NYC Motor Vehicle Collisions - Data Cleaning

HAO DONG, New York University, USA
SIWEI WANG, New York University, USA
YINCHEN SHI, New York University, USA

In New York City, tremendous amount of data are generated every seconds, and data are been collected in massive dataset. Collecting data in many fields has become a ubiquitous function of companies and large research organization, for example, NYC Open Data records thousands of datasets and it supports a variety of data analytic and processing for data professionals, data scientists to accomplish revolutionary breakthroughs in many realms. Despite the significance of data, professional needs high quality of data to This project aims to research and implement different cleaning methods that focus on error detection, missing tackling and validity check for the dataset NYC Motor Vehicle Collision in NYC and several other related datasets.

Additional Key Words and Phrases: Data Cleaning, Large Datasets, Hadoop, Spark, OpenClean, Geopy, Pandas, Matplotlib, Distributed Systems

ACM Reference Format:
Hao Dong, Siwei Wang, and Yinchen Shi. 2021. NYC Motor Vehicle Collisions - Data Cleaning. 1, 1 (December 2021), 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the explosion of the Internet and the rapid development of science and technology, more and more individuals, enterprises and governments share their data on the Internet, for example, Real-Time Traffic Speed Data, Street Tree Census Data, Public Recycling Bins, Motor Vehicle Collisions Data. These data are diverse, massive and abstruse, and it may be tough to deal with these data using a traditional method. Therefore, some technologies and frameworks will be applied to process and analyze large amounts of data, such as MapReduce and Spark. Data cleaning is also a vital process during data processing. People may make mistakes when they record data, so it is necessary to normalize, standardize, and validate data. This article will illustrate a general data cleaning method developed based on Motor vehicle collision dataset, which contains millions of motor vehicle crashes in New York City, and evaluate the efficiency of the method when applied to ten other datasets.

2 PROBLEM FORMULATION

Data profiling will be the first stage of problem formulation. Initially, applying profiling methods from OpenClean to overview the Motor Vehicle dataset, the profiling results will indicate the data types and number of distinct and empty data for each column. Based on the profiling result, the following problems have been observed.

Authors' addresses: Hao Dong, New York University, New York, USA, @nyu.edu; Siwei Wang, New York University, New York, USA, sw5050@nyu.edu; Yinchen Shi, New York University, New York, USA, @nyu.edu.

The CRASH DATE, a date type data, is a vital attribute for Motor Vehicle Collisions Data. If a row of Motor Vehicle Collisions Data has the wrong CRASH DATE, the row will also be meaningless. There are three types of errors for CRASH DATE —— missing, incorrect data types and incorrect values. The first two are easy to understand. As for error values, we need to combine them with the regular time expression to identify. All dates after today, date, error day, error month, and year before the 1970s (zero to computer time) are incorrect.

As for the second column CRASH TIME, it is similar to CRASH DATE. Time and Date data have the same priority, and they have the same type of error. Missing and incorrect data types are also easy to understand. One day begins at 00:00 and ends at 23:59. So it is straightforward to find outliers and anomalies for error values.

Next, move to the third and fourth columns BOROUGH and ZIP CODE. They are complementary data. Every ZIP CODE can match unique BOROUGH. Because BOROUGH is a string type and ZIP CODE is an int type in python, the incorrect data types still exist. Except that, there are new error ways for BOROUGH, spelling mistakes, such as "brookln", "braklyn", "brookly" and the right type "brooklyn" and case error, such as "brooklyn" and "BROOKLYN".

After that, the fifth to seventh columns are location information -— LATITUDE, LONGITUDE and LOCATION. Because LOCATION consists of LATITUDE and LONGITUDE, they appear at a row within the same values. Moreover, for the accuracy of location, LATITUDE and LONGITUDE need to be a double type.

The eighth to eleventh columns are street names, ON STREET NAME, CROSS STREET NAME and OFF STREET NAME. ON STREET NAME and CROSS STREET NAME can location a site by the intersection of two roads, and OFF STREET NAME is the exact address where collisions happen. So ON STREET NAME and OFF STREET NAME are mutually exclusive, and CROSS STREET NAME occurs with ON STREET NAME. The street name is a more complex string than BOROUGH, so that it may have lots of spelling mistakes.

To sum up, there are three significant problems - irregular punctuation, a combination of English numerals and Arabic numerals, and exact location with a different name. For instance, E 16 ST, E 16RD ST, EAST 16RD ST, EAST 16 ST, and EAST 16RD STREET are different formats but point to the same address. Above all, we are supposed to normalize the format of street names. According to data profiling results, some attributes have empty values. For example, missing zip code, latitude, longitude, location, on street name, cross street name, and off street name.

For NUMBER OF PERSONS INJURED, NUMBER OF PERSONS KILLED, NUMBER OF PEDESTRIANS INJURED, NUMBER OF PEDESTRIANS KILLED, NUMBER OF CYCLIST INJURED, NUMBER OF CYCLIST KILLED, NUMBER OF MOTORISTS INJURED and NUMBER OF MOTORISTS KILLED. They are not fallible. According to profiling results, there are some empty values, and we can solve this problem by filling zeros. As for CONTRIBUTING FACTOR VEHICLE 1, CONTRIBUTING FACTOR VEHICLE 2, CONTRIBUTING FACTOR VEHICLE 3, CONTRIBUTING FACTOR VEHICLE 4, and CONTRIBUTING FACTOR VEHICLE 5. It has the common error type of string data, missing, incorrect data types, and spelling mistakes. VEHICLE TYPE CODE 1-5 is the same as CONTRIBUTING FACTOR VEHICLE. According to the entropy of each attribute calculated by openclean, some data have little impact on the results, like CONTRIBUTING FACTOR VEHICLE 3-5 and VEHICLE TYPE CODE 3-5. So we will remove them.

In the end, the last one and the most critical column, COLLISION ID. It can be considered as a primary key of the scheme. Every row of Motor Vehicle Collisions Data has a unique identifier to identify only Motor Vehicle Collisions. Thus, duplication of collision id is prohibited.

3 RELATED WORK

3.1 PySpark

PySpark is an integrated interface for Apache Spark to write in Python.

3.2 Hadoop

Hadoop allows developers to run large datasets in a distributed processing manner.

3.3 Pandas

Pandas is a fast, productive, user-friendly data analysis and manipulation software framework, which built on Python. In particular, Pandas offers special and easy-to-manipulate data structure 'DataFrame', and it could be productively utilized to clean and structure the data.

3.4 OpenClean

Open Clean is a python library for data preprocessing and data cleaning. It provides large number of useful techniques and built-in functions to support data science projects when developers need to clean tricky data structures. Two main features with Open Clean are data profiling, and data cleaning. Open clean provides great metrics which are visible to see a overall statistics.

3.5 Geopy

Geopy is a popular geo information analysis tool that provides easy methods to locate the coordinates (longitude, latitude etc.) and 3rd parties Geocoders API (Google Maps, Nominatim)

3.6 Matplotlib

Data visualization is very important when it comes to cleaning data. Matplotlib offers a low level graph plotting library in python which helps developers to utilizing as a visualization tool.

3.7 scikit-learn

Scikit-Learn is a machine learning related library for Python, it provides many optimized machine learning algorithms and measurement methods such as confusing matrix, precision and recall functions

4 METHODS, ARCHITECTURE AND DESIGN

4.1 Methods

4.1.1 Missing Geographic Information. According to the statistics from the data profiling stage, we discovered that BOROUGH, ZIP CODE, LATITUDE, LONGITUDE, LOCTION, ON STREET NAME, CROSS STREET NAME, OFF STREET NAME have empty values. As for motor collision data, geographic attributes are vital. This is also reflected in the entropy of each attribute. If all the vital information is missing, the record is useless, and it will be deleted. Specifically, the row with empty BOROUGH, ZIP CODE, STREET NAME(ON/OFF/CROSS), and LOCATION(LATITUDE/ LONGITUDE) will be deleted. In order to tackle this problem, sending requests to the Geocoding API using known attributes, the missing value can be found. For example, a record has a known value ON STREET NAME, which is 'BRONX WHITESTONE BRIDGE'; and three missing values BOROUGH, ZIP CODE, LATITUDE, LONGITUDE, and

LOCATION; the known value is the query parameter which will be sent to Geocoding API; the response of the API is a JSON file which includes all the geography information of the known value (Bronx whitestone bridge). Therefore, missing location values can be extracted from the response of the Geocoding API. According to the above patterns and a formula derived from the Google Maps API, if the row without LOCATION or with zero LOCATION and this LOCATION cannot be calculated, the row will be deleted.

4.1.2 Geographic Information Validation. Geocoding API has been used to validate location accuracy in the records. As mentioned above, a motor collision has seven attributes related to geographic information. To verify the correctness of the location information, send the value of ON STREET NAME to Geocoding API; after that, extracting values from the response JSON file and comparing those values to values in the record, if there is a difference between them, replace values in the record by values from API response.

4.1.3 Uppercase. When there are String type values in any attribute sections, they need to be standardized to the same format (Uppercase letters), such that from BOROUGH, ON STREET NAME, CROSS STREET NAME, OFF STREET NAME.

4.1.4 Abbreviations. There are plenty of abbreviations that can be found in ON STREET NAME, CROSS STREET NAME, OFF STREET NAME. However, the full name of those abbreviations also appears in the records, which made the document confusing. In order to solve this problem, all abbreviations have been replaced by their full name. For example, some ON STREET NAME contain 'ST', which is the abbreviation of 'STREET', and it will be replaced by 'STREET'.

4.1.5 Error Data Type. Some rows have an error data type, like int and date type in ON STREET NAME, CROSS STREET NAME, OFF STREET NAME. The regular expression is an ideal way to solve it.

4.1.6 Missing Data. In the vehicle collision data set, some attributes are missing important information. For example, with the missing value - CONTRIBUTING FACTOR VEHICLE 1-5, we can fill up with 'UNSPECIFIED'. As for VEHICLE TYPE CODE 1-2, we can fill up with 'UNKNOWN'. Also, from Column 10-17, 'NUMBER OF PERSONS INJURED' to NUMBER OF MOTORIST INJURED, replacing the empty value in attributes NUMBER OF PERSONS INJURED/NUMBER OF PERSONS KILLED to 0.

4.1.7 Type Error. There are many types errors, like confusion of case, missing or adding characters, and just errors. As for CONTRIBUTING FACTOR VEHICLE, some numbers are ineffective so that they will be replaced by "UNSPECIFIED". VEHICLE TYPE CODE is complete than CONTRIBUTING FACTOR VEHICLE. There are thousands of vehicle types and type errors. For example, just for the ambulance type, there are kinds of spelling, 'AMB', 'AMBU', 'AMBUKANCE', 'AMBUL', 'AMBULACE', 'AMBULANCE', 'AMBULANE', 'AMBULENCE', 'AMBULETTE', 'AMDU', 'AMUBULANCE', 'AMULANCE'. And replace some empty values with default values.

4.1.8 Redundant Spaces. Every type section with more than one value may contain extra white spaces between two words, such as (2819 MIDDLETOWN ROAD), and it could be trimmed as (2819 MIDDLETOWN ROAD). It makes each section much more clean and tidy

4.2 Architecture Design

In order to clean Motor Vehicle Collisions dataset, we design the following data cleaning architecture. There are six stages in our cleaning architecture.5

The first stage is to explore the dataset. In this stage, we used the NYC OpenData website to be familiar with the dataset, for example, the description of each attribute and the data types of each attribute. Besides that, we also applied the visualization tool provided by NYC OpenData website to have a general idea of the dataset, such as the time range of the dataset, the tendency of the collisions data.

After that, we moved to stage 2, profiling the dataset. OpenClean was introduced in this stage, and we utilized some methods from OpenClean to perform profiling. For instance, we used the stats method, which automatically produced an overview of the profiling results. It shows the total number of values, empty values, distinct values, uniqueness, and entropy of each attribute. We can also use types to know each attribute's actual data type of values. It is helpful for us to have a clear idea of the dataset.

The next stage is to formulate problems. With the help of OpenClean, we can find data types that each attribute has and find if there is any wrong data type. For example, based on our common sense, a zip code usually consists of numbers which is an int type in the US. So, if a string type value occurs in a zip code, it is not appropriate. We can apply minmax method from OpenClean to validate the value of attributes related to the year. Moreover, we can also discover problems manually by searching part of the dataset line by line.

When we complete the problem discovery stage, we will consider solving those problems. In solving the problem stage, we apply OpenClean, Geopy, Pyspark and Pandas to tackle problems. For missing geographic information, geopy can help us fill missing data according to known data. For example, acquiring zip code based on known latitude and longitude and correcting wrong geographic information. Furthermore, for standardizing the case of words and filling empty values, we can apply a lambda function on the Pandas data frame; for normalizing the street name, applying StandardizeUSStreetName from openclean geo.

When problems are solved, we will move to the next stage to improve our data cleaning methods. At the beginning of this stage, we will select ten different datasets with a few similar attributes with Motor Vehicle Collisions dataset. Then, we will employ our data cleaning methods on these new datasets, review the running results, and inspect if there still has some invalid data. If there have some invalid data, we will improve our cleaning methods according to those incorrect data.

The last stage is to evaluate data cleaning methods and cleaned dataset. We will use precision and recall to evaluate our results. We will have precision and recall scores of our cleaned data by applying the related formulas. Then, we can analyze and evaluate the result according to precision and recall scores.

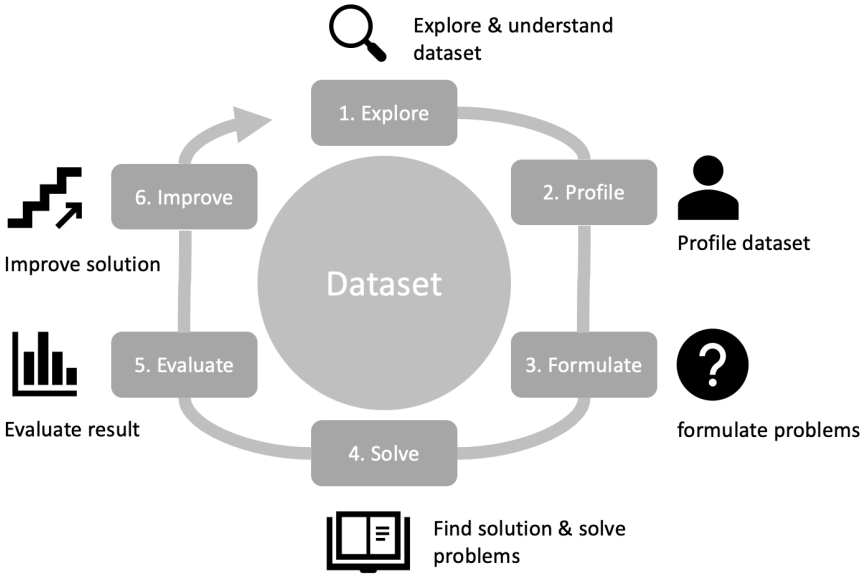

Fig. 1. Data Cleaning Architecture

5 RESULTS

5.1 Visualization and Analysis

To demonstrate the effectiveness of data cleaning, we try to use Pyspark for data analysis and Matplotlib for visualization.

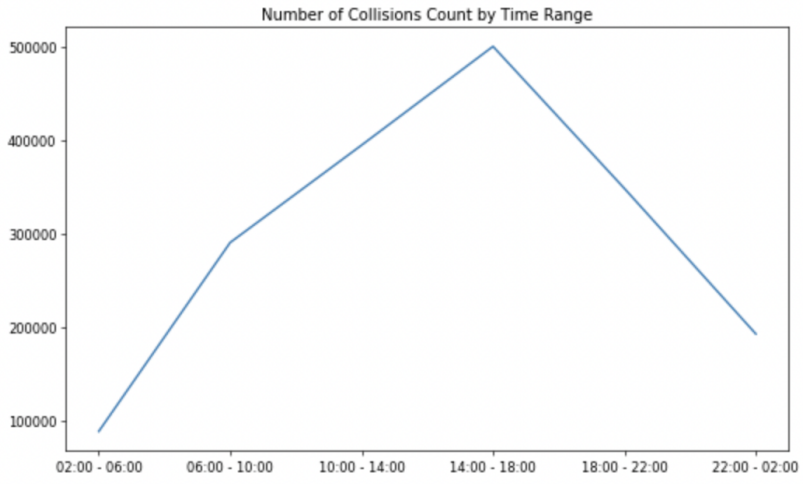

Fig. 2. Number of Collisions Count by Time Range

This figure shows the amount of data in relation to the time of the motor vehicle collisions. There are six temporal intervals, [2:00-6:00), [6:00-10:00),[10:00-14:00),[14:00-18:00),[18:00-22:00),[22:00-2:00). Accorting to the figure, we can find [14:00-18:00) has the most amount of motor vehicle collisions and [2:00-6:00) has the least one.

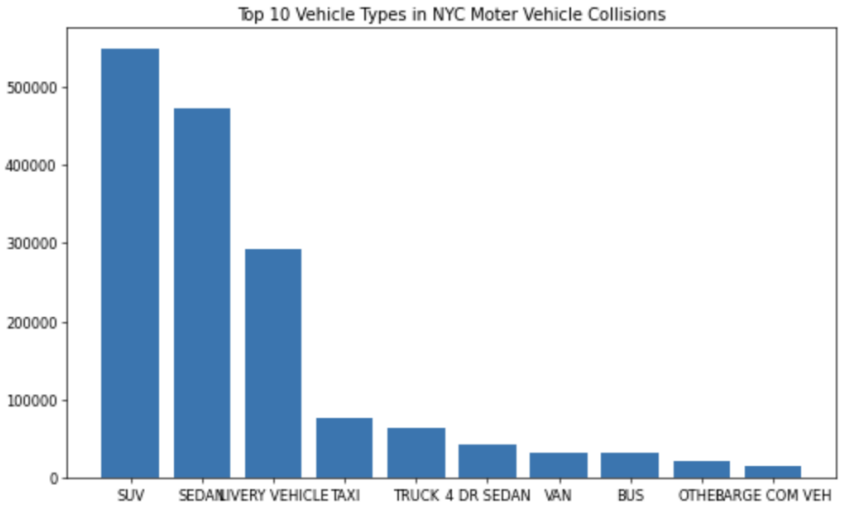

Fig. 3. Vehicle Types in NYC Moter Vehicle Collisions

This figure shows the amount of data in relation to the vehicle types of the motor vehicle collisions. Accorting to the figure, we can find SUV has the most amount of motor vehicle collisions. SEDAN and LIVERY VEHICLE followed. The three of them are head above the rest.

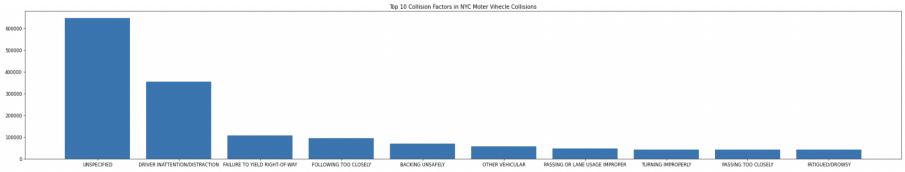

Fig. 4. Factor in NYC Moter Vehicle Collisions

This figure shows the amount of data in relation to the factor of the motor vehicle collisions. Except for UNSPECIFIED, DRIVER INATTENTION/DISTRACTION is is the most common cause of car accidents.

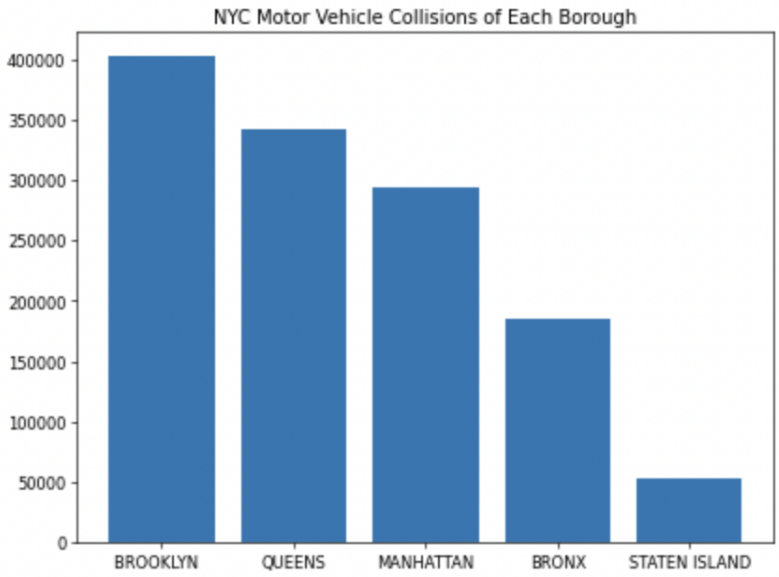

Fig. 5. NYC Moter Vehicle Collisions of each Borough

This figure shows the amount of data in relation to the borough of the motor vehicle collisions. Accorting to the figure, we can find BROOKLYN has the most amount of motor vehicle collisions and STATEN ISLAND has the least one.

5.2 Evaluation - Precision and Recall

Measuring the effectiveness and accuracy when it comes to cleaning the data is very important. Effectiveness and accuracy demonstrated how robust, strong the data cleaning methods are and it also showed the differences when data cleaning methods perform in different datasets. So with the Precision and Recall, it is a comprehensive performance metric to apply to data retrieved from collections.

The precision is for our prediction results, which indicates how many of the samples predicted to be positive are really positive samples. Then there are two possibilities to predict positive, one is to predict positive class as a positive class (TP) and the other is to predict negative class as a positive class (FP), that is $precision = \frac{TP}{TP+FP}$.

And recall is for our original sample, which indicates how many positive cases in the sample were predicted correctly. Then there are also two possibilities, one is to predict the original positive class as positive (TP), and the other is to predict the original positive class as negative (FN), that is $recall = \frac{TP}{TP+FN}$.

To calculate Precision and Recall to evaluation our project, we download 10 datasets with similar columns with NYC Motor Vehicle Collisions in NYC Open Data and Peel HDFS provided by the professor. Then apply the techniques we used before to these 10 datasets. For submission efficiency and feasibility, we only take the first 1000 rows of each data by Spark, After that, we measure the Precision and Recall. The details are as follows:

Dataset: NYPD Complaint Data Historic

Precision(Crash Time): 1.0

Recall (Crash Time): 1.0

Dataset: NYPD Arrest Data

Precision (Date): 1.0
Recall (Date): 1.0
Dataset: FDNY Fire house Listing
Precision (Borough): 0.6789
Recall (Borough): 0.8335
Dataset: Public Recycling Bins
Precision(Latitude): 0.9923
Recall(Latitude): 0.91123
Precision(Longitude): 0.9923
Recall(Longitude): 0.91123
Dataset: Housing New York Units by Building
Precision (Latitude): 0.7393
Recall (Latitude): 0.7090
Dataset: NYC Wi-Fi Hotspot location
Precision (Longitude): 0.8369
Recall (Longitude): 0.6572
Dataset: NYC Condom Availability Program - HIV condom distribution locations
Precision (Location):0.9830
Recall (Location):0.6403
Dataset: HIV Testing Locations
Precision (On Street Name): 0.6179
Recall (On Street Name): 0.7691
Dataset: 2015 Street Tree CensusTree Data
Precision(Address): 0.9923
Recall(Address): 0.8021

5.3 Method Improvement

5.3.1 program security. In the beginning, we only cleaned up for errors in the raw data, ignoring the data that was already correct. This behavior may create potential pitfalls. For example, all boroughs have the right type in the dataset of NYC Motor Vehicle Collisions, therefore, we didn't deal with it. So, there is very low precision in the dataset of FDNY Fire House Listing. After improvement, we simply check every column before dealing with them to make cleaning work more precise and safer.

5.3.2 outlier during cleaning. At the base of program security improvement, we clean the data easily and more efficiently. However, an outlier may also appear during the cleaning process. For example, when we query a borough based on latitude and longitude, the geopy API returns the wrong type of borough, such as "the Bronx." It should have been "BRONX". So, this will make the previous inspection ineffective. The solution is to check the correctness of the data again after cleaning to prevent errors in the processing.

5.4 Limitation

There are several limitations that we encountered during the data cleaning processes.

5.4.1 external API limitation. When we use longitude and latitude to double-check with the attributes like Borough, ZipCode, Addresses, through fast and accuracy interface, Google Map API. Later we found 1-2 million requests to Google API is an industrial level volume of data. So we also tried with Geopy, Nominatim package to calculate all those attributes, However, each item takes about 1 second to run. So actual running time for whole datasets is massive and takes an extremely long time to complete.

5.4.2 hard code limitation. In order to determine if the ZIPCODE is in New York, we manually added all New York zipcodes. However, this is flawed. We can only judge New York City, not generalize to other cities in the United States, or the world.

5.4.3 precision limitation. Even after optimization and enhancement, it still cannot reach very high precision.

REFERENCES

[1] Geopy.readthedocs.io. 2021. Welcome to GeoPy's documentation! — GeoPy 2.2.0 documentation. [online] Available at: <https://geopy.readthedocs.io/en/stable/> [Accessed 11 December 2021].

[2] Python, C. and Villa-Pi241;eros, M., 2021. Convert lat/lon to ZIP/Postal code using Python. [online] Geographic Information Systems Stack Exchange. Available at: <https://gis.stackexchange.com/questions/352961/convert-lat-lon-to-zip-postal-code-using-python> [Accessed 11 December 2021].

[3] Python, C. and Villa-Pi241;eros, M., 2021. Convert lat/lon to ZIP/Postal code using Python. [online] Geographic Information Systems Stack Exchange. Available at: <https://gis.stackexchange.com/questions/352961/convert-lat-lon-to-zip-postal-code-using-python> [Accessed 11 December 2021].

[4] Hellerstein, J., 2021. Quantitative Data Cleaning for Large Databases. [online] Dsf.berkeley.edu. Available at: <https://dsf.berkeley.edu/jmh/papers/cleaning-unece.pdf> [Accessed 11 December 2021].

[5] Hadoop.apache.org. 2021. Apache Hadoop. [online] Available at: <https://hadoop.apache.org/> [Accessed 11 December 2021].

[6] City of New York, N., 2021. NYC Open Data. [online] Opendata.cityofnewyork.us. Available at:<https://opendata.cityofnewyork.us/> [Accessed 12 December 2021].

[7] GitHub. 2021. GitHub - pandas-dev/pandas: Flexible and powerful data analysis / manipulation library for Python, providing labeled data structures similar to R data.frame objects, statistical functions, and much more. [online] Available at: <https://github.com/pandas-dev/pandas> [Accessed 11 December 2021].

[8] Chu, X., Ilyas, I., Krishnan, S. and Wang, J., 2016. Data Cleaning. Proceedings of the 2016 International Conference on Management of Data.

[9] S.Kulkarni, P. and W. Bakal, J., 2014. Hybrid Approaches for Data Cleaning in Data Warehouse. International Journal of Computer Applications, 88(18), pp.7-10.

[10] Greenwood-Nimmo, M. and Shields, K., 2017. An Introduction to Data Cleaning Using Internet Search Data. Australian Economic Review, 50(3), pp.363-372.

A APPENDIX

There is a github repository for this project: https://github.com/hd2327/bigdata-final-project. The relate document and simple description are as follows:

`data_reference : the reference data we commit to https://github.com/VIDA-NYU/reference-data-repository`

`data_analysis.ipynb : the data analysis file with Spark`

`data_cleaning_improve.ipynb : the data cleaning file after improvement`

`data_visualization.ipynb : the data visualization file with Spark and Matplotlib`

`report.pdf : the report`