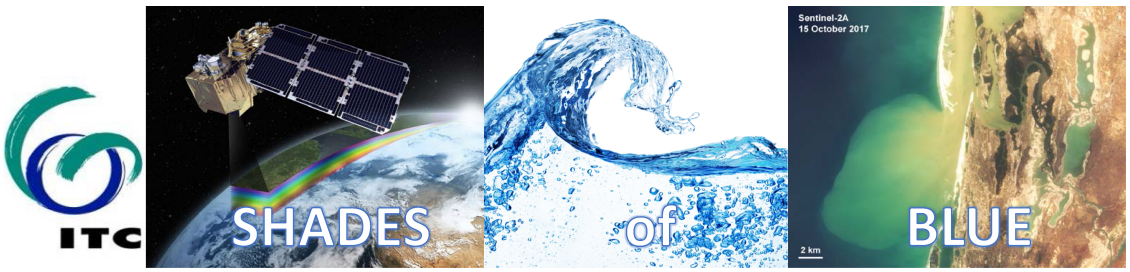
1.1_work_OC_TS

April 21, 2023



UNIVERSITY OF TWENTE.

1 Work with Ocean Color Products

Author: Suhyb Salama, ITC

1.1 Purpose

This exercise introduces you to Google Earth Engine (GEE) and its functionalities from within python script in jupyter notebook. The exercise is a follow up on the first exercise on Google Earth Engine.

1.2 Prerequisites

- You should have an account in Google Earth Engine. If not done yet please sign up now.
- You should have worked out the first exercise on Google Earth Engine

1.3 What you will learn

1. Access Google Earth Engine (GEE) with your account;
2. Access Ocean Color products on GEE;
3. Perform a temporal averaging and visualize the results;
4. Extract area averaged time series as PANDAS data frame.

1.4 What will you achieve

At the end of this exercise you will be able to use Google Earth Engine to access satellite data, visualize and manipulate these data using python scripts in jupyter notebooks.

1.4.1 The work comprises sequential steps that you will perform, are you ready?

1.4.2 *off you go. . .*

You may skip **ee.Authenticate()** step when you come here straight from the previous exercise because the authentication will be still valid.

Import Google Earth API. This API contains all the functionalities of Google Earth Engine (GEE)

[1]:
```
import ee
```

1.5 Initialize the authentication process.

This process will let you log in with your user name and password * if you have already Authenticate your entrance to GEE you can skip this step.

[2]:
```
try:
    ee.Initialize()
except Exception as e:
    ee.Authenticate()
    ee.Initialize()
```

1.6 Import Google Earth mapping capabilities

More information on the functionality of the *geemap* python package can be found at this link, which includes a cheat sheet, tutorials and book with valuable resources. The syntax of many of the Google Earth Engine functions that are also callable with the *geemap* site package can be found under the Client Libraries.

[3]:
```
import pandas as pd
import numpy as np
import geemap
import geemap.colormaps as cm
Map =geemap.Map()
```

1.7 Data Access

Get access to MODIS product of Ocean Color.

[4]:
```
MODIS_OC_DS = ee.ImageCollection("NASA/OCEANDATA/MODIS-Aqua/L3SMI")
```

1.7.1 Interrogate

get some information on the data set of *MODIS_ OC_ DS*

- Select one layer and list the band names
- The descriptions of bands provides details on the content of the data set.

[5]:
```
first_image = MODIS_OC_DS.first()
Bname = first_image.get('system:band_names')
print('system:band_names', Bname.getInfo())
```

```
system:band_names ['chlor_a', 'nflh', 'poc', 'Rrs_412', 'Rrs_443', 'Rrs_469',
'Rrs_488', 'Rrs_531', 'Rrs_547', 'Rrs_555', 'Rrs_645', 'Rrs_667', 'Rrs_678',
'sst']
```

1.7.2 Data filtering

- Select Chlorophyll a concentration products *chlor_ a* and filter the product to staring and ending times

[6]:
```
selected_bands = 'chlor_a'
sY=2003
sM=1
sD=1
eY=2021
eM=12
eD =31
start_date = ee.Date.fromYMD(sY, sM, sD)
end_date   = ee.Date.fromYMD(eY, eM, eD)

ChlaDS = MODIS_OC_DS.select(selected_bands).filter(ee.Filter.date(start_date,␣
 ↪end_date))
```

- Count how many images

[7]:
```
count = ChlaDS.size()
valCount=count.getInfo()
print('Numbre of images is : ', str(valCount)+'\n')
```

```
Numbre of images is :  6849
```

- Compute the average Chl-a for the whole period

[8]:
```
DecadalMeanChla= ChlaDS.reduce(ee.Reducer.mean())
```

1.8 Visualize the calculated mean.

- Select a color palette from available library. GEE uses the same convention used in MatPlotLib package for python.

- Visualization requires three parameters, the min (= 0) and max (= 0.6 works for this example) values on which the color palette will be stretched, and the palette.

- Set the center point {longitude (long), latitude (lat)} of the visualization and a zoom-in scale using *Map.setCenter(long, lat, scale)*

- Add the Layer to the Google map and give it a name, here we use *MeanChl-a*

[9]:
```
val= cm.palettes.Spectral
palette=val.default

Vis = {
  'min': 0,
  'max': 0.65,
  'palette': palette}

Map.setCenter(-30,14,5)
Map.addLayer(DecadalMeanChla, Vis, 'MeanChl-a')
Map
```

```
Map(center=[14, -30], controls=(WidgetControl(options=['position',␣
 ↪'transparent_bg'], widget=HBox(children=(To...
```

1.9 Time aggregation

1.9.1 Compute the monthly mean

To bin the daily data into monthly, we need two *for loops*, one for the years and one for the months. An Alternative to using *for loops* is to apply the *.map* functionality twice, first on years and second on months.

The script below shows a nested functions of two application of *.map*. The core for binning daily to monthly data is the function **createMean**.

createMean: for a specific year, it iterate on the months, whereby per month the daily data are summed. The for loop on the *years* would iterate on the years. this function is encapsulated by the **getMonthlyMean** which run **createMean** on months. Later the **getMonthlyMean** is run on years.

- Create a series of years and months to loop on

[10]:
```
years = ee.List.sequence(sY, eY)
months = ee.List.sequence(sM, eM)
```

- Define the functions **getMonthlyMean** and **createMean**

[11]:
```
def getMonthlyMean(iy):
    def createMean(im):
        sDate = ee.Date.fromYMD(iy, im, sD)
        eDate = sDate.advance(1, 'month')
        monthFiltered = ChlaDS.filter(ee.Filter.date(sDate, eDate))
        msum = monthFiltered.reduce(ee.Reducer.mean())
        return msum.set({
            'system:time_start': sDate.millis(),
            'system:time_end': eDate.millis(),
```

```
            'year': iy,
            'month': im,
            'date': sDate.millis()})
    return months.map(createMean)
```

Apply monthly binning Run the monthly binning function and visualize the first month and investigate the results: 1. count the number of images in the image collection resulting from the function **getMonthlyMean** 2. add the the first image and visualize, use the *'MonthlyChl-a* as the name of the layer.

[12]:
```
monthlyChla = ee.ImageCollection.fromImages(years.map(getMonthlyMean).flatten())

count = monthlyChla.size()
valCount=count.getInfo()
print('Numbre of monthly mean images is : ', str(valCount)+'\n')

first_image = monthlyChla.first()
Map.setCenter(-30,14,5)
Map.addLayer(first_image, Vis, 'MonthlyChl-a')
Map
```

```
Numbre of monthly mean images is :  228
```

```
Map(center=[14, -30], controls=(WidgetControl(options=['position',␣
 ↪'transparent_bg'], widget=HBox(children=(To...
```

1.10 Time series extraction

To work with a time series you can extract a point or a region. Here we will extract the time series of a an area. To be able to do so the following steps are needed 1. Select the region of interest (ROI); 2. Clip the image to the ROI and average over the region; 3. Extract the time series of the areal average;

1.10.1 Spatial Sub set

- Load the bounding box in the the Mediterranean sea
- You can easily create a geojson file using the interactive geojson map

[13]:
```
seas_shapefile_path = "./../0-Course-info/data/Geojason/boxMed.geojson"
seas =  geemap.geojson_to_ee(seas_shapefile_path)
```

- The function **getTM4RoI** will iterate on the images in an image collection and use the nested function **iter_func** to extract the date and calculate the mean of a region using **.reduceRegion.** The function will return these values as a list of two columns ['Dates', 'Chla'].

[14]:
```
def getTM4RoI(imgcol):
    def iter_func(image, newlist):
        date = ee.Number.parse(image.date().format("YYYYMMdd"))
        stat= image.reduceRegion(
            reducer = ee.Reducer.mean(),
            geometry = seas,
            scale = 5000,
            maxPixels=1e15)

        newlist = ee.List(newlist)
        res = newlist.add([date, stat])
        return ee.List(res)
    ymd = imgcol.iterate(iter_func, ee.List([]))
    return list(ee.List(ymd).getInfo())
```

[15]:
```
inList = getTM4RoI(monthlyChla)
```

1.10.2 Convert to a Data Frame

The function **Convert2TM_DF** will take the generated list from **getTM4RoI** and convert to a *Pandas Data frame* with recognizable dates. This step will make time series analysis much easier. *Pandas Data frame* is a python library and is outside GEE.

[16]:
```
def Convert2TM_DF(inList):
    newList =[]
    for i, item in enumerate(inList):
        if np.any(item[1].get('chlor_a_mean')):
            newList.append([item[0],item[1].get('chlor_a_mean')])
    DF=pd.DataFrame(newList,columns=['Dates' ,'Chl-a'])
    datetime_series = pd.to_datetime(DF['Dates'],format='%Y%m%d')
    DF.drop('Dates', axis=1, inplace=True)
    DF.set_index(datetime_series, inplace=True,drop=True)
    DF.mask(DF.eq('None')).dropna()
    return DF
```

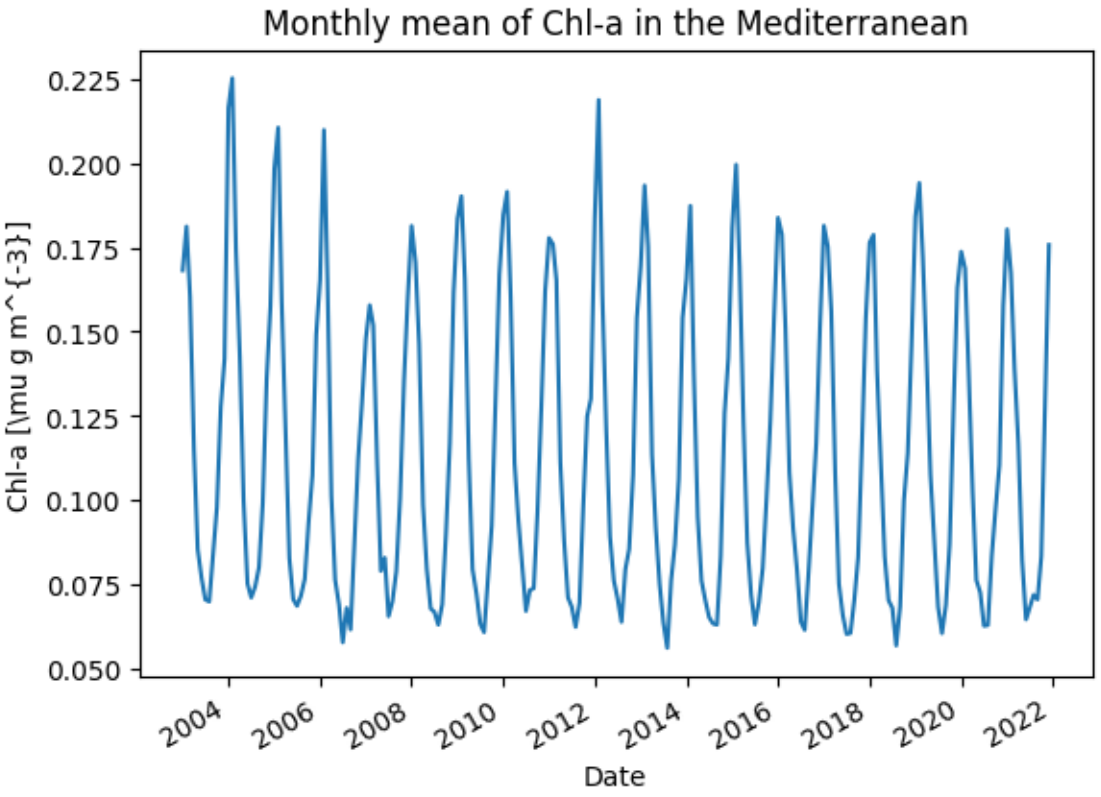
Extract time series on monthly climatology To do so, apply the function **getTM4RoI** first and then use its output in the function **Convert2TM_DF** Or you can do it in one run *res=Convert2TM_DF(getTM4RoI(ImgCol,Name))* as shown in later examples (exercise week 4).

[17]:
```
TM_Chla  = Convert2TM_DF(inList)
```

Visualize the monthly mean with Pandas capabilities

[18]:
```
ax_=TM_Chla.plot(kind='line',
                    legend= False,
                    ylabel='Chl-a [\mu g m^{-3}]',
                    xlabel='Date',
```

```
                    title='Monthly mean of Chl-a in the Mediterranean')
```



1.11 End exercise

[ ]: