Geospatial Data Operations

August 11, 2023

1 Geospatial Data Operations

1.1 Part A. Static Visualization

Access data from "Daily Climate Observations" from GeoMET API. Extract data, using the OWSLIB using Jupyter Notebook for the closest observation station to '560 Rochester St, Ottawa, ON, CA' for the entirety of 2011.

Optional Pre-Requisite : Install packages which may not be installed

[ ]:
```
# Installing packages
!pip install owslib requests pandas matplotlib ipyleaflet
```

[2]:
```
# Importing the necessary modules
import json
from geopy.geocoders import Nominatim
from osgeo import ogr, osr
from owslib.ogcapi.features import Features
import pandas as pd
```

First, the address has to be translated to set of coordinate using a geocoding service. One such service is provided by OpenStreetMap Nominatim.

[3]:
```
# Given address
address = "560 Rochester St, Ottawa, ON, CA"

# Initializing a geolocator object from the geopy library's Nominatim class␣
↪with a specified user agent string.
geolocator = Nominatim(user_agent="my-application")

# Retrieve the coordinates
address_loc = geolocator.geocode(address)

# Retrieving the coordinates of address and assigning it to variables for␣
↪further use
lon, lat = address_loc.longitude, address_loc.latitude
```

Now, define the buffer size (in km), projection and bbox.

[4]:
```
# This allows to display plots and charts directly in the output cell of the␣
↪notebook.
%matplotlib inline
```

[5]:
```
# Buffer size in kilometres
buffer = 2

# ESPG code of the preferred projection to create the buffer
# NAD83 / Statistics Canada Lambert
projection = 3347
# Parameters formatting for the OGC API - Features request

# Bounding box a little bigger than buffer size

# The buffer needs to be transformed in degrees to get the coordinates of the␣
↪corners of the bounding box:
# Latitude: 1 km   0.009°
# Longitude (at the 49th parallel): 1 km   0.014°
bbox = [
    lon - buffer * 0.02,
    lat - buffer * 0.01,
    lon + buffer * 0.02,
    lat + buffer * 0.01,
]
```

- Swagger UI is a helpful tool to understand the API in a GUI format with response codes .
- Maximum limit is 10,000.
- Parameters can be chosen based on requirements using Queryables

[6]:
```
# Retrieval of station data by passing the parameters using the OGC API
oafeat = Features("https://api.weather.gc.ca/")
station_data = oafeat.collection_items("climate-daily", bbox=bbox,␣
↪LOCAL_YEAR=2011,limit=10000,PROVINCE_CODE='ON')

# Verification of the retrieved data and converting to JSON format.
if "features" in station_data:
    station_data = json.dumps(station_data, indent=2)
else:
    raise ValueError(
        "No stations were found nearby. Please verify the coordinates."
    )
```

[7]:
```
# To check the length of data to troubleshoot/test while running, the following␣
↪can be executed. (Optional)
len(station_data)
```

[7]: 1049581

Transforming the source data and projected data into the same Spatial Reference System (SRS)

[8]:
```
# List of stations located inside the buffer zone
# Accessing the hydrometric stations layer
driver = ogr.GetDriverByName("GeoJSON")
data_source = driver.Open(station_data, 0)
layer = data_source.GetLayer()
```

[9]:
```
# Identification of the input spatial reference system (SRS)
SRS_input = layer.GetSpatialRef()
SR = osr.SpatialReference(str(SRS_input))
epsg = SR.GetAuthorityCode(None)
SRS_input.ImportFromEPSG(int(epsg))

# Definition of the SRS used to project data
SRS_projected = osr.SpatialReference()
SRS_projected.ImportFromEPSG(projection)

# Transforms the input SRS into projected SRS using the osr library
transform = osr.CoordinateTransformation(SRS_input, SRS_projected)
```

Creating a point and point buffer using the address coordinates. This will be useful to find the nearby stations using intersection.

[10]:
```
# Creation of a buffer to select stations
point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(lon, lat)
point.Transform(transform)

# The value must be in meters
point_buffer = point.Buffer(buffer * 1000)
```

[11]:
```
# Empty list
stations = []

# Selection of the stations in the buffer zone using intersects
for feature in layer:
    geom = feature.GetGeometryRef().Clone()
    geom.Transform(transform)
# Appending it to list using the feature name from 'Queryables'
    if geom.Intersects(point_buffer):
        stations.append(feature.STATION_NAME)
```

To remove the duplicates, converting it to set and then back to list.

[12]:

```python
st = list(set(stations))
print(st)
```

```
['OTTAWA CDA RCS', 'OTTAWA CDA']
```

Using the st variable, both stations will be used to get the features using the OGC API and write it to a dict and then create a pandas dataframe to plot it.

[13]:

```python
# Data retrieval and creation of the data frames
for station in st:
  # Retrieval of water level data
  data = oafeat.collection_items(
        "climate-daily",
        bbox=bbox,
        STATION_NAME=station,
        LOCAL_YEAR=2011,
    )
  if data["features"]:
        # Creation of a dictionary in a format compatible with Pandas
      hist_data = [
            {
                "LATITUDE": el["geometry"]["coordinates"][1],
                "LONGITUDE": el["geometry"]["coordinates"][0],
                **el["properties"],
            }
  for el in data["features"]
        ]
```

[14]:

```python
# Creation of the data frame with the required columns using the 'queryables'
hdf= pd.DataFrame(
            hist_data,
            columns=[
                "CLIMATE_IDENTIFIER",
                "STATION_NAME",
                "PROVINCE_CODE",
                "LOCAL_MONTH",
                "LOCAL_DAY",
                "LOCAL_DATE",
                "LOCAL_YEAR",
                "MIN_TEMPERATURE",
                "MAX_TEMPERATURE",
                "MEAN_TEMPERATURE",
                "TOTAL_PRECIPITATION",
            ],
        )
```

First, we will convert the month number to the corresponding month name using a lambda function.

[15]:
```python
# Importing calendar module for converting the month number into corresponding
↪month number
import calendar

# create a new column with the month names
hdf['MONTH_NAME'] = hdf['LOCAL_MONTH'].apply(lambda x: calendar.month_name[x])
```

[16]:
```python
# Returns the dataframe with the data and columns.
hdf
```

[16]:
```
    CLIMATE_IDENTIFIER    STATION_NAME PROVINCE_CODE  LOCAL_MONTH  LOCAL_DAY  \
0              6105976      OTTAWA CDA            ON            5         31
1              6105976      OTTAWA CDA            ON            9         29
2              6105976      OTTAWA CDA            ON           10         21
3              6105976      OTTAWA CDA            ON            9          7
4              6105976      OTTAWA CDA            ON            9         17
..                 …             …           …           …          …
495            6105978  OTTAWA CDA RCS            ON            5         11
496            6105978  OTTAWA CDA RCS            ON            5         12
497            6105978  OTTAWA CDA RCS            ON            5         13
498            6105978  OTTAWA CDA RCS            ON            5         14
499            6105978  OTTAWA CDA RCS            ON            5         15

              LOCAL_DATE  LOCAL_YEAR  MIN_TEMPERATURE  MAX_TEMPERATURE  \
0    2011-05-31 00:00:00        2011             15.9             30.5
1    2011-09-29 00:00:00        2011             17.4             25.2
2    2011-10-21 00:00:00        2011              7.4             12.2
3    2011-09-07 00:00:00        2011             12.7             18.4
4    2011-09-17 00:00:00        2011              1.2             18.2
..                   …         …              …              …
495  2011-05-11 00:00:00        2011              7.5             19.4
496  2011-05-12 00:00:00        2011              9.0             21.4
497  2011-05-13 00:00:00        2011             10.4             25.2
498  2011-05-14 00:00:00        2011             10.6             15.6
499  2011-05-15 00:00:00        2011              8.1             13.6

     MEAN_TEMPERATURE  TOTAL_PRECIPITATION MONTH_NAME
0                23.2                  0.0        May
1                21.3                  2.7  September
2                 9.8                  3.2    October
3                15.6                  0.0  September
4                 9.7                  0.0  September
..                …                  …        …
495              13.5                  0.0        May
496              15.2                  0.0        May
497              17.8                  5.7        May
498              13.1                 23.0        May
```

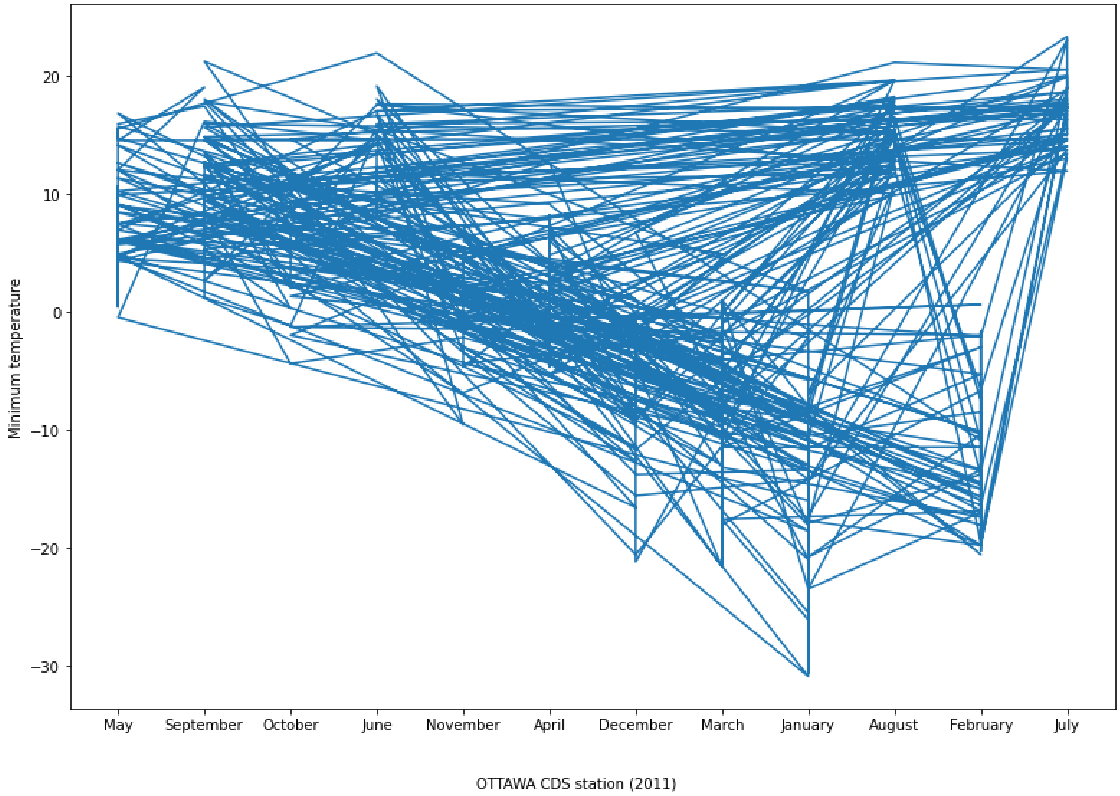
```
499               10.9                        2.4        May

[500 rows x 12 columns]
```

1. Create individual plots for minimum, mean and maximum temperature, and a 4th plot of all temperatures in one plot using the subplot function and aligning the axes.

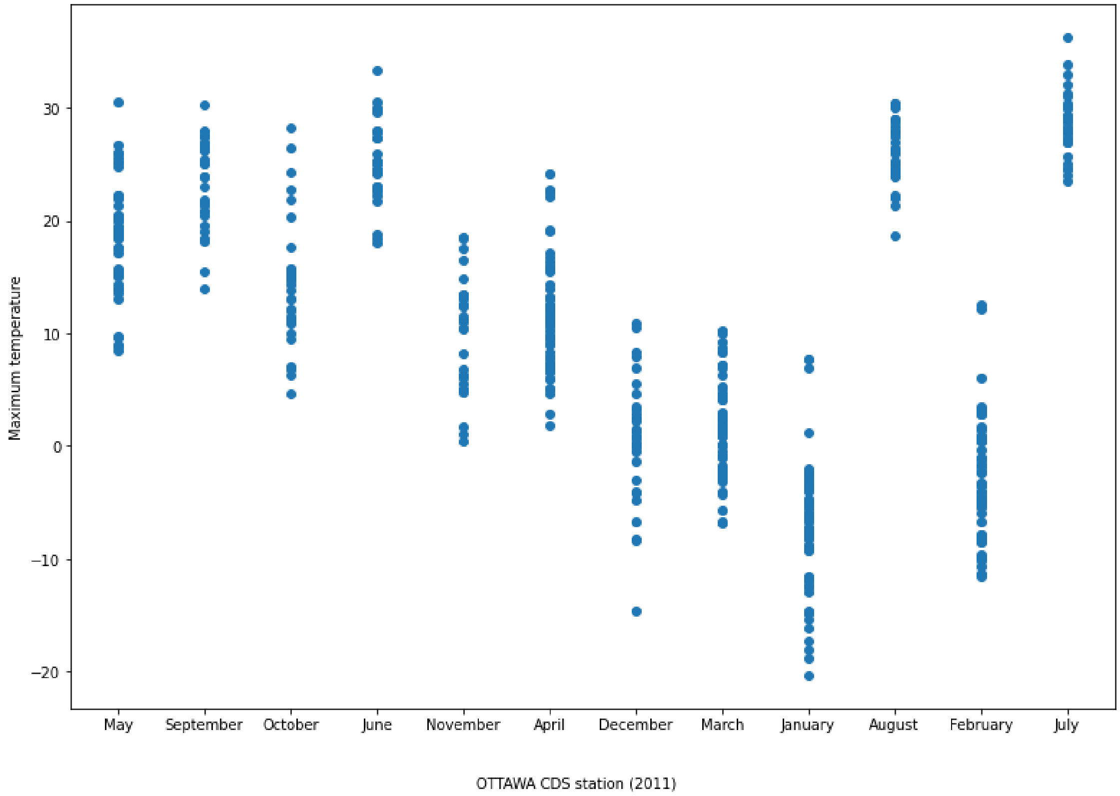
[17]:
```python
# Importing modules
import matplotlib.pyplot as plt
import seaborn as sns
```

Plotting the minimum temperature using line plot, maximum temperature using scatter plot and mean temperature using line plot. X-axis is constant with Y-axis being variable.

[18]:
```python
# Minimum Temperature vs Month Name individual plot
# Adjusting the size of plot
fig, ax = plt.subplots(figsize=(13, 9))

# Create line plot
ax.plot(hdf["MONTH_NAME"], hdf["MIN_TEMPERATURE"])

# Set axis labels
ax.set_ylabel("Minimum temperature")
fig.text(0.5, 0.04, "OTTAWA CDS station (2011)", ha="center")

# Show the plot
plt.show()
```

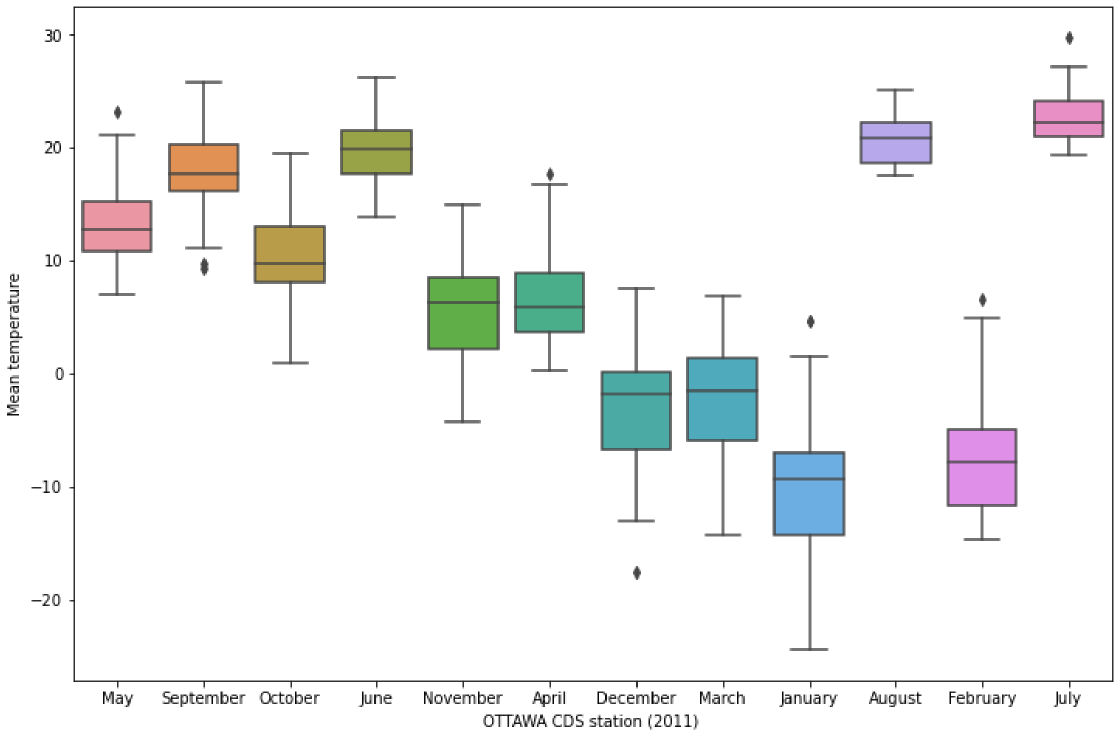


[19]:
```
# Maximum Temperature vs Month Name individual plot
# Adjusting the size of plot
fig_max, ax_max = plt.subplots(figsize=(13, 9))

# Create line plot
ax_max.scatter(hdf["MONTH_NAME"], hdf["MAX_TEMPERATURE"])

# Set axis labels
ax_max.set_ylabel("Maximum temperature")
fig_max.text(0.5, 0.04, "OTTAWA CDS station (2011)", ha="center")

# Show the plot
plt.show()
```



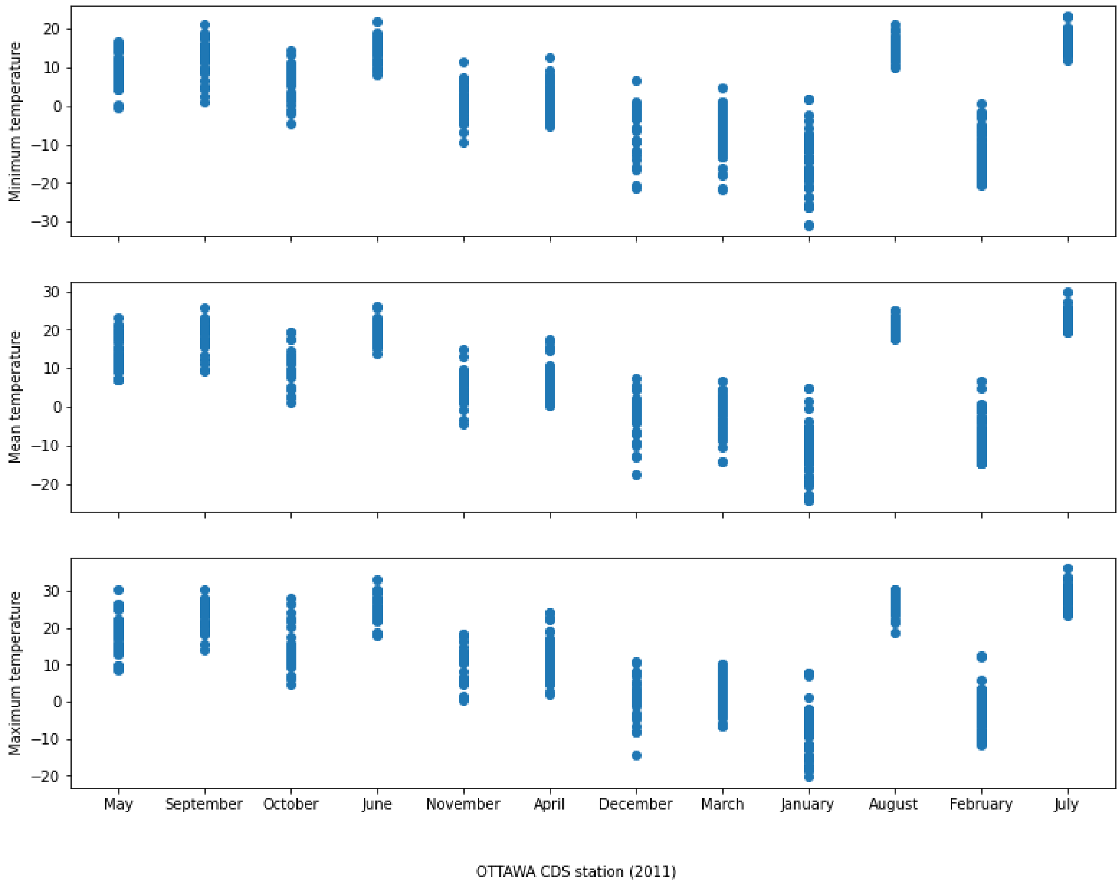
[20]:

```
# Mean Temperature vs Month Name individual plot
# Adjusting the size of plot
plt.figure(figsize=(12, 8))

# Create box plot using Seaborn
sns.boxplot(x="MONTH_NAME", y="MEAN_TEMPERATURE", data=hdf)

# Set axis labels
plt.xlabel("OTTAWA CDS station (2011) ")
plt.ylabel("Mean temperature")

# Show the plot
plt.show()
```



Plotting all temperature plots using subplot function as scatter plots and aligning the axes

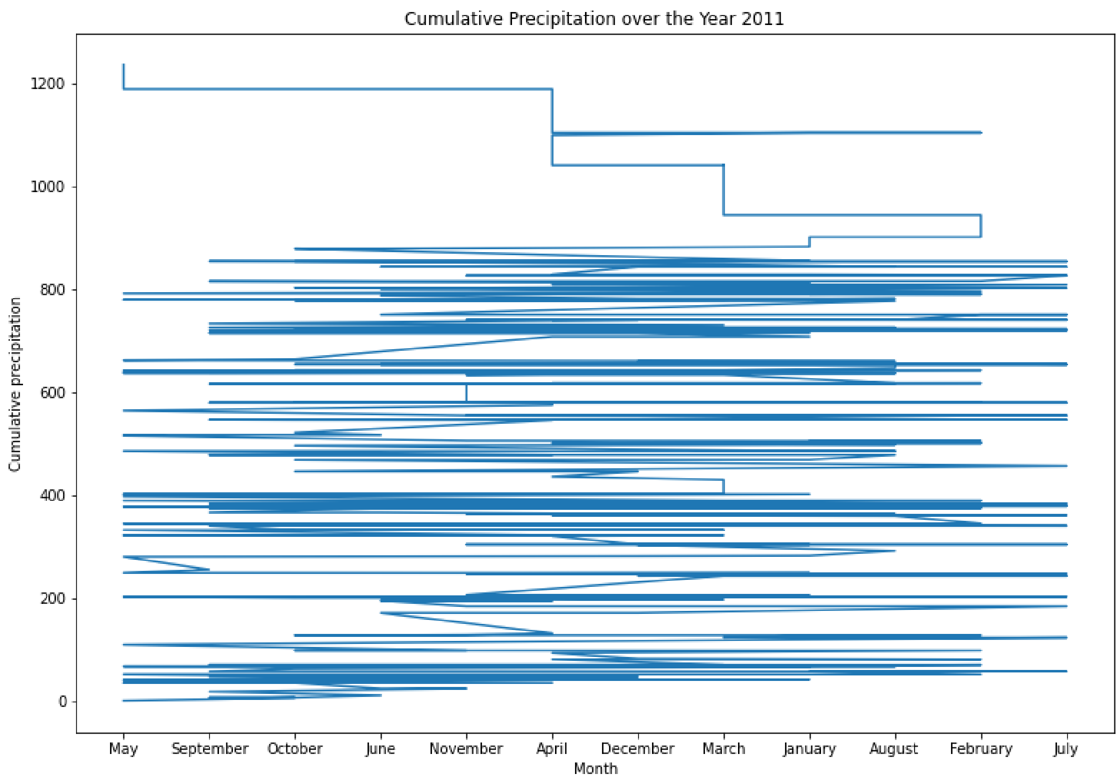
[21]:
```
# Adjusting the size of plot and Creating subplots
fig_merge, axs_merge = plt.subplots(nrows=3, figsize=(13, 10), sharex=True)

# Plot minimum temperature
axs_merge[0].scatter(hdf["MONTH_NAME"], hdf["MIN_TEMPERATURE"])
axs_merge[0].set_ylabel("Minimum temperature")

# Plot mean temperature
axs_merge[1].scatter(hdf["MONTH_NAME"], hdf["MEAN_TEMPERATURE"])
axs_merge[1].set_ylabel("Mean temperature")

# Plot maximum temperature
axs_merge[2].scatter(hdf["MONTH_NAME"], hdf["MAX_TEMPERATURE"])
axs_merge[2].set_ylabel("Maximum temperature")

# Set common x-axis label
fig_merge.text(0.5, 0.04, "OTTAWA CDS station (2011)", ha="center")

# Show the plot
plt.show()
```



2. Plot the cumulative precipitation over the year.

Using line plot to plot the Cumulative Precipitation

[22]:
```
# Calculate cumulative sum of precipitation
cumulative_precipitation = hdf['TOTAL_PRECIPITATION'].cumsum()
fig_prec, ax_prec = plt.subplots(figsize=(13, 9))

# Create line plot
ax_prec.plot(hdf['MONTH_NAME'], cumulative_precipitation)

# Set axis labels and title
plt.xlabel('Month')
plt.ylabel('Cumulative precipitation')
plt.title('Cumulative Precipitation over the Year 2011')

# Show the plot
plt.show()
```



3. Create a geographic map – plot the location of the closest observation station in a folium map using a custom marker with a pop up that provide some details of the station (e.g.: Station Name, Station Number, and Latitude, Longitude) and a different style marker at the address provided.

[23]:
```
# Importing packages
import folium
```

[24]:
```
# Creating a list to extract all the coordinates and add it to the list
temp = []
for i in data['features']:
  temp.append(i['geometry']['coordinates'])
```

[25]:
```
# This will remove the duplicates and convert it to list.
location = list(set(tuple(x) for x in temp))
```

Getting the station location coordinates and then creating folium map with markers

[26]:
```
# Getting the longitude and latitude from the list
station_location = [location[0][1],location[0][0]]

# Define the latitude and longitude of the closest observation station and the␣
↪address
```

```python
address_location = [lat, lon]
```

[27]:
```python
address_location
```

[27]: [45.39864945, -75.70611299803701]

Optional way to use the string which use dynamic variables.

popup_str = f'Station Name: {station}
Station Number: 6105976
Latitude: {station_locat

[28]:
```python
# Define a custom icon for the station marker
station_icon = folium.Icon(color='red', icon='info-sign')

# Create a folium map centered on the station location
fol_map = folium.Map(location=station_location, zoom_start=12)
```

Creating Folium markers for maps : One with custom icon.

[29]:
```python
# Add a marker for the station location with a pop-up showing its details
folium.Marker(
    location=station_location,
    popup='Station Name: OTTAWA  <br>Station Number:  6105976<br>Latitude: 45.
↪38333333333333<br>Longitude: -75.71666666666667',
    icon=station_icon
).add_to(fol_map)

# Add a marker for the address location with a default icon
folium.Marker(
    location=address_location,
    popup='Address: 560 Rochester St, Ottawa, ON, CA<br>Latitude: 45.
↪39864945<br>Longitude: -75.70611299803701'
).add_to(fol_map)
```

[29]: <folium.map.Marker at 0x7ff0400600a0>

[30]:
```python
# Display the map
fol_map
```

[30]: <folium.folium.Map at 0x7ff040efb100>

1.2 Part B. Interactive Visualization

1.2.1 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

[31]:
```
# Importing required packages
from ipyleaflet import Map, basemaps, basemap_to_tiles, WMSLayer,␣
↪SplitMapControl
```

Firstly, two WMS services are defined with different attributes and parameters. * Boundless Geo WMS * Ahocevar WMS

[32]:
```
# Define the two WMS services with params
wms_service_1 = WMSLayer(
    url='https://demo.boundlessgeo.com/geoserver/wms',
    layers='ne:NE1_HR_LC_SR_W_DR',
    format='image/png',
    transparent=True,
    attribution='Natural Earth'
)

wms_service_2 = WMSLayer(
    url='https://ahocevar.com/geoserver/wms',
    layers='topp:states',
    format='image/png',
    transparent=True,
    attribution='GeoServer'
)
```

Now, we will create the map and layers and adding them to the map.

[33]:
```
# Create the map and add the WMS services as layers
interactive_map = Map(center=(56, -106), zoom=4)
```

[34]:
```
# Creating two basemap layers using the basemaps module
wms_layer_1 = basemap_to_tiles(basemaps.OpenStreetMap.Mapnik)
wms_layer_2 = basemap_to_tiles(basemaps.OpenTopoMap)

# Adding these basemap layers to an interactive map object
interactive_map.add_layer(wms_service_1)
interactive_map.add_layer(wms_service_2)
```

[35]:
```
# Add the SplitMapControl to view the two layers side-by-side
control = SplitMapControl(left_layer=wms_layer_1, right_layer=wms_layer_2)
interactive_map.add_control(control)
```

[36]:
```
# Display the map
interactive_map
```

```
Map(center=[56, -106], controls=(ZoomControl(options=['position',
↪'zoom_in_text', 'zoom_in_title', 'zoom_out_t…
```

1.3 References

1. Matplotlib Documentation
2. Seaborn Documentation
3. MSC GeoMet
4. MSC GeoMET daily-climate
5. Use case: Retrieving and displaying hydrometric data
6. ipyleaflet documentation
7. OSGEO documentation
8. OWSLib documentation
9. Pandas documentation
10. Folium documentation