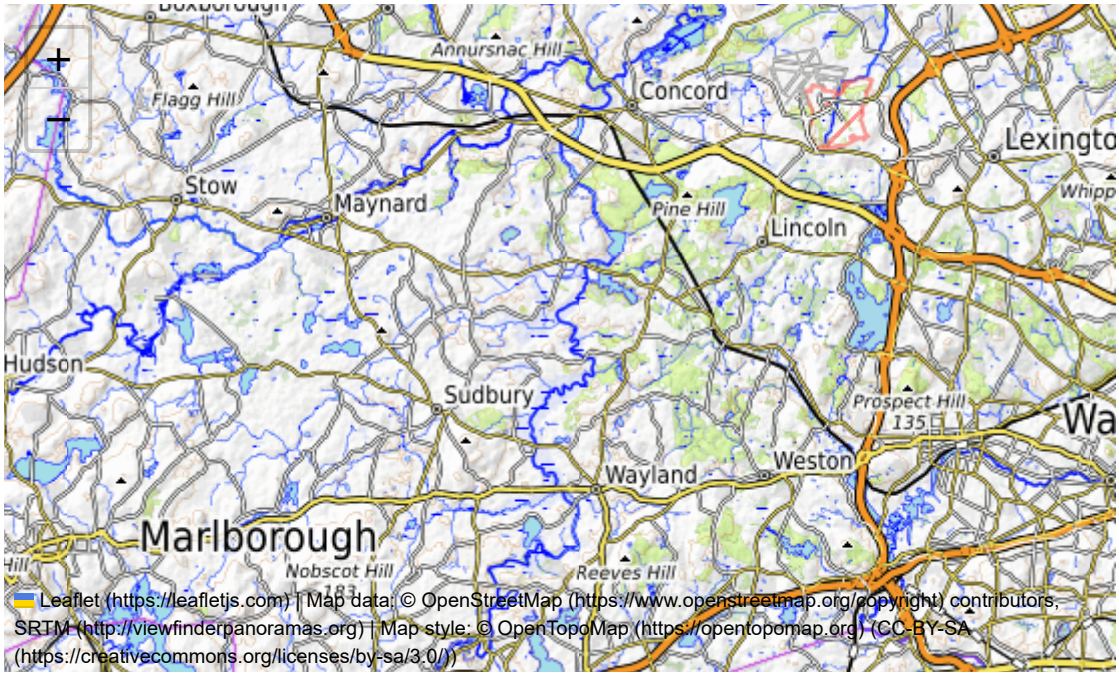
Boston Crime Zone Heatmap made with Python and Folium

In [ ]:
```
# Import the Folium library, which is used for creating interactive maps in Pyth
import folium

# Define the latitude and longitude coordinates for the city of Boston.
boston_lat_lon = [42.302, -71.1500]

# Create a Folium map object:
# - Set the initial location to the coordinates of Boston.
# - Set the initial zoom level to 11.
# - Specify the map tiles to be used from the OpenTopoMap server.
# - Provide attribution for the map data and style.
m = folium.Map(
    location=boston_lat_lon,
    zoom_start=11,
    tiles='https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png',
    attr='Map data: &copy; <a href="https://www.openstreetmap.org/copyright">Ope
)

# Display the created map.
m
```

Out[ ]:



Import data from file

In [ ]:
```
#dataset:    https://www.kaggle.com/datasets/AnalyzeBoston/crimes-in-boston
```

In [ ]:
```
import pandas as pd
# Specify the correct encoding (ISO-8859-1) when reading the CSV file.
# to avoid the following error:
# UnicodeDecodeError: 'utf-8' codec can't decode byte 0xa0 in position 8190: inv

df = pd.read_csv("crime.csv", encoding="ISO-8859-1")
```

Access dataframe values

In [ ]:
```
df.head()
```

Out[ ]:

	INCIDENT_NUMBER	OFFENSE_CODE	OFFENSE_CODE_GROUP	OFFENSE_DESCRIPTION
0	I182070945	619	Larceny	LARCENY ALL OTHERS
1	I182070943	1402	Vandalism	VANDALISM
2	I182070941	3410	Towed	TOWED MOTOR VEHICLE
3	I182070940	3114	Investigate Property	INVESTIGATE PROPERTY
4	I182070938	3114	Investigate Property	INVESTIGATE PROPERTY

In [ ]:
```
# Get the dimensions (number of rows and columns) of the DataFrame.
# num_rows, num_columns = df.shape
df.shape
```

Out[ ]:
```
(319073, 17)
```

In [ ]:
```
# Display a concise summary of the DataFrame's structure.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 319073 entries, 0 to 319072
Data columns (total 17 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   INCIDENT_NUMBER      319073 non-null  object 
 1   OFFENSE_CODE         319073 non-null  int64  
 2   OFFENSE_CODE_GROUP   319073 non-null  object 
 3   OFFENSE_DESCRIPTION  319073 non-null  object 
 4   DISTRICT             317308 non-null  object 
 5   REPORTING_AREA       319073 non-null  object 
 6   SHOOTING             1019 non-null    object 
 7   OCCURRED_ON_DATE     319073 non-null  object 
 8   YEAR                 319073 non-null  int64  
 9   MONTH                319073 non-null  int64  
 10  DAY_OF_WEEK          319073 non-null  object 
 11  HOUR                 319073 non-null  int64  
 12  UCR_PART             318983 non-null  object 
 13  STREET               308202 non-null  object 
 14  Lat                  299074 non-null  float64
 15  Long                 299074 non-null  float64
 16  Location             319073 non-null  object 
dtypes: float64(2), int64(4), object(11)
memory usage: 41.4+ MB
```

In [ ]:
```
# Generate the summary of descriptive statistics for numeric columns in the Data
df.describe()
```

Out[ ]:

	OFFENSE_CODE	YEAR	MONTH	HOUR	Lat	29
count	319073.000000	319073.000000	319073.000000	319073.000000	299074.000000	
mean	2317.546956	2016.560586	6.609719	13.118205	42.214381	
std	1185.285543	0.996344	3.273691	6.294205	2.159766	
min	111.000000	2015.000000	1.000000	0.000000	-1.000000	
25%	1001.000000	2016.000000	4.000000	9.000000	42.297442	
50%	2907.000000	2017.000000	7.000000	14.000000	42.325538	
75%	3201.000000	2017.000000	9.000000	18.000000	42.348624	
max	3831.000000	2018.000000	12.000000	23.000000	42.395042	

Rename columns.

In [ ]:

```
# Define a dictionary 'columns' to map the original column names to the desired
columns = {
    'OCCURRED_ON_DATE': 'date',            # Rename 'OCCURRED_ON_DATE' to 'date'
    'OFFENSE_CODE_GROUP': 'offense',       # Rename 'OFFENSE_CODE_GROUP' to 'off
    'SHOOTING': 'shooting',                # Rename 'SHOOTING' to 'shooting'
    'Lat': 'lat',                           # Rename 'Lat' to 'lat'
    'Long': 'lon',                          # Rename 'Long' to 'lon'
}

# Rename the columns in the DataFrame using the 'columns' dictionary.
df = df.rename(columns=columns)

# Select only the columns specified in the 'columns' dictionary.
# This line keeps only the columns with the new names specified in the dictionar
# It's effectively dropping any columns that are not in the 'columns' dictionary
df = df[list(columns.values())]

# Display the resulting DataFrame with the selected columns.
df
```

Out[ ]:

	date	offense	shooting	lat	lon
0	2018-09-02 13:00:00	Larceny	NaN	42.357791	-71.139371
1	2018-08-21 00:00:00	Vandalism	NaN	42.306821	-71.060300
2	2018-09-03 19:27:00	Towed	NaN	42.346589	-71.072429
3	2018-09-03 21:16:00	Investigate Property	NaN	42.334182	-71.078664
4	2018-09-03 21:05:00	Investigate Property	NaN	42.275365	-71.090361
...	...	...	...	...	...
319068	2016-06-05 17:25:00	Warrant Arrests	NaN	42.336951	-71.085748
319069	2015-07-09 13:38:00	Homicide	NaN	42.255926	-71.123172
319070	2015-07-09 13:38:00	Warrant Arrests	NaN	42.255926	-71.123172
319071	2016-05-31 19:35:00	Warrant Arrests	NaN	42.302333	-71.111565
319072	2015-06-22 00:12:00	Warrant Arrests	NaN	42.333839	-71.080290

319073 rows × 5 columns

Deal with data types

In [ ]:
```
# Check the data type of the first value in the 'date' column.
# This line checks the data type of the first element in the 'date' column of th
# The type() function is used to determine the Python data type of the value.
date_type = type(df.date[0])
date_type
```

Out[ ]:
```
str
```

In [ ]:
```
# Convert the 'date' column to datetime format.
# This line converts the 'date' column of the DataFrame to datetime format using
df.date = pd.to_datetime(df.date)

# Sort the DataFrame by the 'date' column in ascending order.
# This line sorts the DataFrame based on the 'date' column, arranging the rows i
df = df.sort_values(by='date')

# Print the first 10 values in the 'date' column after sorting.
# This line displays the 'date' values of the first 10 rows in the sorted DataFr
print(df.date[0:10])
```

```
129056   2015-06-15 00:00:00
314676   2015-06-15 00:00:00
310350   2015-06-15 00:00:00
253464   2015-06-15 00:00:00
8793     2015-06-15 00:00:00
318414   2015-06-15 00:00:00
317446   2015-06-15 00:00:00
303001   2015-06-15 00:00:00
317447   2015-06-15 00:00:00
318621   2015-06-15 00:01:00
Name: date, dtype: datetime64[ns]
```

Deal with null values.

In [ ]:
```
# Convert the 'shooting' column to a boolean value, True for "Y" and False for o
# This line creates a new boolean column 'shooting' in the DataFrame where True
# to rows with the original value "Y" in the 'shooting' column, and False for al
df.shooting = (df.shooting == "Y")

# Drop rows with any missing values (NaN) from the DataFrame.
# This line removes rows with missing values in any column from the DataFrame.
df = df.dropna()

# Display the first few rows of the cleaned DataFrame.
# This line shows the first few rows of the DataFrame after removing rows with m
# and converting the 'shooting' column to boolean format.
df.head()
```

Out[ ]:

	date	offense	shooting	lat	lon
129056	2015-06-15	Harassment	False	42.291093	-71.065945
314676	2015-06-15	Confidence Games	False	42.300217	-71.080979
310350	2015-06-15	Other	False	42.293606	-71.071887
253464	2015-06-15	Property Lost	False	42.283634	-71.082813
8793	2015-06-15	Property Lost	False	-1.000000	-1.000000

In [ ]:
```
df.shape
```

Out[ ]:
```
(299074, 5)
```

Group data by month

In [ ]:
```
import datetime
from dateutil.relativedelta import relativedelta

# Create an empty list to store DataFrames for each month.
months = []

# Define the starting date for the monthly time periods.
start = datetime.datetime(2015, 6, 1)

# Iterate over months from the start date until the maximum date in the 'date' c
while start < df.date.max():
    # Calculate the end date for the current month.
    end = start + relativedelta(months=+1)

    # Create a mask to select rows within the current month.
    mask = (start <= df.date) & (df.date < end)

    # Extract a subset of the DataFrame for the current month and select only th
    df_month = df[mask]
    df_month = df_month[['lat', 'lon']]

    # Append the DataFrame for the current month to the 'months' list.
    months.append(df_month)

    # Move to the next month.
    start = end
```

```
# Print the contents of the DataFrame for the FIRST month in the 'months' list.
print(months[0])
```

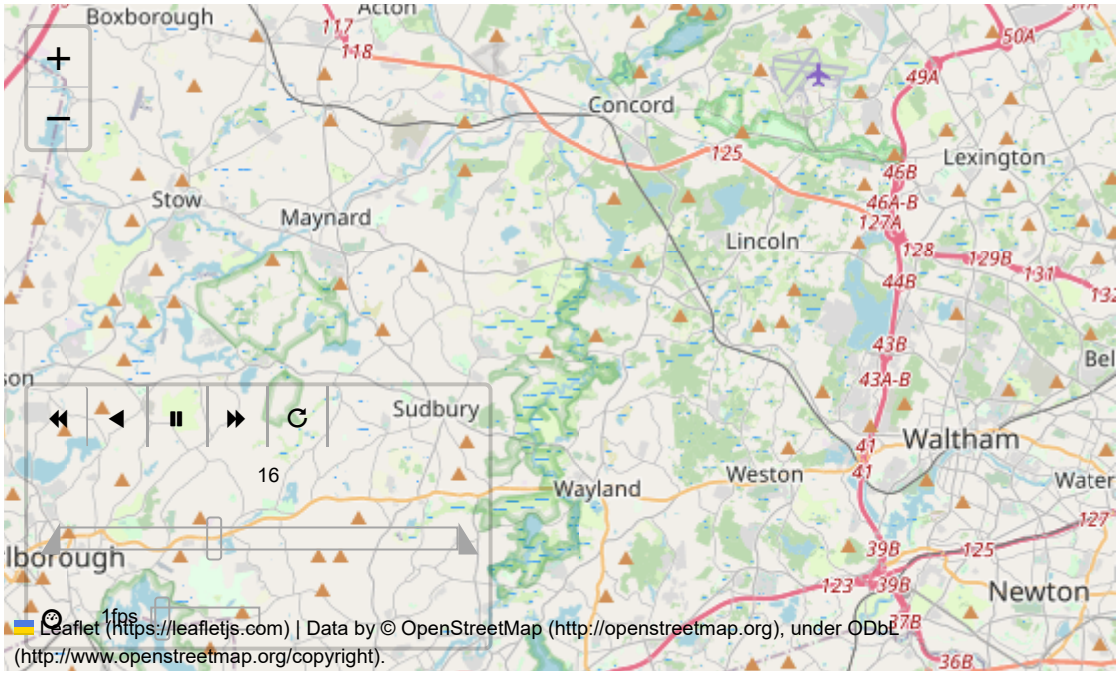
```
             lat        lon
129056  42.291093 -71.065945
314676  42.300217 -71.080979
310350  42.293606 -71.071887
253464  42.283634 -71.082813
8793    -1.000000  -1.000000
...           ...        ...
314737  42.380275 -71.060377
314736  42.380275 -71.060377
314735  42.380275 -71.060377
314708  42.288705 -71.078108
314724  42.280587 -71.074322

[4066 rows x 2 columns]
```

Create heatmap

In [ ]:

```
from folium.plugins import HeatMapWithTime

# Create a base Folium map centered around Boston with a specified zoom level.
m = folium.Map(boston_lat_lon, zoom_start=11)

# Initialize a HeatMapWithTime object.
# The HeatMapWithTime allows you to visualize temporal data as a heatmap that ch
# Here, we're passing a list of data points for each month to the HeatMapWithTim
# Each element in the list corresponds to a DataFrame for a specific month, and
# are obtained by converting the 'lat' and 'lon' columns of those DataFrames to
hm = HeatMapWithTime(
    data=[m.values.tolist() for m in months],
    radius=5,                # Set the radius of the heatmap points.
    max_opacity=0.5,         # Set the maximum opacity of the heatmap points.
    auto_play=True,         # Disable autoplay of the heatmap animation. -->"aut
)

# Add the HeatMapWithTime layer to the base map.
hm.add_to(m)

# Display the map with the HeatMapWithTime layer.
m
```

Out[ ]:



Spatially clustered data

In [ ]:

```
# Define a constant representing the grid resolution for clustering.
LAT_LON_GRID = 0.005

# Define a function to custom round values to a specified resolution.
# This function rounds the input value to the nearest multiple of the specified
def custom_round(val, resolution):
    return round(val / resolution) * resolution

# Define a function for clustering data based on latitude and longitude.
# This function takes a DataFrame 'df_interval' and clusters the data points by
# rounding the latitude and longitude to the specified grid resolution.
# It then aggregates the data by counting occurrences in each cluster and normal
# the cluster weights to a range between 0 and 1.
def cluster(df_interval):
    data = df_interval.copy()

    # Custom round the latitude and longitude values to the specified grid resol
    data = custom_round(data, LAT_LON_GRID)

    # Group the data by clustered latitude and longitude values and count occurr
    data = data.groupby(["lat", "lon"]).size().reset_index(name="weight")

    # Normalize the cluster weights by dividing each weight by the maximum weigh
    data.weight = data.weight / data.weight.max()

    return data

# Define the start date for the time interval.
start = datetime.datetime(2015, 6, 1)

# Calculate the end date for the current month.
end = start + relativedelta(months=+1)

# Create a mask to select rows within the current month.
mask = (start <= df.date) & (df.date < end)

# Extract a subset of the DataFrame for the current month and select only the 'l
```

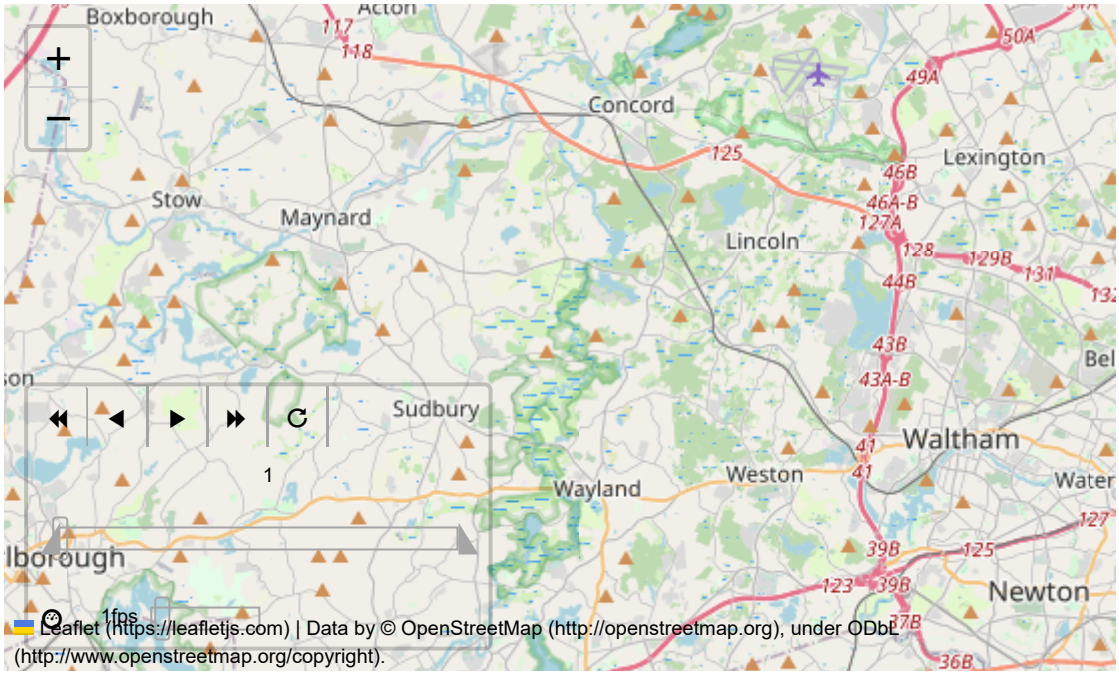
```
df_month = df[mask]
df_month = df_month[['lat', 'lon']]

# Call the 'cluster' function with the DataFrame for the current month.
# The result is a clustered representation of the data for the specified month.
print(cluster(df_month))
```

```
        lat     lon    weight
0    -1.000  -1.000  0.066667
1    42.235 -71.140  0.013333
2    42.235 -71.125  0.013333
3    42.240 -71.140  0.053333
4    42.240 -71.125  0.066667
..      ...     ...       ...
436  42.390 -71.015  0.013333
437  42.390 -71.010  0.093333
438  42.390 -71.005  0.080000
439  42.390 -71.000  0.040000
440  42.395 -71.010  0.080000

[441 rows x 3 columns]
```

Redraw heat map

In [ ]:
```
from folium.plugins import HeatMapWithTime

# Create a base Folium map centered around Boston with a specified zoom level.
m = folium.Map(boston_lat_lon, zoom_start=11)

# Initialize a HeatMapWithTime object.
# The HeatMapWithTime allows you to visualize temporal data as a heatmap that ch
# Here, we're creating a heatmap animation over time by providing clustered data
# The 'data' parameter is generated using a list comprehension that applies the
# to each month's DataFrame (stored in the 'months' list) and converts the resul
# tuples for each cluster (latitude, longitude, weight).
hm = HeatMapWithTime(
    data=[cluster(m).values.tolist() for m in months],
    radius=15,               # Set the radius of the heatmap points.
    max_opacity=0.5,         # Set the maximum opacity of the heatmap points.
    auto_play=False,         # Disable autoplay of the heatmap animation.
)

# Add the HeatMapWithTime layer to the base map.
hm.add_to(m)

# Display the map with the added HeatMapWithTime layer.
m
```

Out[ ]:



Many thanks to pbesser and Analyze Boston (data)