Crimes in Boston

I've used Crimes in boston & Boston public schools dataset from Kaggle

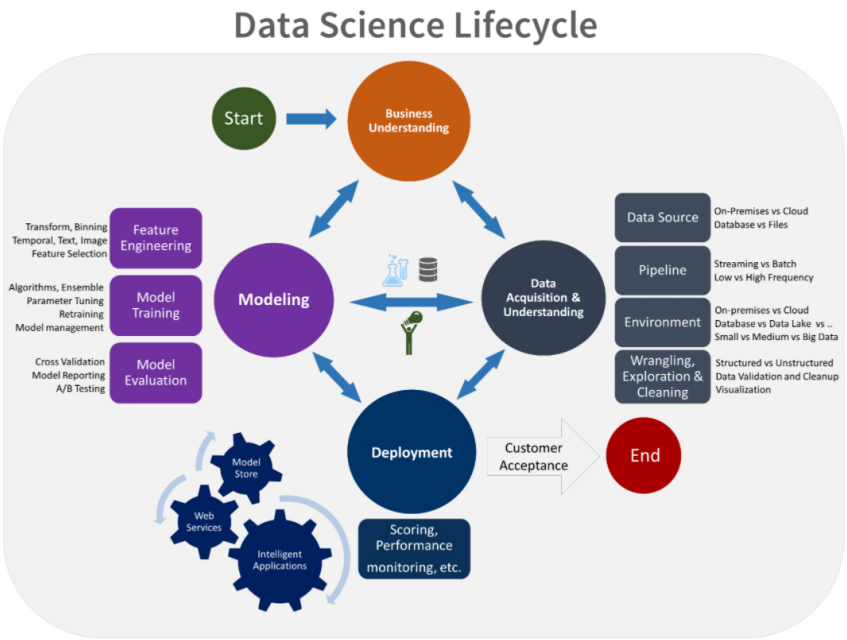
by Nut Chukamphaeng 60070134

Download data from API

In [1]:
```
# install API kaggle
# !pip install kaggle

# move API credentials to environment
# !cp ./utils/kaggle.json ~/.kaggle/kaggle.json

# download dataset from https://www.kaggle.com/AnalyzeBoston/crimes-in-boston
# !kaggle datasets download -d AnalyzeBoston/crimes-in-boston

# download dataset from https://www.kaggle.com/crawford/boston-public-schools
# !kaggle datasets download -d crawford/boston-public-schools
```

In [2]:
```
# unzip files
# !unzip crimes-in-boston.zip -d ./data/
# !unzip boston-public-schools.zip -d ./data/

# remove .zip
# !rm -rf boston-public-schools.zip crimes-in-boston.zip

# add permission
# !chmod 777 ./data/*
```

Note I've used unix system for develop this project. If you run with windows system something not working.



This is a lifecycle process of data science. I'll following this process.

Import libraries

In [41]:
```
import pandas as pd
import numpy as np
# !pip install folium # for install if you haven't package
import folium
# !pip install plotly # for install if you haven't package
import plotly.graph_objs as go
import matplotlib.pyplot as plt
import math

from folium import plugins
from plotly.offline import iplot
from glob import glob
from collections import Counter
from pprint import pprint
from IPython.core.interactiveshell import InteractiveShell

from utils.helper import countNull # helper function
from utils.constants import BOSTON_LAT, BOSTON_LONG, COLORS # constants

InteractiveShell.ast_node_interactivity = 'all'
```

If some package not working, You can install with comment command in above cell.

In [42]:
```
# get all path of data
paths = glob('./data/*.csv')
paths
```

Out[42]:
```
['./data\\crime.csv',
 './data\\offense_codes.csv',
 './data\\Public_Schools.csv']
```

In [43]:
```
# read csv
df_crime  = pd.read_csv(paths[0], encoding="latin") # select index of `crime.csv`
df_school = pd.read_csv(paths[2], encoding="latin") # select index of `Public_Schools.csv`
```

Business Understanding & Data Acquisition

In [44]:
```
# overviews of data
df_crime.head(3)
print("Size:", len(df_crime))
df_school.head(3)
print("Size:", len(df_school))
```

Out[44]:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	SHOOTING	OCCURRED_ON_DATE
0	I182070945	619	Larceny	LARCENY ALL OTHERS	D14	808	NaN	2018-09-02 13:00:00
1	I182070943	1402	Vandalism	VANDALISM	C11	347	NaN	2018-08-21 00:00:00
2	I182070941	3410	Towed	TOWED MOTOR VEHICLE	D4	151	NaN	2018-09-03 19:27:00

```
Size: 319073
```

Out[44]:

	ï»¿X	Y	OBJECTID_1	OBJECTID	BLDG_ID	BLDG_NAME	ADDRESS	CITY	ZIPCODE	CSP_SCH_ID	...	SCH_NAME	SCH_LABEL	SCH_TY
0	-71.004121	42.388799	1	1	1	Guild Bldg	195 Leyden Street	East Boston	2128	4061	...	Guild Elementary	Guild	
1	-71.030480	42.378545	2	2	3	Kennedy, P Bldg	343 Saratoga Street	East Boston	2128	4541	...	Kennedy Patrick Elem	PJ Kennedy	
2	-71.033891	42.375279	3	3	4	Otis Bldg	218 Marion Street	East Boston	2128	4322	...	Otis Elementary	Otis	

3 rows × 21 columns

```
Size: 131
```

Understanding crime dataset

This dataset have 319073 records.

- **INCIDENT_NUMBER**: incident id.
- **OFFENSE_CODE**: offense id.
- **OFFENSE_CODE_GROUP**: type of offense.
- **OFFENSE_DESCRIPTION**: description of offense.
- **DISTRICT**: district that crime was happended.
- **REPORTING_AREA**: area code that reported this crime.
- **SHOOTING**: this crime has shooting or not.
- **OCCURRED_ON_DATE**: occur time.
- **YEAR**: ...
- **MONTH**: ...
- **DAY_OF_WEEK**: ...
- **HOUR**: ...
- **UCR_PART**: UCR is the Crime Index.
- **STREET**: ...
- **Lat**: latitude of this crime.
- **Long**: longtitude of this crime.
- **Location**: pair of latitude and longtitude.

In [45]:
```
# range of data
df_crime['OCCURRED_ON_DATE'].min(), df_crime['OCCURRED_ON_DATE'].max()
```

Out[45]:
```
('2015-06-15 00:00:00', '2018-09-03 21:25:00')
```

In [46]:
```
# count null for each columns
countNull(df_crime)
```

Out[46]:

	number_of_null(%)
SHOOTING	99.680637
Long	6.267845
Lat	6.267845
STREET	3.407057
DISTRICT	0.553165
UCR_PART	0.028207
INCIDENT_NUMBER	0.000000
DAY_OF_WEEK	0.000000
HOUR	0.000000
YEAR	0.000000
MONTH	0.000000
OFFENSE_CODE	0.000000
OCCURRED_ON_DATE	0.000000
REPORTING_AREA	0.000000
OFFENSE_DESCRIPTION	0.000000
OFFENSE_CODE_GROUP	0.000000
Location	0.000000

Wow! column `SHOOTING` have null up to **~99%** from **319073** records. I think we need to drop this column.

We can't handle it.

In [47]:
```
# drop column SHOOTING
df_crime.drop('SHOOTING', axis=1, inplace=True)
```

In [48]:
```
df_crime.describe()
```

Out[48]:

	OFFENSE_CODE	YEAR	MONTH	HOUR	Lat	Long
count	319073.000000	319073.000000	319073.000000	319073.000000	299074.000000	299074.000000
mean	2317.546956	2016.560586	6.609719	13.118205	42.214381	-70.908272
std	1185.285543	0.996344	3.273691	6.294205	2.159766	3.493618
min	111.000000	2015.000000	1.000000	0.000000	-1.000000	-71.178674
25%	1001.000000	2016.000000	4.000000	9.000000	42.297442	-71.097135
50%	2907.000000	2017.000000	7.000000	14.000000	42.325538	-71.077524
75%	3201.000000	2017.000000	9.000000	18.000000	42.348624	-71.062467
max	3831.000000	2018.000000	12.000000	23.000000	42.395042	-1.000000

You can see unusual **min** `Lat` and **max** `Long`, that position not in boston.

I think we need to filter to range of boston **+- 0.5**.

In [49]:
```
BOSTON_LAT, BOSTON_LONG # lat & long of boston
```

Out[49]:
```
(42.3601, -71.0589)
```

In [50]:
```
# filter lat & long
df_crime = df_crime[(df_crime['Lat'] < BOSTON_LAT+0.5) & \
                    (BOSTON_LAT-0.5 < df_crime['Lat']) & \
```

```
                    (BOSTON_LONG+0.5 > df_crime['Long']) & \
                    (df_crime['Long'] > BOSTON_LONG-0.5)]
```

In [51]:
```
df_crime.describe()
```

Out[51]:

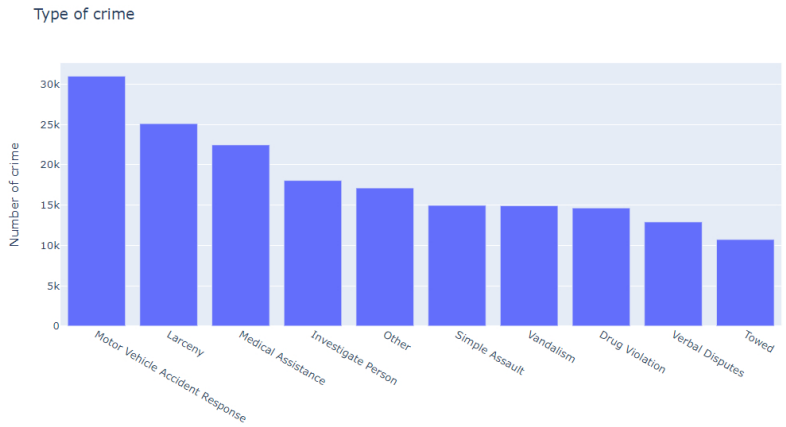
	OFFENSE_CODE	YEAR	MONTH	HOUR	Lat	Long
count	298329.000000	298329.000000	298329.000000	298329.000000	298329.000000	298329.000000
mean	2296.293143	2016.553547	6.613229	13.125352	42.322298	-71.082850
std	1183.110778	1.000330	3.278119	6.278917	0.031881	0.029771
min	111.000000	2015.000000	1.000000	0.000000	42.232413	-71.178674
25%	802.000000	2016.000000	4.000000	9.000000	42.297555	-71.097193
50%	2907.000000	2017.000000	7.000000	14.000000	42.325610	-71.077562
75%	3201.000000	2017.000000	9.000000	18.000000	42.348624	-71.062563
max	3831.000000	2018.000000	12.000000	23.000000	42.395042	-70.963676

Now! It's okay.

Most types of crime

In [52]:
```
# count crime type
type_crime_counts = np.array(Counter(df_crime['OFFENSE_CODE_GROUP']).most_common())
```

In [53]:
```
topN = 10

data = [
    go.Bar(x=type_crime_counts[:topN, 0], y=type_crime_counts[:topN, 1])
]

layout = go.Layout(
    title='Type of crime',
    yaxis=dict(
        title='Number of crime'
    )
)

fig = go.Figure(data=data, layout=layout)

iplot(fig)
```



In the figure from above, We'll see `Motor Vehicle Accident Response` is the best crime in boston.

How many type of UCR **(crime index)**.

In [54]:
```
# count for each type
Counter(df_crime['UCR_PART'])
```

Out[54]:
```
Counter({'Part One': 58912,
         'Part Two': 91076,
         'Part Three': 147100,
         'Other': 1151,
         nan: 90})
```

Top 4 `OFFENSE_CODE_GROUP` for each `UCR_PART`

In [55]:
```
print("UCR I:  ", Counter(df_crime[df_crime['UCR_PART'] == 'Part One']['OFFENSE_CODE_GROUP']).most_common(4))
print("UCR II: ", Counter(df_crime[df_crime['UCR_PART'] == 'Part Two']['OFFENSE_CODE_GROUP']).most_common(4))
print("UCR III:", Counter(df_crime[df_crime['UCR_PART'] == 'Part Three']['OFFENSE_CODE_GROUP']).most_common(4))
print("Other:  ", Counter(df_crime[df_crime['UCR_PART'] == 'Other']['OFFENSE_CODE_GROUP']).most_common(4))
```

```
UCR I:   [('Larceny', 25102), ('Larceny From Motor Vehicle', 10285), ('Aggravated Assault', 7267), ('Residential Burglary', 557
5)]
UCR II:  [('Other', 16160), ('Simple Assault', 14920), ('Vandalism', 14880), ('Drug Violation', 14596)]
UCR III: [('Motor Vehicle Accident Response', 31015), ('Medical Assistance', 22451), ('Investigate Person', 18018), ('Verbal Di
sputes', 12951)]
Other:   [('Auto Theft Recovery', 977), ('Arson', 91), ('License Plate Related Incidents', 64), ('Other', 10)]
```

Visualize map for each `UCR_PART`

In [56]:
```
# define color dict UCR_PART

UCR_colors_dict = {
    "Part One": "red",
    "Part Two": "blue",
    "Part Three": "green",
    "Other": "black"
}
```

Part I (RED): is the most serious crimes.

Part II (BLUE): is the second serious crimes.

Part III (GREEN): is the third serious crimes.

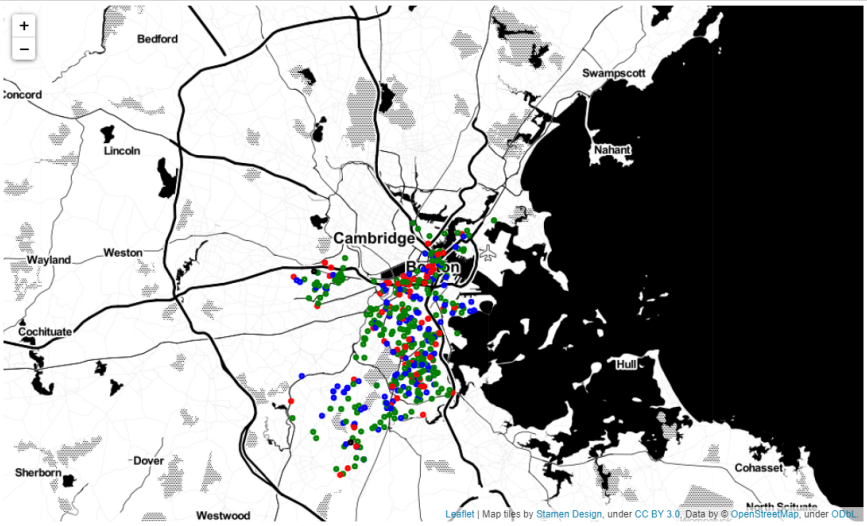
Reference

sample 500 records for visualize map.

In [57]:
```
m = folium.Map(location=[BOSTON_LAT, BOSTON_LONG], zoom_start=11, tiles='Stamen Toner',)

# We should to use apply method instead of manual loop, becuase apply method is faster than manual loop.
```

```
def toCircle(x):
    return folium.Circle(
        radius=100,
        location=[x['Lat'], x['Long']],
        popup=x['UCR_PART'],
        color=UCR_colors_dict[x['UCR_PART']]
    ).add_to(m)

_ = df_crime.dropna().head(500).apply(toCircle, axis=1)
m
```

Out[57]:



In the figure from above, you can't get any insight from data.

We need to focus some `UCR_PART` or `OFFENSE_CODE_GROUP` for easier to capture some insight.

First, I try to focus on `OFFENSE_CODE_GROUP` and drill-down to `Drug Violation` group.

In [58]:
```
df_drug = df_crime[df_crime['OFFENSE_CODE_GROUP'] == 'Drug Violation'] # filter `Drug Violation`
df_drug.head()
```

Out[58]:

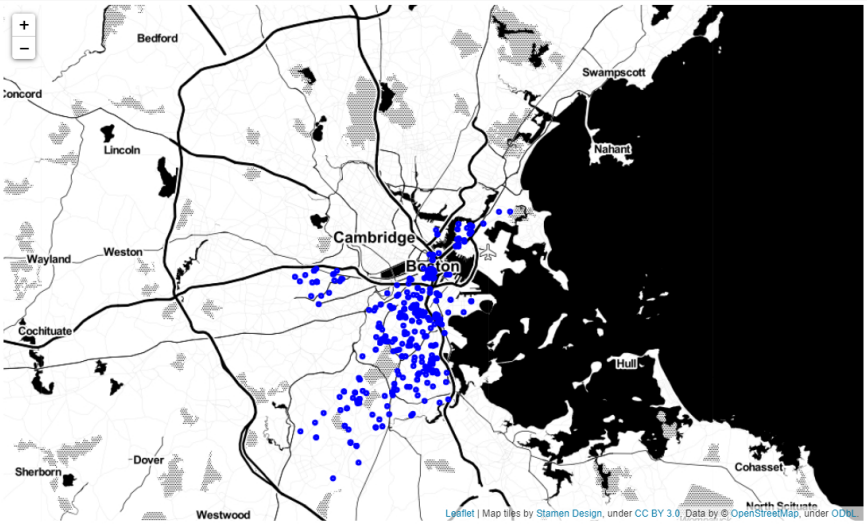
	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	OCCURRED_ON_DATE	YEAR	MC
39	I182070889	1843	Drug Violation	DRUGS - POSS CLASS B - INTENT TO MFR DIST DISP	NaN		2018-09-03 18:05:00	2018	
41	I182070889	1841	Drug Violation	DRUGS - POSS CLASS A - INTENT TO MFR DIST DISP	NaN		2018-09-03 18:05:00	2018	
76	I182070849	1849	Drug Violation	DRUGS - POSS CLASS B - COCAINE, ETC.	C6	177	2018-09-03 11:30:00	2018	
96	I182070829	1810	Drug Violation	DRUGS - SALE / MANUFACTURING	B2	325	2018-09-03 13:45:00	2018	
97	I182070829	1842	Drug Violation	DRUGS - POSS CLASS A - HEROIN, ETC.	B2	325	2018-09-03 13:45:00	2018	

Map `Drug Violation`

I've used only 500 records for plot.

In [59]:
```
m = folium.Map(location=[BOSTON_LAT, BOSTON_LONG], zoom_start=11, tiles='Stamen Toner',)

# We should to use apply method instead of manual loop, becuase apply method is faster than manual loop.
def toCircle(x):
    return folium.Circle(
        radius=100,
        location=[x['Lat'], x['Long']],
        popup=x['OFFENSE_DESCRIPTION'],
        color=UCR_colors_dict[x['UCR_PART']]
    ).add_to(m)

# sample only 500 records
_ = df_drug.dropna().head(500).apply(toCircle, axis=1)
m
```

Out[59]:



After I drill-down to `Drug Violation`

I'm still struck on it, but i can see some position have many data that overlap

And i decided to group data with **DBSCAN** for more insight.

Modeling

Clustering with DBSCAN

Why i use **DBSCAN** instead of **K-MEAN**, becuase i dont know number of cluster and i want to group cluster with density based.

In [60]:
```
# import
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import MinMaxScaler
```

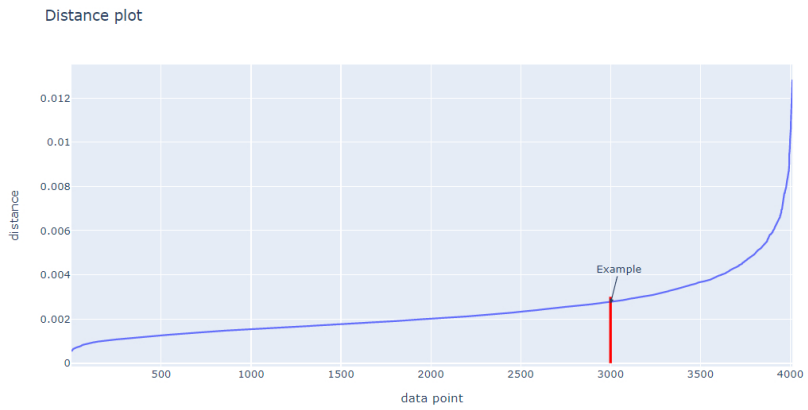
In [61]:
```
# get lat, long of drug
drug_locations = df_drug[['Lat', 'Long']].dropna().drop_duplicates().reset_index()[['Lat', 'Long']]
drug_locations.head()
```

Out[61]:

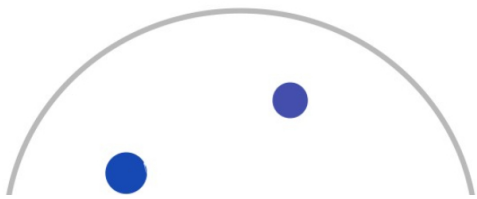
	Lat	Long
0	42.306769	-71.084319
1	42.331521	-71.070853
2	42.315809	-71.076187
3	42.312245	-71.073179
4	42.355214	-71.062064

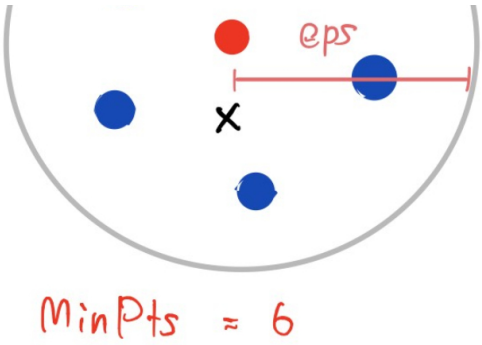
helper plot for tune hyperparameter for DBSCAN

In [62]:
```
size = 10 # closest top N
m = NearestNeighbors(n_neighbors=size).fit(drug_locations)
distances, _ = m.kneighbors(drug_locations)
distances = sorted(distances[:, size-1])

# plot
data = [
    go.Scatter(
        x=list(range(1,len(drug_locations))),
        y=distances[:-1],
    )
]

layout = go.Layout(
    title='Distance plot',
    yaxis=dict(
        title='distance'
    ),
    xaxis=dict(
        title='data point'
    ),
    shapes=[
        # Line Vertical
        go.layout.Shape(
            type="line",
            x0=3000,
            y0=0,
            x1=3000,
            y1=0.003,
            line=dict(
                color="Red",
                width=3
            )
        ),
    ],
    annotations=[
        go.layout.Annotation(
            x=3000,
            y=0.0027,
            text="Example",
            showarrow=True,
            arrowhead=2,
            ax=10,
            ay=-40
        ),
    ]
)

fig = go.Figure(data=data, layout=layout)

iplot(fig)
```



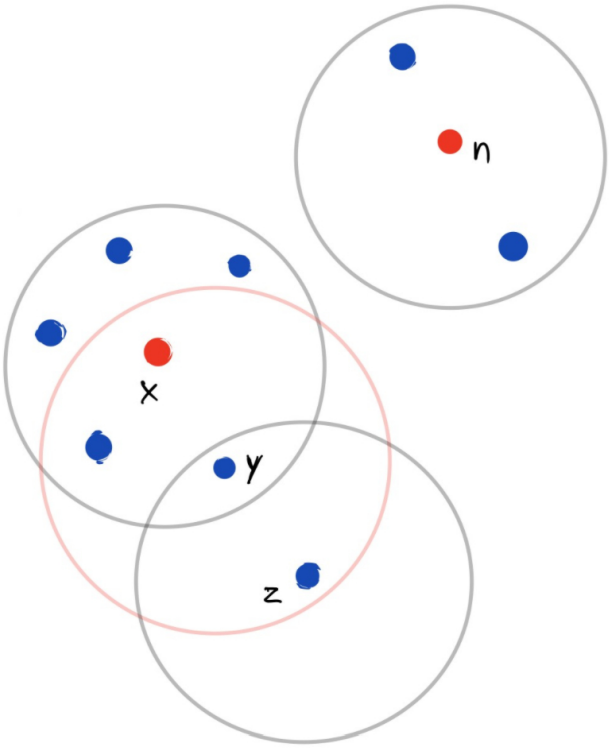
vertical line (RED) is the example of distance if you see y value is 0.0027 in x = 3000, It mean we have 3000 points that have maximum distance between closest top N (`size`) is 0.0027





`eps` is a number describe range to search neighbor in group.

If we select `eps` that have number of point so many, It'll have a few number of group.



And `min_samples`, group must have at least `min_samples` points.

References

In [63]:
```
# define DBSCAN and predict cluster
m = DBSCAN(eps=0.002, min_samples=15) # we select distance = 0.002
pred = m.fit_predict(drug_locations)
```

I've tuned hyperparameter `eps=0.002` and `min_samples=15`, it's mean for each group must have data at least 15 and not far more than 0.002 from each other.

In [64]:
```
# concat cluster to data
cluster_drug = pd.concat(
    [drug_locations, pd.Series(pred)],
    axis=1
)

# re-columns
cluster_drug.columns = list(cluster_drug.columns)[:-1] + ['cluster']

# filter number of cluster for plot map
cluster_drug_filter = cluster_drug[(cluster_drug['cluster'] < len(COLORS)) & (cluster_drug['cluster'] >= 0)]

cluster_drug_filter.head()
```

Out[64]:

	Lat	Long	cluster
0	42.306769	-71.084319	10
1	42.331521	-71.070853	0
2	42.315809	-71.076187	8
3	42.312245	-71.073179	8
4	42.355214	-71.062064	1

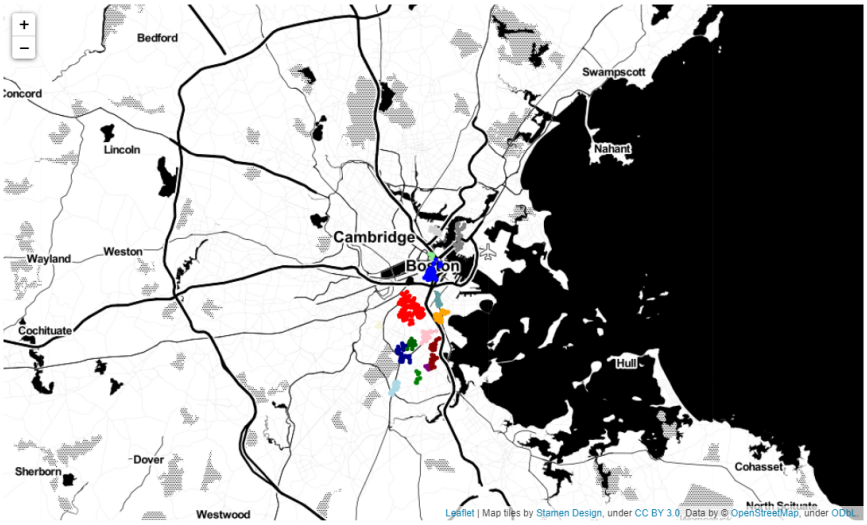
Map after cluster

I've used only 500 records for plot.

In [65]:
```
m = folium.Map(location=[BOSTON_LAT, BOSTON_LONG], zoom_start=11, tiles='Stamen Toner',)

# We should to use apply method instead of manual loop, becuase apply method is faster than manual loop.
```

```
    return folium.Circle(
        radius=50,
        location=[x['Lat'], x['Long']],
        popup=f"{x['cluster']}<br>{x['Lat']}, {x['Long']}",
        color=COLORS[int(x['cluster'])]
    ).add_to(m)

_ = cluster_drug_filter.head(500).apply(toCircle, axis=1)

m
```

Out[65]:



It's looks pretty good!

Each color indicate each cluster.

Feature engineering

Haversine formula:

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

where φ *is latitude,* λ *is longitude,* R *is earth's radius (mean radius = 6,371km); note that angles need to be in radians to pass to trig functions!*

In equation from above, I've used it for calculate diameter in each group for plot group.

In [66]:
```
def getDistance(lat1, long1, lat2, long2):
    '''
    Get the distance between 2 point in lat, long coordinates.
    '''

    lat1   = math.radians(lat1)
    lat2   = math.radians(lat2)

    d_lat  = math.radians(lat1 - lat2)
    d_long = math.radians(long1 - long2)

    a = math.sin(d_lat/2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(d_long/2)**2

    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))

    d = 6371 * c

    return d * 1000 # kilometer to meter
```

In [67]:
```
def info_group(x):
    '''
    Describe how to aggregate.
    '''

    d = {
        'center_lat': np.mean(x['Lat']),
        'center_long': np.mean(x['Long']),
        'diameter': getDistance(min(x['Lat']), min(x['Long']), max(x['Lat']), max(x['Long'])),
        'size': len(x['Lat'])
    }
    return pd.Series(d)

cluster_drug_group = cluster_drug.groupby('cluster').apply(info_group).reset_index()
cluster_drug_group = cluster_drug_group.iloc[1:]
cluster_drug_group.head()
```

Out[67]:

	cluster	center_lat	center_long	diameter	size
1	0	42.333230	-71.077943	1692.638752	341.0
2	1	42.353852	-71.060581	1222.597597	230.0
3	2	42.374879	-71.038854	727.465552	126.0
4	3	42.307528	-71.058669	789.051287	113.0
5	4	42.315502	-71.097936	1040.792523	98.0

Simplify map

I try to put school position as **RED** and drug area as **BLUE** and radius size is a number of drug crime around area that calculated with equation from above.

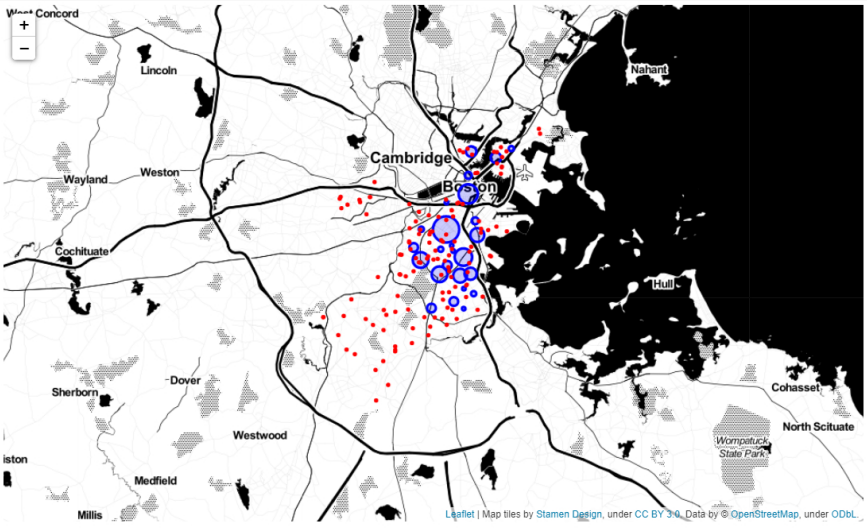
In [71]:
```
df_school["X"] = df_school["ï»¿X"]
```

In [72]:
```
def addSchool(m):

    def toCircle(x):
        return folium.Circle(
            radius=50,
            location=[x['Y'], x['X']],
            popup=x['SCH_NAME'],
            color="red"
        ).add_to(m)
```

```
    _ = df_school.dropna().apply(toCircle, axis=1)

    return m
```

In [73]:
```
m = folium.Map(location=[BOSTON_LAT, BOSTON_LONG], zoom_start=11, tiles='Stamen Toner',)

# add drug area
def toCircle(x):
    return folium.Circle(
        radius=x['diameter']/2,
        location=[x['center_lat'], x['center_long']],
        popup=f"size:{x['size']}<br> diameter:{x['diameter']}",
        color="blue",
        fill=True
    ).add_to(m)

_ = cluster_drug_group.apply(toCircle, axis=1)

# add school
m = addSchool(m)

m
```

Out[73]:



Red Circle: is school.

Blue Circle: is drug area.

It's look very nice.

After that, I created helper function to cluster and plot.

In [74]:
```
from utils.helper import getCluster, getLocations, plotDistance, plotCluster, plotMap
```

Let's see `Simple Assault`

In [75]:
```
df_sa = df_crime[df_crime['OFFENSE_CODE_GROUP'] == 'Simple Assault'] # filter
df_sa.head()
```

Out[75]:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION	DISTRICT	REPORTING_AREA	OCCURRED_ON_DATE	YEAR	MC
27	I182070904	802	Simple Assault	ASSAULT SIMPLE - BATTERY	C11	242	2018-09-03 18:34:00	2018	
32	I182070898	802	Simple Assault	ASSAULT SIMPLE - BATTERY	C11	351	2018-09-03 19:11:00	2018	
52	I182070875	802	Simple Assault	ASSAULT SIMPLE - BATTERY	A1	116	2018-09-03 16:55:00	2018	
72	I182070854	802	Simple Assault	ASSAULT SIMPLE - BATTERY	C11	257	2018-09-03 15:27:00	2018	
73	I182070852	802	Simple Assault	ASSAULT SIMPLE - BATTERY	E5	706	2018-09-02 12:30:00	2018	

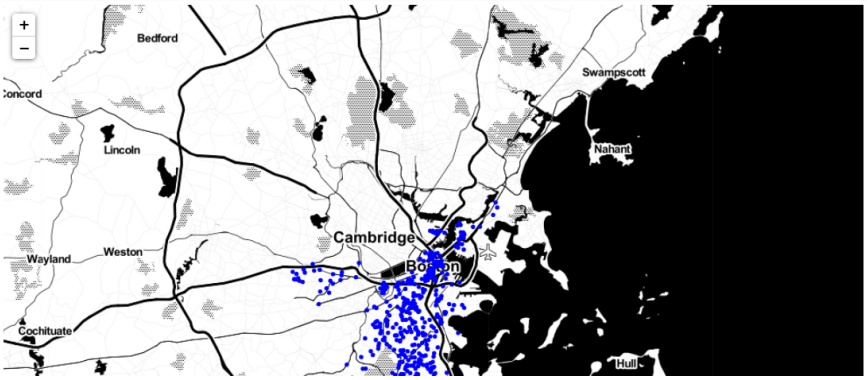
In [76]:
```
# get location
sa_locations = getLocations(df_sa)
sa_locations.head()
```

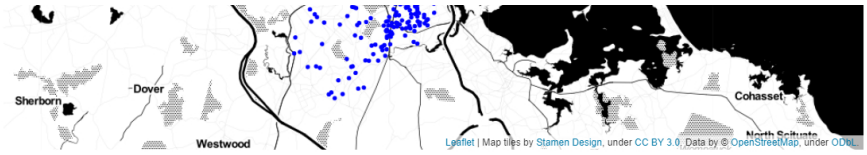
Out[76]:

	Lat	Long
0	42.317319	-71.061509
1	42.299284	-71.059172
2	42.351084	-71.059395
3	42.315267	-71.063069
4	42.271695	-71.146508

In [77]:
```
plotMap(sa_locations) # sampling only 500 records
```

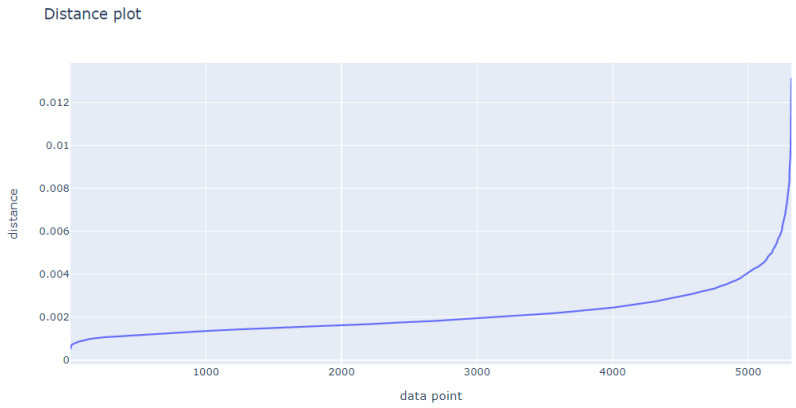
Out[77]:





`Simple assault` map

In [78]:
```
# plot distance for choose eps in DBSCAN
plotDistance(sa_locations)
```



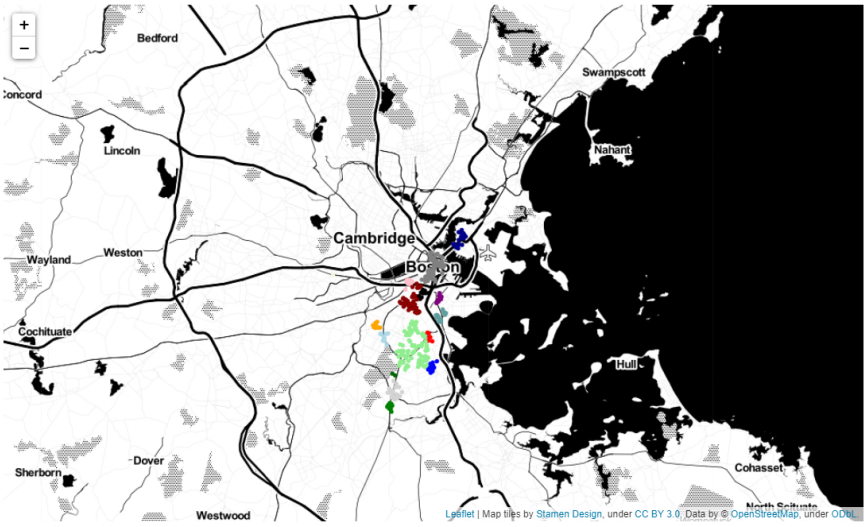
In [79]:
```
cluster_sa = getCluster(sa_locations, eps=0.0016, min_samples=10)
cluster_sa.head()
```

Out[79]:

	Lat	Long	cluster
0	42.317319	-71.061509	0
1	42.299284	-71.059172	1
2	42.351084	-71.059395	2
3	42.315267	-71.063069	0
4	42.339542	-71.069409	17

In [80]:
```
plotCluster(cluster_sa.head(500))
```

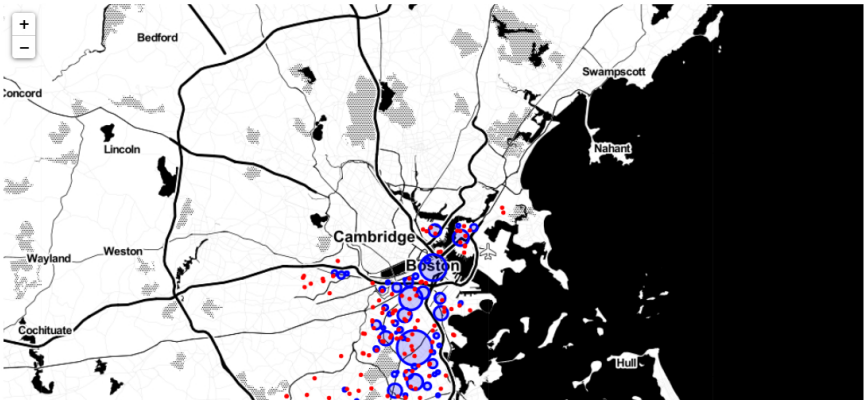
Out[80]:

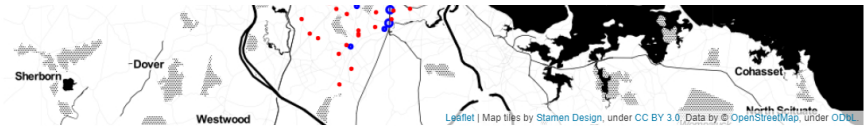


plot cluster

In [81]:
```
m = plotCluster(cluster_sa, density=True)

m = addSchool(m)

m
```

Out[81]:





If you see `Drug Violation` or `Simple Assault` , it's tell you which area has high risk.

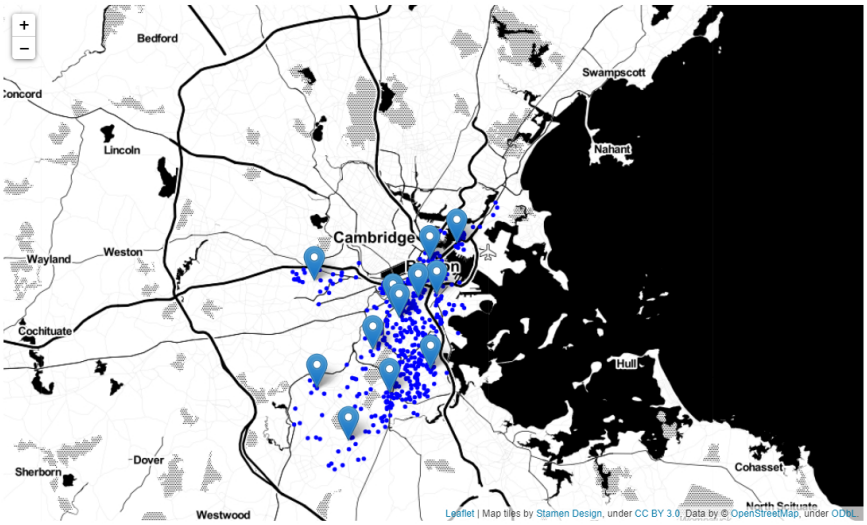
And I have a question that crime close to police station?

Police stations

In [82]:
```
# reference https://bpdnews.com/districts

police_stations = [
    {
        "station name": "Boston Police Headquarters",
        "lat": 42.3338046,
        "long": -71.0914456
    },
    {
        "station name": "District A-1 & A-15 (Downtown & Charlestown)",
        "lat": 42.3618223,
        "long": -71.0624814
    },
    {
        "station name": "District A-7 (East Boston)",
        "lat": 42.3711605,
        "long": -71.0408005
    },
    {
        "station name": "District B-2 (Roxbury)",
        "lat": 42.3280936,
        "long": -71.0862497
    },
    {
        "station name": "District B-3 (Mattapan)",
        "lat": 42.2846692,
        "long": -71.0938884
    },
    {
        "station name": "District C-6 (South Boston)",
        "lat": 42.3411763,
        "long": -71.057119
    },
    {
        "station name": "District C-11 (Dorchester)",
        "lat": 42.2980615,
        "long": -71.0612975
    },
    {
        "station name": "District D-4 (South End)",
        "lat": 42.3396293,
        "long": -71.0713493
    },
    {
        "station name": "District D-14 (Brighton)",
        "lat": 42.3493797,
        "long": -71.1527496
    },
    {
        "station name": "District E-5 (West Roxbury)",
        "lat": 42.2867867,
        "long": -71.1506609
    },
    {
        "station name": "District E-13 (Jamaica Plain)",
        "lat": 42.3095061,
        "long": -71.1067226
    },
    {
        "station name": "District E-18 (Hyde Park)",
        "lat": 42.2564568,
        "long": -71.1264767
    }
]
```

In [83]:
```
def addPolice(m):

    for obj in police_stations:
        _ = folium.Marker(
            location=[obj['lat'], obj['long']],
            popup=obj['station name']
        ).add_to(m)

    return m
```

In [84]:
```
m = plotMap(sa_locations)
m = addPolice(m)
m
```

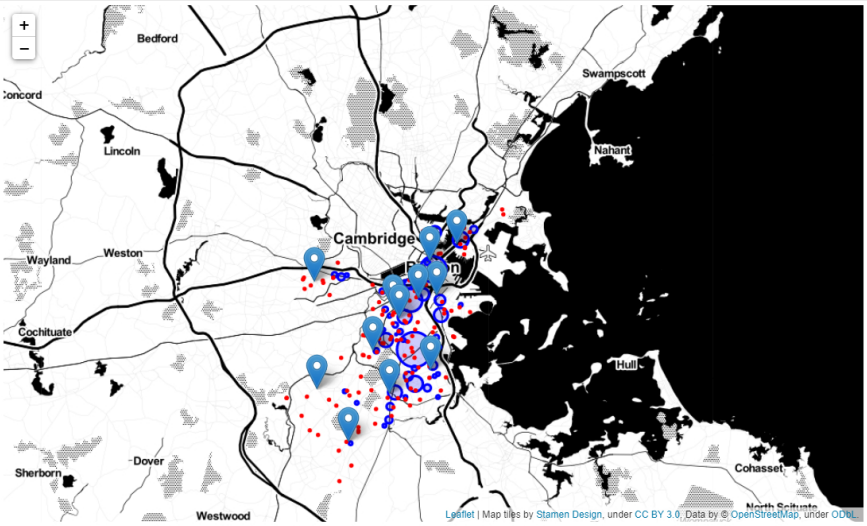
Out[84]:


MarkerPin is police station.

Blue Circle is crime.

In [85]:
```
m = plotCluster(cluster_sa, density=True)

# add school
m = addSchool(m)
# add police
m = addPolice(m)

m
```

Out[85]:



Red Circle is a school.

MarkerPin is a police station.

Blue Circle is a area that we have interesting.

Results

In my result, Police can use it to track and estimate risk in each area.

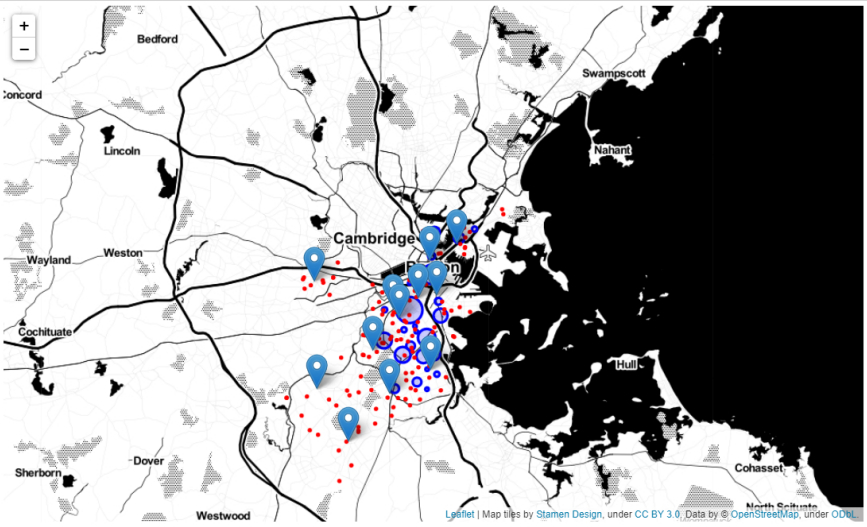
Drug Violation

We can see most drug violation area don't have police station in area, It's make sense becuase drug seller don't want police intervene them.

And we can see which school have high risk (possibilities of student addicted the drug), Police can planning or control to decrease the risk.

In [86]:
```
m = plotCluster(cluster_drug, density=True)

# add school
m = addSchool(m)
# add police
m = addPolice(m)

m
```

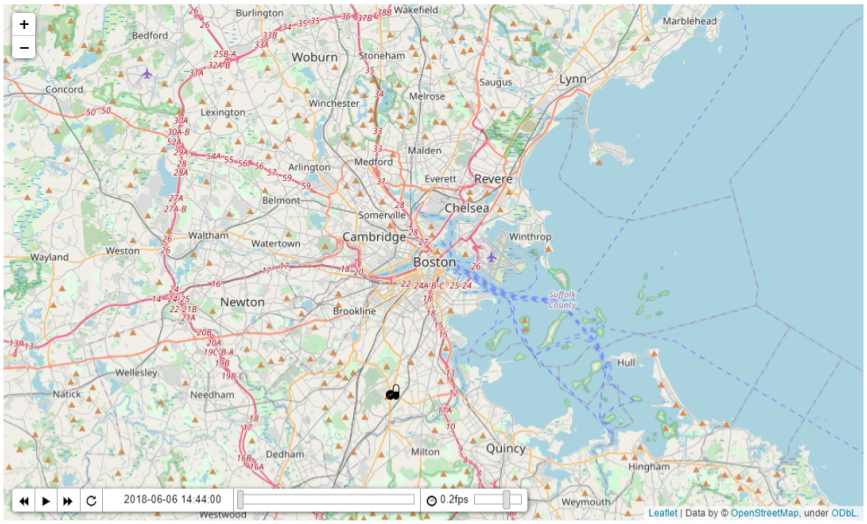
Out[86]:



In [87]:
```
features_col = ['OCCURRED_ON_DATE', 'Lat', 'Long', 'OFFENSE_CODE_GROUP'] # features columns
points = []

def getData(x):
    '''
    Get points for plot interactive map.
    '''
    points.append({
        "time": x['OCCURRED_ON_DATE'],
        "coordinates": [x['Long'], x['Lat']],
        "popup": x['OFFENSE_CODE_GROUP']
    })


_ = df_drug[features_col].head(1000).dropna().apply(getData, axis=1) # I've use 1000 samples

# features for plot
features = [
    {
        'type': 'Feature',
        'geometry': {
            'type': 'Point',
            'coordinates': point['coordinates'],
        },
        'properties': {
            'time': point['time'],
```

```
            'popup': point['popup'],
            'id': 'house',
            'icon': 'marker',
            'iconstyle': {
                'iconUrl': 'https://static.thenounproject.com/png/1220170-200.png',
                'iconSize': [20, 20]
            }
        }
    } for point in points
]

# set position of camera map
m = folium.Map(
    location=[BOSTON_LAT, BOSTON_LONG],
    zoom_start=11,
)

# put points to map
_ = plugins.TimestampedGeoJson(
    {
        'type': 'FeatureCollection',
        'features': features
    },
    period='P1D',
    add_last_point=True,
    auto_play=False,
    loop=False,
    max_speed=0.25,
    loop_button=True,
    date_options='YYYY-MM-DD HH:mm:ss',
    time_slider_drag_update=True,
    duration='P1D'
).add_to(m)

m
```

Out[87]:



From before map plot it's describe history data, area from history doesn't implies it'll be happend in present.

In this map plot, you can focus in each time precisely.

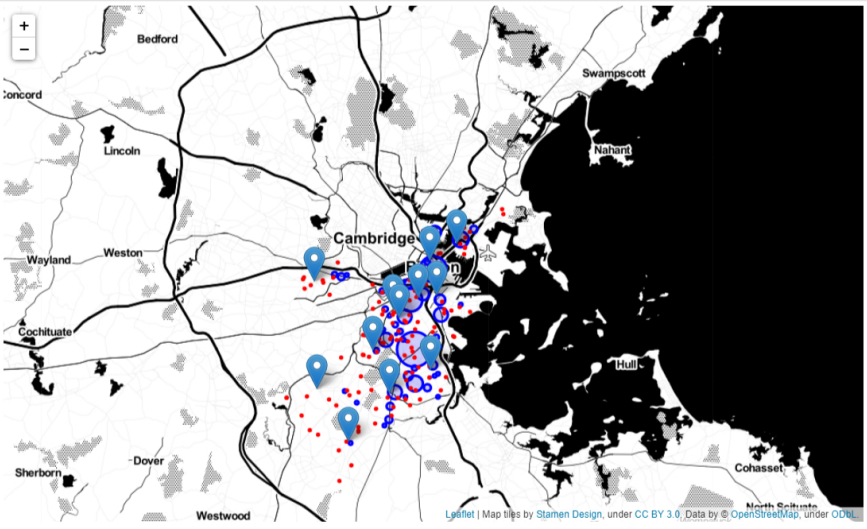
Simple Assault

We can see which area have high risk from assault, Police can see which area need attention.

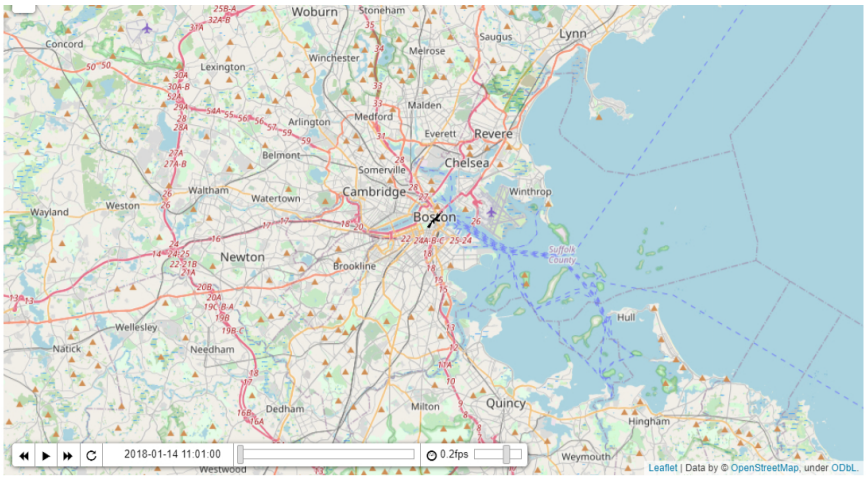
And police will know from some area which police stations is closest.

In [88]:
```
m = plotCluster(cluster_sa, density=True)

# add school
m = addSchool(m)
# add police
m = addPolice(m)

m
```

Out[88]:



In [89]:
```
features_col = ['OCCURRED_ON_DATE', 'Lat', 'Long', 'OFFENSE_CODE_GROUP'] # features columns
points = []

def getData(x):
    '''
    Get points for plot interactive map.
    '''
```



From before map plot it's describe history data, area from history doesn't implies it'll be happend in present.

In this map plot, you can focus in each time precisely.

In [ ]: