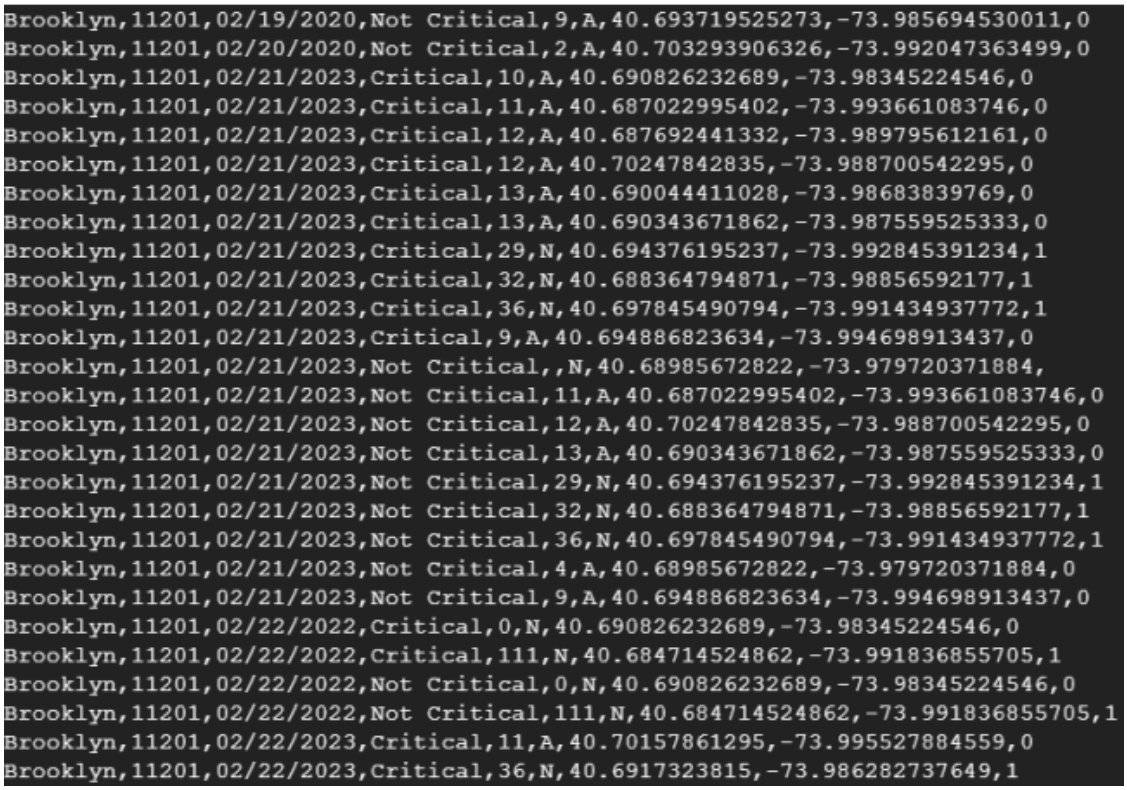
Initial Cleaning:

1. Taking the original data (found in so_etl_code/inspections.csv) I utilized the classes CleanMapper1.class, CleanerReducer.class, and Clean.class (all located in etl_code/so_etl_code/Initial Cleaning) to clean the data. The new clean data is in data_ingest/so_data_ingest/inspections_clean2.csv.

Data cleaning took all the rows from the original dataset that had adhered to our desired schema. For example, boroughs must be a member of the set {Manhattan, Brooklyn, Bronx, Queens, Staten Island}, zip must be a number, critical must be a member of the set {Critical, Not Critical, Not Applicable}, etc.

Second Cleaning:

I cleaned the data further by conducting text formatting and creating a binary column. For the text formatting, I made all records without a valid grade (those with a null, N, P, or Z value) to be normalized to a value of N. For creating a binary column, each record with a health score better than the mean (17, see data profiling section) got a value of 1, while all that had a worse record got a value of 0. The code cleaning was done by running the etl_code/so_etl_code/Second Cleaning/cleaner.jar file on the cleaned data set (etl_code/so_etl_code/Second Cleaning/inspections_clean2.csv) and the result is stored in etl_code/so_etl_code/Second Cleaning/inspections_clean3.csv. Sample running script: hadoop jar cleaner.jar Clean inspections_clean2.csv output. Below is a screenshot of what part of the new data looks like:



Data Profiling:
* All profiling was done with MapReduce

Unclean data can be found at: etl_code/so_etl_code/inspections.csv
Clean data can be found at: etl_code/so_etl_code/Second Cleaning/inspections_clean3.csv

There were several stages of profiling. First, determining how many unique records there are.

1. This was done by simply counting the number of records (lines) present in our CSVs.
2. The original, unclean dataset had 206149 records. This was found by conducting a map-reduce line count on the original, uncleaned data. Sample script:"hadoop jar CountRecs.jar CountRecsinspections.csv output." Here is a screenshot of this profiling that was done for homework 7:

```
rvs6736_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -cat hw7/CountRecs/output/part-r-00000
Total number of records in dataset:     206149
```

3. The clean dataset had 104265. We lost many records due to incorrect adherence to our desired schema. Here is a screenshot of this profiling that was done for homework 7:

```
rvs6736_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -cat hw7/CountRecs/output2/part-r-00000
Total number of records in dataset:     104265
```

Next, we had counting unique grade occurrences, unique critical violation categorizations occurrences, unique zip codes, and unique borough occurrences.

Here are the results from all of these profiling processes. The necessary jar files are all included in the profiling_code/so_profiling_code directory. Here are the results from homework 8:

```
rvs6736_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -cat hw8/UniqueCriticalCount/part-r-00000
Critical        52326
Not Applicable  5727
Not Critical    52454
rvs6736_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -cat hw8/UniqueGradeCount/part-r-00000
        46127
A       52218
B       4464
C       1699
N       2876
P       507
Z       2616
rvs6736_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -cat hw8/uniqueBoroughCounts
Bronx   10432
Brooklyn        29288
Manhattan       41403
Queens  25494
Staten Island   3890
rvs6736_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -cat hw8/UniqueZipcodesCount/part-r-00000
Total Unique Zipcodes:  219
```

I also wanted to know the mean score… it was 17.

```
rvs6736_nyu_edu@nyu-dataproc-m:~$ hdfs dfs -cat hw8/meanScore/part-r-00000
Average Score:  17
```

This can be reproduced by running the meanScore.jar with the MeanScore class on the clean data.