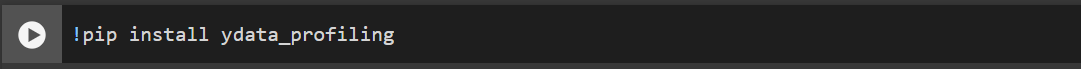
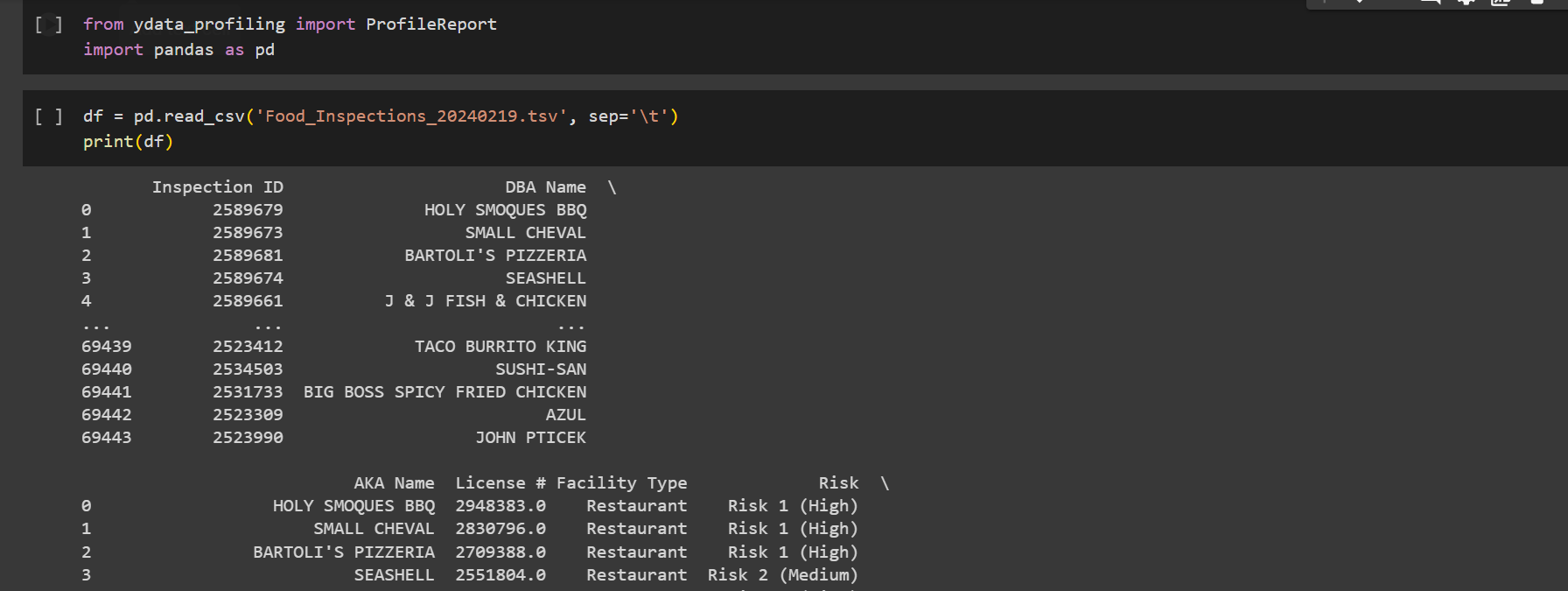
**Y Data Profiling Chicago**



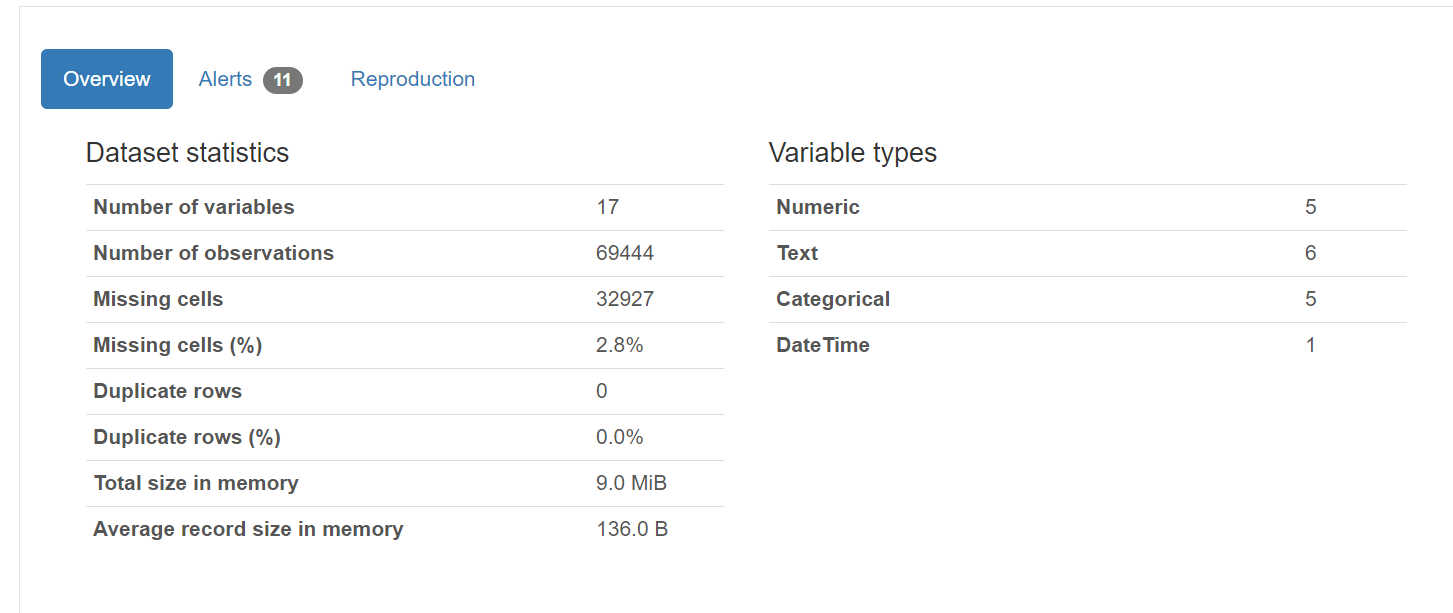


* Initially started with installing the Y data profiling python library.

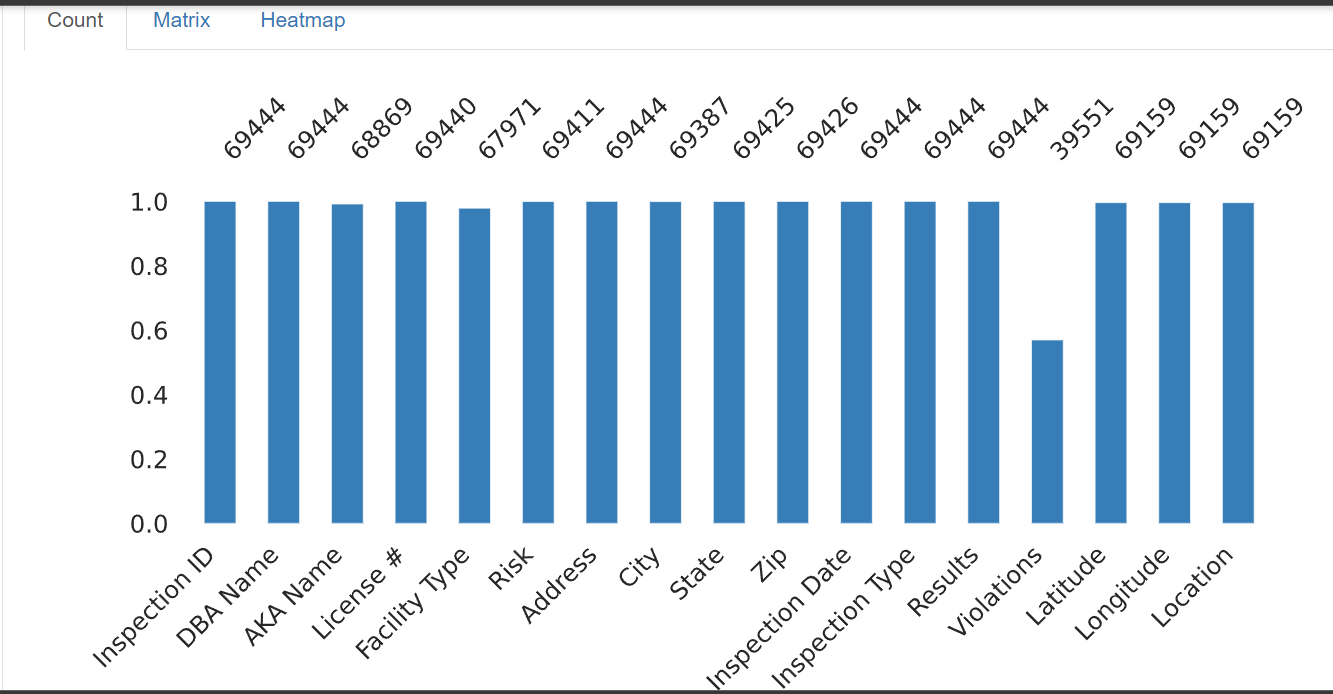
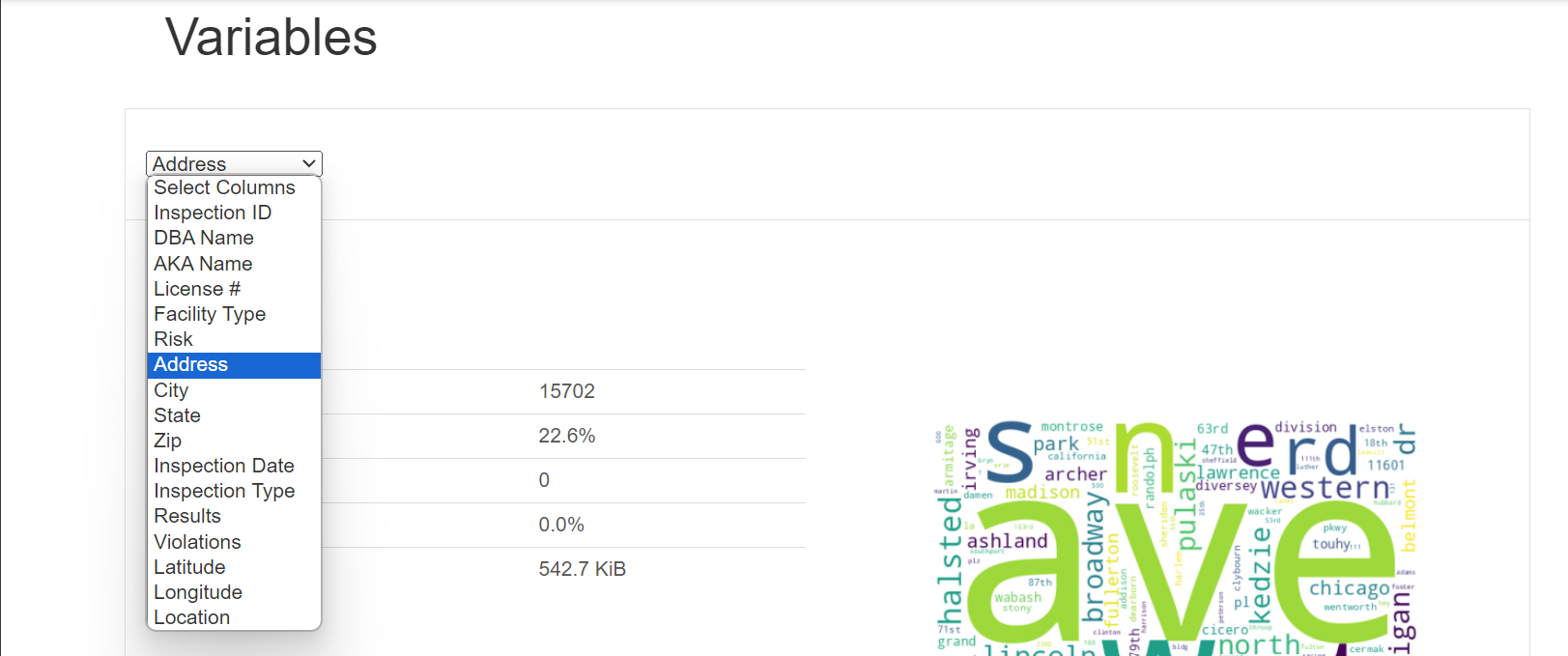


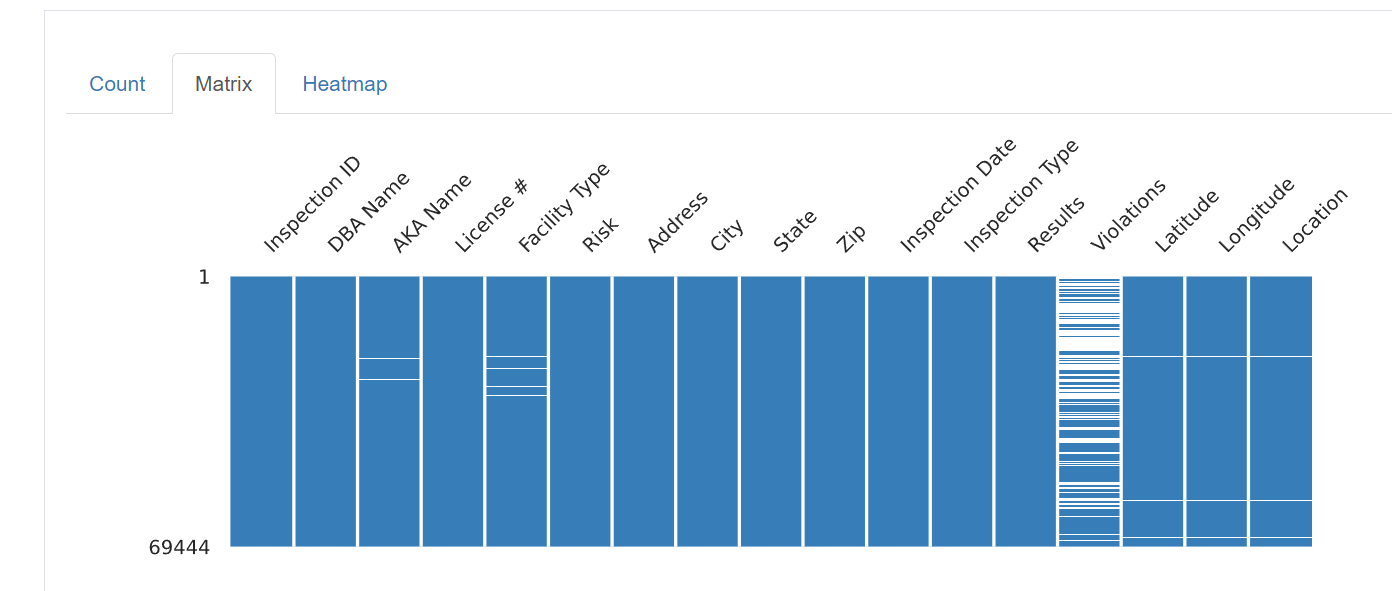


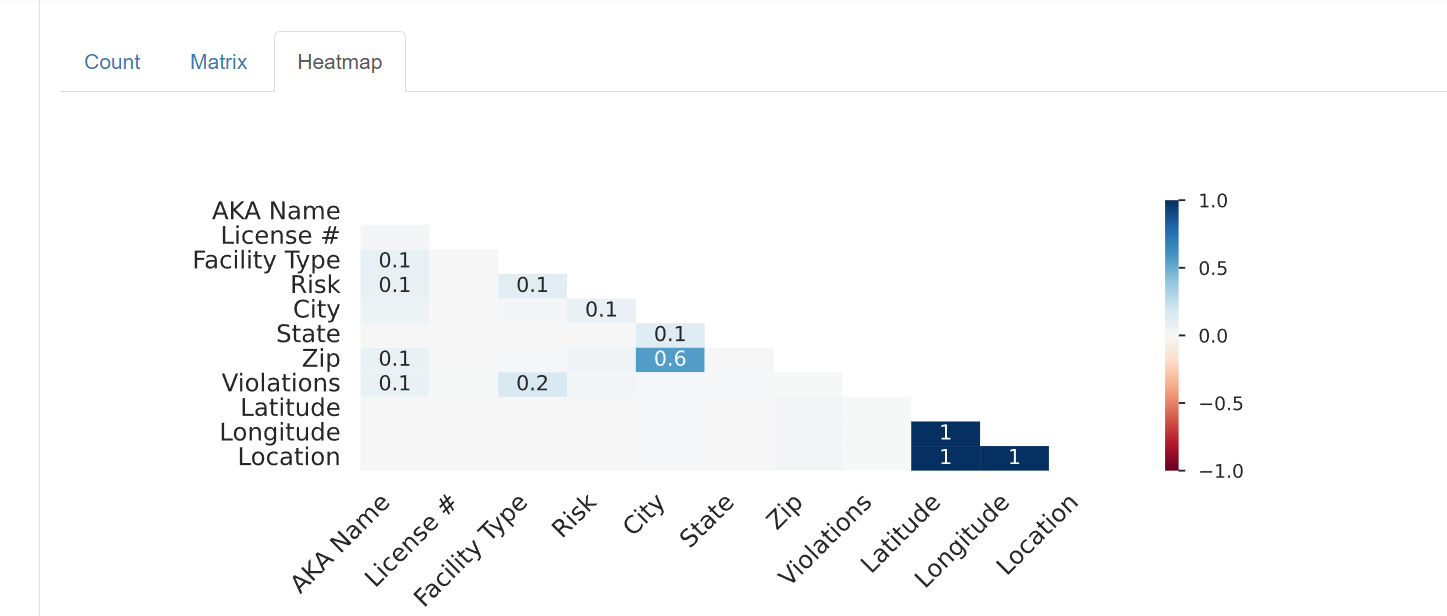
Initialized with the profile report by importing pandas library, then gave the command to read the dataset with tabulation as separator.

1. 

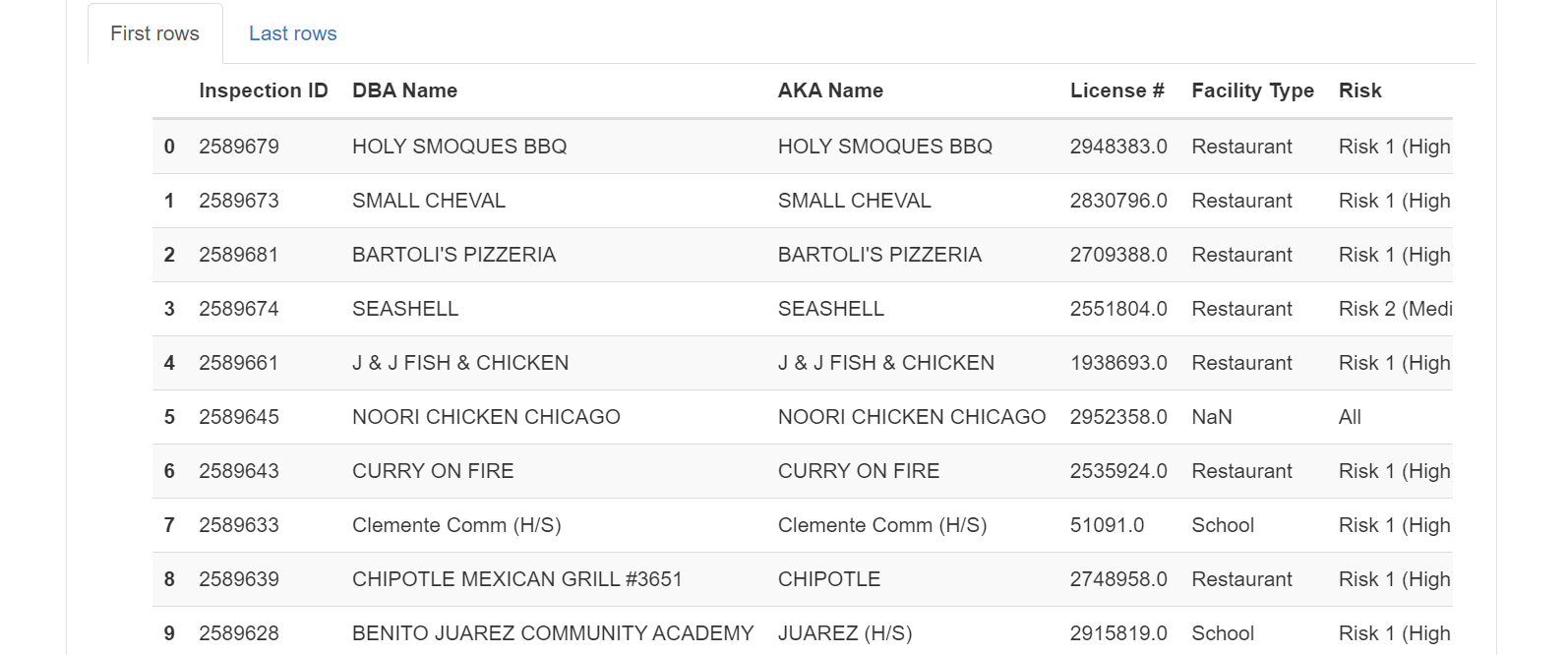
This the complete description/statistics of the dataset.

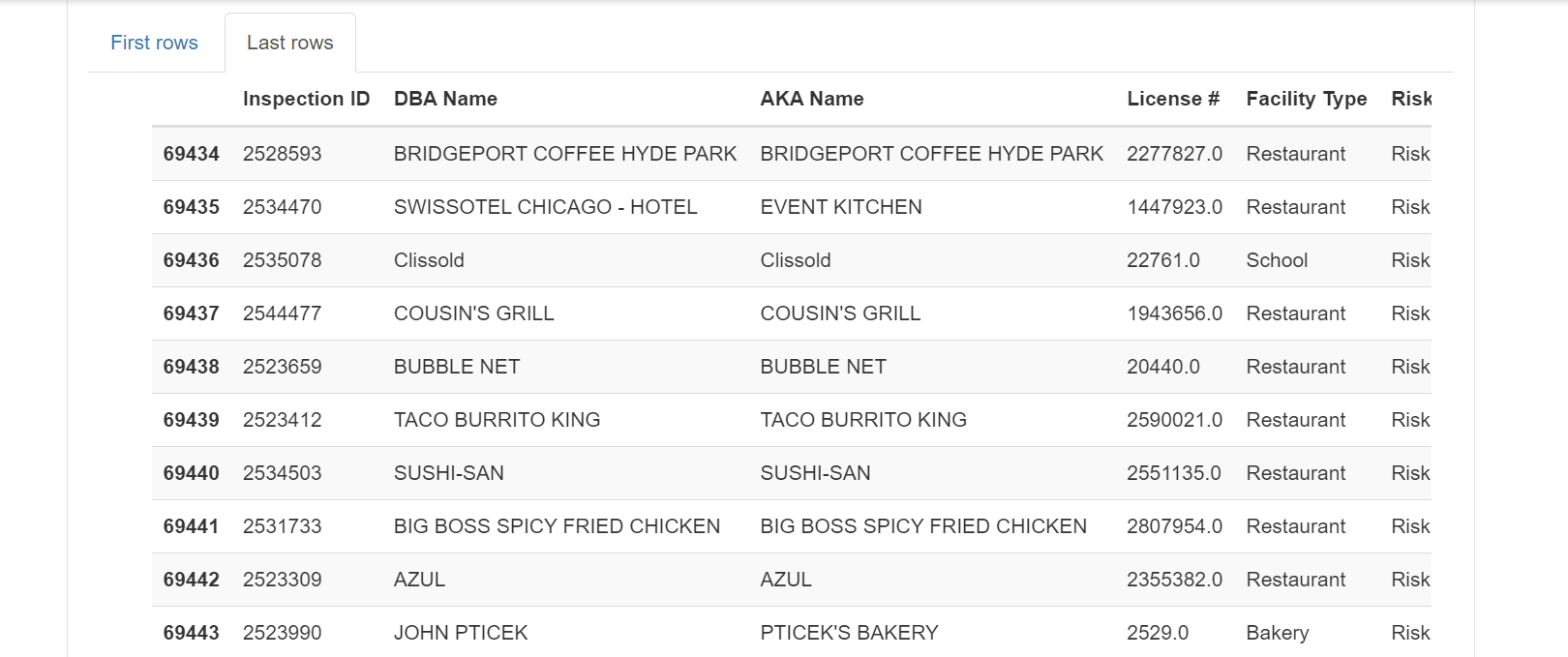
1. We can also get to know various things about the variables such as its length, correlation heat map of the columns of the dataset and we can get the missing values on the basis of count, matrix and heatmap.  
     
   











* Profile report gives top 9 and bottom 9 samples of the dataset
* Inference on how to handle the multivalued attribute and datatype conversion: -   
    
  🡪 **Multi-Valued Attributes**:

Identification: We can review the dataset to identify any columns that may contain multi-valued attributes. These attributes are often found in columns that could list multiple items separated by a delimiter, such as violation codes or categories of services offered.

Assessment:

Assess the significance of each multi-valued attribute in the context of your analysis. Determine whether these attributes need to be separated for individual analysis or can be summarized as is.

* Transformation: For columns deemed necessary to separate, decide on an approach for transformation. This could involve:

Splitting into Multiple Columns: If the number of values is small and consistent, you might split these into several columns.

Expanding into Multiple Rows: In cases where each value needs to be analysed separately, consider transforming each value into its own row while duplicating other column values to maintain context.

* Aggregation: For analysis purposes, it might be beneficial to aggregate or summarize these multi-valued attributes, especially if they relate to violations or compliance checks. Summarization could involve counting occurrences, identifying presence or absence of specific values, or other statistical measures.  
    
  🡺🡺 It does not appear that the dataset contains traditional multi-valued attributes where multiple values are stored in a single column using a delimiter. Instead, violation-related information is spread across multiple columns, each potentially representing a different violation.

Violation Columns: Each set of violation details (description, points, detail, and memo) is spread across multiple columns, one for each potential violation noted during an inspection. Rather than being multi-valued within a single column, they are 'wide' format.

* Solution: If analysis on violations is required, consider consolidating these into a structured format where each row represents a single violation, linking back to the inspection through an identifier. This could involve melting the data frame or using a relational database structure.  
    
  🡺🡺Data Type Conversions

Several columns require data type conversions to be fully usable for analysis:

* Inspection Date: Currently an object (string) type.
* Solution: Convert to datetime to facilitate time-series analysis and chronological sorting.
* Inspection Score, Street Number: These are correctly identified as integers (int64). No conversion needed unless there is a specific requirement for them to be floating-point numbers.
* Inspection Year: Currently an object, likely because it includes a fiscal year notation.
* Solution: If the fiscal year needs to be analysed numerically, consider extracting the year as an integer. If the fiscal notation is crucial, consider keeping as is or creating a separate numeric column.
* Lat Long Location: Appears to contain both address and geolocation data.
* Solution: If geospatial analysis is needed, parse the latitude and longitude into separate numeric columns (float) for use with geospatial libraries.
* Violation Points and Details: These are likely numeric but stored as objects due to potential missing values or mixed types.
* Solution: Convert to numeric types where appropriate. We can use pd.to\_numeric(), handling errors or missing values as needed.
* Categorical Columns like Inspection Type, Street Type, and Inspection Month: Currently stored as objects.
* Solution: Convert these to Pandas categorical type for more efficient storage and faster operations on categorical data.  
    
    
  **🡺 Date and Time Conversions**:

Convert any columns identified as date or time from string formats to Python's datetime format. This facilitates time-series analysis and allows for easier date and time manipulations.

* Numeric Conversions:

Ensure columns that contain numeric values but are stored as strings due to formatting or data entry issues are converted to appropriate numeric data types (int for integers, float for floating-point numbers).

* Categorical Data Handling:

For columns with a limited set of values that represent categories, convert these to Pandas' categorical type. This optimizes memory usage and enables easier application of category-based analysis and visualizations.

* Consistency Checks:   
  After conversions, perform consistency checks to ensure that the transformations have been applied correctly across the dataset. This may involve validating a sample of data or using summary statistics.