Training Manual on

Satellite-Based Agricultural Analysis Using Google Earth Engine (GEE)

Course Director:

Hasan Md. Hamidur Rahman, Director, Computer & GIS Unit

Course Coordinator:

Hasan Mahmud, Senior System Analyst, Computer & GIS Unit

Assistant Course Coordinator:

Al-Helal, Programmer, Computer and GIS Unit

Compiled and Edited by,

Hasan Md. Hamidur Rahman

Hasan Mahmud

Al-Helal

Computer & GIS Unit
Bangladesh Agricultural Research Council

Published by

Computer and GIS Unit, BARC, Farmgate, Dhaka 1215, Bangladesh.

Date of Publication

May 2026

Contributors:

1. Mr. Hasan Md. Hamidur Rahman, Director, Computer and GIS Unit, BARC
2. Mr. Hasan Mahmud, Senior System Analyst, Computer and GIS Unit, BARC
3. Mr. Al-Helal, Programmer, Computer & GIS, BARC
4. Mr. Md. Zahid Hasan Siddiquee, Associate GIS & Remote Sensing Specialist, IWM
5. Mr. Mustafa Kamal, GIS and Remote Sensing Specialist International, Maize and Wheat Improvement Center (CIMMYT)
6. Mr. Mahmudul Hasan, GIS & Remote Sensing Specialist, ESRI, Bangladesh

Designed by

Mohammad Nazmul Islam, Graphics Designer, BARC

Computer & GIS Unit
Bangladesh Agricultural Research Council

Table of Content

Getting Started with Google Earth Engine

Mr. Al-Helal
Programmer (Computer & GIS Unit)
Bangladesh Agricultural Research Council
Email: al.helal@barc.gov.bd

Sign-up for Google Earth Engine (GEE)

A Step-by-Step Guide for Creating a Google Earth Engine Account with a Cloud Project

◆ **For First-Time GEE Users**

1. Visit the official Google Earth Engine registration page:
   Google Earth Engine Project Registration Page
2. Sign in using your Google account.
3. Click **"Register"** to begin the registration process.
4. Create or select a **Google Cloud Project**.
5. Accept the Terms of Service.
6. Submit the registration form.
7. Wait for approval from the Google Earth Engine team.
8. After approval, open the Earth Engine Code Editor:
   Google Earth Engine Code Editor

◆ **For Existing GEE Users**

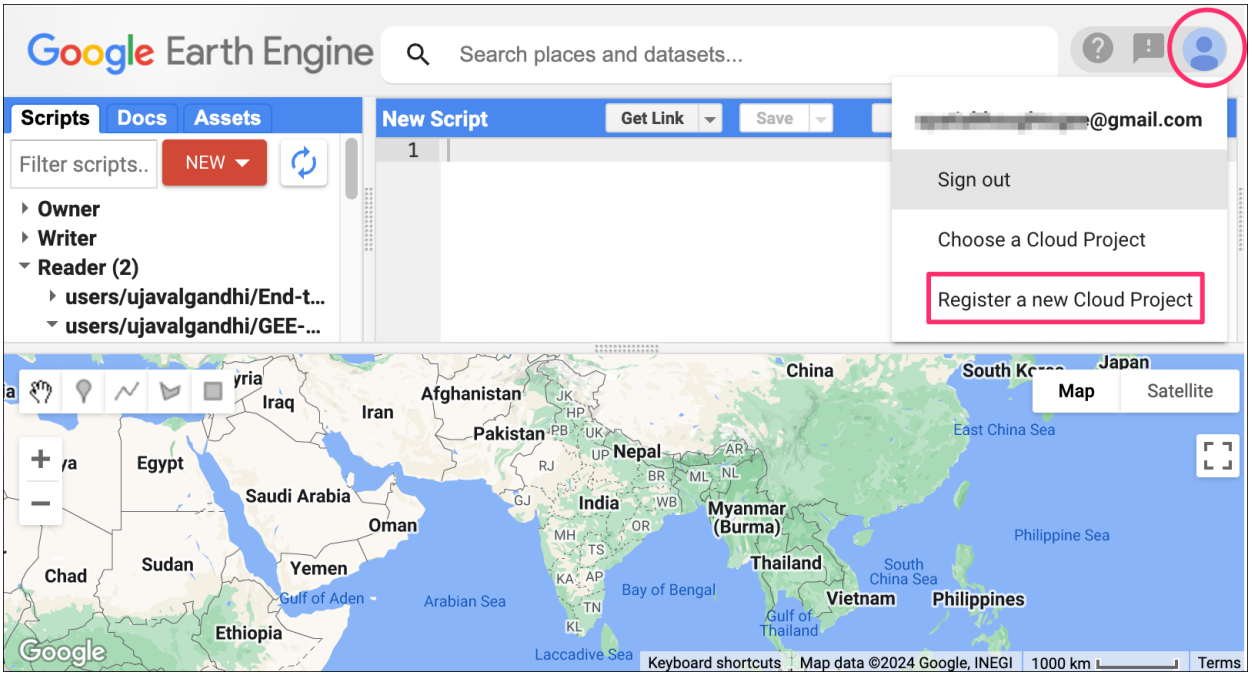
1. Open the Earth Engine Code Editor:
   Google Earth Engine Code Editor
2. Click the **account/profile icon** located in the **top-right corner**.
3. Select **"Register a new Cloud Project"**.
4. Follow the registration flow to connect your existing Earth Engine account with a Google Cloud Project.
5. Complete the setup process and start using Google Earth Engine with the linked Cloud Project.

◆ **Important Notes**

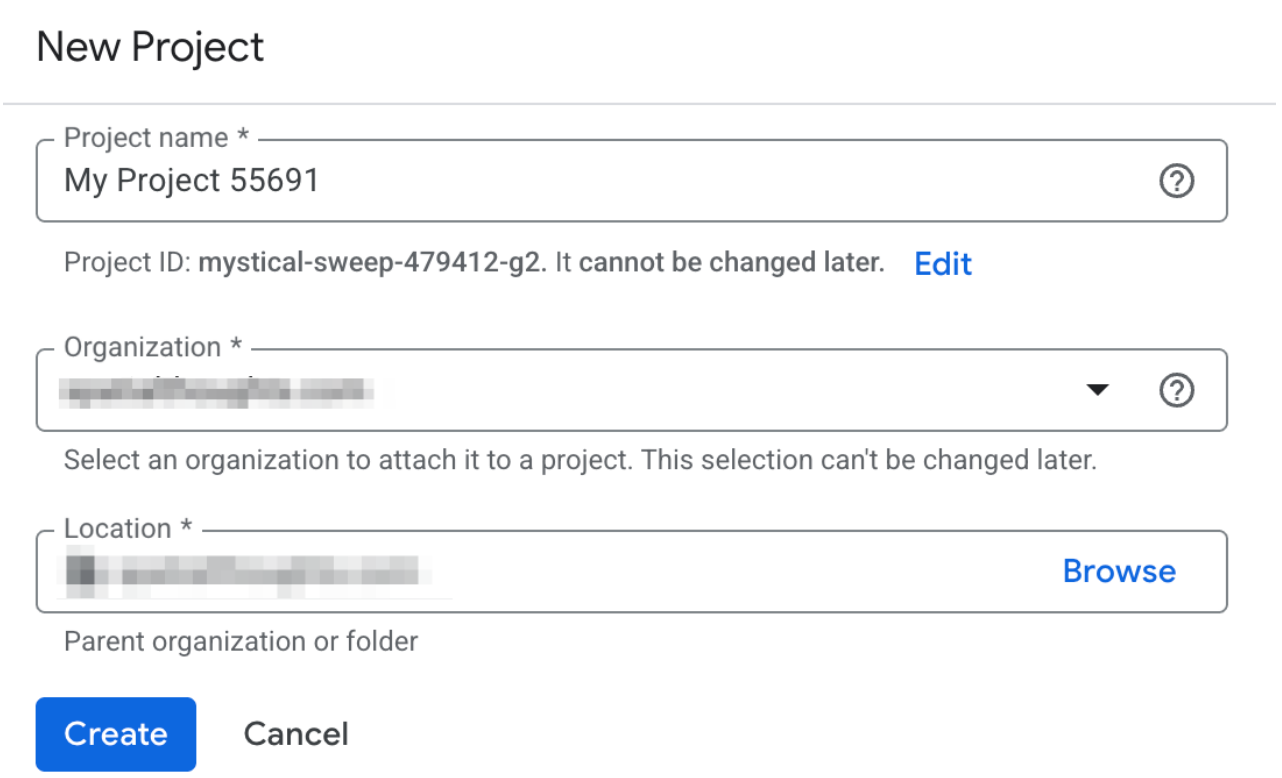
- A **Google account** is required.
- A **Google Cloud Project** is now mandatory for using Earth Engine.
- Approval may take some time depending on Google's verification process.
- Always use the same Google account for both Earth Engine and Google Cloud Project management.

Useful Links

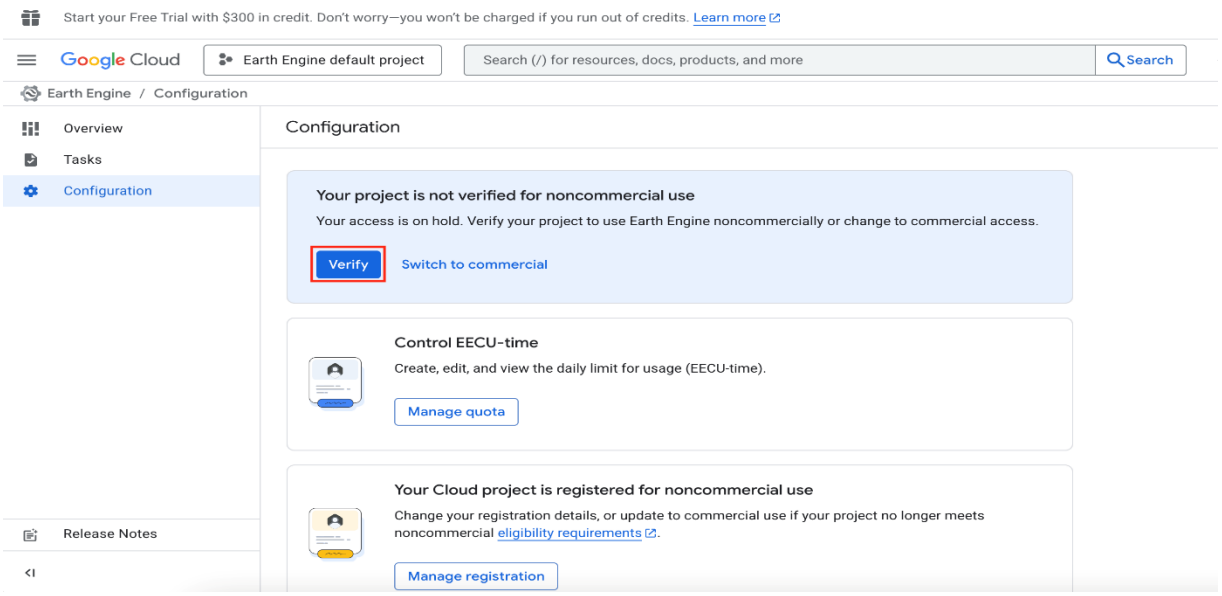
- https://earthengine.google.com/
- https://console.cloud.google.com/
- https://developers.google.com/earth-engine/



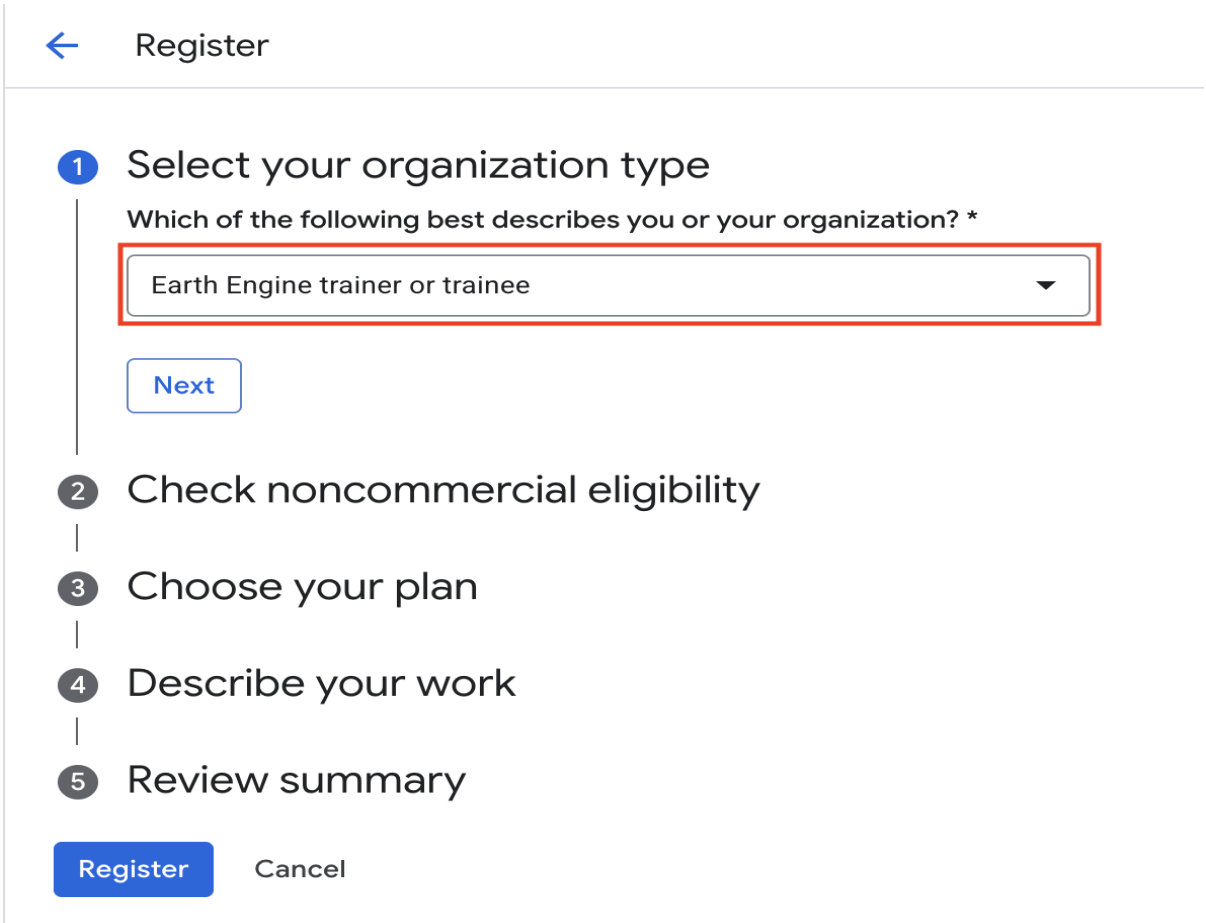
Sign-up Flow



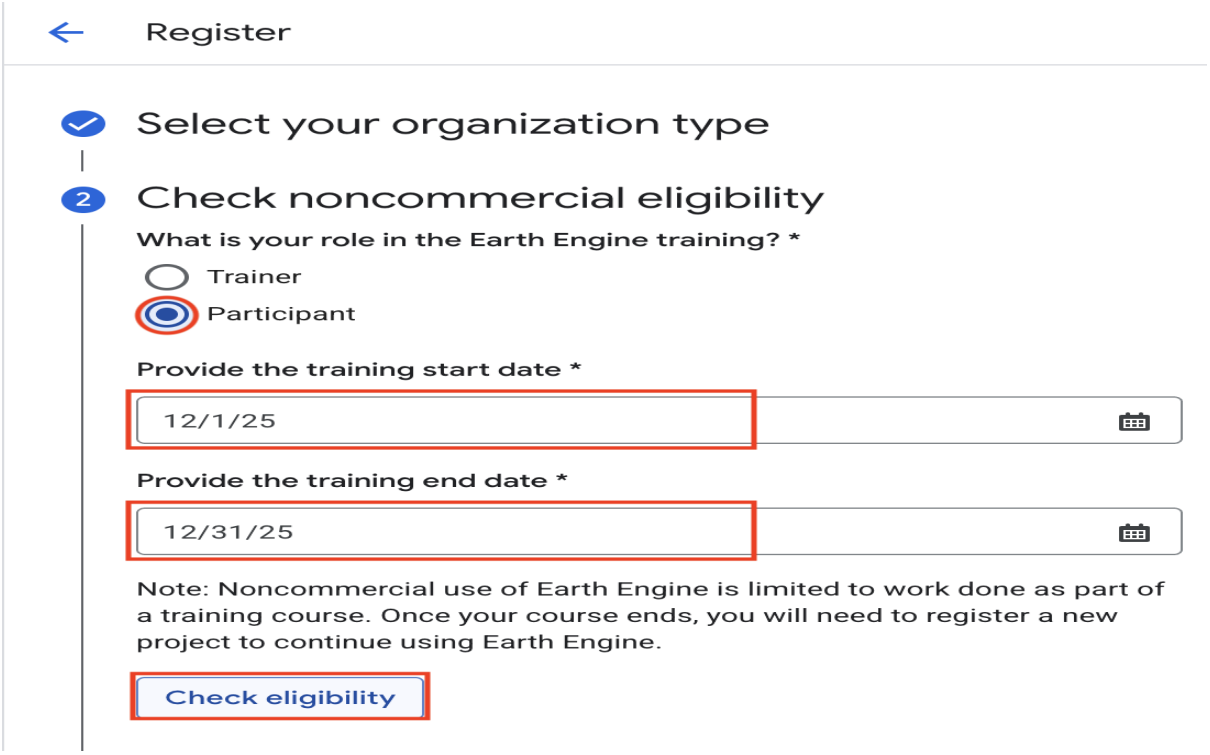
2. On the *Configuration* page, click on Verify to proceed with the registration.



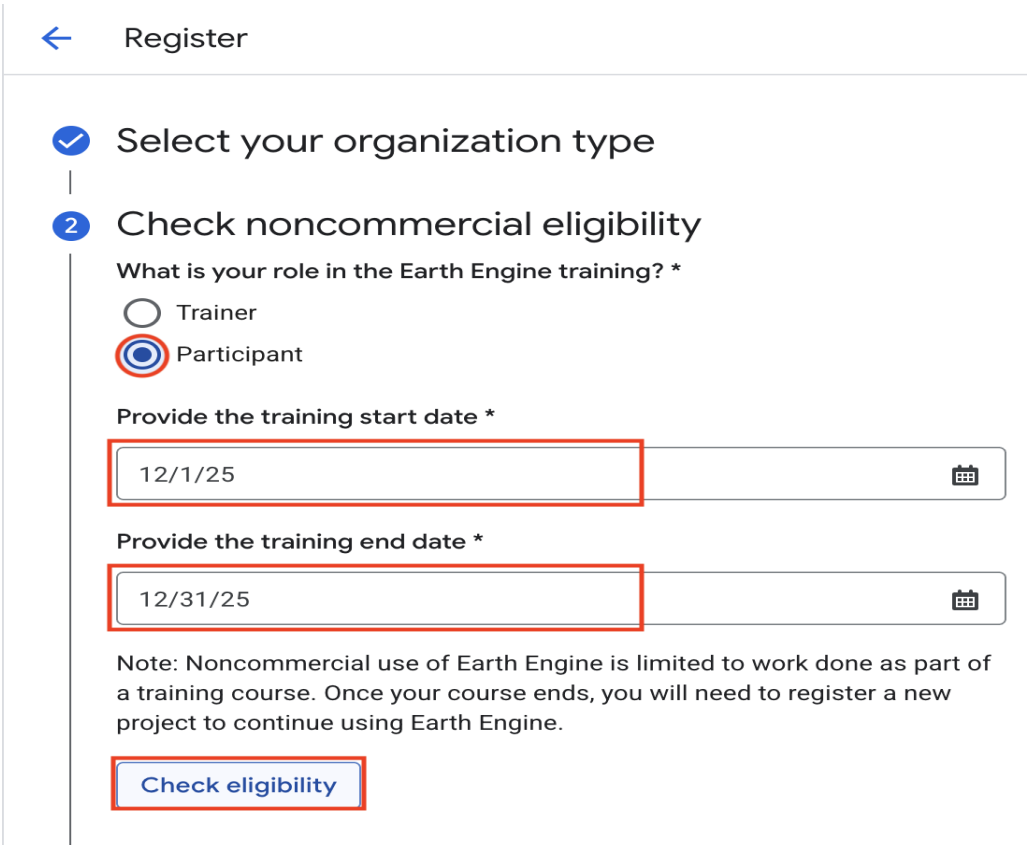
3. In the first configuration step, *Select your organization type* as Earth Engine trainer or trainee and click *Next*.



4. Next, you are at the step to *Check noncommercial eligibility*. Choose your role as Participant and provide start and end dates for the course. Click on on *Check eligibility*.

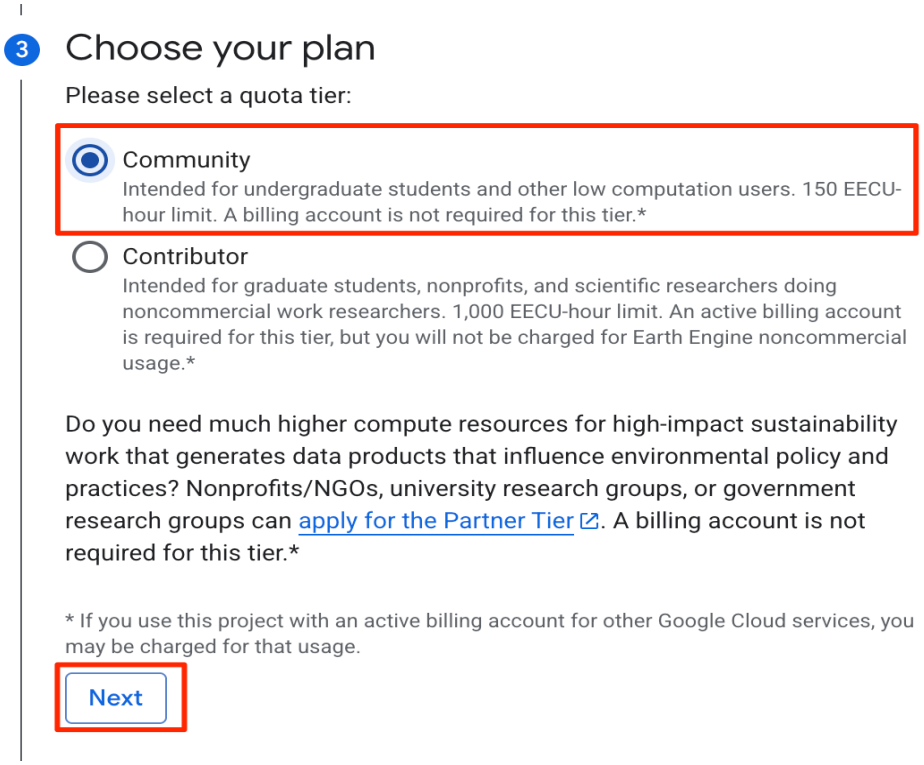


5. You will see the note that stating that you are eligible for non commercial Earth Engine. Click *Next* to proceed.

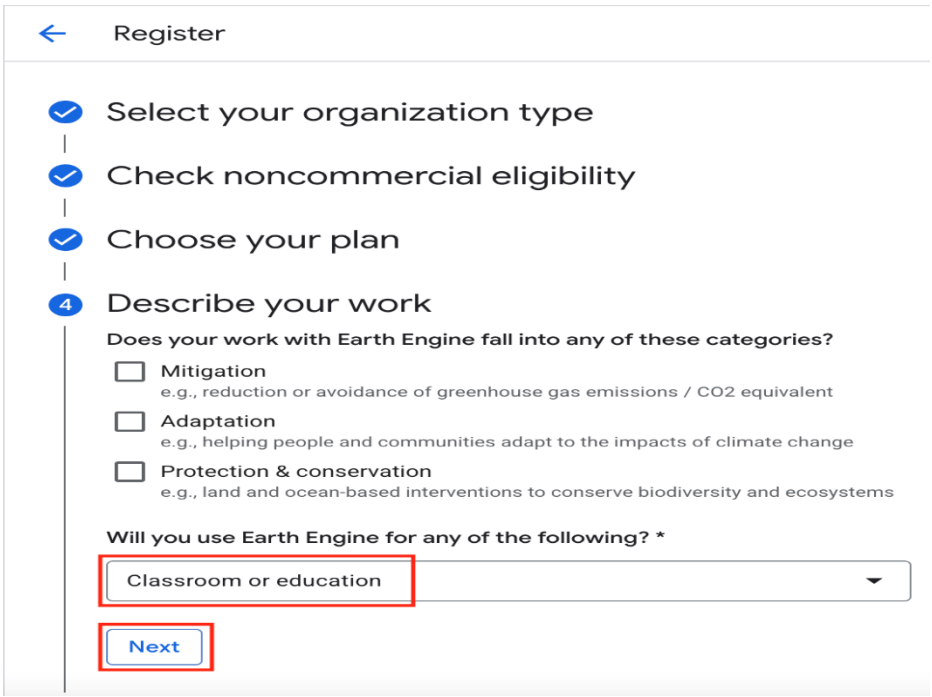


6. Under the *Choose your plan* section, you will be asked to select a non-commercial quota tier. For new users of Earth Engine and participants of our courses, it is recommended to

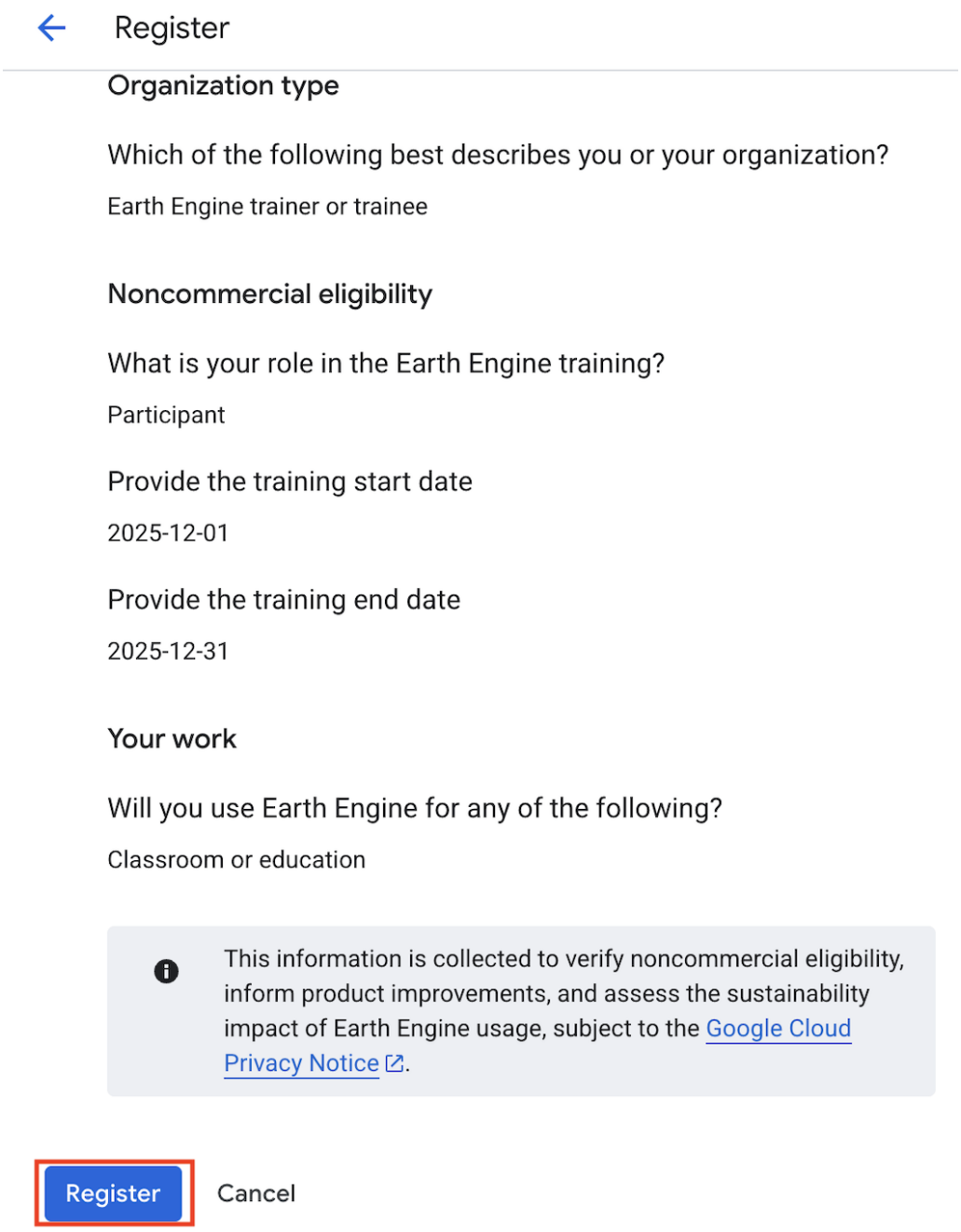
select the Community tier. This will help you get started with no additional configuration and allows you to explore the platform. If you intend to use Earth Engine for large computations, you can later update your project to the Contributor tier. This requires you to setup a billing account to verify your identity and you get higher quota. However, note that setting up a Cloud Billing Account can be tricky and the process depends on your location and available payment methods. Click *Next* and proceed.



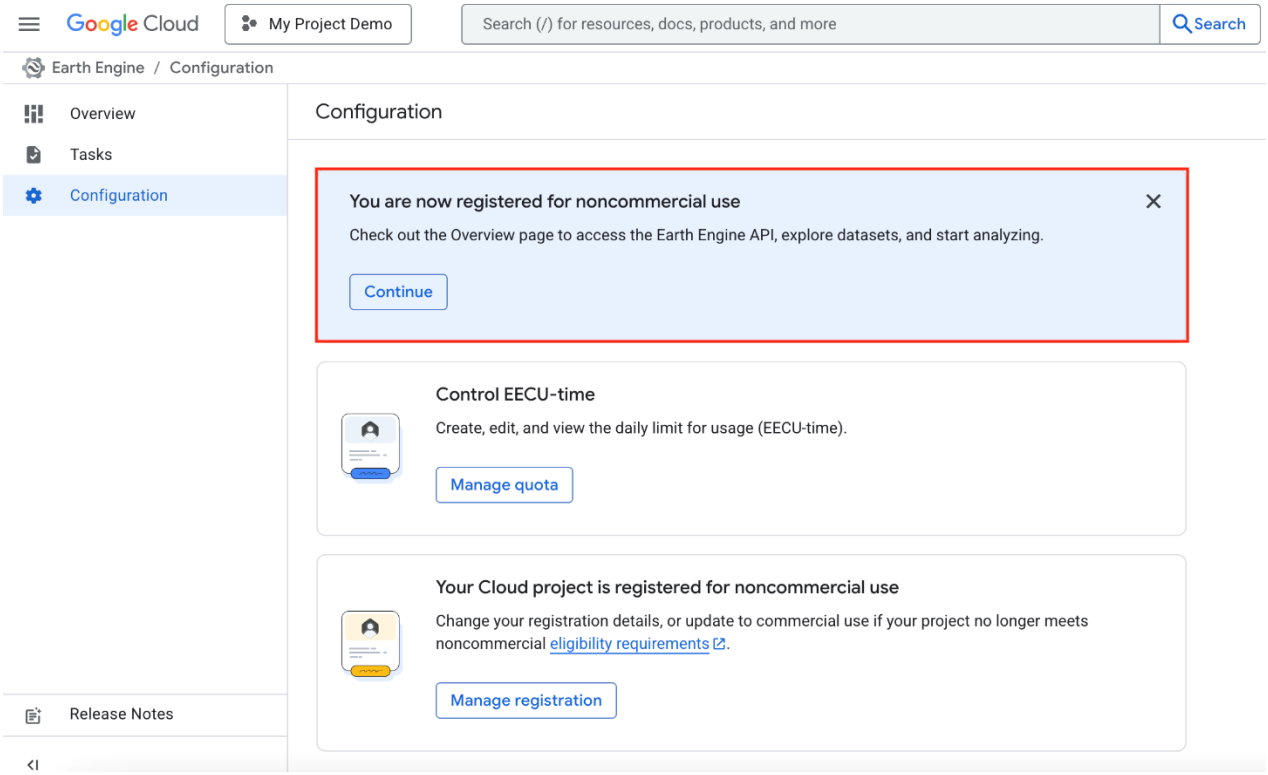
7. To *Describe your work*, choose Classroom or education in the drop-down and click *Next*.



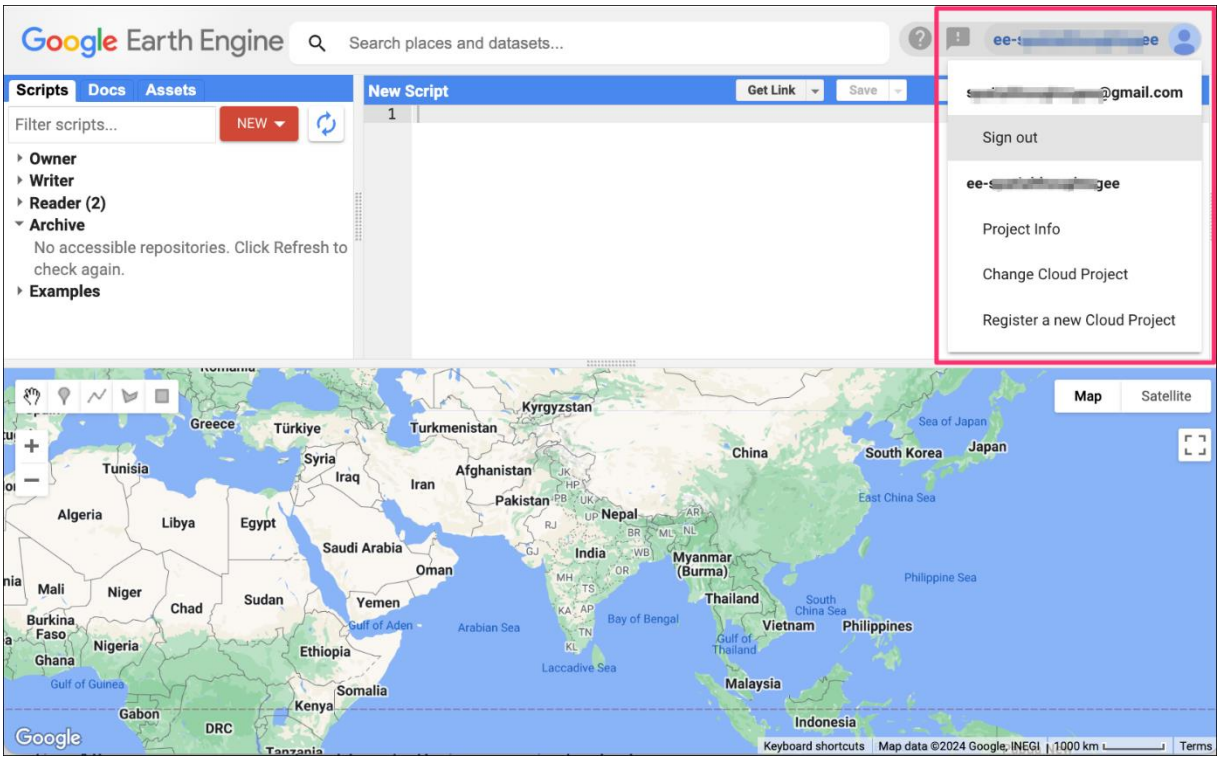
8. The final step is to *Review the summery* and click on *Register* to complete the registration process.



9. After completing the registration process, you will see the conformation that you are registered for the noncommercial use. Continue to be redirected to the API page.



10. Visit the Earth Engine Code Editor. Your code editor will now display information about the linked project.



Programming: Javascript Basics in GEE

Mr. Al-Helal
Programmer (Computer & GIS Unit)
Bangladesh Agricultural Research Council
Email: al.helal@barc.gov.bd

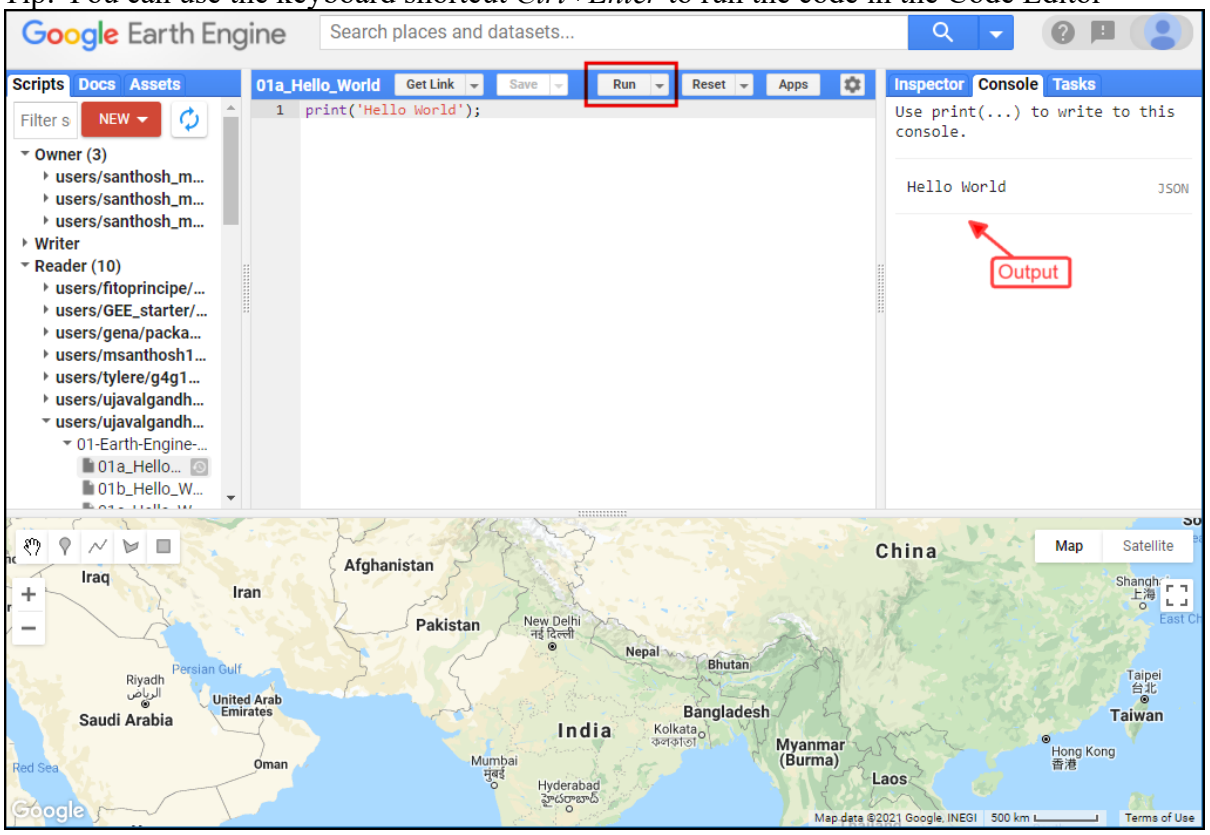
Module 1 is designed to give you basic skills to be able to find datasets you need for your project, filter them to your region of interest, apply basic processing and export the results. Mastering this will allow you to start using Earth Engine for your project quickly and save a lot of time pre-processing the data.

01. Hello World

This script introduces the basic Javascript syntax and the video covers the programming concepts you need to learn when using Earth Engine. To learn more, visit Introduction to JavaScript for Earth Engine section of the Earth Engine User Guide.

The *Code Editor* is an Integrated Development Environment (IDE) for Earth Engine Javascript API.. It offers an easy way to type, debug, run and manage code. Type the code below and click *Run* to execute it and see the output in the *Console* tab.

Tip: You can use the keyboard shortcut *Ctrl+Enter* to run the code in the Code Editor



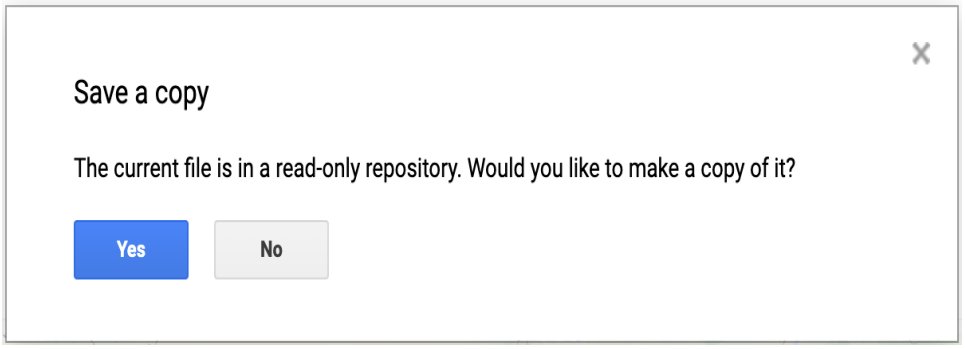
Hello World

```
// ========================================
// Google Earth Engine Basics (Bangladesh)
// Variables, Lists, Dictionary, Functions
// ========================================

// ----------------------------------------
// Print Hello World
// ----------------------------------------
print('Hello Bangladesh 🌿');

// ----------------------------------------
// Variables (Bangladesh Context)
// ----------------------------------------
var city = 'Dhaka';
var country = 'Bangladesh';

print('City & Country:', city, country);

var population = 23000000;  // Approx. Dhaka metro population
print('Population of Dhaka:', population);

// ----------------------------------------
// List of Major Cities in Bangladesh
// ----------------------------------------
var majorCities = [
  'Dhaka',
  'Chattogram',
  'Khulna',
  'Rajshahi',
  'Sylhet'
];

print('Major Cities of Bangladesh:', majorCities);

// ----------------------------------------
// Dictionary (City Data)
// ----------------------------------------
var cityData = {
  city: city,
  country: country,
  population: population,
  elevation: 4  // approx. elevation of Dhaka (meters)
};

print('City Data (Bangladesh):', cityData);

// ----------------------------------------
// Function Example
```

```
// ----------------------------------------
var greet = function(name) {
  return 'Hello ' + name + ' from Bangladesh BD';
};

print(greet('World'));

// ----------------------------------------
// Comment Example
// ----------------------------------------
// This is a simple GEE introductory script for Bangladesh
```

Saving Your Work

When you modify any script for the course repository, you may want to save a copy for yourself. If you try to click the *Save* button, you will get an error message like below

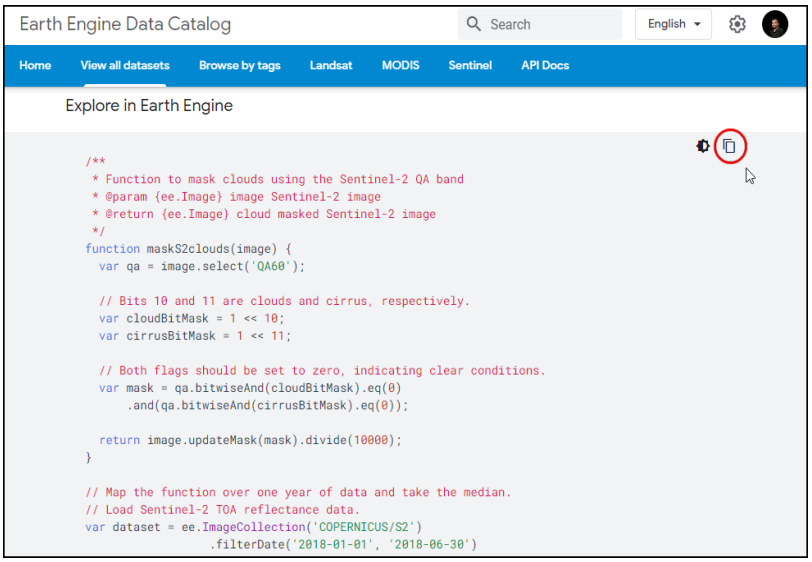


This is because the shared class repository is a *Read-only* repository. You can click *Yes* to save a copy in your repository. If this is the first time you are using Earth Engine, you will be prompted to choose a *Earth Engine username*. Choose the name carefully, as it cannot be changed once created.

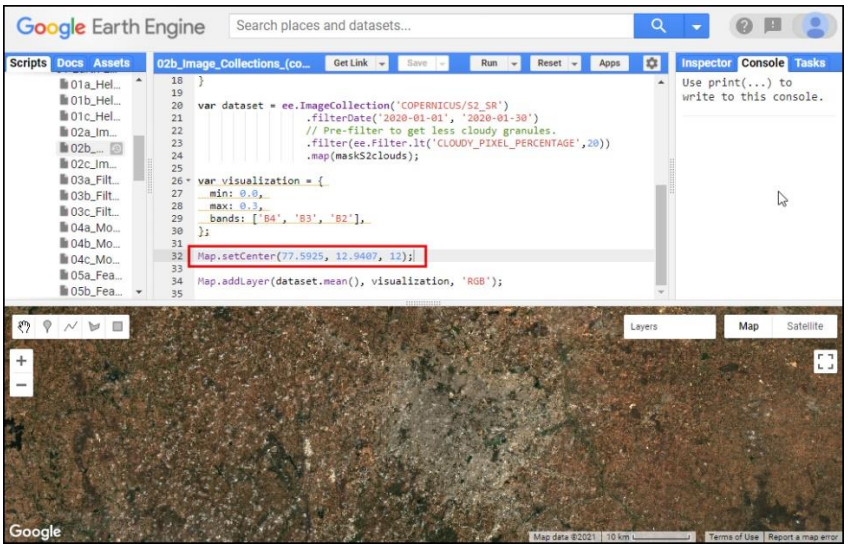
After entering your username, your home folder will be created. After that, you will be prompted to enter a new repository. A repository can help you organize and share code. Your account can have multiple repositories and each repository can have multiple scripts inside it. To get started, you can create a repository named *default*. Finally, you will be able to save the script.

02. Working with Image Collections

Most datasets in Earth Engine come as a ImageCollection. An ImageCollection is a dataset that consists of images takes at different time and locations - usually from the same satellite or data provider. You can load a collection by searching the Earth Engine Data Catalog for the *ImageCollection ID*. Search for the *Sentinel-2 Level 1C* dataset and you will find its id COPERNICUS/S2_SR. Visit the Sentinel-2, Level 1C page and see *Explore in Earth Engine* section to find the code snippet to load and visualize the collection. This snippet is a great starting point for your work with this dataset. Click the **Copy Code Sample** button and paste the code into the code editor. Click *Run* and you will see the image tiles load in the map.



In the code snippet, You will see a function Map.setCenter() which sets the viewport to a specific location and zoom level. The function takes the X coordinate (longitude), Y coordinate (latitude) and Zoom Level parameters. Replace the X and Y coordinates with the coordinates of your city and click *Run* to see the images of your city.



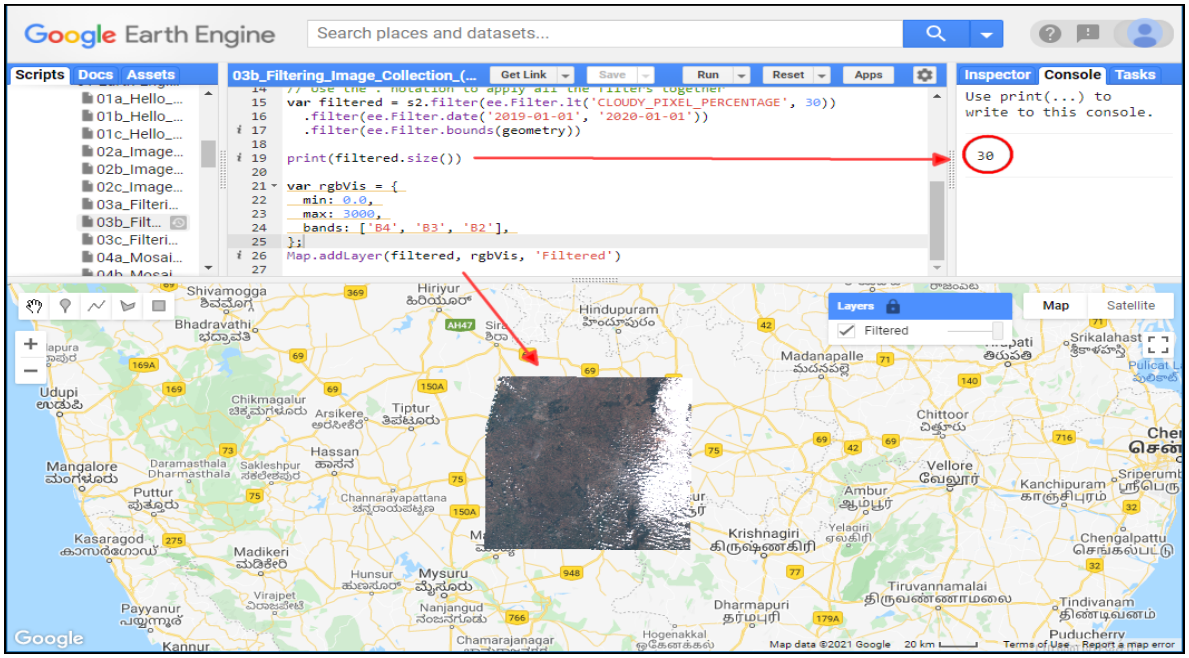
03. Filtering Image Collections

The collection contains all imagery ever collected by the sensor. The entire collections are not very useful. Most applications require a subset of the images. We use **filters** to select the appropriate images. There are many types of filter functions, look at ee.Filter... module to see all available filters. Select a filter and then run the filter() function with the filter parameters.

We will learn about 3 main types of filtering techniques

- **Filter by metadata**: You can apply a filter on the image metadata using filters such as ee.Filter.eq(), ee.Filter.lt() etc. You can filter by PATH/ROW values, Orbit number, Cloud cover etc.
- **Filter by date**: You can select images in a particular date range using filters such as ee.Filter.date().
- **Filter by location**: You can select the subset of images with a bounding box, location or geometry using the ee.Filter.bounds(). You can also use the drawing tools to draw a geometry for filtering.

After applying the filters, you can use the size() function to check how many images match the filters.




```
// Create a point geometry (Dhaka, Bangladesh)
var geometry = ee.Geometry.Point([90.4125, 23.8103]);

// Center the map on the selected location
Map.centerObject(geometry, 10);

// Load Sentinel-2 Harmonized Image Collection
var s2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED');

// ---------------------------------------
// Filter by Cloud Percentage
// ---------------------------------------
```

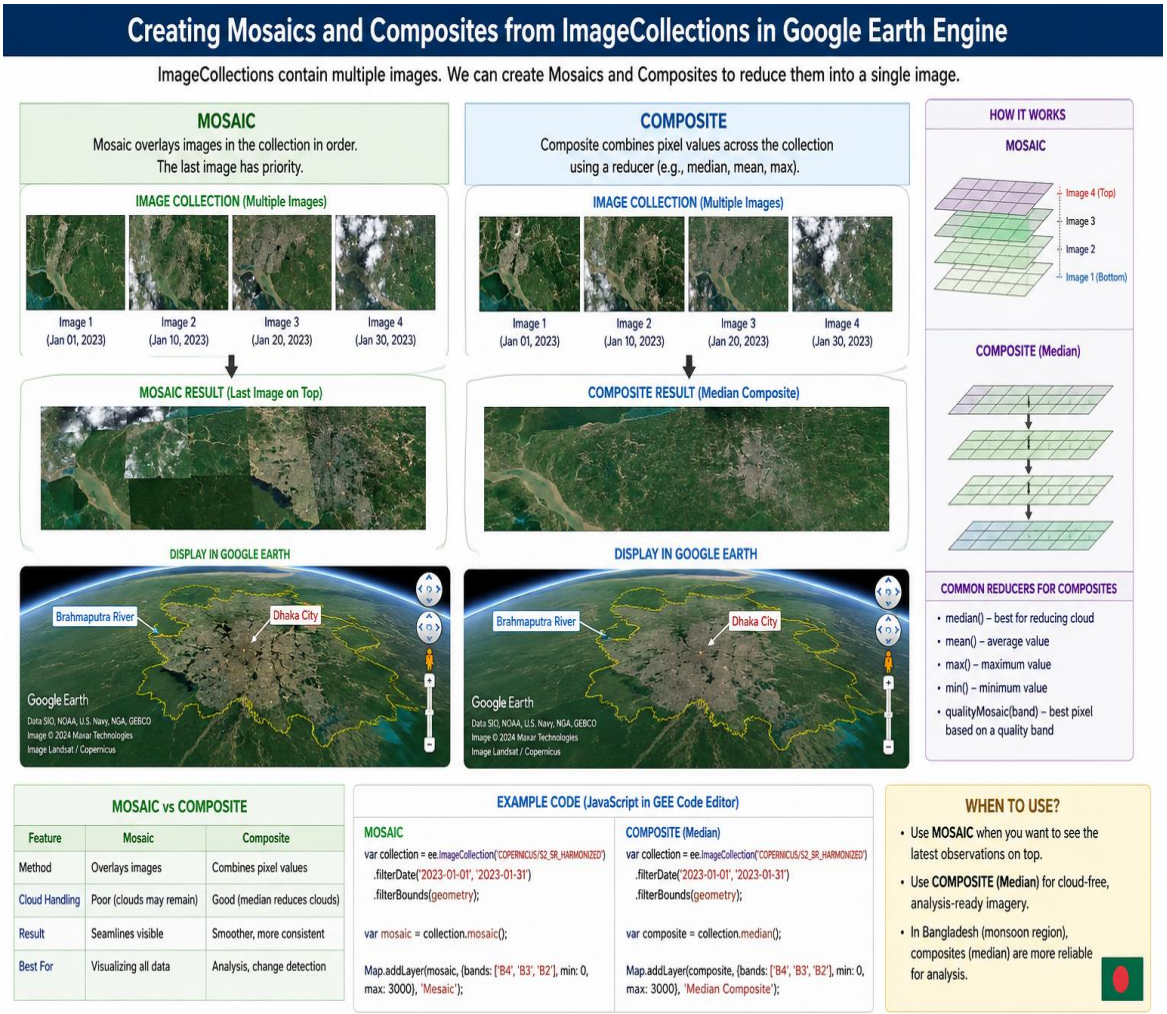
```
var filteredCloud = s2.filter(
  ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30)
);

// ----------------------------------------
// Filter by Date
// ----------------------------------------
var filteredDate = s2.filter(
  ee.Filter.date('2024-01-01', '2025-01-01')
);

// ----------------------------------------
// Filter by Location
// ----------------------------------------
var filteredLocation = s2.filter(
  ee.Filter.bounds(geometry)
);
// ============================================
// Apply All Filters Step-by-Step
// ============================================

// Step 1: Apply cloud filter
var filtered1 = s2.filter(
  ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30)
);

// Step 2: Apply date filter
var filtered2 = filtered1.filter(
  ee.Filter.date('2024-01-01', '2025-01-01')
);

// Step 3: Apply location filter
var filtered3 = filtered2.filter(
  ee.Filter.bounds(geometry)
);

// ============================================
// Apply All Filters Using Chaining
// ============================================

var filtered = s2
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30))
  .filter(ee.Filter.date('2024-01-01', '2025-01-01'))
  .filter(ee.Filter.bounds(geometry));

// Print total number of filtered images
print('Total Filtered Images:', filtered.size());
```

04. Creating Mosaics and Composites from ImageCollections

The default order of the collection is by date. So when you display the collection, it implicitly creates a mosaic with the latest pixels on top. You can call .mosaic() on a ImageCollection to create a mosaic image from the pixels at the top.

We can also create a composite image by applying selection criteria to each pixel from all pixels in the stack. Here we use the median() function to create a composite where each pixel value is the median of all pixels from the stack.

Tip: If you need to create a mosaic where the images are in a specific order, you can use the .sort() function to sort your collection by a property first.



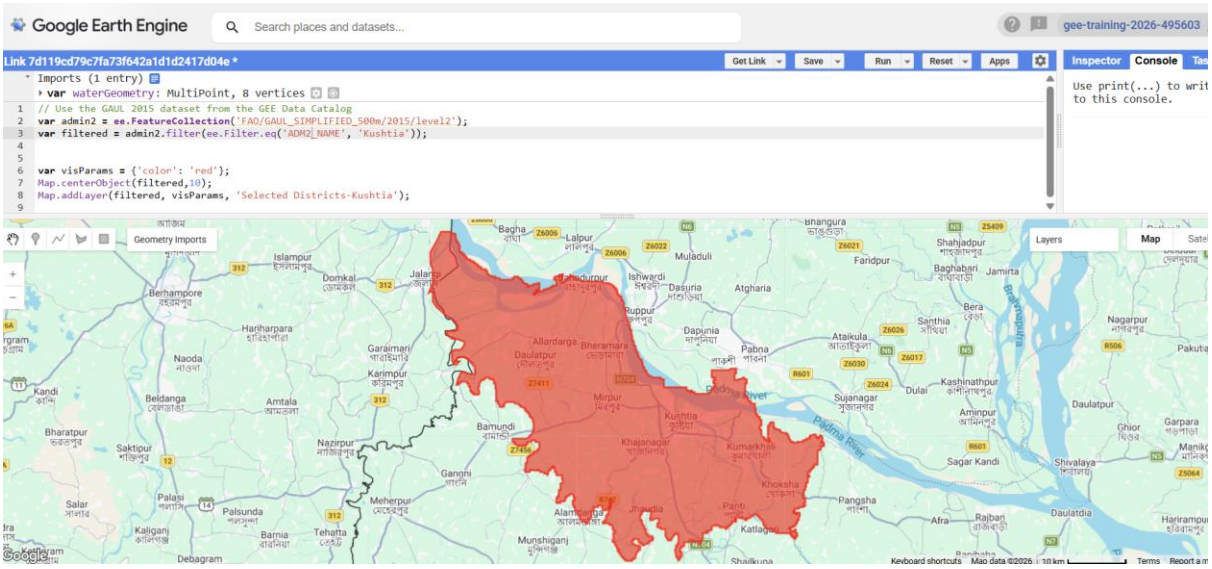
Mosaic vs. Composite

```
/**
 * Sentinel-2 Composite Analysis - Bangladesh
 * Targets a specific region (Dhaka) and creates a cloud-free median
composite.
 */
// 1. Define Point (Dhaka vicinity) and Country Boundary
var geometry = ee.Geometry.Point([90.4125, 23.8103]);
var bd_boundary = ee.FeatureCollection("FAO/GAUL/2015/level0")
                    .filter(ee.Filter.eq('ADM0_NAME', 'Bangladesh'));
// 2. Import Sentinel-2 Harmonized Collection
var s2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED');

// 3. Cloud Masking Function (Optional but recommended for better quality)
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);
}
// 4. Apply Filters
var filtered = s2.filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
  .filter(ee.Filter.date('2022-01-01', '2023-01-01')) // Updated to a more
recent year
  .filter(ee.Filter.bounds(geometry));

// 5. Create Composites
var mosaic = filtered.mosaic().clip(bd_boundary);
var medianComposite = filtered.median().clip(bd_boundary);

// 6. Visualization Parameters
// Adjusting max to 0.3 because we divided by 10000 in the mask function
// If not using the mask function, use 3000
var rgbVis = {
  min: 0.0,
  max: 3000,
  bands: ['B4', 'B3', 'B2'],
};
// 7. Map Display
Map.centerObject(geometry, 11);
Map.addLayer(bd_boundary, {color: 'grey'}, 'Bangladesh Border', false);
Map.addLayer(filtered, rgbVis, 'Filtered Collection (Raw)', false);
Map.addLayer(mosaic, rgbVis, 'Mosaic (Last Pixel on Top)');
Map.addLayer(medianComposite, rgbVis, 'Median Composite (Cloud Free-ish)');
// 8. Print stats
print('Number of images in collection:', filtered.size());
```

05. Working with Feature Collections

Feature Collections are similar to Image Collections - but they contain *Features*, not images. They are equivalent to Vector Layers in a GIS. We can load, filter and display Feature Collections using similar techniques that we have learned so far.

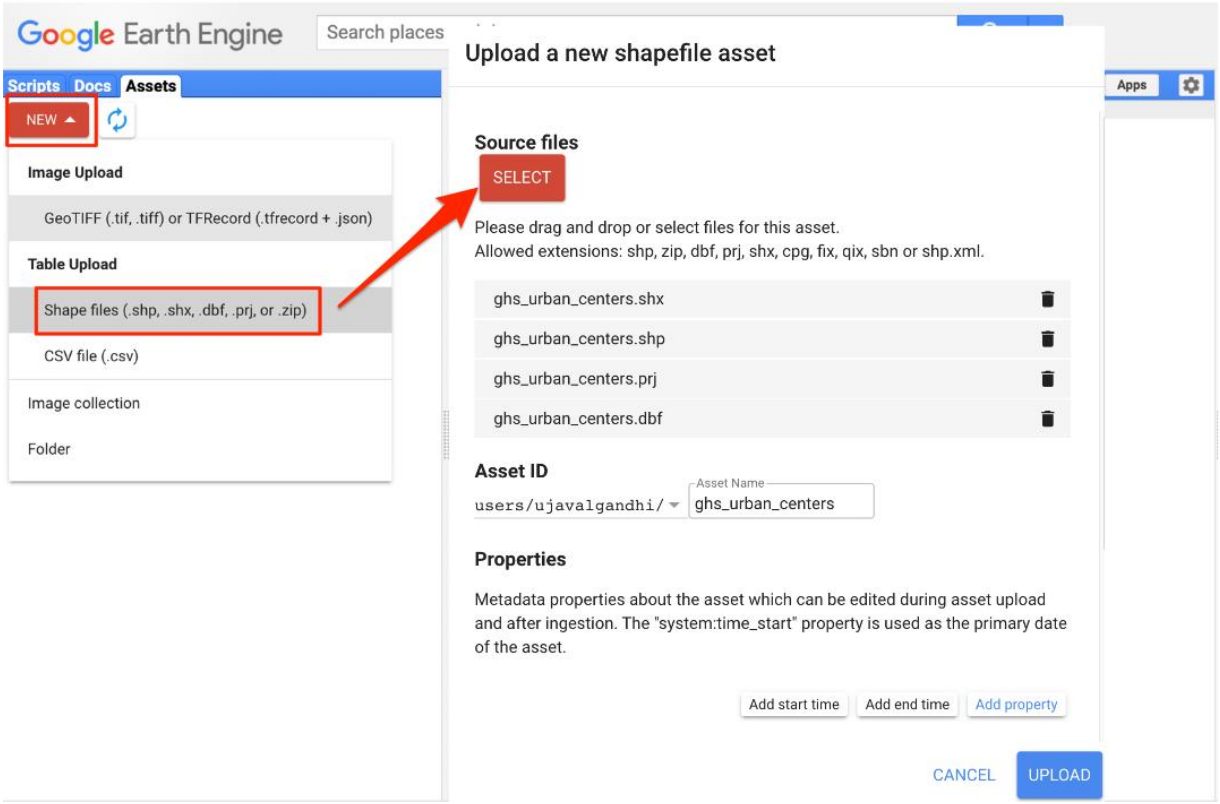
Search for *GAUL Second Level Administrative Boundaries* and load the collection. This is a global collection that contains all Admin2 boundaries. We can apply a filter using the ADM1_NAME property to get all Admin2 boundaries (i.e. Districts) from a state.




```
// Use the GAUL 2015 dataset from the GEE Data Catalog
var admin2 =
ee.FeatureCollection('FAO/GAUL_SIMPLIFIED_500m/2015/level2');
var filtered = admin2.filter(ee.Filter.eq('ADM2_NAME',
'Kushtia'));


var visParams = {'color': 'red'};
Map.centerObject(filtered,10);
Map.addLayer(filtered, visParams, 'Selected Districts-Kushtia');
```

06. Importing Data

You can import vector or raster data into Earth Engine. We will now import a shapefile of Urban Centres from JRC's GHS Urban Centre Database (GHS-UCDB). Unzip the ghs_urban_centers.zip into a folder on your computer. In the Code Editor, go to *Assets* → *New* → *Table Upload* → *Shape Files*. Select the .shp, .shx, .dbf and .prj files. Enter ghs_urban_centers as the *Asset Name* and click *Upload*. Once the upload and ingest finishes, you will have a new asset in the *Assets* tab. The shapefile is imported as a Feature Collection in Earth Engine. Select the ghs_urban_centers asset and click *Import*. You can then visualize the imported data.



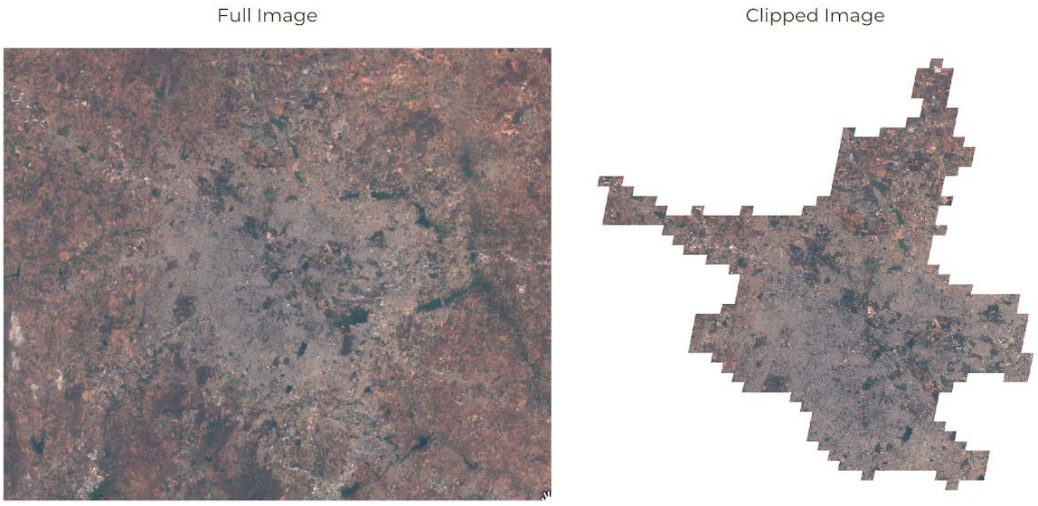
Importing a Shapefile

```
// 1. Define Bangladesh Boundary (ROI)
var roi = ee.FeatureCollection("FAO/GAUL/2015/level0")
            .filter(ee.Filter.eq('ADM0_NAME', 'Bangladesh'));
// 2. Import the GHS Urban Centers collection
// Note: Ensure the asset path matches your uploaded location
var urban =
ee.FeatureCollection('users/ujavalgandhi/e2e/ghs_urban_centers');
// 3. Filter for Urban Centers only within Bangladesh
var bd_urban = urban.filterBounds(roi);

// 4. Visualize the results
// Styling: Blue for urban footprints
Map.centerObject(roi, 7);
Map.addLayer(roi, {color: 'grey'}, 'Bangladesh Boundary', false);
Map.addLayer(bd_urban, {color: 'blue'}, 'Urban Areas (Bangladesh)');

// 5. Inspect counts in Console
print('Number of Urban Centers in Bangladesh:', bd_urban.size());
print('Attribute Table (First 5):', bd_urban.limit(5));
```

07. Clipping Images

It is often desirable to clip the images to your area of interest. You can use the clip() function to mask out an image using a geometry.

While in a Desktop software, clipping is desirable to remove unnecessary portion of a large image and save computation time, in Earth Engine clipping can actually increase the computation time. As described in the Earth Engine Coding Best Practices guide, avoid clipping the images or do it at the end of your script.

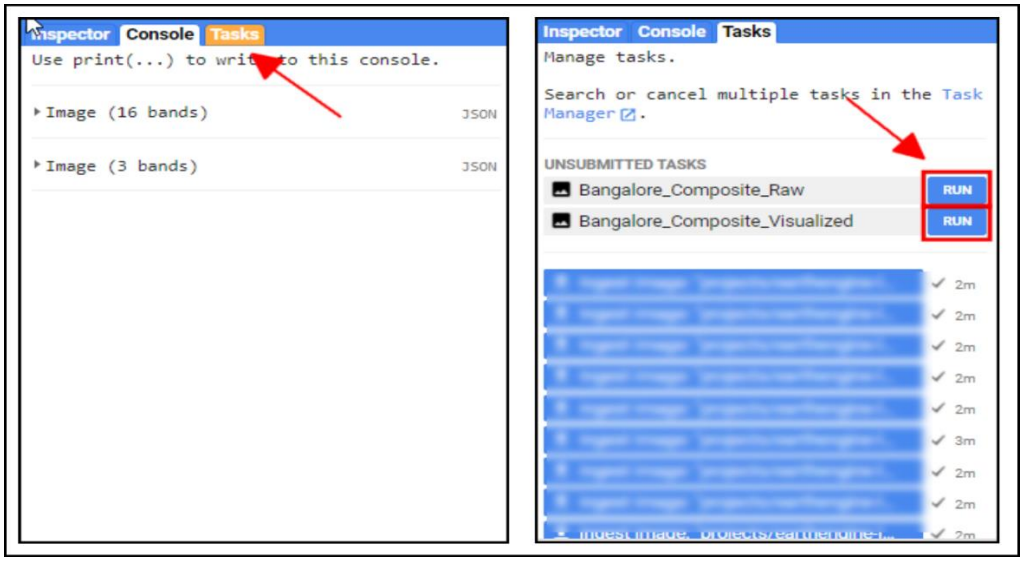


Original vs. Clipped Image

08. Exporting Data

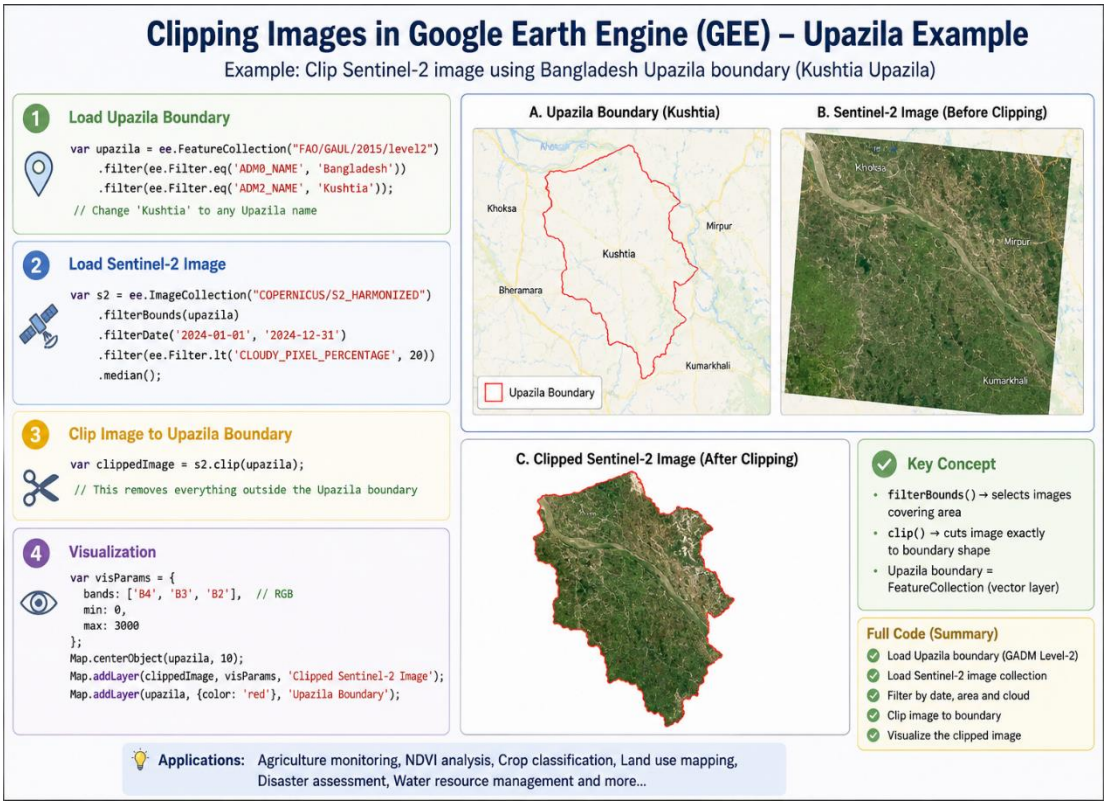
Earth Engine allows for exporting both vector and raster data to be used in an external program. Vector data can be exported as a CSV or a Shapefile, while Rasters can be exported as GeoTIFF files. We will now export the Sentinel-2 Composite as a GeoTIFF file.

Tip: Code Editor supports autocompletion of API functions using the combination *Ctrl+Space*. Type a few characters of a function and press *Ctrl+Space* to see autocomplete suggestions. You can also use the same key combination to fill all parameters of the function automatically. Once you run this script, the *Tasks* tab will be highlighted. Switch to the tab and you will see the tasks waiting. Click *Run* next to each task to start the process.



On clicking the *Run* button, you will be prompted for a confirmation dialog. Verify the settings and click *Run* to start the export.

Once the Export finishes, a GeoTiff file for each export task will be added to your Google Drive in the specified folder. You can download them and use it in a GIS software.



Visualized vs. Raw Composite

```
// Bangladesh Upazila Boundary
var upazila = ee.FeatureCollection("FAO/GAUL/2015/level2")
  .filter(ee.Filter.eq('ADM0_NAME', 'Bangladesh'))
  .filter(ee.Filter.eq('ADM2_NAME', 'Kushtia'));

// Sentinel-2 Image Collection
var s2 = ee.ImageCollection("COPERNICUS/S2_HARMONIZED")
  .filterBounds(upazila)
  .filterDate('2024-01-01', '2024-12-31')
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
  .median();

// Clip image
var clippedImage = s2.clip(upazila);

// Visualization
var visParams = {
  bands: ['B4', 'B3', 'B2'],
  min: 0,
  max: 3000
};

Map.centerObject(upazila, 10);
Map.addLayer(clippedImage, visParams, 'Clipped Image');
Map.addLayer(upazila, {}, 'Upazila Boundary');
```

Google Earth Engine: Functions in GEE

Hasan Mahmud
Senior System Analyst (Computer & GIS Unit)
Bangladesh Agricultural Research Council
Email: hasan.mahmud@barc.gov.bd

Module 2 builds on the basic Earth Engine skills you have gained. This model introduces the parallel programming concepts using Map/Reduce - which is key in effectively using Earth Engine for analyzing large volumes of data. You will learn how to use the Earth Engine API for calculating various spectral indices, do cloud masking and then use map/reduce to do apply these computations to collections of imagery. You will also learn how to take long time-series of data and create charts.

01. Earth Engine Objects

This script introduces the basics of the Earth Engine API. When programming in Earth Engine, you must use the Earth Engine API so that your computations can use the Google Earth Engine servers. To learn more, visit Earth Engine Objects and Methods section of the Earth Engine User Guide.

```
/**
 * Earth Engine Logic: Advanced Lists, Dictionaries, and Dates
 * Demonstrates server-side mapping, type casting, and data manipulation.
 */

// --- 1. ADVANCED LIST OPERATIONS ---
// Sequence of numbers from 1 to 20
var myList = ee.List.sequence(1, 20);

// Define a function for server-side processing
// We use ee.Number methods to ensure it runs on Google's servers
var processNumbers = function(n) {
  var num = ee.Number(n);
  return num.pow(2).round(); // Square the number and round it
};

var squaredList = myList.map(processNumbers);
print('Squared List:', squaredList);

// Extract and Cast
// get() returns a generic Object; we must cast to ee.Number to use .add()
var fifthElement = ee.Number(squaredList.get(4));
print('5th Element + 100:', fifthElement.add(100));


// --- 2. DICTIONARIES & DATA EXTRACTION ---
var cityStats = ee.Dictionary({
  'city': 'Dhaka',
```

```
  'population_2021': 21741000,
  'area_sqkm': 306,
  'country': 'Bangladesh'
});

// Accessing keys and values
var keys = cityStats.keys();
var pop = ee.Number(cityStats.get('population_2021'));

// Calculate density (server-side)
var density = pop.divide(ee.Number(cityStats.get('area_sqkm')));
print('Population Density (per sq km):', density);


// --- 3. DATES & TIME SERIES LOGIC ---
// Creating a date object
var start = ee.Date('2023-01-01');

// Advancing time and calculating differences
var end = start.advance(6, 'month');
var diffDays = end.difference(start, 'days');

print('Start Date:', start);
print('End Date:', end);
print('Days between:', diffDays);

// --- 4. PRACTICAL APPLICATION: Date Mapping ---
// Generate a list of dates for the last 5 years
var years = ee.List.sequence(2018, 2023);

var makeDateRange = function(year) {
  var date = ee.Date.fromYMD(year, 1, 1);
  return date.format('YYYY-MM-dd');
};

var formattedDates = years.map(makeDateRange);
print('Formatted Annual Dates:', formattedDates);
```

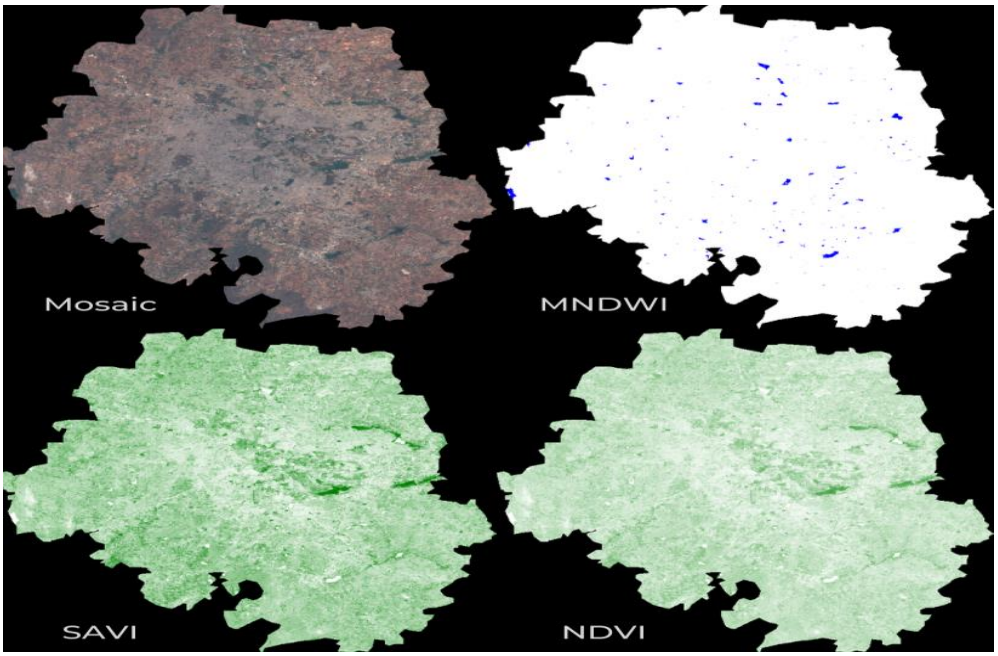
As a general rule, you should always use Earth Engine API methods in your code, there is one exception where you will need to use client-side Javascript method. If you want to get the current time, the server doesn't know your time. You need to use javascript method and cast it to an Earth Engine object.

```
// 1. Client-side: Get current timestamp in milliseconds using standard JS
// This runs in your browser immediately.
var now = Date.now();
print('Browser Timestamp (ms):', now);
```

```
// 2. Server-side: Convert that timestamp into an Earth Engine Date object
// This sends the number to Google's servers to create an ee.Date
var eeNow = ee.Date(now);
print('EE Date Object:', eeNow);

// 3. Proper GEE Way: Get "Now" directly from the server
// This is better for scripts running in automated environments
var serverNow = ee.Date(new Date().getTime());
// Or simply:
var simpleNow = ee.Date(Date.now());
print('Server-verified Now:', simpleNow);

// 4. Formatting the date for humans
var formatted = simpleNow.format('YYYY-MM-dd HH:mm:ss');
print('Readable Date:', formatted);

// 5. Practical Use Case: Filter a collection to the last 30 days
var thirtyDaysAgo = simpleNow.advance(-30, 'day');
print('30 Days Ago:', thirtyDaysAgo);
```

02. Calculating Indices

Spectral Indices are central to many aspects of remote sensing. Whether you are studying vegetation or tracking fires - you will need to compute a pixel-wise ratio of 2 or more bands. The most commonly used formula for calculating an index is the *Normalized Differenc e* between 2 bands. Earth Engine provides a helper function normalized Difference () to help calculate normalized indices, such as Normalized Difference Vegetation Index (NDVI). For more complex formulae, you can also use the expression() function to describe the calculation.



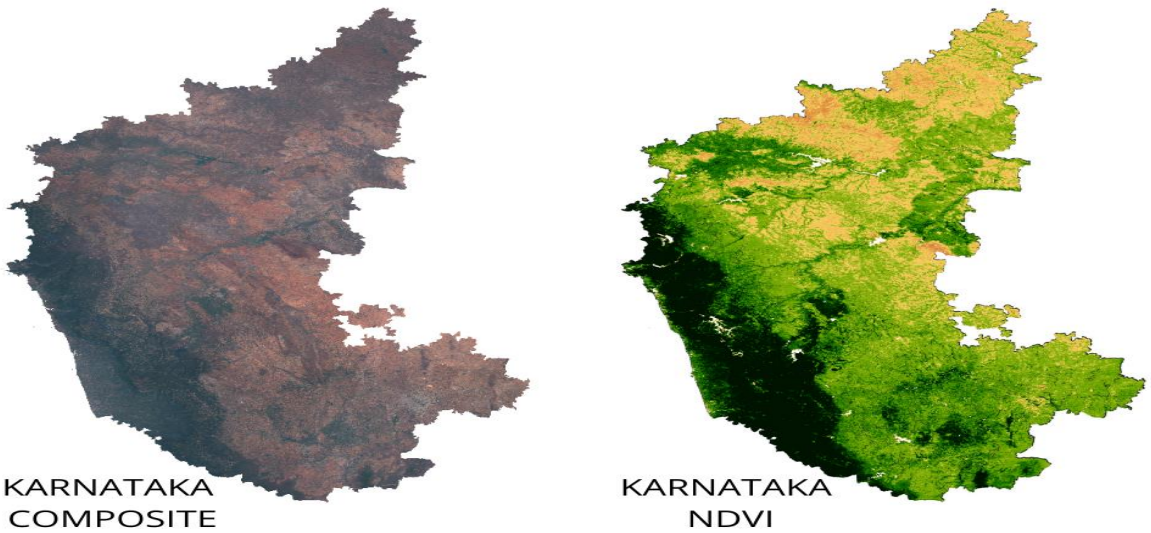
MNDWI, SAVI and NDVI images

```
/**
 * Sentinel-2 Multi-Index Analysis for Bangalore Urban
 * Includes: NDVI, MNDWI, SAVI, and NDBI
 */

var s2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED');
var admin2 = ee.FeatureCollection('FAO/GAUL_SIMPLIFIED_500m/2015/level2');

// 1. Define Region of Interest (ROI)
var filteredAdmin2 = admin2
  .filter(ee.Filter.eq('ADM2_NAME', 'Bangalore Urban'))
  .filter(ee.Filter.eq('ADM1_NAME', 'Karnataka'));

var geometry = filteredAdmin2.geometry();
Map.centerObject(geometry, 11);

// 2. Filter Image Collection
var filteredS2 = s2
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30))
  .filter(ee.Filter.date('2019-01-01', '2020-01-01'))
  .filter(ee.Filter.bounds(geometry));

// Create a median composite and clip to geometry
var image = filteredS2.median().clip(geometry);

// 3. Calculate Indices

// NDVI: Normalized Difference Vegetation Index
var ndvi = image.normalizedDifference(['B8', 'B4']).rename('ndvi');

// MNDWI: Modified Normalized Difference Water Index
// (Green - SWIR1) / (Green + SWIR1)
var mndwi = image.normalizedDifference(['B3', 'B11']).rename('mndwi');

// SAVI: Soil-adjusted Vegetation Index
// Requires reflectance scaling (0.0001) for correct L-factor math
var savi = image.expression(
    '1.5 * ((NIR - RED) / (NIR + RED + 0.5))', {
      'NIR': image.select('B8').multiply(0.0001),
      'RED': image.select('B4').multiply(0.0001),
}).rename('savi');

// NDBI: Normalized Difference Built-up Index
// (SWIR1 - NIR) / (SWIR1 + NIR) -> Good for Bangalore's urban sprawl
var ndbi = image.normalizedDifference(['B11', 'B8']).rename('ndbi');

// 4. Visualization Parameters
var rgbVis = {min: 0.0, max: 3000, bands: ['B4', 'B3', 'B2']};
```

```
var ndviVis = {min: 0, max: 0.8, palette: ['#FFFFFF', '#CE7E45', '#DF923D',
'#F1B555', '#FCD163', '#99B718', '#74A901', '#66A000', '#529400', '#3E8601',
'#207401', '#056201', '#004C00', '#023B01', '#012E01', '#011D01',
'#011301']};
var waterVis = {min: -0.5, max: 0.5, palette: ['white', 'blue']};
var builtVis = {min: -0.5, max: 0.5, palette: ['white', 'red']};

// 5. Add Layers to Map
Map.addLayer(image, rgbVis, 'True Color (RGB)');
Map.addLayer(ndvi, ndviVis, 'Vegetation (NDVI)');
Map.addLayer(savi, ndviVis, 'Soil Adjusted Veg (SAVI)');
Map.addLayer(mndwi, waterVis, 'Water Index (MNDWI)');
Map.addLayer(ndbi, builtVis, 'Built-up Index (NDBI)');

// Outline the boundary for clarity
var outline = ee.Image().paint(filteredAdmin2, 0, 2);
Map.addLayer(outline, {palette: 'black'}, 'Bangalore Boundary');
```

03. Computation on ImageCollections

So far we have learnt how to run computation on single images. If you want to apply some computation - such as calculating an index - to many images, you need to use map(). You first define a function that takes 1 image and returns the result of the computation on that image. Then you can map() that function over the ImageCollection which results in a new ImageCollection with the results of the computation. This is similar to a *for-loop* that you maybe familiar with - but using map() allows the computation to run in parallel. Learn more at Mapping over an ImageCollection



NDVI computation on an ImageCollection

```
var s2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED');

// Bangladesh boundary (Level 0 = country level)
var country = ee.FeatureCollection('FAO/GAUL_SIMPLIFIED_500m/2015/level0');
var bangladesh = country.filter(ee.Filter.eq('ADM0_NAME', 'Bangladesh'));

var geometry = bangladesh.geometry();
Map.centerObject(geometry, 7);

// RGB visualization
var rgbVis = {min: 0.0, max: 3000, bands: ['B4', 'B3', 'B2']};

// Filter Sentinel-2 collection
var filteredS2 = s2
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30))
  .filter(ee.Filter.date('2019-01-01', '2020-01-01'))
  .filter(ee.Filter.bounds(geometry));

// Create composite
var composite = filteredS2.median().clip(geometry);
Map.addLayer(composite, rgbVis, 'Bangladesh RGB Composite');
// NDVI function
function addNDVI(image) {
  var ndvi = image.normalizedDifference(['B8', 'B4']).rename('ndvi');
  return image.addBands(ndvi);
}
// Add NDVI band
var withNdvi = filteredS2.map(addNDVI);

// Median composite with NDVI
var compositeNdvi = withNdvi.median();

// Extract NDVI band
var ndviComposite = compositeNdvi.select('ndvi');

// NDVI visualization
var palette = [
  'FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163', '99B718',
  '74A901', '66A000', '529400', '3E8601', '207401', '056201',
  '004C00', '023B01', '012E01', '011D01', '011301'
];

var ndviVis = {min: 0, max: 0.8, palette: palette};

// Add NDVI layer
Map.addLayer(ndviComposite.clip(geometry), ndviVis, 'NDVI Bangladesh');
```

Cloud Masking in GEE

Mr. Al-Helal
Programmer (Computer & GIS Unit)
Bangladesh Agricultural Research Council
Email: al.helal@barc.gov.bd

Masking pixels in an image makes those pixels transparent and excludes them from analysis and visualization. To mask an image, we can use the updateMask() function and pass it an image with 0 and 1 values. All pixels where the mask image is 0 will be masked. Most remote sensing datasets come with a QA or Cloud Mask band that contains the information on whether pixels is cloudy or not. Your *Code Editor* contains pre-defined functions for masking clouds for popular datasets under *Scripts Tab* → *Examples* → *Cloud Masking*. To understand how cloud-masking functions work and learn advanced techniques for bitmasking, please refer to our article on Working with QA Bands and Bitmasks in Google Earth Engine.

Applying pixel-wise QA bitmask

```
var maskS2Clouds = function(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).copyProperties(image,
['system:time_start']);
};
```

05. Reducers

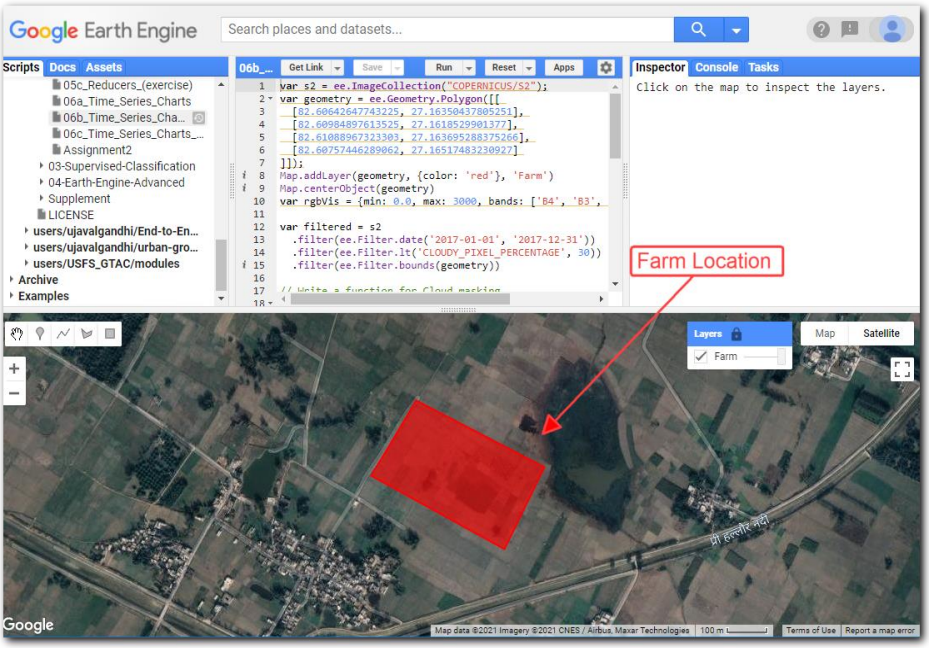
When writing parallel computing code, a *Reduce* operation allows you to compute statistics on a large amount of inputs. In Earth Engine, you need to run reduction operation when creating composites, calculating statistics, doing regression analysis etc. The Earth Engine API comes with a large number of built-in reducer functions (such as ee.Reducer.sum(), ee. Reducer.histogram(), ee.Reducer.linearFit() etc.) that can perform a variety of statistical operations on input data. You can run reducers using the reduce() function. Earth Engine supports running reducers on all data structures that can hold multiple values, such as Images (reducers run on different bands), ImageCollection, FeatureCollection, List, Dictionary etc. The script below introduces basic concepts related to reducers.

```
// BD Bangladesh location (Dhaka)
var geometry = ee.Geometry.Point([90.4125, 23.8103]);
Map.centerObject(geometry, 10);

// ========================================
// Sentinel-2 Image Collection
// ========================================
var s2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED');

// Efficient filtering (chain method)
var filtered = s2
  .filterBounds(geometry)
  .filterDate('2024-01-01', '2024-12-31')
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30))
  .select(['B2', 'B3', 'B4', 'B8']); // important bands only

print('Filtered Image Count:', filtered.size());

// ========================================
// Mean Composite Image (Time Series Average)
// ========================================
var meanImage = filtered.mean();

// RGB Visualization
var rgbVis = {
  min: 0,
  max: 3000,
  bands: ['B4', 'B3', 'B2']
};

Map.addLayer(meanImage.clip(geometry.buffer(5000)), rgbVis, 'Mean Composite
(BD)');
// Farm / Area boundary visualization
Map.addLayer(geometry, {color: 'red'}, 'Point Location');
// ========================================
// Region Statistics (Efficient reduceRegion)
// ========================================
```

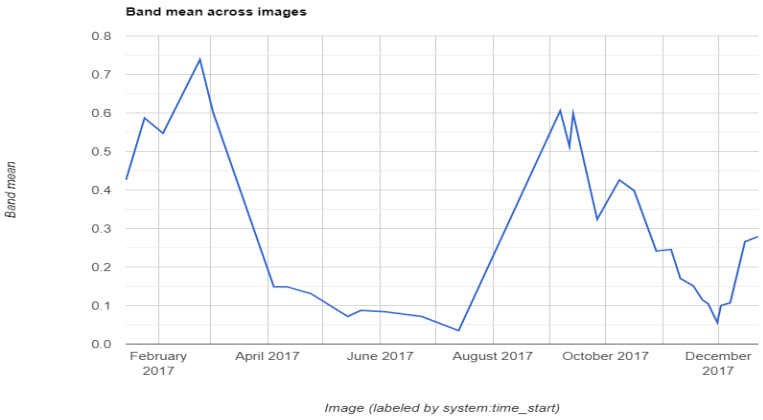
```
var stats = meanImage.reduceRegion({
  reducer: ee.Reducer.mean(),
  geometry: geometry.buffer(1000), // buffer for meaningful stats
  scale: 10,
  maxPixels: 1e13
});
print('Band Statistics (Mean Values):', stats);
// Extract specific band value
print('Mean B4 (Red Band):', stats.getNumber('B4'));
```

06. Time-Series Charts

Now we can put together all the skills we have learnt so far - filter, map, reduce, and cloud-masking to create a chart of average NDVI values for a given farm over 1 year. Earth Engine API comes with support for charting functions based on the Google Chart API. Here we use the ui.Chart.image.series() function to create a time-series chart.



Computing NDVI Time-series for a Farm



NDVI Time-series showing Dual-Cropping Cycle

Basic Concepts of Remote Sensing Technology

Mr. Hasan Mahmud
Senior System Anslyst (Computer & GIS Unit)
Bangladesh Agricultural Research Council
Email: hasan.mahmud@barc.gov.bd

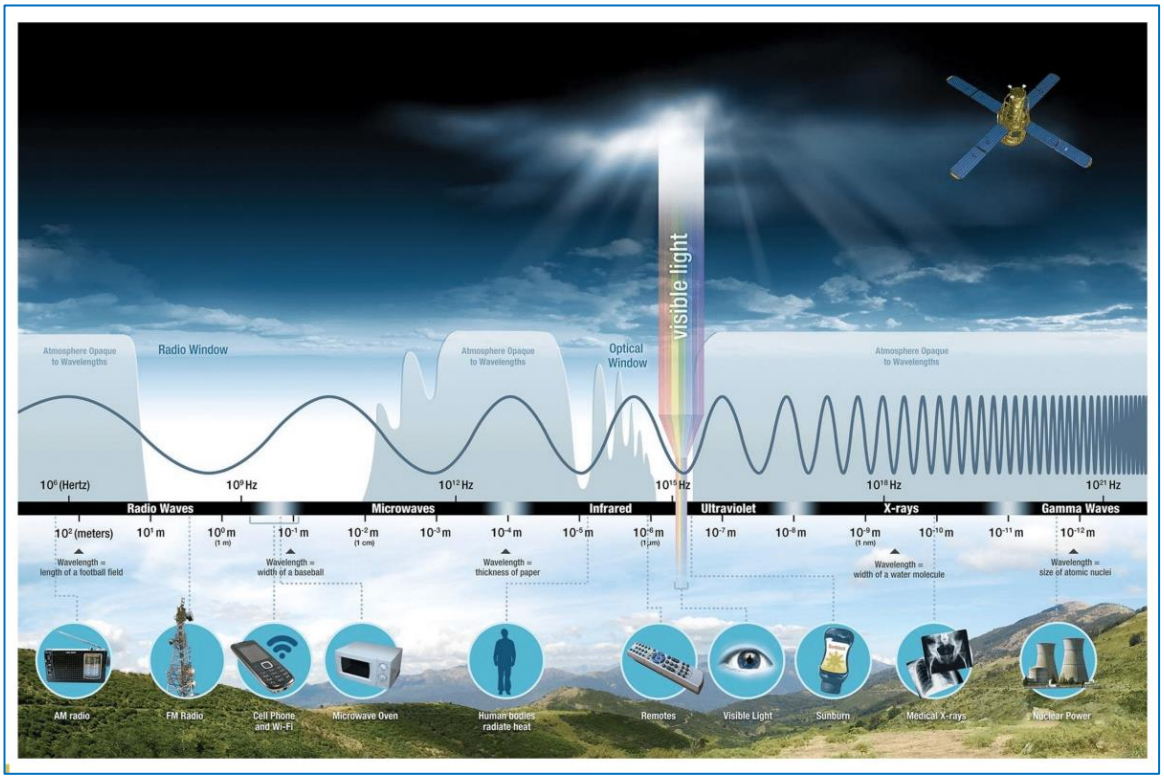
1. Introduction to Remote Sensing

Remote sensing is the science of obtaining information about objects or areas from a distance, typically using satellite or aerial sensor technologies. It plays a crucial role in monitoring and managing natural resources, urban planning, agriculture, disaster response, and climate studies. The technology leverages electromagnetic radiation to gather data without direct contact with the subject.

A formal definition:

"Remote sensing is the science (and to some extent, art) of acquiring information about the Earth's surface without actually being in contact with it. This is done by sensing and recording reflected or emitted energy and processing, analyzing, and applying that information."

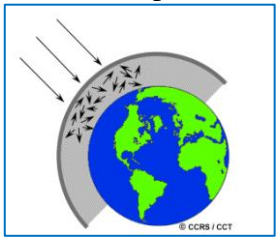
2. Electromagnetic Spectrum and Radiation Principles

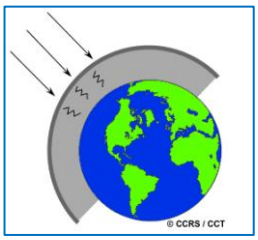


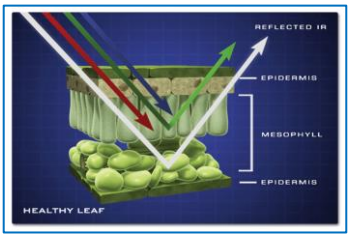
Remote sensing relies on electromagnetic radiation (EMR), which is emitted or reflected by objects. The EM spectrum includes various wavelengths from gamma rays to radio waves. Different materials reflect, absorb, or emit EM radiation uniquely, allowing remote sensors to identify and analyze them through their spectral signatures.

3. Interactions with Atmosphere and Earth's Surface

Before reaching sensors, radiation interacts with the atmosphere through scattering and absorption. Upon striking the Earth, radiation may be absorbed, transmitted, or reflected, depending on surface characteristics. Vegetation, water, and soil each exhibit distinct reflectance patterns across wavelengths.

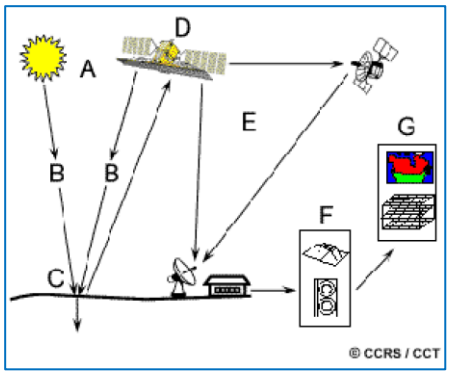


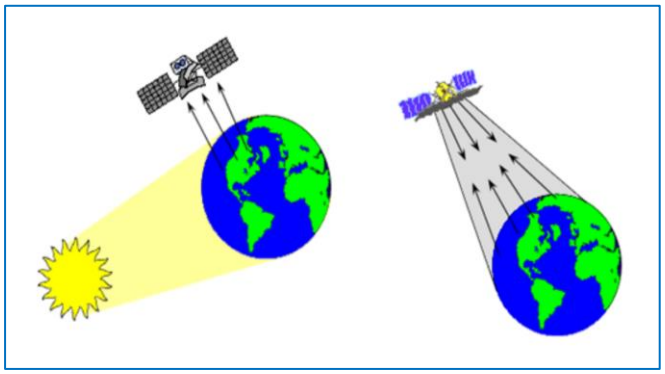




4. Remote Sensing Systems: Passive and Active

Remote sensing systems can be classified as passive or active. Passive systems detect natural energy (typically from the sun) reflected by the surface, while active systems emit their own signal and measure its reflection. Examples include optical sensors (passive) and radar systems (active).





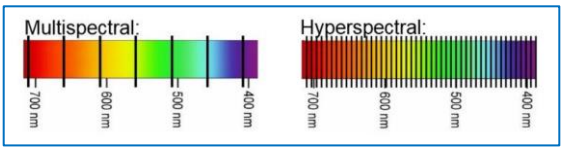
5. Types of Resolutions

Resolution in remote sensing defines the level of detail. The four primary types are:

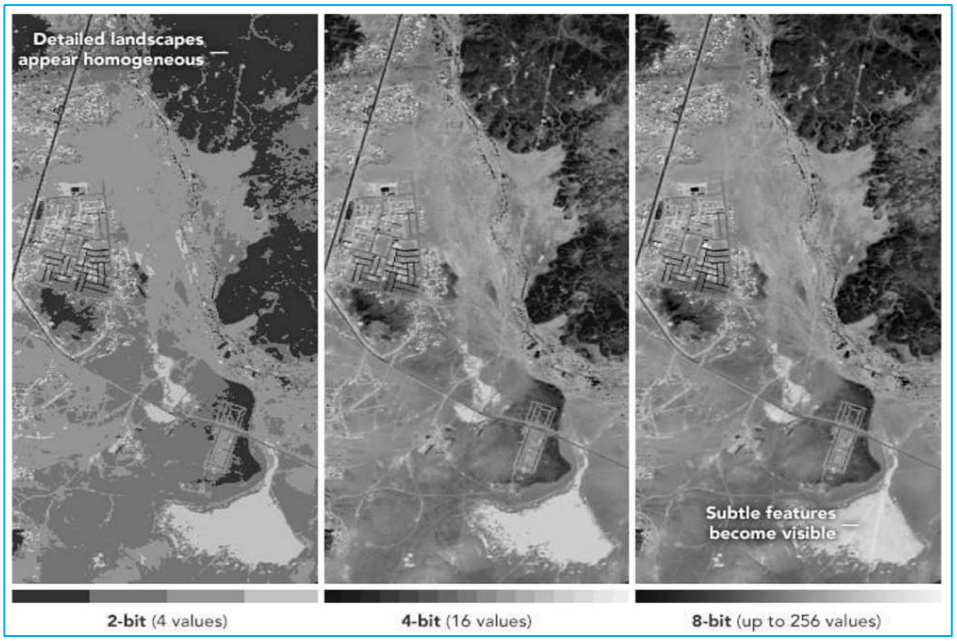
Spatial: Size of the smallest object detectable.

Spectral: Number and width of spectral bands.

Temporal: Frequency of data acquisition.



Radiometric: Sensitivity to differences in signal strength. Represented as how many bits are used to represent a reflectance value.



6. Satellites, Orbits, and Sensors

Satellites operate in various orbits - geostationary, polar, or sun-synchronous - each serving different observation purposes. Sensors onboard these satellites, such as MODIS, Landsat OLI, and Sentinel-1 SAR, differ in design and application. Key satellite parameters include orbit type, swath width, and revisit time.

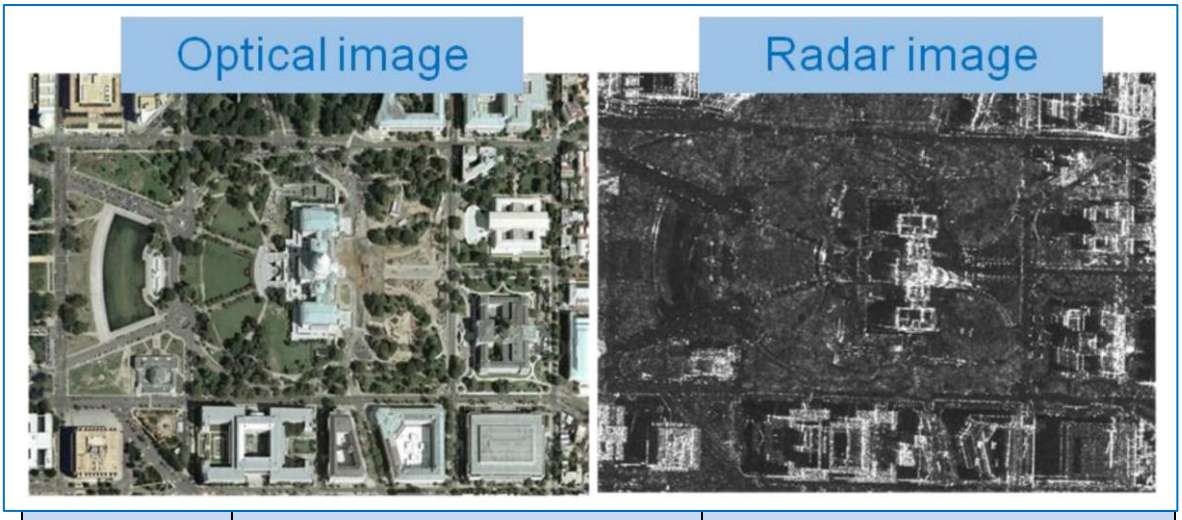
Parameter	Sentinel-1	Sentinel-2	Landsat-8	Landsat-9	MODIS (Terra/Aqua)
Orbit	Sun-synchronous, Near-polar				
Altitude	~693 km	~786 km	~705 km	~705 km	~705 km
Repeat Cycle	12 days (6 days A+B)	10 days (5 days A+B)	16 days	16 days (8 days Landsat-8)	1-2 days (near-global daily)
Sensor	C-SAR	MSI	OLI & TIRS	OLI-2 & TIRS-2	MODIS
Resolution	5-20 m	10 m, 20 m, 60 m	15 m (pan), 30 m (MS), 100 m (Thermal)		250 m, 500 m, 1 km
Swath Width	80-400 km	290 km	185 km	185 km	2330 km
Spectral Bands	HH, VV, HV, VH	13	9 (Vis, NIR, SWIR, Pan) + 2 (Thermal)		36 (Visible to Thermal IR)
Data Type	Radar Backscatter (Intensity & Phase)	TOA Reflectance	TOA Reflectance, Thermal Radiance		Radiance, Reflectance, Temperature, Indices, etc.

7. Data Processing Levels

Satellite data is processed in levels:

- Level 0/1: Raw Sensor Data.
- Level 2: Geophysical Variables (e.g., surface temperature).
- Level 3: Gridded, Calibrated data, Quality Controlled Data.
- Level 4: Model-assimilated products.

8. Optical Vs Radar



Feature	Optical Imagery	Radar Imagery (SAR)
Sensor Type	Passive (detects reflected/emitted light)	Active (emits and receives microwaves)
Illumination	Sun (mostly), thermal emission	Self-illuminated (microwaves)
Weather	Affected by clouds, fog, smoke	Can penetrate clouds, fog, smoke
Day/Night	Primarily daytime	Day and night
Penetration	Limited	Can penetrate vegetation, some ground
Data Captured	Spectral reflectance (color, vegetation health, composition)	Surface roughness, dielectric properties, geometry
Image Look	Visually intuitive (photo-like)	Grayscale, textural, requires specific interpretation
Resolution	Can be very high	High (with SAR processing)

9. Important Indices

NDVI, NDWI, NDSI, and EVI are key vegetation and land surface indices derived from multispectral satellite data. NDVI uses red and NIR bands to assess vegetation greenness, while NDWI (using NIR and SWIR) monitors vegetation water content and open water. NDSI (using green and SWIR) is designed for snow detection. EVI is an enhanced vegetation index utilizing blue, red, and NIR bands to minimize atmospheric and soil noise and reduce saturation in dense vegetation, offering a more robust measure compared to NDVI in certain environments. The Leaf Area Index (LAI) is a crucial biophysical parameter that quantifies the amount of leaf material present in a plant canopy or ecosystem. Defined as one-half of the total green leaf area per unit horizontal ground surface area (for broadleaf canopies) or a related measure for coniferous canopies. Each index provides specific insights into surface characteristics but has limitations related to atmospheric conditions, background influences, and the specific properties they aim to measure.

NDVI = (NIR - Red) / (NIR + Red)

EVI = G * ((NIR - Red) / (NIR + C1 * Red - C2 * Blue + L))

SAVI = ((NIR - Red) / (NIR + Red + L)) * (1 + L)

NDWI (Common) = (NIR - SWIR) / (NIR + SWIR)

NDSI = (Green - SWIR) / (Green + SWIR)

EVI = G * ((NIR - Red) / (NIR + C1 * Red - C2 * Blue + L))

(where G, C1, C2, and L are adjustment factors)

10. Advantages and Challenges

Advantages:

- Provides data where ground access is limited
- Offers global coverage and consistent observations
- Supports real-time and historical analysis
- Information on non-visible regions of the spectrum

Challenges:

- Balancing resolution parameters
- Data volume and complexity
- Need for calibration and validation
- Costly high-resolution Imagery

References:

- NASA's Applied Remote Sensing Training Program
- Canada Centre for Remote Sensing Remote Sensing Tutorial
- Tutorial by TELECAN on Fundamentals of Remote Sensing

Vegetation Indices for Crop Health Monitoring

Mustafa Kamal
GIS and Remote Sensing Specialist
CIMMYT

Session Overview and Objectives

This session provides a comprehensive introduction to crop health monitoring using remote sensing and vegetation indices. It begins with the challenges of traditional field-based monitoring and highlights how satellite remote sensing offers efficient, large-scale, and cost-effective solutions for agricultural observation.

Participants will learn the fundamentals of electromagnetic radiation and its interaction with vegetation, including spectral signatures of healthy crops, soil, and water. The session also explains how satellite sensors capture this information and convert it into usable data for analysis.

The training further focuses on vegetation indices—what they are, how they are classified, and how they are used to assess crop health, biomass, water stress, and nutrient status. Key indices such as NDVI, NDWI, EVI, SAVI, NDMI, and NBR are discussed in terms of their computation, applications, advantages, and limitations.

By the end of the session, participants will be able to understand remote sensing principles in agriculture, interpret vegetation indices, and apply them for practical decision-making in crop monitoring, precision agriculture, and environmental assessment.

1. Introduction to Crop Health Monitoring

1.1 What is Crop Health Monitoring?

Crop health monitoring is the process of observing crop conditions regularly to detect:

- Healthy crop growth
- Nutrient deficiencies
- Water stress
- Pest and disease damage
- Drought impacts
- Flood effects

Traditional field surveys are:

- Time-consuming
- Expensive
- Limited in spatial coverage

Satellite remote sensing provides:

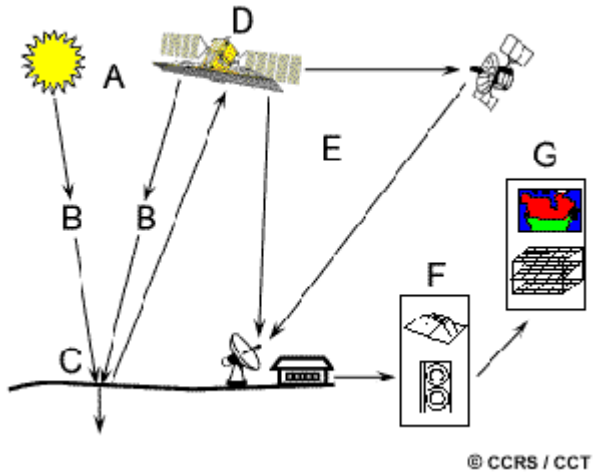
- Large-area monitoring
- Frequent observations
- Cost-effective analysis
- Near real-time information

2. Remote Sensing and Agriculture

2.1 What is Remote Sensing?

Remote Sensing is the science and technology of obtaining information about objects or areas on the Earth's surface without making direct physical contact with them.
It works by detecting and measuring reflected or emitted electromagnetic radiation from a distance, typically using sensors mounted on satellites, aircraft, or drones.
In agriculture, remote sensing is widely used to monitor crop health, soil conditions, water availability, and land-use changes over large areas efficiently and repeatedly.

2.2 Remote Sensing Process



1. Energy Source or Illumination (A)
2. Radiation and the Atmosphere (B)
3. Interaction with the Target (C)
4. Recording of Energy by the Sensor (D)
5. Transmission, Reception, and Processing (E)
6. Interpretation and Analysis (F)
7. Application (G)

Fig. 1: Remote Sensing Data Acquisition and Processing

In agriculture, satellites collect reflected energy from crop surfaces.

Examples of agricultural satellite systems:

- Sentinel-2
- Landsat 8/9
- MODIS
- PlanetScope
- Sentinel-1 (SAR radar system useful for cloud-penetrating observations)
- SPOT (Satellite Pour l'Observation de la Terre)

- WorldView satellites (high-resolution commercial imagery)
- RapidEye (agricultural monitoring and vegetation mapping)
- GeoEye-1 (high-resolution Earth observation)
- Aqua and Terra (NASA Earth Observation System satellites)

2.3 Electromagnetic Spectrum and Crop Reflectance

2.3.1 Electromagnetic Spectrum

Radiation that is not absorbed or scattered in the atmosphere can reach and interact with the Earth's surface. There are three (3) forms of interaction that can take place when energy strikes, or is incident (I) upon the surface. These are: absorption (A); transmission (T); and reflection (R).

Fig. 2: Solar Radiation Interaction with Vegetation Canopy

Satellite sensors measure reflected electromagnetic radiation.

Table 1: Important spectral regions for agriculture

Spectral Region	Approximate Wavelength	Importance
Blue	450–500 nm	Chlorophyll absorption
Green	500–570 nm	Plant reflectance peak
Red	620–750 nm	Chlorophyll absorption
Near Infrared (NIR)	750–1300 nm	Plant cell structure
Shortwave Infrared (SWIR)	1300–2500 nm	Water content

Let's take a look at a couple of examples of targets at the Earth's surface and how energy at the visible and infrared wavelengths interacts with them.

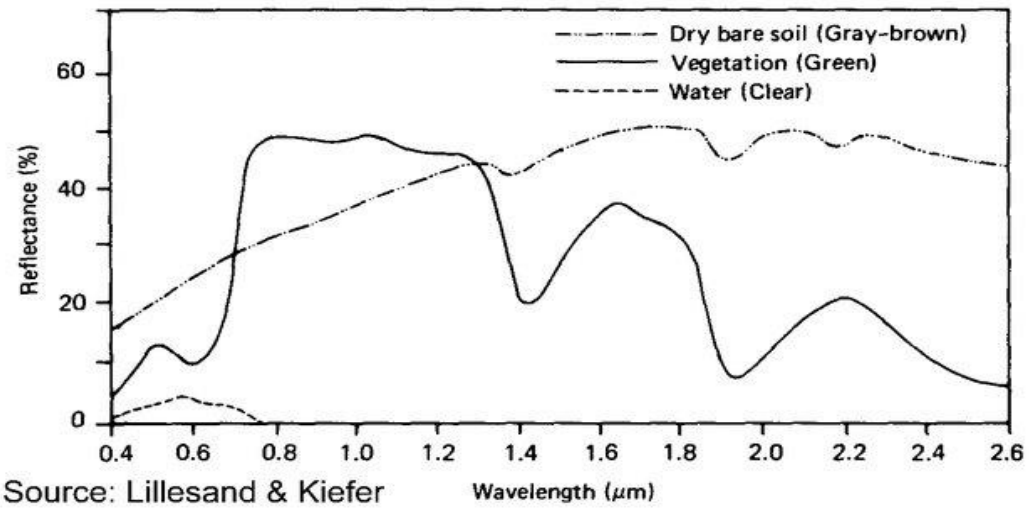


Fig. 3: Spectral Reflectance Curves of Vegetation, Dry Bare Soil, and Water Across the Electromagnetic Spectrum

2.4 How Plants Interact with Light

2.4.1 Healthy Vegetation

Healthy plants:

- Absorb most red light for photosynthesis
- Reflect large amounts of NIR light

2.4.2 Stressed Vegetation

Stressed plants:

- Reflect more red light
- Reflect less NIR light

This difference allows satellites to detect vegetation condition.

2.5 Spectral Signature of Major Agricultural feature

2.5.1 Healthy Vegetation Characteristics

Healthy vegetation shows a distinct spectral response due to chlorophyll and internal leaf structure:

- Low reflectance in the Red band because chlorophyll strongly absorbs red light for photosynthesis
- Very high reflectance in the Near Infrared (NIR) band due to internal leaf cell structure
- Moderate reflectance in the Green band, which is why vegetation appears green to the human eye
- Generally low reflectance in SWIR bands, depending on plant water content

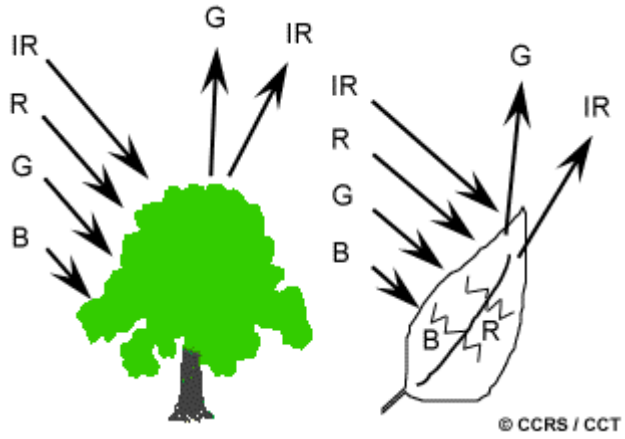


Fig.4: Interaction of Incoming Solar Radiation with Vegetation and Leaf Surfaces Across Spectral Bands

2.5.2 Bare Soil Characteristics

Bare soil has a more uniform spectral response compared to vegetation:

- Moderate and gradually increasing reflectance across visible to infrared bands
- Reflectance varies depending on soil moisture, texture, and organic content
- Dry soil tends to have higher reflectance, while wet soil shows lower reflectance
- Lacks strong absorption or reflection features like vegetation

2.5.3 Water Bodies

Water has a strong absorption signature across infrared wavelengths:

- Very low reflectance in NIR and SWIR bands due to strong absorption by water
- Low reflectance even in visible bands, especially in deep or clear water
- Slight reflectance in blue-green wavelengths, which is why shallow or turbid water may appear brighter
- Reflectance decreases further with increasing water depth and clarity

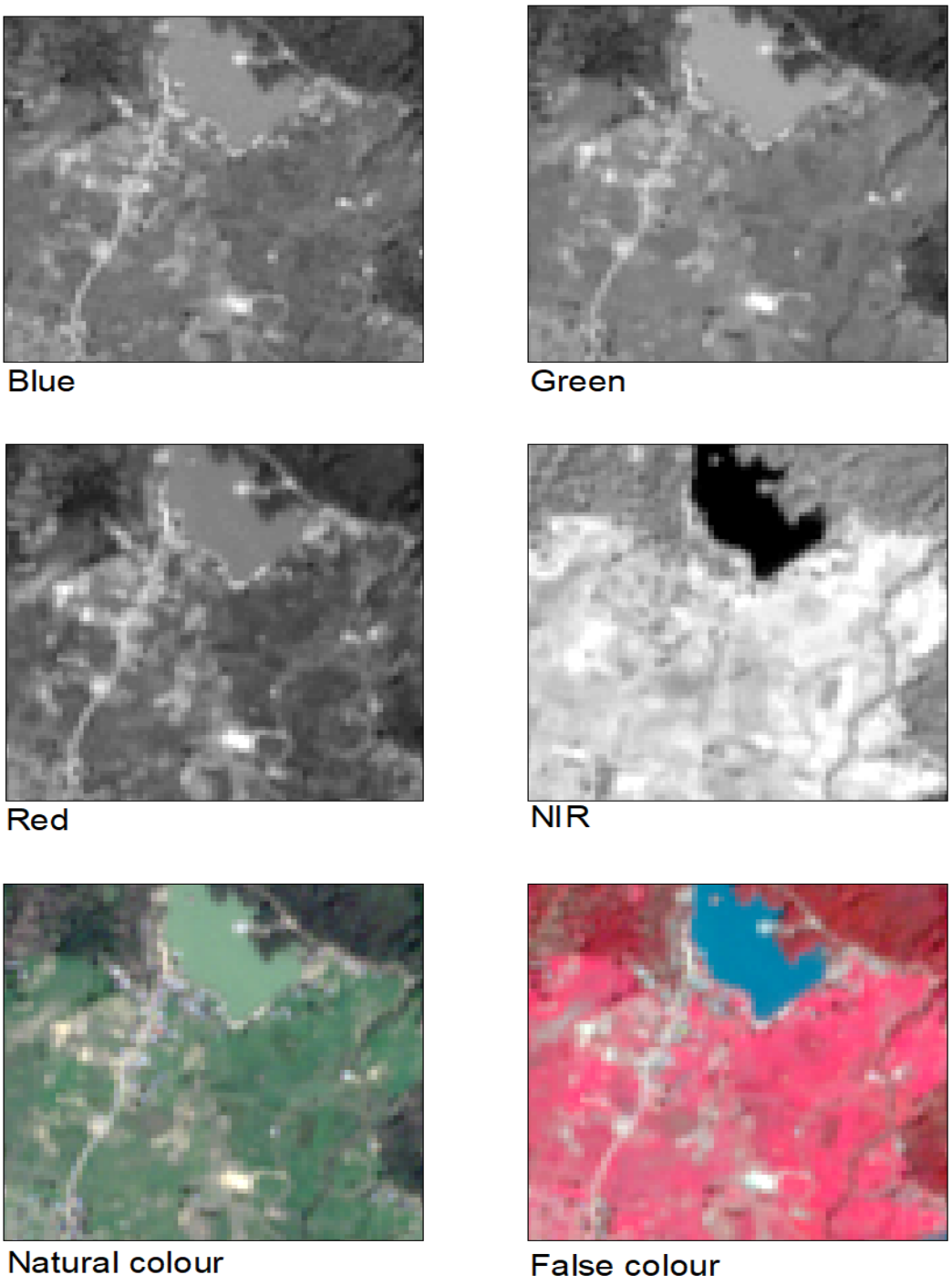


Fig. 5: Common satellite bands and band composition

4 Remote Sensing Indices

A mathematical combination of spectral bands used to monitor and analyze vegetation characteristics from remotely sensed data.

Remote sensing vegetation indices are numerical indicators calculated from satellite, drone, or aerial imagery to measure vegetation properties such as greenness, health, biomass, chlorophyll content, and water stress. In short a mathematical combination of spectral bands used to monitor and analyze vegetation characteristics from remotely sensed data.

They are created by combining reflectance values from different parts of the electromagnetic spectrum, mainly:

- Red light
- Green light
- Near-Infrared (NIR)
- Shortwave Infrared (SWIR)

Healthy vegetation reflects strongly in the NIR region and absorbs red light for photosynthesis. Vegetation indices use this difference to assess plant condition.

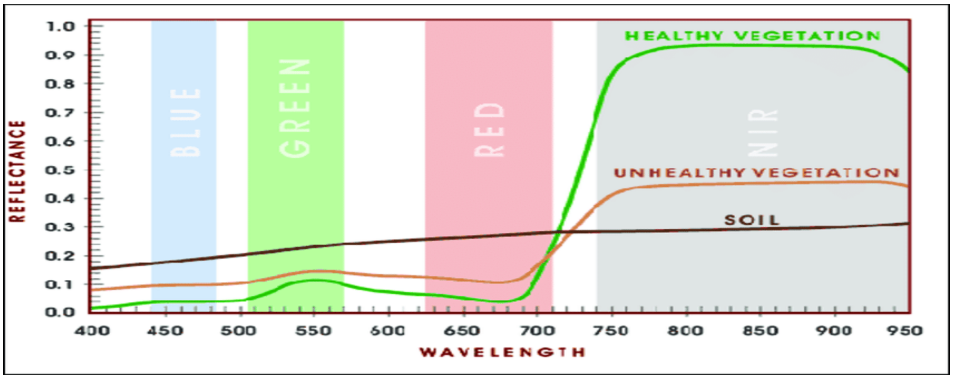


Fig. 6: Spectral reflectance of healthy plant, unhealthy plant, and soil in visible and NIR wavelength. (Source: http://extension.usu.edu/nasa/htm/on-target/near-infrared-tutorial/)

4.1 What are Vegetation Indices?

A Vegetation Index assists in periodic remote observations of vegetation and has been actively used since 1974. Vegetation indices (VIs) are mathematical expressions that transform spectral imaging data to highlight vegetation properties.

Using this algorithm, scientists and other concerned personnel effectively observe photo-centric activities, identify variations in the canopy, and draw accurate comparisons if needed. It includes assessing various aspects, like crop growth, vigor, biomass, and chlorophyll content.

Vegetation indices are mathematical combinations of spectral bands designed to:

- Enhance vegetation signals
- Reduce noise
- Monitor crop conditions

They simplify satellite data into interpretable indicators.

4.2 Why Use Vegetation Indices?

Vegetation indices help:

1. Detect crop stress
2. Monitor crop growth stages
3. Estimate biomass
4. Assess drought conditions
5. Monitor and manage irrigation
6. Detect burned areas
7. Support precision agriculture
8. Classify vegetation types
9. Estimate soil moisture and water content remotely
10. Monitor evaporation and plant transpiration
11. Detect and quantify crop diseases
12. Assess tillage practices
13. Identify changes in biodiversity
14. Track periodic environmental and land-cover changes over time

4.3 Benefits of vegetation indices

Vegetation indices are vital in crop growth analytics, empowering sustainable and responsible food production. Vegetation indices can provide the following benefits:

- Remote intelligence: Provides valuable intelligence into Crop health.
- Data precision: Ensures accurate and reliable measurements of the status of your farmlands; gets data analytics in a preferred format
- Cost efficiency: It is more cost-effective as it reduces the need for physical field inspections.
- Miles-away control: Allows companies to monitor and manage farm operations remotely without disturbing field activities.
- Enable scale: It allows monitoring of large-scale field activities in a short time and can also be scaled quickly and efficiently.
- Constant Monitoring: Monitor fields continuously and provide access to different image sources in one place.

4.4 Common Satellite Sources for Vegetation Indices and Selection Criteria

Several satellites provide vegetation indices data. The most common ones include Advanced Very High-Resolution Radiometer (AVHRR), Moderate- resolution Imaging Spectroradio meter (MODIS) Vegetation 1 and 2 sensors, Landsat series, Sentinel-2, WorldView, Planet.

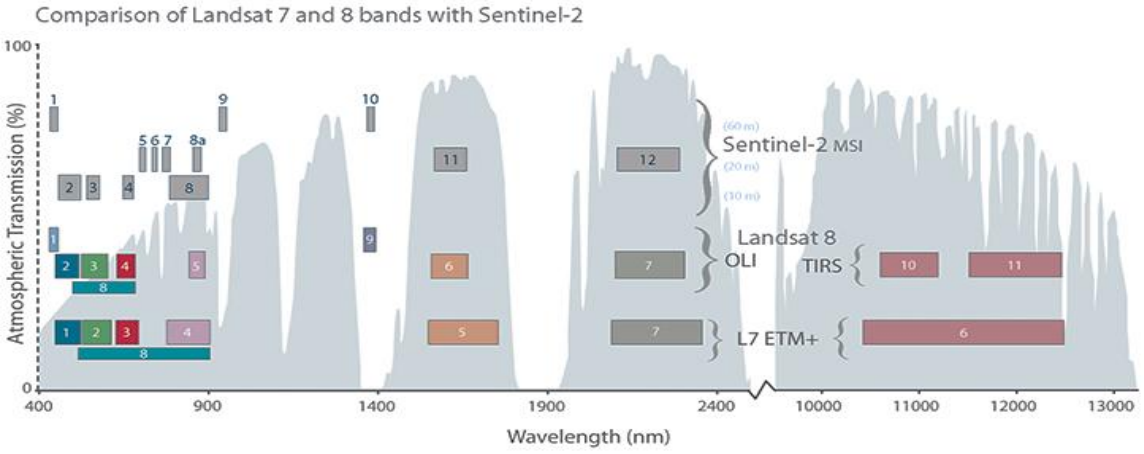


Fig.7: Comparison of Landsat 7, 8 and Sentinel-2 Spectral Bands

Source: https://landsat.gsfc.nasa.gov/wp-content/uploads/2015/06/Landsat.v.Sentinel-2.png

The best choice will depend on your specific needs. Some factors to consider are:

- Resolution: How much detail do you need?
- Coverage area: How large is the area of interest?
- Temporal resolution: How often do you need data updates?
- Cost: Is free data sufficient, or will you pay for higher resolution options?

5. Classification of Vegetation Indices

5.1 Classification by Functional Type

5.1.1 Broadband Vegetation Indices

These use broad spectral bands such as Red, Green, Near-Infrared (NIR), and Shortwave Infrared (SWIR).
Common examples
Normalized Difference Vegetation Index, Enhanced Vegetation Index, Soil Adjusted Vegetation Index, Green Normalized Difference Vegetation Index, Difference Vegetation Index, Atmospherically Resistant Vegetation Index

5.1.2 Chlorophyll / Pigment Indices

These focus on chlorophyll concentration, photosynthetic activity, and nutrient status.
Examples: Red Edge Normalized Difference Vegetation Index, Chlorophyll Vegetation Index, Modified Chlorophyll Absorption Ratio Index, Transformed Chlorophyll Absorption Reflectance Index etc.

5.1.3 Water and Moisture Indices

These estimate vegetation water content and drought stress.
Examples: Normalized Difference Water Index, Moisture Stress Index, Normalized Difference Moisture Index etc.

5.1.4 Thermal Vegetation Indices

These combine thermal infrared information with vegetation data to detect heat and stress.
Examples: Vegetation Health Index, Temperature Vegetation Dryness Index etc.

5.1.5. Red-Edge Vegetation Indices

Use the “red-edge” spectral region (between red and NIR), especially useful with hyperspectral and modern multispectral sensors.
Examples: Red Edge Position, MERIS Terrestrial Chlorophyll Index, Red Edge Simple Ratio

5.1.6. Hyperspectral Vegetation Indices

Derived from hyperspectral imagery with many narrow spectral bands.
Examples: Photochemical Reflectance Index, Structure Insensitive Pigment Index, Water Band Index etc.

5.2 Classification by application:

Purpose	Common Indices
Vegetation vigor	NDVI, EVI
Biomass estimation	SAVI, EVI
Chlorophyll/nitrogen	GNDVI, NDRE, MCARI
Water stress	NDWI, NDMI, MSI
Drought monitoring	TVDI, VHI
Sparse vegetation	SAVI, ARVI
Dense canopy analysis	EVI, NDRE

5.3 Classification of Vegetation Indices Based on Sensor Type:

Table 2: Vegetation Indices and Their Association with Remote Sensing Sensor Types, Applications, and Data Sources

Sensor Type	Common Vegetation Indices	Notes / Applications
RGB Sensors (Visible Bands Only)	VARI, ExG (Excess Green), TGI, GLI	Used with standard digital cameras and low-cost drones where NIR is unavailable
Multispectral Sensors	NDVI, GNDVI, SAVI, EVI, NDWI, NDRE, MSAVI	Most common in agriculture, forestry, and satellite remote sensing
Hyperspectral Sensors	PRI, MCARI, TCARI, Red Edge Indices, SIPI	High spectral resolution for biochemical and stress analysis
Thermal Sensors	TVDI, CWSI, VHI	Detect plant temperature, drought stress, and evapotranspiration
Red-Edge Sensors	NDRE, CIred-edge, MTCI	Sensitive to chlorophyll content and crop nutrient status
Microwave / Radar Sensors (SAR)	RVI (Radar Vegetation Index), Radar Forest Degradation Index	Useful under cloud cover and for biomass/forest structure monitoring
LiDAR Sensors	LAI, Canopy Height Models (CHM), Gap Fraction Metrics	Measures vegetation structure, canopy height, and biomass
Fluorescence Sensors	SIF (Solar-Induced Fluorescence)	Measures photosynthetic activity directly
Satellite Multispectral Systems	NDVI, EVI, SAVI, NDWI	Common with Landsat Program, Sentinel-2, and MODIS
Drone/UAV Sensors	NDVI, VARI, GNDVI, NDRE, SAVI	High-resolution precision agriculture and crop scouting

5.4 Vegetation Index Databases and Resources

5.4.1 Index database: A database for remote sensing indices

https://www.indexdatabase.de/db/i.php?offset=1&order=-rcount

5.4.2 Sentinel-2 RS index collection

The following link is the collection of remote sensing indices that has been constructed from the information available at the Index database (IDB) specifically for Sentinel-2 satellite.

Javascript functions have been parsed from the data automatically.

https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/indexdb/

5.4.3 Radiometric Indices in Sentinel-2 Toolbox

The indices provided with Sentinel-2 Toolbox, detailed below, are grouped into three categories:

- Vegetation indices
- Soil indices
- Water indices

https://step.esa.int/main/wp-content/help/versions/9.0.0/snap-toolboxes/org.esa.s2tbx.s2tbx.radiometric.indices.ui/OperatorsIndexList.html

6. Vegetation Indices and Remote Sensing Features in Agricultural Monitoring

Table 3: Agricultural Applications and Geographic Distribution of Key Vegetation Indices and Remote Sensing Features

Vegetation Index / Data Feature	Rank-Based Application in Agriculture	Geography (Study Areas)	
EVI (Enhanced Vegetation Index)	Mung bean mapping, crop discrimination in fragmented fields, dense canopy crops	Coastal South Asia (Bangladesh), global smallholder systems	[1], [2]
CI (Chlorophyll Index)	Nitrogen stress, crop type separation, yield estimation	South Asia, Europe, North America (crop trials)	[3]
NDWI2 (Water-sensitive index variant)	Crop water stress, irrigation response monitoring	Bangladesh, India, semi-arid Asia	[4]
SAVI (Soil Adjusted Vegetation Index)	Early-stage crops, sparse vegetation, soil background correction	South Asia, Sub-Saharan Africa drylands	[5]
BI2 (Brightness Index)	Bare soil detection, tillage/land preparation mapping	Agricultural landscapes globally (India, Africa, Middle East)	[6]
NDWI (Normalized Difference Water Index)	Irrigation monitoring, drought impact, rice water status	Bangladesh, Vietnam, India, China	[4], [8]
NDVI (Normalized Difference Vegetation Index)	General crop mapping (rice, wheat, maize), biomass, phenology	Global (South Asia, Africa, Europe, USA)	[7], [8], [9]
GNDVI (Green NDVI)	Chlorophyll monitoring, nitrogen stress, crop vigor	Wheat belts (India, USA), maize systems (Africa)	[3]
SAR Backscatter (Sentinel-1 VV/VH)	Rice mapping, flood detection, fall armyworm detection	Bangladesh, Afghanistan, Sub-Saharan Africa	[8], [10]
Time-series NDVI + SAR fusion	In-season rice mapping, crop calendar estimation, multi-cropping	Bangladesh, India, Vietnam, Afghanistan	[8], [11]

Vegetation Index / Data Feature	Rank-Based Application in Agriculture	Geography (Study Areas)	
	detection		
NDVI + EVI + phenology metrics	Wheat and maize classification, yield prediction	Indo-Gangetic Plain (India, Pakistan), USA Corn Belt	[12]

7. Comparative Influence of Vegetation Indices Across Agricultural and Environmental Applications

Table 4: Influence of Key Vegetation Indices on Agricultural and Environmental Applications[13-22]

Application / Influence Area	NDVI	NDWI	EVI	SAVI	NDMI	NBR
1. Detect crop stress	●	◐	●	◐	●	○
2. Monitor crop growth stages	●	◐	●	◐	◐	○
3. Estimate biomass	●	○	●	◐	◐	○
4. Assess drought conditions	◐	●	◐	○	●	◐
5. Monitor and manage irrigation	◐	●	◐	○	●	○
6. Detect burned areas	○	○	○	○	◐	★
7. Support precision agriculture	●	●	●	◐	●	○
8. Classify vegetation types	●	◐	●	◐	◐	○
9. Estimate soil moisture and water content remotely	○	●	○	○	★	◐
10. Monitor evaporation and plant transpiration	◐	●	◐	○	●	○
11. Detect and quantify crop diseases	●	◐	●	◐	●	○
12. Assess tillage practices	○	○	○	●	◐	○
13. Identify changes in biodiversity	◐	○	●	◐	◐	○
14. Track periodic environmental and land-cover changes over time	●	◐	●	◐	◐	●

Legend

★ = Very Strong relevance

● = Strong relevance

◐ = Moderate relevance

○ = Weak relevance

8. Major Vegetation Indices: Concept, Application, Advantages and Limitation

8.1 NDVI – Normalized Difference Vegetation Index

$$NDVI = \frac{NIR - Red}{NIR + Red}$$

Concept

NDVI measures vegetation greenness and vigor.

Healthy vegetation strongly reflects NIR and absorbs red light.

NDVI Value Range

NDVI Value	Interpretation
< 0	Water/cloud/snow
0 – 0.2	Bare soil
0.2 – 0.5	Sparse vegetation
0.5 – 1.0	Dense healthy vegetation

Applications

- Crop health monitoring
- Vegetation mapping
- Biomass estimation
- Drought assessment

Advantages

- Simple and widely used
- Easy interpretation

Limitations

- Saturation in dense vegetation
- Sensitive to soil background

8.2 NDWI – Normalized Difference Water Index

$$NDWI = \frac{NIR - SWIR}{NIR + SWIR}$$

Concept

NDWI estimates vegetation water content.

Water absorbs SWIR radiation strongly.

Applications

- Irrigation monitoring
- Drought analysis
- Water stress detection

Interpretation

Higher NDWI:

- Higher moisture content

Lower NDWI:

- Dry vegetation or water stress

8.3 EVI – Enhanced Vegetation Index

$$EVI = 2.5 \times \frac{NIR - Red}{NIR + 6 \times Red - 7.5 \times Blue + 1}$$

Concept

EVI improves vegetation monitoring in dense vegetation areas.

It reduces:

- Atmospheric effects
- Soil background effects

Applications

- Dense crop monitoring
- Forest monitoring
- Tropical vegetation analysis

Advantages

- Better sensitivity in high biomass areas
- Improved atmospheric correction

8.4 SAVI – Soil Adjusted Vegetation Index

$$SAVI = \frac{(NIR - Red)(1 + L)}{NIR + Red + L}$$

Concept

SAVI minimizes soil brightness effects.

The factor:

- L = soil adjustment factor
- Commonly L = 0.5

Applications

- Sparse vegetation areas
- Early crop growth stages
- Semi-arid agriculture

Advantages

- Better than NDVI where soil is exposed

8.5 NDMI – Normalized Difference Moisture Index

$$NDMI = \frac{NIR - SWIR}{NIR + SWIR}$$

Concept

NDMI measures vegetation moisture condition.

Sensitive to:

- Leaf water content
- Canopy moisture

Applications

- Drought monitoring
- Crop water stress analysis
- Forest moisture monitoring

8.6 NBR – Normalized Burn Ratio

$$NBR = \frac{NIR - SWIR2}{NIR + SWIR2}$$

Concept

NBR is designed to detect burned vegetation areas.

Burned areas:

- Decrease NIR reflectance
- Increase SWIR reflectance

Applications

- Fire severity mapping
- Vegetation disturbance analysis

8.7 Comparison of Vegetation Indices

Index	Main Use	Main Bands	Key Advantage
NDVI	Vegetation vigor	NIR, Red	Simple and widely used
NDWI	Water content	NIR, SWIR	Detects moisture stress
EVI	Dense vegetation	NIR, Red, Blue	Reduces atmospheric effects
SAVI	Sparse vegetation	NIR, Red	Minimizes soil effects
NDMI	Moisture monitoring	NIR, SWIR	Canopy moisture assessment
NBR	Burn severity	NIR, SWIR2	Fire damage detection

10. Interpretation of vegetation indices

Practical Interpretation Examples

Example 1: Healthy Rice Field

- High NDVI
- High NDWI
- Moderate to high EVI

Interpretation:

- Healthy dense vegetation
- Adequate water availability

Example 2: Water-Stressed Crop

- Moderate NDVI
- Low NDWI

- Low NDMI

Interpretation:

- Moisture deficiency
- Possible drought stress

Example 3: Bare Agricultural Land

- Low NDVI
- Low EVI
- High soil influence

Interpretation:

- Harvested or uncultivated field

11. Limitations of Vegetation Indices

Common Limitations

- Cloud contamination
- Atmospheric effects
- Soil background influence
- Mixed pixels
- Seasonal variation

Importance of Preprocessing

Before analysis:

- Cloud masking
- Atmospheric correction
- Image compositing

are essential.

11. Discussion and Q&A

Discussion Questions

1. Why does healthy vegetation reflect more NIR light?
2. Why is NDVI sometimes less effective in dense vegetation?
3. Which index is most useful for drought monitoring?
4. Why is SWIR important for moisture analysis?
5. What challenges exist in satellite-based crop monitoring?

12 Hands-on: NDVI Mapping for Crop Health Monitoring

12.1 Introduction to Practical Session

Objective of the Practical Session

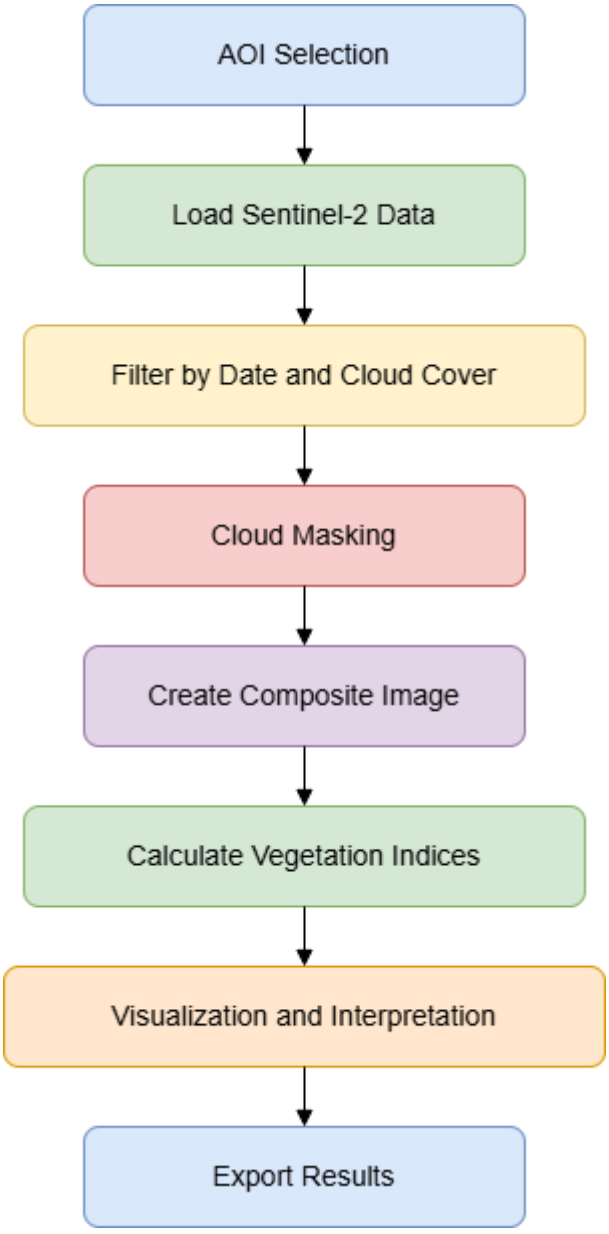
This practical session demonstrates how to use Google Earth Engine (GEE) for satellite-based crop health monitoring using Sentinel-2 imagery.

Participants will learn how to:

- Load Sentinel-2 satellite imagery
- Filter imagery by date, cloud cover, and Area of Interest (AOI)
- Apply cloud masking

- Create a median composite image
- Calculate vegetation indices:
    - NDVI
    - NDWI
    - EVI
    - SAVI
    - NDMI
    - NBR
- Visualize crop health conditions
- Interpret vegetation index outputs
- Export results to Google Drive

12.2 Workflow Overview



12.3 Platform Requirements

Required Platform

- Google Earth Engine (GEE)

Access GEE

https://code.earthengine.google.com/c34f4353aa4d853fab9725009097c882

12.4 Step 1: Define Area of Interest (AOI)

The Area of Interest (AOI) defines the region where analysis will be performed.

- Draw their own AOI in GEE
- Use predefined coordinates

Example AOI Script:

```
var AOI = ee.Geometry.Rectangle([90.20, 23.75, 90.50, 24.00]);
Map.centerObject(AOI, 11);
Map.addLayer(AOI, {color: 'red'}, 'Area of Interest');
```

Component	Description
ee.Geometry.Rectangle	Creates rectangular AOI
Map.centerObject	Centers map on AOI
Map.addLayer	Displays AOI boundary

12.5 Step 2: Load Sentinel-2 Data

Dataset Used

Sentinel-2 Surface Reflectance

Dataset Name:

```
COPERNICUS/S2_SR_HARMONIZED
```

Load Dataset

```
var s2 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED');
```

12.6 Step 3: Filter Satellite Data

Purpose

Filtering removes unnecessary images and improves analysis quality.

Filters applied:

- AOI boundary
- Date range
- Cloud cover percentage

Filter Script:

```
var filtered = s2
  .filterBounds(AOI)
  .filterDate('2024-01-01', '2024-03-31')
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20));
```

Filter	Purpose
filterBounds	Selects images covering AOI
filterDate	Selects images within date range
CLOUDY_PIXEL_PERCENTAGE	Removes cloudy images

Display Collection Information

```
print('Filtered Sentinel-2 Collection:', filtered);
```

12.7 Step 4: Image Preprocessing

12.7.1 Cloud Masking

Purpose

Clouds and cirrus affect vegetation analysis.
The QA60 band stores cloud information.

Cloud Bits

Bit	Description
Bit 10	Clouds
Bit 11	Cirrus Clouds

Cloud Mask Function:

```
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);
}
```

12.7.2 Apply Cloud Mask

```
var cleanCollection = filtered.map(maskS2clouds);
```

12. 7.3 Create Median Composite

Purpose

Median composite:

- Reduces cloud contamination
- Removes noise
- Creates cleaner imagery

Composite Script:

```
var composite = cleanCollection.median().clip(AOI);
```

12.8 Step 5: RGB Visualization

Sentinel-2 RGB Bands

Band	Description
B4	Red
B3	Green
B2	Blue

RGB Visualization Script:

```
var rgbVis = {
  min: 0,
  max: 0.3,
  bands: ['B4', 'B3', 'B2']
};

Map.addLayer(composite, rgbVis, 'Sentinel-2 RGB');
```

12.9. Step 6: Vegetation Indices Calculation

12.9.1 NDVI – Normalized Difference Vegetation Index

NDVI Script

```
var ndvi = composite.normalizedDifference(['B8', 'B4'])
                  .rename('NDVI');
```

NDVI Visualization

```
var ndviVis = {
  min: -1,
  max: 1,
  palette: [
    'blue',
    'white',
    'yellow',
    'green',
    'darkgreen'
  ]
};

Map.addLayer(ndvi, ndviVis, 'NDVI');
```

12.9.2 NDWI – Normalized Difference Water Index

NDWI Script

```
var ndwi = composite.normalizedDifference(['B3', 'B8'])
                  .rename('NDWI');
```

12.9.3 EVI – Enhanced Vegetation Index

EVI Script

```
var evi = composite.expression(
  '2.5 * ((NIR - RED) / (NIR + 6 * RED - 7.5 * BLUE + 1))',
  {
    'NIR': composite.select('B8'),
    'RED': composite.select('B4'),
    'BLUE': composite.select('B2')
  }
```

```
).rename('EVI');
```

12.9.4 SAVI – Soil Adjusted Vegetation Index

SAVI Script

```
var savi = composite.expression(
  '((NIR - RED) / (NIR + RED + 0.5)) * (1.5)',
  {
    'NIR': composite.select('B8'),
    'RED': composite.select('B4')
  }
).rename('SAVI');
```

12.9.5 NDMI – Normalized Difference Moisture Index

NDMI Script

```
var ndmi = composite.normalizedDifference(['B8', 'B11'])
                .rename('NDMI');
```

12.9.6 NBR – Normalized Burn Ratio

NBR Script

```
var nbr = composite.normalizedDifference(['B8', 'B12'])
                .rename('NBR');
```

12.10 Visualization and Interpretation

Healthy vs Stressed Crops

Healthy Crops

- High NDVI
- High EVI
- Moderate to high NDMI

Stressed Crops

- Low NDVI
- Low NDMI
- Sparse vegetation

12.11 Thresholding NDVI

Purpose

Extract healthy vegetation areas.

Script

```
var healthyVeg = ndvi.gt(0.6);

Map.addLayer(
  healthyVeg.selfMask(),
  {palette: ['00FF00']},
  'Healthy Vegetation'
);
```

12.12 Time-Series Comparison

Purpose

Compare vegetation condition between two periods.

Examples:

- Dry season vs wet season
- Before irrigation vs after irrigation

Early Season Composite

```
var earlySeason = filtered
  .filterDate('2024-01-01', '2024-01-31')
  .median()
  .clip(AOI);
```

Late Season Composite

```
var lateSeason = filtered
  .filterDate('2024-02-01', '2024-03-31')
  .median()
  .clip(AOI);
```

NDVI Difference

```
var ndviChange = ndviLate.subtract(ndviEarly)
                         .rename('NDVI Change');
```

12.13 Mean NDVI Calculation

Purpose

Calculate average vegetation condition over AOI.

Script

```
var meanNDVI = ndvi.reduceRegion({
  reducer: ee.Reducer.mean(),
  geometry: AOI,
  scale: 10,
  maxPixels: 1e13
});
```

Display Result

```
print('Mean NDVI:', meanNDVI);
```

12.14 Exporting Results

Export NDVI to Google Drive

```
Export.image.toDrive({
  image: ndvi,
  description: 'NDVI_Map',
  folder: 'GEE_Output',
  fileNamePrefix: 'NDVI_2024',
  region: AOI,
  scale: 10,
  maxPixels: 1e13
});
```

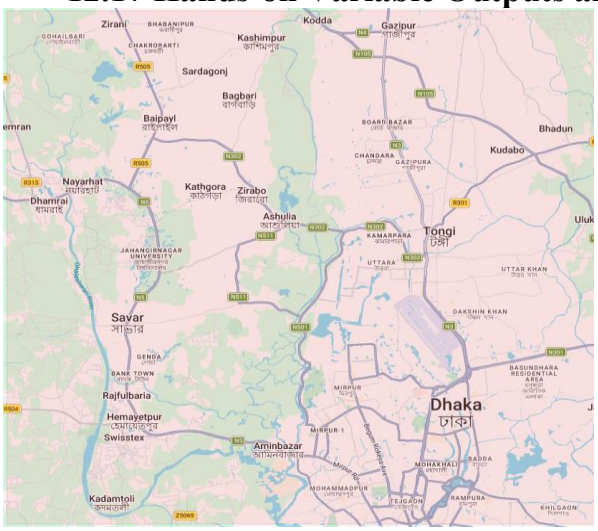
12.15 Expected Outputs

- Sentinel-2 RGB image
- NDVI map
- NDWI map
- EVI map
- SAVI map
- NDMI map
- NBR map
- Healthy vegetation mask
- NDVI change map
- Mean NDVI statistics

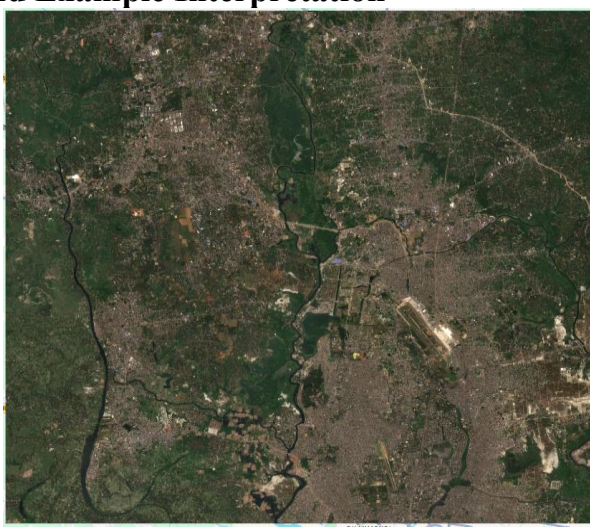
12.16 Common Errors and Solutions

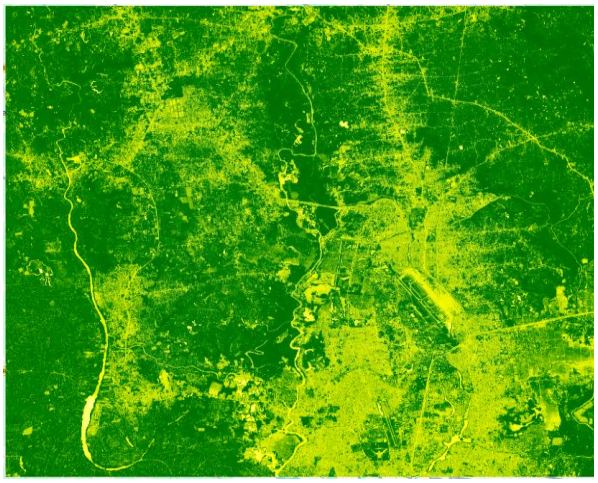
Error	Cause	Solution
Empty collection	Wrong date or AOI	Adjust filters
Black output	Incorrect visualization range	Modify min/max
Cloud contamination	No cloud masking	Apply mask
Export failure	Large AOI	Reduce export size

12.17 Hands-on Variable Outputs and Example Interpretation



Area of interest


Sentinel 2 True colour RGB image



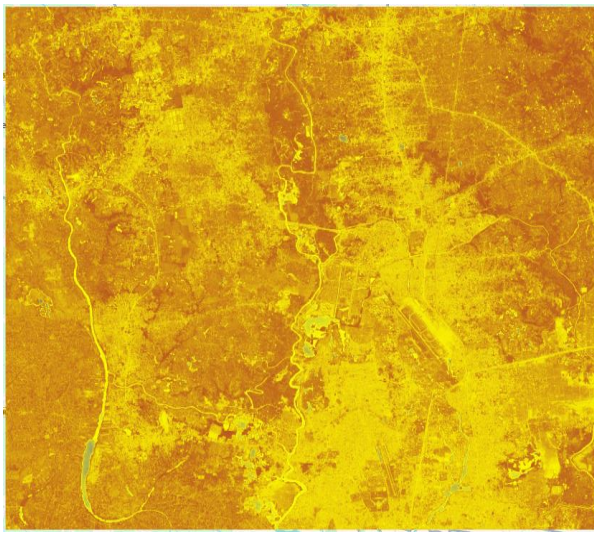
NDVI

Visualization Colors

- Blue → Water or non-vegetation
- White → Bare soil
- Yellow → Sparse vegetation
- Green/Dark Green → Healthy dense vegetation

Example Interpretation

- NDVI > 0.6
  - Healthy crops
  - High chlorophyll activity
  - Dense vegetation cover
- NDVI 0.3 – 0.6
  - Moderate vegetation
  - Possible moderate crop growth
- NDVI < 0.3
  - Sparse vegetation
  - Stressed crops
  - Bare agricultural land
- Negative NDVI
  - Water bodies
  - Clouds
  - Urban/built-up surfaces



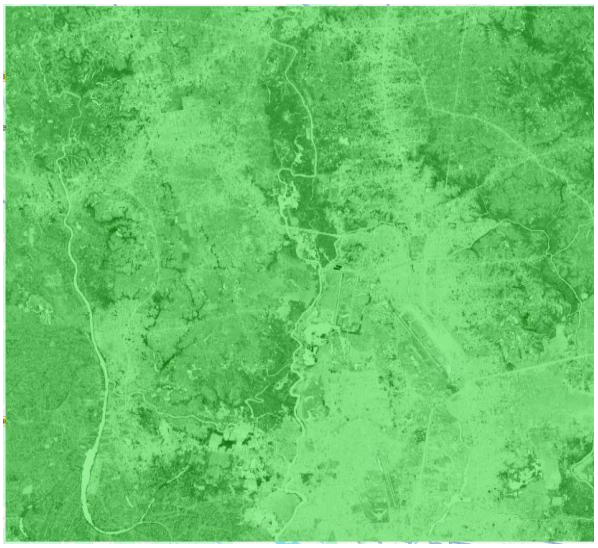
NDWI

Visualization Colors

- Brown → Dry areas
- Yellow → Moderate moisture
- Blue → High moisture/water

Example Interpretation

- High NDWI values
  - Water bodies
  - Well-irrigated vegetation
  - Moist soil conditions
- Low NDWI values
  - Dry vegetation
  - Water stress
  - Drought-prone areas



EVI

Visualization Colors

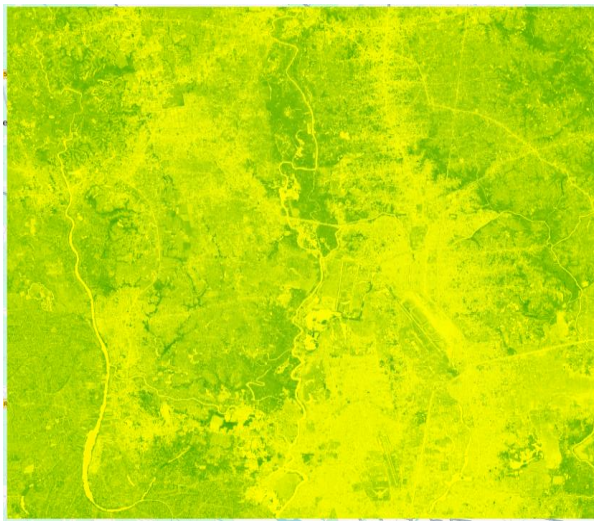
- White → Low vegetation
- Light Green → Moderate vegetation
- Dark Green → Dense healthy vegetation

Example Interpretation

- High EVI
  - Dense crop canopy
  - Strong vegetation growth
  - High biomass
- Low EVI
  - Sparse vegetation
  - Poor crop condition
  - Soil exposure

Additional Notes

- Performs better than NDVI in high biomass regions
- Reduces atmospheric and soil effects



SAVI

Visualization Colors

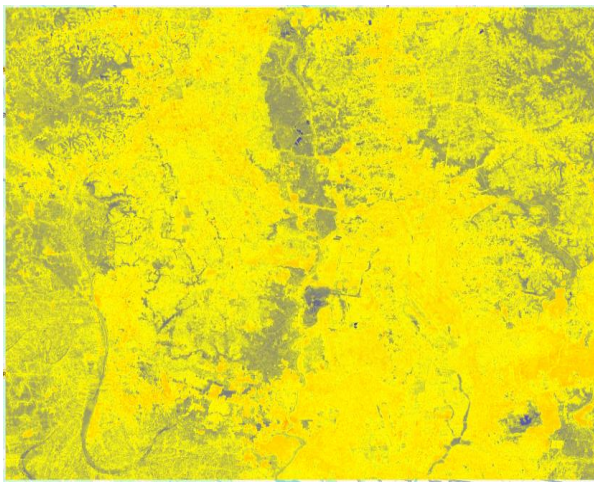
- Brown → Bare soil
- Yellow → Sparse vegetation
- Green → Healthy vegetation

Example Interpretation

- High SAVI
  - Healthy vegetation
  - Good crop cover
- Low SAVI
  - Exposed soil
  - Sparse crop coverage

Additional Notes

- Useful during:
  - Early crop stages
  - Semi-arid regions
  - Sparse vegetation conditions



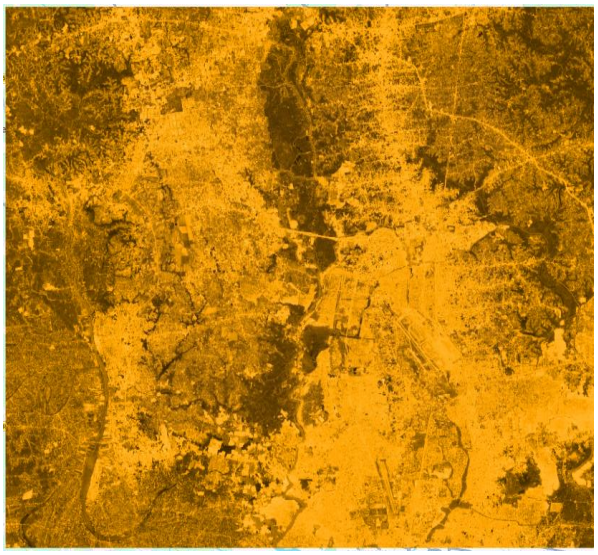
NDMI

Visualization Colors

- Red → Dry vegetation
- Yellow → Moderate moisture
- Blue → High vegetation moisture

Example Interpretation

- High NDMI
  - Adequate crop moisture
  - Healthy canopy water content
- Low NDMI
  - Water stress
  - Dry crop condition
  - Drought effects



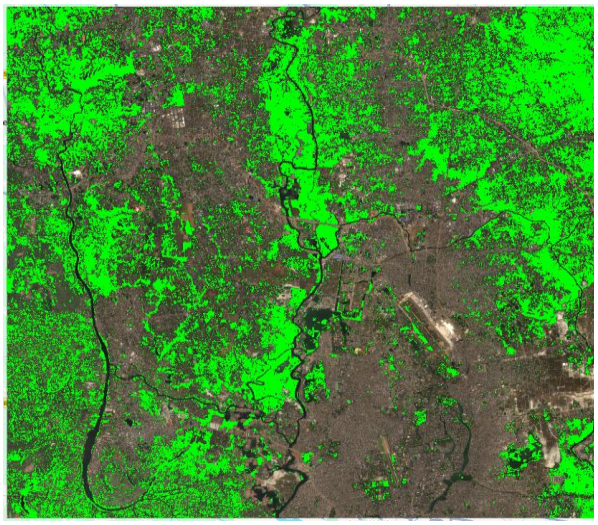
NBR

Visualization Colors

- White → Low disturbance
- Orange → Moderate disturbance
- Black → Burned/severely damaged areas

Example Interpretation

- High NBR
  - Healthy vegetation
  - Unburned areas
- Low NBR
  - Burned vegetation
  - Severe vegetation damage
  - Land disturbance



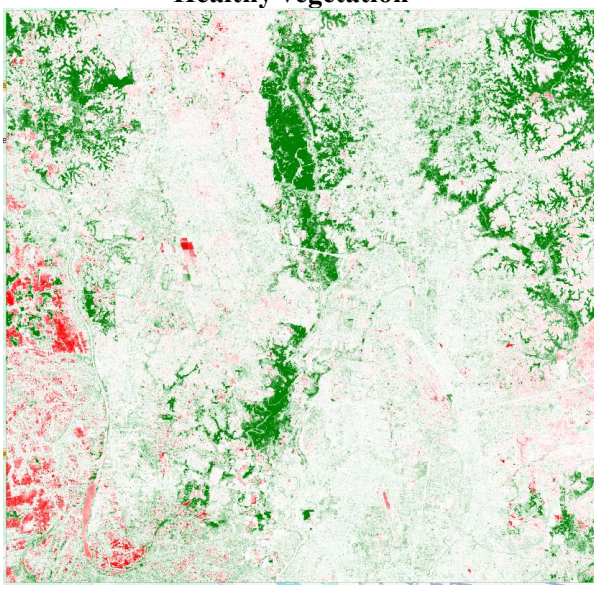
Healthy vegetation

Visualization

- Green pixels → Healthy vegetation only

Example Interpretation

- Areas highlighted in green indicate:
  - Healthy crop growth
  - Dense vegetation cover
  - Good photosynthetic activity
- Non-highlighted areas may indicate:
  - Bare soil
  - Stressed crops
  - Water bodies
  - Built-up land



NDVI Change

Visualization Colors

- Red → Decrease in vegetation
- White → Little/no change
- Green → Increase in vegetation

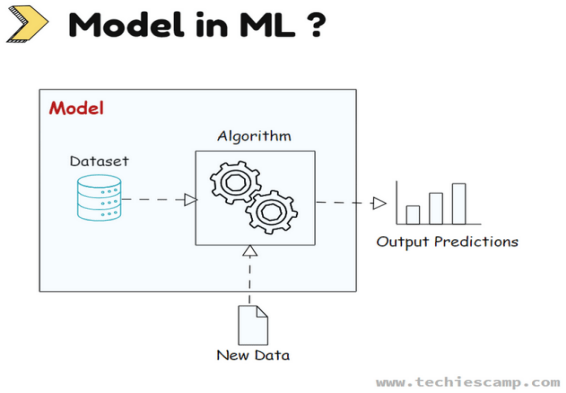
Example Interpretation

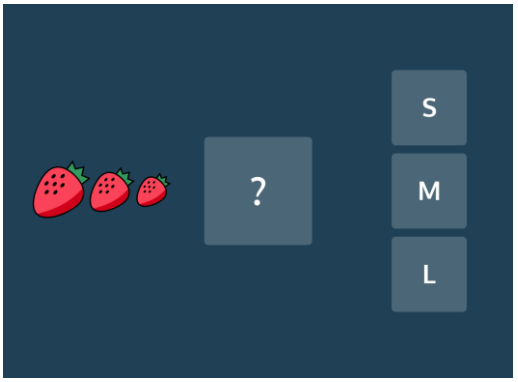
- Green Areas
  - Improved crop growth
  - Increased vegetation cover
  - Successful irrigation or rainfall
- Red Areas
  - Crop decline
  - Harvested fields
  - Water stress or drought impact
- White Areas
  - Stable vegetation condition

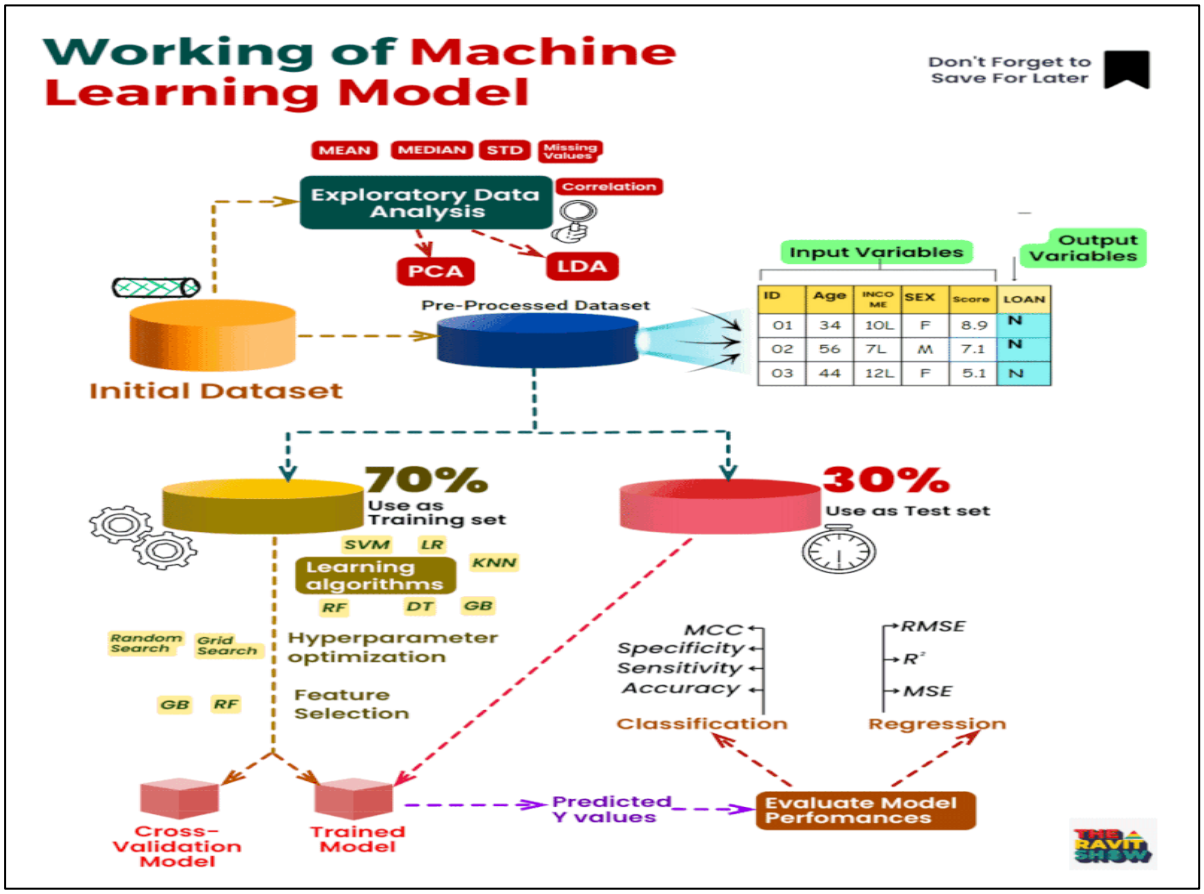
Machine Learning Techniques in GEE: Supervised Classification

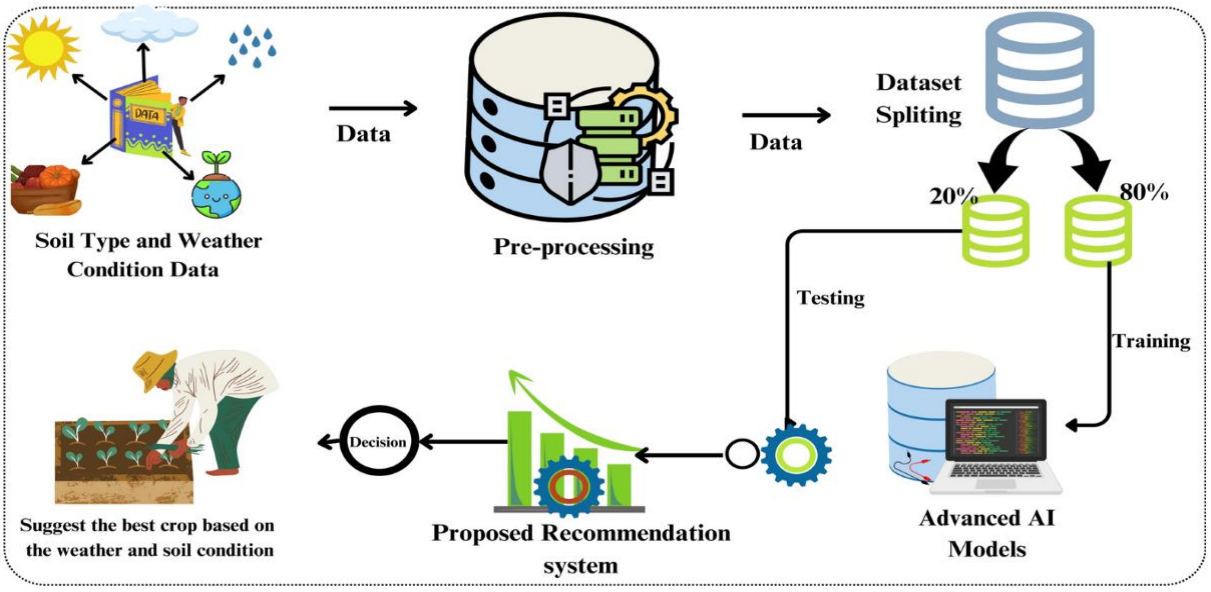
Mr. Al-Helal
Programmer (Computer & GIS Unit)
Bangladesh Agricultural Research Council
Email: al.helal@barc.gov.bd

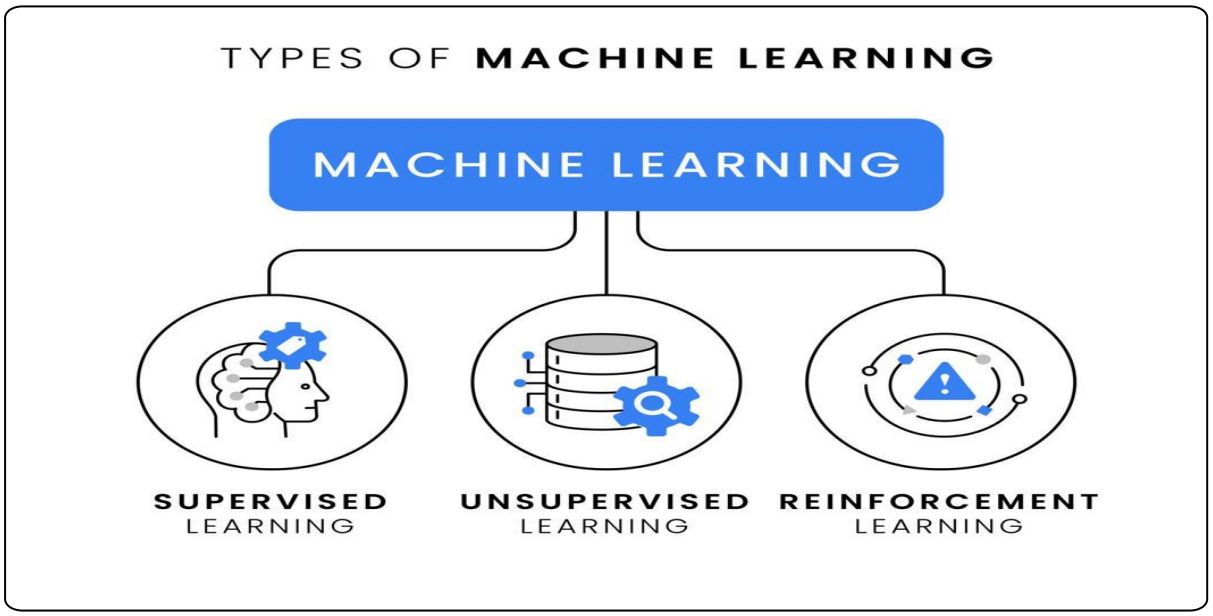
1. In Machine Learning, a Classifier is a tool or function that helps the Computer make similar decisions.

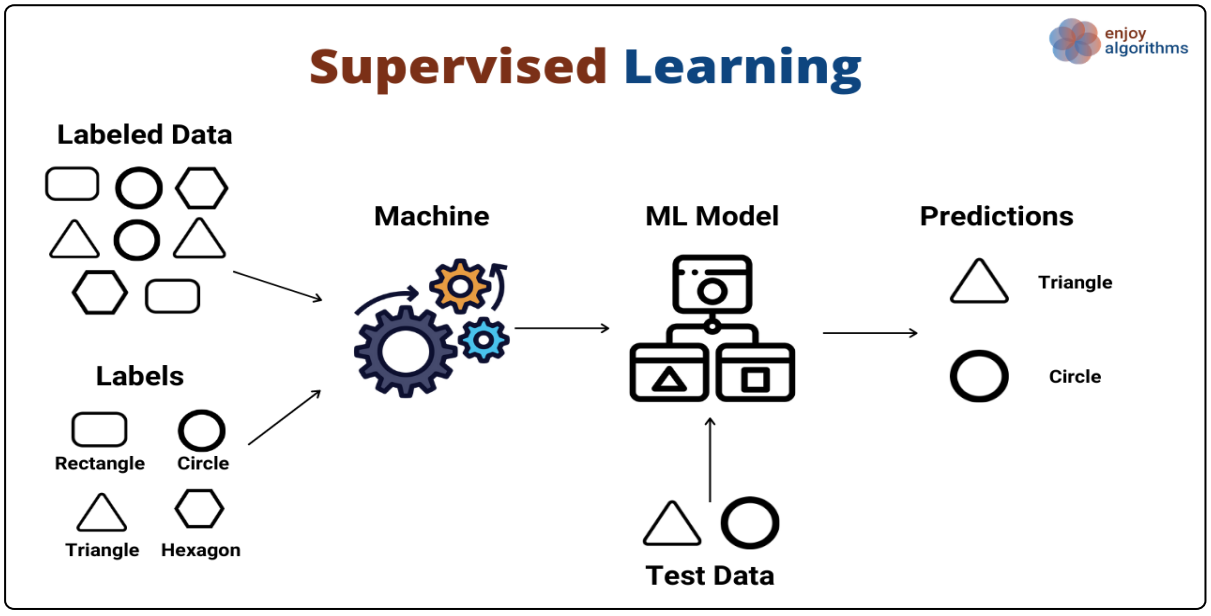


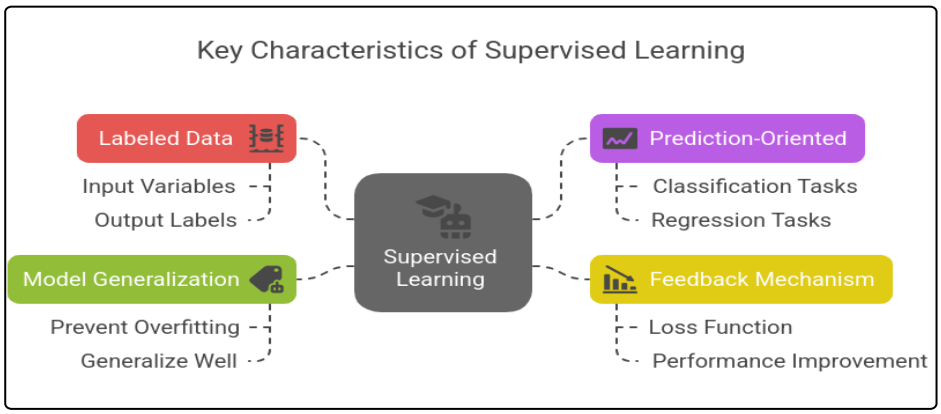





Soil Type and Weather Condition Data
Data
Pre-processing
Data
Dataset Spliting
20%
80%
Testing
Training
Advanced AI Models
Proposed Recommendation system
Decision
Suggest the best crop based on the weather and soil condition

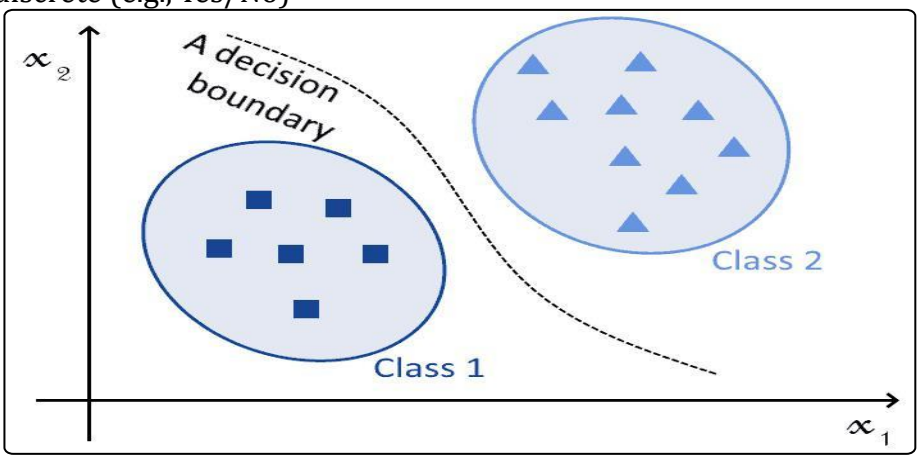

TYPES OF MACHINE LEARNING
MACHINE LEARNING
SUPERVISED LEARNING
UNSUPERVISED LEARNING
REINFORCEMENT LEARNING


Supervised Learning
enjoy algorithms
Labeled Data
Labels
Rectangle
Circle
Triangle
Hexagon
Machine
ML Model
Predictions
Triangle
Circle
Test Data

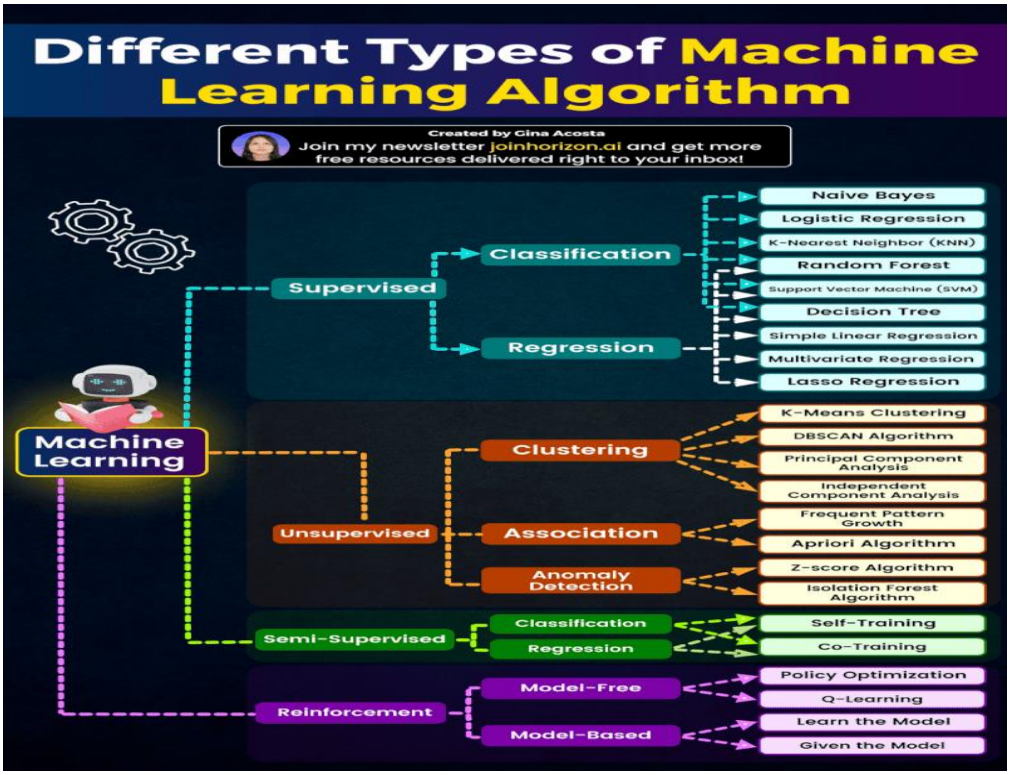


What is Classification Algorithm?

A type of supervised learning Predicts categories (labels)
Output is discrete (e.g., Yes/No)



Different Types of Machine Learning Algorithms



What is Decision Tree?

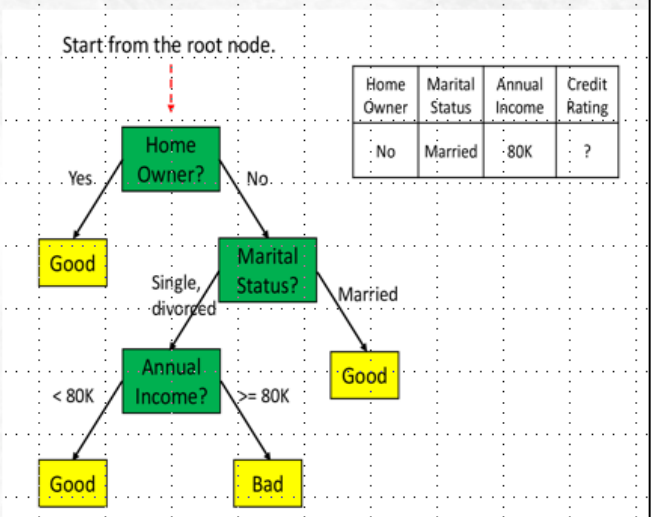
A supervised learning algorithm
Uses a tree-like structure
Makes decisions using if-else rules

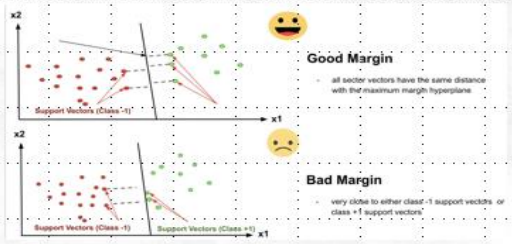
How It Works?

Select best feature (e.g., Age)
Split data into branches
Repeat splitting for each branch
Stop when final decision is reached

Applications:
Crop Diseases recognition, Medical diagnosis (Disease prediction), Spam email detection, Customer decision analysis

Decision Tree





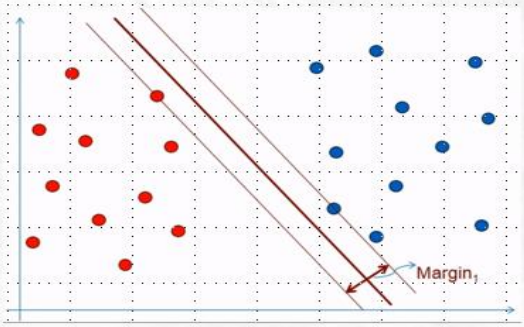
Support Vector Machine (SVM)

A supervised learning algorithm
Used for classification
Finds the best decision boundary (hyperplane)

How It Works

Plot data points
Find possible boundaries
Choose boundary with maximum margin
Use it to classify new data

Applications: Face detection,Text classification,Image classification,Bioinformatics



RANDOM FOREST

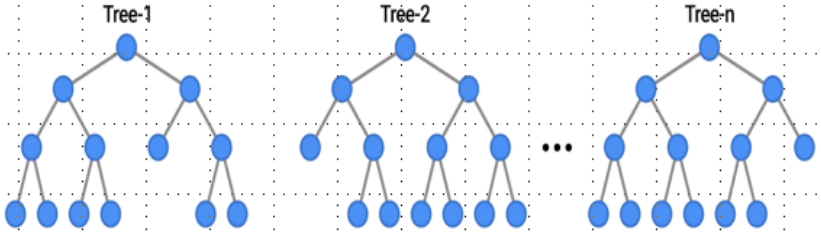
What is Ensemble Learning?

An ensemble learning algorithm
Uses multiple Decision Trees
Combines results for better accuracy

How it works?

Take random samples from dataset Train multiple decision trees Each tree makes a prediction
Final output = majority voting

EXAMPLES



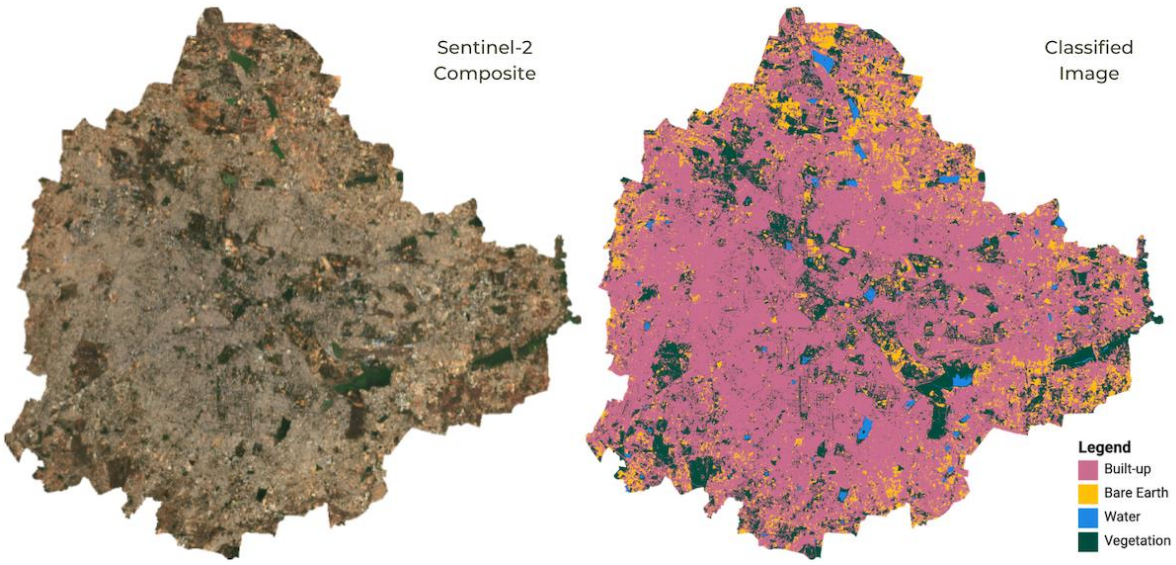
Supervised Classification

Introduction to Machine Learning and Supervised Classification

Supervised classification is arguably the most important classical machine learning techniques in remote sensing. Applications range from generating Land Use/Land Cover maps to change detection. Google Earth Engine is unique suited to do supervised classification at scale. The interactive nature of Earth Engine development allows for iterative development of supervised classification workflows by combining many different datasets into the model. This module covers basic supervised classification workflow, accuracy assessment, hyperparameter tuning and change detection.

01. Basic Supervised Classification

We will learn how to do a basic land cover classification using training samples collected from the Code Editor using the High Resolution basemap imagery provided by Google Maps. This method requires no prior training data and is quite effective to generate high quality classification samples anywhere in the world. The goal is to classify each source pixel into one of the following classes - urban, bare, water or vegetation. Using the drawing tools in the code editor, you create 4 new feature collection with points representing pixels of that class. Each feature collection has a property called landcover with values of 0, 1, 2 or 3 indicating whether the feature collection represents urban, bare, water or vegetation respectively. We then train a *Random Forest* classifier using these training set to build a model and apply it to all the pixels of the image to create a 4 class image.



Supervised Classification Output

02.

```
// BD Bangladesh location (Dhaka)
var geometry = ee.Geometry.Point([90.4125, 23.8103]);
Map.centerObject(geometry, 10);
```

```
// ========================================
// Sentinel-2 Image Collection
// ========================================
var s2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED');

// Efficient filtering (chain method)
var filtered = s2
  .filterBounds(geometry)
  .filterDate('2024-01-01', '2024-12-31')
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30))
  .select(['B2', 'B3', 'B4', 'B8']); // important bands only

print('Filtered Image Count:', filtered.size());

// ========================================
// Mean Composite Image (Time Series Average)
// ========================================
var meanImage = filtered.mean();

// RGB Visualization
var rgbVis = {
  min: 0,
  max: 3000,
  bands: ['B4', 'B3', 'B2']
};

Map.addLayer(meanImage.clip(geometry.buffer(5000)), rgbVis, 'Mean Composite
(BD)');
// Farm / Area boundary visualization
Map.addLayer(geometry, {color: 'red'}, 'Point Location');

// ========================================
// Region Statistics (Efficient reduceRegion)
// ========================================
var stats = meanImage.reduceRegion({
  reducer: ee.Reducer.mean(),
  geometry: geometry.buffer(1000), // buffer for meaningful stats
  scale: 10,
  maxPixels: 1e13
});

print('Band Statistics (Mean Values):', stats);
// Extract specific band value
print('Mean B4 (Red Band):', stats.getNumber('B4'));//
========================================
// Landcover Classification - Bangladesh
// Sentinel-2 + Cloud Score+ + Random Forest
// ========================================
```

```
// BD Bangladesh Area (Dhaka Region Example)
var geometry = ee.Geometry.Point([90.4125, 23.8103]);
Map.centerObject(geometry, 10);

// ========================================
// Sentinel-2 Collection
// ========================================
var s2 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED');

// Time period (Bangladesh analysis year)
var year = 2024;
var startDate = ee.Date.fromYMD(year, 1, 1);
var endDate = startDate.advance(1, 'year');

// Filter Sentinel-2
var filtered = s2
  .filterBounds(geometry)
  .filterDate(startDate, endDate)
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30));

// ========================================
// Cloud Score+ Integration
// ========================================
var csPlus = ee.ImageCollection('GOOGLE/CLOUD_SCORE_PLUS/V1/S2_HARMONIZED');
var csBands = csPlus.first().bandNames();

var s2WithCs = filtered.linkCollection(csPlus, csBands);

// Cloud masking function
function maskLowQA(image) {
  return image.updateMask(
    image.select('cs').gte(0.5)
  );
}

// Apply mask and select bands
var masked = s2WithCs
  .map(maskLowQA)
  .select(['B2', 'B3', 'B4', 'B8']);

// ========================================
// Composite Image (Best practice)
// ========================================
var composite = masked.median().clip(geometry.buffer(5000));

// RGB Visualization
var rgbVis = {
  min: 0,
```

```
  max: 3000,
  bands: ['B4', 'B3', 'B2']
};

Map.addLayer(composite, rgbVis, 'Sentinel-2 Composite (BD)');

// ========================================
// Training Data (GCPs - replace with BD samples)
// ========================================

// Example structure (YOU should replace with BD training data)
var urban = ee.FeatureCollection([]);
var bare = ee.FeatureCollection([]);
var water = ee.FeatureCollection([]);
var vegetation = ee.FeatureCollection([]);

// Merge classes
var gcps = urban
  .merge(bare)
  .merge(water)
  .merge(vegetation);

// ========================================
// Training Sample Extraction
// ========================================
var training = composite.sampleRegions({
  collection: gcps,
  properties: ['landcover'],
  scale: 10
});

// ========================================
// Random Forest Classifier
// ========================================
var classifier = ee.Classifier.smileRandomForest(100).train({
  features: training,
  classProperty: 'landcover',
  inputProperties: composite.bandNames()
});

// ========================================
// Classification
// ========================================
var classified = composite.classify(classifier);

// Color palette (Bangladesh standard landcover)
var palette = [
  '#d73027', // Urban
```

```
  '#fdae61', // Bare land
  '#4575b4', // Water
  '#1a9850'  // Vegetation
];
Map.addLayer(
  classified,
  {min: 0, max: 3, palette: palette},
  'Landcover Classification (BD)'
);
// ========================================
// Optional: Show training points
// ========================================
Map.addLayer(gcps, {}, 'Training Points');
// ========================================
// Center Map
// ========================================
Map.centerObject(composite, 10);
```

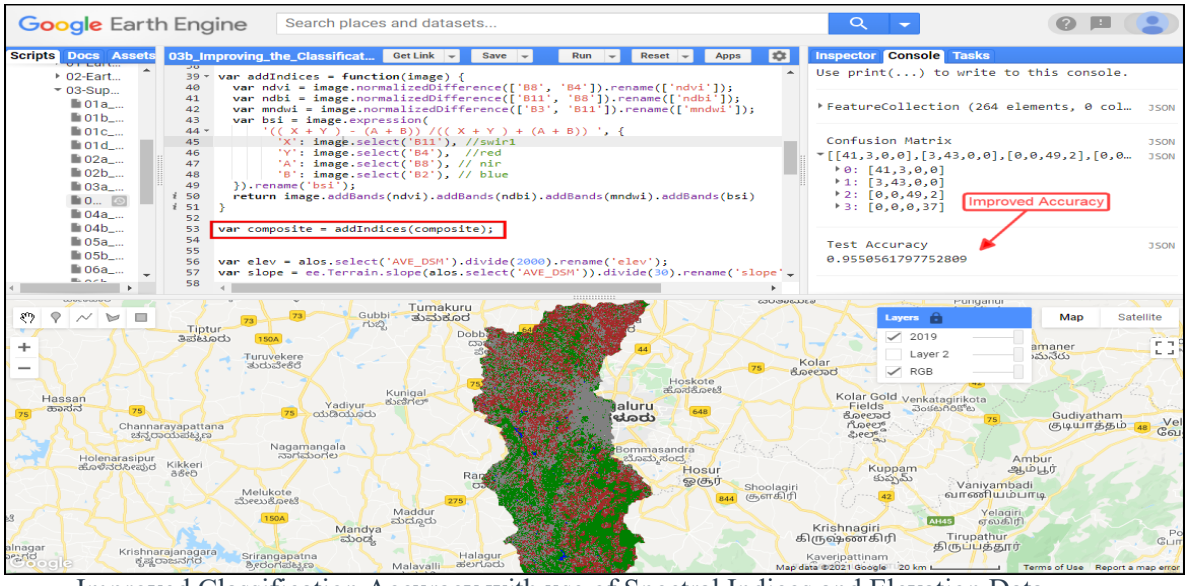
03. Feature Engineering

We can make more a effective machine learning model by adding more relevant inputs to the classification task. This process is known as *Feature Engineering*. Instead of using just the spectral bands as inputs to the model, we can add transformations (such as Spectral Indicies) and other relevant inputs (such as Elevation and Slope). This new *Features* help the classifier learn the patterns in the data more effectively.

Here we take the same example as before and augment it with the following new features:

- *Spectral Indices*: We add bands for different spectral indices such as - NDVI, NDBI, MNDWI and BSI.
- *Elevation and Slope*: We also add slope and elevation bands from the ALOS DEM.

  After adding bands with this new inputs, our training features have more parameters and result is a much improved classification.



Improved Classification Accuracy with use of Spectral Indices and Elevation Data

04. Exporting Classification Results

When working with complex classifiers over large regions, you may get a *User memory limit exceeded* or *Computation timed out* error in the Code Editor. The reason for this is that there is a fixed time limit and smaller memory allocated for code that is run with the *On-Demand Computation* mode. For larger computations, you can use the *Batch* mode with the Export functions. Exports run in the background and can run longer than 5-minutes time allocated to the computation code run from the Code Editor. This allows you to process very large and complex datasets.

```
// ========================================
// Satellite Embedding Classification (BD)
// KNN + Cloud Score+ + Accuracy + Export
// ========================================

// BD Bangladesh study area (replace with your basin if needed)
var geometry = ee.Geometry.Point([90.4125, 23.8103]);
Map.centerObject(geometry, 10);

// ========================================
// Load GCP Training Data
// ========================================
var gcps = ee.FeatureCollection(
  'projects/spatialthoughts/assets/e2e/exported_gcps'
);

// Add random column for train/test split
gcps = gcps.randomColumn('random');

// Split dataset
var trainingGcp = gcps.filter(ee.Filter.lt('random', 0.6));
var validationGcp = gcps.filter(ee.Filter.gte('random', 0.6));

// ========================================
// Sentinel-2 Data
// ========================================
var year = 2024;
var startDate = ee.Date.fromYMD(year, 1, 1);
var endDate = startDate.advance(1, 'year');

var s2 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED');

var filtered = s2
  .filterBounds(geometry)
  .filterDate(startDate, endDate)
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30));

// ========================================
// Cloud Score+ Masking
// ========================================
```

```
var csPlus = ee.ImageCollection('GOOGLE/CLOUD_SCORE_PLUS/V1/S2_HARMONIZED');
var csBands = csPlus.first().bandNames();

var s2WithCs = filtered.linkCollection(csPlus, csBands);

function maskLowQA(image) {
  return image.updateMask(
    image.select('cs').gte(0.5)
  );
}

var masked = s2WithCs
  .map(maskLowQA)
  .select('B.*');

var composite = masked.median().clip(geometry.buffer(5000));

// ========================================
// Satellite Embedding Features
// ========================================
var embeddings = ee.ImageCollection(
  'GOOGLE/SATELLITE_EMBEDDING/V1/ANNUAL'
);

var embeddingImage = embeddings
  .filterBounds(geometry)
  .filterDate(startDate, endDate)
  .mosaic();

// ========================================
// Training Data Sampling
// ========================================
var training = embeddingImage.sampleRegions({
  collection: trainingGcp,
  properties: ['landcover'],
  scale: 10,
  tileScale: 16
});

// ========================================
// KNN Classifier
// ========================================
var classifier = ee.Classifier.smileKNN().train({
  features: training,
  classProperty: 'landcover',
  inputProperties: embeddingImage.bandNames()
});
```

```
// ========================================
// Classification
// ========================================
var classified = embeddingImage.classify(classifier);

// Visualization
var palette = ['#cc6d8f', '#ffc107', '#1e88e5', '#004d40'];

Map.addLayer(
  classified.clip(geometry),
  {min: 0, max: 3, palette: palette},
  'Landcover (KNN - BD)'
);

// ========================================
// Accuracy Assessment
// ========================================
var test = classified.sampleRegions({
  collection: validationGcp,
  properties: ['landcover'],
  scale: 10,
  tileScale: 16
});

var confusionMatrix = test.errorMatrix('landcover', 'classification');

print('Confusion Matrix:', confusionMatrix);
print('Overall Accuracy:', confusionMatrix.accuracy());

// ========================================
// Export Accuracy Table
// ========================================
var accuracyFC = ee.FeatureCollection([
  ee.Feature(null, {
    accuracy: confusionMatrix.accuracy(),
    kappa: confusionMatrix.kappa(),
    matrix: confusionMatrix.array()
  })
]);

Export.table.toDrive({
  collection: accuracyFC,
  description: 'BD_KNN_Accuracy',
  folder: 'earthengine',
  fileNamePrefix: 'knn_accuracy_bd',
  fileFormat: 'CSV'
});
```

```
// ========================================
// Export Classified Image
// ========================================
Export.image.toDrive({
  image: classified.clip(geometry).toFloat(),
  description: 'BD_KNN_Classified',
  folder: 'earthengine',
  fileNamePrefix: 'classified_bd',
  region: geometry,
  scale: 10,
  maxPixels: 1e13
});

// ========================================
// Export to Asset (optional)
// ========================================
var assetPath = 'projects/your_project/assets/bd_embeddings_classification';

Export.image.toAsset({
  image: classified.clip(geometry),
  description: 'BD_Classified_Asset',
  assetId: assetPath,
  region: geometry,
  scale: 10,
  pyramidingPolicy: 'MODE',
  maxPixels: 1e13
});
```

05. Visualize Exported Images

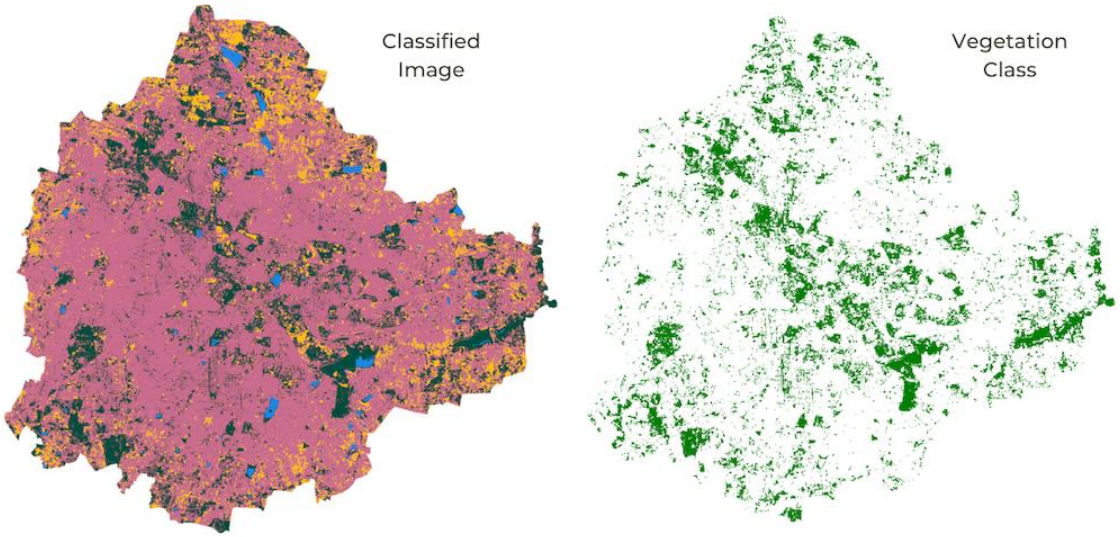
```
// ========================================
// Import & Visualize Exported Assets
// Bangladesh Standard GEE Workflow
// ========================================

// ========================================
// Asset Folder (Update if needed)
// ========================================
var assetFolder = 'projects/spatialthoughts/assets/e2e/';

// ========================================
// Import Composite Image
// ========================================
var composite = ee.Image(assetFolder + 'composite');
```

```
// ========================================
// Import Classified Image
// ========================================
var classified = ee.Image(assetFolder + 'embeddings_classification');

// Get geometry from image
var geometry = classified.geometry();
Map.centerObject(geometry, 10);

// ========================================
// RGB Visualization (Sentinel-2)
// ========================================
var rgbVis = {
  min: 0,
  max: 3000,
  bands: ['B4', 'B3', 'B2']
};

Map.addLayer(composite, rgbVis, 'Sentinel-2 Composite (BD)');

// ========================================
// Classified Map Visualization
// ========================================
var palette = [
  '#cc6d8f', // Class 0
  '#ffc107', // Class 1
  '#1e88e5', // Class 2
  '#004d40'  // Class 3
];

Map.addLayer(
  classified,
  {min: 0, max: 3, palette: palette},
  'Landcover Classification (BD)'
);

// ========================================
// Optional Boundary Outline
// ========================================
Map.addLayer(
  geometry,
  {color: 'red'},
  'Study Area Boundary'
);
```

06. Calculating Area

Now that we have the results of our classification, we will learn how to calculate the area for pixels in each class. The functions used for area computations are different for vectors and raster data.

- **Area of Polygons**: Calculating area for polygons is done using the area() function. It computes area on a sphere (ignoring the ellipsoid flattening) and gives you the area in square meters. You can optionally supply proj and a non-zero maxError parameters to calculate area in a specific projected CRS. For example, area({proj:'EPSG:32643', maxError: 1}) will calculate the area of the polygon after reprojecting it to the *WGS 84/UTM Zone 43* CRS with a tolerance of 1 meter.
- **Area of Image Pixels**: Area of image pixels is computed using the ee.Image.pixelArea() function. This function computes the area inside the 4 corners of each pixel using the WGS84 ellipsoid. The ee.Image.pixelArea() function uses a custom equal-area projection for area calculation. The result is area in square meters regardless of the projection of the input image.



Calculating Green Cover from Classified Image

```
// ========================================
// Area Calculation: Bangladesh Landcover
// GEE Standard Workflow
// ========================================

// BD Study Area (Bangladesh - example: Dhaka region)
var region = ee.Geometry.Point([90.4125, 23.8103]);
Map.centerObject(region, 10);

// You can replace this with your Bangladesh classified asset
var classified = ee.Image('users/your_username/bd_classified');

// Optional: Study boundary (Bangladesh administrative or basin)
```

```
var boundary = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017')
  .filter(ee.Filter.eq('country_na', 'Bangladesh'));

var geometry = boundary.geometry();

// Display boundary
Map.addLayer(boundary, {color: 'blue'}, 'Bangladesh Boundary');

// ========================================
// Landcover Visualization
// ========================================
var palette = ['#d73027', '#fdae61', '#4575b4', '#1a9850'];

Map.addLayer(
  classified.clip(geometry),
  {min: 0, max: 3, palette: palette},
  'Landcover (Bangladesh)'
);

// ========================================
// Bangladesh Total Area (km²)
// ========================================
var bangladeshAreaSqKm = ee.Number(
  geometry.area()
).divide(1e6);

print('Bangladesh Area (sq km):', bangladeshAreaSqKm);

// ========================================
// Vegetation Area (Class 3)
// ========================================
var vegetation = classified.eq(3);

// Display vegetation mask
Map.addLayer(
  vegetation.clip(geometry),
  {min: 0, max: 1, palette: ['white', 'green']},
  'Vegetation'
);

// Pixel area calculation
var areaImage = vegetation.multiply(ee.Image.pixelArea());

// Sum vegetation area over Bangladesh
var vegArea = areaImage.reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: geometry,
  scale: 10,
```

```
  maxPixels: 1e13
});
// Convert to square kilometers
var vegAreaSqKm = ee.Number(
  vegArea.get('classification')
).divide(1e6);
print('Vegetation Area in Bangladesh (sq km):', vegAreaSqKm);
```

Change Detection

Introduction to Change Detection

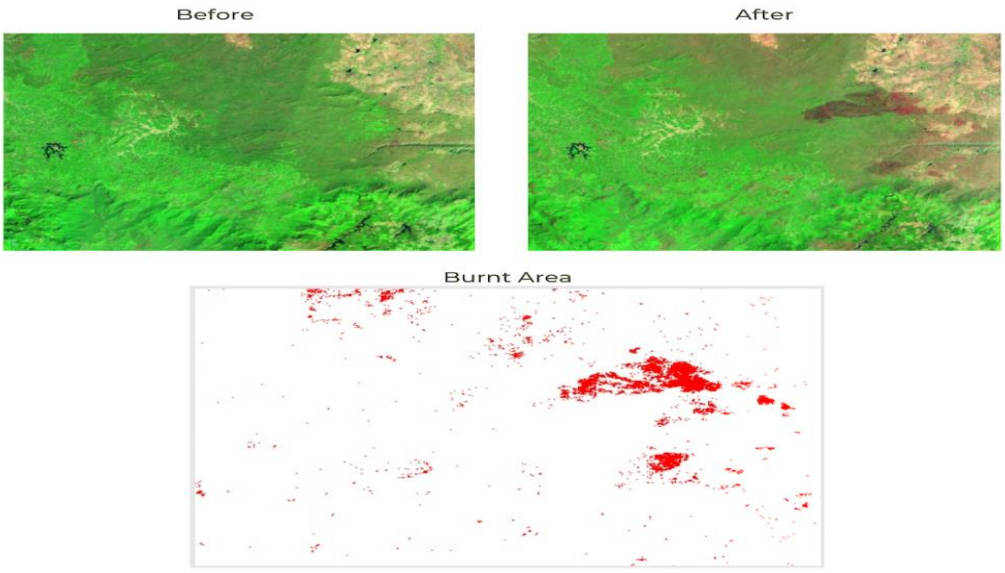
Many earth observation datasets are available at regular intervals over long periods of time. This enables us to detect changes on the Earth's surface. Change detection technique in remote sensing fall in the following categories

- **Single Band Change**: Measuring change in a single band image or a spectral index using a threshold
- **Multi Band Change**: Measuring spectral distance and spectral angle between two multiband images
- **Classification of Change**: One-pass classification using stacked image containing bands from before and after an event
- **Post Classification Comparison**: Comparing two classified images and computing class transitions

01. Spectral Index Change

Many types of change can be detected by measuring the change in a spectral index and applying a threshold. This technique is suitable when there is a suitable spectral index is available for the type of change you are interested in detecting.

Here we apply this technique to map the extent and severity of a forest fire. The **Normalized Burn Ratio (NBR)** is an index that is designed to highlight burnt vegetation areas. We compute the NBR for before and after images. Then we apply a suitable threshold to find burnt areas.



Spectral Index Change Detection

```
// ========================================
// Fire Damage Assessment using NBR / dNBR
// Bangladesh Standard GEE Workflow
// ========================================

// BD Example Study Area (Bangladesh - Sundarbans region proxy)
var geometry = ee.Geometry.Polygon([
  [
    [89.0, 21.5],
    [89.0, 21.0],
    [89.8, 21.0],
    [89.8, 21.5]
  ]
]);

Map.centerObject(geometry, 10);

// Fire event period (Bangladesh fire scenario example)
var fireStart = ee.Date('2024-02-01');
var fireEnd = ee.Date('2024-02-10');

// ========================================
// Sentinel-2 Collection
// ========================================
var s2 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED');

// Efficient filtering
var filtered = s2
  .filterBounds(geometry)
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30))
  .select(['B2', 'B3', 'B4', 'B8', 'B12']);

// ========================================
// Cloud Score+ Masking (Best Practice)
// ========================================
var csPlus = ee.ImageCollection('GOOGLE/CLOUD_SCORE_PLUS/V1/S2_HARMONIZED');
var csBands = csPlus.first().bandNames();

var s2WithCs = filtered.linkCollection(csPlus, csBands);

function maskLowQA(image) {
  return image.updateMask(
    image.select('cs').gte(0.5)
  );
}

var masked = s2WithCs.map(maskLowQA);
```

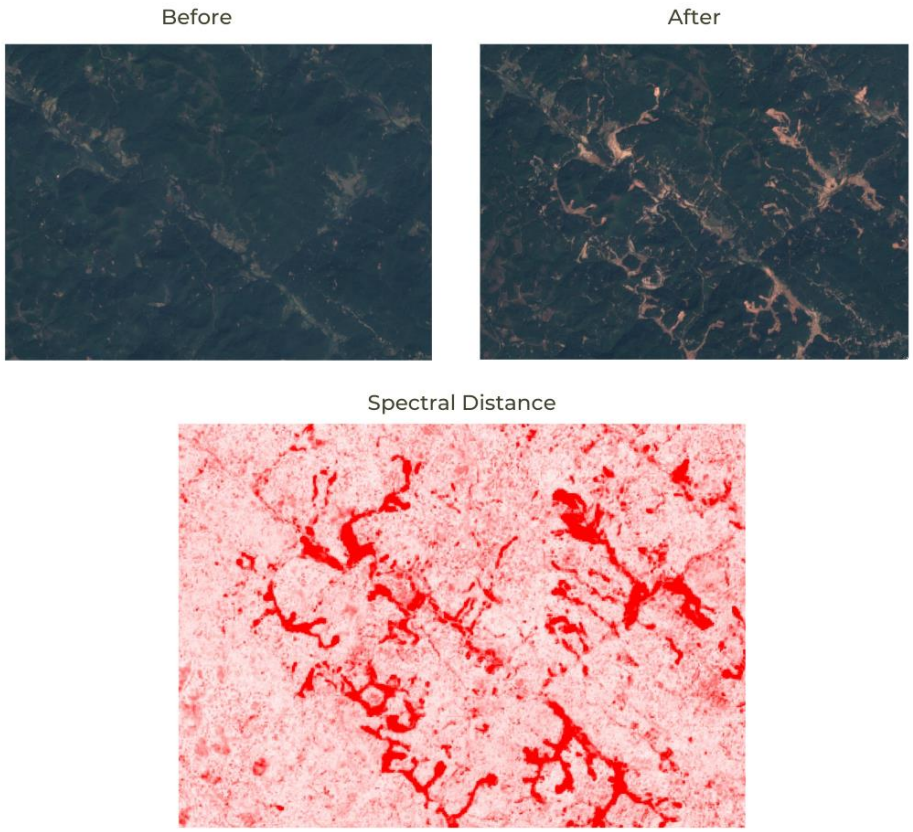
```
// ========================================
// Pre and Post Fire Composites
// ========================================
var before = masked
  .filterDate(fireStart.advance(-2, 'month'), fireStart)
  .median()
  .clip(geometry);

var after = masked
  .filterDate(fireEnd, fireEnd.advance(1, 'month'))
  .median()
  .clip(geometry);

// ========================================
// Visualization (False Color - Burn Detection)
// ========================================
var swirVis = {
  min: 0,
  max: 3000,
  bands: ['B12', 'B8', 'B4']
};

Map.addLayer(before, swirVis, 'Before Fire (BD)');
Map.addLayer(after, swirVis, 'After Fire (BD)');

// ========================================
// NBR Calculation
// NBR = (NIR - SWIR) / (NIR + SWIR)
// ========================================
function addNBR(image) {
  var nbr = image.normalizedDifference(['B8', 'B12'])
    .rename('NBR');
  return image.addBands(nbr);
}

var beforeNBR = addNBR(before).select('NBR');
var afterNBR = addNBR(after).select('NBR');

// ========================================
// Visualization of NBR
// ========================================
var nbrVis = {
  min: -0.5,
  max: 0.5,
  palette: ['white', 'black']
};

Map.addLayer(beforeNBR, nbrVis, 'Pre-Fire NBR');
```

```
Map.addLayer(afterNBR, nbrVis, 'Post-Fire NBR');

// ========================================
// dNBR (Burn Severity Index)
// ========================================
var dNBR = beforeNBR.subtract(afterNBR);

var burnedArea = dNBR.gt(0.3);

Map.addLayer(
  dNBR,
  {min: -0.5, max: 0.5, palette: ['blue', 'white', 'red']},
  'dNBR (Change)'
);

Map.addLayer(
  burnedArea,
  {min: 0, max: 1, palette: ['white', 'red']},
  'Burned Area (BD)'
);

// ========================================
// Burned Area Statistics (Optional)
// ========================================
var areaImage = burnedArea.multiply(ee.Image.pixelArea());

var stats = areaImage.reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: geometry,
  scale: 10,
  maxPixels: 1e13
});

print('Burned Area (sq meters):', stats);
print('Burned Area (sq km):',
  ee.Number(stats.get('constant')).divide(1e6)
);
```

02. Spectral Distance Change

When you want to detect changes from multi-band images, a useful technique is to compute the Spectral Distance and Spectral Angle between the two images. Pixels that exhibit a large change will have a larger distance compared to those that did not change. This technique is particularly useful when there are no suitable index to detect the change. It can be applied to detect change after natural disasters or human conflicts.

Here we use this technique to detect landslides using before/after composites. You may learn more about this technique at Craig D'Souza's Change Detection presentation.



Spectral Distance Change Detection

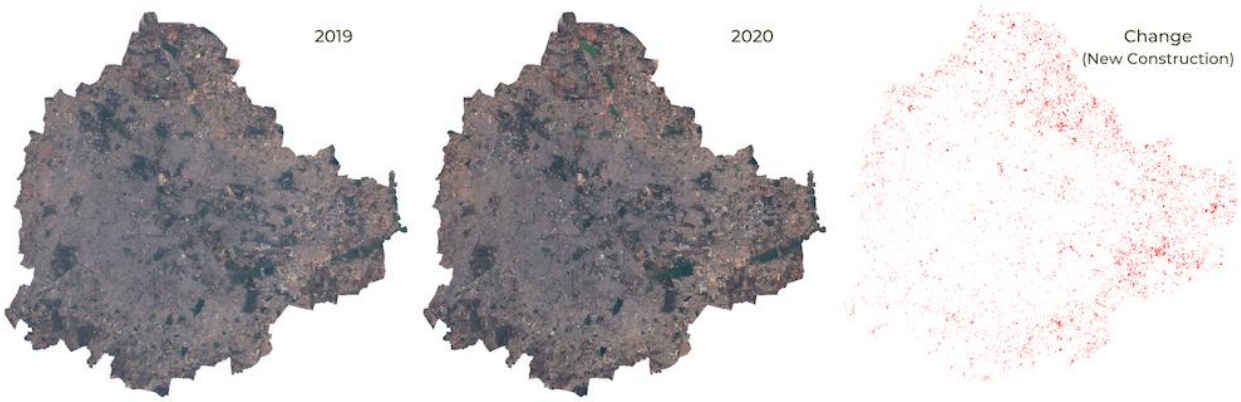
```
// ========================================
// Spectral Change Detection (SAM + Euclidean)
// Bangladesh Standard GEE Workflow
// ========================================

// BD Study Area (Bangladesh - sample coastal/urban zone)
var geometry = ee.Geometry.Polygon([
  [
    [90.35, 23.85],
    [90.35, 23.70],
    [90.55, 23.70],
    [90.55, 23.85]
  ]
]);

Map.centerObject(geometry, 10);

// ========================================
// Sentinel-2 Collection
// ========================================
var s2 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED');

// Filter (efficient pipeline)
var filtered = s2
  .filterBounds(geometry)
```

```
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30))
  .select('B.*');

// ========================================
// Cloud Score+ Masking
// ========================================
var csPlus = ee.ImageCollection('GOOGLE/CLOUD_SCORE_PLUS/V1/S2_HARMONIZED');
var csBands = csPlus.first().bandNames();

var s2WithCs = filtered.linkCollection(csPlus, csBands);

function maskLowQA(image) {
  return image.updateMask(
    image.select('cs').gte(0.65)
  );
}

var masked = s2WithCs.map(maskLowQA);

// ========================================
// Incident Date (Bangladesh scenario)
// ========================================
var dateOfIncident = ee.Date('2024-08-15');

// ========================================
// Pre-event Composite
// ========================================
var before = masked
  .filterDate(dateOfIncident.advance(-2, 'year'), dateOfIncident)
  .filter(ee.Filter.calendarRange(6, 10, 'month')) // monsoon control
  .median()
  .clip(geometry);

// ========================================
// Post-event Composite
// ========================================
var after = masked
  .filterDate(dateOfIncident, dateOfIncident.advance(1, 'month'))
  .median()
  .clip(geometry);

// ========================================
// Visualization
// ========================================
var rgbVis = {
  min: 0,
  max: 3000,
  bands: ['B4', 'B3', 'B2']
```

```
};

Map.addLayer(before, rgbVis, 'Before Event (BD)');
Map.addLayer(after, rgbVis, 'After Event (BD)');

// ========================================
// Spectral Angle Mapper (SAM)
// ========================================
var sam = after.spectralDistance(before, 'sam');

Map.addLayer(
  sam,
  {min: 0, max: 1, palette: ['white', 'purple']},
  'Spectral Angle (SAM)'
);

// ========================================
// Euclidean Spectral Distance
// ========================================
var sed = after.spectralDistance(before, 'sed');
var euclidean = sed.sqrt();

Map.addLayer(
  euclidean,
  {min: 0, max: 1500, palette: ['white', 'red']},
  'Spectral Distance'
);

// ========================================
// Optional: High Change Detection Mask
// ========================================
var changeMask = euclidean.gt(500);

Map.addLayer(
  changeMask,
  {min: 0, max: 1, palette: ['white', 'red']},
  'High Spectral Change'
);
```

03. Direct Classification of Change

This technique of change detection is also known as *One-pass Classification* or *Direct Multi-date Classification*. Here we create a single stacked image containing bands from before and after images. We train a classifier with training data sampled from the stacked image and apply the classifier on the stacked image to find all change pixels.



All pixels that changed from bare ground to built-up

```
// ========================================
// Change Detection using Sentinel-2 (RF)
// Bangladesh Standard GEE Workflow
// ========================================

// BD Study Area (Bangladesh example - Dhaka region)
var geometry = ee.Geometry.Point([90.4125, 23.8103]);
Map.centerObject(geometry, 10);

// ========================================
// Sentinel-2 Collection
// ========================================
var s2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED');

// Filter by location
var filtered = s2.filterBounds(geometry);

// ========================================
// Cloud Score+ Masking
// ========================================
var csPlus = ee.ImageCollection('GOOGLE/CLOUD_SCORE_PLUS/V1/S2_HARMONIZED');
var csBands = csPlus.first().bandNames();

var s2WithCs = filtered.linkCollection(csPlus, csBands);

function maskLowQA(image) {
  return image.updateMask(
    image.select('cs').gte(0.5)
  );
}

var masked = s2WithCs.map(maskLowQA).select('B.*');

// ========================================
```

```
// Pre-Change Image (2019)
// ========================================
var image2019 = masked
  .filterDate('2019-01-01', '2019-02-01')
  .median()
  .clip(geometry);

// ========================================
// Post-Change Image (2020)
// ========================================
var image2020 = masked
  .filterDate('2020-01-01', '2020-02-01')
  .median()
  .clip(geometry);

// ========================================
// Visualization
// ========================================
var rgbVis = {
  min: 0,
  max: 3000,
  bands: ['B4', 'B3', 'B2']
};

Map.addLayer(image2019, rgbVis, '2019 Image (BD)');
Map.addLayer(image2020, rgbVis, '2020 Image (BD)');

// ========================================
// Stack Images (Bi-temporal)
// ========================================
var stacked = image2019.addBands(image2020);

// ========================================
// Training Data (Change / No Change)
// ========================================
// Replace with your BD GCPs if available
var change = ee.FeatureCollection([]);
var nochange = ee.FeatureCollection([]);

var samples = change.merge(nochange);

// Sample training data
var training = stacked.sampleRegions({
  collection: samples,
  properties: ['class'],
  scale: 10,
  tileScale: 16
});
```

```
// ========================================
// Random Forest Classifier
// ========================================
var classifier = ee.Classifier.smileRandomForest(50).train({
  features: training,
  classProperty: 'class',
  inputProperties: stacked.bandNames()
});

// ========================================
// Classification
// ========================================
var classified = stacked.classify(classifier);

// ========================================
// Display Result
// ========================================
Map.addLayer(
  classified.clip(geometry),
  {min: 0, max: 1, palette: ['white', 'red']},
  'Change Detection (BD)'
);
```

04. Post-classification Comparison

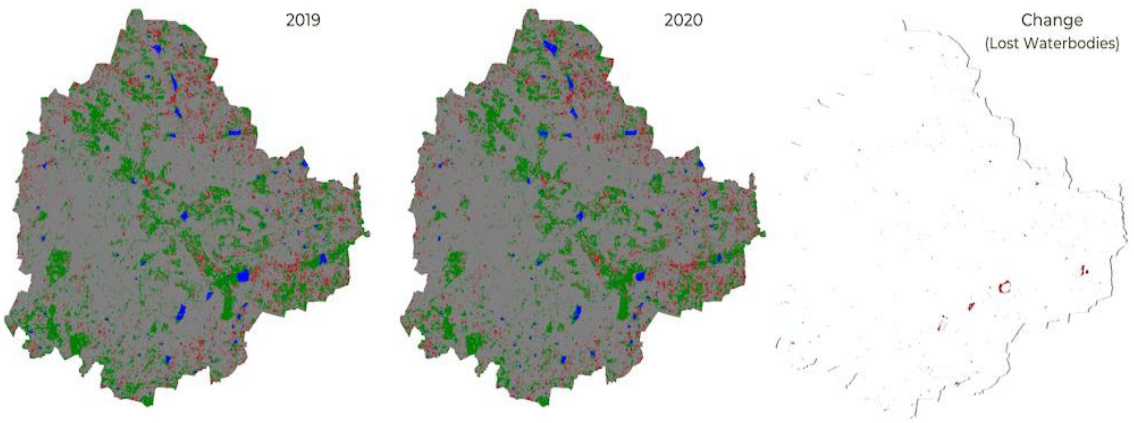
We dealing with multi-class images, a useful metric for change detection is to know how many pixels from class X changed to class Y. This can be accomplished using the ee.Reducer.frequencyHistogram() reducer as shown below.

```
// ========================================
// Landcover Change Detection (2019–2020)
// Random Forest + Transition Matrix
// Bangladesh Standard GEE Workflow
// ========================================

// BD Study Area (Bangladesh example - Dhaka region)
var geometry = ee.Geometry.Point([90.4125, 23.8103]);
Map.centerObject(geometry, 10);

// ========================================
// Sentinel-2 Collection
// ========================================
var s2 = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED');

// RGB Visualization
var rgbVis = {
```

```
  min: 0,
  max: 3000,
  bands: ['B4', 'B3', 'B2']
};

// ========================================
// Cloud-free Composites (2019 & 2020)
// ========================================
function getComposite(year) {
  return s2
    .filterBounds(geometry)
    .filterDate(year + '-01-01', year + '-02-01')
    .select('B.*')
    .median()
    .clip(geometry);
}

var before = getComposite(2019);
var after = getComposite(2020);

Map.addLayer(before, rgbVis, '2019 Image');
Map.addLayer(after, rgbVis, '2020 Image');

// ========================================
// Training Data (GCPs)
// ========================================
// Replace with Bangladesh training data if available
var urban = ee.FeatureCollection([]);
var bare = ee.FeatureCollection([]);
var water = ee.FeatureCollection([]);
var vegetation = ee.FeatureCollection([]);

var trainingPoints = urban
  .merge(bare)
  .merge(water)
  .merge(vegetation);

// ========================================
// Train Sample
// ========================================
var training = before.sampleRegions({
  collection: trainingPoints,
  properties: ['landcover'],
  scale: 10
});

// ========================================
// Random Forest Classifier
```

```
// ========================================
var classifier = ee.Classifier.smileRandomForest(50).train({
  features: training,
  classProperty: 'landcover',
  inputProperties: before.bandNames()
});

// ========================================
// Classification (Before & After)
// ========================================
var beforeClass = before.classify(classifier);
var afterClass = after.classify(classifier);

var palette = ['#cc6d8f', '#ffc107', '#1e88e5', '#004d40'];

Map.addLayer(beforeClass, {min: 0, max: 3, palette: palette}, '2019
Classified');
Map.addLayer(afterClass, {min: 0, max: 3, palette: palette}, '2020
Classified');

// ========================================
// Change Detection (Pixel Difference)
// ========================================
var beforeRe = beforeClass.remap([0,1,2,3],[1,2,3,4]);
var afterRe = afterClass.remap([0,1,2,3],[1,2,3,4]);

var changeMap = afterRe.subtract(beforeRe).neq(0);

Map.addLayer(
  changeMap,
  {min: 0, max: 1, palette: ['white', 'red']},
  'Change Map'
);

// ========================================
// Transition Matrix Creation
// ========================================
var transitions = beforeRe.multiply(100).add(afterRe)
  .rename('transition');

// Frequency table
var transitionHistogram = transitions.reduceRegion({
  reducer: ee.Reducer.frequencyHistogram(),
  geometry: geometry,
  scale: 10,
  maxPixels: 1e13
});
```

```
print('Transition Histogram:', transitionHistogram);

// ========================================
// Area per Transition Class (km²)
// ========================================
var areaImage = ee.Image.pixelArea()
  .divide(1e6)
  .addBands(transitions);

var areaStats = areaImage.reduceRegion({
  reducer: ee.Reducer.sum().group({
    groupField: 1,
    groupName: 'transition'
  }),
  geometry: geometry,
  scale: 100,
  maxPixels: 1e13
});

// ========================================
// Clean Output Formatting
// ========================================
var groups = ee.List(areaStats.get('groups'));

var formatted = groups.map(function(item) {
  item = ee.Dictionary(item);
  return ee.List([
    item.get('transition'),
    item.get('sum')
  ]);
});

print('Transition Area (sq km):', ee.Dictionary(formatted.flatten()));
```



Lost water pixels between 2019 and 2020

Understanding the Confusion Matrix in Machine Learning

Machine learning models are increasingly used in various applications to classify data into different categories. However, evaluating the performance of these models is crucial to ensure their accuracy and reliability. One essential tool in this evaluation process is the confusion matrix. In this article we will work on confusion matrix, its significance in machine learning and how it can be used to improve the performance of classification models.

Understanding Confusion Matrix

A **confusion matrix** is a simple table that shows how well a classification model is performing by comparing its predictions to the actual results. It breaks down the predictions into four categories: correct predictions for both classes (**true positives** and **true negatives**) and incorrect predictions (**false positives** and **false negatives**). This helps you understand where the model is making mistakes, so you can improve it.

The matrix displays the number of instances produced by the model on the test data.

- **True Positive (TP):** The model correctly predicted a positive outcome (the actual outcome was positive).
- **True Negative (TN):** The model correctly predicted a negative outcome (the actual outcome was negative).
- **False Positive (FP):** The model incorrectly predicted a positive outcome (the actual outcome was negative). Also known as a Type I error.
- **False Negative (FN):** The model incorrectly predicted a negative outcome (the actual outcome was positive). Also known as a Type II error.

A **confusion matrix** helps you see how well a model is working by showing correct and incorrect predictions. It also helps calculate key measures like **accuracy**, **precision**, and **recall**, which give a better idea of performance, especially when the data is imbalanced.

Metrics based on Confusion Matrix Data

1. Accuracy

Accuracy measures how often the model's predictions are correct overall. It gives a general idea of how well the model is performing. However, accuracy can be misleading, especially with imbalanced datasets where one class dominates. For example, a model that predicts the majority class correctly most of the time might have high accuracy but still fail to capture important details about other classes.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2. Precision

Precision focuses on the quality of the model's positive predictions. It tells us how many of the instances predicted as positive are actually positive. Precision is important in situations where false positives need to be minimized, such as detecting spam emails or fraud.

$$Precision = \frac{TP}{TP + FP}$$

3. Recall

Recall measures how well the model identifies all actual positive cases. It shows the proportion of true positives detected out of all the actual positive instances. High recall is essential when missing positive cases has significant consequences, such as in medical diagnoses.

$$Recall = \frac{TP}{TP + FN}$$

4. F1-Score

F1-score combines precision and recall into a single metric to balance their trade-off. It provides a better sense of a model's overall performance, particularly for imbalanced datasets. The F1 score is helpful when both false positives and false negatives are important, though it assumes precision and recall are equally significant, which might not always align with the use case.

$$F1 - Score = \frac{2 \times Precision \times Recall}{Precision \ + \ Recall}$$

5. Specificity

Specificity is another important metric in the evaluation of classification models, particularly in binary classification. It measures the ability of a model to correctly identify negative instances. Specificity is also known as the True Negative Rate. Formula is given by:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

6. Type 1 and Type 2 error

- **Type 1 error**
- A **Type 1 Error** occurs when the model incorrectly predicts a positive instance, but the actual instance is negative. This is also known as a **false positive**. Type 1 Errors affect the **precision** of a model, which measures the accuracy of positive predictions.

$$Type\ 1\ Error = \frac{FP}{FP + TN}$$

- **Type 2 error**

  A **Type 2 Error** occurs when the model fails to predict a positive instance, even though it is actually positive. This is also known as a **false negative**. Type 2 Errors impact the **recall** of a model, which measures how well the model identifies all actual positive cases.

$$Type\ 2\ Error = \frac{FN}{TP + FN}$$

Example: A diagnostic test is used to detect a particular disease in patients.

- **Type 1 Error (False Positive):** This occurs when the test predicts a patient has the disease (positive result), but the patient is actually healthy (negative case).
- **Type 2 Error (False Negative):** This occurs when the test predicts the patient is healthy (negative result), but the patient actually has the disease (positive case).

7. Cohen's Kappa (Kappa Coefficient)

7. Cohen's Kappa (κ)

Cohen's Kappa measures the agreement between predicted and actual classifications while correcting for chance agreement.

Formula:

$\kappa = (P_o - P_e) / (1 - P_e)$

Where:

P_o (Observed Agreement):
It represents the actual proportion of correctly classified instances.

$P_o = (TP + TN) / (TP + TN + FP + FN)$

P_e (Expected Agreement by Chance):
It represents the probability that agreement occurs randomly based on class distribution.

$P_e = [(TP + FP)(TP + FN) + (FN + TN)(FP + TN)] / (TP + TN + FP + FN)^2$

Interpretation:

- $\kappa < 0$ → Poor agreement
- 0.0 – 0.20 → Slight agreement
- 0.21 – 0.40 → Fair agreement
- 0.41 – 0.60 → Moderate agreement
- 0.61 – 0.80 → Substantial agreement
- 0.81 – 1.00 → Almost perfect agreement

Confusion Matrix For binary classification

A 2X2 Confusion matrix is shown below for the image recognition having a Dog image or Not Dog image:

	Predicted	**Predicted**
Actual	True Positive (TP)	False Negative (FN)
Actual	False Positive (FP)	True Negative (TN)

- **True Positive (TP):** It is the total counts having both predicted and actual values are Dog.
- **True Negative (TN):** It is the total counts having both predicted and actual values are Not Dog.
- **False Positive (FP):** It is the total counts having prediction is Dog while actually Not Dog.
- **False Negative (FN):** It is the total counts having prediction is Not Dog while actually, it is Dog.

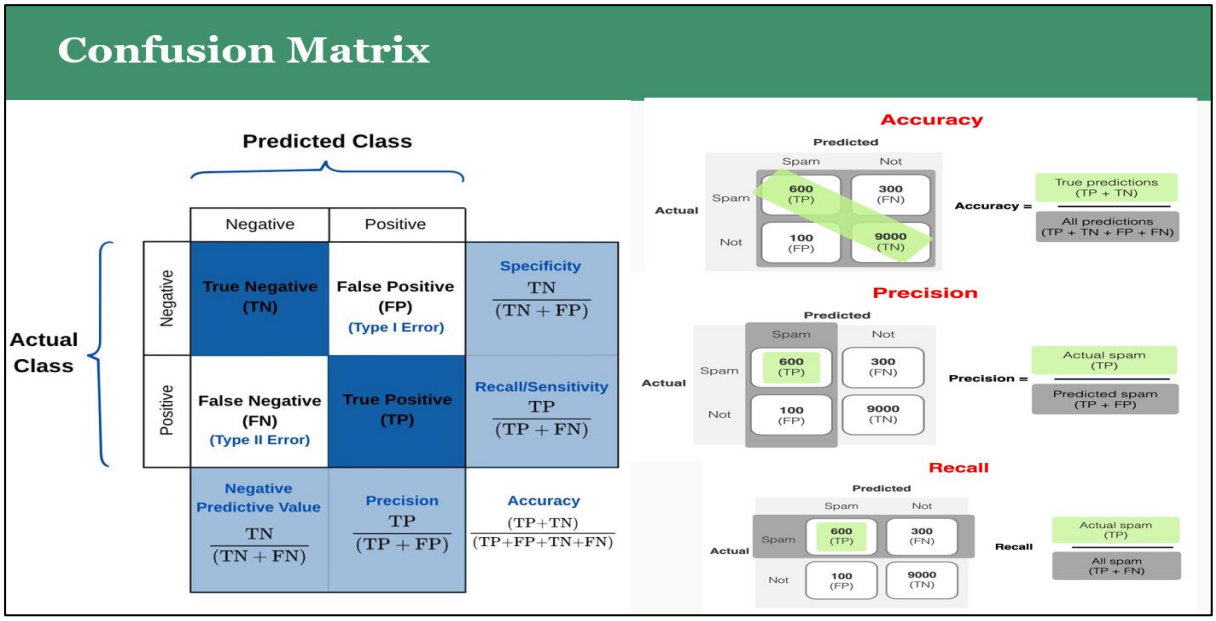
Example: Confusion Matrix for Image Classification (Cat, Dog, Horse)

	Predicted Cat	**Predicted Dog**	**Predicted Horse**
Actual Cat	True Positive (TP)	False Negative (FN)	False Negative (FN)
Actual Dog	False Negative (FN)	True Positive (TP)	False Negative (FN)
Actual Horse	False Negative (FN)	False Negative (FN)	True Positive (TP)

- The definitions of all the terms (TP, TN, FP and FN) are the same as described in the previous example.

Example with Numbers:
Let's consider the scenario where the model processed 30 images:

	Predicted Cat	**Predicted Dog**	**Predicted Horse**
Actual Cat	8	1	1
Actual Dog	2	10	0
Actual Horse	0	2	8



Spatiotemporal Drought Assessment with GEE

Mr. Md. Zahid Hasan Siddiquee
Associate GIS & Remote Sensing Specialist, IWM

Module 1: Introduction to Google Earth Engine (GEE)

Core Concepts

- **What is GEE?** A cloud-based platform for planetary-scale geospatial analysis. It is distinct from Google Earth (3D map) as it allows for the processing of petabytes of satellite imagery without needing high-end local hardware.
- **Key Advantages:**
    - No local storage required for massive datasets.
    - Fast processing using Google's cloud infrastructure.
    - Access to a vast library of satellite data (Landsat, Sentinel, MODIS).

The Code Editor & JavaScript Basics

The Code Editor uses JavaScript. Key syntax includes:

- **Variables:** Defined using var (e.g., var text = 'Hello';).
- **Lists:** Items stored in square brackets (e.g., ['item1', 'item2']).
- **Functions:** Reusable blocks of code (e.g., function calculate(a, b) { ... }).
- **Comments:** Use // for single lines and /* ... */ for multiple lines.

Module 2: Handling Optical Satellite Imagery

Working with Image Collections

Analysis begins by importing and filtering datasets (commonly MODIS for drought).

- **Filtering:** Narrow down data by Date (.filterDate), Bounds/Region (.filterBounds), or Metadata (.filterMetadata).
- **Scaling Factors:** Many satellite products store data as integers to save space. You must apply a scaling factor to get physical values:
    - **NDVI:** Multiply by 0.0001.
    - **LST (Land Surface Temp):** Multiply by 0.02 and subtract 273.15 to convert Kelvin to Celsius.

Reducers

Reducers allow you to aggregate data over time or space, such as calculating the median(), mean(), or max() value of an image collection.

Module 3: Calculating Simple Drought Indices
Vegetation Condition Index (VCI)
VCI compares current NDVI to the historical range (Min and Max) to determine how "green" a region is relative to its potential.

- **Formula:** VCI = 100 * (NDVI - NDVI_min) / (NDVI_max - NDVI_min)

Temperature Condition Index (TCI)
TCI assesses thermal stress by looking at LST relative to historical extremes.

- **Formula:** TCI = 100 * (LST_max - LST) / (LST_max - LST_min)

Working with Shapefiles
To analyze a specific area, you can:

1. Use the GEE Catalog (e.g., USDOS/LSIB_SIMPLE/2017).
2. Upload a local .shp file as a **GEE Asset** to use as a FeatureCollection.

Module 4: Integrated & Advanced Drought Indices
Vegetation Health Index (VHI)
VHI combines both vegetation (VCI) and temperature (TCI) data for a more robust assessment.

- **Formula:** VHI = 0.5 * VCI + 0.5 * TCI

Pre-calculated Indices in GEE
GEE provides access to established datasets like **GRIDMET/DROUGHT**, which includes:

- **PDSI:** Palmer Drought Severity Index.
- **SPI:** Standardized Precipitation Index.
- **EDDI:** Evaporative Demand Drought Index.

Drought Severity Index (DSI)
DSI integrates Evapotranspiration (ET) and Potential Evapotranspiration (PET) ratios with NDVI to monitor drought through water-energy balance.

Module 5: Time Series & Hotspot Analysis
Temporal Data Visualization
To understand drought trends, we use the ui.Chart functions:

- **Series Charts:** Compare a single variable over time across different regions.
- **Day of Year (DOY) Charts:** Compare current year performance against historical seasonal norms.

Zonal Statistics
This process involves calculating the average drought index value within the boundaries of specific administrative units (e.g., districts or municipalities) using .reduceRegions.

List of Abbreviations

- **GEE (Google Earth Engine):** A cloud-based platform for planetary-scale geospatial analysis and processing.
- **NDVI (Normalized Difference Vegetation Index):** A metric used to assess the health and "greenness" of vegetation based on satellite imagery.
- **LST (Land Surface Temperature):** The skin temperature of the Earth's surface, used to identify thermal stress.
- **VCI (Vegetation Condition Index):** A drought index that compares current NDVI to historical maximum and minimum values for a specific period.
- **TCI (Temperature Condition Index):** A drought index used to determine temperature-related stress by comparing current LST to historical extremes.
- **VHI (Vegetation Health Index):** An integrated index (typically 50% VCI and 50% TCI) used to assess overall vegetation health.
- **PCI (Precipitation Condition Index):** An index that evaluates drought based on precipitation data relative to historical norms.
- **SMCI (Soil Moisture Condition Index):** An index used to monitor drought by analyzing soil moisture levels.
- **DSI (Drought Severity Index):** A composite index integrating Evapotranspiration (ET), Potential Evapotranspiration (PET), and NDVI.
- **NDDI (Normalized Difference Drought Index):** A hybrid index combining NDVI and NDWI to identify drought conditions.
- **PDSI (Palmer Drought Severity Index):** A standard index that uses temperature and precipitation data to track long-term cumulative moisture supply and demand.
- **SPI (Standardized Precipitation Index):** An index based solely on precipitation, used to characterize meteorological drought at various timescales.
- **EDDI (Evaporative Demand Drought Index):** An index that examines how atmospheric "thirst" (evaporative demand) relates to drought.
- **ET / PET:** Evapotranspiration and Potential Evapotranspiration; measures of water transfer from the land to the atmosphere.
- **DOY (Day of Year):** A chronological day number (1-365) used for seasonal comparisons in time series analysis.

Important GEE Functions

Data Selection & Filtering

- **ee.ImageCollection()**: Accesses a stack of satellite images from the GEE data catalog (e.g., MODIS or Landsat).

- **.filterDate(start, end)**: Narrow down an image collection to a specific time window.
- **.filterBounds(geometry)**: Restricts data processing to a specific geographic area or point.
- **.select('band_name')**: Isolates specific data layers (e.g., just the 'NDVI' band) from a multi-band image.

Mathematical Operations

- **.multiply(value) / .subtract(value)**: Used to apply **Scaling Factors** to raw satellite data (e.g., converting Kelvin to Celsius).
- **.map(function)**: Applies a specific set of instructions (like a drought formula) to every image within a collection.
- **.reduce(ee.Reducer)**: Aggregates data over time or space (e.g., calculating the min, max, or mean of a 20-year dataset).
- **.where(condition, value)**: A conditional function used to classify continuous index values into discrete drought categories (e.g., 0-10 = Extreme Dry).

Visualization & Mapping

- **Map.addLayer(object, visParams, name)**: Displays a processed image or shapefile on the interactive map.
- **Map.centerObject(object, zoom)**: Automatically centers the map view on your study area.
- **ui.Thumbnail()**: Generates a small preview image, often used for creating color bars in map legends.
- **ui.Label() / ui.Panel()**: Interface tools used to build and organize custom legends and titles on the map.

Analysis & Charting

- **ui.Chart.image.seriesByRegion()**: Generates a line graph showing how an index (like DSI) changes over time for a specific region.
- **ui.Chart.image.doySeriesByRegion()**: Creates a graph comparing current year data against historical patterns for the same calendar days.
- **.clip(geometry)**: Masks or "cuts" an image so that it only shows data within the boundaries of a specific shapefile or asset.
- **.copyProperties(source, properties)**: Ensures that metadata, such as the acquisition date (system:time_start), is preserved after mathematical processing.

Land Use/Land Cover (LULC) Classification Using Google Earth Engine (GEE)

Mr. Mahmudul Hasan
GIS & Remote Sensing Specialist
ESRI, Bangladesh

---

1. Introduction

This manual explains how to perform supervised Land Use/Land Cover (LULC) classification using Google Earth Engine (GEE) with Landsat 8 satellite imagery and the Random Forest classification algorithm.

The workflow includes:

- Importing Landsat imagery
- Creating mosaicked imagery
- Clipping imagery to the study area
- Preparing training samples
- Training a Random Forest classifier
- Producing classified land cover maps
- Performing accuracy assessment
- Exporting classified outputs and confusion matrix

The script is designed for use in the Google Earth Engine JavaScript Code Editor.

---

2. Software and Platform Requirements

Requirement	Version/Platform
Google Earth Engine Account	Required
Web Browser	Google Chrome / Firefox
Internet Connection	Required
Google Drive	Required for export

---

3. Required Input Data

The following datasets and feature collections are required before running the script.

3.1 Study Area Boundary

Variable Name	Description
CCC	Main study area polygon
roi2	Secondary region for mosaic generation

These should be imported as FeatureCollections or Geometry assets.

Example:

```
var CCC = ee.FeatureCollection('users/your_username/CCC');
```

3.2 Training Sample Data

Training samples must be prepared for each land cover class.

Variable Name	Land Cover Class
Vegetation	Vegetation samples
Waterbody	Waterbody samples
Urban	Urban samples
Barren	Barren land samples

Each training sample must contain a field named:

`landcover`

Example class coding:

Class	Code
Vegetation	0
Waterbody	1
Urban	2
Barren	3

3.3 Validation Sample Data

Separate validation datasets are required for accuracy assessment.

Variable Name	Description
valVegetation	Validation samples for vegetation
valWaterbody	Validation samples for waterbody
valUrban	Validation samples for urban
valBarren	Validation samples for barren land

Validation points must also contain the field:

`landcover`

---

4. Workflow Description

4.1 Import Landsat Imagery

The script imports Landsat 8 Top of Atmosphere (TOA) imagery from Google Earth Engine.

```
var landsat_1 = ee.ImageCollection("LANDSAT/LC08/C02/T1_RT_TOA")
```

Filtering operations include:

- Study area filtering
- Date filtering

- Cloud cover sorting

The image with the least cloud cover is selected using:

```
.first();
```

4.2 Create Mosaic Image

Two Landsat images are merged together into a single mosaic image.

```
var mosaic_1 = ee.ImageCollection.fromImages([landsat_2, landsat_1])
.mosaic();
```

Purpose:

- Reduce cloud contamination
- Improve spatial coverage
- Fill missing areas

4.3 Clip Image to Study Area

The mosaicked image is clipped using the study area boundary.

```
var landsat_2000 = mosaic_1.clip(CCC);
```

Purpose:

- Reduce processing time
- Focus analysis only within the study area

4.4 Display Satellite Imagery

The clipped image is visualized using RGB bands.

```
Map.addLayer(landsat_2000, {
  bands: ['B3','B2','B1'],
  min: 0,
  max: 0.3,
  gamma: 1.4
}, 'CCC-2000')
```

Band combination:

Band	Description
B3	Red
B2	Green
B1	Blue

4.5 Merge Training Samples

All training classes are merged into one FeatureCollection.

```
var landcover = Vegetation.merge(Waterbody)
                          .merge(Urban)
                          .merge(Barren);
```

Purpose:

- Create a unified training dataset
- Simplify classifier training

4.6 Select Spectral Bands

The following Landsat bands are selected for classification.

```
var bands = ['B1', 'B2', 'B3', 'B4', 'B5', 'B7'];
```

Band	Description
B1	Coastal/Aerosol
B2	Blue
B3	Green
B4	Red
B5	Near Infrared (NIR)
B7	Shortwave Infrared (SWIR2)

4.7 Generate Training Data

Training data are extracted from the selected bands using the training samples.

```
var training = landsat_2000.select(bands).sampleRegions({
  collection: landcover,
  properties: ['landcover'],
  scale: 30
});
```

Parameters:

Parameter	Description
collection	Training samples
properties	Target classification field
scale	Spatial resolution (30m)

4.8 Train Random Forest Classifier

A Random Forest classifier is created and trained.

```
var classifier = ee.Classifier.smileRandomForest({
  numberOfTrees: 10
});
```

Training process:

```
var trainedClassifier = classifier.train({
  features: training,
  classProperty: 'landcover',
  inputProperties: bands
});
```

Parameters:

Parameter	Description
features	Training dataset
classProperty	Class label field
inputProperties	Spectral bands

4.9 Perform Image Classification

The trained classifier is applied to the Landsat image.

```
var classified = landsat_2000.select(bands)
                              .classify(trainedClassifier);
```

Output:

- Classified raster image
- Pixel-wise land cover categories

4.10 Define Classification Color Palette

A visualization palette is created.

```
var palette = [
  '6db700',
  '23dae8',
  'e60e0e',
  'e4ae26'
];
```

Color	Class
Green	Vegetation
Sky Blue	Waterbody
Red	Urban
Yellow	Barren

4.11 Display Classification Result

```
Map.addLayer(classified, {
  min:0,
  max:2,
  palette: palette
}, 'LULC_2000');
```

This displays the final classified map in the GEE map viewer.

4.12 Validation and Accuracy Assessment

Validation datasets are merged.

```
var valNames = valVegetation.merge(valWaterbody)
                            .merge(valUrban)
                            .merge(valBarren);
```

Validation samples are extracted:

```
var validation = classified.sampleRegions({
  collection: valNames,
  properties: ['landcover'],
  scale: 30
});
```

Error matrix generation:

```
var testAccuracy = validation.errorMatrix('landcover', 'classificati
on');
```

Overall accuracy:

```
testAccuracy.accuracy();
```

---

5. Exporting Results

5.1 Export Classified Image

The classified raster is exported to Google Drive.

```
Export.image.toDrive({
  image: classified,
  description: 'classified_image',
  folder: 'Thesis',
  fileNamePrefix: 'classified_image',
  scale: 30,
  region: CCC,
  fileFormat: 'GeoTIFF'
});
```

Output format:

- GeoTIFF

Export location:

- Google Drive → Thesis folder

5.2 Export Confusion Matrix

The confusion matrix is converted into a FeatureCollection.

```
var errorMatrixFC = ee.FeatureCollection([
  ee.Feature(null, testAccuracy.array())
]);
```

Export operation:

```
Export.table.toDrive({
  collection: errorMatrixFC,
  description: 'Confusion_Matrix',
  folder: 'Thesis',
  fileFormat: 'CSV'
});
```

Output format:

- CSV file

---

6. Running the Script

Step 1

Open Google Earth Engine Code Editor.

Step 2

Import all required assets:

- Study area
- Training samples
- Validation samples

Step 3

Paste the script into the editor.

Step 4

Modify date range and asset names if necessary.

Step 5

Click Run.

Step 6

Check outputs in:

- Console
- Map Viewer
- Tasks tab

Step 7

Click Run in the Tasks panel to start export.

---

7. Important Notes

7.1 Landsat Collection

The script uses:

`LANDSAT/LC08/C02/T1_RT_TOA`

which corresponds to:

- Landsat 8
- Collection 2
- Tier 1
- Real-Time TOA Reflectance

7.2 Date Range

Current script:

```
.filterDate('2025-01-01', '2025-12-31')
```

Modify according to the required study year.

7.3 Cloud Cover

Images are sorted using:

```
.sort('CLOUD_COVER')
```

The least cloudy image is selected automatically.

7.4 Classification Accuracy

Higher accuracy depends on:

- Good training samples
- Balanced sample distribution
- Proper class separation
- Sufficient validation points

---

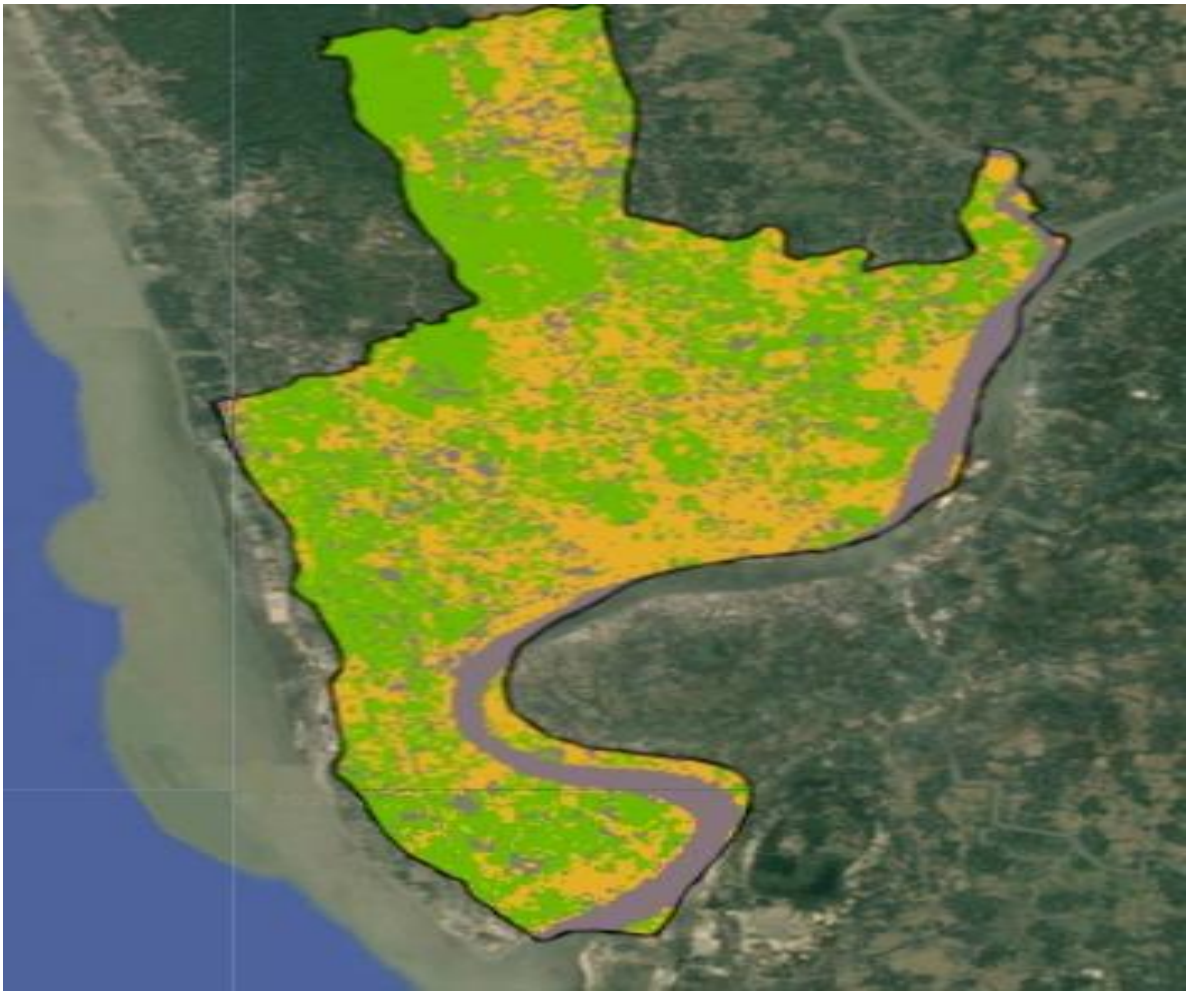
8. Conclusion

This workflow provides a complete supervised land cover classification framework using Google Earth Engine and Landsat imagery. The methodology supports:

- Satellite image preprocessing
- Supervised classification
- Accuracy assessment
- Exporting GIS-ready outputs

The generated land cover maps can support:

- Environmental monitoring
- Urban expansion analysis
- Water resource studies
- Agricultural analysis
- Change detection studies

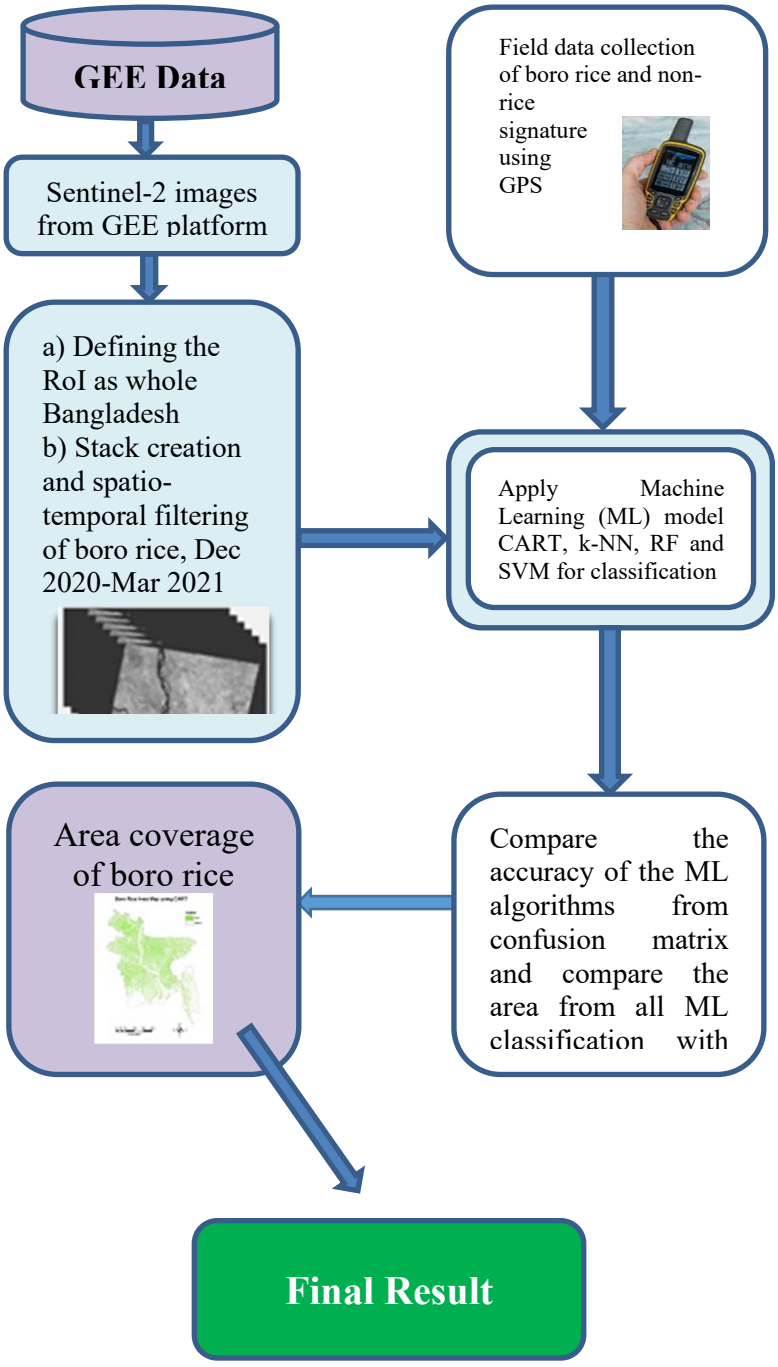


Estimation of Boro Rice Cultivated Area using Google Earth Engine (GEE)

Hasan Md. Hamidur Rahman
Director (Computer & GIS Unit)
Bangladesh Agricultural Research Council
Email: h.rahman@barc.gov.bd

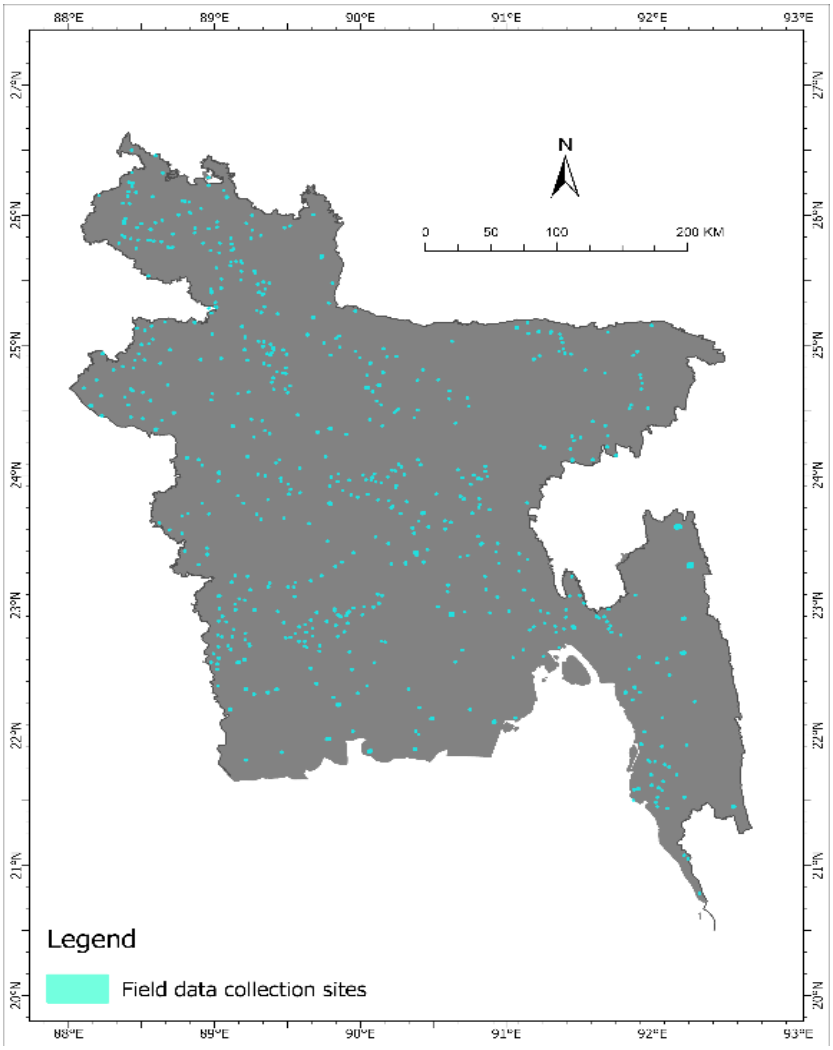
Steps for Boro rice area estimation

Flowchart of Boro rice area delineation:

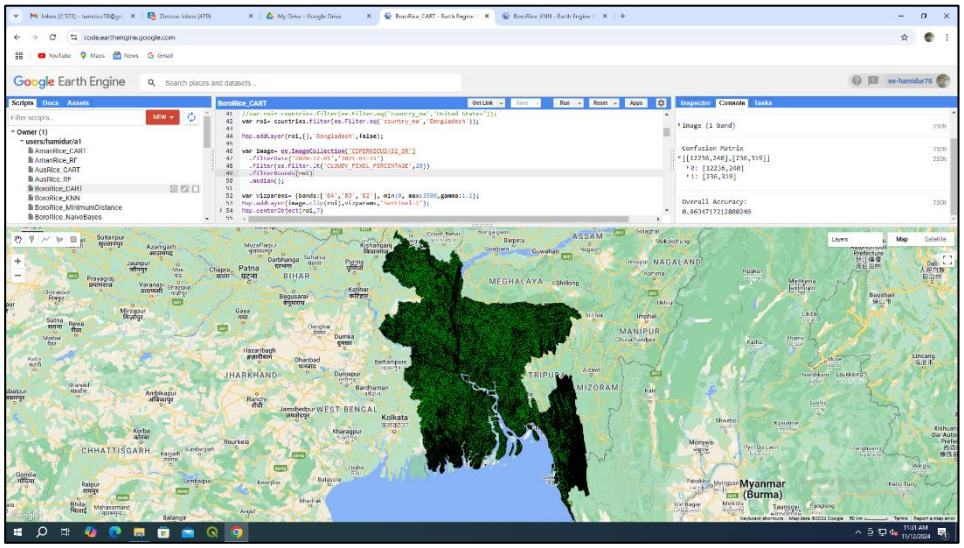


Field/ Signature data collection and processing

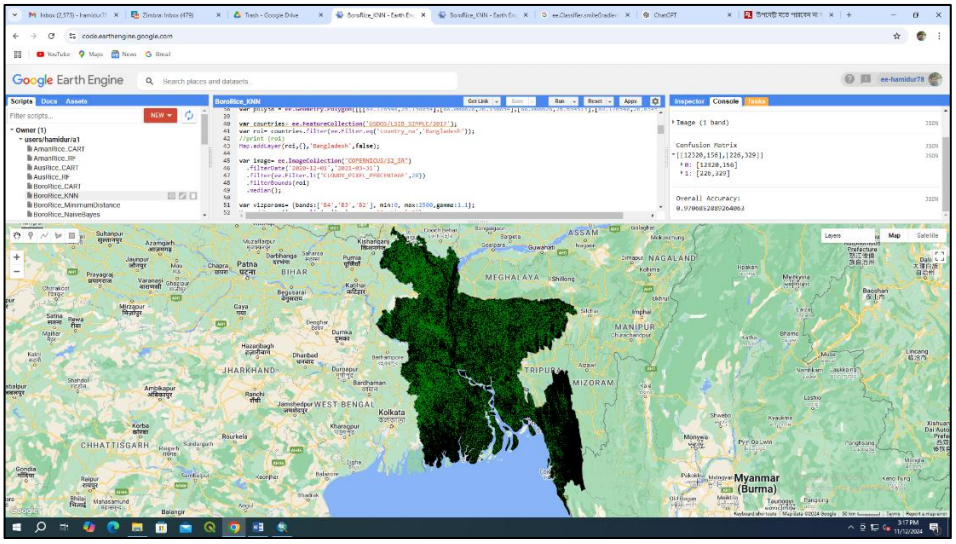
1. Field Data Collection through GPS device
2. The field data collected during boro seasons are compiled into an Excel sheet. The validated data is then saved as a CSV file, which serves as the attribute file.
3. Simultaneously, the GPX file from the handheld GPS devices is downloaded and imported into Google Earth Pro to verify the survey locations.
4. Polygons are drawn using the GPS points and digital photos for most of the fields.
5. The file is exported in KML format and opened in QGIS, where it is converted into a shapefile.
6. The shapefile is then joined with the attribute file (CSV) to merge the land types and other parameters.
7. Non-rice land types are generalized into a single class called 'non-rice'. In the shapefile, there are two columns named 'id' and 'feature_name', where the value '0' represents 'non-rice' and '1' represents ‘rice’ features.
8. The generalized shapefile is imported into the Google Earth Engine (GEE) asset, where it is used as the signature data in the machine learning (ML) model.



Screenshot of boro rice classified GEE raster data (CART)



Screenshot of boro rice classified GEE raster data (k-NN)



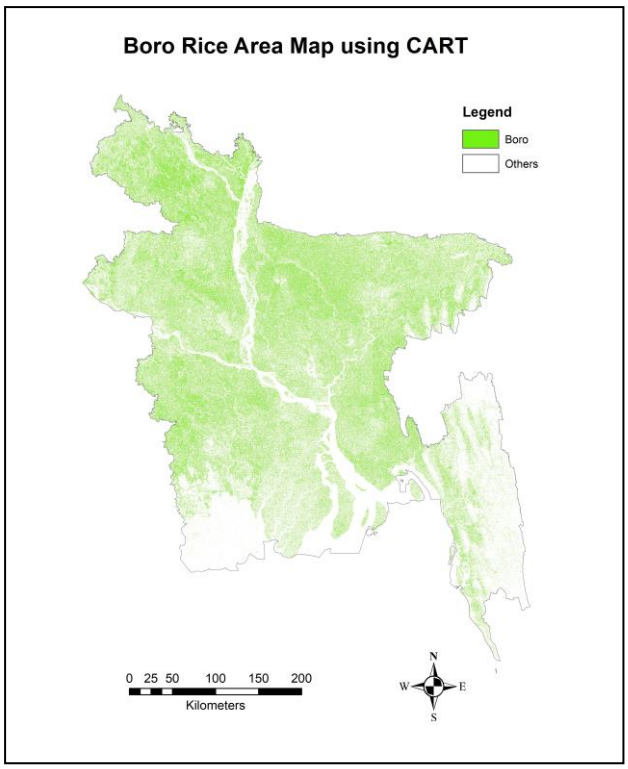
Steps to process data in ArcGIS

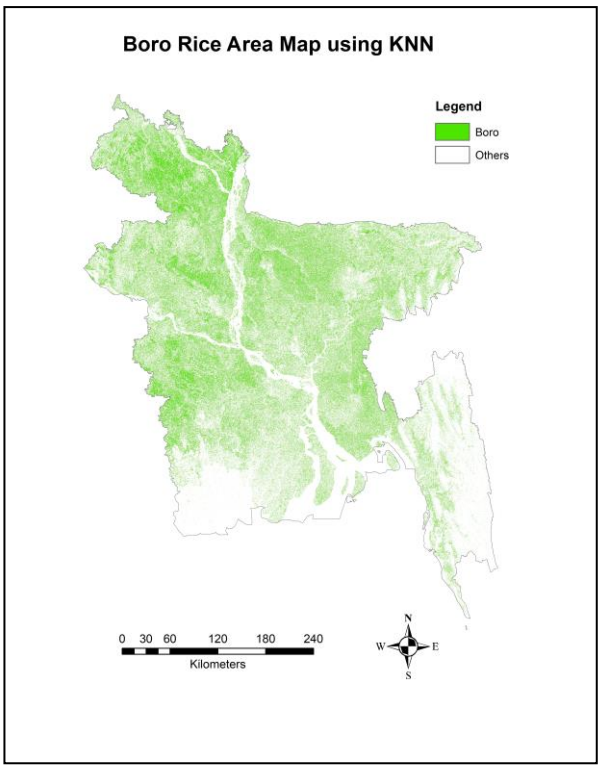
1. Boro Classified GEE Raster Data
2. Boro Reclassified Mosaiced Raster Data
3. Resampled Raster Data
4. Boro Final Area Result.

ArcGIS Tool for Raster data processing:

- Arc Tool Box-> Spatial Analysis Tools->Reclass->Reclassify
- Arc Tool Box-> Data Management Tools->Raster->Raster Dataset->Mosaic to New Raster
- Arc Tool Box-> Data Management Tools->Raster->Raster Processing->Resample

Boro rice area map





Boro Rice Area Comparison

Sl.	Classification Algorithm	Pixel count (10m x 10m)	Area in Sq. Meter	Area in Sq. km	Area in Ha	Classification Accuracy of Algorithm (%)	Area Accuracy with respect to BBS (%)
1	**CART**	419957945	41995794500	41996	**4,199,579**	**96**	**88**
2	**k-NN**	384058603	38405860300	38406	**3,840,586**	97	**80**
BBS Statistics 2020-21:					**4,786,621**	-	-

Conclusion

The training program on “Satellite-Based Agricultural Analysis Using Google Earth Engine (GEE)”, organized by the Bangladesh Agricultural Research Council through its Computer & GIS Unit, successfully introduced participants to the practical applications of remote sensing, satellite data analysis, and cloud-based geospatial technologies for agricultural monitoring and decision-making. Through theoretical sessions and hands-on exercises, participants developed skills in Google Earth Engine, vegetation index analysis, land use and land cover mapping, drought assessment, crop monitoring, and machine learning applications in agriculture. This manual serves as a valuable reference for applying these techniques in research, precision agriculture, and sustainable resource management in Bangladesh. The Council expresses its sincere appreciation to all resource persons, contributors, and participants for their active support and hopes that the knowledge gained from this training will contribute to advancing modern, data-driven agricultural development in Bangladesh.