Using 311 Call Data to Inform Housing Choices in New York City

Dwiref Oza, Peter Malinverni
dso2119, plm2130
E6893: Big Data Analytics, Final Project
Department of Electrical Engineering, Columbia University

***Abstract*—Most housing recommendation websites filter properties and areas based on price, number of rooms and availability of amenities in the immediate neighborhood. These filters provide the consumer with an incomplete perspective of the housing conditions and neighborhood character in the properties suggested to them by these websites. We devise a web-app using Django that visualizes zip codes using a leaflet.js based python library on a map of New York City using GeoJSON data. The user has to provide a few characteristics they would like to avoid in their potential neighborhood, such as reports of rat infestations, frequent noise complaints, etc. This is made possible through NYC OpenData's 311 dataset, which has detailed records of all 311 complaints from 2010 to the present. In this report, we discuss the methodology used to fetch and process the data, the techniques involved in deployment of the web-app and the manner of visualization that the end user is intended to see.**

***Index Terms*—NYC OpenData, Big Data, Django, Mapping**

I. Introduction

Deciding where to live is often a daunting task. For many people, moving to New York City can be difficult because they lack the local knowledge of which neighborhoods would be ideal for them. This lack of knowledge causes many people to simply pick the neighborhood that is closest to their workplace or has the cheapest rent. This is especially true for university students - the tendency for students to live close to their school inflates housing prices near universities and keeps students in the social bubble of their institution.

There is a need for a tool that provides the general public and students alike with relevant neighborhood-specific information that empowers them to make better choices about where to live in the New York City, rather than simply choosing the closest and cheapest housing option.

New York City has released a public dataset called "311 Service Requests from 2010 to Present", which lists information about every 311 call in New York City since 2010. The information filed for each call may include but is not limited to the date and time a call was made, the topic of the call, the status of whether the call's topic has been resolved, and the zip code from where the call was made. Some of the call topics include noise complaints, rat sightings, burst pipe complaints, and water quality complaints. These are geographically-linked quality of life indicators that might help inform a prospective homeowner or renter about where in the city best matches their preferences for neighborhoods.

We created a web-app using `Django`, `Bootstrap`, `HTML`, `Python`, `Pandas`, and `folium`. A django form is filled out and submitted by a user, which generates an `HTML` file of a choropleth map of New York City postal codes. This map is generated using Python, folium and a `GeoJSON` map of New York City postal codes. The values that color-code the postal zones come from a Python program called 'mapping.py' that generates a livability index for each postal code based on the user's input to the form.

Our hope is that with a tool to evaluate and visualize livability for neighborhoods based on a user's personal preferences, we can help people find the home that is right for them in more ways than just affordability and vicinity to their workplace or university.

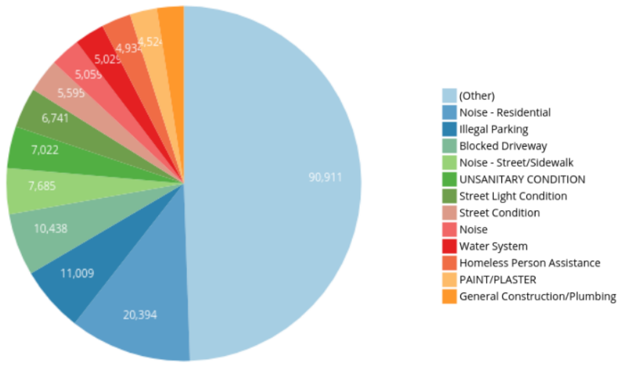
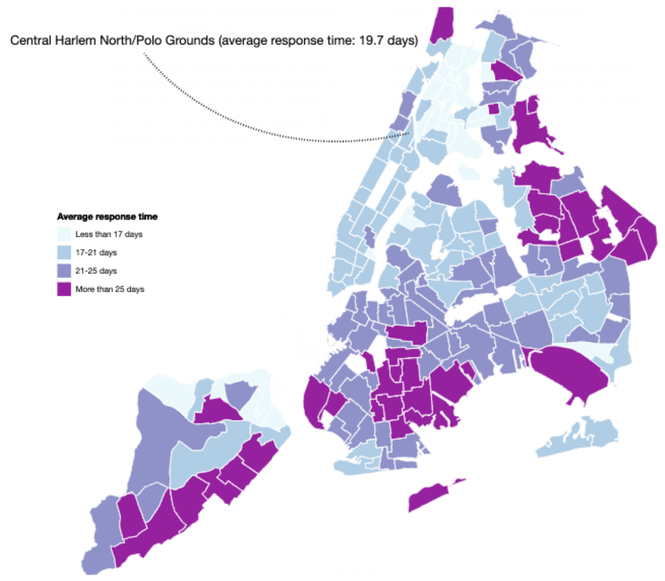


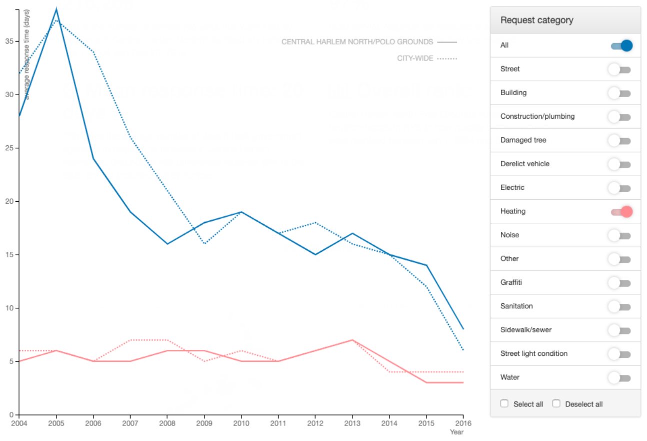
Fig. 1: The distribution of complaint types recorded in the 311 dataset. Nearly 50% of the data is filed as "Other".

II. Related Work

The 311 dataset is massive and presents challenges in sparse data representation. Li et. al [2] used the 311 dataset to develop an adaptive technique for sparse factorization. Even with challenges associated with the large size of the dataset, it has been used in a number of academic papers. These include an analysis of response times to 311 calls [3], a study of ethno-racial diversity's impact on neighborhood conflict in New York City [4], and an analysis of citizen engagement in New York City government [5].

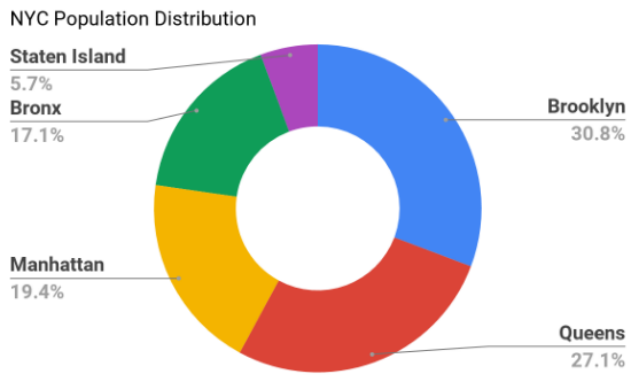


(a)



(b)

Fig. 2: (a) A screenshot of a choropleth map from 'visreq' [1]. The color key is always the average response time to all service request types for a given neighborhood. The user cannot choose which service request type to visualize.
(b) A screenshot of a plotting tool from 'visreq' [1]. The plot shows both citywide and neighborhood-specific service request response times as a function of time. Here, the service request response time for all types and for heating service requests in Harlem compared to the city-wide rate over the period of 2004 to 2016. The switches on the side can be toggled and off depending on the user's preference.



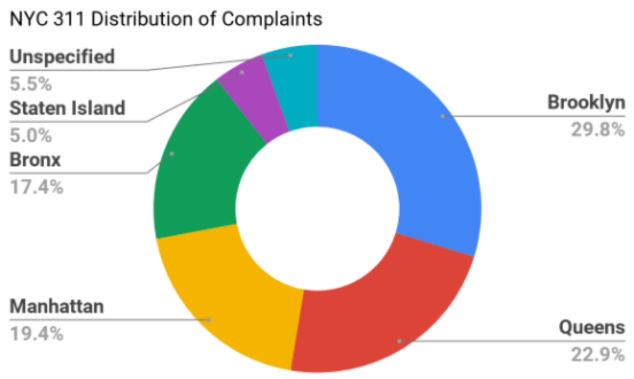


Fig. 3: There is strong correlation between the distribution of complaints in the 311 dataset and the distribution of population in New York City across its boroughs.

These papers are excellent resources for academics in political science and sociology, and for policy makers, but they fall short of providing the general public with useful information. To that end, there are GitHub repositories that visualize 311 data using heat maps [6], choropleth maps, interactive line graphs [1], and more. Of these repositories, 'visreq' by S. Ejdemyr is one of the most useful for potential renters and homeowners. The app generated from this repository allows users to submit the name of a neighborhood and receive straight line plots comparing their neighborhood to the city as a whole on the basis of the average response time to a particular type of service request.

We think that this type of tool is a step in the right direction, but it still does not allow its users to compare different neighborhoods in terms of the types of service requests submitted. When you are searching for a place to live, side-by-side comparisons of different neighborhoods on indicators like residential noise and water quality would be useful. We have found no prior work using the 311 dataset from NYC OpenData that enables users to do this.

III. Data

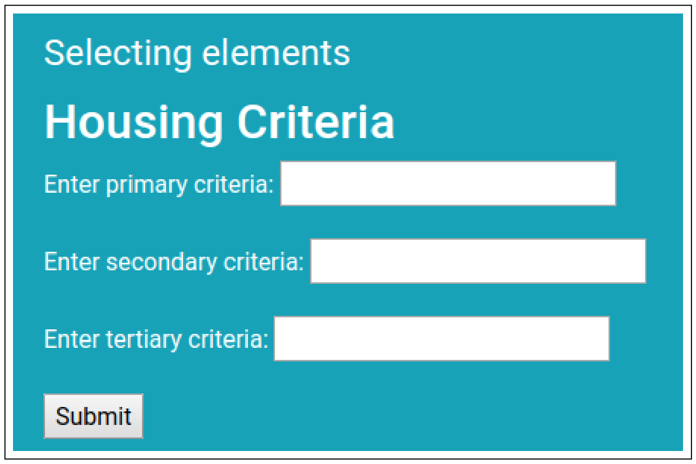
The data used to visualize zip codes for this project is the *NYC 311 Service Requests from 2010 to Present* dataset from NYC OpenData. The raw data contains 21.8 million rows and 36 columns. Each row is a record of every complaint from 2010 to the present, while the 36 columns describe the nature of the complaint with descriptors such as complaint type, date and time, street address, zip code of the recorded complaint, the relevant agency that will or has respond to the complaint, along with whether or not the complaint was addressed, and how long it took to address. This data can be downloaded directly from NYC OpenData in a number of formats including CSV, JSON, and GeoJSON, or called by the Socrata API as a dictionary (JSON). For this web-app, the Socrata API was used to access the most recent $500,000$ 311 calls. This number was chosen because it corresponds to just under 3 years of calls, and produced tolerable speeds for the web-app.

A. Feasibility

Prior to actually using the data for the web-app, its quality had to be ascertained. As shown in Fig. 1, the complaint types for close to 50% of the rows in the dataset are marked "Other". Further examination of the data reveals several rows with N/A values for complaint type, and a few columns where a majority of the rows have no value reported. This makes it clear that a significant amount of data cleanup is required. Additionally, it is useful that the rate of 311 complaints in each borough corresponds to the relative population of that borough. This was determined by comparing the population distributions of New York City in each borough with the distribution of complaints in the dataset. This ensures that zip code suggestions could be made reliably from the dataset without risk of bias. Fig. 3 illustrates this correlation.

B. Data Preprocessing

The challenge with our problem statement is picking zip codes which signify favorable neighborhoods using complaint data. Therefore, low frequency of incidence for a complaint will be a positive indicator for a given zip code. The rows with empty complaint type or empty zip code values were dropped. Additionally, we did not use any columns in the dataset except for the most relevant columns, including complaint type, complaint time, and the incident zip code. With these columns, we could determine which neighborhoods had which complaint types at what frequency and at what time. Thus, if a certain zip code had a high number of a certain type of complaint relative to its total number of complaints, then that neighborhood would not be ideal for someone trying to avoid that type of complaint. The reasoning for this is that a neighborhood with fewer complaints of a certain kind than the rest would indicate that the area was not affected by that nature of problem as severely as other zip codes. Such conditions would make a neighborhood less livable.


Selecting elements

Housing Criteria

Enter primary criteria:	
Enter secondary criteria:	
Enter tertiary criteria:	

Fig. 4: A screenshot of the Django form on the web-app homepage. Users input the complaint types in text fields.

IV. Methods

A. Fetching the data

The dataset is hosted on the Socrata database platform, which provides the Socrata Open Data API (SODA) to interact with the datasets on their Open Data Network. An API endpoint is required to access the dataset, without which access to the data is limited to a few thousand rows and is subject to throttling. Using an app key and the python package `sodapy`, an API call can be made to fetch the data, which is then stored as a `Pandas` dataframe. A Pandas dataframe was chosen because it is excellent at processing large amounts of data and is frequently used by data scientists. Socrata provides their own database querying language, `SoQL` (Socrata Querying Language), whose syntax is very similar to `SQL`. Rather than fetch the entire dataset and then filtering and trimming it as a dataframe, we found it easier and faster to fetch the data through SODA with an appropriately worded query to preemptively select the columns and rows we needed for the task.

B. Preprocessing the Dataframe

Once the data has been fetched with the API call and converted to a Pandas data frame, using the `.dropna()` method, the rows with N/A values for zip code and complaint type are dropped, and the dataframe index is reset to reflect linear numbering.

The choropleth map that we will use to depict liveability of a zip code essentially depicts density. Thus, for a given criteria input by the user, a dataframe is created to aggregate the number of 311 calls for that complaint type.

C. Deploying on Django

Django provides a class for creating forms. Through class inheritance a form for the main webpage was created. The user inputs 3 preferred criteria in a character field. The form method points to an `html` page for the form to be displayed

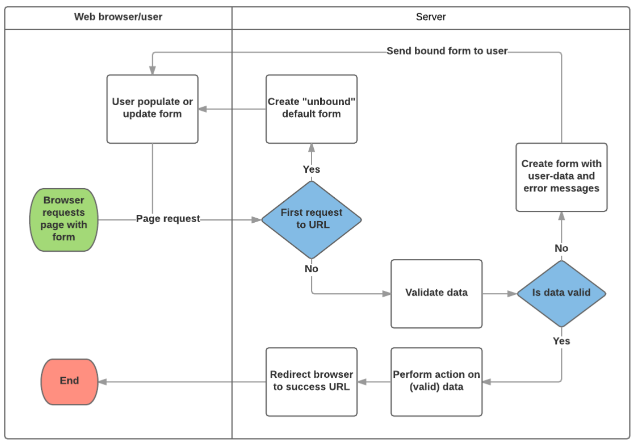

Web browser/user
Server
Send bound form to user
User populate or update form
Create "unbound" default form
Yes
Create form with user-data and error messages
Browser requests page with form
Page request
First request to URL
No
No
Validate data
Is data valid
Yes
End
Redirect browser to success URL
Perform action on (valid) data

Fig. 5: An overview of how form handling is performed and what role Django plays in the process. **Image credit:** Mozilla Developer Network Documentation.

via a path reference in the `urls` method of the web-app. Fig. 4 depicts the form with some `CSS` styling with `Bootstrap`.

An overview of how Django handles forms and how inputs are conducted within the framework is presented in Fig. 5.

D. *Visualization with Folium*

Folium is a `leaflet.js` based map visualization library for Python that uses `GeoJSON` data to display dynamic and interactive maps. In simple terms, the GeoJSON file outlines the polygons that make up regions on a map, which when given to a `folium` method, (such as `choropleth` in our case), it returns a map visualization based on arguments that dictate color and opacity, which are directly dependent on the quantity whose 'density' is to be depicted through the choropleth.

V. Experiments

A. *Experiment 1 - Fetching the Data*

Data handling presents significant challenges irrespective of the application area. Since our project is relatively lightweight, we committed to an experiment to measure the time benefits of fetching an entire dataset and then filtering it, versus using a querying language to preemptively filter the rows and columns we needed for the project. This experiment ties in nicely with the concepts learnt in class regarding the advantages of Database Querying paradigms and their necessity.

The experiment reveals a relatively constant time required to fully call the dataset from batch sizes 1 to 100,000. This time is from 40 to 60 seconds. After this point, the time required skyrockets to above 300 seconds. For calling the dataset with Socrata's SQL-like language "SoQL", the time requirements are far lower at less than a second until a batch size of 50,000. After this, the time requirements increase steadily to just under 10 seconds at a batch size of 500,000. It is clear that properly querying the dataset dramatically improves the speed of our

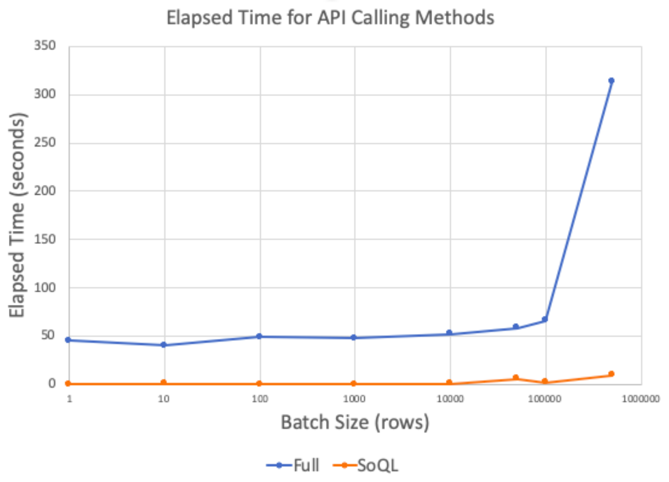


Fig. 6: A plot of elapsed time for different API calling methods. A dramatic increase in call time is observed between 100,000 and 500,000 a batch size. Batch size refers to the number of rows of data that are being called and saved to a Pandas dataframe.

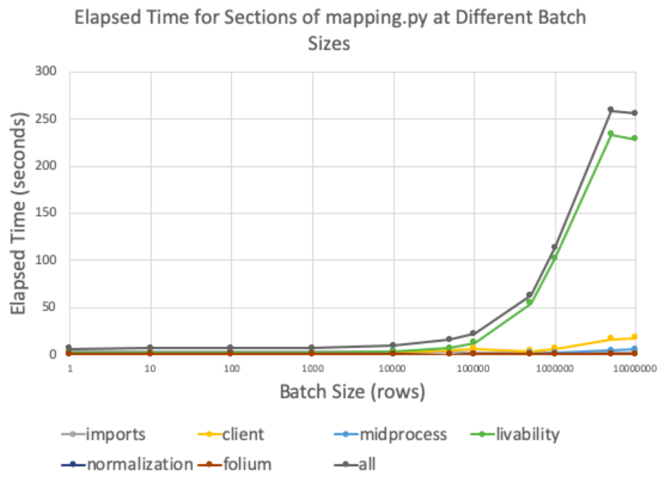


Fig. 7: A plot of elapsed time at the various sections of mapping.py for different batch sizes. The majority of the total time elapsed at higher batch sizes can be attributed to the 'livability' section. mapping.py is a python script that takes input from the django form at our homepage, calls the SODA API for the 311 data, processes the data, and produces the map HTML file. Batch size refers to the number of rows of data that are being called and saved to a pandas dataframe.

web-app, and it eliminates the need to store data that will eventually be filtered out anyway.

B. *Experiment 2 - Timing of the Mapping Script*

Here, the speed performance of our web-app was measured at different stages of mapping.py. The performance of these different modules was of concern because knowing the performance of each module allows more rigorous optimization efforts by identifying bottlenecks. The stages of mapping.py evaluated were ‘imports’, ‘client’, ‘midprocess’, ‘livability’, ‘normalization’, and ‘folium’, with ‘all’ being a sum of the individual stage times.

This experiment revealed that the majority of the time elapsed for ’all’ stages could be accounted for by the ’livability’ section. This section is a for-loop in mapping.py that iterated through different elements in the main dataframe. ’livability’ accounted for 89.4% of ’all’ section time at a batch size of 10,000,000.

VI. System Overview

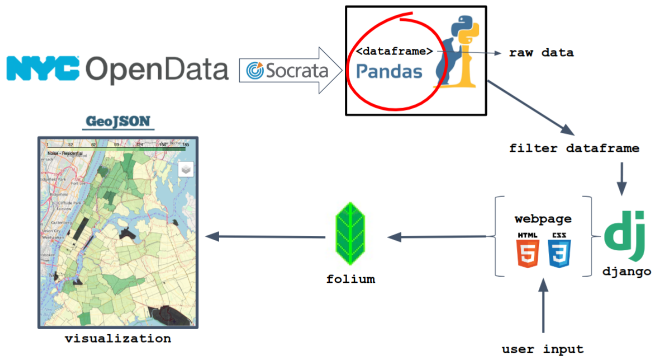


Fig. 8: The software architecture of the map visualization web-app. Data is fetched from the Socrata servers through the SODA API, queried using SoQL, passed as a pandas dataframe to the Django server wherein user input is taken to determine the complaint type to visualize on the choropleth map, which is rendered using Folium.

As explained in the previous section, there was an easy choice to be made about dependence on pandas to filter and rework the data. Using the Socrata Querying Language (SoQL) in conjunction with the sodapy library allowed us to fetch data much faster, while reducing the overhead from dealing with a large dataframe.

For the web framework, we had the option of using Flask to deploy the web-application. However, while Django is more complicated to set up, it provides neat templates, the options for database migration, and classes for common blocks such as forms, which we were able to take advantage of.

As for visualization, three possible modules could have been used. Bokeh, a python library for data visualization, which works well with pandas; plot.ly, an online graphical visualization library with support for python, and folium, a leaflet.js-based library. We chose folium since it integrates well with webpages, and has great support for GeoJSON inputs. This allowed for more detailed mapping.

VII. Conclusion

We discovered that the distributions of 311 complaint calls is strongly correlated with the population distribution of New York City. This property turned out to be useful since it directly ensures that the visualizations are strictly representative of actual neighborhood conditions in every borough of the city.

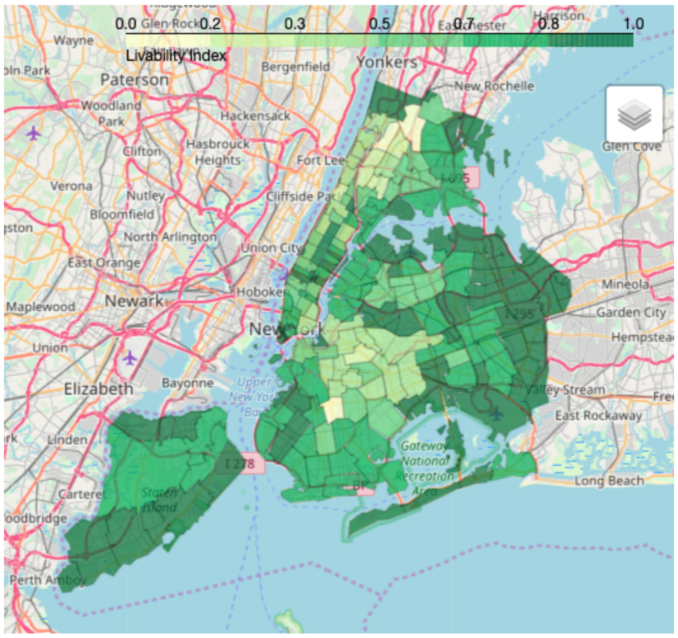


Fig. 9: The choropleth map visualizing the livability index for a user’s input.

The pros and cons of using Django against Flask as the web framework were closely evaluated. Ultimately, even though Django is aimed at full stack development, we chose to deploy the web-application using Django since it provides useful tools and templates that allowed us to streamline the design of the application. Additionally, Django automatically handles server side interactions, which allowed us to focus on the remaining aspects of the project.

To ensure that the functionality of the web-app was ready for a demo, the administrative and authorization aspects of the Django framework have not been configured. For real-world deployment, these would be essential first-steps. Since the computational demand for processing this data was not great, we chose not to deploy this on the cloud. However, a fully-fledged app would require greater distribution of computing and storage resources, for which an E2C instance from Amazon or a Dataproc cluster on Google Cloud would be essential.

VIII. Individual Contribution

	dso2119	plm2130
Data Exploration	60%	40%
Final Code	40%	60%
Report	50%	50%

TABLE I: A breakdown of contribution and effort for each individual towards this project.

REFERENCES

[1] S. Ejdemyr, "visreq," in *github.com*, 2018.

[2] M. Li, H. Yin, Y. Wang, F. Gao, and C. Cai, "An adaptive method for n-dimensional tensor factorization on sparse data processing," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, Dec 2017, pp. 2384–2389.

[3] R. Ibrahim and M. O. Shafiq, "Measurement and analysis of citizens requests to government for non-emergency services," in *2017 Twelfth International Conference on Digital Information Management (ICDIM)*, Sep. 2017, pp. 286–291.

[4] J. Legewie and M. Schaeffer, "Contested boundaries: Explaining where ethnoracial diversity provokes neighborhood conflict," in *AJS*, July 2016, vol. 122, pp. 125–161.

[5] S. Minkoff, "Nyc 311: A tract-level analysis of citizen–government contacting in new york city," in *Urban Affairs Review*, 2016, vol. 52, p. 211–246.

[6] F. Berhane, "Visualize_nyc311calls_with_kibana," in *github.com*, July 2017, vol. fissehab.