GGE5404: Online Spatial Data Handling

Assignment 2

Name: Mohammadali Rahnama

Student Number: 3709515

Date: 2023-03-09

Part A. Static Visualization

Access data from “Daily Climate Observations” from GeoMET API (https://eccc-msc.github.io/opendata/msc-geomet/readme_en/ (https://eccc-msc.github.io/opendata/msc-geomet/readme_en/)). Extract data, using Jupyter Notebook for the closest observation station to '560 Rochester St, Ottawa, ON, CA' for the entirety of 2011.

First, we install and import the required packages in order to use them throughout the exercise.

In [1]:
```
import requests
import geocoder
from ipyleaflet import Map, WMSLayer, SplitMapControl
import folium
import matplotlib.pyplot as plt
```

In this code we take the provided address as input and convert it into a latitude and longitude location using the geocoder library.

Then, we use the latitude and longitude to search for the closest weather station within a specified bounding box size.

The code makes a request to the MSC GeoMet API endpoint to retrieve climate station data within the bounding box and then iteratively decreases the bounding box size until only one station is found.

The MSC GeoMet platform - Climate Stations: https://api.weather.gc.ca/collections/climate-stations/queryables (https://api.weather.gc.ca/collections/climate-stations/queryables)

The closest station information is extracted and printed, including the station name, ID, latitude, and longitude.

If no station is found, the code prints a message indicating that no climate station was found within the provided location.

In [2]:

```python
# Convert the address to a location
address = '560 Rochester St, Ottawa, ON, CA'
location = geocoder.arcgis(address).latlng

# Define the minimum and maximum sizes of the bounding box
min_size = 0.001
max_size = 0.5

# Define the API endpoint and initial query parameters
endpoint = 'https://api.weather.gc.ca/collections/climate-stations/items'
params = {
    'bbox': ','.join(str(coord) for coord in [
        location[1] - max_size,  # min x (longitude)
        location[0] - max_size,  # min y (latitude)
        location[1] + max_size,  # max x (longitude)
        location[0] + max_size   # max y (latitude)
    ]),
    'f': 'json',
    'limit': 1
}

# Initialize the closest station information
closest_station = None
closest_distance = float('inf')

# Loop until the bounding box size reaches the minimum size
while max_size > min_size:
    # Send the API request and get the response
    response = requests.get(endpoint, params=params)

    # Check if the request was successful
    if response.status_code == 200:
        # Convert the response JSON to a Python dictionary
        data = response.json()

        # Extract the number of climate stations found
        count = data.get('numberMatched')
        if count > 0:
            print(f'Found {count} climate stations within the bounding box, making the bounding box smaller until only one statio

        # Check if any stations are found
        if count > 0:
            # Loop over the stations and find the closest one
            for feature in data.get('features'):
                # Extract the location of the station
                lat = feature.get('geometry', {}).get('coordinates', [])[1]
                lon = feature.get('geometry', {}).get('coordinates', [])[0]

                # Calculate the distance between the station and the provided location
                distance = ((lat - location[0])**2 + (lon - location[1])**2)**0.5

                # Check if the station is closer than the current closest station
                if distance < closest_distance:
                    # Update the closest station information
                    closest_station = feature
                    closest_distance = distance

            # Make the bounding box smaller and update the query parameters
            size = max_size / 2
            params['bbox'] = ','.join(str(coord) for coord in [
                location[1] - size,  # min x (longitude)
                location[0] - size,  # min y (latitude)
                location[1] + size,  # max x (longitude)
                location[0] + size   # max y (latitude)
            ])
            max_size = size
        else:
            # Make the bounding box smaller and update the query parameters
            size = max_size / 2
            params['bbox'] = ','.join(str(coord) for coord in [
                location[1] - size,  # min x (longitude)
                location[0] - size,  # min y (latitude)
                location[1] + size,  # max x (longitude)
                location[0] + size   # max y (latitude)
            ])
            max_size = size
    else:
        # Handle errors
        print(f'Request failed with status code {response.status_code}: {response.reason}')
        break

# Check if a closest station is found
if closest_station:
    # Extract the station information
    station_name = closest_station.get('properties', {}).get('STATION_NAME', '')
    stn_id = closest_station.get('properties', {}).get('STN_ID', '')
```

```
    station_LATITUDE = closest_station.get('properties', {}).get('LATITUDE', '')
    station_LONGITUDE = closest_station.get('properties', {}).get('LONGITUDE', '')


    # Print the station information
    print(f'The closest station is: {station_name} ID: {stn_id} Longitude: {lon} Latitude: {lat} .')
else:
    print('No climate station found within the provided location.')
```

```
Found 83 climate stations within the bounding box, making the bounding box smaller until only one station is found.
Found 49 climate stations within the bounding box, making the bounding box smaller until only one station is found.
Found 25 climate stations within the bounding box, making the bounding box smaller until only one station is found.
Found 13 climate stations within the bounding box, making the bounding box smaller until only one station is found.
Found 5 climate stations within the bounding box, making the bounding box smaller until only one station is found.
Found 3 climate stations within the bounding box, making the bounding box smaller until only one station is found.
The closest station is: OTTAWA CDA RCS ID: 30578 Longitude: -75.71666666666667 Latitude: 45.38333333333333 .
```

1. Plot minimum, mean and maximum temperature on one plot.

In this Python code we retrieve the minimum, mean, and maximum temperature values for a found station's location (defined by lat and lon) and year 2011 (defined by year__annee) using the Environment and Climate Change Canada's Annual Historical Canadian Climate Data (AHCCD) API.

Environment and Climate Change Canada's Annual Historical Canadian Climate Data (AHCCD) API.: [https://api.weather.gc.ca/collections/ahccd-annual/items/ (https://api.weather.gc.ca/collections/ahccd-annual/items/)](https://api.weather.gc.ca/collections/ahccd-annual/items/)

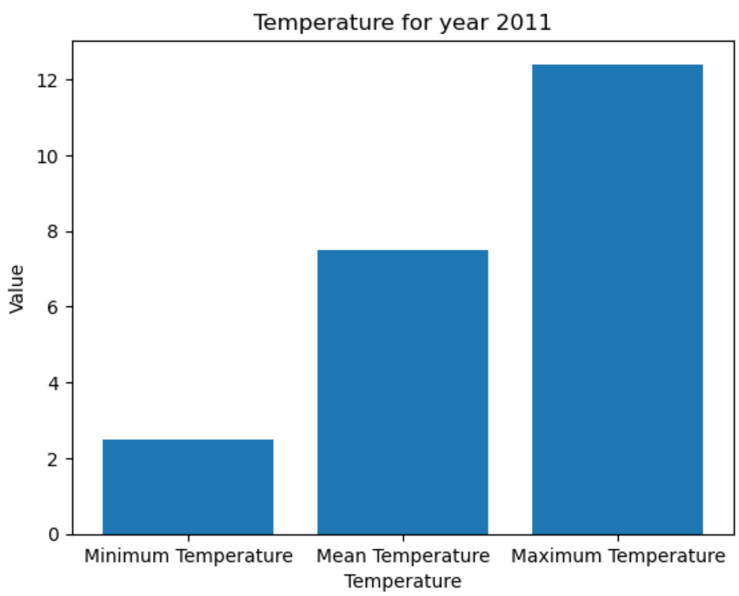
The code constructs a URL to the AHCCD API endpoint and sets query parameters to specify the location, year, and desired temperature properties (temp_min__temp_min, temp_mean__temp_moyenne, and temp_max__temp_max). The requests.get() method is used to send a GET request to the API endpoint with the specified query parameters.

The response from the API is a JSON object, which is parsed using the response.json() method to extract the temperature values for the first feature returned by the API. The minimum, mean, and maximum temperature values are then extracted from this data.

Finally, the temperature values are plotted as a bar plot using the matplotlib library.

In [3]:

```python
import requests

# Define the API endpoint URL and query parameters
url = 'https://api.weather.gc.ca/collections/ahccd-annual/items'
params = {
    'bbox': ','.join(str(coord) for coord in [
        lon - 0.5,  # min x (longitude)
        lat - 0.5,  # min y (latitude)
        lon + 0.5,  # max x (longitude)
        lat + 0.5   # max y (latitude)
    ]),
    'f': 'json',
    'year__annee': '2011',
    'properties': 'temp_min__temp_min,temp_mean__temp_moyenne,temp_max__temp_max'
}

# Send a GET request to the API endpoint with the query parameters
response = requests.get(url, params=params)

# Extract the data from the response JSON
data = response.json()['features'][0]['properties']

# Extract the minimum, mean and maximum temperature values from the data
temp_min = data['temp_min__temp_min']
temp_mean = data['temp_mean__temp_moyenne']
temp_max = data['temp_max__temp_max']

# Create a bar plot of the temperature values
fig, ax = plt.subplots()
ax.bar(['Minimum Temperature', 'Mean Temperature', 'Maximum Temperature'], [temp_min, temp_mean, temp_max])
ax.set_xlabel('Temperature')
ax.set_ylabel('Value')
ax.set_title('Temperature for year 2011')
plt.show()
```



2. Create a geographic map – plot the location of the station in a folium map

In this code we create a new folium map centered at the found stations's location using the folium.Map() function.

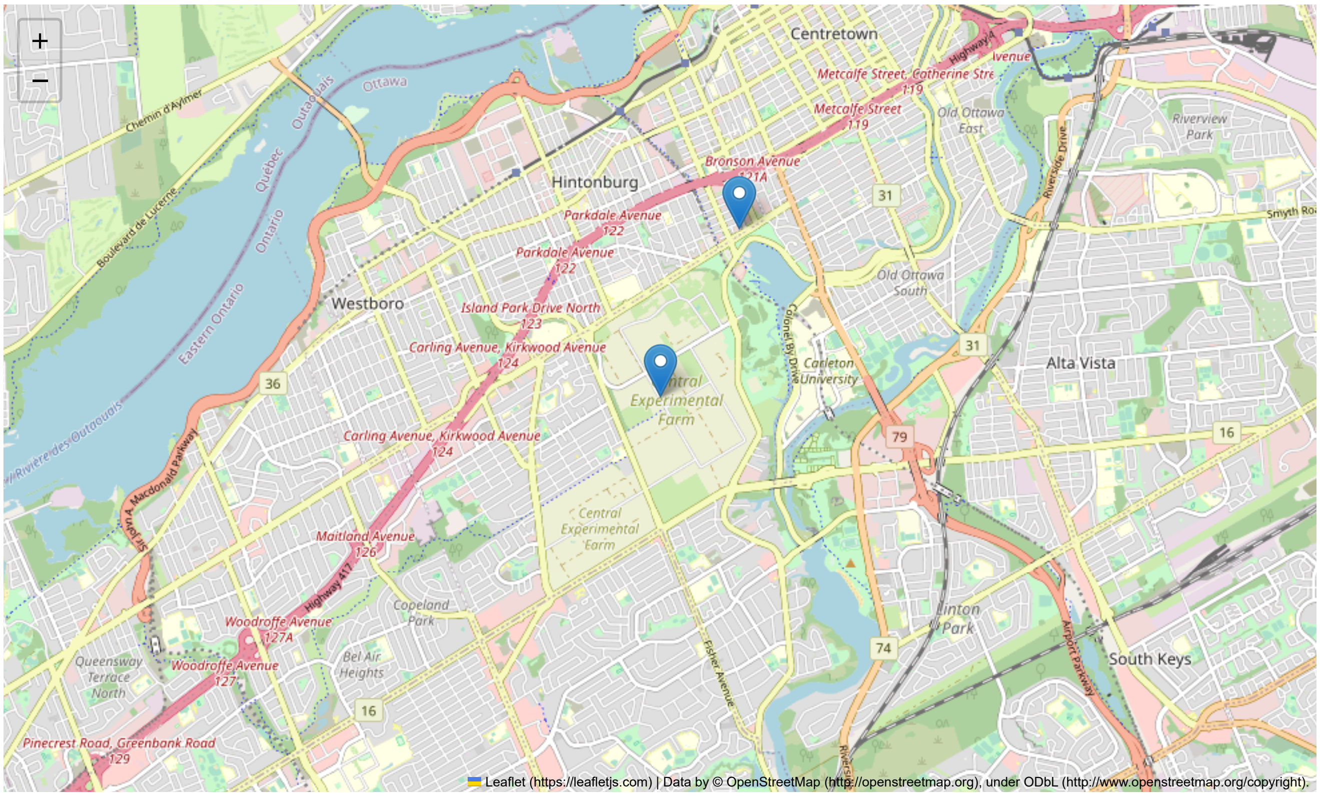
We also adds a marker for both the address and the station with popups showing their respective information.

Next, we creates an HTML popup for the address marker with the address information. It then creates an HTML popup for the station marker with the station name, station ID, latitude, and longitude information.

Finally, we displays the map with both markers and their popups.

In [4]:
```python
# create a map centered on the address location
station_location = [lat,lon]
map = folium.Map(location=station_location, zoom_start=13)

# add a marker for the address with a popup showing its information
address_popup = f'<b>Address:</b> {address}'
address_marker = folium.Marker(location=location, popup=address_popup)
address_marker.add_to(map)

# add a marker for the station with a popup showing its information
station_popup = f'<p><b>Station:</b> {station_name}</p></b><p><b>Station ID:</b> {stn_id}</p><p><b>Latitude:</b> {lon}</p><p><b>L
station_marker = folium.Marker(location=station_location, popup=station_popup)
station_marker.add_to(map)

# display the map
map
```

Out[4]:



Part B. Interactive Visualization

Create an interactive map in ipyleaflet, accessing two WMS services and viewing them using the Split Map control. You can select the WMS services of your choosing.

To summarize what this code does:

1. We use the geocoder library to convert the address 'Fredericton, NB, Canada' to a latitude and longitude coordinate pair.
2. We create a new Map object centered on the resulting location with a zoom level of 13.
3. We create two WMSLayer objects for the Environment and Climate Change Canada and Federal Government's Canadian Public Transit Systems WMS servers.

Environment and Climate Change Canada: https://eccc-msc.github.io/open-data/msc-geomet/readme_en/#usage-tutorials-and-technical-documentation (https://eccc-msc.github.io/open-data/msc-geomet/readme_en/#usage-tutorials-and-technical-documentation) Federal Government's Canadian Public Transit Systems: https://open.canada.ca/data/dataset/b8241e15-2872-4a63-9d36-3083d03e8474/resource/07325fec-ca9e-4554-a6d4-4f5001f79290 (https://open.canada.ca/data/dataset/b8241e15-2872-4a63-9d36-3083d03e8474/resource/07325fec-ca9e-4554-a6d4-4f5001f79290)

4. We add the two WMSLayer objects to the Map.
5. We add a SplitMapControl to the Map, using the weather_wms layer as the left layer and the transit_wms layer as the right layer.
6. We display the Map.

This results in a map that shows the two WMS layers side by side, with the left side showing weather radar data and the right side showing public transit systems.

In [5]:
```python
# Get the location of the address using geocoder
address = "Fredericton, NB, Canada"
location = geocoder.arcgis(address)
latitude, longitude = location.latlng

# Create the map
m = Map(center=(latitude, longitude), zoom=13)

# Weather WMS layer
weather_wms = WMSLayer(
    url='https://geo.weather.gc.ca/geomet',
    layers='RADAR_1KM_RDBR:RADAR_1KM_RDBR',
    transparent=True,
    attribution='Environment and Climate Change Canada'
)

# Transit WMS layer
transit_wms = WMSLayer(
    url='https://maps-cartes.services.geo.ca/server_serveur/services/INFC/public_transit_systems_en/MapServer/WMSServer',
    layers='1',
    transparent=True,
    attribution='Government of Canada'
)

# Add the layers to the map
m.add_layer(weather_wms)
m.add_layer(transit_wms)

# Add the Split Map control with two layers
control = SplitMapControl(
    left_layer=weather_wms,
    right_layer=transit_wms
)
m.add_control(control)

# Show the map
m
```