Big Data Team Rangers Final Project

Fall-2019

Smit Sheth
Computer Science
NYU Tandon School of Engineering Brooklyn, NY US
shs572@nyu.edu

Akshat Khare
Computer Science
NYU Tandon School of Engineering Brooklyn, NY US
ak7674@nyu.edu

Rajeev Reddy Ilavala
Computer Science
NYU Tandon School of Engineering Brooklyn, NY US
rri223@nyu.edu

ABSTRACT

The first part of the project aims to profile a large collection o/f NYC open data sets and derives metadata that can be used for data discovery, querying, and identification of data quality problems. The second part aims to identify and summarize semantic types present in each column of a given cluster. The third part aims to analyse the NYC 311 complaints for each of the five boroughs that form New York City. Based on the results of our analysis, we intend to further explore these trends and discover the reasons behind the increase or decrease in the number of complaint types.

INTRODUCTION

In this digitalized world, we are producing a huge amount of data every minute. The amount of data produced in every minute makes it challenging to store, manage, utilize, and analyze it. With around 1900 tables in NYC open dataset, it is a challenge to analyze the data and extract meaningful information out of it. To extract meaningful information we first need to clean the data as the real problem arises when we try to combine unstructured and inconsistent data from diverse sources, it encounters errors. Missing data, inconsistent data, logic conflicts, and duplicates data all result in data quality challenges. To improve the quality of data and then perform analysis on it, we use Apache Spark which is an open source framework for big data which is much faster and has many other advantages over other frameworks.

TASK 1

In this task we profile a large collection of open data sets and derive metadata that can be used for data discovery, querying, and identification of data quality problems. The dataset used is NYC open dataset. Below is the metadata we extracted -

- Number of non-empty cells This was done with the help of a simple sql query.
- Number of empty cells
- Number of distinct values
- Top -5 most frequent values
- Data types

The information for the number of non-empty cells, empty cells and number of distinct values was done by writing simple sql queries. A major challenge for us was to find top 6 most frequent values and data types. Below are the methods we used to find top 5 frequent values and data types.

Tools Used:

- Pyspark (Apache Spark) for parsing and cleaning large datasets into smaller ones.
- DateUtil Package for parsing the data in to date time.
- Matplotlib for bar plots.

Methods

We implemented a Spark UDF that returns the data type of each cell in a column, here we are looping through each column of dataset and with the help of UDF all the required statistics are obtained in a Json format. To identify the date/time data type we used the parser library from dateutil package of python. Integer and float values are identified by parsing them to primitive data types int and float respectively.

Challenges

- **Parsing datetime format** Initially we tried using a list of valid date time formats but we observed that the number of different date type formats are not limited so to overcome that we are using parser library from dateutil python package. This helped in reducing the false negatives but at the same time we faced an issue with increase in false positives since this parser util is labelling few strings as date times even though the data type of that column is a string.
- **Special characters** Some of the data sets have special characters in their columns and even in the headers,

this made reading the contents difficult since its not in UTF-8.

- **Strings misclassified as date time** Even though few columns have only strings in it, a few strings are being misclassified as date time because of the date time parser. For example 'DBN' column of dataset -'vxxs-iyt2.tsv' has strings in all the cells but few strings such as '0M0115' are being parsed as date time.

Optimizations

- **UDF to return data type** Initially we appended each parsed value into its corresponding list and calculated the statistics over it, the major drawback is memory utilization as it needs to store each value in another list again. To overcome this we used UDF to get the data type and this helped improve the performance significantly

Evaluation

After doing the optimizations were were able to bring down the time for profiling from approximately 28 days to approximately 3 days.

1- frequent itemsets

Data Type	Integer	Real	Text	Date/Time
Frequency	10586	3457	9137	1350

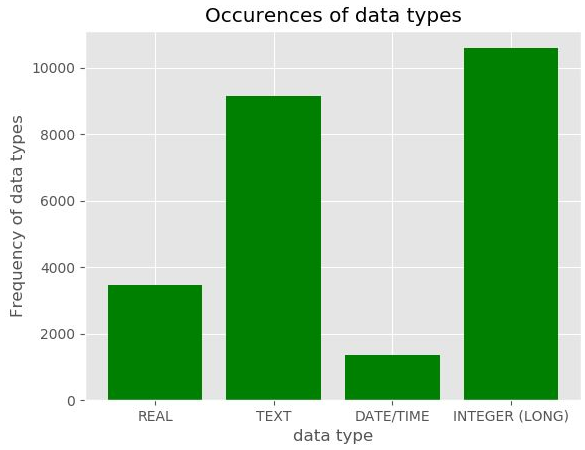

Figure 1: Statistics for 1 frequent itemset

Figure 1 shows, for each data type, how many columns contain that type**.** Here we can see that the integer is the most frequent data type. Text data type appears to be the second most but this is quite inappropriate because almost all of the columns used a string 'No Data' to indicate Null and this lead to have a Text data type for most of the columns.

2- frequent itemsets

Data Types	Real, Text	Date, Text	Integer Real	Date, Integer	Integer, Text	Date, Real
Freq uency	842	1077	2111	417	2696	123

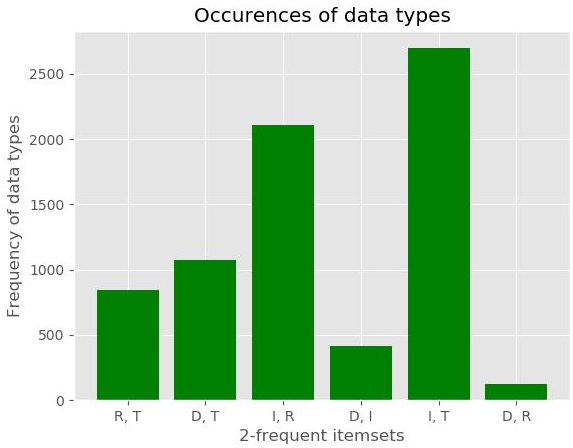

Figure 2: Statistics for 2 frequent itemsets

3- frequent itemsets

Data Types	I, R, T	D, I, R	D, R, T	D, I, T
Frequency	288	111	123	374

(D- Date/Time, I- Integer, R- Real, T- Text)

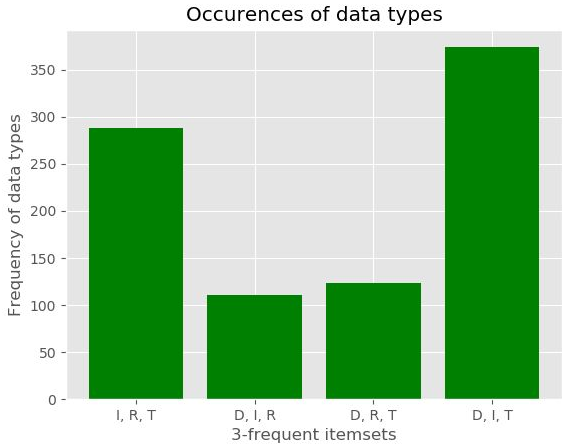

Figure 3: Statistics for 1 frequent itemset

4- frequent itemsets

Data Types	Date, Real, Integer, Text
Frequency	111

TASK 2

For this task we extracted and summarized semantic types present in a column for a subset of the dataset used in Task1. We first manually labelled all the columns in the cluster then compared it with the results of the code to calculate precision and recall scores.

Tools Used:

- Pyspark (Apache Spark) for parsing and cleaning large datasets into smaller ones that could be used with pandas.
- Jupyter Notebooks for neat and clean data processing in Python.
- Ms Excel to calculate precision and recall scores and to generate histograms.

The code is a sequence of if-else statements with each if statement containing the logic for each semantic type. To get the most accurate results we used SequenceMatcher function of difflib which is a python library, this function gives similarity scores of two words, we took all the words whose similarity score was greater than 0.75. Below are the semantic types we used to categorize the columns and the strategy we used for them:

● **Person name** - We identified the columns with person-name using a regex '([a-zA-Z]{3,30}\s*)+' which takes in all the rows which do not contain any special symbols or characters. Person name was the most difficult to identify so to optimize it we put it at the end of if else statements so that all the other semantic types get detected first. However, the algorithm did give out some false positives, the precision for person name is coming out to be 0.27 and recall is 1.

● **Business name** - One major challenge for identifying a business name is that there are a variety of businesses and all of them contain different keywords like PIZZA, DELI, RESTAURANT, so it gets difficult to identify if a row is of business name type. We created a dictionary of all those common words and matched it with each word of the row to get a match. Our algorithm got a precision score of 0.66 and a recall for 0.8. These scores can be improved by finding more such keywords and adding it to the dictionary.

● **Phone Number -** Phone number are usually a 10 digit number with spaces or brackets or country code. We used the following regex to identify these types : '\D?(\d{3})\D?\D?(\d{3})\D?(\d{4})'. Our algorithm was 100% accurate for phone number with precision 1 and recall 1.

● **Address** - Different people write different formats for address and identifying all of them is a difficult task. The most common type of address people write is like '101 W Main St.' so to identify all such addresses of this type we used a regex '(\d{3,})\s?(\w{0,5})\s([a-zA-Z]{2,30})\s([a-zA-Z]{2,15})\.?\s?(\w{0,5})' which is an expression for splitting up a generic street address like the one mentioned above. The precision for this regex is coming out to be 1 and recall 0.71.

● **Street name -** Street usually contains keywords like 'street', 'Avenue', 'boulevard' etc. So we made a dictionary of such words compared it with each word of the row to get a match. The precision for this is 0.61 and recall is 0.85.

● **City -** We identified city by creating a dictionary of all the cities in New York state and then matching it with the rows. Since we only added cities of New York and not from other states, our algorithm did not run well for cities. The bigger the dictionary the better the results but this will obviously slow down our algorithm.

● **Neighborhood -** We again created the dictionary containing all the neighborhoods of NYC. However, there are many neighborhoods whose names match with many cities therefore it gets difficult to identify which one is what. The precision for our algorithm is 0.64 and recall is 0.94.

● **LAT/LON coordinates -** We used the following regex to identify coordinates, it detects the standard format of coordinates like (40.5665301462, -74.1385338765): '(\()([-+]?)([\d]{1,2})(((\.)(\d+)(,)))(\s*)(([-+]?)([\d]{1,3})((\.)(\d+))?(\))'. This regex is 100% accurate to detect the results. But, since we were unable to run two datasets containing coordinates as the file size was too big, the precision score is 0.8 and recall is 1.

● **Zip code -** Following regex was used to identify zip codes '\d{5}$|^\d{5}-\d{4}', it detects standard zip codes like 5 digit numbers eg. 01843. This regex is 100% accurate with precsion 1 and recall also 1.

● **Borough -** Borough is identified by matching words in the rows with borough names. Since our dictionary for city was small, many columns of type city were classified as borough as it also contained names of boroughs that's why we got a precision score of 0.63 but all the columns of type borough were correctly identified (recall 1).

● **School name-** (Abbreviations and full names) School names usually contain the word school. So we compared all the words of the row with school to get a match. The precision score for school is 0.73 and recall is also 0.73.

- **Color -** We created a dictionary containing most of the colors and compared it rows. The SequenceMatcher function of difflib library proved very helpful to identify the colors as most of the colors had spelling mistakes for example black was written as blk or blck etc. The SequenceMatcher function helped in identifying most of the colors having spelling mistakes otherwise they would have been counted in some other category.

- **Car make -** Car make is again done by creating a dictionary containing most of the car makes. The car-make column in NYC open dataset is fairly clean so we got a good precision score of 0.82 and recall 0.87.

- **City agency -** City agency is also done by creating a dictionary of city agencies. The precision score for this is 0.91 and recall is 1.

- **Subjects in school** - We created a dictionary containing most of the subjects. The subjects are coming out to be 100% accurate with precision 1 and recall 1.

- **School Levels** - Created a dictionary containing most of the school levels. This type did get mixed with many columns of type school as many school levels contain the keyword 'school'. Therefore the precision is coming out to be 0.58 and recall 0.62.

- **College/University names -** These usually contain keywords like 'institute', 'university' etc. So we created a small dictionary containing all such words and matched them with rows. There was only one column containing college names which was detected correctly.

- **Websites -** (e.g., ASESCHOLARS.ORG) Websites were identified using the following regex which identifies common website names, 'https?:\/\/)?(www\.)?([a-zA-Z0-9]+(-?[a-zA-Z0-9])*\.)+[\w]{2,}(\/\S*)?'. The columns which were not detected were like '<A HREF=WWW.BSFHC.ORG>VISIT OUR WEBSITE</A>'. Therefore this regex got a precision of 1 and recall 0.66.

- **Building Classification** - This was again done by creating a small dictionary of all such words and then matching it with rows. The precision for building classification is 1 and recall is also 1.

- **Vehicle Type** - We created a small dictionary containing such keywords and compared it with rows. The vehicle type columns were not straightforward, containing a lot of false entries therefore we got a precision of 1 and recall 0.33.

- **Parks/Playground** - These usually contain keywords like 'park', 'playground' so we created a small dictionary and compared it with rows. Some of the columns were not identified were because the count of rows not containing the keywords which we put in our dictionary was more than the rows whose words were in our dictionary. Therefore some of the columns were identified as person name instead of parks. The precision is 1 and recall is 0.33.

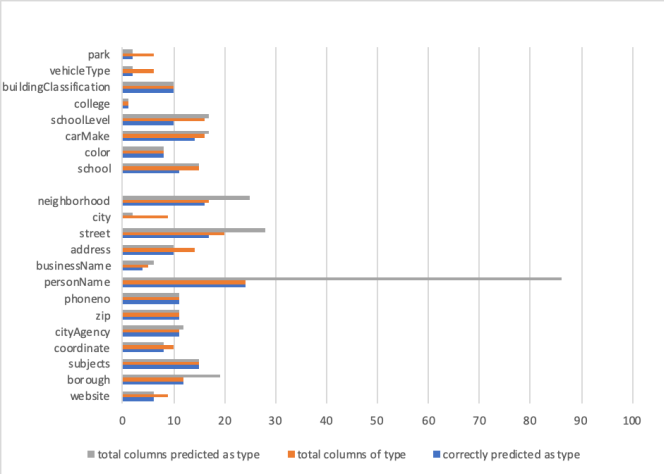

Figure 4: the Figure shows histogram of each type and its count

Our algorithm gave up to three semantic labels for a particular column. The first label was the type which occurs most frequently in a column call it type 1, the second and third label are the types which has count at least 50% of type 1.

Overall precision of the algorithm we implemented is 0.65 and recall is 0.82. However, if we remove the person-name semantic label the precision and recall both increase to 0.72 and 0.89 respectively.

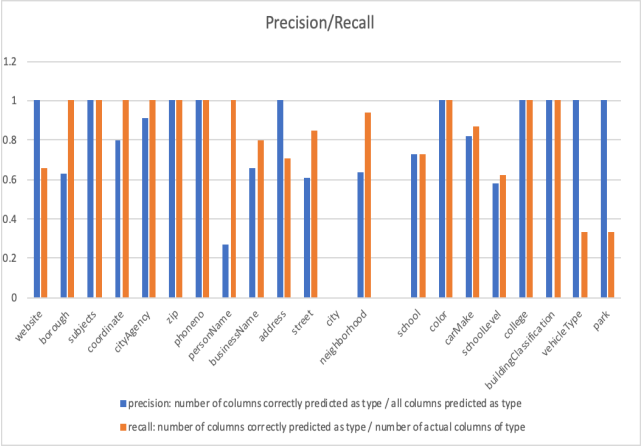

Figure 5: the Figure shows precision and recall scores for different semantic types

TASK 3

For this part of the project we did analysis on the NYC 311 complaints dataset for the year 2018. The dataset contains all the service request reported through the 311. The dataset is organized into 41 columns much of which was not used for analysis with over 2.6 million records.

We analyzed the number of reported service requests with respect to Boroughs that form New York City. To do so, we extracted eight columns of our interest from the main data set: 'Created Date', 'Closed Date', 'Agency', 'Complaint Type', 'Descriptor', 'Incident Zip', 'City', 'Borough'.

Tools Used:

- Pyspark (Apache Spark) for parsing and cleaning large datasets into smaller ones that could be used with pandas.
- Jupyter Notebooks for neat and clean data processing in Python.
- Pandas and matplotlib library for data analysis and exploration.

We began our analysis with a set of specific questions we formed from exploring and thinking about the dataset, owing to which we had extracted the above-mentioned columns from the main data set.

1. What are the 3 most frequent 311 complaints types for each borough.
2. Monthly analysis of complaints to see if there are months were the complaints increases/ decreased.
3. Does more rental houses in neighborhood influence specific type of complaint.
4. Does more number of complaints in each borough mean more time to process the requests.

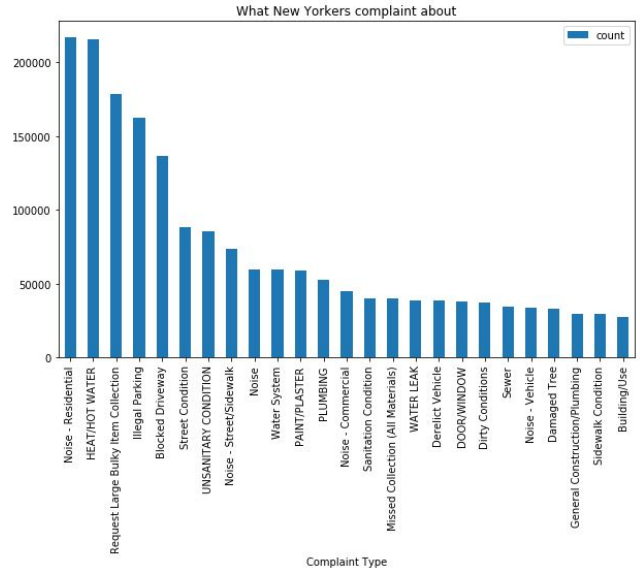

Figure 6: Figure shows complaint distribution for New York city

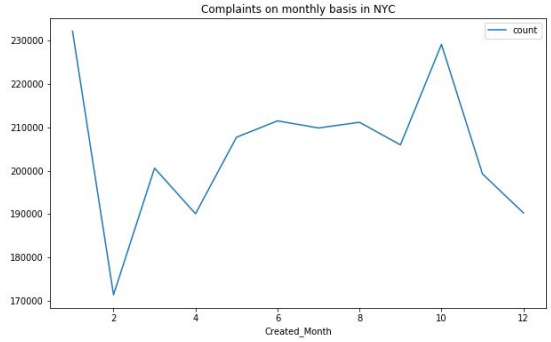

Figure 7: Figure shows monthly complaint distribution in New York city

The decline in complaints volume during the month of December can be related to the vacation period as more people are out of the city to celebrate Christmas and New Year's Eve and also there are holidays at work as well as schools.

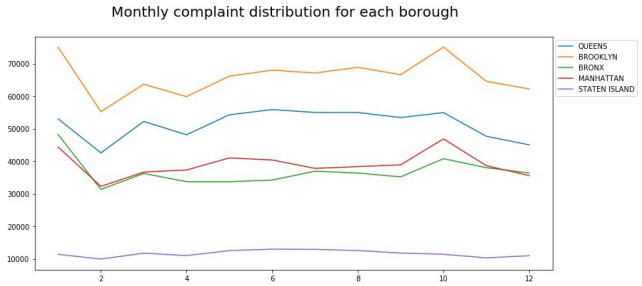

Figure 8: Figure shows monthly complaint distribution across each borough

The visualizations by borough reveal that the complaint types in New York are uneven across the boroughs. They do seem to vary by roughly according to the population. Brooklyn is the most populated borough and has the highest number of complaints followed by Queens. Staten Island is much less populated and has the lowest complaints as well.

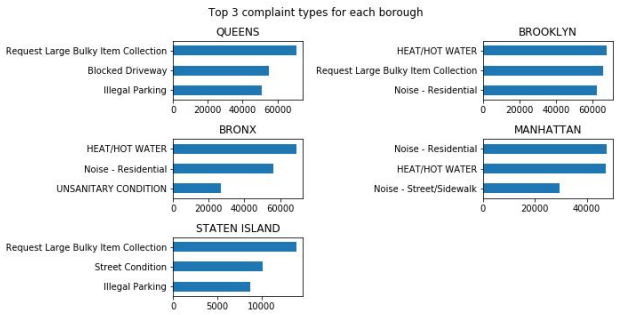

Figure 9: Figure shows the top 3 most frequent complaint types in each borough

One interesting point seen from the above chart is though it appears that Brooklyn saw the most Heat/ Hot water complaints, Bronx (1.4M) with its population at 1.2M less than Brooklyn (2.6M), sees nearly 1000 more complaints of this type. This leads us to consider whether the living conditions in Bronx are much more difficult, especially during the winter season.

Before we looked at the data, we had predicted that there would be a spike in the complaints during the winters- because heat/hot water complains would drastically increase during winters than in summers. The resulting graphs support our hypothesis.

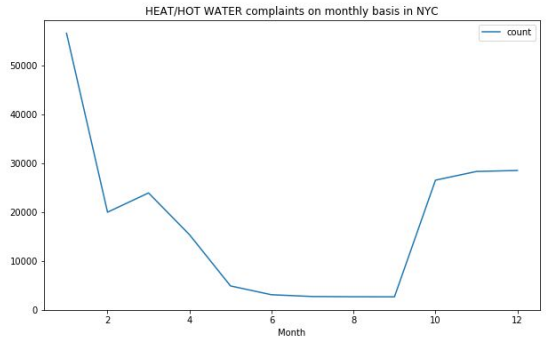

Figure 10: Figure shows the monthly breakdown for the number of Heat/Hot water complaint type

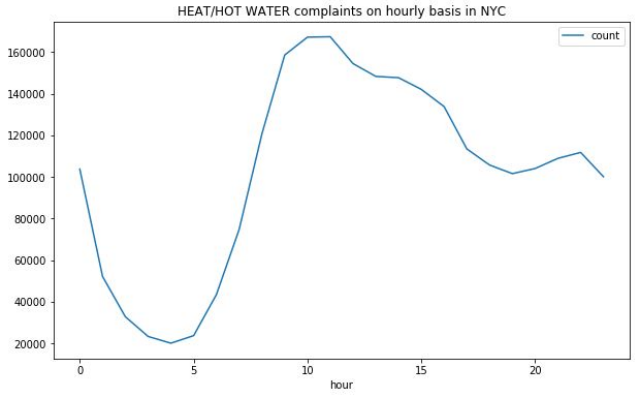

Figure 11: Figure shows the breakdown of hourly complaints for Heat/ Hot water

After visualizing the distribution of complaints related to heat/hot water which is the single most-frequent housing complaint in **most of the** borough**s**, we decided to drill down further to find out why the borough Staten Island did not have heat/hot water as one of its most frequent complaints throughout the year. For this we explored the Housing NYC buildings by Units dataset which gives us the information about the construction projects and the type of houses it has.

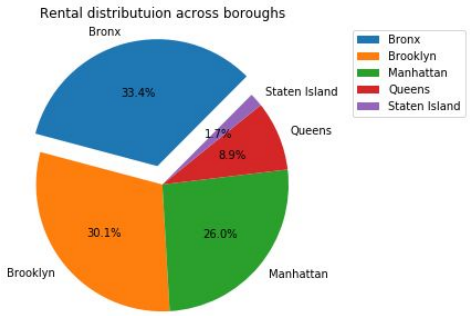

Figure 12: Figure shows rental units distribution

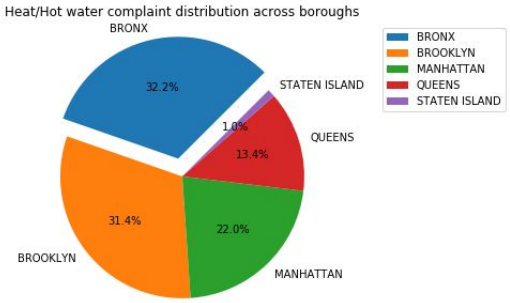

Figure 13: Figure shows heat/hot water complaint distribution

As seen from the above diagram, Staten Island only accounts for 1% of the city-wide complaints. The reason is that there aren't many rental houses available in Staten Island and most Islanders own their house and hence they are not going to complain about heat or water issues instead they'll fix them themselves. Whereas in Brooklyn, Bronx and Manhattan there are many people staying on rent and hence complain lots of heat and water issues as they are the basic needs. The resulting visualizations support our hypothesis that more rental houses the more the number of heat/water complaints.

Borough	**Rental Units**	**Total Units**
Bronx	37212	42666
Brooklyn	33505	53831
Manhattan	28932	53341
Queens	9898	15641
Staten Island	1936	2756

Figure 14: Figure shows the breakdown of the number of units constructed from 2014 to 2018

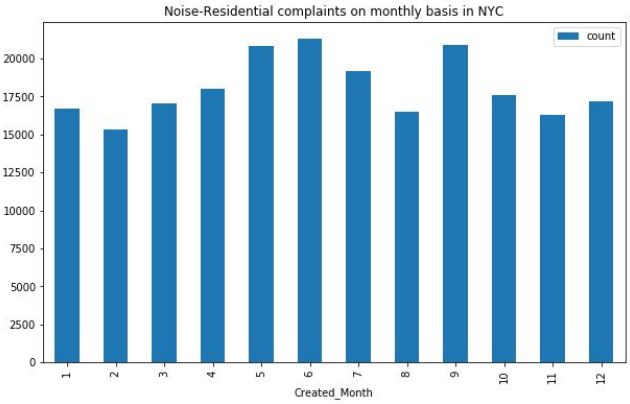

Figure 15: Figure shows the breakdown of the number of Noise-Residential complaints on monthly basis

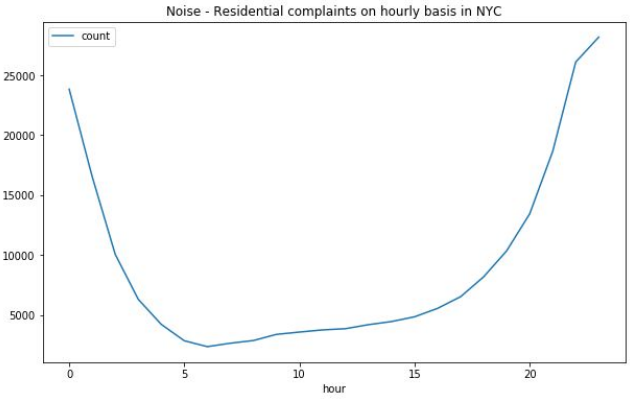

Figure 16: Figure shows the hourly breakdown for the number of Noise-Residential complaint type

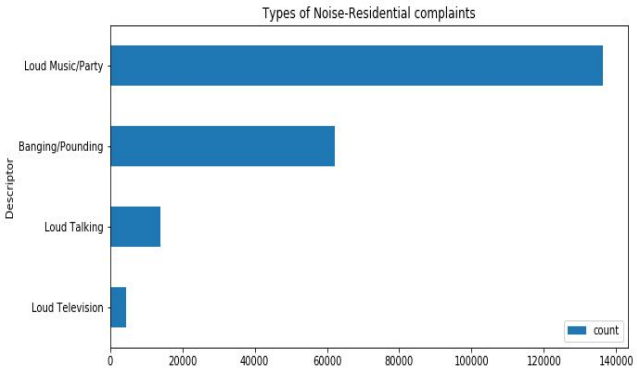

Figure 17: Figure shows the types of Residential noise

Noise – Residential (which is the complaint about other residential units) is the second most complaint in New York city which makes sense in such a crowded and densely populated city. It hits the highest peak of submission during the 11th hour in the night. The majority of the calls tend to be placed during 10 pm - 1 am. Complaints usually start to ramp up starting at 4pm when usually start returning to their home from their work/school.

Overall Loud Music/Partying had the highest volume of calls across almost all zip codes all boroughs with Brooklyn, Bronx and Manhattan having the most, which is not surprising given the large numbers of younger residents in those neighborhoods.

Loud noise complaints starts to peak from the month of May-September when it becomes warmer, schools are out, and there are more activities to do outside. June has the highest volume of calls.

PROJECT LINKS

1. Path where task1 jsons are stored - /user/ak7674/team_rangersFinalProject
2. Path where task2 jsons are stored - /user/ak7674/team_rangersFinalProject
3. Git-hub repository link - https://github.com/RajeevReddyIlavala/BigData_TeamRangers.git

INDIVIDUAL CONTRIBUTIONS

1. Strategy for task1 - Rajeev, Smit, Akshat
2. Task1 implementation - Rajeev, Smit, Akshat
3. Task1 visualizations- Rajeev
4. Strategy for task2- Akshat, Smit, Rajeev
5. Task2 implementation - Akshat, Smit, Rajeev
6. Task2 visualizations - Akshat
7. Task2 calculations- Akshat, Smit
8. Task3 strategy- Smit, Akshat, Rajeev
9. Task3 implementation - Smit, Akshat, Rajeev
10. Task3 visualizations - Smit
11. Shell scripts - Akshat
12. Report- Rajeev, Smit, Akshat

ACKNOWLEDGMENTS

We would like to thank Professor Julia Stoyanovich and Professor Juliana Freire for providing us the opportunity to analyze NYC open dataset. We would also like to thank NYC open data for providing us the dataset.

REFERENCES

[1] "311 Service Requests" from https://nycopendata.socrata.com/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9/data

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

[3] https://docs.python.org/2/library/difflib.html

[4] Yeye He and Dong Xin. SEISA: set expansion by iterative similarity aggregation. International conference on World Wide Web (WWW '11), 2011

[5]http://regexlib.com

[6]https://docs.databricks.com/spark/latest/spark-sql/udf-python.html for UDF functions

[7] Profiling relational data: a survey. Abedjan et al., VLDB Journal 2015 (https://link.springer.com/article/10.1007/s00778-015-0389-y)

[8] Mining Database Structure; Or, How to Build a Data Quality Browser. Tamraparni Dasu et al., ACM SIGMOD 2002.