

→ EARTH OBSERVATION FOR SUSTAINABLE DEVELOPMENT

Climate Resilience

**Webinar Series on how to use Earth
Observation to tackle Climate Change**

*Webinar 06: How-to' Session: Using the EO4SD
CR Platform to access EO data (hands-on)*

4) Examples of pre-existing Notebooks

Dora Perrou, Research Associate, NOA



Examples of pre-existing Notebooks



jupyter Logout Control Panel

Files Running Clusters

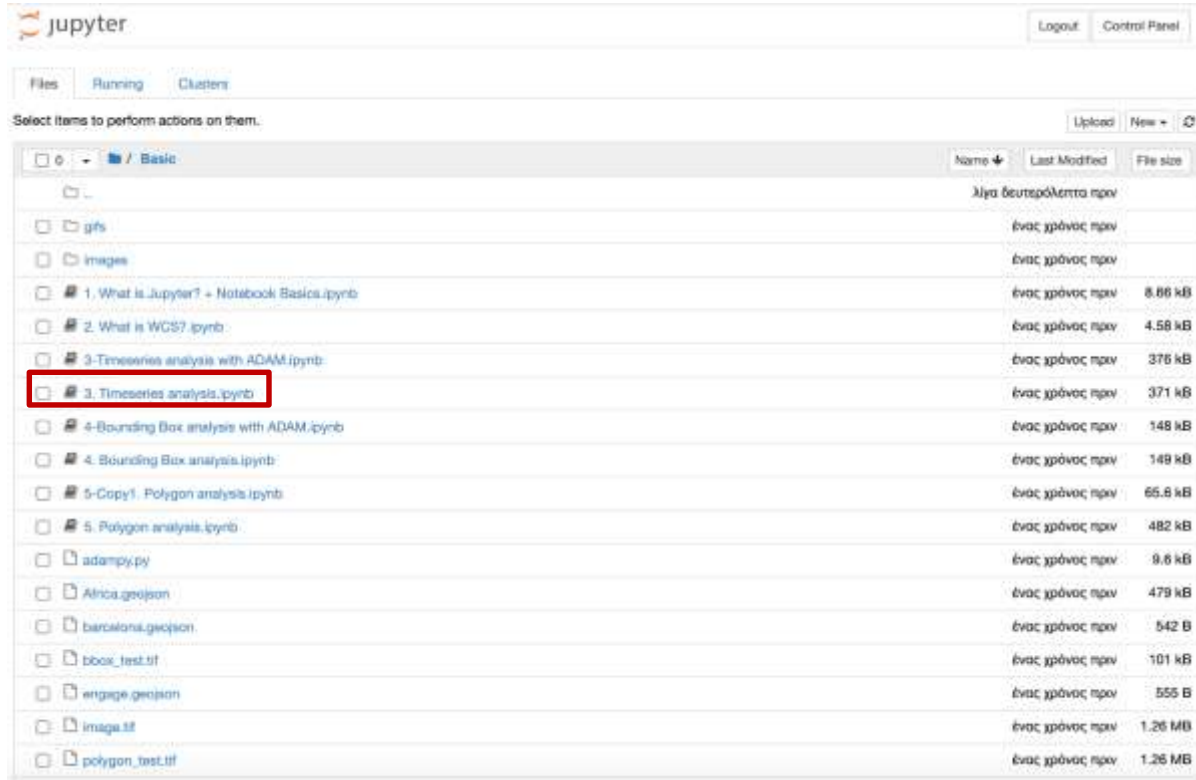
Select items to perform actions on them. Upload New ↻

0 Advanced Name Last Modified File size

	Name	Last Modified	File size
<input type="checkbox"/>	..	λίγα δευτερόλεπτα πριν	
<input type="checkbox"/>	1. Timeseries analysis and data visualization.ipynb	ένας χρόνος πριν	230 kB
<input type="checkbox"/>	2. Bounding Box analysis.ipynb	ένας χρόνος πριν	149 kB
<input type="checkbox"/>	3. Polygon analysis.ipynb	ένας χρόνος πριν	481 kB
<input type="checkbox"/>	4. Rainfall event.ipynb	ένας χρόνος πριν	158 kB
<input type="checkbox"/>	5-Copy1.1 Sentinel 2 data access and visualization.ipynb	6 μήνες πριν	1.03 MB
<input type="checkbox"/>	5.1 Sentinel 2 data access and visualization.ipynb	8 μήνες πριν	2.04 MB
<input type="checkbox"/>	Africa.geojson	ένας χρόνος πριν	479 kB
<input type="checkbox"/>	bbox_test.tif	ένας χρόνος πριν	101 kB
<input type="checkbox"/>	Level-2A_Algorithms_Equation_2.jpg	ένας χρόνος πριν	14.9 kB
<input type="checkbox"/>	Level-2A_Algorithms_Equation_2a.jpg	ένας χρόνος πριν	18.1 kB
<input type="checkbox"/>	meeo_sso_cli.py	ένας χρόνος πριν	1.6 kB
<input type="checkbox"/>	ndvi.tif	8 μήνες πριν	482 MB
<input type="checkbox"/>	polygon_test.tif	ένας χρόνος πριν	1.26 MB
<input type="checkbox"/>	RGB.tif	ένας χρόνος πριν	181 MB
<input type="checkbox"/>	RGB_2.tif	ένας χρόνος πριν	723 MB
<input type="checkbox"/>	RGB_3.tif	6 μήνες πριν	181 MB
<input type="checkbox"/>	S2A_MS1L1C_B02.tif	6 μήνες πριν	187 MB



1. Time series analysis of temperature



The screenshot shows the JupyterLab interface with a file browser view. The current directory is 'Basic'. The file list includes:

Name	Last Modified	File size
..
gfs
images
1. What is Jupyter? + Notebook Basics.ipynb	...	8.86 kB
2. What is WGS? .ipynb	...	4.58 kB
3. Timeseries analysis with ADAM.ipynb	...	376 kB
3. Timeseries analysis.ipynb	...	371 kB
4. Bounding Box analysis with ADAM.ipynb	...	148 kB
4. Bounding Box analysis.ipynb	...	148 kB
5-Copy1. Polygon analysis.ipynb	...	66.6 kB
5. Polygon analysis.ipynb	...	482 kB
adampy.py	...	9.6 kB
Africa.geojson	...	479 kB
barcelona.geojson	...	542 B
book_test.tif	...	101 kB
engage.geojson	...	555 B
image.tif	...	1.26 MB
polygon_test.tif	...	1.26 MB

Timeseries analysis of temperature

- For the timeseries example we will get **temperature** data from the **ERA-Interim** dataset over **Nairobi** for the month of **October 2015** and **plot it**
- The ERA-Interim temperature dataset has data every 6 hours starting at 00:00, therefore we have 4 temperature data per day

3. Timeseries Analysis

First we will import some libraries in order to retrieve data from WCS using Python

```
In [1]: import requests
import os
import xml.etree.ElementTree as ET
import numpy as np
from datetime import datetime, timedelta, date
from statistics import mean
```

Timeseries

For the timeseries example we will get temperature data from the ERA-Interim dataset over Nairobi for the month of October 2015 and plot it. The ERA-Interim temperature dataset has data every 6 hours starting at 00:00, therefore we have 4 temperature data per day.

Let's define a function that we will use to retrieve time series data from the WCS service

```
In [2]: def download_data_timeseries(time_t, collection, latitude, longitude):
timeseries = []
data_mean = []
#to create the URL with the request. Note that the parameters include the "format" function at the end of the
#URL will fill the {} in order
url = 'http://wcs-eo4dhr.esmip1atform.eu/wcs/service/WCSRequest?GetCoverage&version=2.0.0&subset=xmls({})&format=application/xml'
#since the url is build, we can send a request to the server
result = requests.get(url)

#In order to work with the xml data we transform it to text
xml_data = result.text

tree = ET.fromstring(xml_data)

#extract the time delta from the xml file in order to build later the time array
for t in tree.iter('http://www.opengis.net/gml/3.2:timeCoverage'):
    # split string into list of coordinates
    time_delta = t.text.split(',')
    time_delta = time_delta[0]

for t in tree.iter('http://www.opengis.net/gml/3.2:tupleList'):
    # split string into list temps
    data = t.text
    #split(data)

#split
data = data.split()
#split again
data = data[0].split(',')

#the data is in Nairobi, and we want them in Celsius
data_mean = np.array(data).astype(float) - 273.15
data_mean = data_mean[:].flatten()

for t in tree.iter('http://www.opengis.net/gml/3.2:rangeOfCoefficients'):
```

Timeseries analysis of temperature

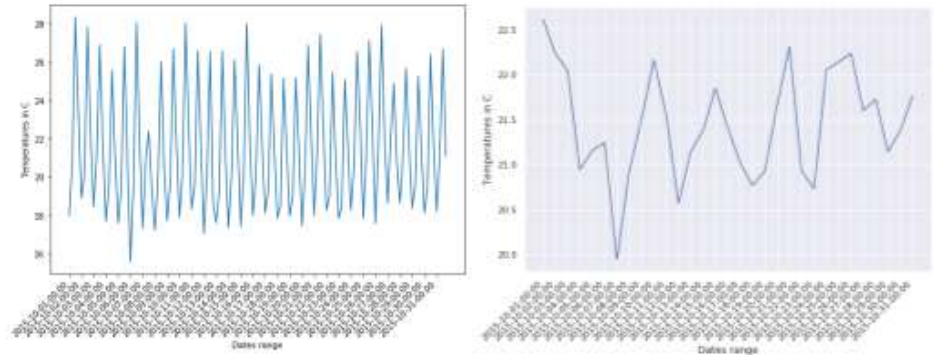
- We define the **time range**, the **latitude** and **longitude** of the point and the **data collection**

```
time_t = '2015-10-01T00:00:00,2015-10-31T23:59:59'  
latitude = -1.286046  
longitude = 36.820959  
collection = 'ERA-Interim_temp2m_4326_05'
```

- We can **print the raw data**

```
data.times  
array([[17.997, 20.718, 28.36 , 28.368, 28.899, 27.099, 27.614, 22.138,  
18.438, 21.288, 26.882, 21.743, 17.991, 16.548, 28.272, 20.92 ,  
17.402, 29.848, 26.801, 20.882, 28.579, 19.817, 28.032, 21.848,  
17.32 , 20.821, 22.374, 19.227, 17.217, 19.854, 26.038, 20.574,  
17.124, 29.884, 26.89 , 21.024, 27.879, 20.099, 28.008, 22.862,  
18.327, 19.837, 26.187, 21.757, 17.039, 20.026, 28.848, 18.852,  
17.402, 28.172, 26.271, 21.228, 17.288, 20.054, 28.207, 21.893,  
17.428, 20.184, 27.997, 21.772, 17.988, 20.124, 28.84 , 21.66 ,  
28.139, 19.238, 29.349, 21.227, 17.843, 18.494, 28.158, 21.565,  
17.883, 19.01 , 28.189, 21.587, 17.883, 20.15 , 26.878, 22.141,  
17.991, 21.218, 27.494, 22.477, 18.288, 18.953, 28.441, 21.028,  
17.848, 18.425, 28.068, 21.884, 18.285, 20.484, 28.858, 22.973,  
17.86 , 21.114, 27.19 , 28.462, 17.074, 21.954, 27.821, 21.086,  
18.634, 22.222, 24.881, 20.778, 18.437, 21.052, 28.687, 21.815,  
18.362, 19.866, 28.248, 21.99 , 19.132, 18.499, 18.881, 21.884,  
18.212, 21.074, 24.472, 21.094]])  
('2015-10-01-00:00',  
'2015-10-31-06:00',  
'2015-10-31-12:00',  
.....)
```

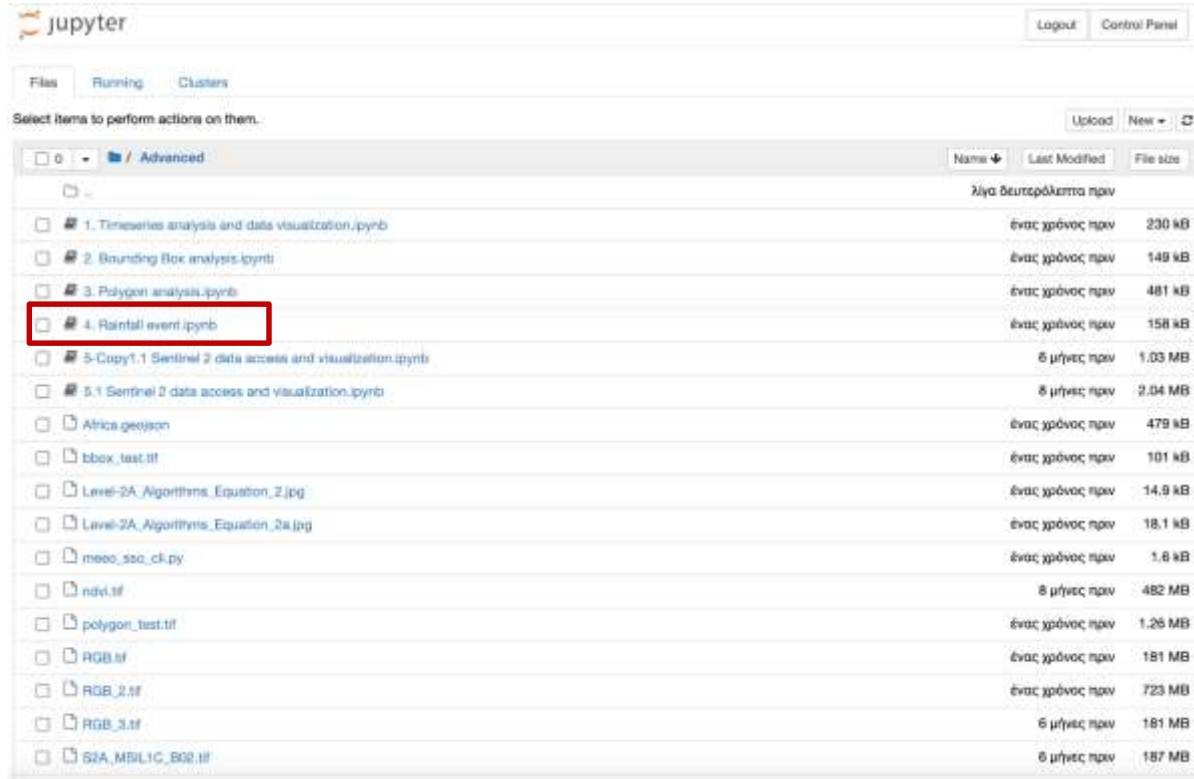
- Or we can **plot the data**



- We can also get the **max**, **min** and **mean** temperature

```
Maximum temperature: 28.360 degrees celsius  
Minimum temperature: 15.579 degrees celsius  
Mean temperature: 21.452 degrees celsius
```


2. Rainfall event detection



The screenshot shows the JupyterLab interface with a file browser view. The current directory is '/ Advanced'. A list of files and notebooks is displayed with columns for Name, Last Modified, and File size. The notebook '4. Rainfall event.ipynb' is highlighted with a red box.

Name	Last Modified	File size
0		
Advanced		
-	για δευτερόλεπτα πριν	
1. Timeseries analysis and data visualization.ipynb	έντας χρόνος πριν	230 kB
2. Bounding Box analysis.ipynb	έντας χρόνος πριν	149 kB
3. Polygon analysis.ipynb	έντας χρόνος πριν	481 kB
4. Rainfall event.ipynb	έντας χρόνος πριν	158 kB
5-Copy1.1 Sentinel 2 data access and visualization.ipynb	6 μήνες πριν	1.03 MB
5.1 Sentinel 2 data access and visualization.ipynb	6 μήνες πριν	2.04 MB
Africa.geojson	έντας χρόνος πριν	479 kB
bbox_test.tif	έντας χρόνος πριν	101 kB
Level-2A_Algorithms_Equation_2.jpg	έντας χρόνος πριν	14.9 kB
Level-2A_Algorithms_Equation_2a.jpg	έντας χρόνος πριν	18.1 kB
meo_sso_cli.py	έντας χρόνος πριν	1.8 kB
ndvi.tif	8 μήνες πριν	482 MB
polygon_test.tif	έντας χρόνος πριν	1.26 MB
RGB.tif	έντας χρόνος πριν	181 MB
RGB_2.tif	έντας χρόνος πριν	723 MB
RGB_3.tif	6 μήνες πριν	181 MB
S2A_MSLIC_B02.tif	6 μήνες πριν	187 MB

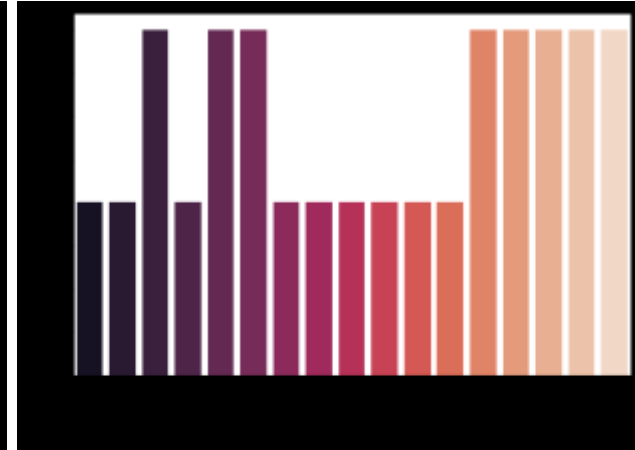
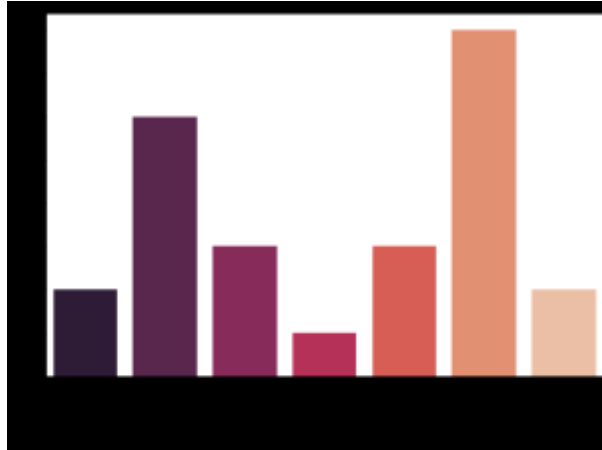
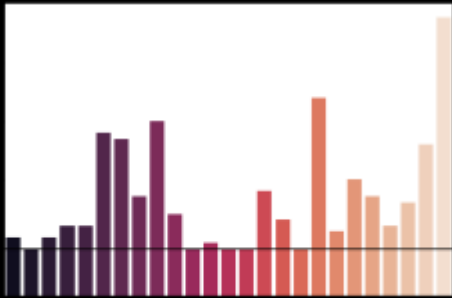
Rainfall event detection

- We will check precipitation data and detect when a rainfall event occurs.
- This is very useful for Agriculture research in order to analyze historical data and detect potential risks in the fields
- We can check rainfall events during 2015 and **aggregate them monthly**
- As an example, we define a rainfall event: if the cumulative rain over **3 days is greater than 80 mm**

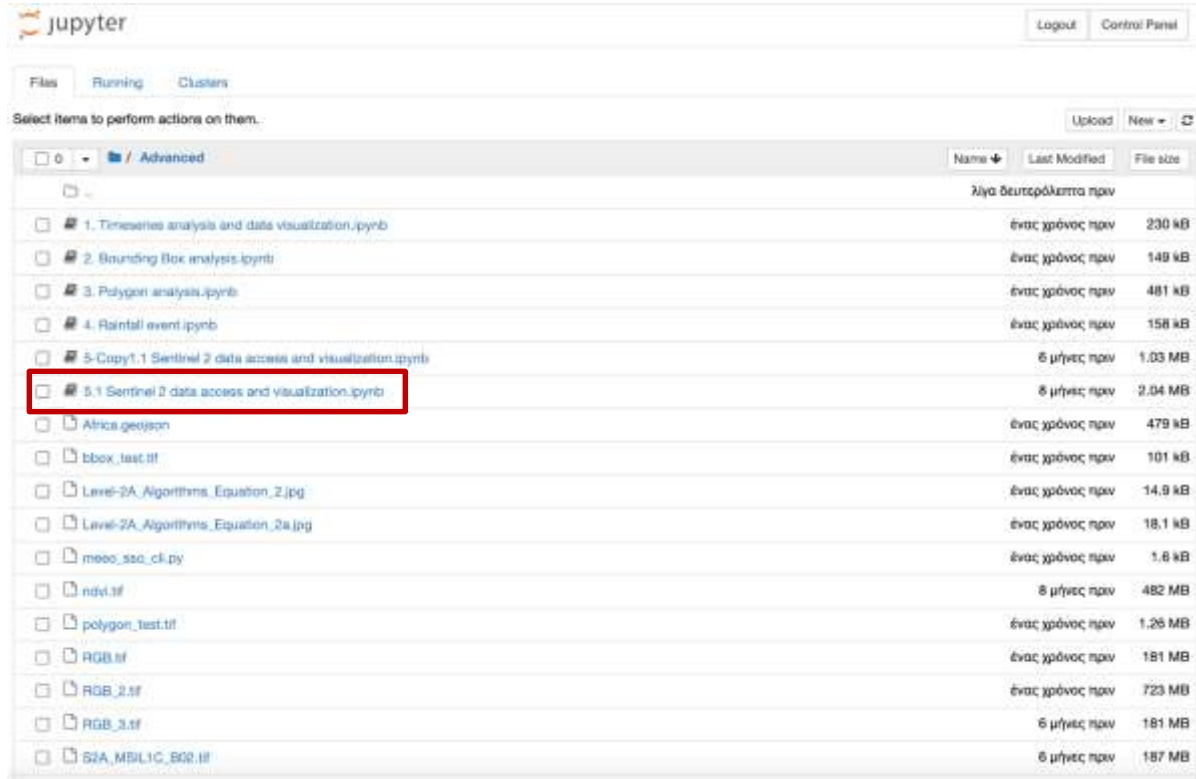
```
time_t = '2015-01-01T00:00:00,2015-12-31T23:59:59'  
agg = 'm'  
min_lat = max_lat = -1.286046  
min_long = max_long = 36.820959  
days = 3  
amount = 80
```

Rainfall event detection

- We can check at the precipitation events and see in a barplot graph the amount of cumulative precipitation during the event
- We can also check at the data by the aggregation period we asked for, in this case the number of times:
 - per month or
 - per week



3. Sentinel 2 data access and visualization

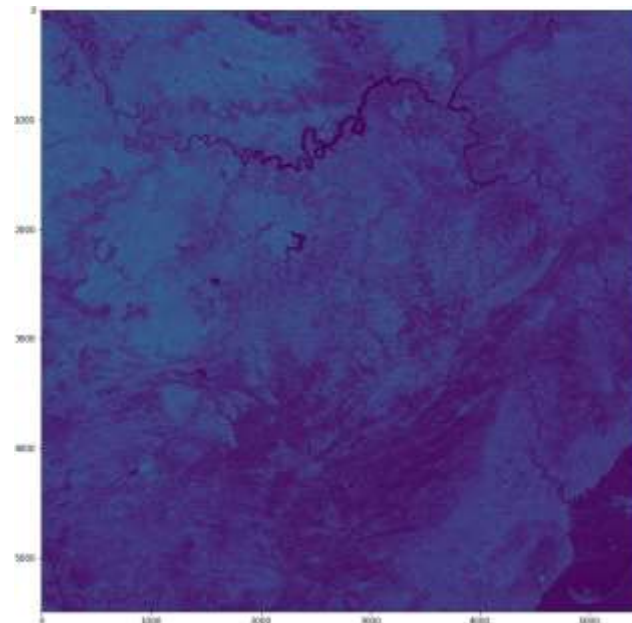


The screenshot shows the JupyterLab interface with a file browser. The file list includes:

Name	Last Modified	File size
-	για δευτερόλεπτα πριν	
1. Timeseries analysis and data visualization.ipynb	ένιας χρόνος πριν	230 kB
2. Bounding Box analysis.ipynb	ένιας χρόνος πριν	149 kB
3. Polygon analysis.ipynb	ένιας χρόνος πριν	481 kB
4. Rainfall event.ipynb	ένιας χρόνος πριν	158 kB
5-Copy1.1 Sentinel 2 data access and visualization.ipynb	6 μήνες πριν	1.03 MB
5.1 Sentinel 2 data access and visualization.ipynb	6 μήνες πριν	2.04 MB
Africa.geojson	ένιας χρόνος πριν	479 kB
bbox_test.tif	ένιας χρόνος πριν	101 kB
Level-2A_Algorithms_Equation_2.jpg	ένιας χρόνος πριν	14.9 kB
Level-2A_Algorithms_Equation_2a.jpg	ένιας χρόνος πριν	18.1 kB
meo_sso_cli.py	ένιας χρόνος πριν	1.6 kB
ndvi.tif	8 μήνες πριν	482 MB
polygon_test.tif	ένιας χρόνος πριν	1.26 MB
RGB.tif	ένιας χρόνος πριν	181 MB
RGB_2.tif	ένιας χρόνος πριν	723 MB
RGB_3.tif	6 μήνες πριν	181 MB
S2A_MSI_L1C_B02.tif	6 μήνες πριν	187 MB

Sentinel 2 data access and visualization

Build RGB



e 31

```
In [4]: import mgrs

In [5]: #we should define a latitude and longitude (this is Nairobi's centroid coordinates)
latitude = 41.38
longitude = 0.57

#then we initialize mgrs
m = mgrs.MGRS()
#and pass the coordinates as arguments
c = m.toMGRS(latitude, longitude)
#since the response is a byte object, we need to decode it into a string
c = c.decode("utf-8")
#we finally add the 'T' and select only the first 3 characters so it matches the WCS format
tiles = 'T'+c[:3]
tiles
```

```
Out[5]: 'T31T8F'
```

```
In [6]: bbox = ['1-2','1','35','17'] #we are not going to use them now
#the names match the collection names in the server
band_list = ['S2A_MSIL1C_B11', 'S2A_MSIL1C_B08', 'S2A_MSIL1C_B02']
#this is the most recent cloud free image of Nairobi
time_t = '2019-06-30T00:00:00,2019-06-30T23:59:59'
```

Now we can download the 3 bands. We will loop the band_list array to make it easier.

```
In [7]: access_token = connect_to_server()
for band in band_list:
    download(time_t,band,tiles, access_token)

print('Done!!!')
```

```
https://creodias01.eodataservice.org/wcs?service=WCS&request=GetCoverage&version=2.0.0&format=image/tiff&filter=false&subset=
n1x(2019-06-30T00:00:00,2019-06-30T23:59:59)&CoverageId=S2A_MSIL1C_B11&mgrs_tile=T31T8F&token=9c9e774660e74f4d9f65d0d4d04350d3
https://creodias01.eodataservice.org/wcs?service=WCS&request=GetCoverage&version=2.0.0&format=image/tiff&filter=false&subset=
n1x(2019-06-30T00:00:00,2019-06-30T23:59:59)&CoverageId=S2A_MSIL1C_B08&mgrs_tile=T31T8F&token=9c9e774660e74f4d9f65d0d4d04350d3
https://creodias01.eodataservice.org/wcs?service=WCS&request=GetCoverage&version=2.0.0&format=image/tiff&filter=false&subset=
n1x(2019-06-30T00:00:00,2019-06-30T23:59:59)&CoverageId=S2A_MSIL1C_B02&mgrs_tile=T31T8F&token=9c9e774660e74f4d9f65d0d4d04350d3
Done!!!
```

Once we have the images downloaded we can use Rasterio to stack the 3 images together and create an RGB image.

```
In [8]: band_list = ['S2A_MSIL1C_B11', 'S2A_MSIL1C_B08', 'S2A_MSIL1C_B02']
band_list = ['S2A_MSIL1C_B11', 'S2A_MSIL1C_B08', 'S2A_MSIL1C_B02']

# Read metadata of first file
with rasterio.open(band_list[0]) as src:
    meta = src.meta

# Update meta to reflect the number of layers
meta.update(count = len(band_list))

# Read each layer and write it to stack
with rasterio.open('rgb.tif', 'w', **meta) as dst:
    for i, layer in enumerate(band_list, start=1):
        with rasterio.open(layer) as src:
            dst.write(src.read(1), src.index(i))
```

[min_lat,max_lat,min_long,max_long] that we can

Sentinel 2 data access and visualization

Build NDVI

- The [Normalized Difference Vegetation Index](#) (NDVI) is a simple graphical indicator that can be used to assess whether an image being observed contains live green vegetation or not
- The formula for NDVI for Sentinel 2:
$$\text{Normalized Difference Vegetation Index (NDVI)} = \frac{\text{Band 8} - \text{Band 4}}{\text{Band 8} + \text{Band 4}}$$
- Depending on the sensor we are working on, the bands might change
- For Sentinel 2, the European Space Agency (ESA) has a dedicated [website](#) explaining how to build different indexes
- First, we define which bands we need to download, in this case band 4 and band 8
- Then we load the two bands with rasterio
In this way what we have are 2 numpy arrays

```
ndvi_list = ['S2A_MSIL1C_B04', 'S2A_MSIL1C_B08']
access_token = connect_to_server()
for band in ndvi_list:
    download(time_t,band,tiles, access_token)

print('Done!!')
```

```
src = rasterio.open('S2A_MSIL1C_B04.tif')
b04 = src.read(1)
src.close()

src = rasterio.open('S2A_MSIL1C_B08.tif')
b08 = src.read(1)
src.close()
```

Sentinel 2 data access and visualization

Build NDVI

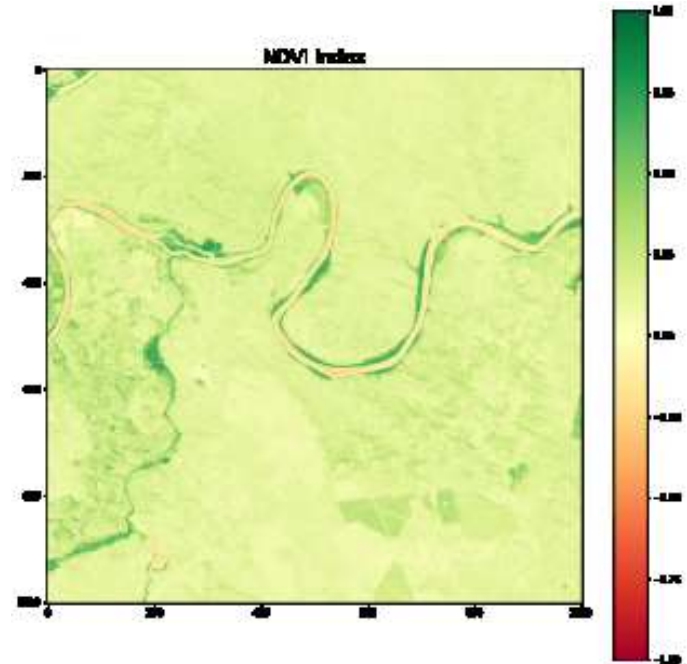
- We can compute the NDVI image
- We can pass all the spatial characteristics as arguments to the function
- And finally, we can write out the image

```
# Allow division by zero.  
np.seterr(divide='ignore', invalid='ignore')  
  
# Calculate NDVI.  
#ndvi = (b04.astype(float) - b08.astype(float)) / (b04 + b08)  
ndvi = (b08.astype(float) - b04.astype(float)) / (b08 + b04)
```

```
# Define spatial characteristics of output object  
#(basically they are analog to the input)  
kwargs = src.meta  
  
# Update kwargs (change in data type)  
kwargs.update(  
    dtype=rasterio.float32,  
    count = 1)  
  
# Let's see what is in there  
print(kwargs)
```

```
with rasterio.open('ndvi.tif', 'w', **kwargs) as dst:  
    dst.write_band(1, ndvi.astype(rasterio.float32))
```

```
# Take a spatial subset of the ndvi layer produced  
ndvi_sub = ndvi[2000:3000, 2000:3000]  
  
# Plot  
plt.subplots(figsize=(13,13))  
plt.imshow(ndvi_sub, cmap='RdYlGn', vmin=-1, vmax=1)  
plt.colorbar()  
plt.title('NDVI index', size=20)
```



Sentinel 2 data access and visualization

NDVI Timeseries

- We can access precomputed NDVI products using a WCS query

```
def download_timeseries(time_t, access_token):
    url = 'https://eod1.s2.meteosat.com/api/coverage/coverage?bbox=2.0,0.0,2.0,0.0&format=application/vnd+xml'
    url = 'https://eod1.s2.meteosat.com/api/coverage/coverage?bbox=2.0,0.0,2.0,0.0&format=application/vnd+xml'
    print(url)
    # Once the url is build, we can send a request to the server
    result = requests.get(url)
    xml_data = result.text

    tree = ET.fromstring(xml_data)

    # Extract the time delta from the xml file in order to build later the time array
    for t in tree.iter('http://www.opengis.net/gml/3.2/layerCover'):
        # split string into list of coordinates
        time_delta = t.text.split()
        time_delta = time_delta[1]

    for t in tree.iter('http://www.opengis.net/gml/3.2/layerCover'):
        # split string into list of ranges
        data = t.text
        Append(data)

    times = []
    # split input xml string element as list
    data = data.split()
    # split again
    data = data[0].split(',')

    data = [float(i) for i in data]
    for t in tree.iter('http://www.opengis.net/gml/3.2/rgrid/coefficients'):
        # extract dates
        dates = t.text

        dates = dates.split()

    for i in range(0, len(dates)):
        dates[i] = int(dates[i]) - int(time_delta)

    for i in range(0, len(dates)):
        times.append(datetime.fromtimestamp(int(dates[i])).strftime('%Y-%m-%d')) # Add hour also

    return times, data
```

- We define a time range e.g. period of one month (19.01.2019 – 18.02.2019)

```
time_t = '2019-01-19T00:00:00,2019-02-18T23:59:59'
access_token = connect_to_server()
times, data = download_timeseries(time_t, access_token)
```

- and we plot the data

```
f, ax = plt.subplots(figsize=(9, 6))
plt.plot(times, data)
ax.set_xticklabels(labels=times, rotation=45, ha='right')
ax.set(xlabel='Dates range', ylabel='NDVI')
```

