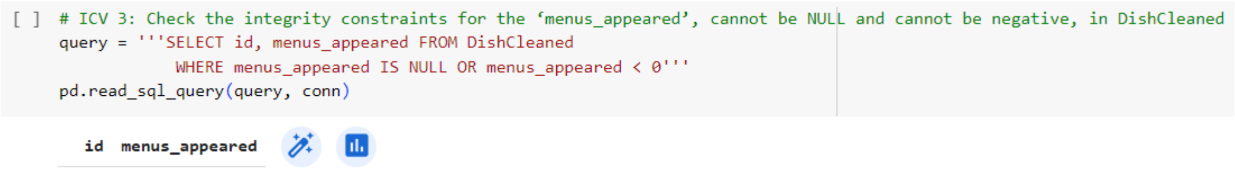
3. **<u>“Check the integrity constraints for the ‘menus_appeared’, cannot be NULL and cannot be negative, in Dish.”</u>** This ICV report intended to expose another basic assumption, that menus_appeared must be a non-null, non-negative value. This constraint was held for both D and D’, so for this integrity constraint, D and D’ are equally clean.

Original Dataset



```
[ ] # ICV 3: Check the integrity constraints for the ‘menus_appeared’, cannot be NULL and cannot be negative, in Dish.csv
    query = '''SELECT id, menus_appeared FROM Dish
               WHERE menus_appeared IS NULL OR menus_appeared < 0'''
    pd.read_sql_query(query, conn)

      id  menus_appeared
```

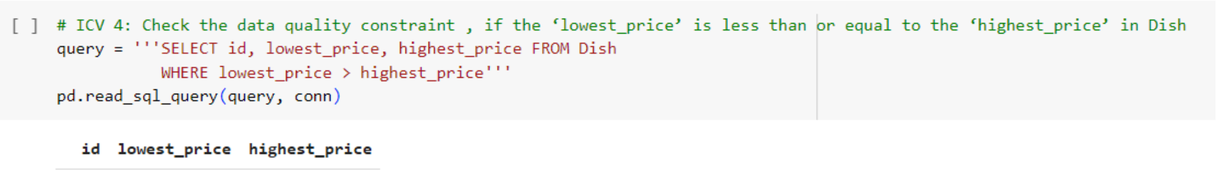
Cleaned Dataset



```
[ ] # ICV 3: Check the integrity constraints for the ‘menus_appeared’, cannot be NULL and cannot be negative, in DishCleaned
    query = '''SELECT id, menus_appeared FROM DishCleaned
               WHERE menus_appeared IS NULL OR menus_appeared < 0'''
    pd.read_sql_query(query, conn)

      id  menus_appeared
```

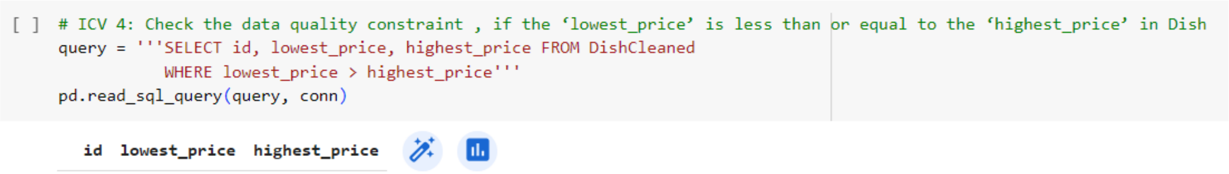
4. **<u>“Check the data quality constraint , if the ‘lowest_price’ is less than or equal to the ‘highest_price’ in Dish”</u>** This ICV report intended to expose any violations of the fact that lowest price of a dish should logically be less than or equal to the highest price of a dish. This was particularly important to ensure for U1, as any data which contained this contradiction that might have appeared in the most expensive dishes in the U1 query and hence could have impacted the results of the use case. However, this constraint was held for both D and D’, so for this integrity constraint, D and D’ are equally clean.

Original Dataset



```
[ ] # ICV 4: Check the data quality constraint , if the ‘lowest_price’ is less than or equal to the ‘highest_price’ in Dish
    query = '''SELECT id, lowest_price, highest_price FROM Dish
               WHERE lowest_price > highest_price'''
    pd.read_sql_query(query, conn)

      id  lowest_price  highest_price
```

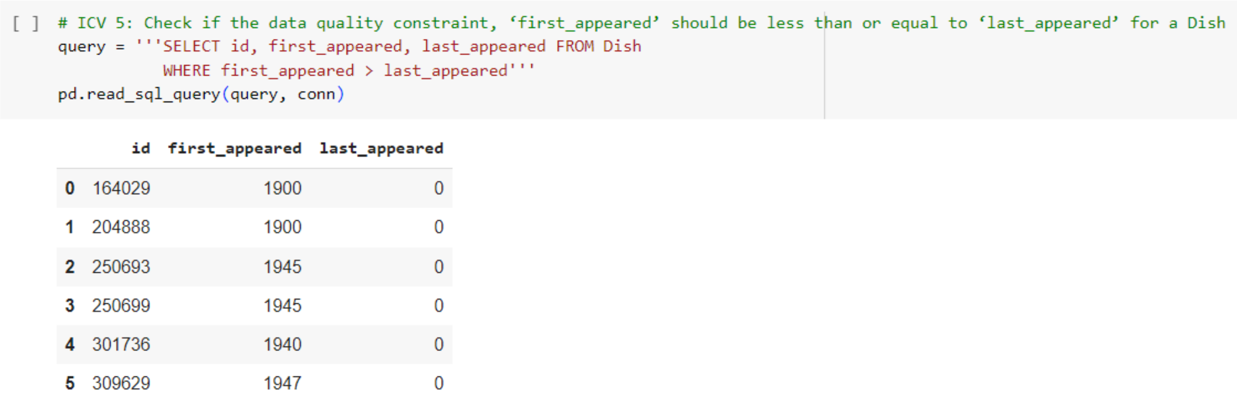
Cleaned Dataset



```
[ ] # ICV 4: Check the data quality constraint , if the ‘lowest_price’ is less than or equal to the ‘highest_price’ in Dish
    query = '''SELECT id, lowest_price, highest_price FROM DishCleaned
               WHERE lowest_price > highest_price'''
    pd.read_sql_query(query, conn)

      id  lowest_price  highest_price
```

5. **<u>“Check if the data quality constraint, ‘first_appeared’ should be less than or equal to ‘last_appeared’ for Dish.”</u>** The ICV report intended to reveal the data which violated a basic logical assumption, namely that when a dish first appeared in a menu, it should have last appeared at a later date. As seen below, we found in the original dataset that there were 6 instances, where ‘last_appeared’ is equal to 0, which is at an earlier year than ‘first_appeared’. Now, this could have just been a placeholder value, but it still indicates a key missing attribute and should be thus regarded as incorrect. As seen in the cleaned dataset ICV report, the cleaned dataset has no such violations, and hence for this integrity constraint, D’ is cleaner than D (for 6 rows of data).

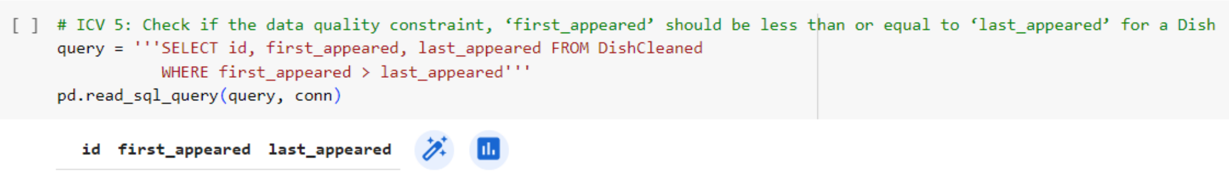
Original Dataset



```
[ ] # ICV 5: Check if the data quality constraint, ‘first_appeared’ should be less than or equal to ‘last_appeared’ for a Dish
    query = '''SELECT id, first_appeared, last_appeared FROM Dish
               WHERE first_appeared > last_appeared'''
    pd.read_sql_query(query, conn)
```

	id	first_appeared	last_appeared
0	164029	1900	0
1	204888	1900	0
2	250693	1945	0
3	250699	1945	0
4	301736	1940	0
5	309629	1947	0

Cleaned Dataset

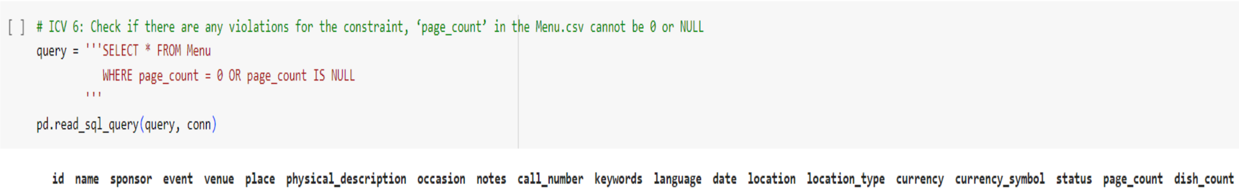


```
[ ] # ICV 5: Check if the data quality constraint, ‘first_appeared’ should be less than or equal to ‘last_appeared’ for a Dish
    query = '''SELECT id, first_appeared, last_appeared FROM DishCleaned
               WHERE first_appeared > last_appeared'''
    pd.read_sql_query(query, conn)
```

id	first_appeared	last_appeared

6. **<u>“Check if there are any violations for the constraint, ‘page_count’ in the Menu.csv cannot be 0 or NULL.”</u>** This ICV report intended to expose whether there were any rows of data with a page_count violating a logical assumption, that page_count should not be 0 (e.g. a pageless Menu) or NULL (a Menu with some missing data). In this case, we have that both D and D’ both satisfy this constraint, and are thus in this regard equally clean.

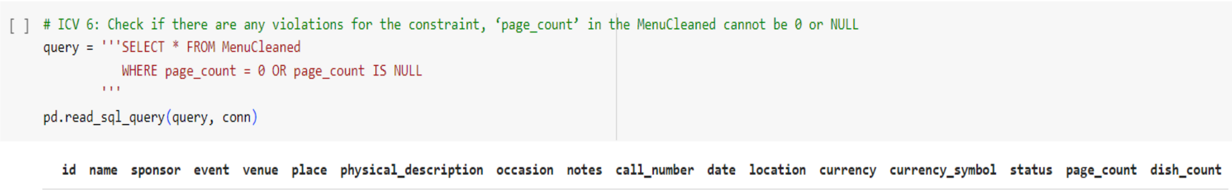
Original Dataset



```
[ ] # ICV 6: Check if there are any violations for the constraint, ‘page_count’ in the Menu.csv cannot be 0 or NULL
    query = '''SELECT * FROM Menu
               WHERE page_count = 0 OR page_count IS NULL
            '''
    pd.read_sql_query(query, conn)
```

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

Cleaned Dataset

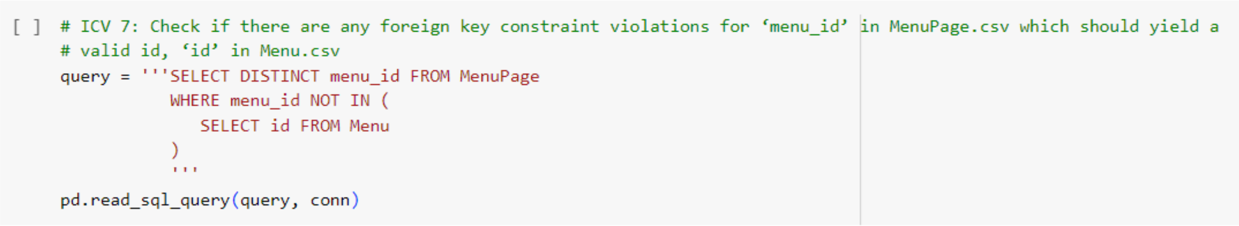


```
[ ]  # ICV 6: Check if there are any violations for the constraint, ‘page_count’ in the MenuCleaned cannot be 0 or NULL
     query = '''SELECT * FROM MenuCleaned
                WHERE page_count = 0 OR page_count IS NULL
             '''
     pd.read_sql_query(query, conn)
```

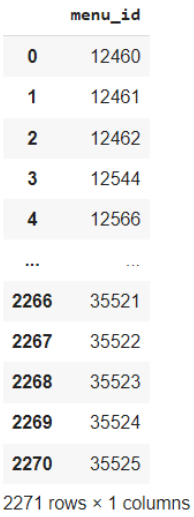
id	name	sponsor	event	venue	place	physical_description	occasion	notes	call_number	date	location	currency	currency_symbol	status	page_count	dish_count

7. **“Check if there are any foreign key constraint violations for ‘menu_id’ in MenuPage which should yield a valid id, ‘id’ in Menu.”** This ICV report intended to reveal which foreign keys (menu_id) violated a basic foreign key constraint, namely that a foreign key should be pointing to a valid key; in this case, the foreign key ‘menu_id’ in MenuPage should refer to a valid key ‘id’ in Menu. As shown in the ICV reports below, the original dataset had a large number of ICVs: 2271 distinct foreign keys which violated the IC. However, the cleaned dataset fared much better, with 0 violations. Hence, in regards to this integrity constraint, D’ is cleaner than D.

Original Dataset



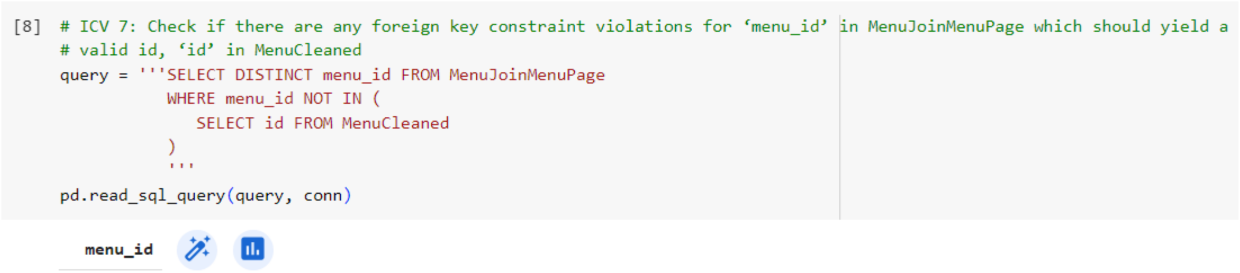
```
[ ]  # ICV 7: Check if there are any foreign key constraint violations for ‘menu_id’ in MenuPage.csv which should yield a
     # valid id, ‘id’ in Menu.csv
     query = '''SELECT DISTINCT menu_id FROM MenuPage
                WHERE menu_id NOT IN (
                   SELECT id FROM Menu
                )
                '''
     pd.read_sql_query(query, conn)
```



	menu_id
0	12460
1	12461
2	12462
3	12544
4	12566
...	...
2266	35521
2267	35522
2268	35523
2269	35524
2270	35525

2271 rows × 1 columns

Cleaned Dataset

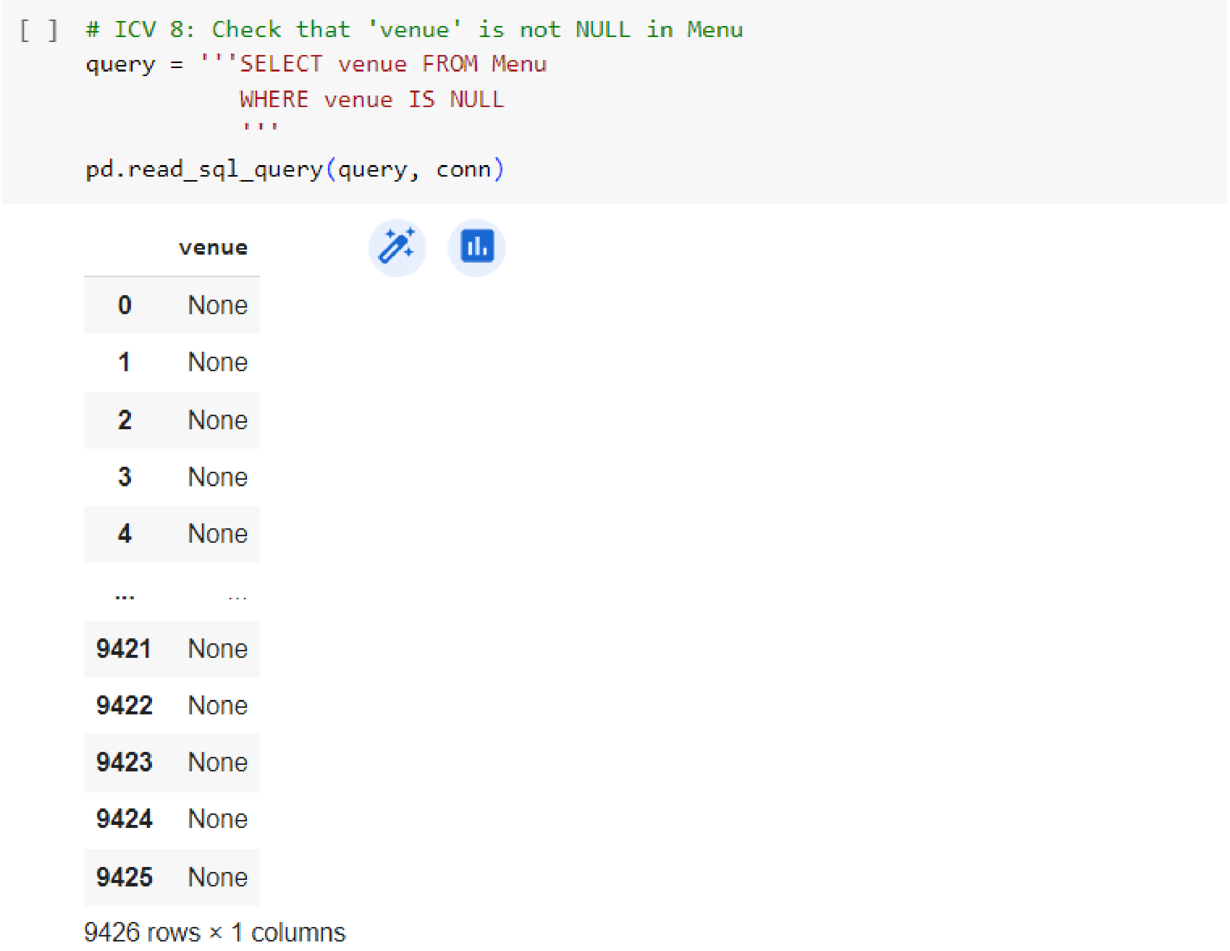


```
[8] # ICV 7: Check if there are any foreign key constraint violations for ‘menu_id’ in MenuJoinMenuPage which should yield a
    # valid id, ‘id’ in MenuCleaned
    query = '''SELECT DISTINCT menu_id FROM MenuJoinMenuPage
               WHERE menu_id NOT IN (
                  SELECT id FROM MenuCleaned
               )
               '''
    pd.read_sql_query(query, conn)
```

```
menu_id
```

8. **<u>“Check that 'venue' is not NULL in Menu.”</u>** This ICV report intended to expose which venues were missing data. For our use case U1, this would be especially important, as in order to find the venues which serve the most expensive dish, we would want to ensure that the venue actually has a name. As shown below, the original dataset has a whopping 9426 rows of data with missing venue information, whereas the cleaned dataset has 0. Hence, for this constraint, D’ is cleaner than D.

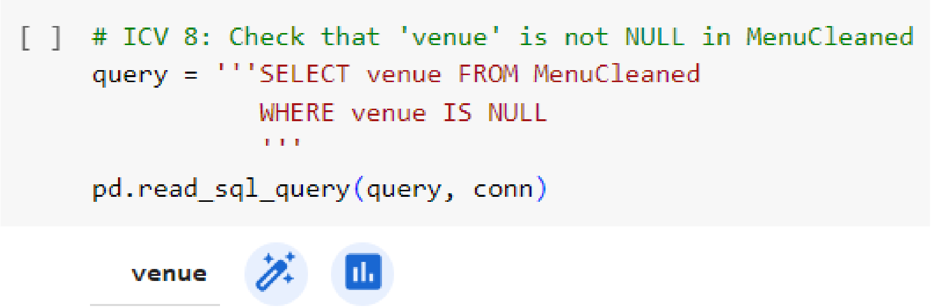
Original Dataset



```
[ ] # ICV 8: Check that 'venue' is not NULL in Menu
    query = '''SELECT venue FROM Menu
               WHERE venue IS NULL
               '''
    pd.read_sql_query(query, conn)
```

```
        venue
   0     None
   1     None
   2     None
   3     None
   4     None
 ...      ...
9421     None
9422     None
9423     None
9424     None
9425     None

9426 rows × 1 columns
```

Cleaned Dataset

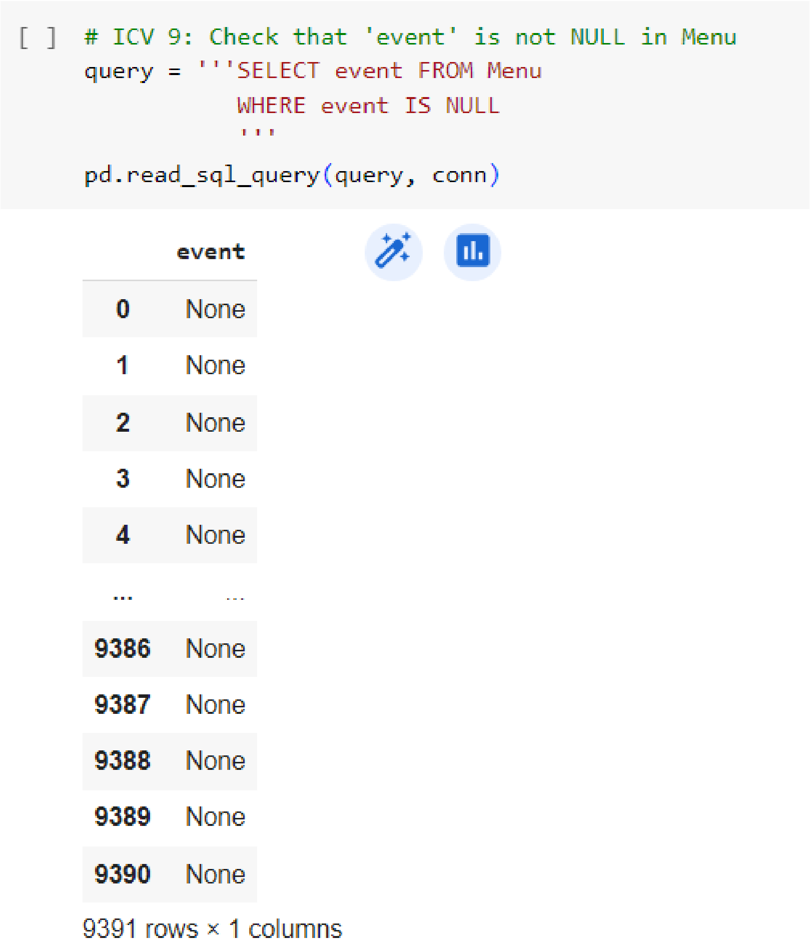


```
[ ] # ICV 8: Check that 'venue' is not NULL in MenuCleaned
    query = '''SELECT venue FROM MenuCleaned
               WHERE venue IS NULL
               '''
    pd.read_sql_query(query, conn)
```

```
venue
```

9. **“<u>Check that ‘event’ is not NULL in Menu.</u>”** This ICV report intended to reveal which events were missing data and hence should be removed for the purposes of event analysis. As shown below, the original dataset has a large number of rows missing data with 9391 rows, whereas the cleaned dataset has 0. Hence, for this constraint, D’ is cleaner than D.

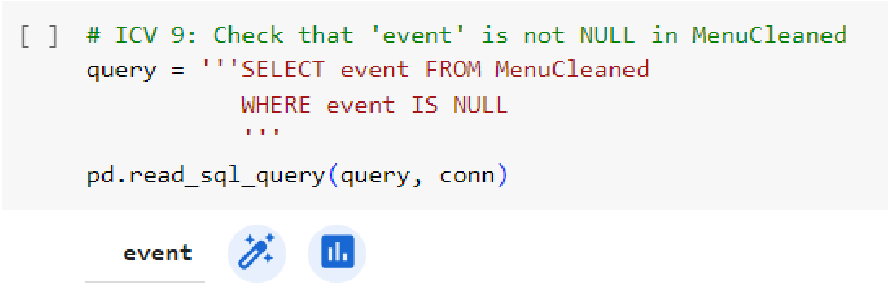
Original Dataset



```
[ ] # ICV 9: Check that 'event' is not NULL in Menu
    query = '''SELECT event FROM Menu
               WHERE event IS NULL
               '''
    pd.read_sql_query(query, conn)
```

```
        event
   0    None
   1    None
   2    None
   3    None
   4    None
 ...     ...
9386    None
9387    None
9388    None
9389    None
9390    None
9391 rows × 1 columns
```

Cleaned Dataset



```
[ ] # ICV 9: Check that 'event' is not NULL in MenuCleaned
    query = '''SELECT event FROM MenuCleaned
               WHERE event IS NULL
               '''
    pd.read_sql_query(query, conn)
```

```
event
```

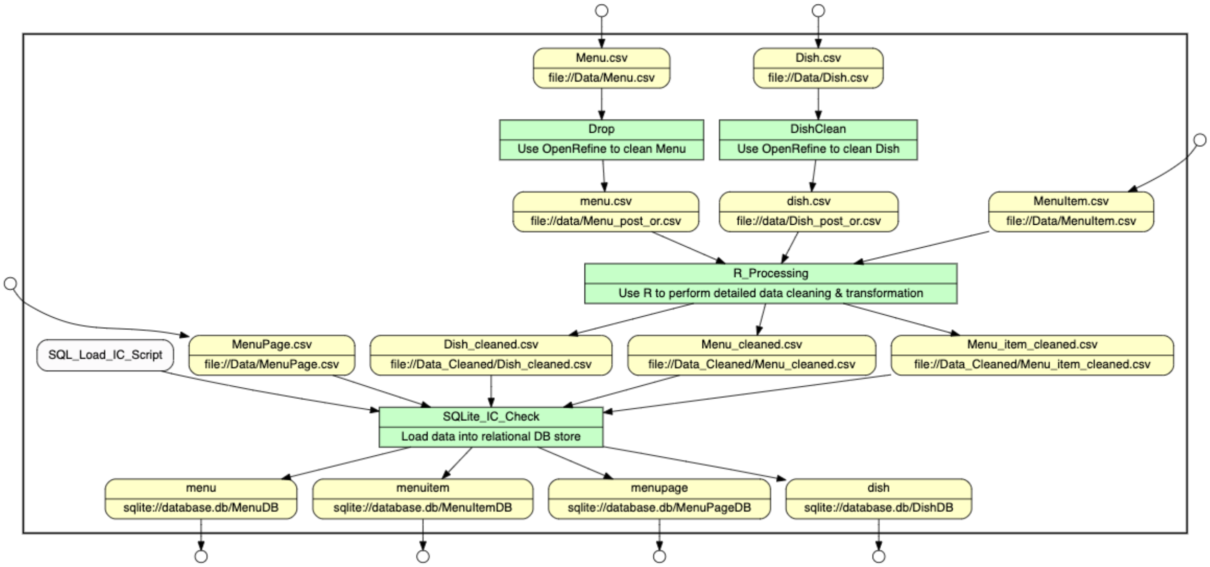
3. Create a workflow model

When performing data cleaning, it is often useful to have workflow models to assist with capturing provenance details. In our case, we use aspects of both prospective and retrospective provenance to outline our process. There are four main workflow models shown below (1) Outer Workflow (2) Inner Workflow - R (3) Inner Workflow - OpenRefine (Dishes) and (4) Inner Workflow OpenRefine (Menu). For the latter 2, we only depict the data views to maintain readability of the diagrams. Refer to our Github repo to see the combined views and the relevant .gv, .yw files that were used to produce them.

We chose YesWorkflow, OpenRefine, R, and SQLite as they are tools that our team members are familiar with or we learned them during the course. The idea was to choose technologies that matched with our team's skillset so we could focus more on the objectives of the project.

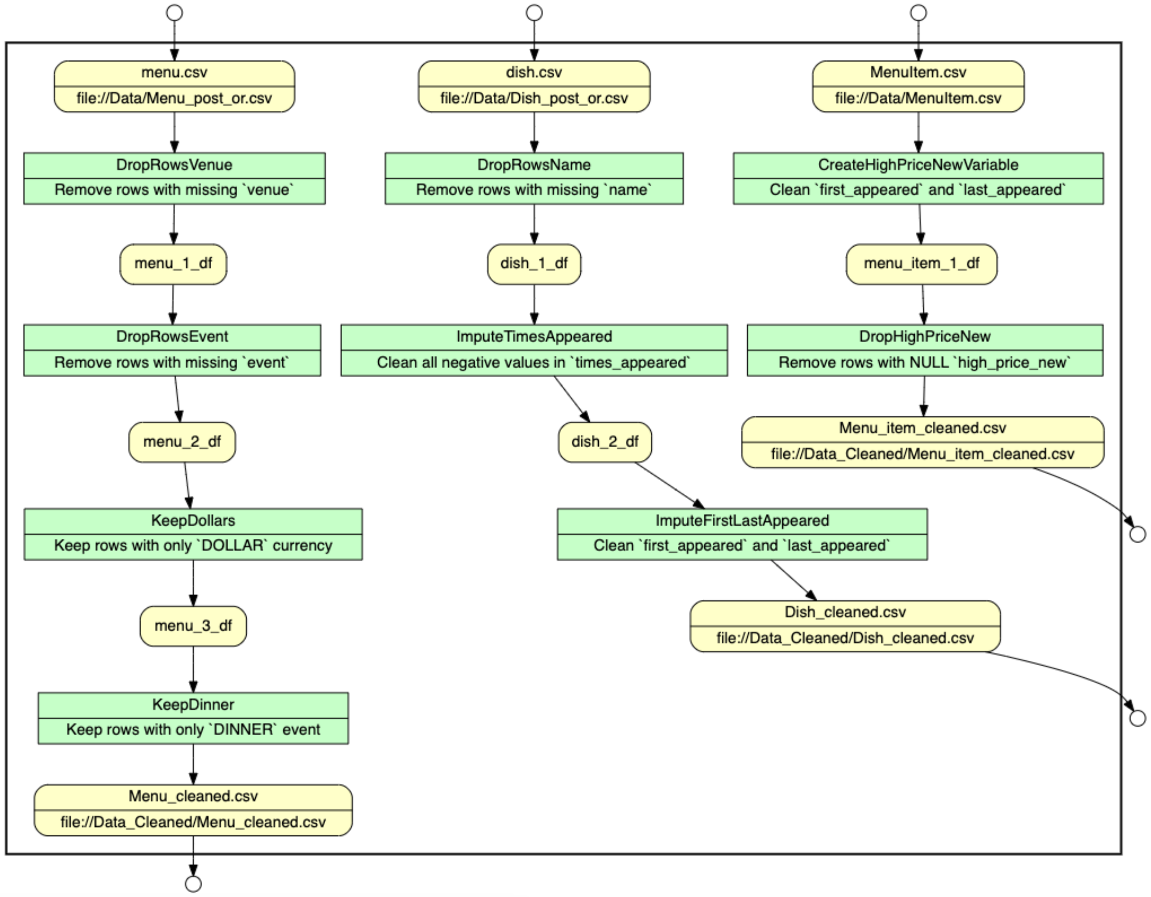
The Outer Workflow below can be seen as a form of prospective provenance, to outline our overall data processing flow without containing the detailed operations. This workflow was chosen to optimize for getting the cleanest data in preparation for our main use case. At a high-level, we use OpenRefine to perform a variety of data cleaning (see sections 1a, 1b) for details. There were certain datasets we didn't need for our use case at this point. We then perform a second level of detailed data cleaning using R (see sections 1a, 1b, and 1c). Finally, once the CSVs are cleaned, we load them into SQLite using a load script and perform the ICV check. After that, our final dataset is ready to be queried.

Outer Workflow

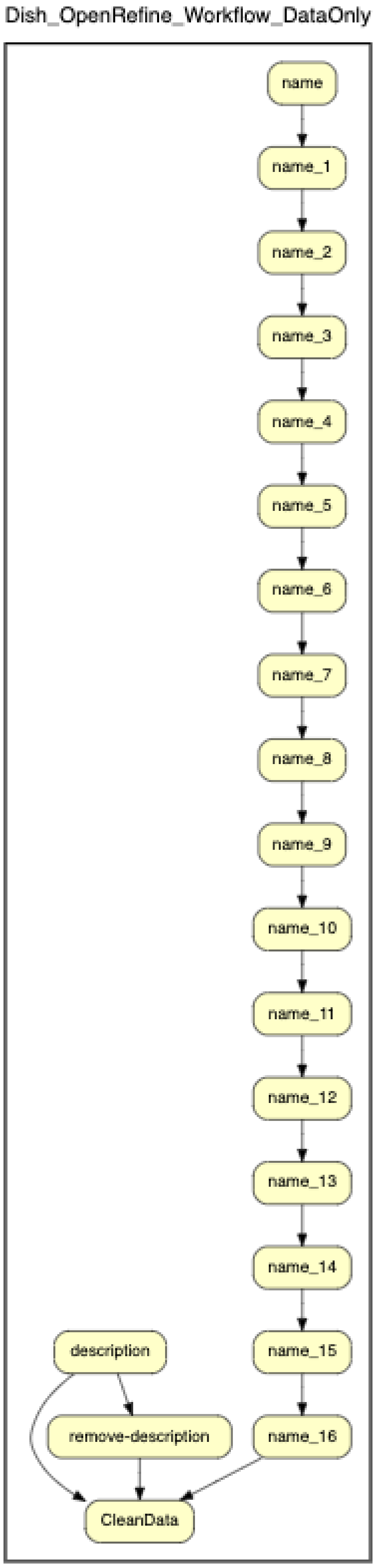

Menu.csv
file://Data/Menu.csv
Dish.csv
file://Data/Dish.csv
Drop
Use OpenRefine to clean Menu
DishClean
Use OpenRefine to clean Dish
menu.csv
file://data/Menu_post_or.csv
dish.csv
file://data/Dish_post_or.csv
MenuItem.csv
file://Data/MenuItem.csv
R_Processing
Use R to perform detailed data cleaning & transformation
SQL_Load_IC_Script
MenuPage.csv
file://Data/MenuPage.csv
Dish_cleaned.csv
file://Data_Cleaned/Dish_cleaned.csv
Menu_cleaned.csv
file://Data_Cleaned/Menu_cleaned.csv
Menu_item_cleaned.csv
file://Data_Cleaned/Menu_item_cleaned.csv
SQLite_IC_Check
Load data into relational DB store
menu
sqlite://database.db/MenuDB
menuitem
sqlite://database.db/MenuItemDB
menupage
sqlite://database.db/MenuPageDB
dish
sqlite://database.db/DishDB

The three Inner Workflow diagrams below show the trace operations from the data cleaning tools we used. In the R example, we build the workflow diagram from the bottom up, aligning with the data terminology in the Outer Workflow for consistency. The latter two workflow diagrams, we utilized the or2yw which made it simple to take the operations from an OpenRefine trace file and convert it into a workflow diagram. It's worth highlighting that these diagrams are quite challenging to follow given the # of operations, so we reduced the view to the data nodes only.

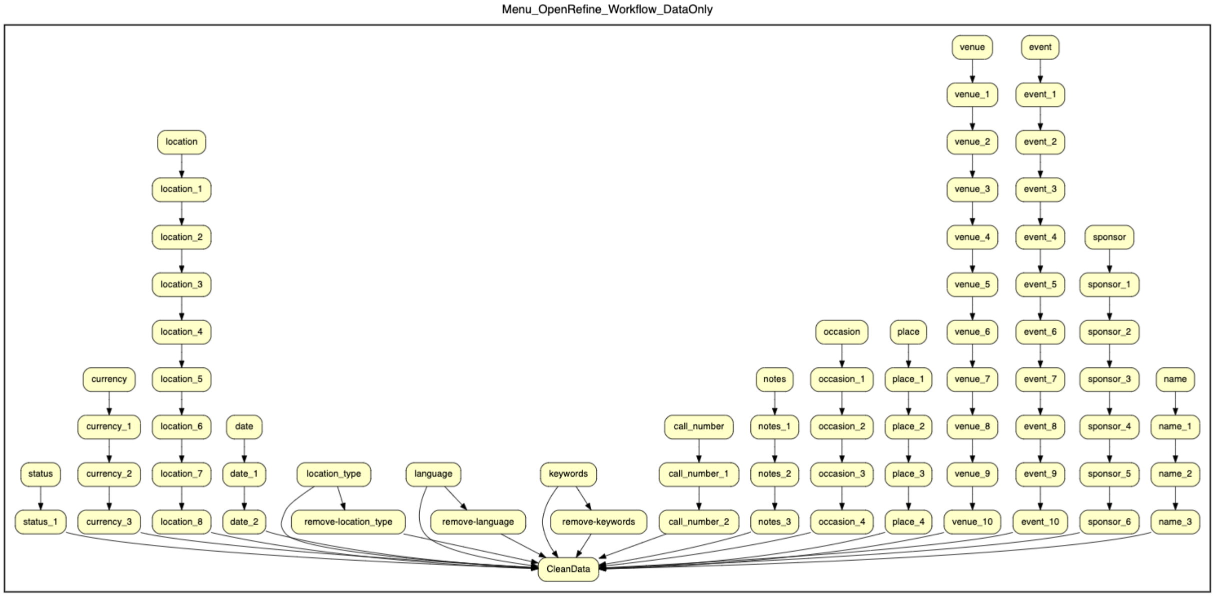
Inner Workflow - R


menu.csv
file://Data/Menu_post_or.csv
DropRowsVenue
Remove rows with missing `venue`
menu_1_df
DropRowsEvent
Remove rows with missing `event`
menu_2_df
KeepDollars
Keep rows with only `DOLLAR` currency
menu_3_df
KeepDinner
Keep rows with only `DINNER` event
Menu_cleaned.csv
file://Data_Cleaned/Menu_cleaned.csv
dish.csv
file://Data/Dish_post_or.csv
DropRowsName
Remove rows with missing `name`
dish_1_df
ImputeTimesAppeared
Clean all negative values in `times_appeared`
dish_2_df
ImputeFirstLastAppeared
Clean `first_appeared` and `last_appeared`
Dish_cleaned.csv
file://Data_Cleaned/Dish_cleaned.csv
MenuItem.csv
file://Data/MenuItem.csv
CreateHighPriceNewVariable
Clean `first_appeared` and `last_appeared`
menu_item_1_df
DropHighPriceNew
Remove rows with NULL `high_price_new`
Menu_item_cleaned.csv
file://Data_Cleaned/Menu_item_cleaned.csv

Inner Workflow - OpenRefine (Dish)



Inner Workflow - OpenRefine (Menu)



4. Conclusions & Summary

Summary

Data cleaning is an important and essential step for any kind of data analysis and predictions. It improves the quality of the data enabling end users to be confident with the insights and forecasts generated out of the *clean* data. From our use case it is evident that without data cleaning the results obtained are wrong and misleading. As noted in the ICV reports in section 2, without proper data cleaning, we would have thousands of rows of data violating the most basic assumptions we held about our data, which would have also undoubtedly impacted our use case. The workflow models allow us to understand the data provenance and makes it easier to share insights and the processing workflows with others. The birds eye view that these provenance diagrams provide will help guide future modifications to the dataset. Data cleaning, especially for this dataset with 4 files, is by no means complete and so we plan on adding more Integrity constraints in the future which will help in building use case such as 'predicting the food preference in a region based on the years and events'

Learnings

We used OpenRefine for the majority of data cleaning tasks in this project and what we like about it is the fact that the user still has the control to intervene and select which clusters are similar and should stay together vs which are not unlike the programming languages like Python, R where the user does not have much of control on the algorithm output. OpenRefine was also far more intuitive and user friendly. We also learnt to work with a new tool, the YesWorkflow, it neatly puts all the operational steps in a way. We also found new ways of quantifying the accuracy of the data cleaning tasks by way of writing integrity constraints and by measuring the delta between the *dirty* data and the *cleaned* data. We plan on applying all the skills developed in this course in every data exploration step that we perform.

Challenges:

From our experience data cleaning is by far the most time consuming operation when it comes to working with any dataset. OpenRefine had some memory limitations. It wasn't easy to perform operations on the menu_item.csv which had ~1.3M records. Also, the part we enjoyed for the most part and a little bit daunting was to manually examine the clusters built by OpenRefine.

Team Contribution:

We all enjoyed working with each other and met regularly to plan and work on this project to encourage teamwork.

Sushma	1. Brainstorm and devise the Use case U1 2. Worked on Data cleaning tasks for menu, menu_item, dish, menu_page in OpenRefine 3. Created the R script for data cleaning and generated the clean datasets 4. Built the Use case U1 query 5. Generated Tableau charts 6. Documentation for the report
Michael	1. Brainstormed integrity constraints for new and old dataset 2. Setup Jupyter notebook with creation of tables from csv files and SQLite ICV checks 3. Ran ICV on old and new dataset 4. Analyzed differences in cleanliness of dataset per ICV checks
Sanjeev	1. Brainstorm and help refine U1 2. Learn and use YesWorkflow for workflow modeling tasks 3. Built Outer and Inner Workflow - R diagrams 4. Use or2yw for creating the OpenRefine operations workflow diagrams

Please find the Project repository here GitHub