Wild Fire Map by Sebastian Gmür

Created for Geo876, 17.03.2024

Description

This project makes use of the FIRMS project (Fire Information for Resource Management) of Nasa. They provide data of 5 Sensors with different API calls. The goal of this project was, to gather all usefull sensors and visualize them. To achieve this the following layers should be part of the map:

- An overall Heatmap, which takes all available sources into account.
- A Cluster Marker map, which combines all 3 satellites which have a VIIRS Sensor. When zooming in, the clusters split up into their original point markers.
  - Important to know: the single points are colored by their satellite. This infor couldn't be added to the map because folium doesn't support categorical colormaps.
- The MODIS data as single layer, especially to highlight the differences between VIIRS and MODIS

About the code:

So, this coding project is probably way to overengineered. It basicaly consists of 3 classes (and a subclass):

- **API Class**: An object to store an api token, URL, the gathered Data and a Timestamp, when data was last accessed.
  - a private API token
  - API URL
  - a Method to get Data from URL & the raw data itself
  - a timestamp, when API was last accessed
- **A "Wildfire Layer" Class**: To store the API class, together with folium.layer functions
  - Data Methods:
    - a "gather_data" method, which checks the API object's timestamp and either calls API again, or use allready stored API data. (If timestamp is older than 2 hours, call again)
    - a clean data method, which converts raw data into geodataframes
  - Layer Methods:
    - A method functioning like a switch, and depending on a keyword calls another create_layer method.

        - implemented layer methods are: HeatMap, CircleMarker ("Points") and MarkerCLuster
    - Some helper funcitons, mostly because color manipulation with folium is not very nice.
- **A "Overview Layer" Class**: A special case of Wildfire Layer Class, where you can add multiple API-objects, which will be rowbinded together
- **A "Wildfire Map" Class**:
    - creates a standardized map.
    - can be given an unspecified number of Wildfire Layer Objects (or folium layer objects), which are added to the map.

Problems encountered:

- On my way to the final map, I encountered many different problems. I started with ipyleaflet instead of folium for the leaflet library. However, with ipyleaflet, there was no option to **not** show a layer, when loading a map (This is what the parameter "show" does in folium). So when I created the map with ipyleaflet, all available layers where shown on the map by default.
    - -> After some research and trying out different things, I decided to rewrite my code with folium, where this is possible
- After switching to folium, I tried to use the folium internal GeoJson format, which can easily handle geopanda geodataframes as input and create folium layers, such as CircleMarkers. However, I wanted to use MarkerClusters, which is a separate folium plugin. While MarkerCluster supports geojson in theory, so it should have worked. However, when creating the popups, I realized that only the gejson-markers are supported and not the geojson-popup class.
    - -> I deleted the create_gejson function and instead created the create_circle_marker Function, and included the "popup_fields" parameter.
- After implementing the create_circle_marker funciton, I wanted to dynamically change the coloring, depending on a column in the original data. This was more complicated then thought and a lot of the whole code is just to make this work.
    - Categorical or continuous data columns can be used to color. If categorical data is used however, no legend can be created, only colorbars.
    - colorbars are not natively connected to their layer, but to the map itself. When adding a lot of layers with colorbars, all e.g. 4 colorbars are shown all the time. This leads to confusion, so I researched and found the BindColormap function, which I copied.

Imports

In [ ]:
```
import requests
import pandas as pd
import geopandas as gpd
from datetime import datetime, timedelta
from shapely.geometry import Point
import folium
from folium.plugins import MarkerCluster, HeatMap
import numpy as np
import branca.colormap as cm
#import math
#import matplotlib.colors as mcolors
```

In [ ]:
```
## This first block is only necessary, such that the colorbar in the map is

from branca.element import MacroElement

from jinja2 import Template

class BindColormap(MacroElement):
    """Binds a colormap to a given layer.
    This code was directely copied from:
    https://nbviewer.org/gist/BibMartin/f153aa957ddc5fadc64929abdee9ff2e
    and is mentioned in Github Forum: https://github.com/python-visualizatio

    """
    def __init__(self, layer, colormap):
        super(BindColormap, self).__init__()
        self.layer = layer
        self.colormap = colormap
        self._template = Template(u"""
        {% macro script(this, kwargs) %}
            {{this.colormap.get_name()}}.svg[0][0].style.display = 'block';
            {{this._parent.get_name()}}.on('overlayadd', function (eventLaye
                if (eventLayer.layer == {{this.layer.get_name()}}) {
                    {{this.colormap.get_name()}}.svg[0][0].style.display = '
                }});
            {{this._parent.get_name()}}.on('overlayremove', function (eventL
                if (eventLayer.layer == {{this.layer.get_name()}}) {
                    {{this.colormap.get_name()}}.svg[0][0].style.display = '
                }});
        {% endmacro %}
        """)  # noqa
```

Class & Methods Definition:

In [ ]:
```
## I chose to create an API object, to include the different sensor systems
class Firms_source_object:
    ''' FIRMS API Objects
    FIRMS (Fire Information for Resource Management System) is NASAs open da
    They provide several sensors with various days.
    Source: https://firms.modaps.eosdis.nasa.gov/
    '''
    def __init__(
        self, api_token: str, sensor_source: str,
```

```
        day_range = 1, base_url = "https://firms.modaps.eosdis.nasa.gov/api/
        valid_for_x_minutes = 120
    ):
        self.base_url = base_url
        self.__api_token = api_token
        self.sensor_source = sensor_source
        self.day_range = day_range
        self.valid_for_x_minutes = valid_for_x_minutes
        self.__api = f'{self.base_url}/{self.__api_token}/{self.sensor_sourc


    def get_data(self):
        '''The API returns a csv file, which can directely be opened with pa
        self.data = pd.read_csv(self.__api) ## Makes API call
        self.api_timestamp = datetime.now() ## Time when API call was made
        return self.data


    def is_uptodate(self):
        '''Checks if API has been called yet and returns True, if call is no
        if hasattr(self, "api_timestamp"):
            time_delta = datetime.now() - self.api_timestamp
            if time_delta < timedelta(minutes = self.valid_for_x_minutes):
                return True
        return False


class Wildfire_layer:
    """
    A Layer-Class, specified for the Firms Wildfire API
    This class provides many different methods to use while creating folium
    The idea of this class was, to hardcode / Specify default values for all

    Parameters for Init:
        firms_source: A Firms_source_object
        layertype: A String defining what layertype should be created. ("Mar
    """

    def __init__(self,firms_source: Firms_source_object, layertype, **kwargs
        self.firms_source = firms_source
        self.kwargs = kwargs
        self.data = self.retrieve_data(self.firms_source)
        self.layertype = layertype
        self.layer = self.create_layer(layertype)

    # # Data Prep: ------------------------------------------------------------
    def retrieve_data(self, firms_object): ##This function checks when API w
        if firms_object.is_uptodate():
            if hasattr(self, "data_clean"):
                return self.data_clean
            else: # API Data is Up-to-date, but data_clean is missing -> don
                self.data_clean = self.clean_data(firms_object.data)
                return self.data_clean
        # If API Data is not Up-to-date: api.get_data()
        data_raw = firms_object.get_data()
        self.data_clean = self.clean_data(data_raw)
```

```
        return self.data_clean

    def clean_data(self, data_raw):
        data_clean = data_raw
        data_clean["geometry"] = [Point(xy) for xy in zip(data_clean['longit
        data_clean = data_clean.drop(["latitude","longitude"],axis = 1)
        self.data_clean = gpd.GeoDataFrame(data_clean, crs="EPSG:4326")
        return self.data_clean

    ## Create folium Layers: ----------------------------------------------
    def create_layer(self, layer_type):
        if layer_type == "MarkerCluster":
            group = self.create_circlemarker(self.data)
            layer = self.create_marker_cluster(group)
        if layer_type == "Marker":
            layer = self.create_circlemarker(self.data)
        if layer_type == "HeatMap":
            layer = self.create_heatmap(self.data)
        return layer

    def create_heatmap(self, data):
        radius = self.kwargs.get("radius", 15)
        name = self.kwargs.get("name", self.firms_source.sensor_source + " H
        points = [[point.xy[1][0], point.xy[0][0]] for point in data["geomet
        return HeatMap(points, radius=radius, name = name)


    def create_circlemarker(self, data):
        ##Kwargs:
        name = self.kwargs.get("name",self.firms_source.sensor_source) ## Ei
        stroke = self.kwargs.get("stroke", False)
        show = self.kwargs.get("show", True)
        popup_fields = self.kwargs.get("popup_fields", ["scan", "track", "ac
        #Color parameter:
        color_col_name = self.kwargs.get("color_col_name", "bright_ti4")
        colormap = self.kwargs.get("colormap", cm.linear.Spectral_07)
        color_caption = self.kwargs.get("color_caption", color_col_name)
        data["color"] = self.add_color_column(data, color_col_name, caption

        # Iteratively create Markers for every row in dataframe
        self.group = folium.FeatureGroup(name, show = show)
        for idx, row in data.iterrows():
            lat, lon = row.geometry.y, row.geometry.x
            circle_marker = folium.CircleMarker(location=[lat, lon],
                                                fill=True, radius = 4, strok
                                                fill_color=row.color, fill_c

            popup_content = ""
            for col_name in popup_fields:
                popup_content += f"<b>{col_name.capitalize()}:</b> {row[col_

            circle_marker.add_child(folium.Popup(popup_content))
            circle_marker.add_to(self.group)
        return self.group
```

```
    def create_marker_cluster(self, markers_layer_group):
        ##Options:
        name = self.kwargs.get("name",
                               self.firms_source.sensor_source + " Cluster"
        show = self.kwargs.get("show", True)
        disableClusteringAtZoom = self.kwargs.get("disableClusteringAtZoom",

        marker_cluster = MarkerCluster(name = name, show = show, options= {"
        markers_group = markers_layer_group
        for marker in list(markers_group._children.values()):
            marker.add_to(marker_cluster)
        return marker_cluster
        return marker_cluster


    ## Helper Functions:   ------------------------------------------------


    def add_color_column(self, df, column_name, caption, colormap= cm.linear

        if df[column_name].dtype == "object": ## The indented part was more
            unique_values = df[column_name].unique()
            color_dict = dict(zip(unique_values, colormap.colors[:len(unique
            color = df[column_name].map(color_dict)
            color = self.convert_column_to_hex(color)
            self.colorbar = colormap
            self.colorbar.caption = caption
        else:
            if reverse:
                colormap = self.reversed_colormap(colormap)
            self.colorbar = colormap.scale(df[column_name].min(), df[column_
            self.colorbar.caption = caption
            color = df[column_name].apply(self.colorbar)
        return color

    def rgb_to_hex(self, rgb): ## Chat-GPT, converts rgb as tuples to string
        r = int(rgb[0] * 255)
        g = int(rgb[1] * 255)
        b = int(rgb[2] * 255)
        return "#{:02x}{:02x}{:02x}".format(r, g, b)

    def convert_column_to_hex(self,column): ## Chat-GPT
        hex_column = [self.rgb_to_hex(rgb) for rgb in column]
        return hex_column


    def reversed_colormap(self, existing): ## Source: https://stackoverflow.
        return cm.LinearColormap(
            colors=list(reversed(existing.colors)),
            vmin=existing.vmin, vmax=existing.vmax
            )


## Subtype of Wildfire Layer: Overview_Layer.
## Basically can just handle multiple Firms_source_objects and row_binds the
class Overview_Layer(Wildfire_layer):
```

```
    def __init__(self, shared_column_names,layertype, *args, **kwargs):
        self.shared_column_names = shared_column_names
        self.layertype = layertype
        self.args = args
        self.kwargs = kwargs
        self.firms_source = Firms_source_object("default_Source", sensor_sou
        merged_data = []
        for arg in args:
            data = self.retrieve_data(arg)
            data["Sensor_Source"] = arg.sensor_source
            data = data[shared_column_names + ["Sensor_Source"]]
            merged_data.append(data)
        self.data = pd.concat(merged_data, ignore_index=True)
        self.layer = self.create_layer(layertype)


class Wildfire_map:
    def __init__(self, *args):
        self.m = folium.Map(location=(46.78513, 9.07178), zoom_start = 6,til
        for arg in args:
            self.add_to_map(arg)
        folium.LayerControl().add_to(self.m)

    def add_to_map(self,l):
        if hasattr(l, "add_to"):
            l.add_to(self.m)
        elif hasattr(l, "layer"):
            l.layer.add_to(self.m)

    def __call__(self):
        return self.m
```

Creating API Objects:

In [ ]:
```
token = "36ece617d2f514d4e90f01418d1b9e28" ##This is my private token
## Create various FIRMS API Objects, with different sensors:
snpp = Firms_source_object(api_token = token, sensor_source= "VIIRS_SNPP_NRT
modis = Firms_source_object(token, "MODIS_NRT")
noaa20 = Firms_source_object(token, "VIIRS_NOAA20_NRT")
noaa21 = Firms_source_object(token, "VIIRS_NOAA21_NRT")
```

Creating Map Layers

In [ ]:
```
## Create Heatmap Layer:

columns = ["acq_date","acq_time","instrument", "satellite","confidence","day
overview = Overview_Layer(
    columns, "HeatMap",                    # shared_column_names, and layert
    modis, snpp, noaa20, noaa21)          ## The different API objects (as *

## Create Marker_Cluster specificly for all VIIRS Sensor Layers:
columns = ["bright_ti4","scan","track","acq_date","acq_time","satellite","in
viirs_cluster = Overview_Layer(
```

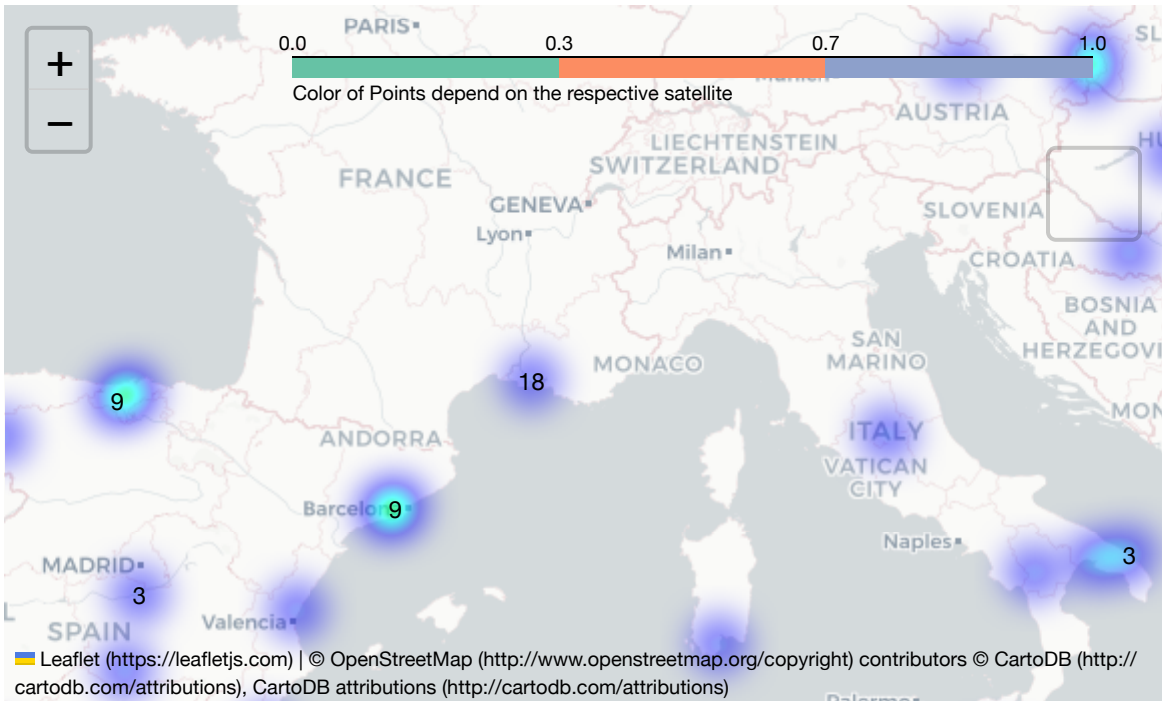
```
    columns, "MarkerCluster",                    # shared_column_names, and
    snpp, noaa20, noaa21,           ## The different API's (as *args)
    name = "VIIRS Sensors Fire Cluster", show = False, disableClusteringAtZo
    color_caption = "Color of Points depend on the respective satellite",
    color_col_name = "Sensor_Source",
    colormap = cm.linear.Set2_03.to_step(3), ## discrete colormap with only
    popup_fields = columns + ["Sensor_Source"])

## MODIS as separate Points:
modis_points = Wildfire_layer(
    modis, "Marker",
    color_col_name = "brightness",
    color_caption = "Channel 21/22 brightness temperature of the fire pixel
    show = False,
    name = "MODIS Fire Detection"
)
```

Creating main map:

In [ ]:
```
map = Wildfire_map(
    viirs_cluster,
    viirs_cluster.colorbar,
    BindColormap(viirs_cluster.layer, viirs_cluster.colorbar),
    overview,
    modis_points,
    modis_points.colorbar,
    BindColormap(modis_points.layer, modis_points.colorbar)
)
map()
```

Out[ ]:



In [ ]:
```
map.m.save("Wildfire_Map.html")
```

Second map:

I only create a second map here to show, how flexible the code is. The main map had a lot of individualisation, but a basic map can be created instantly:

In [ ]:
```
map2 = Wildfire_map(
    Wildfire_layer(modis, "Marker", color_col_name = "brightness"),
    Wildfire_layer(noaa20, "Marker"),
    Wildfire_layer(noaa20, "Marker").colorbar
)
map2()
```

Out[ ]: