## Calculating NDVI from Sentinel 2 data

// Step 1. Use the “Draw a shape” tool to identify the area for analysis. (In this code, “aoi” (area of interest) is that shape or area.

// Set the map to Satellite mode

Map.setOptions('SATELLITE');

// Assign the area to a variable called “geometry”. All analysis will be done for “geometry”. This helps in the easy substitution of study areas in the code.

var geometry = aoi;

Map.centerObject(geometry);

// Import Sentinel 2 data

var s2 = ee.ImageCollection("COPERNICUS/S2");

// Filter the complete dataset to extract only the data corresponding to the required maximum cloudy pixel percentage, dates, and area.

var filtered = s2.filter(ee.Filter.lt('CLOUDY\_PIXEL\_PERCENTAGE', 30))

.filter(ee.Filter.date('2010-01-01', '2011-01-01'))

.filter(ee.Filter.bounds(geometry));

var im = filtered;

print(im)

//Obtain the median values for the Sentinel bands over the filtered dates.

var image = im.median();

print(image)

// Calculate Normalized Difference Vegetation Index (NDVI)

// 'NIR' (B8) and 'RED' (B4)

var ndvi = image.normalizedDifference(['B8', 'B4']).rename(['ndvi']);

// Calculate Modified Normalized Difference Water Index (MNDWI)

// 'GREEN' (B3) and 'SWIR1' (B11)

var mndwi = image.normalizedDifference(['B3', 'B11']).rename(['mndwi']);

// Calculate NDRE (NIR B8, RedEdge B5) - nitrogen status

var ndre = image.normalizedDifference(['B8', 'B5']).rename(['ndre']);

// Calculate SRRE (NIR/RedEdge = B8/B5) - nitrogen content

var srre = image.expression(

'NIR / RedEdge', {

'NIR': image.select('B8'),

'RedEdge': image.select('B5'),

}).rename('srre');

// Calculate Soil-adjusted Vegetation Index (SAVI)

// 1.5 \* ((NIR - RED) / (NIR + RED + 0.5))

// Note:

// For the SAVI formula, the pixel values need to converted to reflectances

// Multiplying the pixel values by 'scale' gives us the reflectance value

// The scale value is 0.0001 for Sentinel-2 dataset

var savi = image.expression(

'1.5 \* ((NIR - RED) / (NIR + RED + 0.5))', {

'NIR': image.select('B4').multiply(0.0001),

'RED': image.select('B3').multiply(0.0001),

}).rename('savi');

var ndviVis = {min:0, max:1, palette: ['brown', 'white', 'green']}

var ndwiVis = {min:-1, max:1, palette: ['red', 'white', 'blue']}

var ndreVis = {min:-0.5, max:1, palette: ['red', 'white', 'blue']}

var srreVis = {min:0, max:4, palette: ['red', 'white', 'blue']}

Map.addLayer(mndwi.clip(geometry), ndwiVis, 'mndwi',1)

Map.addLayer(savi.clip(geometry), ndviVis, 'savi',1)

Map.addLayer(ndvi.clip(geometry), ndviVis, 'ndvi',1)

Map.addLayer(ndre.clip(geometry), ndreVis, 'ndre',1)

Map.addLayer(srre.clip(geometry), srreVis, 'srre',1)

## Calculating time series of indices

## // Can be used as extension of the previous code. Or independently after defining the “geometry” and // centering Map over it.

## // Define what kind of reducer to use

## var redu = ee.Reducer.mean();

## // Write a function for Cloud masking

## function maskS2clouds(image) {

## var qa = image.select('QA60');

## var cloudBitMask = 1 << 10;

## var cirrusBitMask = 1 << 11;

## var mask = qa.bitwiseAnd(cloudBitMask).eq(0).and(

## qa.bitwiseAnd(cirrusBitMask).eq(0));

## return image.updateMask(mask)//.divide(10000)

## .select("B.\*")

## .copyProperties(image, ["system:time\_start"]);

## }

## var filtered = filtered.map(maskS2clouds).map(function(image){return image.clip(geometry)});

## // Write a function that computes NDVI and NDWI for an image and adds it as a band

## function addNDVI(image) {

## var ndvi = image.normalizedDifference(['B8', 'B4']).rename('ndvi');

## var ndwi = image.normalizedDifference(['B3', 'B11']).rename('ndwi');

## var nddi = image.expression(

## '(NDVI-NDWI)/(NDVI+NDWI)', {

## 'NDVI':ndvi.multiply(0.0001),

## 'NDWI':ndwi.multiply(0.0001),

## }).rename('nddi');

## return image.addBands([ndvi, ndwi, nddi]);

## }

## // Map the function over the collection

## var withNdvi = filtered.map(addNDVI);

## // Map.addLayer(withNdvi.select('nddi').limit(1))

## var nddi\_minMax = withNdvi.select('nddi').reduce({

## reducer: ee.Reducer.minMax()

## });

## // Display a time-series chart

## var chart = ui.Chart.image.series({

## imageCollection: withNdvi.select(['ndvi', 'ndwi']),

## region: geometry,

## reducer: redu

## }).setOptions({

## lineWidth: 1.5,

## title: ['Time Series'],

## interpolateNulls: true,

## vAxis: {title: 'Index'},

## hAxis: {title: '', format: 'YYYY-MMM'}

## });

## print(chart);

## var chart1 = ui.Chart.image.series({

## imageCollection: withNdvi.select('nddi'),

## region: geometry,

## reducer: redu

## }).setOptions({

## lineWidth: 1.5,

## title: 'Drought Index Time Series',

## interpolateNulls: true,

## vAxis: {title: 'NDDI'},

## hAxis: {title: '', format: 'YYYY-MMM'}

## });

print(chart1);

## Creating GIF animation of MODIS-derived NDVI change over time

// First define the area where you want to do the analysis. This will be the “Tatarstan” shapefile.

// Fetch a MODIS NDVI collection and select NDVI.

var col = ee.ImageCollection('MODIS/006/MOD13A2').select('NDVI');

var mask = ee.FeatureCollection("users/raghujana/Tatarstan\_boundary");

// or use from my account //“https://code.earthengine.google.com/?asset=users/raghujana/Tatarstan\_boundary”

// Map.setCenter(39.422, 45.25, 7)

Map.centerObject(mask)

// Define the regional bounds of animation frames. The coordinates should be the corners of a box around the area of interest (“Tatarstan”).

var region = ee.Geometry.Polygon(

[[[46.72, 57.0],

[46.72, 53.8],

[54.67, 53.8],

[54.67, 57.0]]],

null, false

);

// Add day-of-year (DOY) property to each image.

col = col.map(function(img) {

var doy = ee.Date(img.get('system:time\_start')).getRelative('day', 'year');

return img.set('doy', doy);

});

// Get a collection of distinct images by 'doy'.

var distinctDOY = col.filterDate('2018-01-01', '2021-01-01');

// Define a filter that identifies which images from the complete

// collection match the DOY from the distinct DOY collection.

var filter = ee.Filter.equals({leftField: 'doy', rightField: 'doy'});

// Define a join.

var join = ee.Join.saveAll('doy\_matches');

// Apply the join and convert the resulting FeatureCollection to an ImageCollection.

var joinCol = ee.ImageCollection(join.apply(distinctDOY, col, filter));

// Define RGB visualization parameters.

var visParams = {

min: 0.0,

max: 9000.0,

palette: [

'FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163', '99B718', '74A901',

'66A000', '529400', '3E8601', '207401', '056201', '004C00', '023B01',

'012E01', '011D01', '011301'

],

};

// Create RGB visualization images for use as animation frames.

var rgbVis = joinCol.map(function(img) {

return img.visualize(visParams).clip(mask);

});

// Define GIF visualization arguments.

var gifParams = {

'region': region,

'dimensions': 600,

'crs': 'EPSG:3857',

'framesPerSecond': 10,

'format': 'gif'

};

print(ui.Thumbnail(rgbVis, gifParams))

// Print the GIF URL to the console.

print(rgbVis.getVideoThumbURL(gifParams));