Introduction to Folium for interactive maps in python

Installing folium

In [1]:
```
pip install folium
```

```
Requirement already satisfied: folium in c:\users\vishal\anaconda3\lib\site-packages
(0.14.0)
Requirement already satisfied: branca>=0.6.0 in c:\users\vishal\anaconda3\lib\site-p
ackages (from folium) (0.6.0)
Requirement already satisfied: numpy in c:\users\vishal\anaconda3\lib\site-packages
(from folium) (1.21.5)
Requirement already satisfied: requests in c:\users\vishal\anaconda3\lib\site-packag
es (from folium) (2.28.1)
Requirement already satisfied: jinja2>=2.9 in c:\users\vishal\anaconda3\lib\site-pac
kages (from folium) (2.11.3)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\vishal\anaconda3\lib\sit
e-packages (from jinja2>=2.9->folium) (2.0.1)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\vishal\anaconda3
\lib\site-packages (from requests->folium) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\vishal\anaconda3\li
b\site-packages (from requests->folium) (1.26.11)
Requirement already satisfied: idna<4,>=2.5 in c:\users\vishal\anaconda3\lib\site-pa
ckages (from requests->folium) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\vishal\anaconda3\lib\s
ite-packages (from requests->folium) (2022.9.14)
Note: you may need to restart the kernel to use updated packages.
```

Initialize a folium map object with map center and zoom

In [2]:
```
import folium
mapObj = folium.Map(location=[23.437730075416685, 72.585859375],
zoom_start=10, tiles='OpenStreetMap')
mapObj
```

Out[2]:

Keep tiles=None to create a map without tiles The options for tile sources in folium can be found here

Layer controls button to show/hide layers

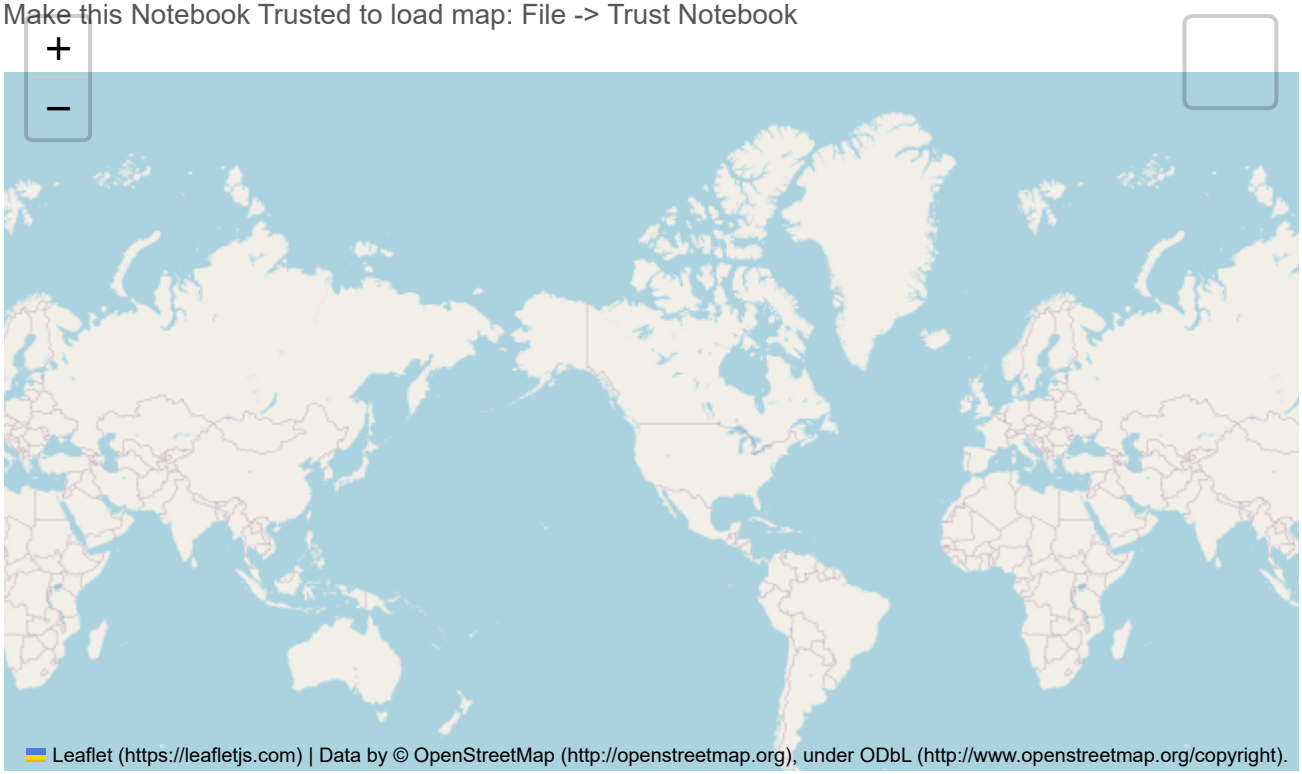
In [3]:
```
import folium
mapObj = folium.Map()

# add layers control over the map
folium.LayerControl().add_to(mapObj)
```

Out[3]:
```
<folium.map.LayerControl at 0x256d67407c0>
```

In [4]:
```
1 mapObj
```

Out[4]:

Make this Notebook Trusted to load map: File -> Trust Notebook



Create tile layers from built-in sources

In [5]:
```
1  import folium
2  mapObj = folium.Map()
3
4  # add new tile layers
5  folium.TileLayer('openstreetmap').add_to(mapObj)
6  folium.TileLayer('stamenterrain', attr="stamenterrain").add_to(mapObj)
7  folium.TileLayer('stamenwatercolor', attr="stamenwatercolor")
8                  .add_to(mapObj)
9
10 # add layers control over the map
11 folium.LayerControl().add_to(mapObj)
```

Out[5]:
```
<folium.map.LayerControl at 0x256d67432e0>
```

Map tileset to use. Can choose from this list of built-in tiles:

”OpenStreetMap”

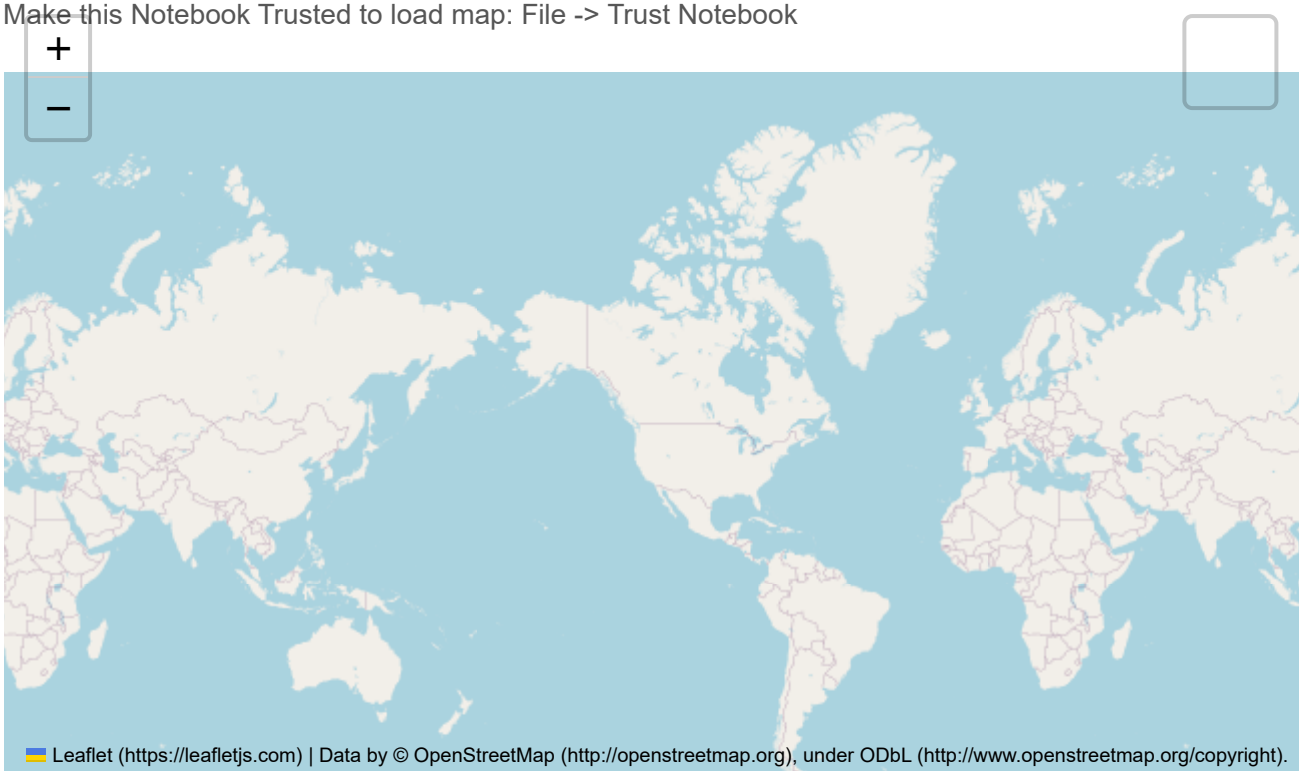
”Stamen Terrain”, “Stamen Toner”, “Stamen Watercolor”

”CartoDB positron”, “CartoDB dark_matter”

In [6]:
```
1 mapObj
```

Out[6]:

Make this Notebook Trusted to load map: File -> Trust Notebook



Create tile layers from other sources

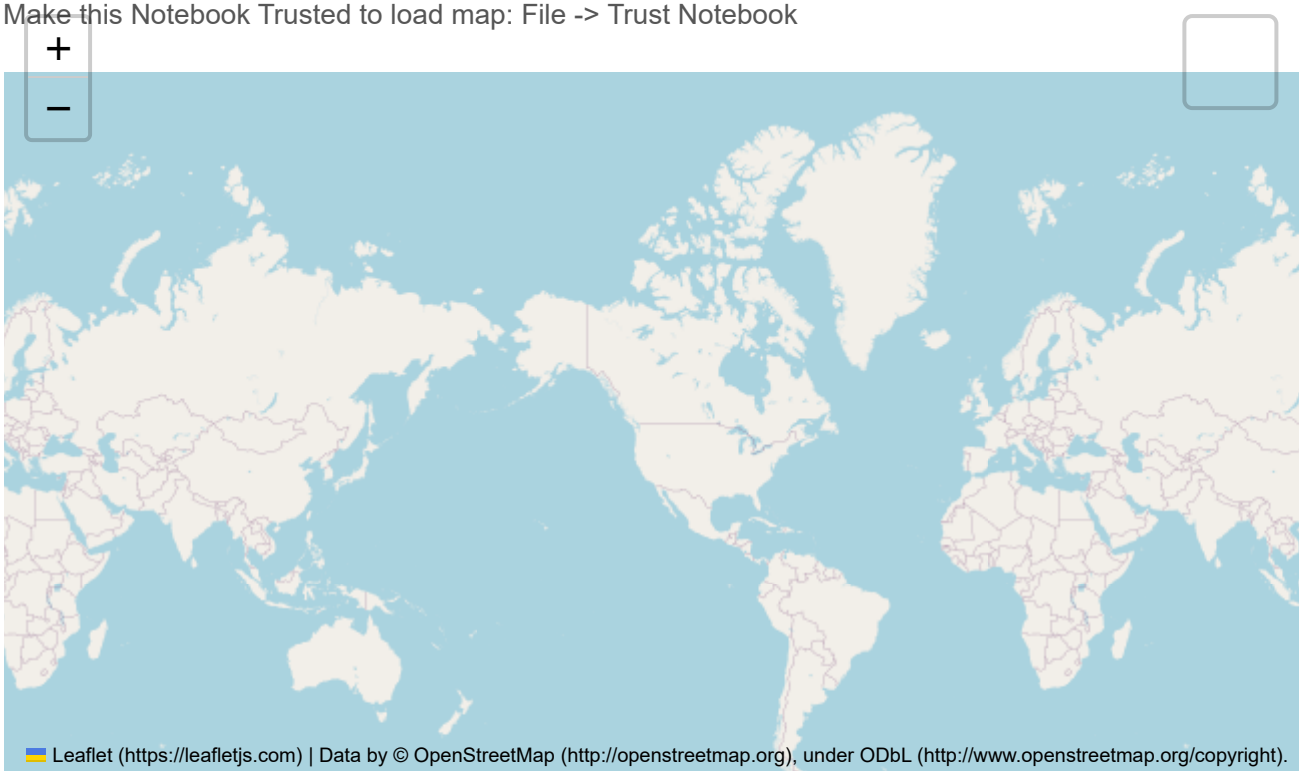
In [7]:
```
1  import folium
2  mapObj = folium.Map()
3
4  # add new tile layer from external source
5  folium.TileLayer('https://{s}.basemaps.cartocdn.com/dark_all/
6                   {z}/{x}/{y}{r}.png',
7                   name='CartoDB.DarkMatter', attr="CartoDB.DarkMatter")
8  .add_to(mapObj)
9
10 # add layers control over the map
11 folium.LayerControl().add_to(mapObj)
```

Out[7]:
```
<folium.map.LayerControl at 0x256d672d700>
```

In [8]:
```
1 mapObj
```

Out[8]:

Make this Notebook Trusted to load map: File -> Trust Notebook



Save a map object as html file

In [9]:
```
1 import folium
2 mapObj = folium.Map()
3
4 # save the map object as a html file
5 mapObj.save('output1.html')
```

In [10]:
```
1  import folium
2
3  # initialize a map with center and zoom
4  mapObj = folium.Map(location=[23.437730075416685, 72.585859375],
5                      zoom_start=4, tiles=None)
6
7  # add tile layers
8  folium.TileLayer('openstreetmap').add_to(mapObj)
9  folium.TileLayer('stamenterrain', attr="stamenterrain").add_to(mapObj)
10 folium.TileLayer('stamenwatercolor', attr="stamenwatercolor")
11                 .add_to(mapObj)
12 folium.TileLayer('https://{s}.basemaps.cartocdn.com/dark_all/
13                 {z}/{x}/{y}{r}.png'
14                 , name='CartoDB.DarkMatter',
15                 attr="CartoDB.DarkMatter").add_to(mapObj)
16
17 # add layers control over the map
18 folium.LayerControl().add_to(mapObj)
19
20 # save the map as html file
21 mapObj.save('folium_intro.html')
```

In [11]:
```
1 mapObj
```

Out[11]:

In [12]:
```
1  import folium
2
3  mapObj = folium.Map(location=[23.437730075416685, 72.585859375],
4                      zoom_start=5)
5
6  layer1 = folium.GeoJson(
7      data=(open("Indian_States.geojson", 'r').read()),
8      name="India")
9  layer1.add_to(mapObj)
10
11 mapObj.save('output.html')
```

In [13]:
```
1 mapObj
```

Out[13]:

Control border and fill style of GeoJSON objects

- use the style_function input of folium.GeoJSON function to control the styling of the paths.
- style_function should be a function that returns a dictionary with styling properties specified in the documentation here

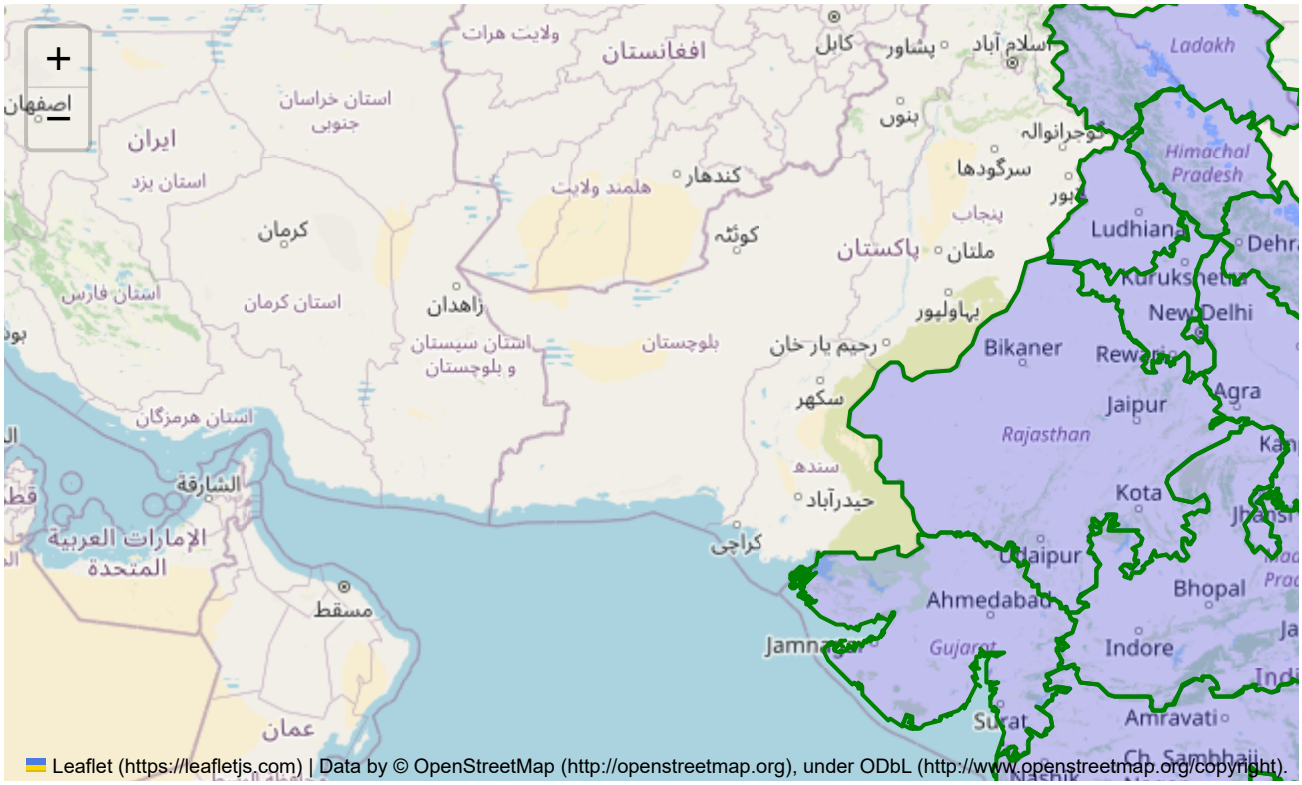
Some of the important styling properties are

- color - line stroke color
- weight - line stroke width in pixels
- opacity - line stroke opacity
- fillColor - fill Color
- fillOpacity - ranges between 0 to 1. 0 means transparent, 1 means opaque

In [14]:

```
import folium

mapObj = folium.Map(location=[23.437730075416685, 72.585859375],
                    zoom_start=5)

# style options - https://leafletjs.com/reference-1.7.1.html#path
bordersStyle = {
    'color': 'green',
    'weight': 2,
    'fillColor': 'blue',
    'fillOpacity': 0.2
}

folium.GeoJson(
    data=(open("Indian_States.geojson", 'r').read()),
    name="India",
    style_function=lambda x: bordersStyle).add_to(mapObj)

mapObj.save('output2.html')
mapObj
```

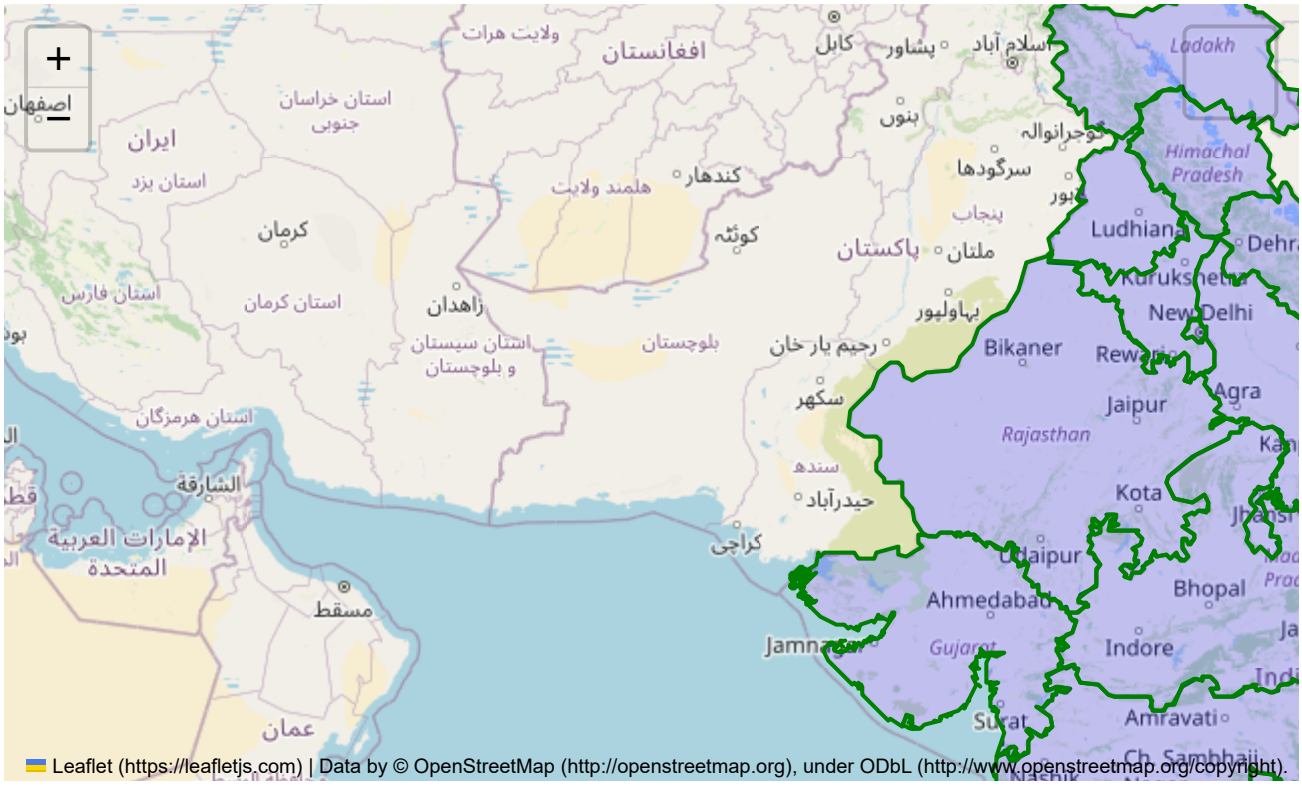
Out[14]:



In [15]:

```
import folium

# initialize a map with center and zoom
mapObj = folium.Map(location=[23.437730075416685, 72.585859375],
                    zoom_start=5)


# border styles dictionary
bordersStyle = {
    'color': 'green',
    'weight': 2,
    'fillColor': 'blue',
    'fillOpacity': 0.2
}

folium.GeoJson(
    data=(open("Indian_States.geojson", 'r').read()),
    name="India",
    style_function=lambda x: bordersStyle).add_to(mapObj)


# add layer control over the map
folium.LayerControl().add_to(mapObj)

# save the map as html file
mapObj.save('output3.html')
mapObj
```

Out[15]:



Draw a simple circle with location and radius

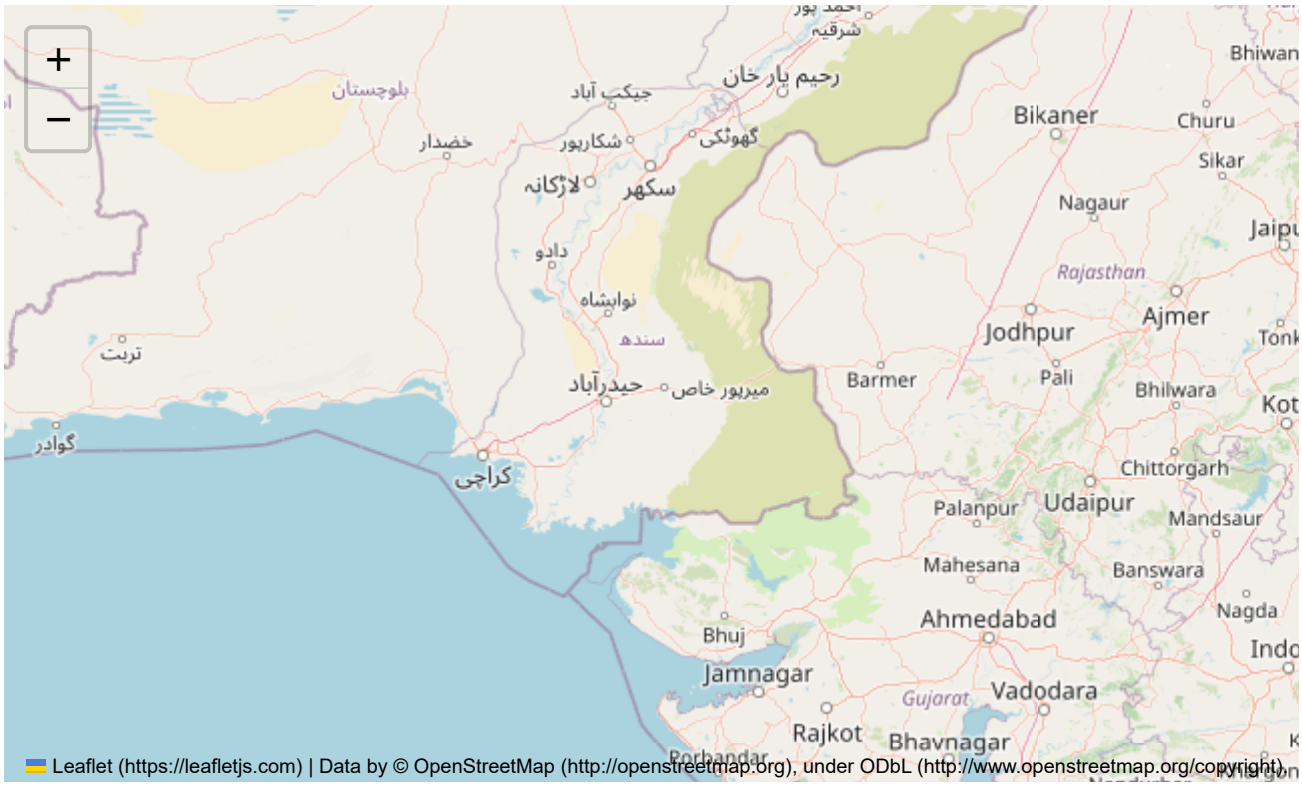
location - latitude, longitude location of the center of the circle

radius - radius of the circle in meters

In [16]:

```
import folium

mapObj = folium.Map(location=[23.437730075416685, 72.585859375],
                    zoom_start=6)

folium.Circle(location=[23.294059708387206, 78.26660156250001],
              radius=50000
              ).add_to(mapObj)

mapObj.save('output.html')
mapObj
```

Out[16]:



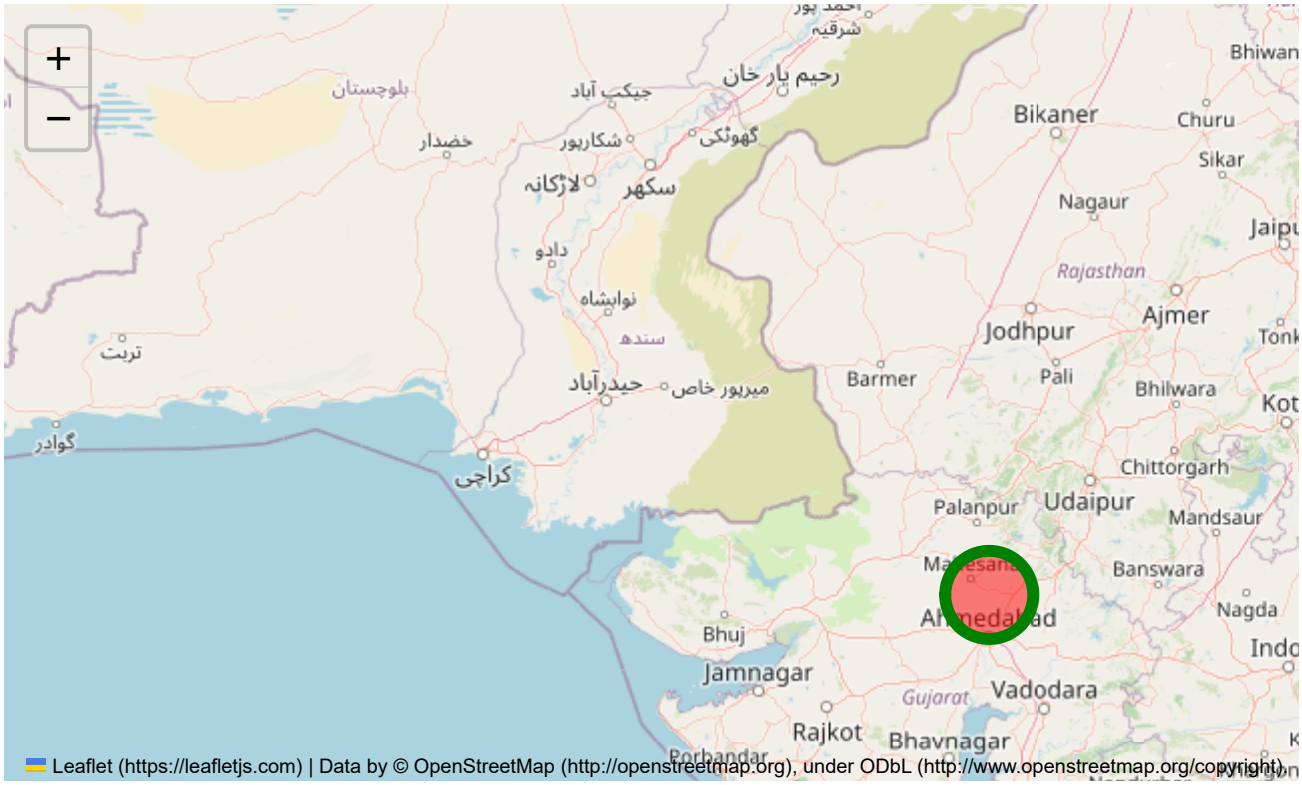
Some of the important styling options are

- stroke - set to True to enable line stroke, default is True
- weight - line stroke width in pixels, default is 5
- color - line stroke color
- opacity - line stroke opacity
- fill - set to True to enable filling with color, default is False
- fill_color - fill Color
- fill_opacity - ranges between 0 to 1. 0 means transparent, 1 means opaque

In [17]:

```
import folium

mapObj = folium.Map(location=[23.437730075416685, 72.585859375],
                    zoom_start=6)

folium.Circle(location=[23.437730075416685, 72.585859375],
              radius=50000,
              color='green',
              weight=6,
              fill_color='red',
              fill_opacity = 0.5
              ).add_to(mapObj)

mapObj.save('output.html')
mapObj
```

Out[17]:



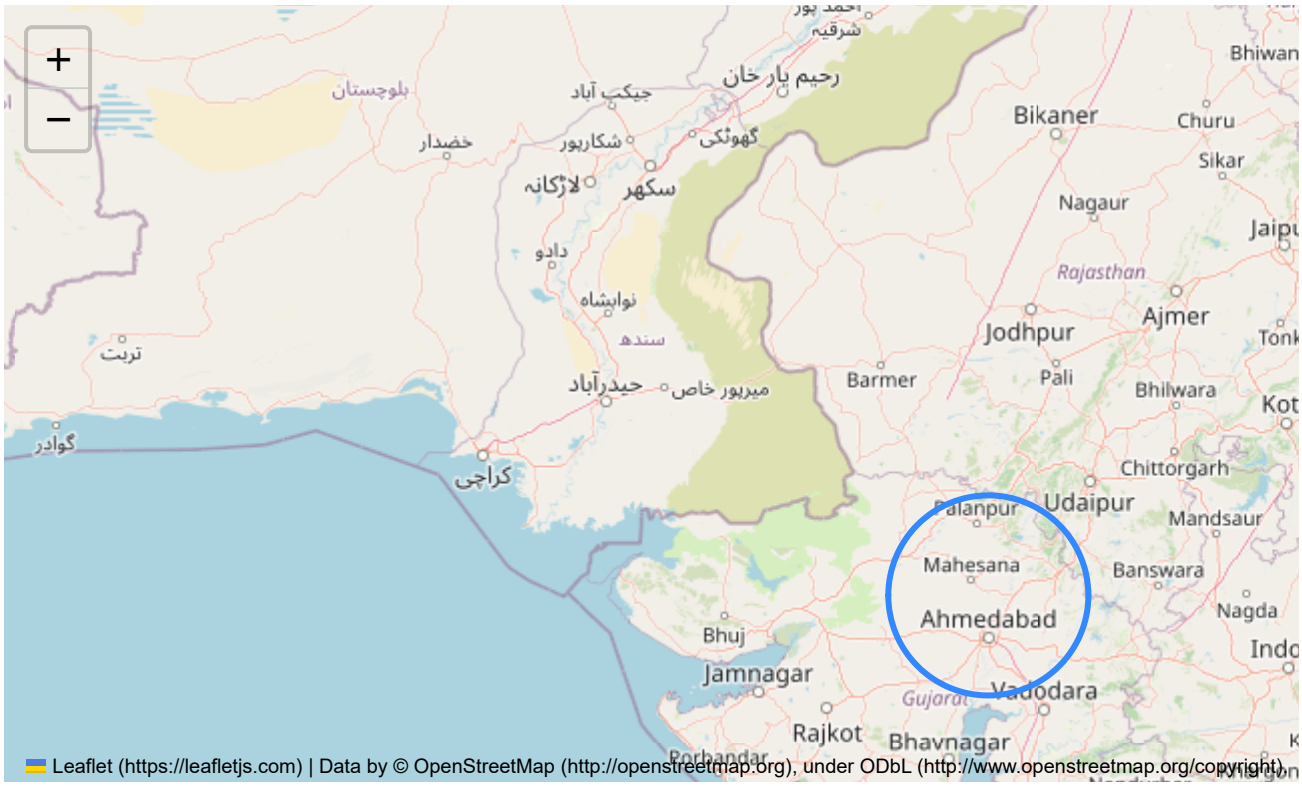
Difference between Circle and CircleMarker

- The radius in Circle is defined in meters, where as the radius in CircleMarker is defined in pixels

In [18]:

```
import folium

mapObj = folium.Map(location=[23.437730075416685, 72.585859375],
                    zoom_start=6)

folium.CircleMarker(location=[23.437730075416685, 72.585859375],
              radius=50
              ).add_to(mapObj)

mapObj.save('output.html')
mapObj
```

Out[18]:

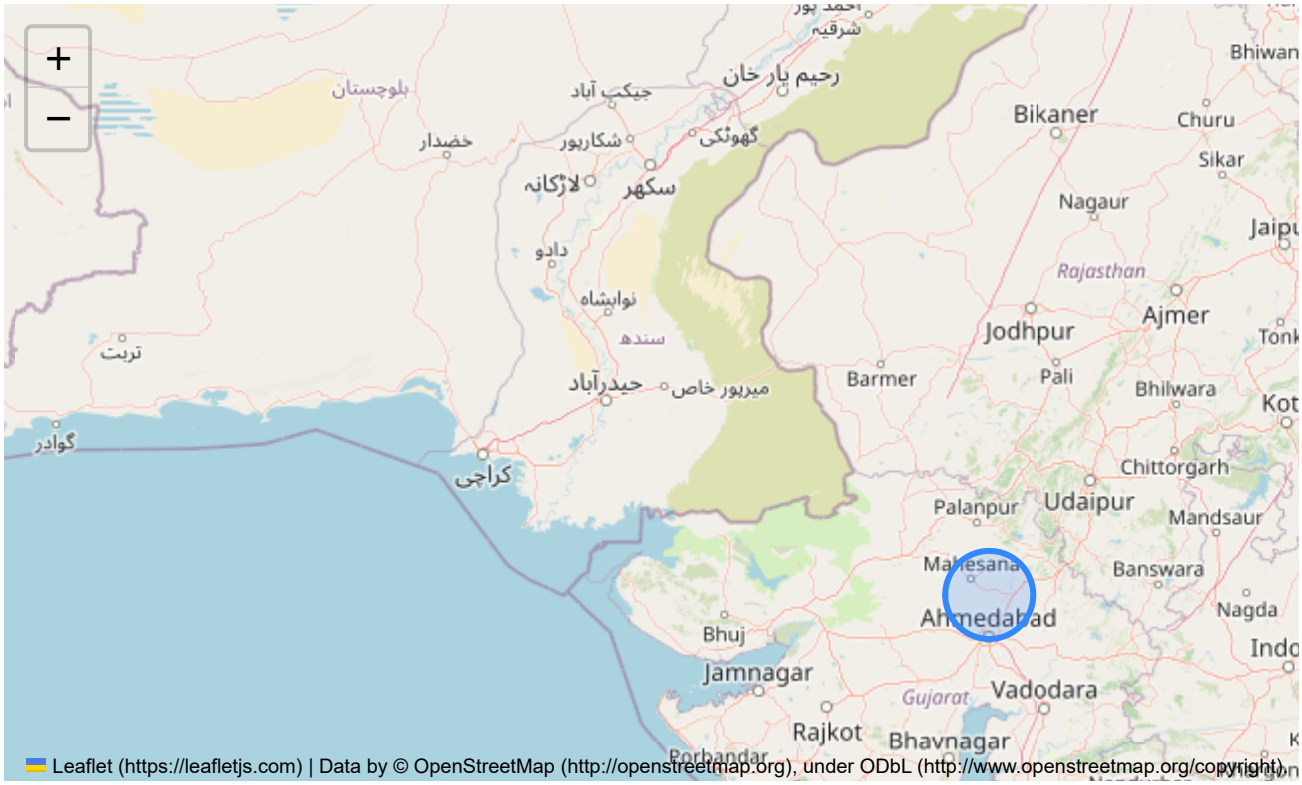


Circle with tooltip and popup

In [19]:

```
import folium

mapObj = folium.Map(location=[23.437730075416685, 72.585859375],
                    zoom_start=6)

folium.Circle(location=[23.437730075416685, 72.585859375],
              radius=50000,
              fill=True,
              tooltip="This is a tooltip text",
              popup=folium.Popup("""<h2>This is a popup</h2><br/>
              This is a <b>new line</b><br/>
              <a href='https://www.ljku.edu.in'> hi</a>""", max_width=500)
              ).add_to(mapObj)

mapObj.save('output.html')
mapObj
```

Out[19]:

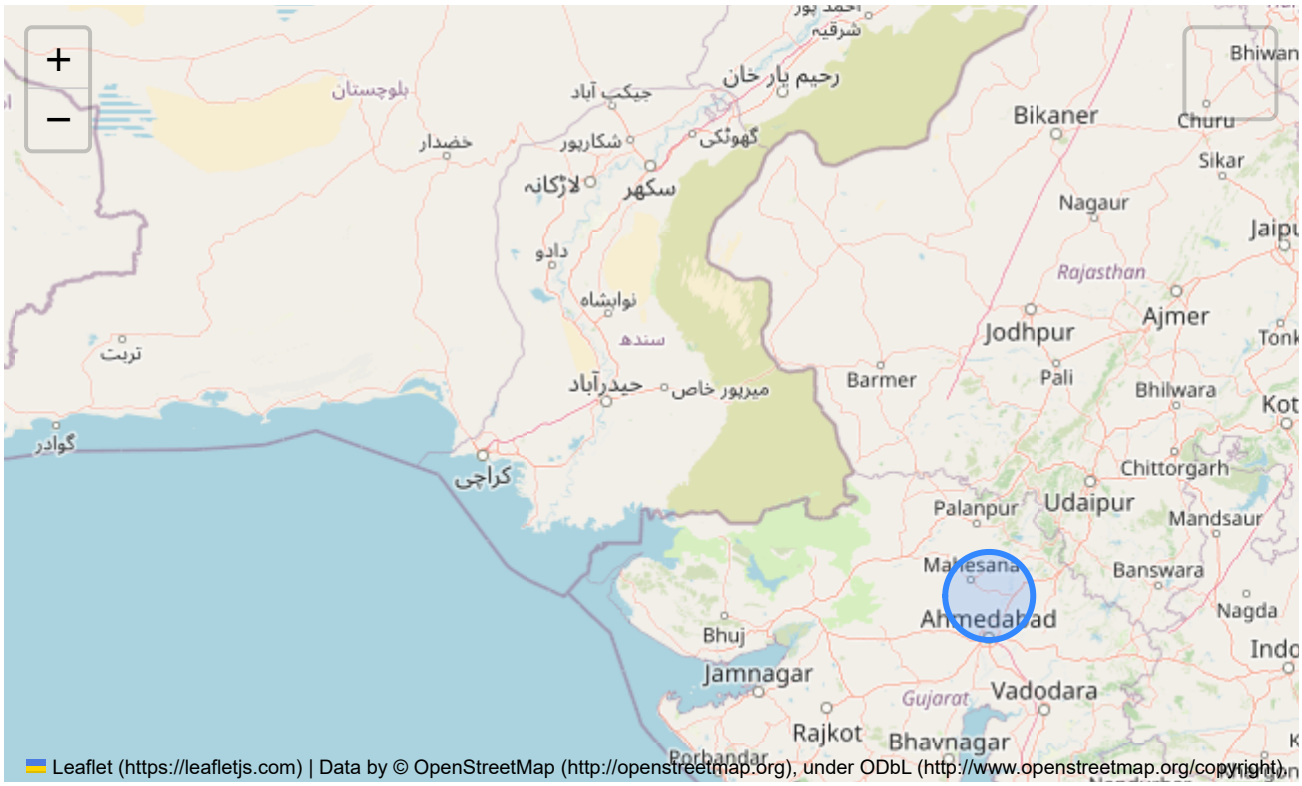


Keep shapes in different layer

In [20]:

```
from os import name
import folium

mapObj = folium.Map(location=[23.437730075416685, 72.585859375],
                    zoom_start=6)

# create a layer on the map object
shapesLayer = folium.FeatureGroup(name="circles").add_to(mapObj)

folium.Circle(location=[23.437730075416685, 72.585859375],
              radius=50000,
              fill=True
              ).add_to(shapesLayer)

folium.LayerControl().add_to(mapObj)

mapObj.save('output.html')
mapObj
```

Out[20]:



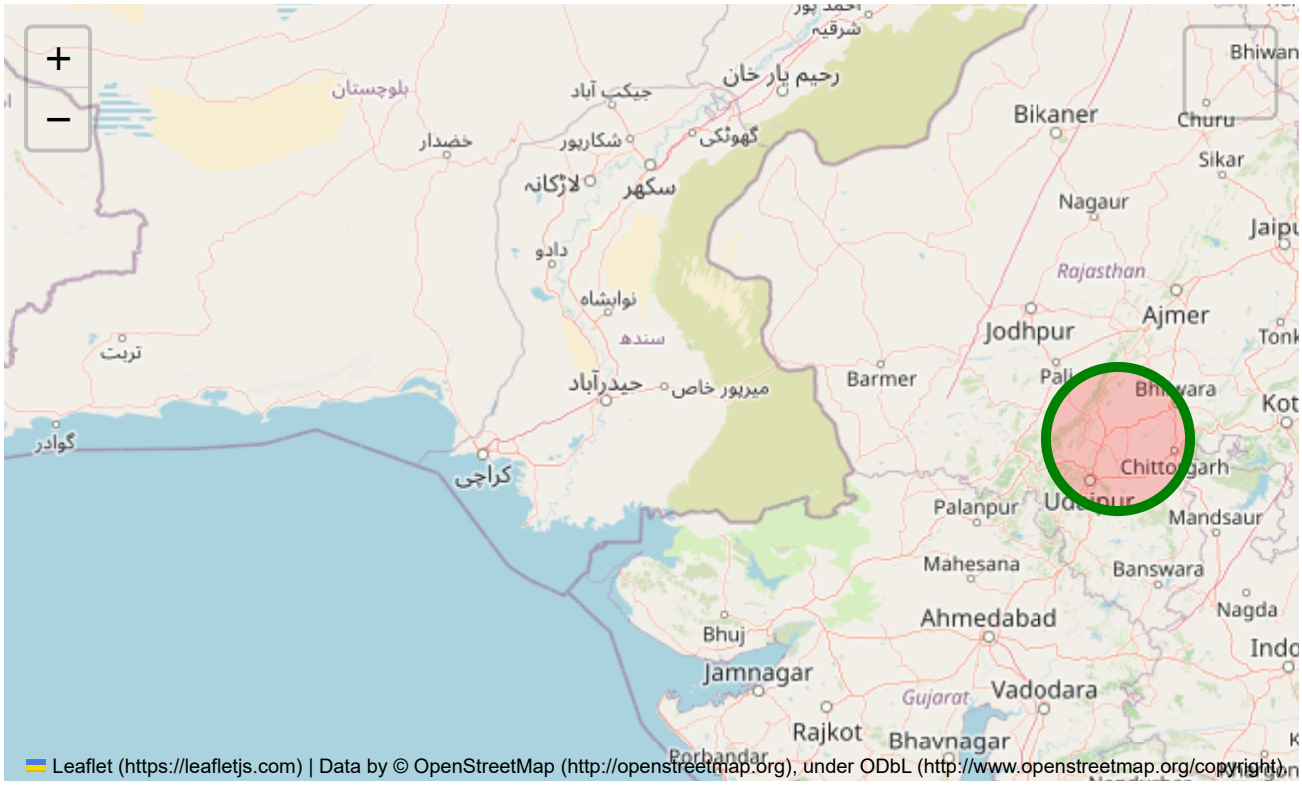
In [21]:

```
import folium

mapObj = folium.Map(location=[23.437730075416685, 72.585859375],
                    zoom_start=6)

shapesLayer = folium.FeatureGroup(name="circles").add_to(mapObj)

circlesData = [
    [25, 74, 80000],
    [22, 79, 60000],
    [26, 82, 90000]
]

for cData in circlesData:
    folium.Circle(location=[cData[0], cData[1]],
                  radius=cData[2],
                  weight=5,
                  color='green',
                  fill_color='red',
                  tooltip="Tooltip text",
                  popup=folium.Popup("""<h2>This is a popup</h2><br/>
                    This is a <b>new line</b><br/>
                     <a href='https://www.ljku.edu.in'> hi</a>""",
                                     max_width=500)
                  ).add_to(shapesLayer)

folium.LayerControl().add_to(mapObj)

mapObj.save('output.html')
```

In [22]:

```
mapObj
```

Out[22]:



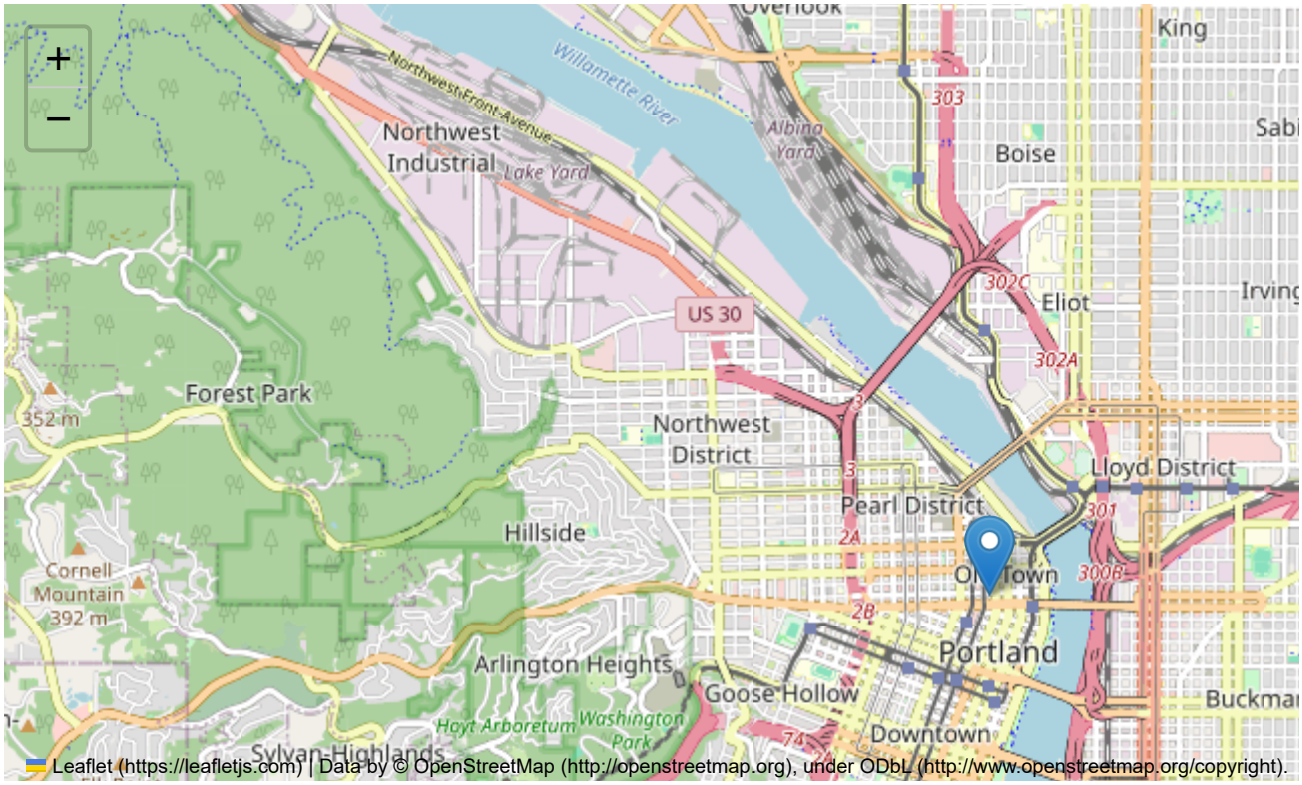
FeatureGroup()

- Folium is a Python library used for visualizing geospatial data with Leaflet.js maps. The FeatureGroup() class in Folium is used to create a container for grouping multiple layers or markers. It allows for the management of layers as a single entity, making it easier to add or remove them from the map.
- To use FeatureGroup() in Folium, you need to first create a Map() object and then add one or more FeatureGroup() objects to it. You can add various types of layers to a FeatureGroup(), including markers, polygons, and lines, and then add the group to the map using the add_to() method.

In [23]:

```
import folium

# Create a map object
m = folium.Map(location=[45.5236, -122.6750], zoom_start=13)

# Create a FeatureGroup object
fg = folium.FeatureGroup(name="My Feature Group")

# Add a marker to the FeatureGroup
folium.Marker(location=[45.5236, -122.6750],
              popup="Portland, OR").add_to(fg)

# Add the FeatureGroup to the map
fg.add_to(m)

# Display the map
m

```

Out[23]:



folium.features is a module in the Folium library that provides various classes and functions for creating visual features on Leaflet maps. This module contains classes for creating markers, icons, polylines, polygons, rectangles, circles, and other map features.

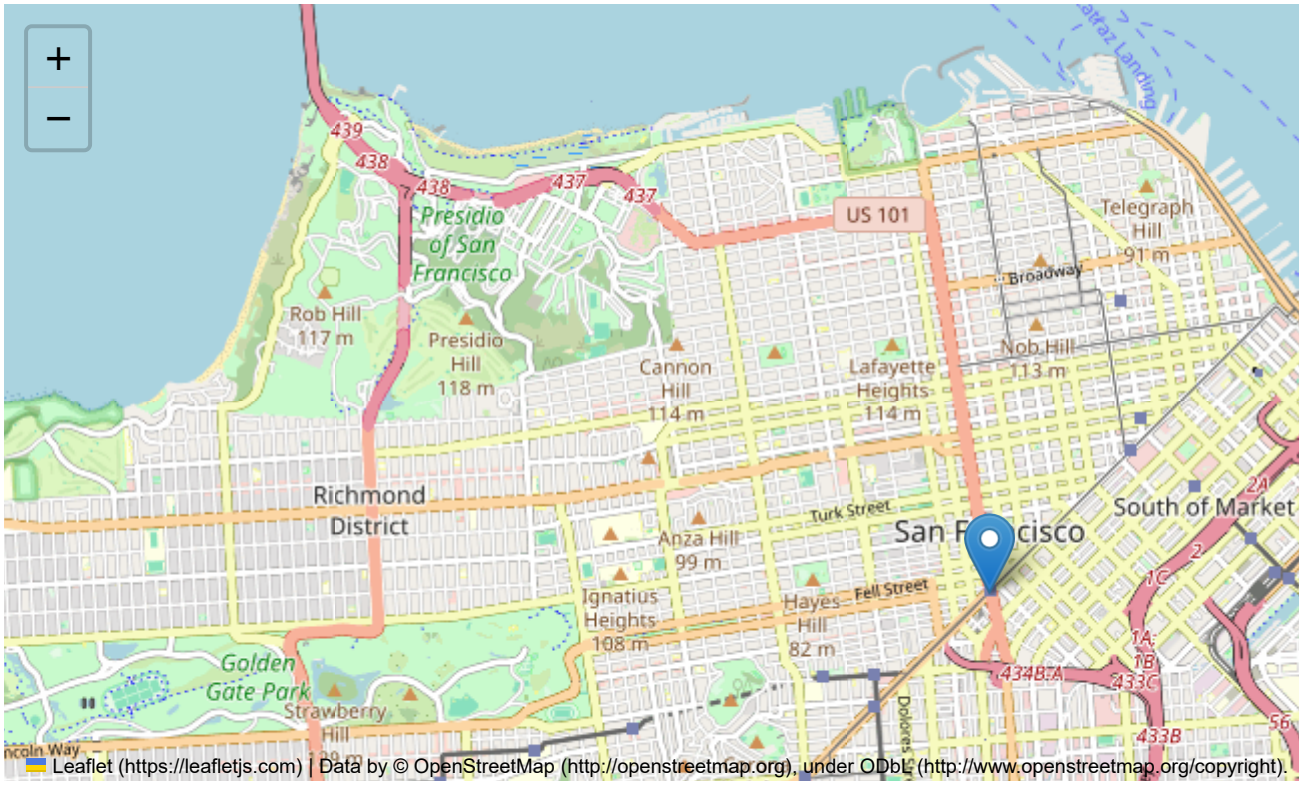
Here are some examples of the classes and functions available in the folium.features module:

- Marker: A class used to create a marker on a Leaflet map. It takes a location argument specifying the latitude and longitude of the marker, and various other optional arguments such as popup and icon to customize the appearance of the marker.
- Icon: A class used to create an icon for a marker on a Leaflet map. It takes a icon_url argument specifying the URL of the image file to use as the icon, and various other optional arguments such as icon_size and icon_anchor to customize the appearance and position of the icon.
- PolyLine: A class used to create a polyline on a Leaflet map. It takes a locations argument specifying the latitude and longitude coordinates of the vertices of the polyline, and various other optional arguments such as color and weight to customize the appearance of the polyline.
- Polygon: A class used to create a polygon on a Leaflet map. It takes a locations argument specifying the latitude and longitude coordinates of the vertices of the polygon, and various other optional arguments such as color and fill_color to customize the appearance of the polygon.
- Rectangle: A class used to create a rectangle on a Leaflet map. It takes a bounds argument specifying the latitude and longitude coordinates of the northeast and southwest corners of the rectangle, and various other optional arguments such as color and fill_color to customize the appearance of the rectangle.
- Circle: A class used to create a circle on a Leaflet map. It takes a location argument specifying the latitude and longitude coordinates of the center of the circle, a radius argument specifying the radius of the circle in meters, and various other optional arguments such as color and fill_color to customize the appearance of the circle.

In [24]:

```
import folium

# Create a map object centered on San Francisco
m = folium.Map(location=[37.7749, -122.4194], zoom_start=13)

# Create a marker object with a popup message
marker = folium.features.Marker(location=[37.7749, -122.4194],
                                popup='This is San Francisco!')

# Add the marker to the map
marker.add_to(m)

# Display the map
m

```

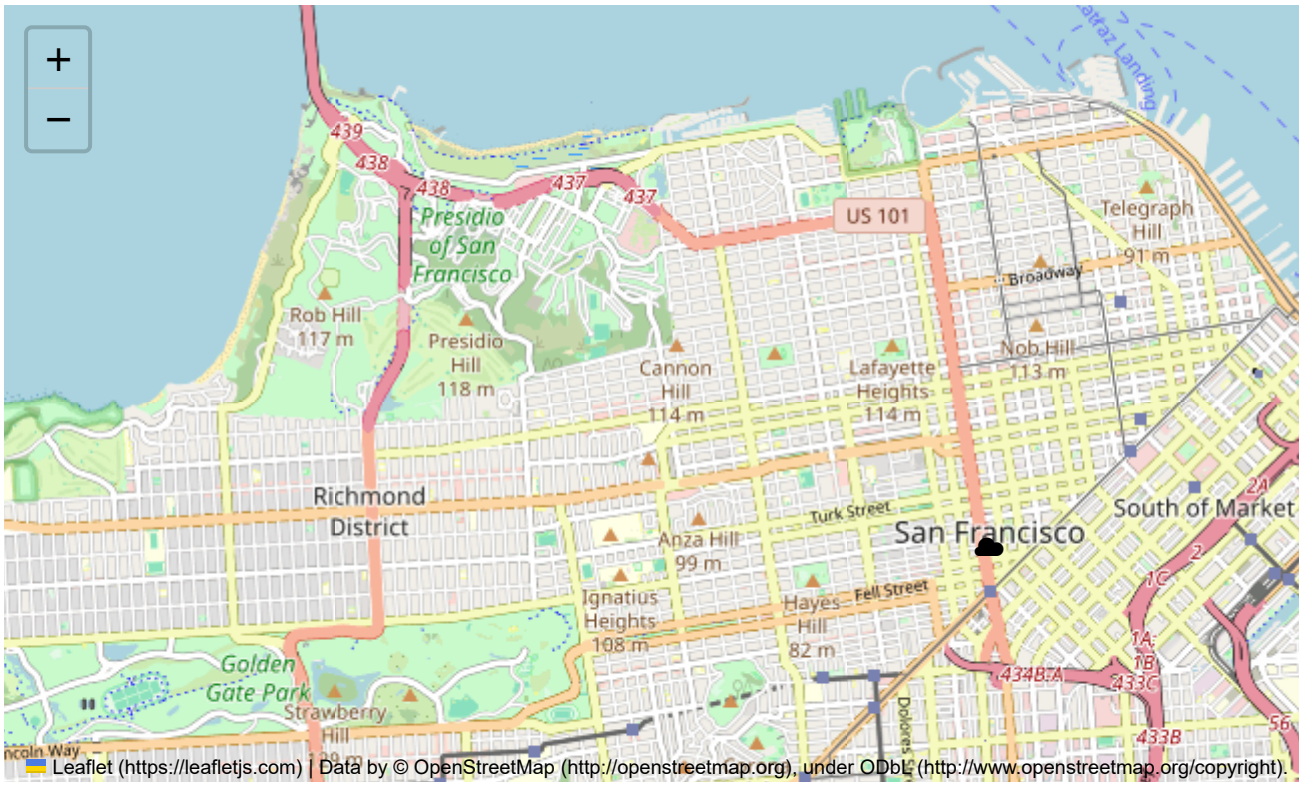
Out[24]:



In [25]:

```
import folium

# Create a map object centered on San Francisco
m = folium.Map(location=[37.7749, -122.4194], zoom_start=13)

# Create an icon object with an image file
icon = folium.features.Icon(icon='cloud')

# Create a marker object with the custom icon and a popup message
marker = folium.Marker(location=[37.7749, -122.4194],
                       popup='This is San Francisco!', icon=icon)

# Add the marker to the map
marker.add_to(m)

# Display the map
m

```

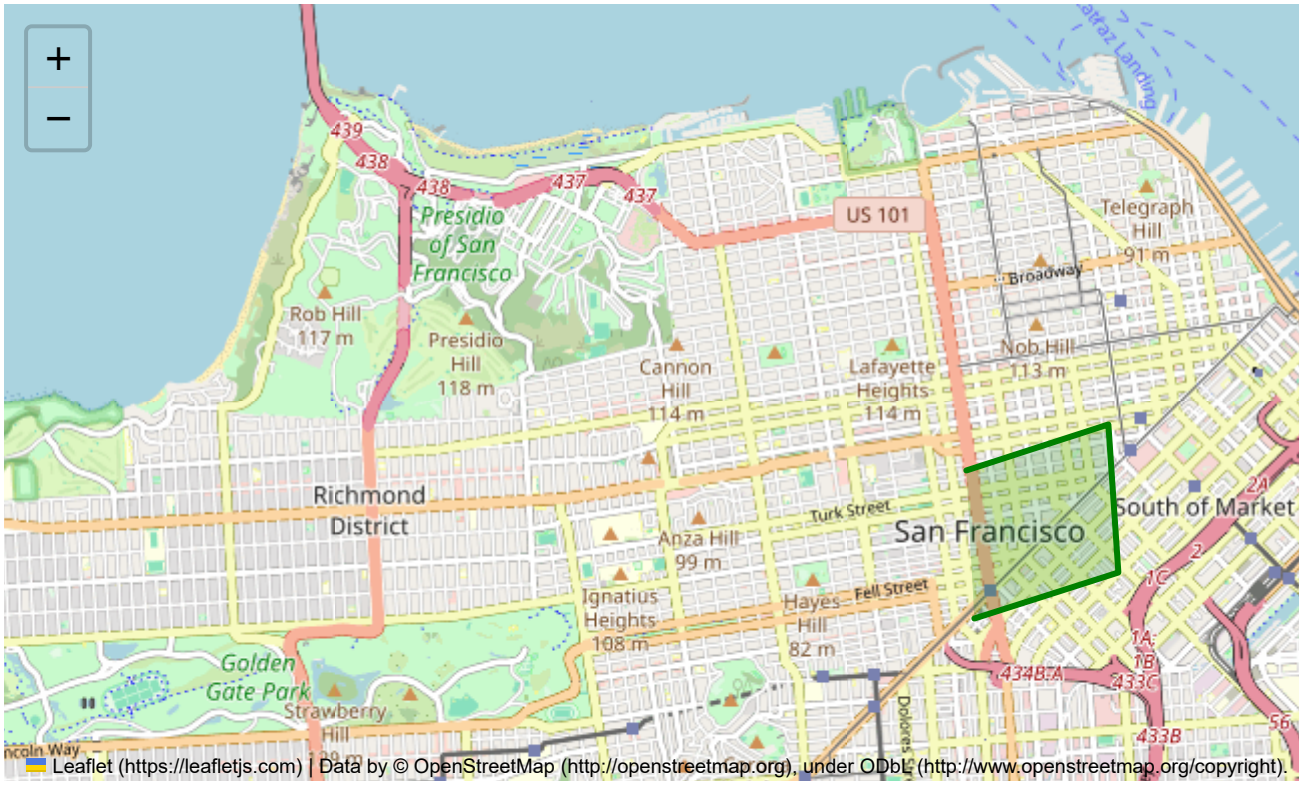
Out[25]:



In [26]:

```
import folium

# Create a map object centered on San Francisco
m = folium.Map(location=[37.7749, -122.4194], zoom_start=13)

# Define the coordinates of the polygon
coords = [[37.7833, -122.4214], [37.7864, -122.4092],
          [37.7764, -122.4083], [37.7733, -122.4206]]

# Create a polygon object with the polygon coordinates
f=folium.features.PolyLine(locations=coords,
                           color='green', fill_color='green')

# Add the polygon to the map
f.add_to(m)

# Display the map
m

```

Out[26]:



add_child

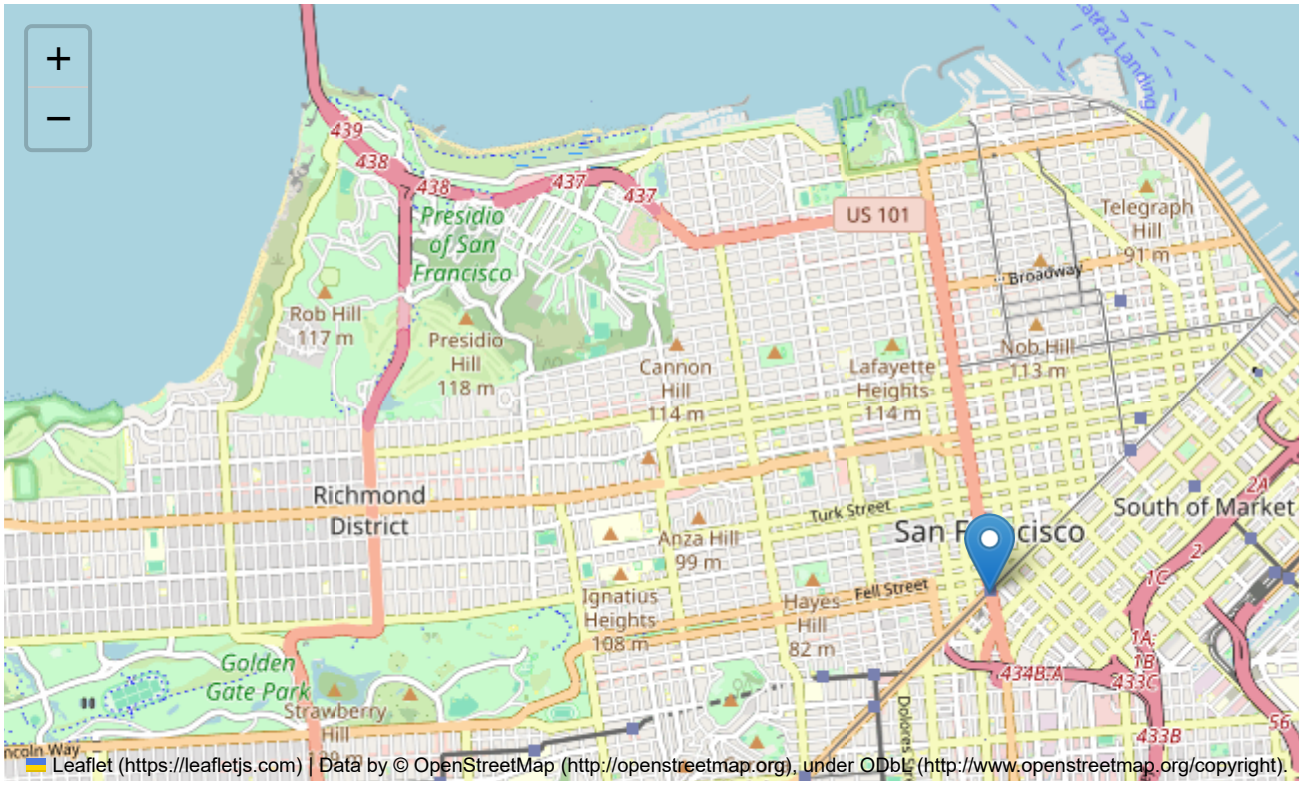
In Folium, add_child() is a method that can be used to add a child element to a parent element. It is commonly used to add layers or features to a map object.

For example, let's say we have created a Map object in Folium:

In [27]:

```
import folium

m = folium.Map(location=[37.7749, -122.4194], zoom_start=13)
marker = folium.Marker(location=[37.7749, -122.4194])
#Then, we can add this marker to the map using the add_child() method:
m.add_child(marker)
m
```

Out[27]:



Download the dataset and read it into a *pandas* dataframe:

In [28]:

```
import pandas as pd
df_incidents = pd.read_csv('Police_Department_Incidents_-_Previous_Year__2016_.cs
```

In [29]:
```
1 df_incidents.head()
```

Out[29]:

	IncidntNum	Category	Descript	DayOfWeek	Date	Time	PdDistrict	Resolution	A
0	120058272	WEAPON LAWS	POSS OF PROHIBITED WEAPON	Friday	01/29/2016 12:00:00 AM	11:00	SOUTHERN	ARREST, BOOKED	80 of B
1	120058272	WEAPON LAWS	FIREARM, LOADED, IN VEHICLE, POSSESSION OR USE	Friday	01/29/2016 12:00:00 AM	11:00	SOUTHERN	ARREST, BOOKED	80 of B
2	141059263	WARRANTS	WARRANT ARREST	Monday	04/25/2016 12:00:00 AM	14:59	BAYVIEW	ARREST, BOOKED	KEI SH
3	160013662	NON-CRIMINAL	LOST PROPERTY	Tuesday	01/05/2016 12:00:00 AM	23:50	TENDERLOIN	NONE	JON OFA
4	160002740	NON-CRIMINAL	LOST PROPERTY	Friday	01/01/2016 12:00:00 AM	00:30	MISSION	NONE	16 MI

- IncidntNum: Incident Number
- Category: Category of crime or incident
- Descript: Description of the crime or incident
- DayOfWeek: The day of week on which the incident occurred
- Date: The Date on which the incident occurred
- Time: The time of day on which the incident occurred
- PdDistrict: The police department district
- Resolution: The resolution of the crime in terms whether the perpetrator was arrested or not
- Address: The closest address to where the incident took place
- X: The longitude value of the crime location
- Y: The latitude value of the crime location
- Location: A tuple of the latitude and the longitude values
- PdId: The police department ID

In [30]:
```
1 df_incidents.shape
```

Out[30]: (150500, 13)

So the dataframe consists of 150,500 crimes, which took place in the year 2016. In order to reduce computational cost, let's just work with the first 100 incidents in this dataset.

In [31]:
```
1 # get the first 100 crimes in the df_incidents dataframe
2 limit = 100
3 df_incidents = df_incidents.iloc[0:limit, :]
```

In [32]:
```
1 df_incidents.shape
```

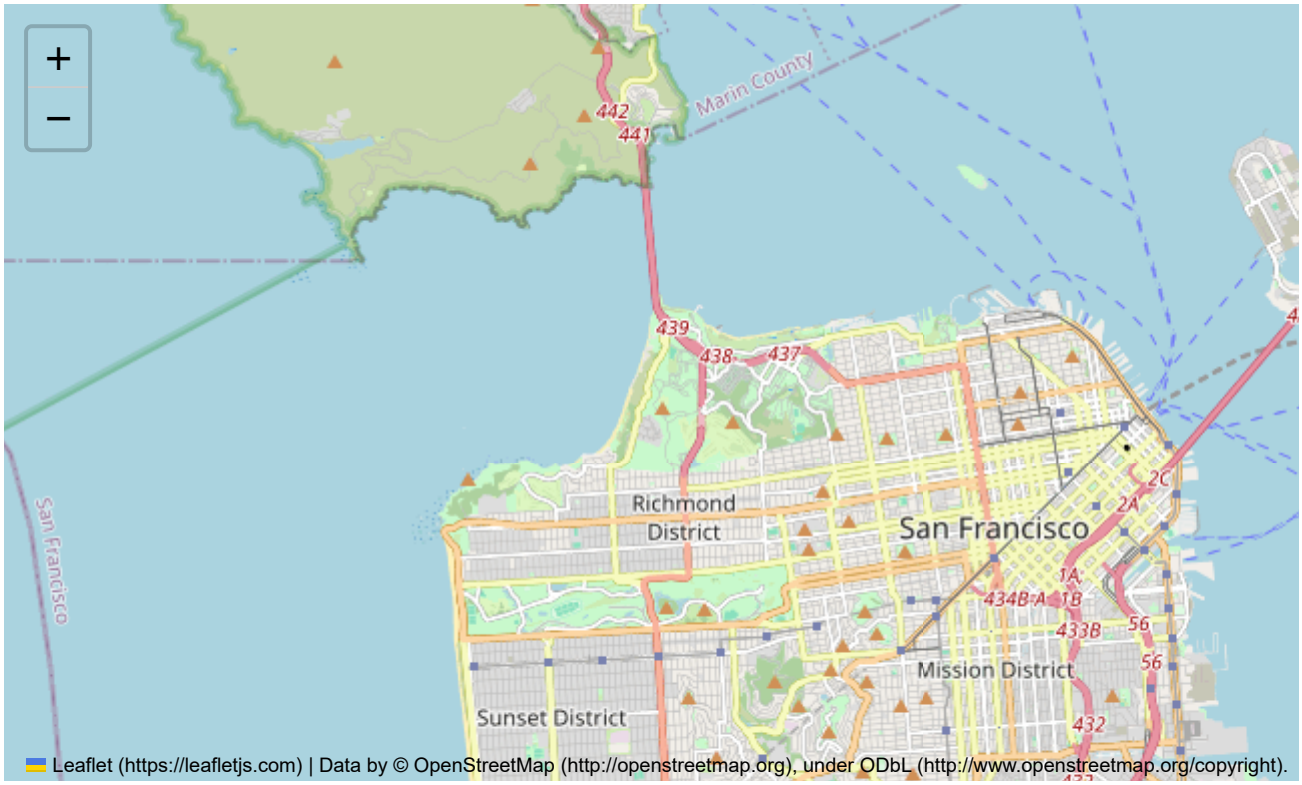
Out[32]: (100, 13)

Now that we reduced the data a little, let's visualize where these crimes took place in the city of San

In [33]:
```
# San Francisco latitude and longitude values
latitude = 37.77
longitude = -122.42
```

In [34]:
```
# create map and display it
sanfran_map = folium.Map(location=[latitude, longitude],
                         zoom_start=12)

# display the map of San Francisco
sanfran_map
```

Out[34]:

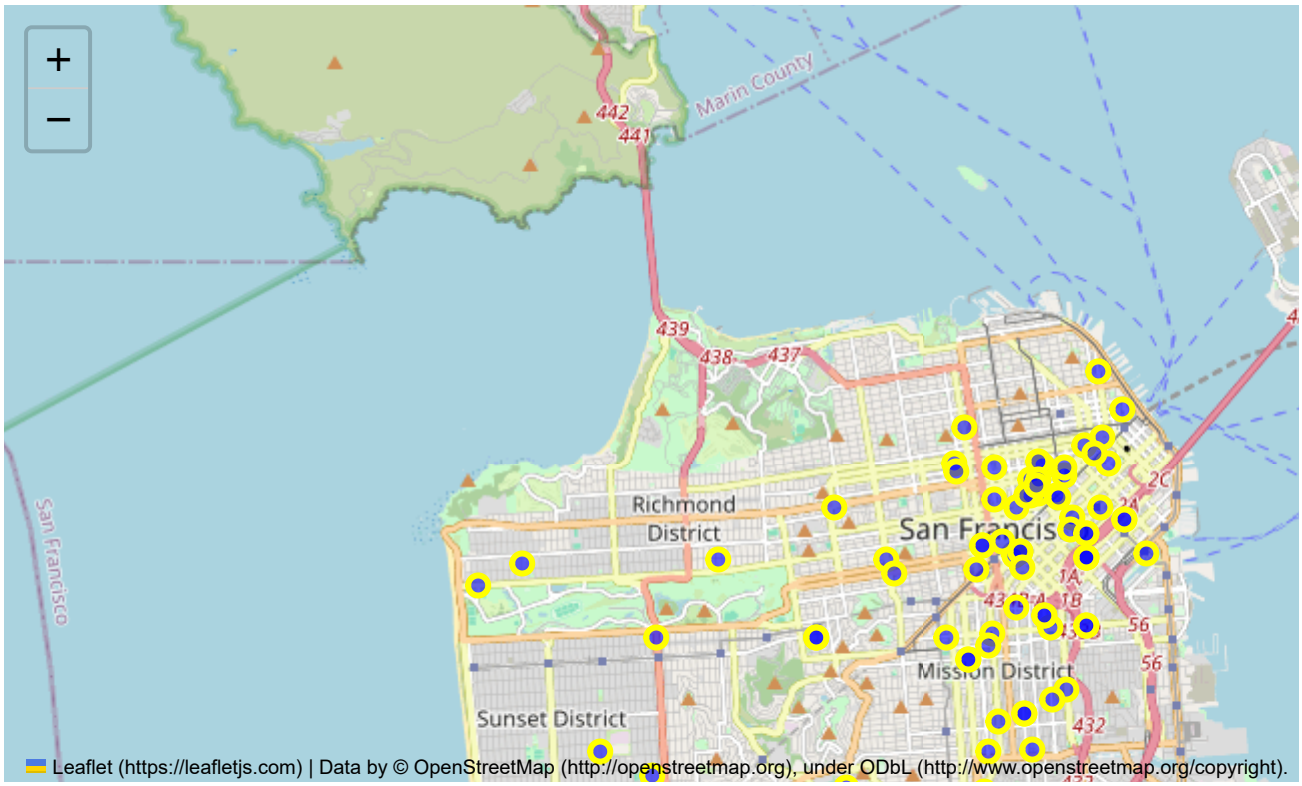


Now let's superimpose the locations of the crimes onto the map. The way to do that in Folium is to create a feature group with its own features and style and then add it to the sanfran_map.

In [35]:

```
# instantiate a feature group for the incidents in the dataframe
incidents = folium.map.FeatureGroup()

# loop through the 100 crimes and add each to the incidents feature group
for lat, lng, in zip(df_incidents.Y, df_incidents.X):
    incidents.add_child(
        folium.features.CircleMarker(
            [lat, lng],
            radius=5, # define how big you want the circle markers to be
            color='yellow',
            fill=True,
            fill_color='blue',
            fill_opacity=0.6
        )
    )

# add incidents to map
sanfran_map.add_child(incidents)
```

Out[35]:

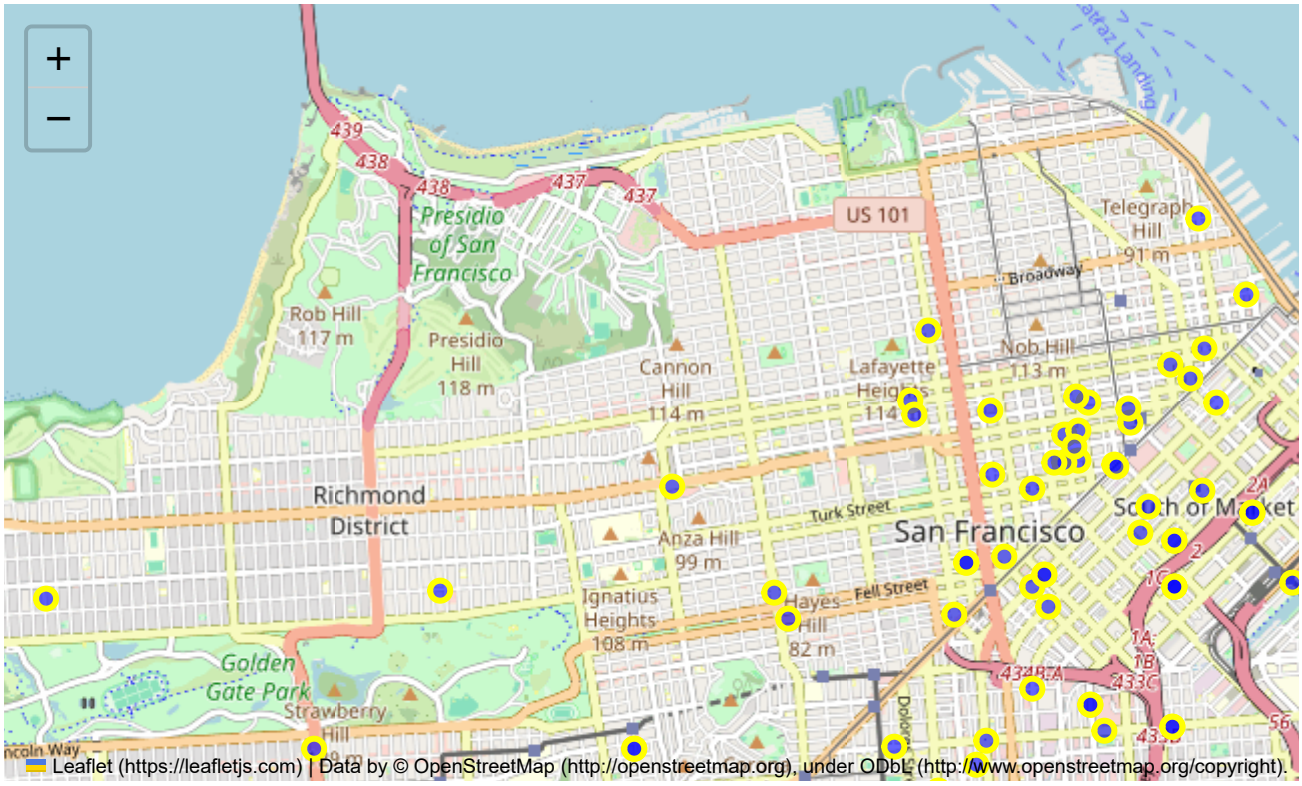


You can also add some pop-up text that would get displayed when you hover over a marker. Let's make each marker display the category of the crime when hovered over.

In [36]:

```
import pandas as pd
import folium

# Load the crime dataset
df_incidents = pd.read_csv('Police_Department_Incidents_-_Previous_Year__2016_.cs
limit = 100
df_incidents = df_incidents.iloc[0:limit, :]
# Create a map centered on San Francisco
m = folium.Map(location=[37.7749, -122.4194], zoom_start=13)

# Create a feature group for the incidents
incidents = folium.map.FeatureGroup()

# Loop through each crime in the dataset and add a circle marker to the feature g
for lat, lon, category in zip(df_incidents.Y, df_incidents.X,
                              df_incidents.Category):
    popup_text = f'Category: {category}' # Create pop-up text
                               # with the category of the crime
    marker = folium.CircleMarker(location=[lat, lon],
            radius=5, color='yellow', fill=True,
            fill_color='blue', fill_opacity=0.6, popup=popup_text)
    marker.add_to(incidents)

# Add the incidents feature group to the map
incidents.add_to(m)

# Display the map
m


```

Out[36]:

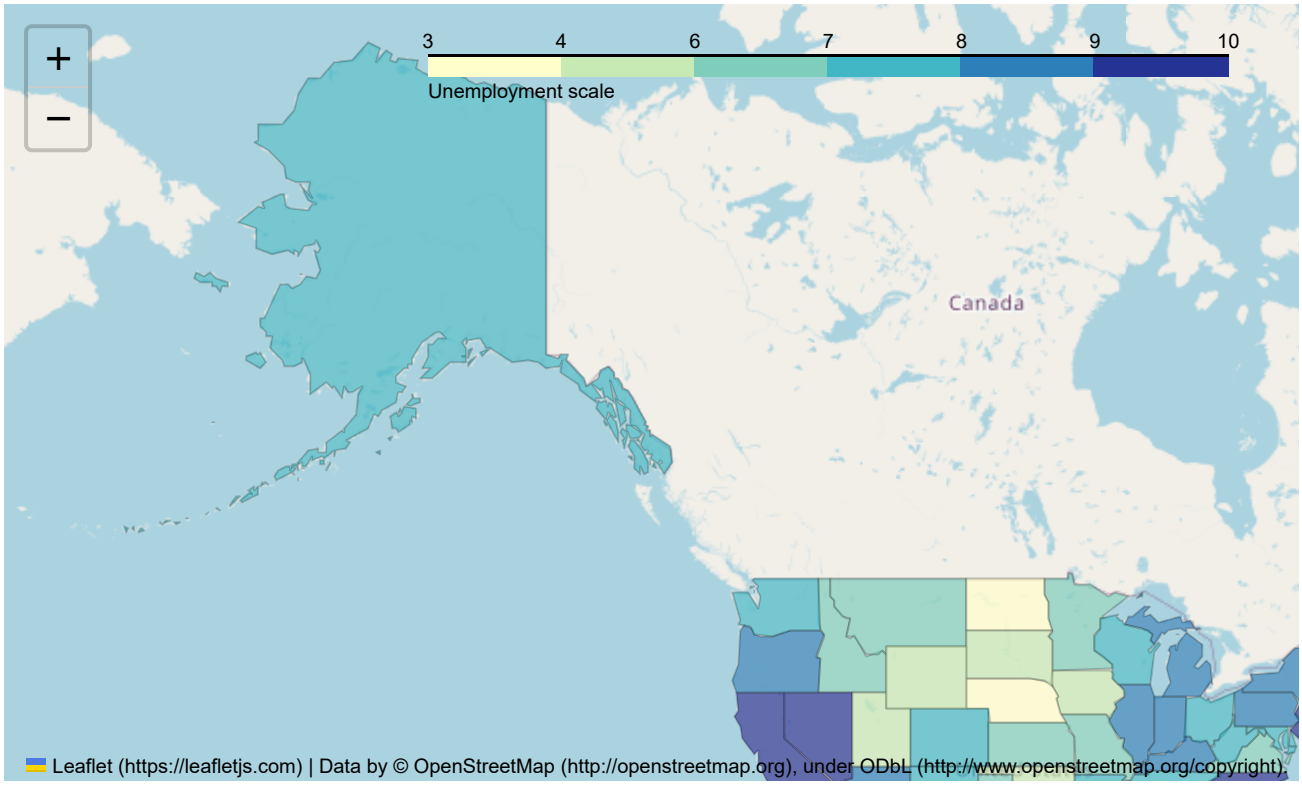


Choropleth Maps

In [37]:

```
usa_state = folium.Map(location=[48, -102], zoom_start=3)
folium.Choropleth(
    geo_data = 'us-states.json',                    #json
    name ='choropleth',
    data = pd.read_csv("US_Unemployment_Oct2012.csv"),
    columns = ['State', 'Unemployment'], #columns to work on
    key_on ='feature.id',
    fill_color ='YlGnBu',      #I passed colors Yellow,Green,Blue
    fill_opacity = 0.7,
    line_opacity = 0.2,
   legend_name = "Unemployment scale"
).add_to(usa_state)
usa_state
```

Out[37]:



- In this example, we first load the US_Unemployment_Oct2012.csv dataset using pd.read_csv(). This dataset contains the unemployment rate for each US state in October 2012.
- We then create a Map object centered on the US using folium.Map(). Next, we add a choropleth layer to the map using folium.Choropleth(). The geo_data argument specifies the shapefile for the US states, which is loaded from the us-states.json file. The data argument specifies the unemployment dataset, and the columns argument specifies which columns in the dataset contain the state names and unemployment rates. The key_on argument specifies how to match the state names in the shapefile with the state names in the dataset. The fill_color, fill_opacity, and line_opacity arguments specify the color and opacity of the choropleth layer, and the legend_name argument specifies the name of the legend.
- We then add a layer control to the map using folium.LayerControl(), which allows the user to toggle the visibility of each layer on the map.

In [ ]:

```
1
```