CS513 Summer 2024 Project Phase II

# Team – ID:

* John Margineanu, [johnm15@illinois.edu](mailto:johnm15@illinois.edu)
* Joel Feddes, [feddes2@illinois.edu](mailto:feddes2@illinois.edu)
* Anthony Piroli, [pirolli2@illinois.edu](mailto:pirolli2@illinois.edu)

# Description of Data Cleaning Performed.

## ‘Menu Item’ table:

For this table we opted to use Python Jupyter Notebook due to the size restrictions in OpenRefine.

1. High-Level cleaning steps:
   1. Investigate data and understand the nature of each column. In this case with df.info().
   2. Address Null values by converting null values to zero.
   3. Address Dytpes.
   4. ‘id’ is unique. Create a search for any violations. Luckily there are none in this table.
2. Rationale:
   1. Investigation. This reveals the number of records, number of non-null values, as well as Dtype. Investigation and understanding of data is necessary regardless of its use for use case.
   2. Null Values. Null values can be problematic when analyzing data for example such large complex data set may produce incorrect results if it is queries with SQL since SQL does not pick up Null values
   3. Dytpes. The table contains dates which were coded as objects that need to be converted to datetime so that they can be read as such by users.
   4. Logical consistency. Ensure the record identifier is unique else it will not be possible to analyze information correctly.
   5. Relation to the use case. Our use cases focus on understanding the number of menus of a particular dimension. While the data in this table is not directly related to that use case it may be used for following up cases. For example, given a dimension what positions is a particular item found. The clean up for this table is not expensive and therefore should be performed.

## ‘MenuPage’ table:

1. High-Level Cleaning steps. The provided OpenRefine history JSON describes a series of data cleaning operations. In the uuid column, variations in case and format were standardized to "510d47db-491f-a3d9-e040-e00a18064a99\r". The full\_height column had its blank cells filled with "0", which were then transformed to "NULL". Similarly, blank cells in the full\_width column were replaced with "NULL". Additionally, blank cells in the page\_number column were filled with "N/A". Moving forward, there was a script that was created to look for where image\_id, menu\_id, and page\_number were the same and occurred more than once as these are IC violations containing the same Foreign IDs with the same primary IDs. The subsequent occurrences were removed. Finally, there was script created to check if all the MenuId’s within the MenuPage table were valid IDs in the Menu table. These operations aimed to ensure data consistency by standardizing values and filling in blanks with appropriate placeholders.
2. Rationale:
   1. UUID Standardization. There were a few values that were not properly cased to the rest of the column. Standardizing this column helped to ensure data consistency. The rationale behind this was to standardize the column so when we look for a certain identifier, we are certain that every UUID can be queried with the same format.
   2. Full Height and Full Width Entry. There were some empty values in this column that I thought would make more sense to be filled in with NULL. The data was simply not entered into the table and as an end user using the data, it would make more sense to have these values be filled with NULL rather than nothing. This step did not support the use case, but we think this is a necessary step to answer and doubt that could have arisen initially.
   3. Page Number to N/A. There were also some values that were empty within this column. I looked at a few Menu’s where the page was linked by a foreign key and it turned out that the Menu was only 1 page long. Due to this, I filled in empty values to N/A to showcase that the page itself is the menu and not a subset of pages.
   4. Removal of duplicate entries. It looks as if whoever was inserting the data, did it multiple times for the same MenuPages. Removing duplicate entries ensures that users can locate the exact information regarding the Menu along with the image that is associated with that page. There should not be multiple MenuPage entries for the same object referencing the same page, menu, and image.

## ‘Dish’ table:

1. High-level cleaning steps:
   1. Remove double spaces and trailing white space
   2. Clustering similar names/dishes.
   3. Transformation of fields Dtype
   4. Handle illogical data.
2. Rationale:
   1. Cleaning extraneous data from cell values allows for more precise querying and analysis. Elimination of whitespace and double spaces will prevent dishes appearing as two separate dishes.
   2. Clustering will allow standardization by removing unnecessary distinctions. For example, “Honey Toast” vs. “Toast with Honey”.
   3. Fields that appeared numeric were made numeric to allow for aggregations and other numerical analysis.
   4. The ‘times\_appeared’ values were illogical, i.e. negative values and therefore converted to nulls as this is not a valid time.

## ‘Menu’ table:

1. High-level cleaning steps:
   1. Remove double and white space.
   2. Clustering similar items. A similar process was performed for “sponsor”, “event”, “venue”, “place”, “physical\_description”, “location”, “currency” and “occasion”.
   3. Parsing Physical Description column to extract dimensions into a dimension field using Pyhon.
   4. Invalid dates set to null.
   5. Convert numerical fields from text to numeric.
   6. Removed blank columns.
   7. Ensure ‘id’ is unique.
2. Rationale:
   1. Cleaning extraneous data from cell values allows for more precise querying and analysis.
   2. Clustering as mentioned above will further improve the searchability and subsequent analysis of the data. This point is crucial for our use case because we are required to count menus of specific dimensions. Not having clean data here will result in wrong counts.
   3. This parsing is crucial for use case else we would not be able to perform the query.
   4. Invalid dates will make add on analysis that can result from use case invalid.
   5. Converting to appropriate type will allow full use of query functionality by having the system correctly read the data.
   6. Removal of blank columns produced lighter data set making computations less expensive.
   7. Our use case requires us to count menus of dimension and duplicate records would be problematic. In this case there are no violations.

# Document Data Quality Changes.

## Quantification:



## Demonstrate Quality improvement:

From a use case perspective, the most relevant improvement is related to the Menu table and specifically the physical description column and parsing into the dimensions column.

As you can see below the physical description column is not in any shape to perform query on dimensions. Providing a dimension column in text format can be queried and records counted.



# Conclusions and Summary.

In summary the team reviewed four interconnected tables related to NYPL Dataset. The analysis yielded a schema and relationships between the tables. From there we set cases for three scenarios, a use case, a zero-cleaning case and never enough case.

We discovered that that there were many data quality issues such as non-possible years in tables with year and/or date column, multiple spellings for similar items, illogical values, for example a different value for same event, and the list can go on.

The team proceeded to break the tables among us and proceed with cleaning. As noted above we encountered all the standard cleaning issues one would expect.

To conclude, the number of items to clean and integrity checks to perform could be quite large however from a use case perspective the only table that required to be cleaned is the Menu table because we want to count how many menus are of a particular dimension. This is not to say that cleaning isn’t important. The cost to clean other tables is not that great and having clean data will lend itself to follow up queries on the back of use case if the user wishes.

The work was completed as follows:

Joel – cleaning Menu and dish table via OpenRefine

Anthony – cleaning MenuPage via OpenRefine, analysis of schema and data quality issues.

John – cleaning MenuItem with Python, Parsing ‘Physcial\_descritption’ column on Menu table with Python and ensure that each record is unique.