Potholes in Boston

DATA SCIENCE PROJECT

Thibault Dody | Data Wrangling Report

https://github.com/tdody/Project-Boston-Potholes

TABLE OF CONTENTS

1 INTRODUCTION

The purpose of this report is to describe the methodology used to clean the data used for the analysis of the pothole repairs of the city of Boston.

2 DATA SOURCES

The datasets used for this project are the following:

1. The weather data of the city obtained from the National Centers for Environmental Information (NOAA). The dataset was available upon request on the agency website.
2. The neighborhoods population and size dataset was obtained from ZipAtlas[1].
3. The neighborhoods map file was downloaded from the Boston Open Data website[2].
4. The pothole dataset was obtained from the City of Boston website[3]

The folder containing the project file is located in the GitHub repository Springboard-Capstone-1[4] (It contains the following:

- 00_Data_Wrangling-Weather.ipynb
- 01_Data_Wrangling_Boston.ipynb
- 02a_Data_Wrangling_Potholes.ipynb
- 02b_Google_Geo_API_Fetcher.ipynb
- Folder Original Data
- Folder Intermediate Data
- Folder Cleaned Data

The following sections summarize the cleaning process used for each of the Jupyter Notebooks.

[1] http://zipatlas.com/us/ma/zip-code-comparison/population-density.htm
[2] http://bostonopendata-boston.opendata.arcgis.com/datasets/53ea466a189b4f43b3dfb7b38fa7f3b6_1
[3] https://data.cityofboston.gov/City-Services/Requests-for-Pothole-Repair/n65p-xaz7/data
[4] github.com/tdody/Springboard-Capstone-1

3 WEATHER DATA

3.1 Data structure

The weather data is imported directly as a data frame from the csv file. The data set contains the following features:

- *STATION*: (17 characters) is the station identification code.
- *NAME*: (max 50 characters) is the name of the station (usually city/airport name). This is an optional output field.
- *DATE*: is the year of the record (4 digits) followed by month (2 digits) and day (2 digits).
- *CDSD*: Cooling Degree Days (season-to-date). Running total of monthly cooling degree days through the end of the most recent month. Each month is summed to produce a season-to-date total. Season starts in January in Northern Hemisphere and July in Southern Hemisphere. Given in Celsius or Fahrenheit degrees depending on user specification.
- *CLDD*: Cooling Degree Days. Computed when daily average temperature is more than 65 degrees Fahrenheit/18.3 degrees Celsius. CDD = mean daily temperature - 65 degrees Fahrenheit/18.3 degrees Celsius. Each day is summed to produce a monthly/annual total. Annual totals are computed based on a January – December year in Northern Hemisphere and July – June year in Southern Hemisphere. Given in Celsius or Fahrenheit degrees depending on user specification.
- *DP01*: Number of days with >= 0.01 inch/0.254 millimeter in the month/year.
- *DP10*: Number of days with >= 1.00 inch/25.4 millimeters in the month/year
- *DSND*: Number of days with snow depth >= 1 inch/25 millimeters.
- *DSNW*: Number of days with snowfall >= 1 inch/25 millimeters.
- *DT00*: Number of days with maximum temperature <= 0 degrees Fahrenheit/-17.8 degrees Celsius.
- *DT32*: Number of days with minimum temperature <= 32 degrees Fahrenheit/0 degrees Celsius.
- *DX32*: Number of days with maximum temperature <= 32 degrees Fahrenheit/0 degrees Celsius.
- *DX70*: Number of days with maximum temperature >= 70 degrees Fahrenheit/21.1 degrees Celsius.
- *DX90*: Number of days with maximum temperature >= 90 degrees Fahrenheit/32.2 degrees Celsius.
- *EMNT*: Extreme minimum temperature for month/year. Lowest daily minimum temperature for the month/year. Given in Celsius or Fahrenheit depending on user specification.
- *EMSD*: Highest daily snow depth in the month/year. Given in inches or millimeters depending on user specification.
- *EMSN*: Highest daily snowfall in the month/year. Given in inches or millimeters depending on user specification
- *EMXP*: Highest daily total of precipitation in the month/year. Given in inches or millimeters depending on user specification.
- *EMXT*: Extreme maximum temperature for month/year. Highest daily maximum temperature for the month/year. Given in Celsius or Fahrenheit depending on user specification.
- *HDSD*: Heating Degree Days (season-to-date). Running total of monthly heating degree days through the end of the most recent month. Each month is summed to produce a season-to-date

total. Season starts in July in Northern Hemisphere and January in Southern Hemisphere. Given in Celsius or Fahrenheit degrees depending on user specification.

- *HTDD*: Heating Degree Days. Computed when daily average temperature is less than 65 degrees Fahrenheit/18.3 degrees Celsius. HDD = 65(F)/18.3(C) – mean daily temperature. Each day is summed to produce a monthly/annual total. A
- *PRCP*: Total Monthly/Annual Precipitation. Given in inches or millimeters depending on user specification.
- *SNOW*: Total Monthly/Annual Snowfall. Given in inches or millimeters depending on user specification
- *TAVG*: Average Monthly/Annual Temperature. Computed by adding the unrounded monthly/annual maximum and minimum temperatures and dividing by 2.
- *TMAX*: Monthly/Annual Maximum Temperature. Average of daily maximum temperature given in Celsius or Fahrenheit depending on user specification
- *TMIN*: Monthly/Annual Minimum Temperature. Average of daily minimum temperature given in Celsius or Fahrenheit depending on user specification

3.2 Data cleaning

The data set contains 221 records and 26 features. Since the choice was made to use only the data from one weather station, the features *STATION* and *NAME* are removed from the data frame since they only contain the same value for each record. Moreover, the features *DSND* and *EMSD* do not contain data for the entire time period. For some unknown reasons, these parameters were no longer monitored by the station. Therefore, they were removed from the data set. The data is presented as a daily record, since the set contains only one record per day, the *DATE* feature was chosen as the index of the data frame. This will facilitate the manipulation of the set.

3.3 Data exploration and validation

After cleaning the data set, it was crucial to explore the data and asses its quality. Using different plotting technics, the following plots were produced:

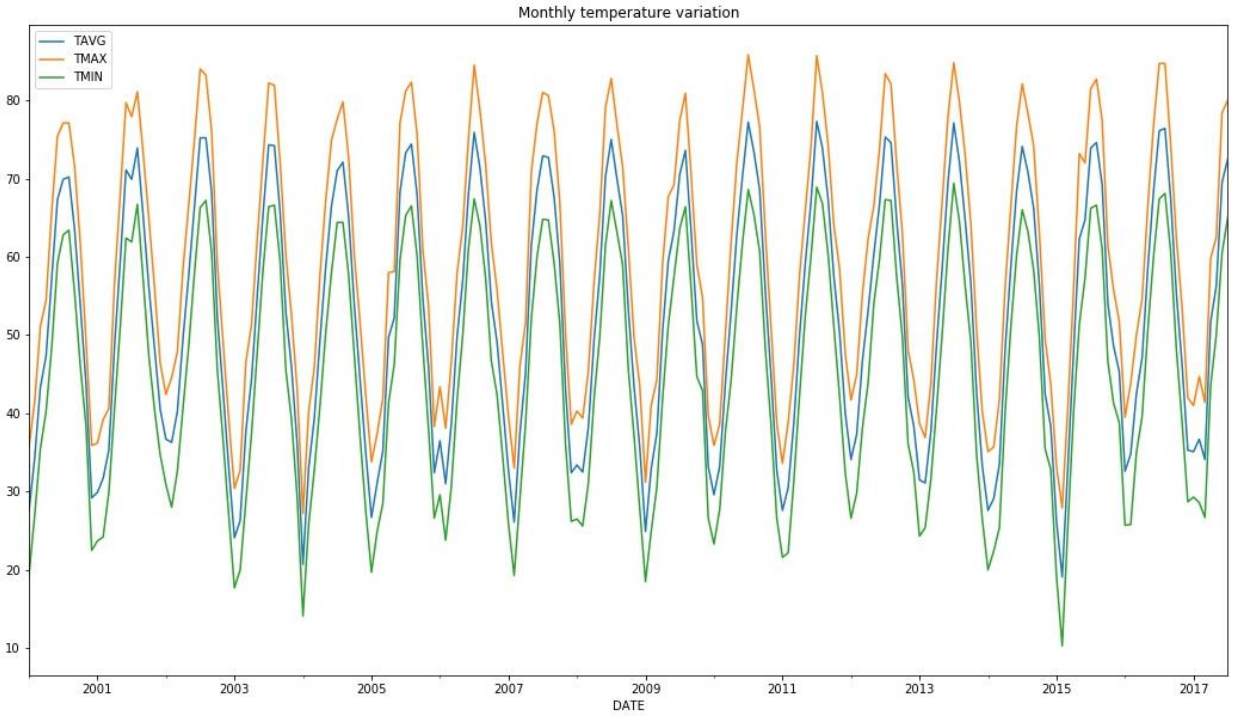

Figure 3-1: Monthly temperature variation

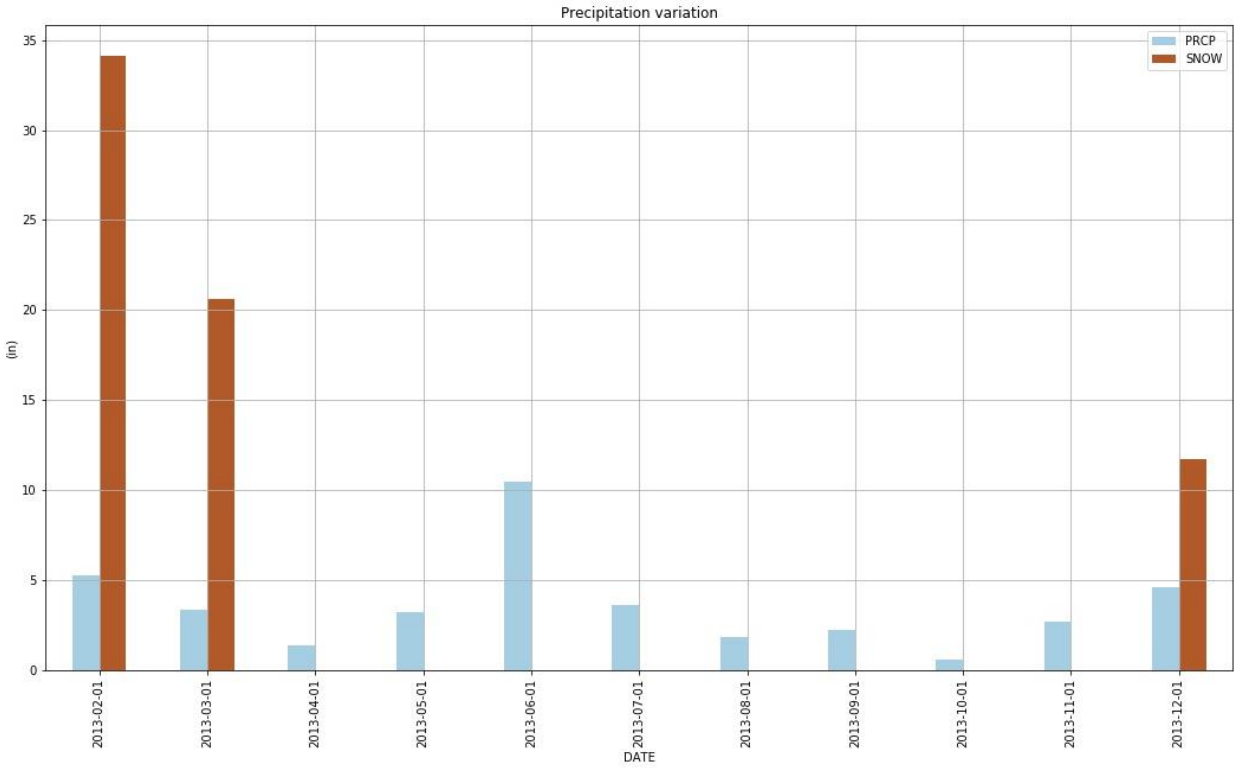

Figure 3-2: Precipitation variation in 2013

The temperature variation and the precipitation distribution make sense. This relatively cleaned data set did not require too much effort to be cleaned.

4 ZIP CODE DATA

4.1 Data structure

The neighborhood data is imported directly as a data frame from the csv file. The data set contains the following features:

- *#*: Index
- *Zip Code*: Neighborhood zip code
- *Location*: Neighborhood longitude and latitude
- *City*: Neighborhood city
- *Population*: Neighborhood population
- *People / Sq. Mile*: Neighborhood density
- *National Rank*: Density national rank
- *Unnamed: 7*: Blank column
- *Unnamed: 8*: Blank column
- *Unnamed: 9*: Blank column
- *Unnamed: 10*: Blank column

4.2 Data cleaning

The data set contains 31 records and 24 features. The last four features (*Unnamed 7* to *Unnamed 10*) do not contain anything, they are removed from the set. In order to be able to merge our various data frames, we rename the features of this data frame to merge the feature names of the pothole dataset.

4.3 New feature

The *Latitude* and *Longitude* features are created by extracting their values from the *Location* feature using pandas extract method. The *Location* feature is then removed from the set. Moreover, for clarity of the plots, the dataset is sorted by zip codes and re-indexed.
Since the dataset includes the population density and the population of each neighborhoods, the neighborhood area is computed by dividing the population by the density of the area. For convenience, the results are converted in acres.

4.4 Data exploration and validation

The main aspect of the data exploration and validation is first to verify that we have all the Boston neighborhoods contain in the set. Using the folium[5] package, we plot the centers of the neighborhoods (Latitude and Longitude) on top of the neighborhoods map. The map is obtained as a reshape file (converted into a JSON file using mapshaper[6] website). The result is provided below. When inspecting the map, we notice two important ideas:

- First the neighborhood dataset does not contain the small zip codes used to “square” the city boundary. These were created along the year as the city’s boundaries were adjusted. This is not an issue for us as these “ghost” zip codes do not contain any (or only a few) house.

[5] https://folium.readthedocs.io/en/latest/
[6] http://mapshaper.org/

- Moreover, Boston is not a common city when it comes to the shape of its neighborhoods. Indeed, several blocks located downtown (for instance the city hall) possess their own zip codes. These are not included in the dataset but do not alter the quality of the data and our analysis.
- Finally, the neighborhood Chestnut Hill (West-most on the map below) does not entirely belong to the city. Therefore, the population and neighborhood area that we posse applies to the entire neighborhood while the pothole data might only cover the area under Boston's jurisdiction.

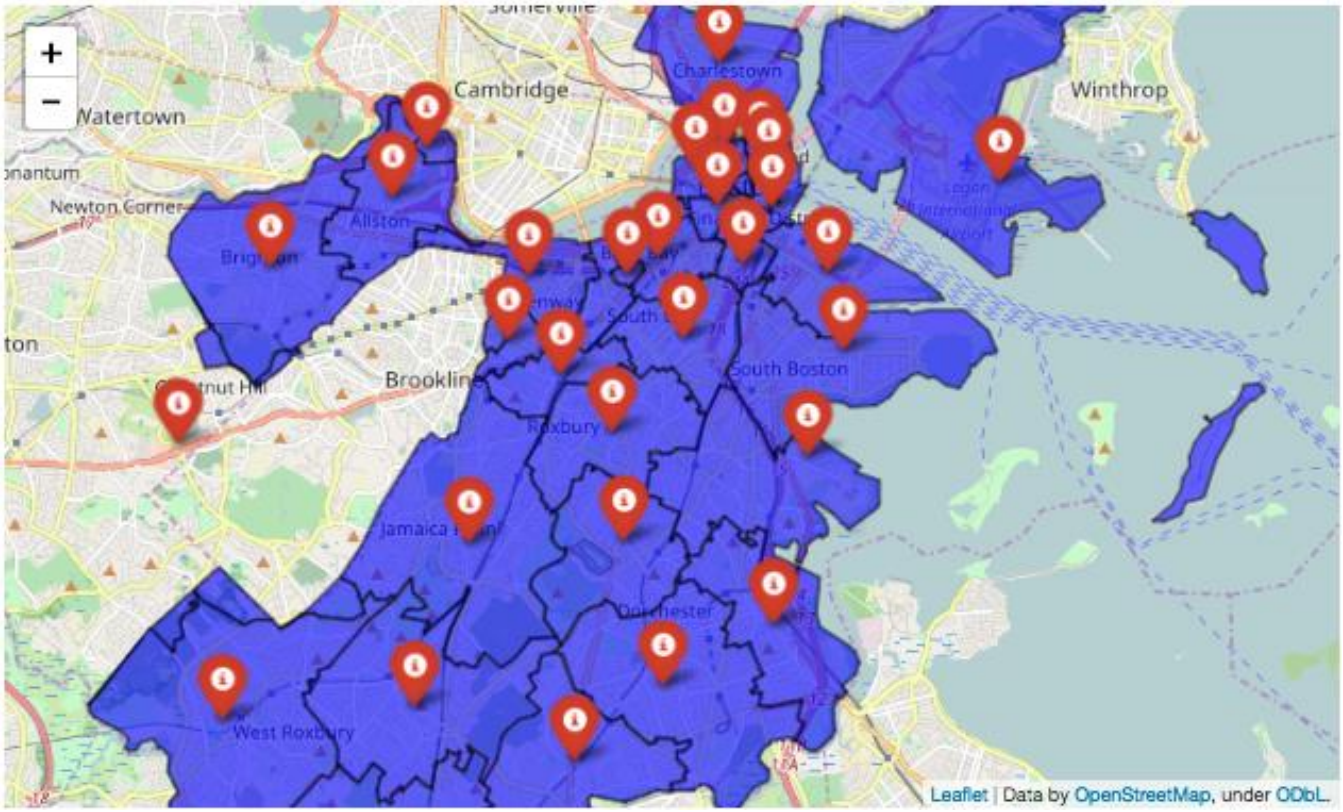

Figure 4-1: Boston zip codes and neighborhood centers

The following plots are created to assess the quality of the data by comparing the results per zip codes.

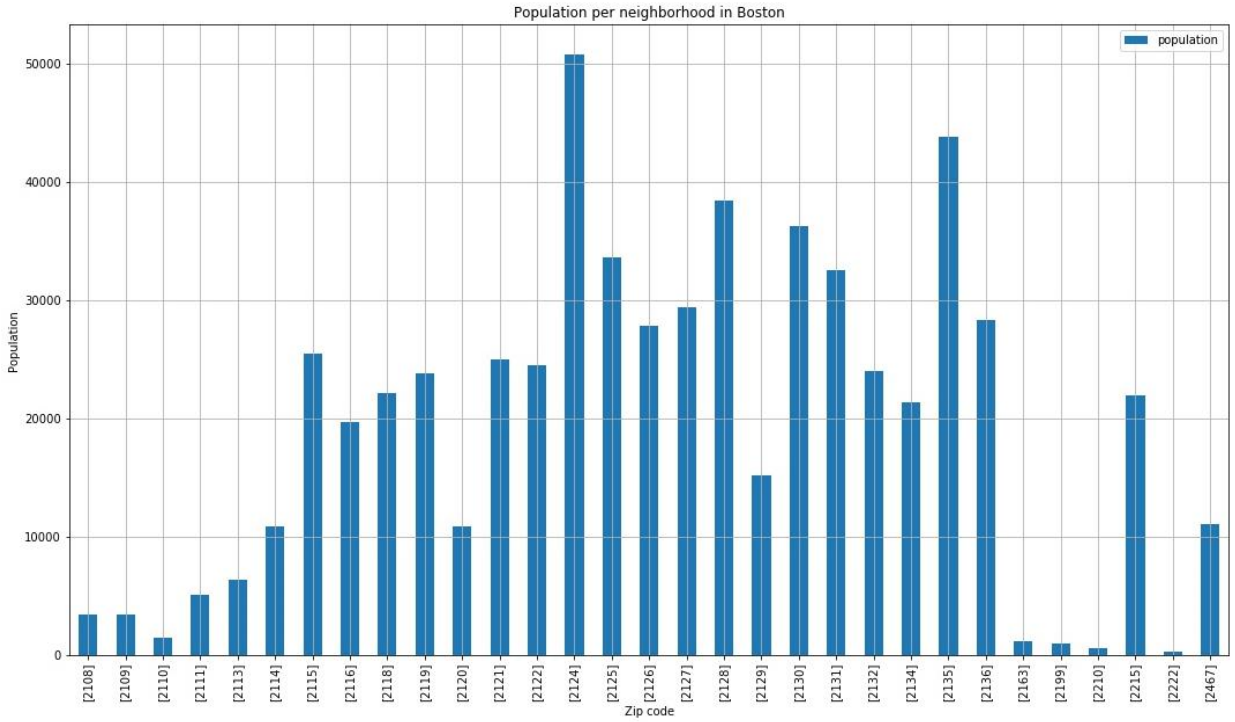

Figure 4-2: Population per neighborhood in Boston

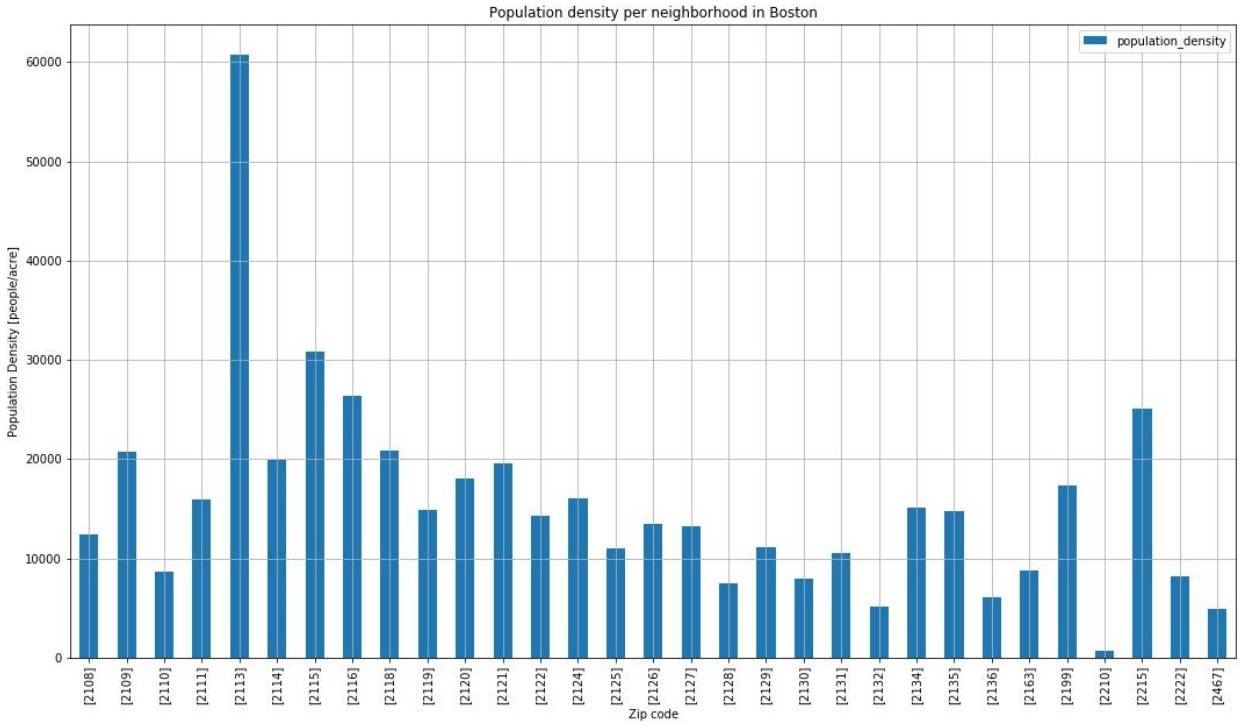

Figure 4-3: Population density per neighborhood in Boston

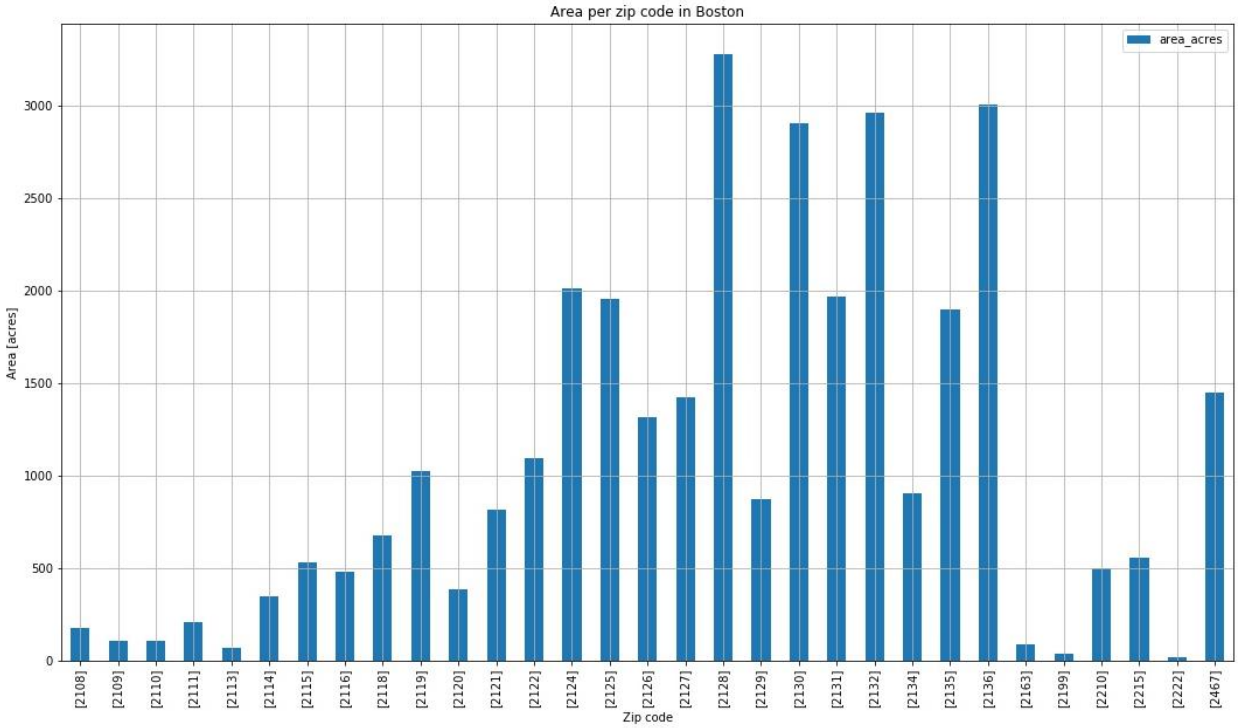

Figure 4-4: Area per zip code in Boston

5 POTHOLE DATA

5.1 Data structure

The weather data is imported directly as a data frame from the csv file. The data set contains the following features:

- *CASE_ENQUIRY_ID*: Case number assigned to the request to repair the pothole.
- *OPEN_DT*: Date and time of the repair request.
- *TARGET_DT*: Scheduled time for repair.
- *CLOSED_DT*: Date and time the case was closed.
- *OnTime_Status*: ONTIME if *CLOSED_DT*>TARGET_DT
- *CASE_STATUS*: Case status.
- *CLOSURE_REASON*: Reason for the case closure.
- *CASE_TITLE*: Request type. In this case, the type is "Request for Pothole Repair".
- *SUBJECT*: The city department in charge of the request.
- *REASON*: Reason for the case opening.
- *TYPE*: Specific reason for the case opening. In this case, the type is "Request for Pothole Repair".
- *QUEUE*: Code corresponding to the department per neighborhood in charge of the repair.
- *Department*: Code corresponding to the department in charge of the repair.
- *SubmittedPhoto*: URL of the photo taken to support the claim.
- *ClosedPhoto*: URL of the photo taken to support the repair.
- *Location*: Address of the pothole.
- *fire_district*: Fire district corresponding to the pothole location.
- *pwd_district*: Public Work district corresponding to the pothole location.
- *city_council_district*: City Council district corresponding to the pothole location.
- *police_district*: Police district corresponding to the pothole location.
- *neighborhood*: Neighborhood corresponding to the pothole location.
- *neighborhood_services_district*: Neighborhood Services district corresponding to the pothole location.
- *ward*: Ward corresponding to the pothole location.
- *precinct*: Precinct corresponding to the pothole location.
- *land_usage*: Blank column.
- *LOCATION_STREET_NAME*: Street number and street name corresponding to the pothole location.
- *LOCATION_ZIPCODE*: Zip code corresponding to the pothole location.
- *Property_Type*: Blank column.
- *Property_ID*: Blank column.
- *LATITUDE*: Latitude of the pothole location.
- *LONGITUDE*: Longitude of the pothole location.
- *Source*: Source of the request.
- *Geocoded_Location*: Blank column

5.2 Data cleaning

5.2.1 Empty features

After a dive into the data frame structure, several columns were deleted. The features *land_features, Property_Type, Property_ID*, and *Geocoded_location* do not contain any data and are therefore removed from the data frame.
We ran an analysis function on the data set that returns the number of unique values per feature. This tool lead to the deletion of the *SUBJECT, REASON, TYPE,* and *Department* features since they only contain a single unique value for the entire column.

5.2.2 Claim text boxes

One of the main challenges when dealing with user input text is the lack of consistency. In our case, the claim form contains a text box than can be filled by the employee in charge of the repair. Unfortunately, the conclusion of the claim is contained within this feature. For instance, if the pothole was repaired, the box will contain “Case Resolved”. However, if for any reason the pothole could not be repaired, the case is closed by filling the same text box. Because of the large variety of reason and the different writing styles, this field contains everything and anything from “wrong address” to the most unexpected: “Case Closed Case Noted no pot hole shows picture of coffee mug.” Using Excel for convenience, the *CLOSURE_*REASON column was inspected to identify patterns amongst all the invalid case. The following list was used as a filter:

- *CLOSURE_REASON* contains the string "Case Resolved" => Case is acceptable
- *CLOSURE_REASON* contains the word "duplicate" => Case to be removed. (typos include duplcate, duplicte)
- *CLOSURE_REASON* contains the word "invalid" => Case to be removed.
- *CLOSURE_REASON* contains the words "better location" => Case to be removed.
- *CLOSURE_REASON* contains the words "please contact" => Case to be removed.
- *CLOSURE_REASON* contains the words "please call" => Case to be removed.
- *CLOSURE_REASON* contains the word "test" => Case to be removed.
- *CLOSURE_REASON* contains the words "could not find" => Case to be removed.
- *CLOSURE_REASON* contains the word "cannot" => Case to be removed.
- *CLOSURE_REASON* contains the word "private" => Case to be removed. (versions include prvt)
- *CLOSURE_REASON* contains the word "wrong" => Case to be removed.
- *CLOSURE_REASON* contains the word "nothing" => Case to be removed.
- *CLOSURE_REASON* contains the word "re-subnmit" => Case to be removed. (versions include resubmit)
- *CLOSURE_REASON* contains the words "no pot hole" => Case to be removed. (versions include no potholes, no sink hole)

5.2.3 Ward names

Another inconsistency in the input format appears in the *ward* feature. Indeed, some of them are provided using a simple number, others are given using the string “Ward X”. In order to homogenize the data, this feature was converted into a integer by trimming the unnecessary letters.

Once this transformation performed, we looked at the missing entries for the ward feature. In order to retrieve the ward number from a record, we needed at least the location of the pothole. For this reason, all records without location and ward were simply removed from the dataset.

When inspecting the range of values taken by the ward column, we noticed that a missing/inaccurate location was sent to the ward 0. Indeed, all the potholes that could not be located were given the following properties:

- Ward: 0
- Latitude: 42.3594
- Longitude: -71.0587

Later on, the Google Geo-API will be used. But because of the utilization limit, it was chosen to manually locate the addresses corresponding to the Ward 0 using Google Maps and replace their coordinates and zip in the data frame. This subset consisted of eight unique addresses. The retrieved data was merged to the main data frame.

5.2.4 Missing closed date

Since the purpose of this study is to assess trends in the pothole reparations, we need to work with cases that were closed. Therefore, any record without a *CLOSED_DT* is removed from the set.

5.2.5 Default locations

After inspection, a large number of cases were assigned to the default location presented above. A mapping by zip code was first suggested, however because of the relatively messy correlation between neighborhood designation and zip codes, a one-to-one match could only be found for two zip codes which would apply to three cases only. To save time, these cases were not fixed with this method.

Since this subset is too large to be dropped, we decided to come up with a location fetcher algorithm. The first step consisted on a feasibility study, to do so, we looked at the record in the subset and defined if whether or not the available data was sufficient to geo-localize the addresses. After a first glance, the data was judged good enough to move on to the next step of the retrieval process. The subset of the potholes without zip code and assigned to the default location (See previous) was extracted and saved as a csv file. The reason behind this choice is that the limitation of the API we decided to use would not work well if integrated in the main note book. Therefore, we created a specific dedicated to the address search. The Google Geo-API was used, it is based on the algorithm behind Google Maps. Our algorithm will go through the csv file and restart to where it previously stopped, it will extract a missing location, send the request to the API and try to retrieve the zip code, latitude, and longitude corresponding to the location description. When creating our API request function, we had to consider two limitations. First, the API has a search limit of 2500 requests per day (we had roughly 4500 locations to search) and the API limits the frequency of the searches. This last constraint forced us to use a time delay of 0.2s (value based on trial and error) between each iteration in the main loop. Once a location has been retrieved, it is stored in an output csv file.

Moreover, they were two groups of addresses that could potentially lead to issues. Addresses without specific address number were assign a 0, others were assigned a address number range. For the first

group, we drop the zero and search for the street name alone (by default, the API locates the address corresponding to the middle of the street). For the second group, we decided for simplicity to keep the second street number (12-15 First Street becomes 15 First Street).

Once this process was completed, several locations were still missing, out of solution to retrieve their location, we decided to drop them from the set. It was determined after inspection that these locations were not retrieved because the zip code did not match the street name. Finally, the retrieved locations were merged with the main data frame.

5.2.6 Precinct feature

Since the precinct map is regularly updated, it was decided not to use this feature in the final analysis. The *precinct* column is removed from the set.

5.3 New features

5.3.1 Photos

The original data set contained the *SubmittedPhoto* and *ClosedPhoto* features. Since each feature contained the URL of the pictures taken to support the claim or the picture of the repair, we judge important to keep these features for the analysis. After all, a claim with picture might for a quicker repair… However, the provided URL addresses were not really useful as is. In order to be integrated in the analysis, both of them were converted into the Boolean features *SubmittedPhoto_Bool* and *ClosedPhoto_Bool.* If a record contains an URL, its corresponding Boolean feature is set to True.

5.3.2 Time to repair

By subtracting the claim closed date and the claim creation date, we create a new feature *time_repair*. When examining the results, we found that some of the results were negative or extremely large. We decided to remove record with negative repair time or repair time greater that six months. Indeed, with constant road constructions going on in the city and the harsh winters, it is possible for a pothole to be inaccessible for six months. For larger repair time, these values are either extremely unlikely or simply wrong.

5.3.3 Intersections

During the location retrieval process, we noticed that many pothole cases were listed on road intersection. In order to assess whether or not the potholes develop more often on intersection, a new feature was added to the set: *is_intersection*. If a pothole location contains the word “intersection”, then a True value is assigned to the record.

5.4 Chestnut Hill

The Chestnut Hill neighborhood is located west of the center of Boston. The particularity of this region is to be divided into two fragments, one that falls under the jurisdiction of the city of Boston, the other part is independent. In order to understand to what fragment of the zip code 02467 our data belongs to, we had to plot the potholes cases marked as 02467 on top of the boundaries of the city of Boston. The figure below shows this plot.

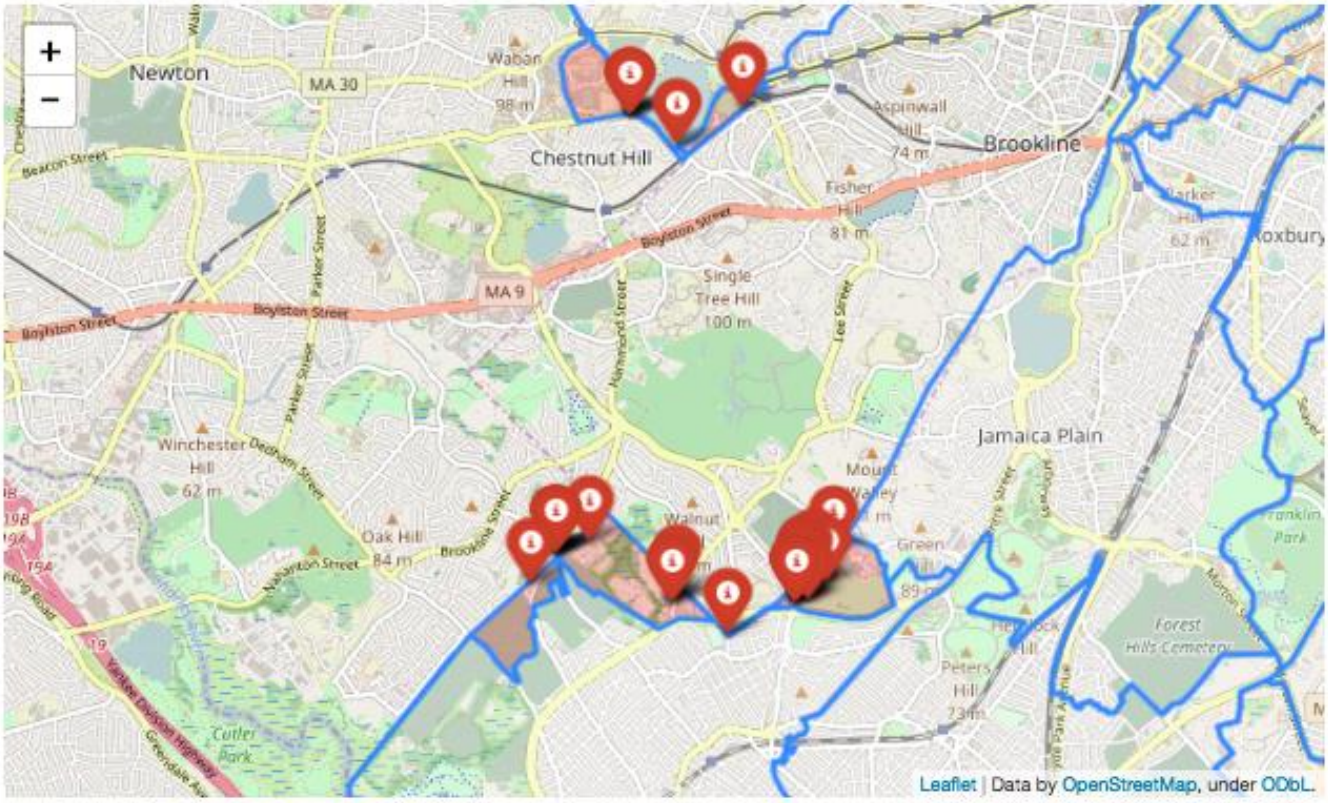

Figure 5-1: Chestnut Hill claims

The features of the neighborhood dataset apply for the entire 02467 zip code. We need to scale them down so they can be used later. The first step consists of extracting the sum of the area of the polygons associated to the zip code 02467 in the JSON file. This is done through a simple for loop. In order to scale the population, we need to make the assumption that the population of Chestnut Hill is homogenously distributed over the area. By making this assumption, we assume that the population density is constant, we can now multiply the population density by the new area to obtain the population of Chestnut Hill that is located under the jurisdiction of the city of Boston.

6 Conclusion

Finally, the data frame is converted back into a csv file that can be later imported to perform the rest of the analysis.

Before we move on to the analysis of the extracted data it is important to mention the followings:

1. Overall, a lot of records were dropped because of the quality of the initial data. The 311-claim platform contains a lot of text boxes and parameters that are not automatically filled. A first advice to the city of Boston is to move to a smarter and more systematic platform. For instance, the closure of a case should not be defined only by an operator text input. It should be done through a dropdown menu (“Pothole repair”, “Wrong location”, “No location”,…) then the user

could add comment inside a text box but the critical content of the data is made homogenous throughout the entire claim platform.

2. The location selection seems to be an issue. Again, as the user is responsible to type the address, this can lead to unusable input. An interactive map could be a good solution, the user could search an address and point to the exact location of the pothole by dropping a marker on a map.
3. Finally, a lot of case are entered with the same open and closing date and time. This is because the platform is used by both the city workers and the inhabitants of the city. When city workers notice a pothole that has not been signaled, they make the repair and later log the repair in the system. While this will not affect our analysis when it comes to pothole distribution and frequency of road damage, it will need to be treated before our model can be used with a machine learning algorithm.