Lecture 09

GEE Time Series:

extract pixel values over time

2024-04-29

Sébastien Valade

Previous lecture:

GEE change detection (naive two-images differencing):

⇒ selection of pre/post-event images & `subtract` bands

⇒ change visualization as *map layers*

Today:

GEE time series analysis:

⇒ `map` through ImageCollections to extract pixel values over time

⇒ change visualization as *time series plots*

1. Introduction

2. Case example: Kutupalong refugee camp
    1. Get areas of interest
    2. Get image collection
    3. Map through collection
    4. Get time series, plot, analyze
    5. Visualize specific images, export timelapse

Understanding the temporal dynamics of a phenomenon is best achieved by extracting quantities of a variable over time, and plotting them as time series plots.

⇒ In previous lectures you learned how to `filter` `ImageCollections`, and how to use `reducers` to recover statistics from an `Image`:

EX: `filter` an image collection by date, location, metadata, etc.
EX: `reduce` an image to get its mean, median, min, max, etc.

⇒ In this lecture, we will learn how to:

- `map` through `ImageCollections`, in order to extract relevant values over time
  EX: apply mathematical operations to each image (e.g. compute spectral indices), and successively apply a reducer to extract statistics over time
- export the relevant values as `Pandas dataframes`, in order to plot/analyze the change through time
- carry your analysis on specific regions, imported as `FeatureCollections` from file formats commonly used in Geographic Information Systems (e.g., kml, geojson, shapefile)

Understanding the temporal dynamics of a phenomenon is best achieved by extracting quantities of a variable over time, and plotting them as time series plots.

⇒ In previous lectures you learned how to `filter` `ImageCollections`, and how to use `reducers` to recover statistics from an `Image`:

EX: `filter` an image collection by date, location, metadata, etc.
EX: `reduce` an image to get its mean, median, min, max, etc.

⇒ In this lecture, we will learn how to:

- `map` through `ImageCollections`, in order to extract relevant values over time
  EX: apply mathematical operations to each image (e.g. compute spectral indices), and successively apply a reducer to extract statistics over time
- export the relevant values as `Pandas dataframes`, in order to plot/analyze the change through time
- carry your analysis on specific regions, imported as `FeatureCollections` from file formats commonly used in Geographic Information Systems (e.g., kml, geojson, shapefile)

Exercise:

⇒ The exercise consists in detecting the changes related the Kutupalong refugee camp, the world's largest refugee camp located in Bangladesh, inhabited by Rohingya refugees who fled from ethnic and religious persecution in neighboring Myanmar in late 2017.

⇒ The workflow will be as followed:

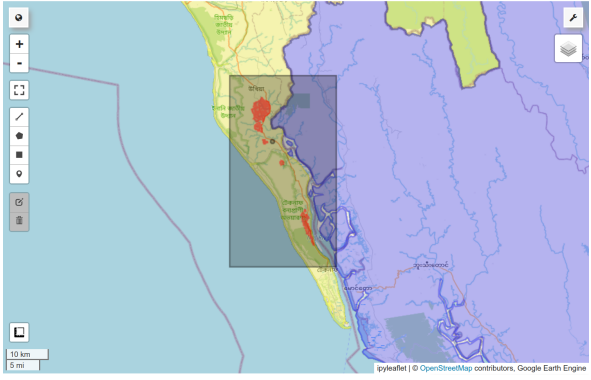
1. Get area(s) of interest
   ⇒ import camp boundaries (kml file imported as `FeatureCollection`), country boundaries, etc.
2. Get image collection(s)
3. Map through image collection(s)
4. Get time series as a DataFrame, plot and analyze

Step 1: Get areas of interest

⇒ Several areas of interested can be defined:

- camp boundaries ⇒ download kml file (link), save to Google Drive, & import as `FeatureCollection`
- country boundaries

Areas of interest



Step 1: Get areas of interest

⇒ Several areas of interested can be defined:

- camp boundaries ⇒ download kml file (link), save to Google Drive, & import as `FeatureCollection`
- country boundaries

```
# Get country boundaries
fc_bangladesh = ee.FeatureCollection("FAO/GAUL/2015/level0").filter(ee.Filter.eq('ADM0_NAME', 'Bangladesh'))
fc_myanmar = ee.FeatureCollection("FAO/GAUL/2015/level0").filter(ee.Filter.eq('ADM0_NAME', 'Myanmar'))

# Get camp boundaries
f_kml = '/content/drive/MyDrive/Colab Notebooks/DIP/rois/Kutupalong_20230412_a1_camp_outlines.kml'
fc_camp = geemap.kml_to_ee(f_kml)                    # Import camp boundaries to FeatureCollection
geom_campbounds = fc_camp.geometry().bounds()        # Get camp bounding box
geom_aoi = geom_campbounds.buffer(5000).bounds()     # Get camp bounding box with buffer
geom_poi = fc_camp.geometry().centroid(maxError=1)   # Get camp centroid

Map = geemap.Map()
Map.centerObject(fc_camp, zoom=11)
Map.addLayerControl()
Map.addLayer(fc_bangladesh, {'color':'yellow', 'opacity':.5}, "Bangladesh")
Map.addLayer(fc_myanmar, {'color':'blue', 'opacity':.5}, "Myanmar")
Map.addLayer(geom_aoi, {'opacity': 0.5}, "AOI")
Map.addLayer(geom_poi, {'opacity': 0.5}, "POI")
Map.addLayer(fc_camp.draw(color='red'), {'opacity': 0.5}, 'Kutupalong refugee camp') # draw multiple features
```

Step 2: Get image collection(s)

⇒ We can start with the VIIRS (*Visible Infrared Imaging Radiometer Suite*) collection NOAA/VIIRS/DNB/MONTHLY_V1/VCMCFG:

→ stores monthly average radiance composite images (band `avg_rad`, in nW/sr/cm^2), wich are computed from VIIRS nighttime data in the Day/Night Band (DNB)

→ images are composited monthly, meaning that it takes for each pixel only the cloud-free observations: the band `cf_cvg` stores the total number of observations that went into each pixel (You can use it to identify areas with low numbers of observations where the quality is reduced).

```
# Get VIIRS Nighttime average radiance collection
ti, tf = '2014-01-01','2024-01-01'
ic_viirs_avgrad = ee.ImageCollection("NOAA/VIIRS/DNB/MONTHLY_V1/VCMSLCFG").filterDate(ti, tf)
```

Step 3: Map through collection(s)

1. Create a function to:
   - calculate the *mean value* of a band in the area of interest
   - store as properties both the *mean value* and the *image date*
2. `Map` through the image collection to apply the function to each image

```
# Function to get mean value in aoi
def aoi_mean(img, geometry, band):
    mean = img.reduceRegion(reducer=ee.Reducer.mean(), geometry=geometry, scale=30).get(band)
    return img.set('date', img.date().format()).set(f'mean_{band}', mean)

# Map through image collection to get mean values
geometry = fc_camp.geometry()
ic_viirs_avgrad = ic_viirs_avgrad.map(lambda img: aoi_mean(img, geometry, 'avg_rad'))
ic_viirs_avgrad = ic_viirs_avgrad.map(lambda img: aoi_mean(img, geometry, 'cf_cvg'))
```

Step 4: Get time series as a DataFrame, plot and

NB: contrary to EE's JavaScript API (available through Earth Engine Code Editor), the Python API does not provide the `ui module`, which is used to generate interactive charts. For this reason, we here export to Pandas dataframe and plot with matplotlib.

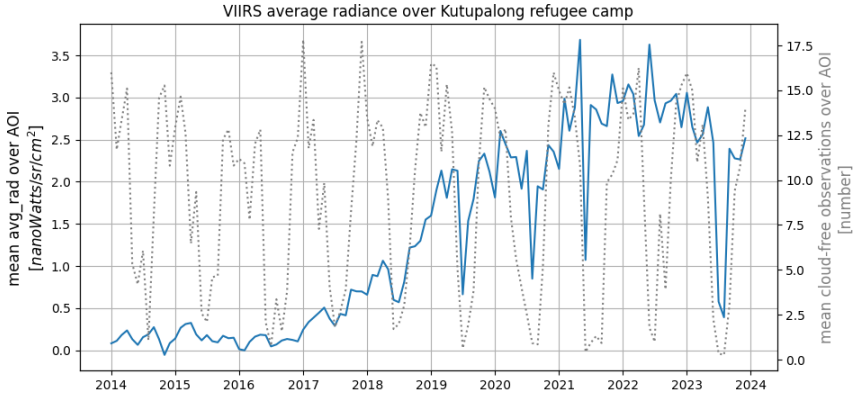
1. use `.reduceColumns` to convert selected `ImageCollection` properties to list
2. use `Pandas` to convert the list to a `DataFrame`

```
# Convert properties to list
list_datemean = ic_viirs_avgrad.reduceColumns(ee.Reducer.toList(3),
                                              ['date', 'mean_avg_rad', 'mean_cf_cvg']
                                              ).get('list')

# Convert list to dataframe
import pandas as pd
df_avgrad = pd.DataFrame(list_datemean.getInfo(), columns=['date', 'mean_avg_rad', 'mean_cf_cvg'])
df_avgrad['date'] = pd.to_datetime(df_avgrad['date'])
```

Step 4: Get time series as a DataFrame, plot and analyze

3. plot the `DataFrame`

NB: notice the increase in the radiance values in 2017, which corresponds to the arrival of the refugees and the development of the refugee camp.

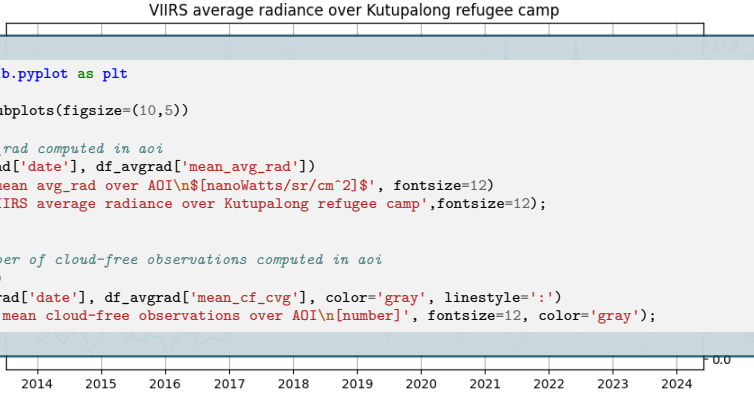


Step 4: Get time series as a DataFrame, plot and analyze

4. plot the `DataFrame`

NB: notice the increase in the radiance values in 2017, which corresponds to the arrival of the refugees and the development of the refugee camp.

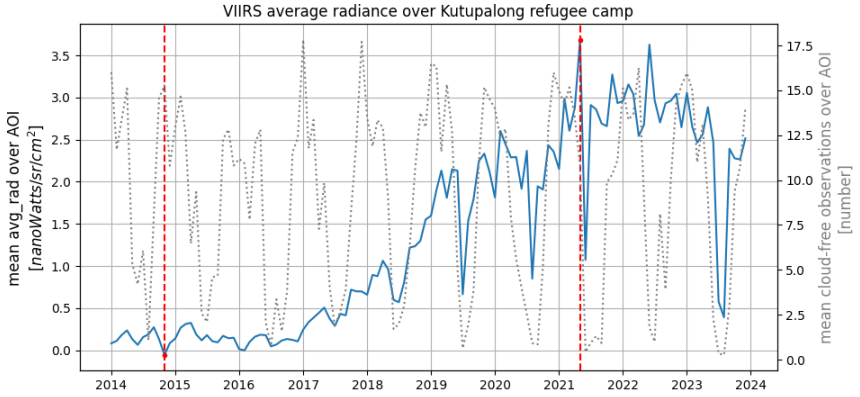
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10,5))

# Plot mean avg_rad computed in aoi
ax.plot(df_avgrad['date'], df_avgrad['mean_avg_rad'])
ax.set_ylabel('mean avg_rad over AOI\n$[nanoWatts/sr/cm^2]$', fontsize=12)
ax.set_title('VIIRS average radiance over Kutupalong refugee camp',fontsize=12);
ax.grid()

# Plot mean number of cloud-free observations computed in aoi
ax2 = ax.twinx()
ax2.plot(df_avgrad['date'], df_avgrad['mean_cf_cvg'], color='gray', linestyle=':')
ax2.set_ylabel('mean cloud-free observations over AOI\n[number]', fontsize=12, color='gray');
```



Step 4: Get time series as a DataFrame, plot and analyze

5. analyze the `DataFrame`

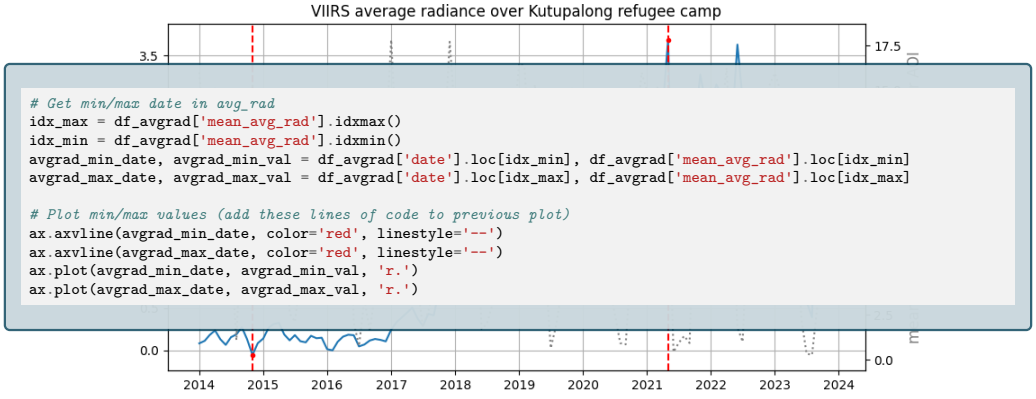
NB: the type of analysis depends on what information you want to extract. We here show a simple example of how to get the date of the minimum and maximum avg_radiance values, and recover the associated images.



Step 4: Get time series as a DataFrame, plot and analyze

5. analyze the `DataFrame`

NB: the type of analysis depends on what information you want to extract. We here show a simple example of how to get the date of the minimum and maximum avg_radiance values, and recover the associated images.

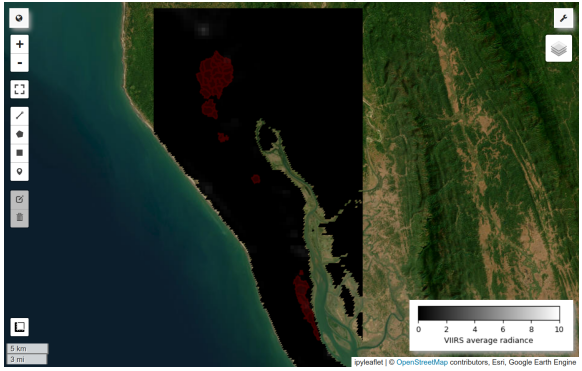



```
# Get min/max date in avg_rad
idx_max = df_avgrad['mean_avg_rad'].idxmax()
idx_min = df_avgrad['mean_avg_rad'].idxmin()
avgrad_min_date, avgrad_min_val = df_avgrad['date'].loc[idx_min], df_avgrad['mean_avg_rad'].loc[idx_min]
avgrad_max_date, avgrad_max_val = df_avgrad['date'].loc[idx_max], df_avgrad['mean_avg_rad'].loc[idx_max]

# Plot min/max values (add these lines of code to previous plot)
ax.axvline(avgrad_min_date, color='red', linestyle='--')
ax.axvline(avgrad_max_date, color='red', linestyle='--')
ax.plot(avgrad_min_date, avgrad_min_val, 'r.')
ax.plot(avgrad_max_date, avgrad_max_val, 'r.')
```

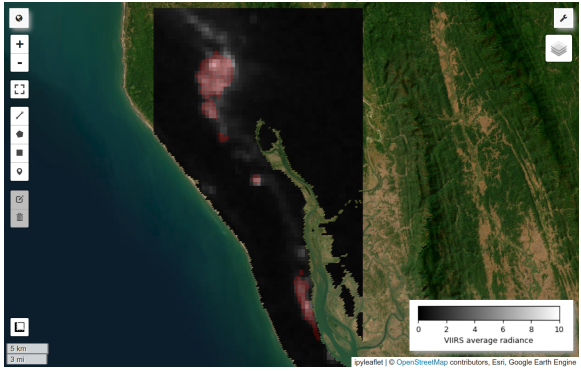
Step 5: **Visualize specific images, export timelapse, etc.**

1. Display VIIRS images corresponding to min/max avg_rad
   $\Rightarrow$ *filter collection with corresponding date + clip to area of interest + mask out water*

VIIRS avg_rad minimum (2014-11-01)



VIIRS avg_rad maximum (2021-05-01)

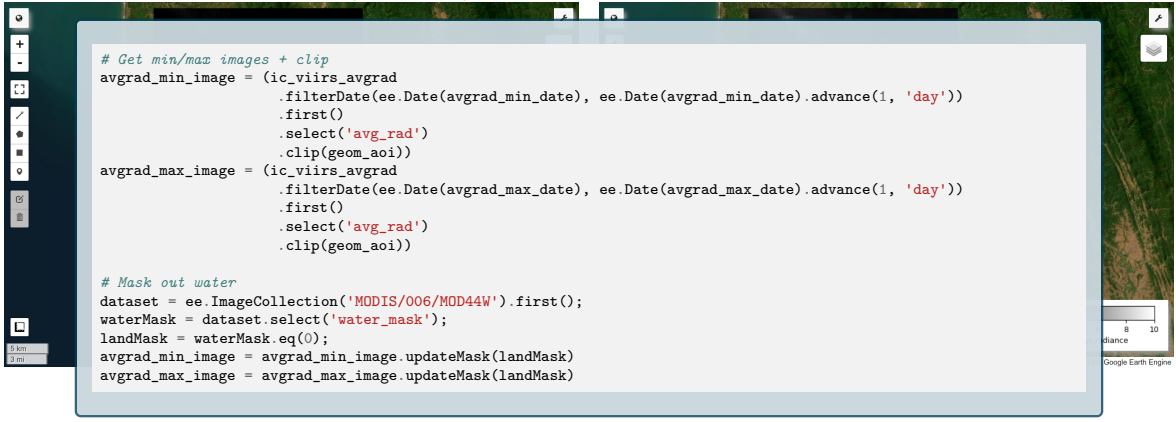


Step 5: Visualize specific images, export timelapse, etc.

1. Display VIIRS images corresponding to min/max avg_rad
   ⇒ *filter collection with corresponding date + clip to area of interest + mask out water*

VIIRS avg_rad minimum (2014-11-01)

VIIRS avg_rad maximum (2021-05-01)



```
# Get min/max images + clip
avgrad_min_image = (ic_viirs_avgrad
                    .filterDate(ee.Date(avgrad_min_date), ee.Date(avgrad_min_date).advance(1, 'day'))
                    .first()
                    .select('avg_rad')
                    .clip(geom_aoi))
avgrad_max_image = (ic_viirs_avgrad
                    .filterDate(ee.Date(avgrad_max_date), ee.Date(avgrad_max_date).advance(1, 'day'))
                    .first()
                    .select('avg_rad')
                    .clip(geom_aoi))

# Mask out water
dataset = ee.ImageCollection('MODIS/006/MOD44W').first();
waterMask = dataset.select('water_mask');
landMask = waterMask.eq(0);
avgrad_min_image = avgrad_min_image.updateMask(landMask)
avgrad_max_image = avgrad_max_image.updateMask(landMask)
```

Step 5: Visualize specific images, export timelapse, etc.

2. Export a timelapse of the `ImageCollection`

   ⇒ *use e.g.* geemap.create_timelapse *to get an animated gif/mp4 of the* `ImageCollection`

   2.1 Install the `ffmpeg` in the Colab environment:

```
!apt-get install ffmpeg
```

   2.2 Create the timelapse, then download from the folder `/content/` in your GoogleDrive:

```
geemap.create_timelapse(ic_viirs_avgrad,
       start_date='2013-01-01',
       end_date='2024-01-01',
       region=geom_aoi,
       bands='avg_rad',
       vis_params={'palette':['black', 'white'], 'min':0, 'max':10},
       font_color='white',
       frames_per_second=10,
       frequency='month',
       reducer='mean',
       dimensions=600,
       add_text=True,
       add_progress_bar=True,
       out_gif='Kutupalong_VIIRS_2013-2014.gif'
)
```

Step 5: **Visualize specific images, export timelapse, etc.**

3. Display high-resolution images before/after the camp development using `geemap.split_map`
   ⇒ *use e.g. Sentinel-2 images (>2015) with 10 meter resolution optical bands*

Step 5: Visualize specific images, export timelapse, etc.

4. Display high-resolution images before/after the camp development using `geemap.split_map`

   ⇒ *use e.g. Sentinel-2 images (>2015) with 10 meter resolution optical bands*

```
# Select collection
collection, ti_pre, ti_post = 'COPERNICUS/S2_HARMONIZED', '2015-01-01', '2018-01-01'
vis_params = {'bands': ['B4', 'B3', 'B2'], 'min': 0, 'max': 3000}

# Select image_pre
ti = ee.Date(ti_pre); tf = ti.advance(1, 'year')
image_pre = (ee.ImageCollection(collection).filterBounds(geom_poi).filterDate(ti, tf)
.filter(ee.Filter.calendarRange(11, 12, 'month')).sort("CLOUD_COVER").first())
image_pre_date = ee.Date(image_pre.get('system:time_start')).format('YYYY-MM-dd').getInfo()

# Select image_post
ti = ee.Date(ti_post); tf = ti.advance(1, 'year')
image_post = (ee.ImageCollection(collection).filterBounds(geom_poi)
.filterDate(ti, tf).filter(ee.Filter.calendarRange(11, 12, 'month')).sort("CLOUD_COVER").first())
image_post_date = ee.Date(image_post.get('system:time_start')).format('YYYY-MM-dd').getInfo()

# Plot split map
left_layer = geemap.ee_tile_layer(image_pre, vis_params, f'Pre {image_pre_date}')
right_layer = geemap.ee_tile_layer(image_post, vis_params, f'Post {image_post_date}')
Map = geemap.Map()
Map.centerObject(geom_poi, 10)
Map.split_map(left_layer, right_layer, left_label=image_pre_date, right_label=image_post_date)
```