UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

3250 Foundations of Data Science

Course Project

City of Baltimore

911 Calls Data Analysis

Prepared by:

[Team A]

Adil Alkhateeb

Mohammad Ali

Asim Ali

Submitted on: Dec 8th 2017

Table of Contents

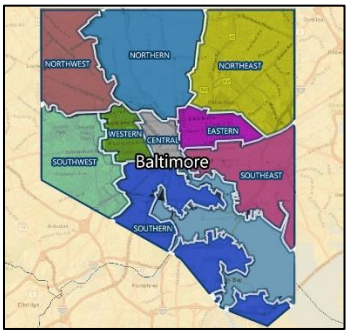
1. Introduction

This report was produced as a project requirement for the course "3250 Foundation of Data Science" at the University of Toronto, School of continuing studies. A real dataset of 911 calls recorded by the City of Baltimore was the subject of this data analysis project. This includes data preparation, analysis and visualization via graphical charts and interactive heat maps. The following sections describe the data preparation strategies followed, the analysis methodologies and results revealed using a generic data analysis platform developed for this exercise. References to Jupyter notebooks are denoted in the form of [NB#] and listed in the Appendix.

1.1. The City of Baltimore

Baltimore is the largest city in the U.S. state of Maryland, and is the 30th most populous city in the United States [i]. Baltimore was established by the Constitution of Maryland and is an independent city that is not part of any county. With a population over 600,000 in 2015, Baltimore is the largest independent city in the United States.

The city of Baltimore is divided into nine major neighborhoods (districts) which will be highlighted as appropriate throughout the Data Analysis section.



Crime in Baltimore is generally concentrated in areas high in poverty and has been above the national average for many years. Overall reported crimes have dropped by 60% from the mid-1990s to the mid-2010s, but homicide rates remain high and exceed the national average. Baltimore's homicides in 2015 represented the highest homicide rate in the city's recorded history and the second-highest for U.S.[i] Baltimore's 911 Communications Center is perceived to be very critical given the city's crime records.

1.2. Dataset Overview

The dataset is a set of actual 911 calls that was kindly made available by the City of Baltimore. The data gets updated daily, therefore the subject dataset was downloaded from Kaggle[ii] representing records of 2.8M calls made to the 911 Communication Center for the period 2015-01-01 to 2017-08-29.

The dataset records time, priority, reason for calls under description and call location address with coordinates. Callers' numbers are masked to protect their privacy, therefore call numbers have no significant value except as a "call ID" sort to speak. Below is a snapshot of the raw data:

	callDateTime	priority	district	description	callNumber	incidentLocation	location
1	2015-07-13 10:47:00	Medium	CD	911/NO VOICE	P151941003	600 E FAYETTE ST	(39.2906737,-76.6071600)
2	2015-07-13 10:42:00	Medium	CD	911/NO VOICE	P151941004	200 E BALTIMORE ST	(39.2898910,-76.6120720)
3	2015-07-13 10:45:00	Low	CD	PRKG COMPLAINT	P151941005	800 PARK AV	(39.2985163,-76.6184754)

2. Data Preparation

The dataset had several issues with the records that needed to be addressed prior to analysis. *callDateTime* was stored as text, therefore it was necessary to parse this column into *dataTime* type and use it as index to convert the dataset into a TimeSeries set.

The following data were deemed missing and necessary assumptions/procedures were required to fill the gaps:

Priority: 6655 missing records as '*NaNs'.*

Location: 82532 missing coordinates stored as '*(,)'.*

Description: The reason of calls, *Description* column, required intensive cleaning to fix spelling and case errors along with invalid numbers only descriptions that needed to be flagged. The following sections describe in details the data preparation process for each of the above.

2.1. Priority Missing Data

The very first step in cleaning priority missing data was to examine the type of priorities and their counts in the dataset. Below is a summary of the pre-cleaning priorities:

Priority	Count
Medium	1399431
Low	636380
High	434022
Non-Emergency	321619
Emergency	1004
Out of Service	803
NaN	***6655***

The initial approach was to Forward Fill and Back Fill missing priorities based on their adjacent records, however that was too random and would not add any value in the analysis. Therefore a different more logical approach had to be followed.

The filling strategy was to examine the priorities for all calls descriptions, and then use the priorities of the same or similar descriptions to fill the missing priorities. In order to achieve that, all data was grouped by *description* and *priority* columns and counted. Then the highest priority count for each description was identified and used as a key to fill the missing priorities.

In order to illustrate this further, below is a code and output sample for the grouping and key generation zoomed down to 'BURGLARY' calls for illustration purposes:

```
In [7]: grouped = data.groupby(['description','priority']).count()[['callNumber']].loc[['BURGLARY']]

In [8]: grouped
Out[8]:
```

description	priority	callNumber
BURGLARY	High	80
	Low	3
	Medium	58416
	Out of Service	15

```
In [9]: key=grouped.loc[grouped.groupby(level=0).idxmax()['callNumber'].values]

In [10]: key
Out[10]:
```

description	priority	callNumber
BURGLARY	Medium	58416

The Key dataframe was then converted into a Series with *description* as index and *priority* as data so that it can be used to fill missing priorities as following:

```
In [14]: key = key.reset_index().drop('callNumber',axis=1).set_index('description')['priority'] #key needs to be a Series

In [15]: mask = data.priority.isnull()

In [16]: data.loc[mask,'priority'] = key.loc[data.loc[mask,'description']].values

In [17]: data.priority.isnull().value_counts()
Out[17]: False    2799910
         True           4
         Name: priority, dtype: int64
```

This has successfully filled almost all missing priorities with the most common priority per description. There were 4 remaining records that had unique descriptions which had no key entry, therefore manual analysis per record was necessary to choose the best fitting priority based on similar descriptions. That was accomplished by a substring analysis using *Series.str.contains()* method. Cleaning code details can be found under [NB1]. Below is the final priority counts for the cleaned dataset:

Priority	Count
Medium	1403898
Low	637275
High	435239
Non-Emergency	321695
Emergency	1004
Out of Service	803
NaN	***0***

2.2. Descriptions Handling

Description column has undergone an intensive cleaning process to eliminate invalid characters, repeated descriptions with different spellings, extra white space and inconsistent letter cases [NB2]. All non-alphanumeric characters were removed and all descriptions were converted into upper case, then an extensive dictionary was produced to correct spelling errors as much as possible. Finally all data were stripped off wrapping white spaces and non-alpha descriptions were replaced with `'NO DESC'` values. Prior to final clean dataset export, five records with description `'TEST Call'` were dropped.

The data analysis was conducted moving forward focusing on all descriptions repeated over 1000 times due to their significance given the large dataset in hand, the total descriptions in focus were 115 major descriptions, below is a snapshot of the cleaned top 10:

```
In [36]: grouped[grouped.priority>1000].head(n=10)
```

Out[36]:

	description	callDateTime	priority	district	callNumber	incidentLocation	location
221	911 NO VOICE	489758	489752	489758	489758	489758	489758
1821	DISORDERLY	250189	249481	250189	250189	250189	250189
5746	TRAFFIC STOP	167912	167912	167912	167912	167912	167912
1508	COMMON ASSAULT	125568	123685	125568	125568	125568	125568
676	AUTO ACCIDENT	116050	115980	116050	116050	116050	116050
3851	NARCOTICSOUTSIDE	99549	99455	99549	99549	99549	99549
4117	OTHER	91515	91504	91515	91515	91515	91515
5349	SILENT ALARM	84249	84163	84249	84249	84249	84249
2248	FAMILY DISTURB	72673	72470	72673	72673	72673	72673
5040	REPAIRS SERVICE	67050	67050	67050	67050	67050	67050

2.3. Missing Coordinates and Geocoding

The objective of handling missing coordinates was to examine the address of each record and resolve for valid longitude and latitude values. Google's Maps geocoding API use was necessary to accomplish this task. Therefore, *pygeocoder*, a python wrapper for the above API based JSON[iii], was used to geocode, reverse geocode, validate and format addresses.

The initial approach was to validate all 2.8M addresses and coordinates. The subject package was installed and the initial test records had their addresses validated successfully as following [NB3]:

callDateTime	priority	district	description	callNumber	incidentLocation	location	address	zipCode
2015-07-13 10:47:00	Medium	CD	911/NO VOICE	P151941003	600 E FAYETTE ST	(39.2906737,-76.6071600)	560-618 E Fayette St, Baltimore, MD 21202, USA	21202
2015-07-13 10:42:00	Medium	CD	911/NO VOICE	P151941004	200 E BALTIMORE ST	(39.2898910,-76.6120720)	200 E Baltimore St, Baltimore, MD 21202, USA	21202

Upon running the *pygeocoder* package on the entire dataset, it was discovered that Google imposes requests and throughput restrictions on using their subject API[iv]. Google offers 2,500 free requests only per day per IP with a limitation on the number of requests per second, and charges $0.50 USD per 1,000 additional requests up to 100,000 max daily. A completely different approach was adopted to try to overcome this show stopper, and that was to use Google Maps Embed API web calls which offers unlimited free usage. The trick was to use *urllib* package and initiate web requests to Google Maps, then analyze and slice the response string to capture the coordinates [NB4]. This has worked well too, but again Google deploys robots detectors and blocks requests coming from scripts rather than actual human interactions.

Given the above the plan of validating all the addresses was deemed infeasible and a new workaround was necessary. It was decided that missing coordinates (82k records) will be addressed instead of validating the entire dataset. A robust and persistent complex python script was carefully developed to utilize Google Maps geocoding API again using *pygeocoder* [NB5]. The code was designed to break the missing dataset into 82 patches of 1000 records each then run each patch separately. It was engineered in a robust way to overcome throughput limitations per second, it keeps retrying via iterations and

carefully places short sleeps before retrying until the daily limit is reached for the used IP. Each run would take around 5-15 mins to complete with the following output:

```
Patch: 10000-10999
Starting missing:  1000
Ending missing:  0
Resolved coordinates:  1000
Success rate: % 92.10
```

Once the daily limit is reached for the used IP, a manual IP change was required by shifting to any other available network (public, Mobile Data, Work, Home …etc) to be able to run the script again. The set of patches was distributed among the team members and the script was used around the clock for one week until 90% of the missing coordinates with valid addresses in Baltimore were resolved successfully. The very last step was to consolidate the coordinates and split them into separate *lat* and *lon* columns for easier visualization handling [NB6].

The final stage was to bring all the cleaned data together into a clean dataset that can be used for the data analysis process [NB7].

3. Data Analysis

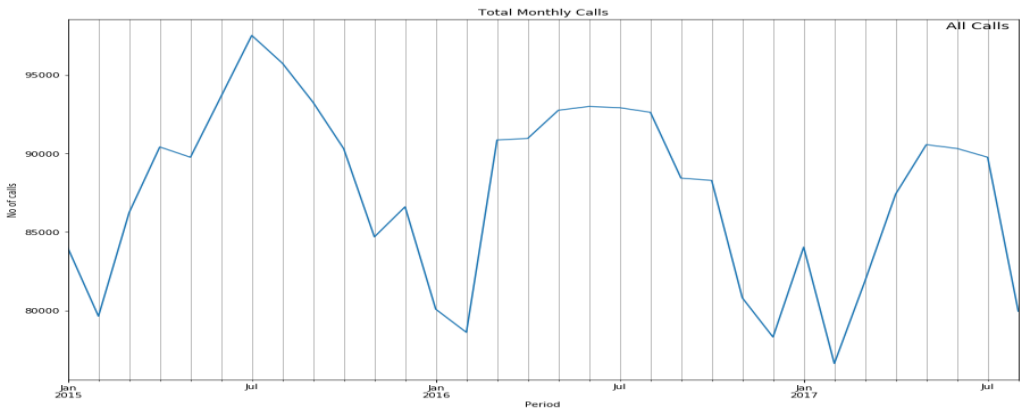
The data preparation phase has produced a clean TimeSeries dataset that has been the subject of a thorough analysis as will be discussed in the following sections:

3.1. High-level Analysis

The total post data preparation calls has netted at 2,799,860 spanning through the period from Jan 2015 to Aug 2017 [NB8]. Below is a snapshot from the cleaned dataset:

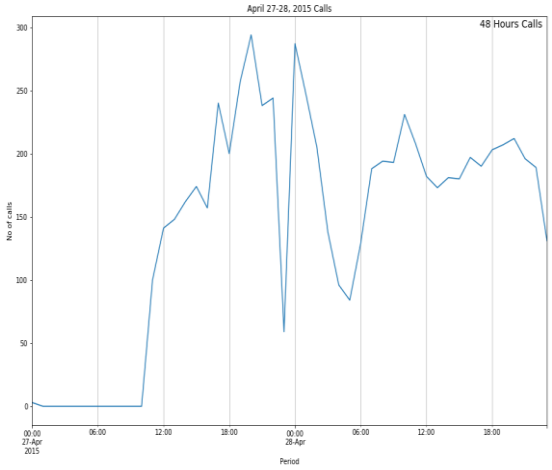
callDateTime	priority	district	description	callNumber	incidentLocation	location	lat	lon
2015-07-13 10:47:00	Medium	CD	911 NO VOICE	P151941003	600 E FAYETTE ST	(39.2906737,-76.6071600)	39.290674	-76.607160
2015-07-13 10:42:00	Medium	CD	911 NO VOICE	P151941004	200 E BALTIMORE ST	(39.2898910,-76.6120720)	39.289891	-76.612072
2015-07-13 10:45:00	Low	CD	PRKG COMPLAINT	P151941005	800 PARK AV	(39.2985163,-76.6184754)	39.298516	-76.618475

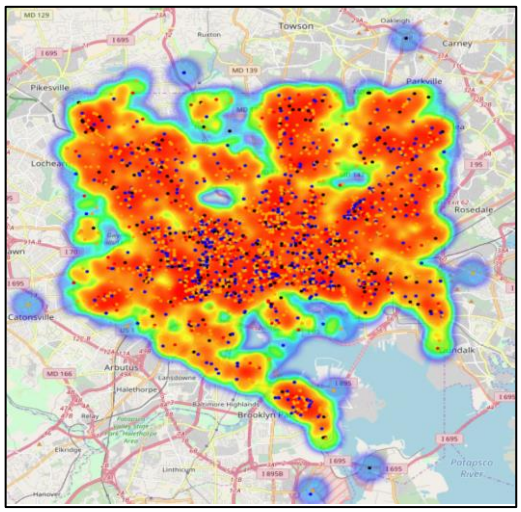
By visualizing the calls counts on monthly basis, below gives an impression on how calls were flowing in throughout the analyzed period:



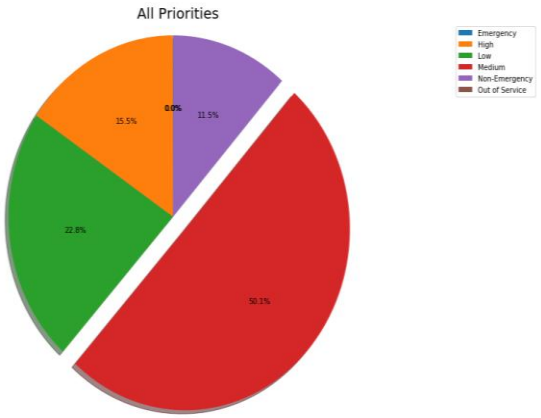
The number of calls has averaged at 87,496 calls per month, 2,881 calls per day, peaking on Fridays and slowing down the most on Sundays. The total monthly calls has peaked in the month of July 2015 at a record high of 97,511 calls.

Interestingly, April 28th 2015 had the highest peak throughout the dataset at a record of 4,441 calls. By cross checking this peak day online, it was found that the City of Baltimore was on violent riots after the Apr 27th funeral of Freddie Gray who was a young Black American man arrested by the Baltimore Police and died from his injuries allegedly caused by the arresting officers[v] [vi]. Below is an overview on how the calls were flowing in during the riots along with a heat map of the peak day [Map1].



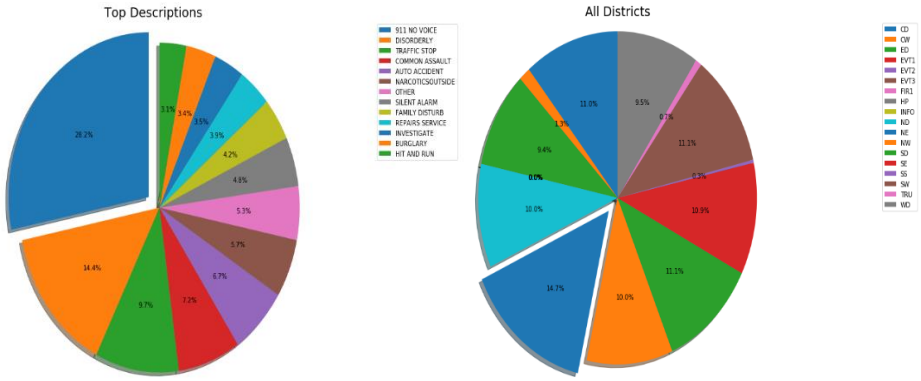


Calls were flowing in different priorities with 50% recorded as 'Medium', 23% as 'Low' and 16% as 'High'. Below pie chart illustrates the priority mix:



Interestingly, the top reported description was '911 No Voice' (28%), followed by 'Disorderly' at 14%, 'Traffic Stop' 10%, 'Common Assault' at 7%, 'Auto Accident' at 6% and 'Narcotics outside' at 6%.

The districts with the highest calls were 'Northeast' 15%, 'Central' 11%, 'Southwest' 11%, 'Southern' 11%, 'Southeast' 10% and 'Northern' 10%.

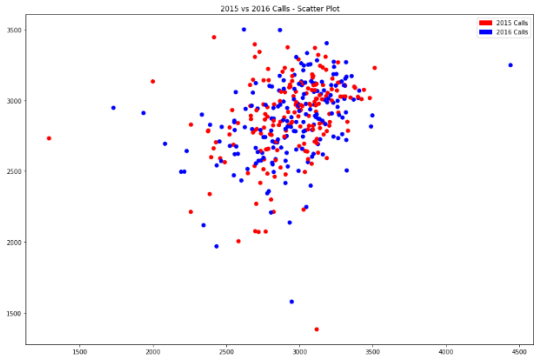


3.2. Trend Analysis and Correlation

A broader annual analysis was conducted to determine the similarity between the two full years covered by the dataset, 2015 and 2016 [NB9]. Calls were resampled on the day-of-year basis and visualized on top of each other to detect their trend properties:

At first sight, calls appear to reflect a seasonal trend. Calls are trending positively from winter all the way to mid-summer, then starts to gradually drop from fall until Christmas. A further normalized analysis was conducted to smooth out high calls oscillation by taking the rolling mean of the calls count per day at a window of 30 days (shown above). The results supported the seasonal trend finding and was consistent for both years.

The calls were found to have a barely noticeable correlation that is highly influenced by the seasonal trend only as reflected on the below scatter plot:



3.3. Analysis Framework

The cleaned dataset has around 6,539 unique reported problems '*description*'. Therefore it was necessary to develop an Analysis Framework that accepts a text as description search keyword criteria, and conducts an in-depth analysis producing detailed statistics, trends, charts, and maps visualizations [NB10].

The framework is equipped with python scripts to load, search and produce results based on a given keyword. For example, search for 'assault' keyword would produce full analysis on all assault records like 'Common Assault', 'Other Assault', 'Any Assaults' etc.

```
assault_calls = calls.loc[calls.description.str.contains('ASSAULT')==True]
```

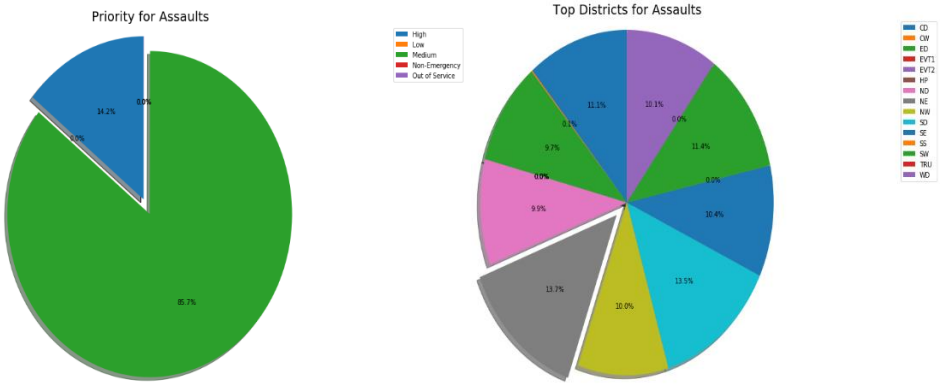
The results set is then grouped and analyzed by priorities and districts, and then resampled on daily basis to analyze weekdays trends and on monthly basis to produce visual analysis charts using the following pre-set charts configuration functions:

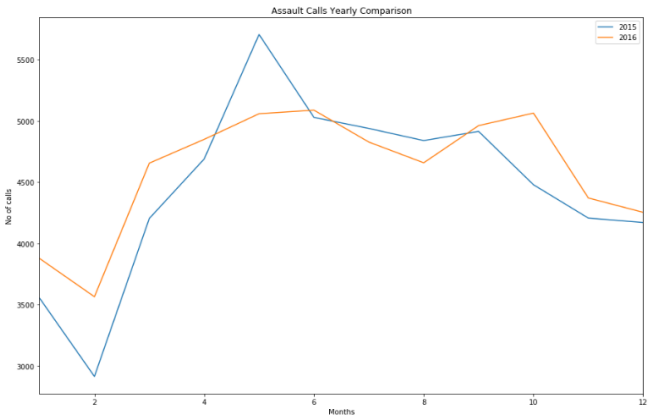
a. *DrawPieChart()* - Helps in drawing pie charts and calculating maximum percentage to 'explode' the highest value on the pie chart.
b. *SetChartProperties()* - Pre-set chart properties for smoother and neater output.
c. *DrawAutoCorrelation()* - Draws line charts along with autocorrelation analysis.

3.3.1. Assault Analysis

Based on the above developed Analysis Framework, an analysis was conducted on all Assault calls and revealed the following [NB11]:

- The total number of assaults calls were 146,280 and contributed to 5.22% of total calls.
- Analyzed calls included several assault types such as COMMON ASSAULT, AGGRAV ASSAULT, DOMESTIC ASSAULT and SEXUAL ASSAULT.
- The assault calls priorities were reported 86% as 'Medium' and 14% as 'High'.
- The highest Assault calls came from the 'North East' district (14%) followed by 'South District' (13%), 'South West (11%) and 'Central District' (11%).



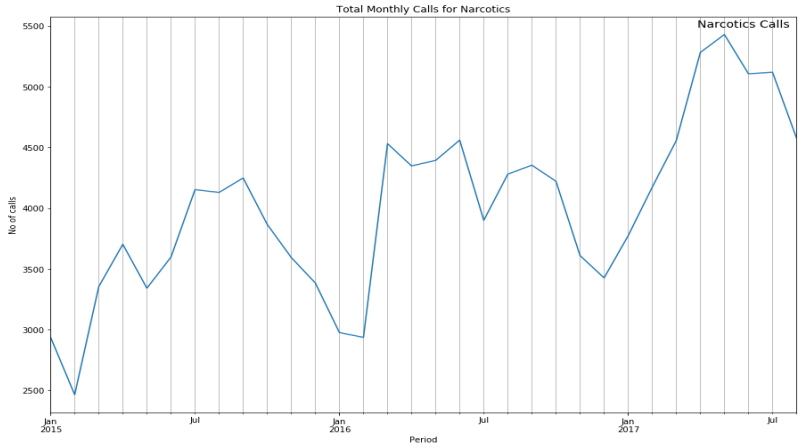


- Assault calls have surged in May 2015 as per above chart, a search online found supporting news naming this month as Baltimore's deadliest month since 1970s[vii], above is a heat map of Baltimore that reflects assault calls during the month of May 2015 [Map2].

3.3.2. Narcotics Analysis

Based on the above developed Analysis Framework, an analysis was conducted on all reported calls related to Narcotics and revealed the following [NB12]:

- The total number of Narcotics calls were 128,332 and contributed to 4.58% of total calls.
- Analyzed calls included several Narcotics observed indoors and outdoors.
- Almost all of the Narcotics calls priorities were reported as 'Medium'.
- The highest Narcotics reports were from the 'West District' (19%) followed by 'North West' (14%), 'South District' (13%) and 'South West' (12%).
- Narcotics calls have noticeably increased starting Jan 2017 and peaking by May 2017, the chart below reflect the overall seasonal trend but also reflect the spike in 2017.

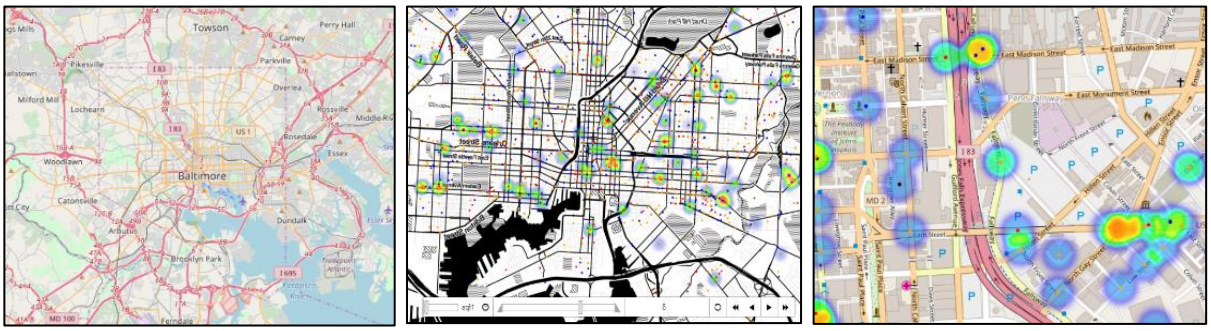


- A search over the news outlets revealed supporting article of increased drugs related incidents and the police operations to seize illegal drugs on the streets.[viii]

3.4. Maps Geo-Plotting

Visualizing 911 calls locations over Baltimore's map was vital to complement the dataset analysis. *Basemap* package was initially used after a lengthy process to get it installed as there is no official release compatible with python 3.6. However, the maps quality and functionality was not adequate to fulfill the visualization expectations. Therefore another solution was needed to plot better maps quality.

The *folium* package was used at the end. It brings python strength and *Leaflet.js* mapping library together and makes is possible to visualize data over interactive *Leaflet* maps[ix]. *OpenStreetMap* and *StamenToner* tiles were used along with heat maps and time-lapse maps to visualize data [NB13].



Worth mentioning that Jupyter was unable to render maps with large data points due to low *iopub_data_rate_limit* value in Jupyter config file. To overcome this, it was necessary to set that parameter to a higher limit by starting Jupyter using the following setup:

```
jupyter notebook --NotebookApp.iopub_data_rate_limit=10000000
```

4. Conclusion

The 911 operations at the City of Baltimore's communication center is clearly very critical given the City's Crime rates and the supporting analysis results in this report.

Data capturing efficiency can be enhanced if the agents would have preset descriptions and automated address intake which can improve future analysis and produce more useful insights.

Further analysis can be conducted on the given dataset to reveal more incidents types such as theft, accidents, or any other desired problem category by utilizing the developed Analysis Framework in this project.

Given the large volume of calls recorded by the dataset, this analysis can be extended to predict a forecast of 2018 calls on various levels to support resources planning at the communication center or on the ground utilizing the geo-tagging functionalities implemented in this project.

5. Appendix: Jupyter Notebooks

NB1: Priority Solution_Final.ipynb
NB2: Description_Solution_Final.ipynb
NB3: Geocoding - Full Address validations.ipynb
NB4: Getting Coordinates using Google Maps URL requests.ipynb
NB5: Get Coordinates Script v2.ipynb
NB6: Coordinates Consolidation Final.ipynb
NB7: Analysis_Data_Consolidation_Final.ipynb
NB8: High_level_Analysis.ipynb
NB9: Trend Analysis with Rolling Means.ipynb
NB10: Analysis_Framework.ipynb
NB11: Story1-Assault-Analysis.ipynb
NB12: Story2-Narcotics-Analysis.ipynb
NB13: Geo plotting v3.ipynb
Map1: 2015-04-28_Heatmap.html
Map2: May2015_Assaults.html

6. References

[i] Baltimore – Wikipedia: https://en.wikipedia.org/wiki/Baltimore

[ii] Baltimore 911 Calls – kaggle: https://www.kaggle.com/sohier/baltimore-911-calls

[iii] pygeocoder 1.2.5 – python: https://pypi.python.org/pypi/pygeocoder/

[iv] Google Maps APIs, Pricing and Plans: https://developers.google.com/maps/pricing-and-plans/

[v] Wikipedia - Death of Freddie Gray: https://en.wikipedia.org/wiki/Death_of_Freddie_Gray

[vi] CNN – Baltimore riots: http://www.cnn.com/2015/04/27/us/baltimore-unrest/index.html

[vii] The Baltimore Sun: http://www.baltimoresun.com/news/maryland/crime/bal-may-2015-baltimores-deadliest-month-in-15-years-sg-storygallery.html

[viii] BCS Baltimore: http://baltimore.cbslocal.com/2017/06/02/9-million-heroin-seized

[ix] Folium 0.2.0 documentation: http://folium.readthedocs.io/en/latest/index.html