# CS-GY 6513 Big Data Final Project

ROBERT RONAN, rr3749
SHENG TONG, st4048
JERRY LEE, jl11517

Github repo: https://github.com/darkcb109/DOB-Job-Application-Cleaning

## 1 INTRODUCTION

For this project we profiled, analyzed, and cleaned the New York City Department of Buildings "Job Applications Filings" dataset, which is publicly available on data.cityofnewyork.us/. As increasingly large amounts of data are collected, and as the push for government agencies to share their data with the public, the capacity to clean large amounts of data will become increasingly important. Frequently, datasets contain erroneous values, and have data stored in multiple, incompatible formats. When datasets that have not been properly cleaned are joined, this problem can become even worse.

In this project, we cleaned the DOB Job Application Filings dataset, applied the techniques we used on that dataset to ten similar datasets, and measured their effectiveness. All of our code is written in python, and we utilize multiple tools including Pandas and VIDA-NYU's OpenClean library. We present some prominent issues we encountered and some methods we developed or used to try and address those problems. Finally, we will discuss our findings and other problems we discovered that proved more difficult to address.

The objective of our work was to find and fix most of the problems in a large dataset and to extend this work to handle more numerous, or larger datasets. We cover many of the problems addressed in the DOB Job Application Filing dataset, how we addressed these issues, and how we attempted to generalize our approach to encompass additional datasets.

## 2 PROBLEM FORMULATION

Firstly, we analyze column names and discover that many columns had names which were vague and inconsistent with the rest. We address misspellings, capitalization inconsistencies, as well as how certain names and symbols, such as "No.", "#", "Number," should be represented. Other naming issues we encountered were columns with vague names; for instance, the "City" column was referring to the city of the owner's house rather than the city where the job for which the application was filed takes place.

Next, we looked at some fields with some simple profiling and separated the issues address based on the type of field (numeric, monetary, datetime, phone numbers, binary data, etc.) and modified them to be self-consistemt, as well as correcting some improper values. The most frequent problem we encountered were the inconsistent uses of None, NaN, and blank space. Additional problems involving incorrect data such as zip codes outside NYC, misspelling of names, and inconsistently formatted name values (e.g. NYC, New York, New York City).

One problem that we discovered which was particularly hard to address was the incoherence between certain values, such as assigned dates being later than approved dates. However, since the fraction of jobs that had this issue was very minimal, and we did not possess the domain knowledge to be sure this was impossible, we left these values as is. We

Authors' netid: Robert Ronan, rr3749; Sheng Tong, st4048; Jerry Lee, jl11517.

further used kNN clustering to find outliers for some names, cities, and dates. We also took a quick glance at the GIS fields such as latitude and longitude and noticed entries with zeros in these fields, and looking further, found out that the entire row of data was filled with incorrect information. We removed these rows entirely,

After cleaning and profiling our data, we tried to broaden our scope, so that our methods would work on different but similar datasets. We took a look at many other Department of Buildings datasets since they had many similar fields, but also a handful of datasets from other agencies. A problem that had failed to account for originally was that many of the fields were hard-coded in our data cleaning code. We tried to address this issue by rewriting our code to be an extremely general model of our data cleaning techniques. However, there were other methods that we couldn't generalize successful, and had to address manually. One of these is the usage of clustering on extremely large datasets, which tended to result in out of memory errors. Other fields were also too ambiguous for us to determine in general which type of data they contained. For instance, we wouldn't be able to tell necessarily if a column of mostly ten-digit values referred to phone numbers or some other ID number, and thus we could not always clean it.

## 3 RELATED WORK

Since we used Pandas and OpenClean to assist with the data cleaning, we used the OpenClean examples provided in the OpenClean repository as guides. Although we didn't use many of the OpenClean functions, as two out of three of us encountered issues DOB Job Applications Filings dataset through OpenClean, we used Pandas to clean and profile easier fields. However, clustering and soundex was much easier to implement, since it was pre-programmed, so we also used the OpenClean's clustering functions to help determine outliers and clean the data. Link to OpenClean repository: https://github.com/VIDA-NYU/openclean-core

## 4 METHODS, ARCHITECTURE AND DESIGN

We utilized a number of distinct methods to profile and clean the DOB Job Application Filings data set. Below, we will discuss different types of errors and inconsistencies we found, and how we attempted to address them.

### 4.1 Column Names

We identified a number of issues with the original column names. Some column names were missing spaces between words, some had multiple spaces. Some column names used underscores to separate words, and some used spaces. Different column names used entirely different conventions for capitalization and abbreviations. For example, the word "number" was sometimes written out entirely, sometimes written as "No." and sometimes as a #. Words in column names were occasionally abbreviated, whereas other column names which were significantly longer contained no abbreviations. Finally, some column names were ambiguous to the point of causing confusion. The column "Paid", did not actually indicate a boolean value, but was the date that payment was rendered on.

To address these issues, we settled on a uniform naming convention for the column names. We replaced all underscores with spaces, made columns title case except for all capital abbreviations. We changed all versions of "number" to the word "number"; and we append "Date" to datetime column names that did not contain it.

### 4.2 Numerical Columns

For numerical columns, our goal was ultimately to convert them into integers or floats. To this end, we removed non-numeric values and dollar signs. Doing this allowed us to then check the distribution of the column values more easily, and to spot outliers. In particular, checking the minimum and maximum values of the columns often identified obvious outliers and errors. When looking at monetary values, we discovered a number of instances in which the amount of money listed was astronomically large, or was negative in cases where that would not have made sense. We also looked at columns containing the proposed number of feet and stories for buildings, and we were able to spot erroneously large values by comparing these against the tallest buildings in New York City.

Unfortunately, while these methods allow us to address issues in our numerical columns, many of them are context specific and difficult to automate. For instance, none of our numerical values could be negative, but this is clearly not true in general for arbitrary numerical columns. Examining outlying values in these columns allows us to identify a number of problems, but also requires 'expert knowledge' of the columns, which is not always present.

### 4.3 Boolean/Binary Columns

We identified a number of columns that were implicitly boolean, and which were more usefully represented with boolean values. Often these column names were questions in the form of statements like, "pre-filed", or "landmark." Frequently they contained values of "Yes/No" or "Y/N", but often a significant number of NaNs. We identified numerous columns that appear to have been checkboxes at some point because their only values were "X" and NaN.

For these columns, we converted values indicating yes, including "X" to True, and other values to False, including NaNs. Some of these columns also contained a very small number of erroneous values, like one instance of the letter 'H.' However, There were columns that looked very close to binary, but did have legitimate third or fourth options, so our approach was somewhat column specific.

### 4.4 Phone Numbers

Columns containing phone numbers were riddled with errors. There was no consistent format for the phone numbers (some had dashes, some spaces, some parentheses). Frequently, the listed phone numbers were missing digits or had too many.

For the phone numbers, we stripped all non-numeric characters, and dropped all numbers that were not ten digits, or that started with a 0 or a 1, which are not valid phone numbers in the United States.

### 4.5 House Numbers

House numbers were difficult to parse and identify errors in because house numbering is not as uniform as one would hope. Some house numbers were spelled out words, which we removed, but since there was no apartment column, many of the house numbers included either letters or the words "Garage" or "Rear".

While something like "200 Garage" may not typically be considered a "house number", our data set was about Department of Buildings construction job applications, and so in our case, it is likely very relevant whether the work is happening on the garage instead of the house, or on the apartment in the rear of the house. Thus, we standardized the formatting by inserting spaces between the numerical component of the house number and any letters.

### 4.6 New York City Specific Information

Here we examined the associated Council Districts, Latitude and Longitude, and confirmed that all of these values fell in the appropriate range for New York City based data.

### 4.7 DateTime Data

Primarily the issues we identified with datetime data was that dates were stored seemingly randomly in month/day/year format, and in year-month-day format. After fixing the datetime formatting, we checked values from multiple datetime columns for coherency.

We used Pandas's built-in to_datetime function to convert all the datetimes to the preferred year-month-day format. In a contextual manner, we checked the coherency of various datetime columns by assessing what proportion of them indicated events occurring in the wrong order. For example we looked for data instances where the "Paid Date" was strictly after the "Fully-Paid Date", or where the job was approved before it was assigned, or where the 'Pre-Filing Date" was after the "Latest Action Date." We discovered that in a relatively small number of examples, one of these was the case, but as we lacked the expert knowledge of whether this was technically possible in some way, we did not remove these values.

### 4.8 Name-Related Data

For columns with name data, like business names, applicant professional titles, first and last names, we used OpenClean's K-Nearest-Neighbor clustering to identify and rectify outliers, as well as to standardize the formatting. There were more potential formatting problems than we could reasonably list here, but we identified no consistency in capitalization, punctuation, abbreviation, or alliteration. We also identified typos.

Given the breadth of potential errors, we figured our best bet would be to use some type of clustering algorithm to remove formatting issues, and fix misspellings. For this task, we used a very high confidence level that multiple values were in the same cluster before we associated them all with the clustered value. This approach proved quite successful in addressing punctuation, capitalization, and some other issues like abbreviations. However, we did notice multiple clusters that mapped a number of correct, but distinct values to the same name. It is possible that we needed to tweak the settings more. We were also unable to cluster on some columns we had hoped to, due to their size and complexity.

### 4.9 Cities

For city name data (like (business) owner's city), we identified issues by using OpenClean's Soundex implementation alongside an encyclopedia of United States Cities. We identified frequent misspellings of cities, and by also using OpenClean's functional dependency violations module, we were able to correct a number of incorrect city-state pairs that users had entered.

Many of these errors were corrected by hand, which unfortunately did not scale well to other data sets. However, since all the data sets were New York City based, some corrections still applied.

## 5 RESULTS & DISCUSSION

We ran our code over a sample of 50 instances per column that we cleaned, and measured our precision and recall over this set of data. We measured the precision and recall before we improved our code, and afterwards.

| Dataset | Precision (Original) | Recall (Original) | Precision (Improved) | Recall (Improved) |
|---|---|---|---|---|
| DOB Cellular Antenna Filings | 0.995 | 0.577 | 0.863 | 0.984 |
| Open Restaurant Applications | 0.474 | 0.538 | 0.995 | 0.986 |
| DOB NOW: Safety – Facades Compliance Filings | 0.996 | 0.793 | 0.864 | 0.979 |
| DOB Permit Issuance | 0.990 | 0.746 | 0.987 | 0.984 |
| DOB NOW: Build | 0.947 | 0.904 | 0.944 | 0.962 |
| DOB NOW: Electrical Permit Applications | 0.986 | 0.917 | 0.877 | 0.903 |
| Historical_DOB_Permit | 0.945 | 0.738 | 0.977 | 0.843 |
| Housing New York Units by Building | 0.933 | 0.767 | 1.0 | 0.885 |
| DOB NOW: Build – Approved Permits | 0.975 | 0.826 | 0.845 | 0.760 |
| Buildings Selected for the Alternative Enforcement Program (AEP) | 1.0 | 0.778 | 0.935 | 1.0 |

The precision and recall differ heavily across dataset, which may imply our original methods did not cover every field. This also meant that we knew what parts need to be fixed. These 10 new datasets have similar columns, but the data is very different. We manually looked into each of the datasets and performed data profiling and then data cleaning on them. Below, we discuss some of the main issues we found during the process:

A majority of our false negatives were from NaN values in non-numeric values. Our decision was to make empty or none for numerical values NaN while names become an empty string, and NaT for time values. However, when we used clustering techniques, there were columns that couldn't be changed significantly without creating many false positive errors. For example, the names of businesses, and people's first and last names were extremely challenging to address in this manner without unintentionally modifying correct names. In the Job Applications Filings dataset, we ran into an issue where "NYC Housing Authority" had many different spellings which led to a problem when clustering because if we used a stronger classification, or equivalently a lower confidence value, many other businesses would get clustered into various clusters with different spellings of NYC Housing Authority, and would then usually be mapped to the name NYC Housing Authority. Ultimately, we decided to use a very high confidence/similarity level for the clusters.

For datetime data, we have different formats, often within the same column, and were unable to convert them correctly with clustering. However, we used Pandas to_datetime function to successfully correct these columns.

A serious problem we encountered with our initial code, was the need to carefully look into each dataset to decide how to clean the columns in detail, so the results were a high quality of cleaned data; however, this would require a tremendous amount of time and effort to perform on every dataset in the NYC Open Data database.

We generalized our code in such a way that we hope it could run on most of the NYC Open Data datasets. To run our cleaning strategy over all datasets would require it to be entirely column-name agnostic, and to infer the data types directly from the columns themselves. We have partially done this; by trying to infer column data types from a combination of non-hardcoded column names and data values, so the code now automatically finds columns to clean, and determines how to clean the data based on our experience of manually cleaning these datasets.

For instance, we search column names for the word 'date', and presume that any column with 'date' in its name is a datetime column, which we proceed to clean as if it were. Similarly, we search for columns with 'number' in their names to try and determine numerical columns. From those columns, we also search for"building" and "number" or "house" and "number" to find house numbers which we clean in a specific way, as they are more akin to strings than numerical values. We also try to infer when columns are adequately represented as boolean columns, by searching for columns with a small number of distinct values that also contain values like "yes", or "N", or True.

Our Improved version sometimes yields slightly worse results on certain columns, since the improvement we made to our methods was to generalize every method to work on a multitude of datasets rather than hard code some values. This would naturally mean that some edge cases would not be covered. Unfortunately, sometimes our code will clean one column twice in different ways, for instance, we automatically find columns related to "city" and "name" to fix, but then the column "City Name" will be first treated as city and then treated as name which may cause a lot of false positives.

After several changes and improvements to our generalized code, we had it automatically run on all 10 datasets, and we manually evaluated its result in terms of precision and recall. As the chart above shows, the precision often (but not always) decreased somewhat since our improved code was less specific than our original, but typically this decrease in precision was more than offset by an increase in recall. Overall, our improved results were significantly more balanced in precision and recall. Additionally, the average F1 score for our datasets improved from 0.828 in our original code to 0.927 in our improved code, suggesting the improved results are of higher standard.

Our newer implementation means that technically and practically, our code can be run across all NYC Open Data datasets to clean them. However, because we've made our code so much more general, it's possible it will create many false positives and false negatives on datasets dissimilar to ours. To provide equally broad, but more accurate cleaning across all these datasets, we would ideally have to implemented, or more effectively utilized, something similar to what Pandas attempts to do in inferring column types when the information is not provided.

## 6 REFERENCES

[1] DOB Job Application Filings
https://data.cityofnewyork.us/Housing-Development/DOB-Job-Application-Filings/ic3t-wcy2

[2] Housing New York Units by Building
https://data.cityofnewyork.us/Housing-Development/Housing-New-York-Units-by-Building/hg8x-zxpr

[3] DOB NOW: Build – Job Application Filings
https://data.cityofnewyork.us/Housing-Development/DOB-NOW-Build-Job-Application-Filings/w9ak-ipjd

[4] DOB NOW: Electrical Permit Applications
https://data.cityofnewyork.us/City-Government/DOB-NOW-Electrical-Permit-Applications/dm9a-ab7w

[5] DOB NOW: Build – Approved Permits
https://data.cityofnewyork.us/Housing-Development/DOB-NOW-Build-Approved-Permits/rbx6-tga4

[6] DOB Permit Issuance
https://data.cityofnewyork.us/Housing-Development/DOB-Permit-Issuance/ipu4-2q9a

[7] DOB NOW: Safety – Facades Compliance Filings
https://data.cityofnewyork.us/Housing-Development/DOB-NOW-Safety-Facades-Compliance-Filings/xubg-57si

[8] Historical DOB Permit Issuance
https://data.cityofnewyork.us/Housing-Development/Historical-DOB-Permit-Issuance/bty7-2jhb

[9] Buildings Selected for the Alternative Enforcement Program (AEP) https://data.cityofnewyork.us/Housing-Development/Buildings-Selected-for-the-Alternative-Enforcement/hcir-3275

[10] Open Restaurant Applications
https://data.cityofnewyork.us/Transportation/Open-Restaurant-Applications/pitm-atqc

[11] DOB Cellular Antenna Filings
https://data.cityofnewyork.us/Housing-Development/DOB-Cellular-Antenna-Filings/iz2q-9x8d

[12] OpenClean
https://github.com/VIDA-NYU/openclean-core