CS513: Theory and Practice of Data Cleaning
Data Cleaning Project
Phase II Report

Team name: **Team15**
Kesavar Kabilar : kabilar2@illinois.edu
Samarth Patel: spate504@illinois.edu
Benjamin Fridkis: fridkis2@illinois.edu

1. Description of Data Cleaning Performed

We completed our primary data cleaning in two phases[1]. First, for phase I we used OpenRefine to clean data based on "intra-table" issues, or issues that presented themselves entirely within a single table and did not involve cross-table references. We then used datalog (via Clingo) to verify these results, comparing the dirty dataset with the now partially cleaned dataset, to check these intra-table issues and/or IC violations (i.e. checks within a single table). All datalog checks on the (partially) cleaned data came back with 0 violations. In phase II of our data cleaning, we took the partially cleaned data from phase I and ran inter-table IC violation checks (i.e. referential checks between two or more tables). Instances without valid foreign key references were removed, and stored values in the Dish relation (menus_appeared, times_appeared, highest_price, and lowest_price) were updated/corrected to produce the final cleaned data using MySQL. The resulting tables in SQL were then exported to a CSV file to produce the final cleaned files.

Each of these steps was either critical to our U1 case [of providing the lowest_price subset of Dish options to a target user] or was needed to clean data that we deemed of possible value for ancillary/secondary features and/or future features of our application. See the following summary (beginning on the next page) for details.

[1] Note, the use of the term "phases" here is not to be confused with Phase I and Phase II of the overall project. We will throughout this document refer to "phase I" and "phase II" of the data cleaning operation itself. That is, we broke Phase II of the overall project into its own two distinct phases.

A. Menu

i. <u>Removed Columns (Phase I - Open Refine):</u>
   a. This was done in OpenRefine using the “Remove Column” operation. Some of the empty columns do not have a direct impact on the U1.

ii. <u>Trimmed Consecutive space and trimmed tailing spaces (Phase I - Open Refine):</u>
   a. This cleaning step involved removing unnecessary spaces, both consecutive and at the end of text entries, to standardize data format and improve consistency.

iii. <u>Removed the unknown characters and clustered the like names together (Phase I - Open Refine):</u>
   a. This step involved identifying and removing unrecognized characters from text entries, and then grouping similar names together to standardize entries and eliminate variations. This Operation directly supports the U1 to find the lowest cost subset of Dish offerings by providing accurate and readable associated Menu information.

iv. <u>Converted Columns to Number (Phase I - Open Refine)</u>:
   a. This was done in OpenRefine using the “Text transform on cells” operation with the ‘value.ToNumber()“ expression.

B. MenuPage

i. <u>Removed Column - uuid (Phase I - Open Refine):</u>
   a. This was done in OpenRefine using the “Remove Column” operation. The uuid column is not relevant to our U1 or any other analytics that we intend to perform. Removing unnecessary columns provides a cleaner table targeted towards our U1. This is not required for U1 but uuid is removed to simplify the dataset.

ii. <u>Converted Columns to Number - id, menu_id, page_number, image_id, full_height, full_width (Phase I - Open Refine):</u>
   a. This was done in OpenRefine using the “Text transform on cells” operation with the ‘value.ToNumber()“ expression. All necessary columns were converted to numbers to ensure proper use and calculations in the next step. This step is required for U1 as improper types can have various unintended effects like wrong foreign keys, and referential integrity.

iii. <u>Deleted MenuPage Instances with Invalid Foreign Key Reference to the Menu Table (Phase II - MySQL):</u>
   a. If a MenuPage does not have a corresponding Menu, we are assuming any products listed thereon are not actually available and so we do not want to present MenuItem/Dish results to our users from these “orphaned MenuPages”.

C. MenuItem

★ Split MenuItem.csv into six separate files manually (i.e. using text editor), with files 1-5 containing 250,000 rows each and the 6th file containing the remaining 82,726 rows.

   a. This step was completed to ensure full functionality with OpenRefine software. The resulting 'cleaned' data from OpenRefine output is merged back together in a later step. (See the matching '★' bullet point item below.)

ii. <u>Removed column 'high_price'. (Phase I - Open Refine):</u>

   a. Not needed for U1.

iii. <u>Used 'Customized Facet - Facet by blank (null or empty string)' to exclude "true" matches on price attribute. (I.e., to exclude/filter out all instances with missing price data.) (Phase I - Open Refine):</u>

   a. MenuItem entries without price data cannot support U1 of providing a lowest_price subset.

iv. <u>Converted price column to numeric value via 'Common transforms'. (Phase I - Open Refine)</u>

v. <u>Utilized a 'Custom numeric facet' to convert all entries with price = 0 to non-numeric representation (i.e. the string "Zero"), then excluded the resulting non-numeric values, to remove all entries with price = '0'. (Phase I - Open Refine):</u>

   a. Used GREL language here, syntax as follows:

      i. if(value==0,"zero",value)

   b. We are assuming a price value of '0' is invalid and therefore this removal is necessary to support U1.

vi. <u>Removed "UTC" string from all timestamps. (Phase I - Open Refine):</u>

   a. This was necessary for the MySQL import (Phase II).

vii. <u>Removed instances with blank/empty string for 'dish_id' attribute. (Phase I - Open Refine):</u>

   a. MenuItem instances with no dish_id cannot be identified and therefore are of no use for U1.

★ Used Python script 'CombineCleanedMenuItemOpenRefineOutputs.py' to re-merge split and now partially cleaned MenuItem csv files after Phase I cleaning of MenuItem.

viii. <u>Removed instances without a valid foreign key reference to the MenuPage table (Phase II - MySQL):</u>

   a. MenuItems without a valid reference to the MenuPage table (i.e. MenuItem instances without a MenuPage_id present in the MenuPage table) are assumed to be unavailable, and therefore are of no use for U1.

ix. <u>Removed instances without a valid foreign key reference to the Dish table (Phase II - MySQL):</u>

a. MenuItems without a valid reference to the Dish table (i.e. MenuItem instances without a Dish_id present in the Dish table) cannot be identified, and therefore are of no use for U1.

D. Dish

i. <u>Standardize Text Formatting - name (Phase I - Open Refine):</u>
   a. This was done in OpenRefine by applying the “Text transform on cells” operation with “value.trim()” and “value.replace(/[\\p{Zs}\\s]+/,' ')” expression. Trimmed leading and trailing whitespaces, and normalized spaces in between characters. This ensures consistency for the dish name, preventing duplicate dish names. This step is required for our U1 as consistent names are critical for proper searches and matching for dishes.

ii. <u>Convert Columns to Numbers - menus_appeared, times_appeared, first_appeared, last_appeared, lowest_price, highest_price (Phase I - Open Refine):</u>
   a. This was done in OpenRefine using the “Text transform on cells” operation with the ‘value.ToNumber()“ expression. All necessary columns were converted to numbers to ensure proper use and calculations in the next step. This step is required for U1 as improper types can have various unintended effects like wrong foreign keys, and referential integrity.

iii. <u>Remove Column - description (Phase I - Open Refine)</u>
   a. This was done in OpenRefine using the “Remove Column” operation. The description column is not relevant to our U1 or any other analytics that we intend to perform. Removing unnecessary columns provides a cleaner table targeted towards our U1. This is not required for U1 but description is removed to simplify the dataset.

iv. <u>Missing Values - lowest_price, highest_price (Phase I - Open Refine)</u>
   a. This was done in OpenRefine by applying the “Text transform on cells” operation with the “grel:if(isBlank(value),null,value)” expression. If the values in the lowest_price, or highest_price are blank then the values are set to null. This ensures that missing prices are clearly differentiated from a price of zero. This step is not required for our U1 but it ensures our overall data accuracy and clarity of the presented values.

v. <u>Perform Internal Consistency Checks by creating Temporary Columns - negative_appeared, Negative times appeared, negative price (Phase I - Open Refine)</u>
   a. These columns were created in OpenRefine using the “Column Addition” operation, then tested with “Text transform on cells”, and then deleted after

the checks were completed. Temporary columns were created to perform logical checks. (example: times_appeared is not less than menus_appeared, and highest_price is not less than lowest_price, and last_appeared is not before first appeared). These temporary columns were then removed. This process is for data validation allowing logical data. This step is required for our U1 as it ensures our data is logically accurate.

vi. <u>Validated and Updated (where applicable) lowest_price field (Phase II - MySQL):</u>
   a. Cross-referenced the MenuItem table to determine the lowest price for each dish instance, updating the field where necessary. This is in direct support of U1.

vii. <u>Validated and Updated (where applicable) highest_price field (Phase II - MySQL):</u>
   a. Cross-referenced the MenuItem table to determine the highest price for each dish instance, updating the field where necessary. This is not directly needed for U1 but might be of interest to users as a secondary data point or for a future feature.

viii. <u>Validated and Updated (where applicable) menus_appeared column (Phase II - MySQL):</u>
   a. Cross-referenced the MenuItem, MenuPage, and Menu tables to determine the menus_appeared for each dish instance, updating the field where necessary. This is not directly needed for U1 but might be of interest to users as a secondary data point or for a future feature.

ix. <u>Validated and Updated (where applicable) times_appeared column (Phase II - MySQL):</u>
   a. Cross-referenced the MenuItem table to determine the times_appeared for each dish instance, updating the field where necessary. This is not directly needed for U1 but might be of interest to users as a secondary data point or for a future feature.

2. Data Quality Changes

DATA CLEANING PHASE I - OpenRefine

Menu	
Field	**Changes (Made With OpenRefine)**
ID	● Numeric and no blank values assigned to ID
name	****** Removed the Unnecessary Field ***** ○ **Rationale:** Initial data analysis revealed a significant anomaly: 14,348 instances of an "empty field" within the dataset.
Sponsor	● Consecutive spaces were trimmed and removed. ● Symbols were removed from the beginning and end of names. ● Same sponsors were clustered and merged.
Event	● Consecutive spaces were trimmed and removed. ● Symbols were removed from the beginning and end of names. ● Same Events were clustered and merged.
Venue	● Trimmed and removed the consecutive spaces ● Removed the following symbols from beginning and end of the name ● Cluster and merged the same Venues
Place	● Trimmed and removed the consecutive spaces ● Removed the following symbols from beginning and end of the name ● Cluster and merged the same Venues
Physical_description	● Trimmed and removed the consecutive spaces ● Removed the following symbols from beginning and end of the name ● Cluster and merged the same Physical description
Occasion	● Trimmed and removed the consecutive spaces

	● Removed the following symbols from beginning and end of the name ● Cluster and merged the same occasion
Notes	● Trimmed and removed the consecutive spaces ● Removed the following symbols from beginning and end of the name ● Cluster and merged the same Notes
Call_number	● Trimmed and removed the consecutive spaces
Keywords	****** Removed the Blank column***** ○ **Rationale:** The column was entirely empty, resulting in a total of 17,545 blank values.
Language	****** Removed the Blank column***** ○ **Rationale:** The column was entirely empty, resulting in a total of 17,545 blank values.
Date	● Trimmed and removed the consecutive spaces ● Converted to date
Location	● Trimmed and removed the consecutive spaces ● Removed the following symbols from beginning and end of the name ● Cluster and merged the same Location
Location_type	****** Removed the Blank column ***** ○ **Rationale:** The column was entirely empty, resulting in a total of 17,545 blank values.
Currency	● Trimmed and removed the consecutive spaces
Currency_symbol	● Trimmed and removed the consecutive spaces
Status	● Trimmed and removed the consecutive spaces ● Cluster and merged the same Location
Page_count	● Converted to Number
Dish_count	● Converted to Number

Menu Data Removal Summary:

Total Records Removed = 2,184:

i. 2,184 records with blank values identified and removed.
ii. 4 records with blank/empty columns identified and removed.

Total Attributes Removed = 4:

i. Name - Unnecessary, high anomaly count with ~82% of records having a blank/empty string entry.
ii. Keywords - All instances with empty/blank string entry.
iii. Language - All instances with empty/blank string entry.
iv. Location_type - All instances with empty/blank string entry.

Menu Page	
Fields	**Changes (Made With OpenRefine)**
MenuPageId	● Numeric and no blank values assigned to ID
menu_id	● Converted to Number
page_number	● Converted to Number ○ **Rationale:** During data cleaning, 1202 missing values were identified.
image_id	● Converted to Number
Dimensions Full_height /Full_width	● Converted to Number ○ **Note:** We identified 329 missing values for both Full_height and Full_width during the data cleaning process.
Uuid	****** Removed the Field ***** ○ **Rationale:** The column is not directly used for the U1 case.

MenuPage Data Removal Summary:

Total Records Removed = 1,860:

i. 1,860 records with blank values identified and removed.

Total Attributes Removed = 1:

i. Uuid - Unrelated to U1, no other possible uses identified.

Menu Item	
Fields	**Changes (Made With OpenRefine)**
MenuItemId	None ● Converted to 'INT' datatype for data validation via MySQL import, see next section.

Menu_page_id	None ● Converted to 'INT' datatype for data validation via MySQL import, see next section.
Price	● Converted to Number ● Filtered out blank and '0' values, considering these invalid data entries. ○ **Rationale:** This field is directly used for U1, in which uses identify the lowest cost menu items of a certain type. Unreliable/Inaccurate price information will negatively impact the user experience.
High_price	****** Removed the Field ***** ○ **Rationale:** Not needed for U1, and can be inferred by the application by searching the MenuItem table. Requires more complicated configuration (e.g. update trigger) to maintain accurate stored value. Assuming application search frequency for high price is infrequent enough to warrant removal of stored value.
Dish_id	● Filtered out blank and '0' values, considering these invalid data entries. ○ **Rationale:** This field is used directly for U1, as the application's users will be searching across Menus (and therefore Menu Items) for a target Dish's lowest cost options. If a MenuItem has no associated Dish, it cannot be identified, and so cannot be linked back to any 'real world' product for consumption.
Created_at	● Trimmed Trailing " UTC" string. ○ **Rationale:** Necessary for importing into a 'DATETIME' field of MySQL for further data validation. (This column is not needed for U1 directly , but could be used for an application's other features, or if a user would want to know dates associated with MenuItems.)
Updated_at	● Trimmed Trailing " UTC" string. ○ **Rationale:** Necessary for importing into a 'DATETIME' field of MySQL for further data validation. (This column is not needed for U1 directly , but could be used for an application's other features, or if a user would want to know dates associated with MenuItems.)
Xpos	None
YPos	None

MenuItem Data Removal Summary[2]:

a. MenuItem_1.csv → MenuItem_1-CleanedwOpenRefine.csv
    i. 131,457 records with blank/null prices identified and removed.
    ii. 51 records with prices of ‘0’ identified and removed.
b. MenuItem_2.csv → MenuItem_2-CleanedwOpenRefine.csv
    i. 107,944 records with blank/null prices identified and removed.
    ii. 185 records with prices of ‘0’ identified and removed.
    iii. 2 records with blank dish_ids identified and removed.
c. MenuItem_3.csv → MenuItem_3-CleanedwOpenRefine.csv
    i. 96,536 records with blank/null prices identified and removed.
    ii. 52 records with prices of ‘0’ identified and removed.
    iii. 1 record with a blank dish_id identified and removed.
d. MenuItem_4.csv → MenuItem_4-CleanedwOpenRefine.csv
    i. 66,894 records with blank/null prices identified and removed.
    ii. 28 records with prices of ‘0’ identified and removed.
e. MenuItem_5.csv → MenuItem_5-CleanedwOpenRefine.csv
    i. 26,211 records with blank/null prices identified and removed.
    ii. 31 records with prices of ‘0’ identified and removed.
f. MenuItem_6.csv → MenuItem_6-CleanedwOpenRefine.csv
    i. 16,874 records with blank/null prices identified and removed.
    ii. 20 records with prices of ‘0’ identified and removed.

Total Records Removed = 445,286:
  i. 445,916 records with blank/null prices identified and removed.
  ii. 367 records with prices of ‘0’ identified and removed.
  iii. 3 records with blank/empty dish_ids identified and removed.

Total Attributes Removed = 1:
  i. High_price - Can be inferred from other attributes. Do not need to store this value for U1.

Dish	
Fields	**Changes (Made With OpenRefine)**
DishId	• Numeric and no blank values assigned to ID
Name	• Consecutive spaces were trimmed and removed. • Symbols were removed from the beginning and end of names. • Same names were clustered and merged.

[2] Due to OpenRefine input data size limitations, MenuItem was split into six different identical operations/projects, as described in this data removal summary.

Dish Data Removal Summary:
Total Records Removed = 0:
  i. No rows missing critical data identified.
Total Attributes Removed = 1:
  i. Description - All instances with empty/blank string entry.

DATA CLEANING PHASE II - MySQL

Menu	
IC Violation Identified	**Action Taken and Rationale (via MySQL)**
Identified 12,887 instances of MenuPage records with invalid Menu_ids. (I.e. MenuPages with Menu_ids that did not have a corresponding id in the Menu table.) This is a foreign key violation.	● These 12,887 records were deleted from the MenuPage table. If a MenuPage does not have a corresponding Menu, we are assuming any products listed thereon are not actually available and so we do not want to present MenuItem results to our users from these "orphaned MenuPages".
Identified 140,649 instances of MenuItems with invalid Menu_Page_ids. (I.e.	● These 140,649 records were deleted from

MenuItems with Menu_Page_ids that did not have a corresponding id in the MenuPage table.) Like the previous violation identified, this is a foreign key violation.	the MenuItem table. In similar vein to the previous violation identified, we are assuming that a MenuItem that does not have a valid reference to a MenuPage is not actually available, and so we do not want to present any such MenuItem as a possible candidate to our users to fulfill the target U1 function, as this could result in inaccurate feedback.
Identified 2 instances of MenuItems with invalid Dish_ids. (I.e. MenuItems with Dish_ids that did not have a corresponding id in the Dish table.) This is also a foreign key violation.	● These 2 records were deleted from the MenuItem table. Like the previous 2 violations discussed, we make the assumption that a MenuItem without an actual corresponding dish is not available. Furthermore, in this case if there is no associated dish, we do not know what the given MenuItem represents. (I.e. there is no Dish ‘name’ to reference, to know what product is being offered.) Essentially, without a Dish_id a MenuItem is of no use to our application in light of U1.
Identified 87,540 invalid lowest_price values in the Dish table.	● Using a MIN group by aggregation query, we were able to both identify and update/correct the Dish’s lowest_price values where applicable, in reference to each dish’s corresponding MenuItems. Similarly, we were also able to validate correct values in a like manner. *This field is directly related to U1, which again is providing the lowest_price option (or set of options, which will always include the single lowest_price option) to the user for a target dish. As such, it is crucial this data is up-to-date and accurate.* ○ Keeping this stored value field up to date

	in the running application would require a triggered calculation/update whenever a new MenuItem is introduced or an existing MenuItem updated. We left it in as such, however, because it may be a frequently requested item so we do not want to run the calculation for each request.
Identified 105,141 invalid highest_price values in the Dish table.	● Using a MAX group by aggregation query, we were able to both identify and update/correct the Dish's highest_price values where applicable, in reference to each dish's corresponding MenuItems. Similarly, we were also able to validate correct values in a like manner. ○ While not directly related to U1, we decided to preserve this stored value in our database because it might be a data point that is still of interest to our users. As such, it must remain accurate and up-to-date for the best user experience.
Identified 37,079 invalid menus_appeared values in the Dish table.	● Using a COUNT group by aggregation query, we were able to both identify and update/correct the Dish's menus_appeared values where applicable, in reference to each dish's corresponding Menus (via the MenuItem → MenuPage → Menu relation). Similarly, we were also able to validate correct values in a like manner. ○ While not directly related to U1, we decided to preserve this stored value in our database because it might be a data point that is still of interest to our users. As such, it must remain accurate and up-to-date for the best user experience.
Identified 35,022 invalid times_appeared values in the Dish table.	● Using a COUNT group by aggregation

	query, we were able to both identify and update/correct the Dish's times_appeared values where applicable, in reference to each dish's corresponding MenuItems. Similarly, we were also able to validate correct values in a like manner. ○ While not directly related to U1, we decided to preserve this stored value in our database because it might be a data point that is still of interest to our users. As such, it must remain accurate and up-to-date for the best user experience.

DATA QUALITY IMPROVEMENT SUMMARY

Menu (Phase I - After OpenRefine, Before SQL - Checked with Datalog [Clingo]):

Integrity Constraint	Original Violations	Cleaned Violations
Duplicate ID: This constraint ensures that each record has a unique identifier, preventing redundant or incorrect data entries.	0	0
Invalid/Null date: This constraint identifies and addresses records where the date field is either missing or contains an unreadable format, ensuring all date entries are valid and complete.	591	0
Invalid status: This constraint checks for and corrects any entries in the status field that do not conform to predefined valid status options.	0	0
Null sponsor/event/venue/place: This constraint targets records where the sponsor/event/venue/place field is empty, indicating a missing or incomplete data point that needs to be addressed.	1,561	0
Total Violations	2,184	0

Summary:

"Menu" dataset, comparing original and cleaned integrity constraint violations. All integrity constraints—including "Duplicate ID," "Invalid status," "Invalid/Null date," and "Null sponsor/event/venue/place"—showed zero violations after cleaning, down from a combined total of 2184 original violations. This indicates that all identified data quality issues in the "Menu" table were successfully addressed through the cleaning process. (This report was generated using datalog/Clingo. See supplemental file 'Menu.lp'.)

Menu Page (Phase I - After OpenRefine, Before SQL - Checked with Datalog [Clingo]):

Integrity Constraint	Original Violations	Cleaned Violations
Duplicate IDs: This constraint ensures that each record has a unique identifier, preventing redundant or incorrect data entries.	0	0
Null menu_id: This constraint specifically addresses records where the menu ID field is empty, ensuring that all menu items are properly linked to a menu.	0	0
Non-positive page_number: This constraint identifies records where the page number is zero or negative, ensuring that all page numbers are valid and positive. **Note:** It is currently configured to zero. Since page_number can not be non-positive all negative page numbers are set to 0	1,202	1202
Null image_id: This constraint addresses records where the image ID field is empty, ensuring that all images are properly linked to their respective entries.	0	0
Invalid height: This constraint identifies records with height values that are invalid, ensuring all height entries are accurate.	329	329
Invalid width: This constraint identifies records with width values that are invalid, ensuring all width entries are accurate.	329	329
Total Violations	1,860	1860

Summary:
The "Menu Page" table shows that while some data integrity issues like duplicate IDs or null menu/image IDs had no violations before or after cleaning, persistent problems with non-positive page numbers and invalid height/width values led to a total of 1860 unaddressed violations. This indicates that despite cleaning efforts, specific issues related to page numbering and image dimensions remain uncorrected. (This report was generated using datalog/Clingo. See supplemental file 'MenuPage.lp'.)

Menu Item (Phase I - After OpenRefine, Before SQL - Checked with Datalog [Clingo]):

Integrity Constraint	Original Violations	Cleaned Violations
Invalid/Null Price: This constraint identifies and addresses records where the price field is either missing or contains an unreadable format, ensuring all price entries are valid and complete.	445,916	0
Zero Price: This constraint identifies and addresses records where the price field is zero, ensuring all price entries are valid and complete.	367	0
Invalid/Null Dish ID: This constraint identifies and addresses records where the dish id field is either missing or contains an unreadable format, ensuring all dish id entries are valid and complete.	3	0
Total Violations	446,286	0

Summary:
"MenuItem" dataset, comparing original and cleaned integrity constraint violations. All integrity constraints—including "invalid_price," "zero_price," and "invalid_dishid"—showed zero violations after cleaning, down from a combined total of 446286 original violations. This indicates that all identified data quality issues in the "Menu" table were successfully addressed through the cleaning process. (This report was generated using datalog/Clingo. See supplemental file 'MenuItem.lp'.)

Dish (Phase I - After OpenRefine, Before SQL - Checked with Datalog [Clingo]):

Integrity Constraint	Original Violations	Cleaned Violations
Duplicate IDs: This constraint ensures that each record has a unique identifier, preventing redundant or incorrect data entries.	0	0
Null name: This constraint identifies records where the 'name' value is a negative number, ensuring that all 'name' entries are positive	0	0
Times_appeared < Menus_appeared: This constraint checks for instances where the 'times appeared' value is logically inconsistent by being less than the 'menus appeared' value. **Note:** This value is set to 0 because it has the essential Dish item.	8,274	0
Last_appeared < First_appeared: This constraint identifies instances where the 'last appeared' date is logically inconsistent by being earlier than the 'first appeared' date. **Note:** This value is not removed because it has the essential Dish item.	6	6
Negative menus_appeared: This constraint identifies records where the 'menus appeared' value is a negative number, ensuring that all 'menus appeared' entries are positive.	0	0
Negative lowest_price: This constraint identifies records where the 'lowest price' value is a negative number, ensuring all 'lowest price' entries are positive.	0	0
Negative highest_price: This constraint identifies records where the 'highest price' value is a negative number, ensuring all 'highest price' entries are positive.	0	0
Total Violations	8,280	6

Summary:
The "Dish" table's data quality significantly improved after cleaning, with total violations decreasing from 8280 to 6, primarily by addressing inconsistencies in appearance dates and price ranges, as well as ensuring logical relationships between 'times appeared' and

'menus appeared'. (This report was generated using datalog/Clingo. See supplemental file 'Dish.lp'.)

PHASE II Summary - MySQL:
In this final step, we imported the partially cleaned data from phase I (in csv format[3]) into tables in a MySQL based relational database, to perform the following referential checks and updates in order to produce our final, cleaned dataset. See 'NYCMenuDataVerification.sql' and 'MySQL-Phase2Cleaning.log' for the script and resulting output log for a complete, step-by-step analysis of steps taken and results achieved. Below is a summary:

Phase II - MenuPage, MenuItem, Dish - MySQL:

Integrity Constraint	Original Violations	Cleaned Violations
Foreign Key Reference Violation - MenuPage → Menu: This constraint ensures that each MenuPage has a valid reference to the Menu table.	12,877	0
Foreign Key Reference Violation - MenuItem → MenuPage: This constraint ensures that each MenuItem has a valid reference to the MenuPage table.	140,649	0
Foreign Key Reference Violation - MenuItem → Dish: This constraint ensures that each MenuItem has a valid reference to the Dish table.	2	0
Inaccurate Stored Value: Dish.Lowest_Price This constraint ensures that data is consistent across tables and stored values reflect accurately.	87,540	0
Inaccurate Stored Value: Dish.Highest_Price This constraint ensures that data is consistent across tables and stored values reflect accurately.	105,141	0
Inaccurate Stored Value: Dish.Menus_Appeared This constraint ensures that data is consistent across tables and stored values reflect accurately.	37,079	0

[3] Note, for our particular test environment, there were two slight modifications required to import the datasets into MySQL which we do not consider actual elements of data cleaning. First was the need to escape backslash characters in strings, by adding an additional backslash character to each occurrence thereof. The second was the need to create 'placeholder' columns as the last column [in the csv files] where custom logic was used to convert empty strings directly to NULL values [for INT or DECIMAL fields] upon import, if the last column was targeted for an INT or DECIMAL field before the addition of said placeholder. (See comments in 'NYCMenuDataCleaningAndValidation.sql' for more details.) In these instances, the final data in the database did not have duplicated backslash characters and/or the 'placeholder' column was dropped after import, such that the starting table data still matched the original (partially cleaned) input data exactly. Nevertheless, this is worth mentioning if attempting to reproduce these results on certain systems.

Inaccurate Stored Value: Dish.Times_Appeared This constraint ensures that data is consistent across tables and stored values reflect accurately.	35,022	0
Total Violations	418,310	0

OVERALL DATA QUALITY IMPROVEMENT SUMMARY

TABLE	PHASE I ICs REMOVED	PHASE II ICs REMOVED	TOTAL ICs REMOVED
Menu	**2,184**	**0**	**2,184**
MenuPage	**0**	**12,877**	**12,877**
MenuItem	**446,286**	**140,649**	**586,935**
Dish	**8,274**	**264,782**	**273,056**
TOTALS	**456,744**	**418,308**	**875,052**

3. Workflow Model

As described in previous sections, our workflow consisted of two main phases. In ‘phase I’ we used OpenRefine to initially clean the data, to address any intra-table (i.e. single-table) issues and/or integrity constraints. We then used datalog (via Clingo) to validate our IC checks, comparing our dirty data IC counts to our cleaned data IC counts. In “phase II” of our data cleaning operation, we used MySQL to address inter-table issues and/or integrity constraints (i.e. issues or constraints involving two or more tables). We felt that OpenRefine provided a simple yet powerful interface to deal with the initial phase 1 cleaning, addressing all our phase 1 needs while also affording a readily available way to export our data cleaning “recipes” and, in conjunction with OR2YW, generate “inner” workflow diagrams easily. Additionally, MySQL lent itself well to the final cleaning stage in which data was cross-referenced between two or more tables and stored values were validated and/or updated as needed. All rationales for these decisions are described in detail in previous sections. We used LucidChart to manually generate our final “outer” workflow diagram.

Please see the following files for our inner and outer workflow diagrams (continued on the next page):

- **Menu_Clean.pdf**
  - Inner Workflow Diagram for Menu Cleaning Phase I
  - Generated with OR2YW
- **MenuPage_Clean.pdf**
  - Inner Workflow Diagram for MenuPage Cleaning Phase I
  - Generated with OR2YW
- **MenuItem_Clean.pdf**
  - Inner Workflow Diagram for MenuItem Cleaning Phase I
  - Generated with OR2YW
- **Dish_clean.pdf**
  - Inner Workflow Diagram for Dish Cleaning Phase I
  - Generated with OR2YW
- **Workflow.pdf**
  - Outer Workflow Diagram - Phase I and Phase II Cleaning
  - Diagram “Source File” - See “Workflow.vsdx” (Vizio File)
    - Created using LucidChart.com

4. Conclusion & Summary

A. In this phase of the project, we tackled the disorderly job of cleaning up the Menu, MenuPage, MenuItem, and Dish datasets. Our goal was to identify and correct inconsistencies and quality issues to make sure the data was reliable and ready for analysis on the U1 case and (potentially) other related use cases as well. We approached this process with a two-part strategy, blending hands-on data wrangling tools with smart, logic-driven programming. We used OpenRefine to handle the usual headaches that come with real-world data—like getting rid of extra spaces that can mess up searches or cause errors, and trimming off trailing spaces that make string matching a nightmare. OpenRefine also helped us catch and fix invalid characters, which is important for keeping data attributes consistent and avoiding integrity violations.

We also performed datalog integrity checks to enforce stricter rules and catch harder problems that might be missed just by looking at the data. By setting up clear rules about how the data should be formatted and related (in the case of foreign keys), Datalog helped us flag columns and rows like menu items with no dish references or menu pages with no menu references. Combining OpenRefine’s flexibility with Datalog’s logical precision provided us a powerful way to thoroughly clean up all four interconnected datasets, setting a solid foundation for all use cases during phase I of our cleaning protocol. Finally, for the second part of our cleaning strategy, we used MySQL to eliminate referential constraint violations[4], update/correct stored values, and validate the remaining data in its final form.

This project has provided a comprehensive overview of a complete data cleaning process, including both the planning and implementation phases. Through this process, the usage of various tools and techniques has greatly assisted with the completion of learning objectives. For instance, the use of OpenRefine, along with its related OR2YWTool visualization package, demonstrated a powerful, straightforward way to both clean data and track the provenance of newly created (i.e. “cleaned”) files along the way. OpenRefine’s “recipe” exports are a very convenient and powerful way to account for each step of the data cleaning process, and the OR2YWTool is a great way to visualize the workflow with relatively minimal input required from the user. Similarly, the steps taken in MySQL to work with relations between related datasets helped to demonstrate firsthand the power of the relational model and modern relational database platforms to bring into focus workable, usable data that is fit for a production application.

Some challenges were encountered throughout the process. For example, we did not initially foresee data size limitations with the OpenRefine platform, which required us to rework our approach a bit when handling the MenuItem data (i.e. splitting the original dataset as input and then re-merging the resultant cleaned output). There were also several minor issues encountered when loading the (partially) cleaned into the database platform (MySQL). One of these was described in a footnote above, involving the escaping of backslash characters. We also encountered an issue importing empty strings into numeric columns (e.g. type = INT or DECIMAL), and needed to use some custom logic to convert these to NULL during the import process. (See the ‘NYCMenuCleaningAndVerification.sql’ file for additional details.) Alternatively, we could have worked some additional logic in OpenRefine to deal with this requirement, but we did not foresee the issue until we began the “Phase II” cleaning process. Details like these provide good learning opportunities and we enjoyed overcoming the challenges we encountered.

In large part, we stuck to our initial “Phase I” plan in completing the data cleaning implementation described in this report. (And this is not to be confused with the “Phase I” and “Phase II” of the cleaning itself as described above, but rather Phase I

[4] While OpenRefine allowed us to filter out rows missing foreign key references altogether in phase I of our cleaning, MySQL provided the means to validate the remaining rows which did have foreign key references, and filter further to remove those that were invalid (i.e. not present in their respective/referenced table).

of the overall project in which we planned for the cleaning.) We did make some minor changes from our original Phase I plan, such as removing the “high_price” column of MenuItem, which we had not planned to do initially. One of the more major deviations from the Phase I plan was in the use of the specific SQL database platform for Phase II. We initially planned to use SQLite, but ended up using MySQL. (This was mainly due to the fact that we already had the MySQL infrastructure in place.) Other minor items, such as the need to remove the trailing “ UTC” string of MenuItem datetime values, necessary for the database imports, were also not foreseen in the planning phase (i.e. Phase I). However, in large part our planning efforts in Phase I did align with our execution efforts in Phase II.

Lastly, the next hypothetical steps for the implementation of our application would include building the web hosting infrastructure along with pertinent SQL queries to deliver the primary use case results, along with any metadata and/or related query results requested by our user base. I.e., the remaining steps would include those necessary to deliver a search platform on which users could attempt to find a subset of lowest price offerings for a given dish in the greater NYC area, based on the NYC Menu data provided by the New York Public Library for this project. Protocols would need to be implemented to ensure data remained up-to-date and accurate for the user base, and as such, the data cleaning process would likely prove to be a continuous one. We believe the completion of this project has aided in providing insight into the important field of data cleaning, both the planning and implementation aspects thereof, as well as the capturing and documenting of related workflow processes. We hope to use the skills gained here in many more projects to come!

B. Team Members Contributions:

Member	Task
Samarth Patel	The cleaning of the Menu, MenuPage, and Dish datasets was conducted, alongside the identification of integrity constraint violations in both the original and refined datasets.
Kesavar Kabilar	Generated ‘outer’ workflow diagram using LucidChart. Completed datalog integrity constraint violation checks for MenuItem, on both the original and refined datasets.
Benjamin Fridkis	Splitting and cleaning MenuItem data in phase I with OpenRefine, and generating YesWorkflow diagrams (parallel and linear versions) for this process using or2ywtool. Completed all SQL data loading, cleaning, and verifications, along with associated scripts and output log (“phase 2”

Member	Task
	cleaning). Finalized and submitted report and associated files/documentation.
All Team Members	Written report completion.