Profiling NYC OpenData

Kartikeya Shukla*
ks5173@nyu.edu
New York University
Brooklyn, NY

Cheng Shi*
cs5615@nyu.edu
New York University
Brooklyn, NY

Xue Xia*
xx859@nyu.edu
New York University
Brooklyn, NY

ABSTRACT

This paper reviews and presents distributive preprocessing methods for profiling over 1900 datasets available on NYC Open Data. The analysis was performed using Spark, SQL, Pandas and Plotly. Additionally, the text explores, as to how to check for multiple generic/semantic types within a single column. This paper focuses on how to use Spark in combination with similarity measures and regular expressions to perform Generic/Semantic profiling. Furthermore, the paper represents a detailed analysis of NYC 311 complaints with inferences that can be used by the NY's government.

KEYWORDS

generic profiling, semantic profiling, levenshtein distance, similarity, regular expressions

ACM Reference Format:
Kartikeya Shukla, Cheng Shi, and Xue Xia. 2019. Profiling NYC OpenData. In *Course Project, Big Data at NYU Tandon September 03 – December 11, 2019, Brooklyn, NY*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The massive growth in the scale of data has been observed in recent years being a key factor of the Big Data scenario. Big Data can be defined as high volume, velocity and variety of data that require a new high-performance processing. Addressing big data is a challenging and time-demanding task that requires a large computational infrastructure to ensure successful data processing and analysis. Once such data lake/warehouse is NYC OpenData, which contains over 1900 datasets generated by various New York City agencies and other City organizations available for public use. As part of an initiative to improve the accessibility, transparency, and accountability of City government. This text attempts to profile NYC OpenData to determine any potential quality issues. And to generate meta-data to enable NYC agencies with data lineage and provenance. Initially, one starts with generic profiling, and then delve deeper for the semantics.

Big Data Course Project, Tandon School of Engineering, Brooklyn, NY
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

2 TASK 1: GENERIC PROFILING

Generic profiling only involved checking "multiple-generic" types (i.e. integer, float, string, date-time) for each column in each of those 1900 datasets available. To accomplish this, Spark was used.

Approach:

(1) We went over all the datasets in an iterative manner.
(2) Only one column of the dataset in a particular iteration was kept in-memory (i.e. cache)
(3) Using this column, 4 other columns were created–called Int-Column, String-Column, Date-Column, Float-Column. Each of these contained the values of the given type from the column being processed, and the rest None. For instance, if a column had length 10, with 3 int types, 2 string types and 5 date type values. Then Int-Column will be length 10, with 3 int values and rest None, similarly Date-Column will contain 5 date values and rest None and so on and so forth.

Potential Issues Found:

(1) Numerous columns that contained integer type data, were actually stored as strings. For instance, a Column called **Amount** contained values such as '123', '4.69'.
(2) We accounted for these types in our generic checking –as in '123' went in the String-Column and the Int-Column as well as 123. We casted back forth accordingly to detect these issues for: String-Float type, String-Integer type and String-Date type dual conversions.

Tradeoffs/Limitations:

(1) The potential drawback of our approach was that– since one was checking for multiple types in a single column. Due to this our computational time went up significantly. For large datasets –our computational complexity was exponential as compared to using df.dtypes method in Spark (which just assigns a single type to the column under the hood)

3 TASK 2: SEMANTIC PROFILING

Thereafter, we delved deeper to extract more detailed information about the semantics of columns, and work for a subset of the *NYCOpenData* called *NYCColumns*. For each column, different semantic types were identified and summarized.

We looked for the following semantic types given below using the either/all of the following methods.

3.1 Methods

Regular Expressions (Regex) | Scraped a global list from google and other outlets (Global-list) | levenshtein distance (i.e. edit

distance) using python's fuzzy matching library(i.e. Fuzzy-Wuzzy) . Thus we accounted for minor spelling mistakes as well, for **fuzzy-matching (FM)** we set our threshold ratio = 80, for it be classified as a particular semantic type.

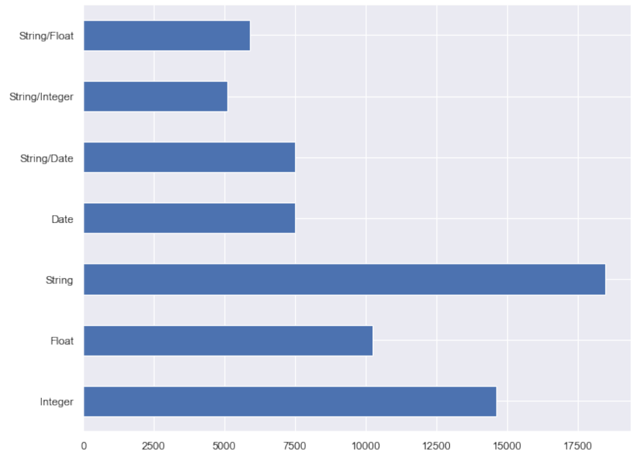


Figure 1: The bar plot, shows count of the no of occurrences in columns for each type. String/Integer, Float/Integer, Date/String represent count of some common dual type columns

- `Person name`: Regex, cannot contain number though
- `Business name`: Regex
- `Phone Number`: Regex
- `Address`: String matching, Regex, Global-list, FM
- `Street name`: String matching, Regex, Global-list, FM
- `City`: String matching, Regex, Global-list, FM
- `Neighborhood`: String matching, Regex, Global-list, FM
- `LAT/LON coordinates`: Regex
- `Zip code`: Regex
- `Borough`: String matching, Regex, Global-list, FM
- `School name`: String matching, Regex, Global-list, FM
- `Color`: String matching, Regex, Global-list, FM
- `Car make`: String matching, Regex, Global-list, FM
- `City agency`: String matching, Regex, Global-list, FM
- `Areas of study`: String matching, Regex, Global-list, FM
- `School Levels`: String matching, Regex, Global-list, FM
- `College/University names`: String matching, Regex, Global-list, FM
- `Websites`: String matching, Regex, Global-list, FM
- `Building Classification`: String matching, Regex, Global-list, FM
- `Vehicle Type`: String matching, Regex, Global-list, FM
- `Type of location`: String matching, Regex, Global-list, FM
- `Parks/Playgrounds`: String matching, Regex, Global-list, FM

3.2 Detailed overview of methods used:

3.2.1 Regular Expression. for these labels ***'phone number', 'lat lon cord', 'zip code', 'website', 'business name'***, we use regular expression to match the format of each label.

- Benefits: *1).Mature, well tested technology. If you think your problem can be solved with a regex pattern, please use it. Don't try to reinvent the wheel and create your own text parser.2).Powerful tool. With one line of code you can create amazing searches.3).Available in most programming languages. 4)There are many online regex tools available, where you can quickly test and fix your patterns. These online tools simplify a lot the debugging and testing of regex expressions. 5).Regular Expression allows code to be a lot more manageable in many cases.*
- Limitations: *1)Depending on the situation, the same meta character has many different meanings, which makes reading a regex a complicated task. 2).Regex expressions could have bad performance in some instances. Unbounded repetitions can match a string in many different ways, and regex engines usually need to do many steps and backtracking to find all of these matches. 3).Regular Expressions are not suited for very complex, recursive data formats, like XML or HTML. In these cases, it's better to use an XML parser. 4).There are many different regex engines, and each one has different syntax. Therefore depending on the language, you need to learn some particular flags and meta characters. 5).Poor readability: There will be some difficulty when we try to understand the format of the matched string according to the regular expression.*

3.2.2 Dictionary. for these labels: ***'vehicle-type', 'location-type', 'zip-code', 'park/playground', 'college/university-name', 'city-agency', 'area of study', 'subject in school', 'school-name', 'car-make', 'school-level', 'borough', 'neighborhood', 'address', 'city', 'color'***. we create different dictionary for each label.Then for each value in column, check if the value exists in the dictionary. If exists, which means this value has the given label, vice versa, this value does not have this label.

- Benefits: *1). Fastness: For the type of matching using a dictionary, we can quickly create the dictionary of given label by referring to the existing dictionary. Examples include names of people, names of surrounding areas, and car manufacturers. 2).Easy to optimize: By correctly expanding the contents of the dictionary, you can effectively improve the correctness of the match.*
- Limitations: *1) Not easy to manage: When there is too much content, we need to ensure that the dictionary content is accurate and effective, and we cannot add wrong content.to the designated dictionary, otherwise the accuracy will be reduced. 2) Matching efficiency: The length of the dictionary is too long, which will slow down the matching efficiency.*

3.2.3 FuzzyWuzzy Lib. FuzzyWuzzy is a library of Python which is used for string matching.for labels using Dictionary, in addition to improve the accuracy of our algorithm, we import FuzzyWuzzy Library. It is one of the very easiest method to compare string in Python and we can have a score out of 100, which denotes two string are equal by giving similarity index. Fuzzy string matching is the process of finding strings that match a given pattern. Basically it uses

Levenshtein Distance to calculate the differences between sequences. FuzzyWuzzy has been developed and open-sourced by SeatGeek, a service to find sport and concert tickets.

- Benefits: *1) Easy to use: This is a simple and easy to use fuzzy string matching toolkit, which is very simple to use. Requirements of fuzzywuzzy: Python 2.4 or higher, python-Levenshtein. Installation is convenient, and getting started quickly. 2) The Levenshtein Distance algorithm is used to calculate the difference between the two sequences and this algorithm performs well. 3) Documentation: easy to query APIs, there are ready-made documents for querying APIs.*
- Limitations: *Because the existing library is used to match the label, there is still a limit to the matching performance of a particular label when using the library's method.*

3.3 Limitations/Assumptions/Tradeoffs

(1) **Assumptions:** To differentiate between Street Name and address, we assumed addresses start with a number while street names do not. Similarly person names cannot contain a number. We only accounted for 10 digit phone numbers, thus 911 will be false negative. For wherever we've used global lists, there, you'll get a false negative, if there is no partial match with any element in the list.

(2) **Limitations** One of the major caveats of this approach was to handle **collisions**. One cannot come up with an "exact" regex, string pattern to differentiate between business names and building classification/parks/colleges etc. Thus we decided on a "precedence" for each semantic type. That is–if a value classifies as both business name and building classification–then building classification will have higher precedence. To see the full order of precedence, please go through the jupyter notebook, available on our github repository called– Task-2 (Semantic-Profiling).ipynb

(3) **Tradeoffs:** One of the best ways to avoid collisions was to process the column name for some kind of "heuristic", as in if the column name contained the string 'Location' then we might check for addresss, street name, lat/long coordinates, location type first (i.e. higher "precedence"). Or if the column name contains the string 'classification', one might give higher precedence to building classification. But using this approach would have resulted in limited "multiple-semantic" type checking. That is we'd not be able to **comprehensively** cover all semantic types for a particular column, although this approach would've handled collisions pretty well. Since our focus, was checking for multiple semantic types **exhaustively**, we decided not to preprocess the column names for any additional heuristic/info. Our algorithm/strategies don't make any kind of assumptions about the column using the column name. Thus there are always tradeoffs.

3.4 Link of code

A link to the github repository where your code for task2 resides. https://github.com/kart2k15/Big-Data--Bits_please/blob/master/Task-2%20(Semantic-Profiling).ipynb

3.5 Calculation of Precision and Recall

This part include the calculation and visualization of precision and recall for the our strategies.

3.5.1 ***Manual Labeling.*** We check every file of cluster1 in NYUColumns and get labels for each column.Based on the true type of each column, Note that a given column may have values of different types, therefore, its true type may consist of multiple labels, we use a list to store lables for each column. For each file in cluster1, we manually divided the labels of each column and saved it in the dictionary format. The format example is as follows: *'735p-zed8.EMPCITY': ['city','borough']*. And thus created a dictionary/json called manual-labels

3.5.2 ***Link for task2-manual-labels.json.*** Here is the link for the file we created using manual labeling dictionary:https://github.com/kart2k15/Big-Data--Bits_please/blob/master/devise_final

3.5.3 ***Calculation.***

(1) **precision** = number of columns correctly predicted as type / all columns predicted as type

(2) **recall** = number of columns correctly predicted as type / number of actual columns of type

- **Method:** *Using the manual-label dictionary, we created a dictionary called actual-as-type, which contained keys as the semantic types and values as the columns that were labeled as that type. Then using our JSONS generated from Task2, we pre-processed them and created dictionary called pred-as-type, which had keys as the semantic types and values as the the no. of times our algorithm/strategy predicted that semantic type for some column. Finally we aggregated across both our jsons from Task2 and the manual-label dictionary to create another dictionary called correct-pred which contained key as the semantic types and values as the count of the columns that were predicted correctly for that semantic type. Thus we finally have all the components for calculating precision/recall for each semantic type* ***x*** *using the equations below*

$$\textbf{precision = correct-pred[x]/pred-as-type[x]} \tag{1}$$

$$\textbf{recall = correct-pred[x]/actual-as-type[x]} \tag{2}$$

Precision/Recall Results

(1) Precision for type person-name = 0.183
(2) Recall for type person-name = 1.0
(3) Precision for type business-name = 0.276
(4) Recall for type business-name = 0.889
(5) Precision for type phone-number = 0.8
(6) Recall for type phone-number = 0.64
(7) Precision for type address = 0.689
(8) Recall for type address = 0.861
(9) Precision for type street-name = 0.283
(10) Recall for type street-name = 0.933
(11) Precision for type city = 0.611
(12) Recall for type city = 0.846
(13) Precision for type neighborhood = 0.56

(14) Recall for type neighborhood = 1.0
(15) Precision for type lat-lon-cord = 0.258
(16) Recall for type lat-lon-cord = 0.8
(17) Precision for type zip-code = 0.714
(18) Recall for type zip-code = 0.667
(19) Precision for type borough = 0.355
(20) Recall for type borough = 0.647
(21) Precision for type school-name = 1.0
(22) Recall for type school-name = 0.647
(23) Precision for type color = 0.571
(24) Recall for type color = 1.0
(25) Precision for type car-make = 0.652
(26) Recall for type car-make = 0.882
(27) Precision for type city-agency = 0.5
(28) Recall for type city-agency = 0.167
(29) Precision for type area-of-study = 0.667
(30) Recall for type area-of-study = 0.631
(31) Precision for type subject-in-school = 0.92
(32) Recall for type subject-in-school = 0.884
(33) Precision for type school-level = 0.933
(34) Recall for type school-level = 1.0
(35) Precision for type college-name = 0.0
(36) Recall for type college-name = 0
(37) Precision for type website = 0.5
(38) Recall for type website = 1.0
(39) Precision for type building-classification = 0.270
(40) Recall for type building-classification = 1.0
(41) Precision for type vehicle-type = 0.5
(42) Recall for type vehicle-type = 1.0
(43) Precision for type location-type = 0.060
(44) Recall for type location-type = 1.0
(45) Precision for type park-playground = 0.312
(46) Recall for type park-playground = 0.625
(47) Precision for type other = 0.353
(48) Recall for type other = 0.862

- Code: *Here is the link for the code about computation of precision and recall.* https://github.com/kart2k15/Big-Data--Bits_please/blob/master/Task-2--Visualization%2C%20Precision%20%26%20Recall.ipynb

3.6 Visualization of Precision and Recall

This part include visualizations that summarize our findings.Based on the result of precision and recall. We can get the following visualization.

- Visualization1: *Number of columns in which the type appears. In Figure 1, x axis represent the number of count we get for each label for all columns, y axis represent the name of type in label lists. The top five labels for those columns is* ***other, person-name, street-name, address, building-classification.***
- Visualization2: *Prevalence of heterogeneous and homogeneous columns.In Figure 2, the x axis represent the prevalence of heterogeneous and homogeneous, y axis represent the two type heterogeneous and homogeneous.*

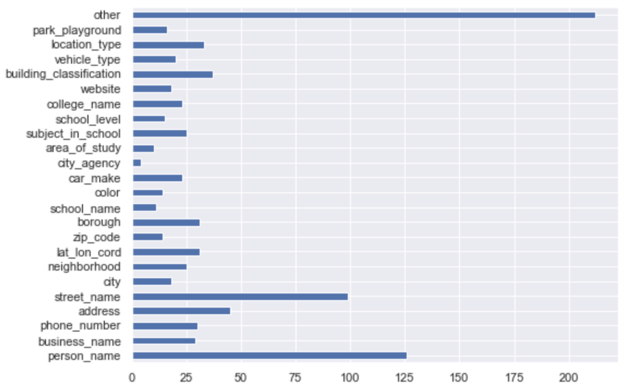


Figure 2: Number of columns in which the type appears

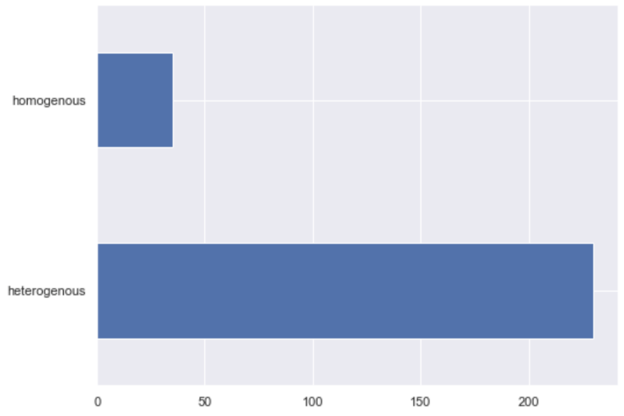


Figure 3: Prevalence of heterogeneous and homogeneous columns

4 TASK 3: DATA ANALYSIS

For Task 3, we chose this topic below:
Identify the three most frequent 311 complaint types by borough. Are the same complaint types frequent in all five boroughs of the City? How might you explain the differences? How does the distribution of complaints change over time for certain neighborhoods and how could this be explained?
In order to answer these questions with solid proof and confidence, we decided to bring out a separate data analysis notebook using *Jupyter Notebook* and *Plotly Chart Studio*.

4.1 Data

The data we used comes from NYCOpenData. We donwload the dataset *.csv* file called *311.csv* versioned from 2010 until 11/30/2019. Link to the data: https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9

4.2 Data Outline

The data has approximately 21 million rows and 41 columns. Its columns contained different semantics of data ranging from spatial, temporal, categorical and numerical features. Some key features are listed below:

- Complaint Type: *Brief description of the complaint*
- Time of Complaint: *Timestamp of the complaint*
- Agency: *Name of the department to which the complaint is related*
- Descriptor: *Nature of the complaint*
- Location Type: *The approximate location where the complaint occurs*
- City/Borough: *The city and borough where the complaint is located*
- Latitude/Longitude: *The latitude and longitude information of the complaint*

4.3 Preprocessing

The data downloaded from NYCOpenData website was thoroughly cleaned so for the preprocessing part we are only filtering the columns/features we need and eliminating all the cells with *Null* values. After preprocessing, we narrowed down the dataset from 41 columns to 9 columns, and compressed the meaningful rows down to approximately 15 million rows.
The second step was to transform our dataset from dataframe to a *SQLite* Format for easy querying. In order to implement this, we used *Sqlalchemy* to load the whole dataframe into an local *SQLite* container and stored it in the same directory as the original dataset. We split the dataframe into chunks of size 50,000 and appended it to the SQL Schema.

4.4 Visualization and Analysis

We then carried out our main data visualization and analysis with *Plotly*, a interactive visualization library enabled for *Python* and *SQL* to plot and store graphs in the cloud.

4.4.1 Tools Used. For this section, we used *Jupyter Notebook*, together with *Plotly online Chart Studio*. Since *Jupyter Notebook* stores your previous results as well as plots and *Chart Studio* is a cloud based storage open to public by default, both are easy to reproduce the results on any *Python* enabled computer.

4.4.2 Assumptions. Our analysis and visualizations are conducted under the assumption that 311 complaints are mainly correlated to the following 9 columns: ***'Agency', 'CreatedDate', 'ClosedDate', 'ComplaintType', 'Descriptor', 'CreatedDate', 'ClosedDate', 'TimeToCompletion'***, and ***'City'***, that no selection bias is generated when we complete feature engineering.
We also assume that data from year 1/1/2010 to 11/30/2019 is able to represent the general pattern of New York City's 311 complaint calls that no augmentation or auxiliary datasets are necessary.

4.4.3 Most Common Complaints by Agency. We rank the complaints by total number in descending order then grouped them by agency. As shown in **Figure 4**, NYC Department of Housing Preservation and Development (**HPD**) and New York City Police Department (**NYPD**) received the most complaints, and they almost doubled the other departments in total. Since housing, crime and safety issues are among the most frequent as it shows, let's see how the complaints distribute across the types and cities.

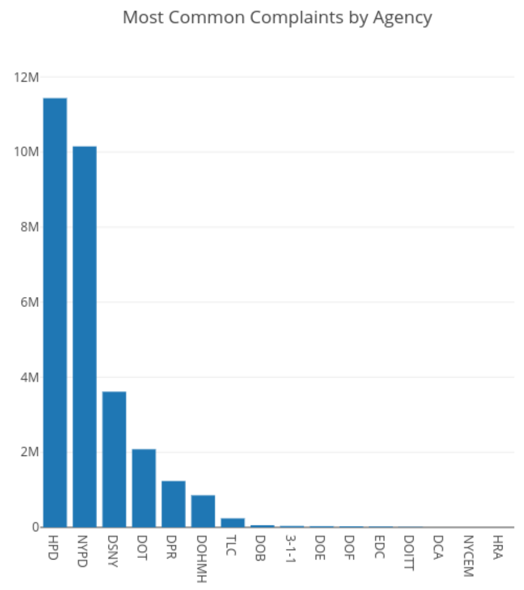


Figure 4: Most Common Complaints by Agency

4.4.4 Most Frequent Complaint Types by City. We rank the complaints by their types in descending order then grouped them by city. We used stacked bar charts to reflect multiple categorical values. We colored cities with categorical color scale chosen from *ColorBrewer*. We can see from **Figure 5** that Noise from all sources are among the most complained issues. Especially in Brooklyn and Bronx, where city developments lead to a lot of new constructions and civil projects, noises tend to be the most unsatisfying aspect of 311 calls.
Apart from noise, we can see that illegal parking in New York (Manhattan) is also a very disturbing issue. This generally matched our expectation as it is never easy to find a spot in Manhattan.
One interesting thing to point out that we used *Cities* instead of *Boroughs* in Profiling this visualization. It came to our mind that some boroughs are so diversified that people may address their complaints according to their multiple lifestyles. Therefore, it is better to map down the dimension a bit and take a city as a whole.
To get a more precise view of how different types of noise contribute in each city, we split the stack bar chart in juxtaposition across complaint types on the x-axis. **Figure 6**

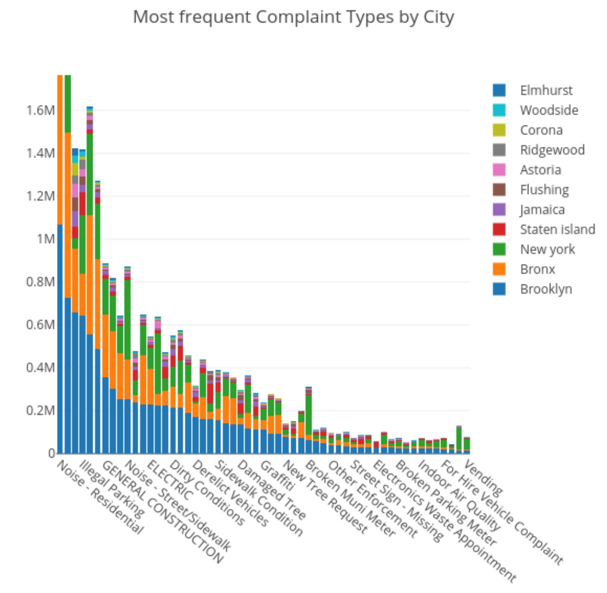


Figure 5: Most frequent Complaint Types by City

clearly showed the different patterns of composition among different cities. From the **Figure 6** we can observe that although residential noise still dominates in most cities, other issues also have their spike in specific locations. For example, Heat and hot water problems happen most frequently in

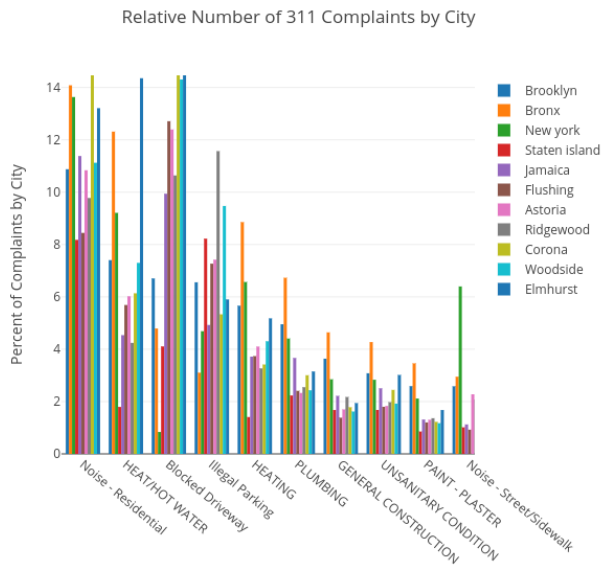


Figure 6: Relative Number of 311 Complaints by City

Brooklyn, while Manhattan has the most severe street noise issues.

4.4.5 Complaints per Hour in A Day. After we have analyzed around the categorical and spatial features, it is time to view the complaints in a temporal manner. We ordered the complaints by their created time chronologically then grouped all of the complaints together within the same hour. **Figure 7** is a hourly bar chart showing the trend of all complaints. It is surprising to see that hours 0 to 1 has the most

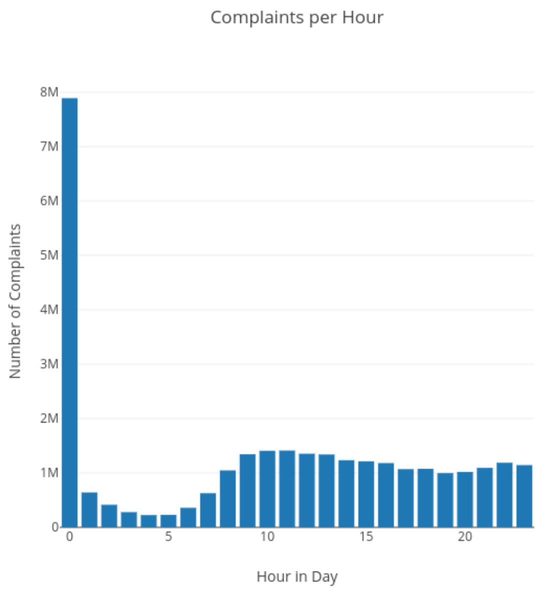


Figure 7: Complaints per Hour in A Day

complaints. One of our inferences is that some complaints are manually grouped to 00:00 as they have data consistency issues or the audit person who registered them only added date without a specific time.
The rest data did reflect the trend that residents tend to complain mostly in day hours.

4.4.6 Noise Complaints in NYC per Hour in A Day. Lets split the noise complaints from the rest and put them on the same grid.
Figure 8 shows that noise complaints have a nightly pattern. During daytime, citizens at work don't seem to be disturbed by noises, while their complaints start to balloon from 8pm until midnight. High midnight noises reflect that New York is indeed a city that never sleeps. From our point of view, noise sourcing from **nightclubs**, **trucks** and **street music** are among the main factors.

4.4.7 Most Common 311 Complaints per Hour in A Day. Using the previous method, let's stack the complaint types for each hour. **Figure 9** displays the most common 311 complaints per hour. Apart from the noise and columns indicating 'Other', Most daytime hours possess the same color

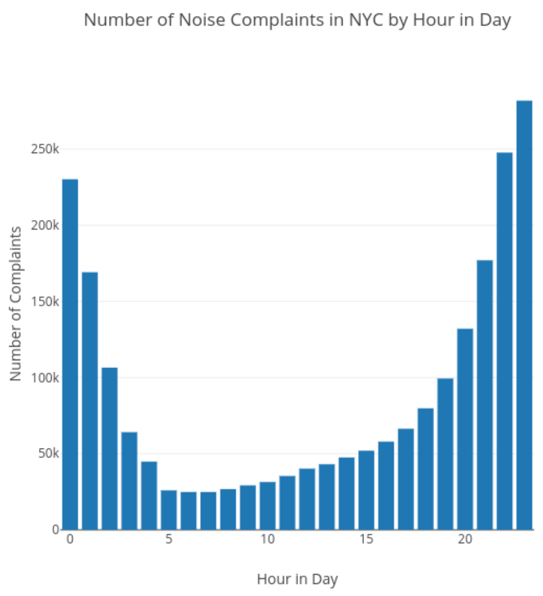


Figure 8: Noise Complaints in NYC per Hour in A Day

patterns and the distributions between complaint types are fairly split. In night time however, *Blocked Driveway* tends to drop and *Large Bulk Item Collection* tends to rise. From the shift of trend we can infer that during off-work time, residents' complaints focused more on the housing and living issues than commuting and convenience.

4.4.8 Number of Complaints per 15 Minutes. After dividing complaints by type let's look at the data in a more 'old fashioned' way. **Figure 10** shows the shift of total number of complaints every 15 minutes in the 5 days from 1/1/2010 to 1/5/2010.
Although the series generally follows a periodic behavior as we analyzed previously, the data on New Year's Day was distributed more evenly. As everyone stayed up late to celebrate the new decade, problems also emerge from all walks of life. Increased people activity 'smoothed' the skewed distribution as if daytime and nighttime were the same.

4.4.9 Number of Complaints per Day. Enlarge the time scale to to the entire data set, **Figure 11** returns a time series map of total number of complaints versus time. Generally, complaints increases in a linear trend. The sharp gap in 2019 might contribute to that calls are yet to be dialed for December, when most heating and plumbing issues happen. Therefore we look at years from 2010 to 2018 to eliminate bias.
There are spikes occurring very periodically during each year. We try to relate the spikes to some kind of natural events which does but not frequently happen in New York. It turns out that issues resulted from snowfalls contribute to the spikes.

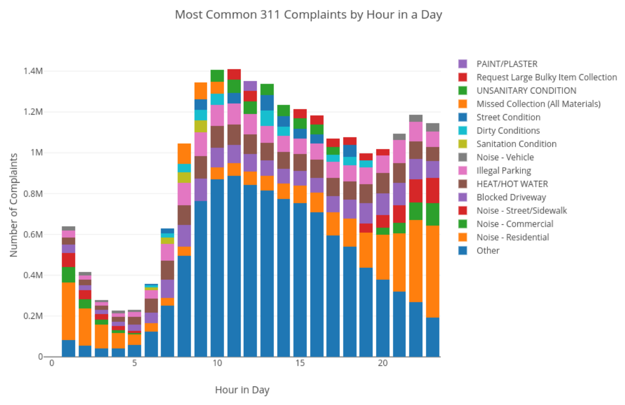


Figure 9: Most Common 311 Complaints per Hour in A Day

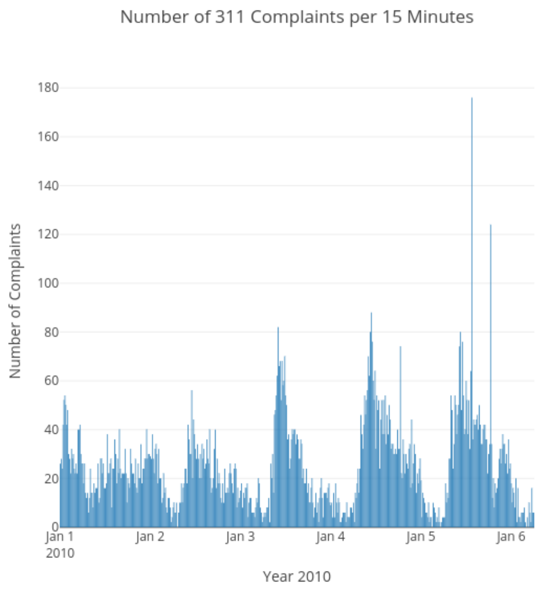


Figure 10: Number of Complaints per 15 Minutes

Figure 12 shows the monthly and annual snowfall at Central Park. Here we assume Central Park's snowfall can roughly represent all the snowfalls throughout New York City. From **Figure 12** we can witness that snowfalls hit the peak from late December until early February, that is exactly when the spikes occur each year. It is obvious here that New York City's over-aged infrastructure can no longer fight the fierce weather.

4.5 Conclusion

In the analysis above, we expanded the data to multiple dimensions like city, time and complaint types so as to answer the questions we initially raised. In conclusion, New

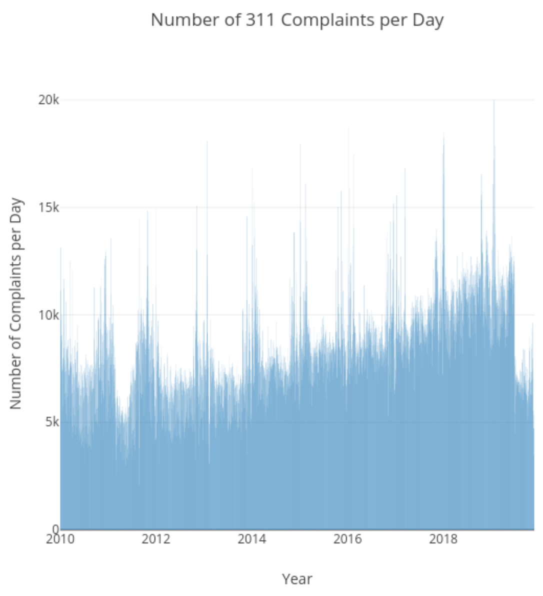
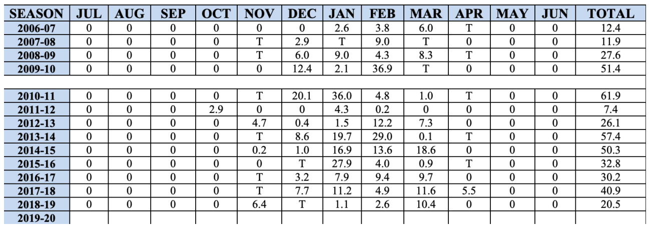


Figure 11: Number of Complaints per Day



SEASON	JUL	AUG	SEP	OCT	NOV	DEC	JAN	FEB	MAR	APR	MAY	JUN	TOTAL
2006-07	0	0	0	0	0	0	2.6	3.8	6.0	T	0	0	12.4
2007-08	0	0	0	0	T	2.9	T	9.0	T	0	0	0	11.9
2008-09	0	0	0	0	T	6.0	9.0	4.3	8.3	T	0	0	27.6
2009-10	0	0	0	0	0	12.4	2.1	36.9	T	0	0	0	51.4
2010-11	0	0	0	0	T	20.1	36.0	4.8	1.0	T	0	0	61.9
2011-12	0	0	0	2.9	0	0	4.3	0.2	0	0	0	0	7.4
2012-13	0	0	0	0	4.7	0.4	1.5	12.2	7.3	0	0	0	26.1
2013-14	0	0	0	0	T	8.6	19.7	29.0	0.1	T	0	0	57.4
2014-15	0	0	0	0	0.2	1.0	16.9	13.6	18.6	0	0	0	50.3
2015-16	0	0	0	0	0	T	27.9	4.0	0.9	T	0	0	32.8
2016-17	0	0	0	0	T	3.2	7.9	9.4	9.7	0	0	0	30.2
2017-18	0	0	0	0	T	7.7	11.2	4.9	11.6	5.5	0	0	40.9
2018-19	0	0	0	0	6.4	T	1.1	2.6	10.4	0	0	0	20.5
2019-20													

Figure 12: Monthly & Annual Snowfall at Central Park

Table 1: Division of Labor

Member Responsible	Comments
Kartikeya	Task1/Task2,
Xue	Fuzzy-matching,true-label, Task2
Cheng	Task3, Data Analysis, Visualization

York City's diversified lifestyle, imbalanced city development , over-aged infrastructure and round-the-year snowfalls formed its unique city complaint behavior. Although it is noisy all the time and chilling cold during winter, New York City, like no where else, still attracts the love and focus from all over the world. And as New York City residents, we are really proud of it.

5 TEAM CONTRIBUTION

Our team possesses exactly 3 members so we divided the workload evenly, with each member responsible for a single task. **Table 1** shows the division of labor in our team.

REFERENCES

(1) Buillding Classifications. https://www1.nyc.gov/assets/finance/jump/hlpbldgcode.html
(2) Vehicles. https://data.ny.gov/api/assets/83055271-29A6-4ED4-9374-E159F30DB5AE
(3) Car Make. https://data.ny.gov/api/assets/83055271-29A6-4ED4-9374-E159F30DB5AE
(4) Type of Location. https://www.researchgate.net/figure/Examples-of-location-categories-used-and-the-list-of-the-categories-for-which-names-are_fig4_276361716
(5) Areas of Study https://www.princetonreview.com/majors/all
(6) College/University Names. https://en.wikipedia.org/wiki/List_of_colleges_and_universities_in_New_York_City
(7) City Agency. https://www1.nyc.gov/nyc-resources/agencies.page
(8) Subjects in School. https://www.quora.com/What-are-the-common-middle-school-subjects-taught-in-the-United-States
(9) Type of Location. https://developers.google.com/places/web-service/supported_types
(10) Vehicle Type Car Make The Image Below Was Collected From. https://data.ny.gov/api/assets/83055271-29A6-4ED4-9374-E159F30DB5AE