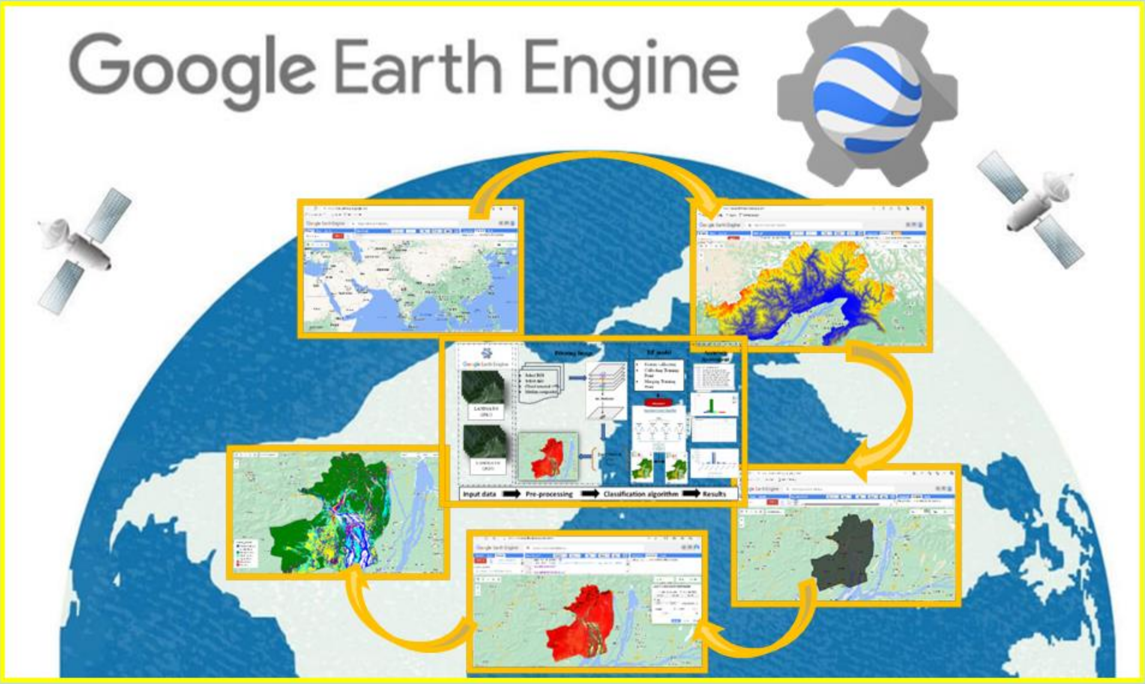
ICARNEH/PME-TU/Pub/2023/406

Google Earth Engine: Application in land use classification

(E-Learning lesson for beginners in Cloud-Based Remote Sensing)



Prepared by:

Ashwini Suryawanshi
Burhan U. Choudhury
Nishtha Sharnagat
Snehil Dubey

ICAR Research Complex for NEH Region
Umiam, Meghalaya-793103

ICARNEH/PME-TU/Pub/2023/406

Google Earth Engine: Application in land use classification

E-Learning lesson for beginners in Cloud-Based Remote Sensing

Prepared by:

Ashwini Suryawanshi
Burhan U. Choudhury
Nishtha Sharnagat
Snehil Dubey

DIVISION OF NRM
ICAR Research Complex for NEH Region
Umiam- 793103, Meghalaya (India)

Citation: Suryawanshi, A., Choudhury, B.U., Sharnagat, N., and Dubey, S. 2024. Google Earth Engine: *E-Learning lesson for beginners in Cloud-Based Remote Sensing*. ICAR Research Complex for NEH Region. Pages: 1-38.
Publication No: ICARNEH/PME-TU/Pub/2023/406

ICARNEH/PME-TU/Pub/2023/406

Corresponding author: burhan.icar@gmail.com (B. U. Choudhury, Principal Scientist)

Citation: Suryawanshi, A., Choudhury, B.U., Sharnagat, N., and Dubey, S. 2024. Google Earth Engine: *E-Learning lesson for beginners in Cloud-Based Remote Sensing*. ICAR Research Complex for NEH Region. Pages: 1-38.
Publication No: ICARNEH/PME-TU/Pub/2023/406

ICARNEH/PME-TU/Pub/2023/406

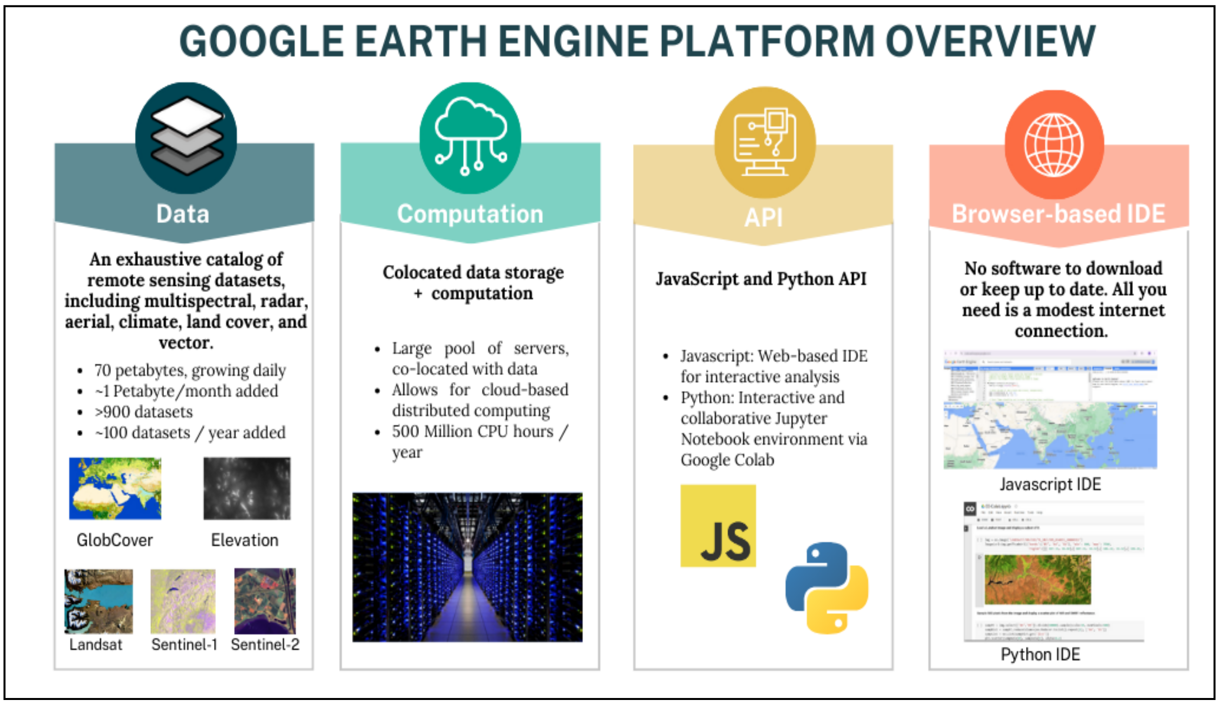
TABLE OF CONTENTS

Chapter 1.0: Google Earth Engine (GEE)

1.1 Platform Overview

Google Earth Engine (GEE) is a cloud-based platform for processing geospatial data, that enables environmental monitoring and analysis on a large scale (Gorelick et al., 2017). Since its release in 2010, GEE has been freely accessible for academic and research purposes, offering allocated quotas for commercial applications (Waleed & Sajjad, 2023). The GEE platform offers extensive capabilities, including (Figure 1):

- Accessing large amounts of remote sensing data that is publicly available that reaches to petabyte scale and pre-processed data products providing access for over 40 years of satellite datasets via an interactive explorer web application.
- High-performance machine learning (ML) algorithms and parallel computing leveraging Google's powerful computational network.
- A comprehensive Application Programming Interfaces (APIs) library that is compatible with widely used programming languages like JavaScript and Python.
- The browser-based Integrated Development Environment (IDE) in Google Earth Engine (GEE) eliminates the need for software installation or maintenance, providing a fully online platform for geospatial analysis.


GOOGLE EARTH ENGINE PLATFORM OVERVIEW

Data

An exhaustive catalog of remote sensing datasets, including multispectral, radar, aerial, climate, land cover, and vector.

- 70 petabytes, growing daily
- ~1 Petabyte/month added
- >900 datasets
- ~100 datasets / year added

Computation

Colocated data storage + computation

- Large pool of servers, co-located with data
- Allows for cloud-based distributed computing
- 500 Million CPU hours / year

API

JavaScript and Python API

- Javascript: Web-based IDE for interactive analysis
- Python: Interactive and collaborative Jupyter Notebook environment via Google Colab

Browser-based IDE

No software to download or keep up to date. All you need is a modest internet connection.

Fig. 1 Overview of Google Earth Engine Platform

The catalog largely consists of remote sensing imagery from Earth observation missions, notably the full Landsat archive and the complete datasets of Sentinel-1 and 2. Additionally, it encompasses datasets of climate prediction and forecast, land use and cover data, and various geophysical, atmospheric, and socio-economic (Tamiminia et al., 2020). These datasets get updated continuously with nearly 6,000 new scenes that get added daily from ongoing and active missions, with an average delay of 24 hours post-acquisition (Gorelick et al., 2017). Users based on their requirements may also request for the extension of the latest datasets to the public domain or can also upload their data, accessible through command-line tools or browser-based, with options that can be shared with other groups or different users.

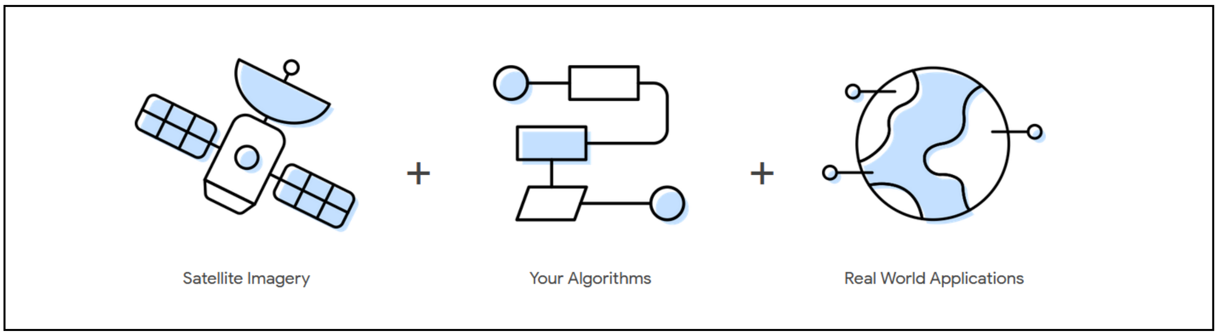


Fig. 2 Core components of Google Earth Engine Platform (Source: https://developers.google.com/earth-engine)

Apart from its extensive data catalog, a notable advantage of Google Earth Engine (GEE) is its robust system architecture. Google Earth Engine runs on powerful technology from Google's data systems. It uses tools to manage lots of computers working together, store data across different places, and process multiple tasks at the same time to get things done faster (Gorelick et al., 2017). This design allows users to leverage GEE's extensive library of nearly a thousand functions, spanning from simple algebraic operations to complex geostatistical, image processing, and machine learning algorithms. The Figure 2 illustrates the core components of Google Earth Engine (GEE), highlighting its integration of satellite imagery, user-defined algorithms, and real-world applications.

1.2 Applications of GEE

Google Earth Engine (GEE) is invaluable for its ability to significantly reduce analysis time by utilizing Google's powerful computing infrastructure, enabling researchers to perform tasks that were previously unfeasible by running computations across thousands of machines. With access to petabytes of data, GEE facilitates efficient large-scale analyses. Its advanced features, such as tools for cloud and haze removal, streamline image pre-processing, enhancing both usability and precision. Moreover, GEE fosters collaboration and standardization by offering a unified platform for global data analysis, enabling shared research efforts and consistent methodologies across diverse scientific disciplines.

It provides a robust platform for advanced spatial analysis, empowering researchers to address diverse challenges across multiple scientific disciplines (Figure 3). Its extensive analytical capabilities enable the integration and exploration of a wide array of geospatial datasets, supporting the development of innovative solutions to pressing global issues (Velastegui-Montoya et al., 2023). GEE is pivotal in facilitating real-time monitoring of deforestation, forecasting droughts, assessing climate change impacts, and responding rapidly to natural disasters (Amani et al., 2020). Furthermore, it contributes to disease control through spatial analytics, enhances food security by tracking crop health, and advances sustainability through urban planning and environmental assessments.

In addition, GEE plays a crucial role in addressing land use changes and deforestation, both of which place significant stress on forest ecosystems, essential for water regulation and soil stabilization (Suryawanshi et al., 2023; Singh et al., 2024). The advent and availability of advanced sensors and satellite products have further escalated the complexity of processing and interpreting vast spatial datasets, presenting new challenges and opportunities for researchers (Suryawanshi et al., 2021, Wangchu et al., 2024).

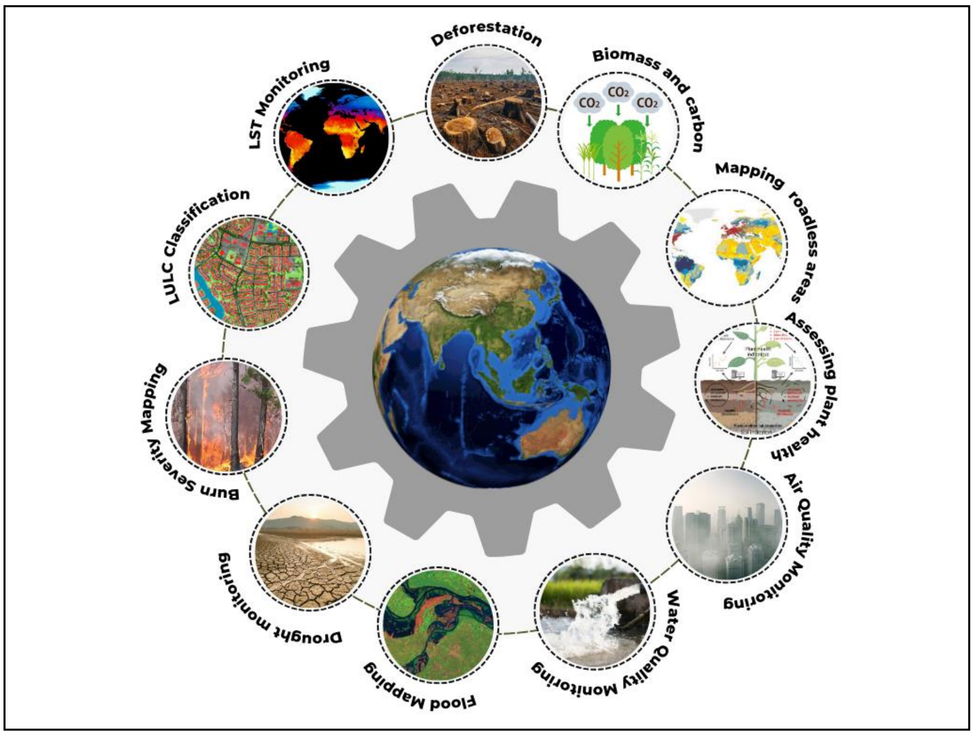
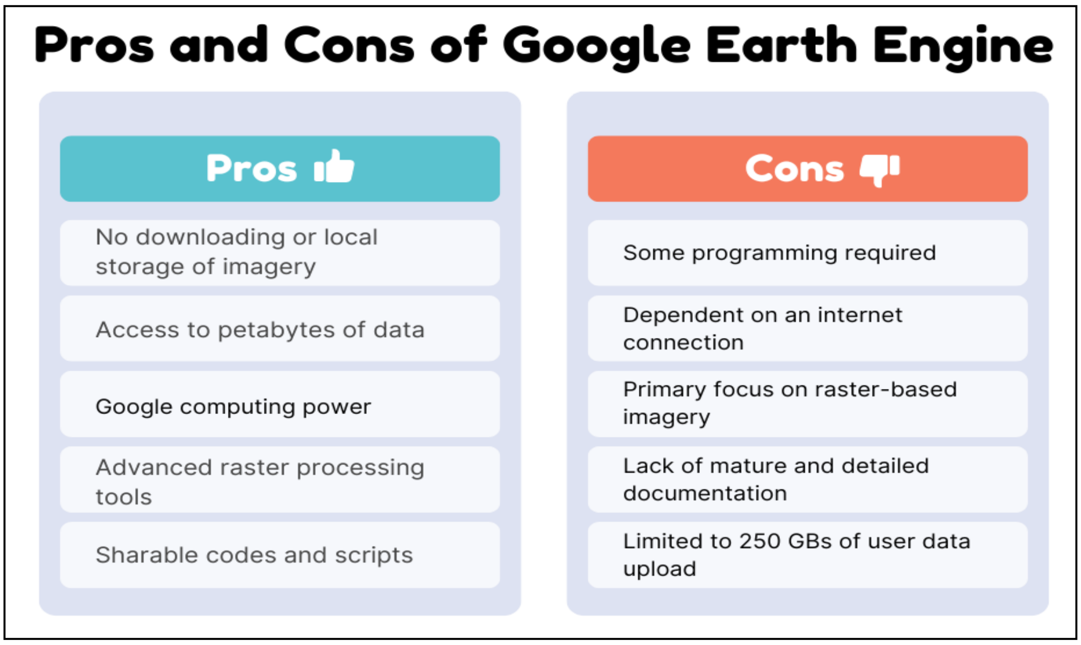


Fig. 3 Applications and utilities of Google Earth Engine



Pros and Cons of Google Earth Engine

Pros	Cons
No downloading or local storage of imagery	Some programming required
Access to petabytes of data	Dependent on an internet connection
Google computing power	Primary focus on raster-based imagery
Advanced raster processing tools	Lack of mature and detailed documentation
Sharable codes and scripts	Limited to 250 GBs of user data upload

Fig. 4 Pro and cons of GEE

The pros and cons of Google Earth Engine (GEE) highlight both its strengths and limitations as a platform for large-scale geospatial analysis (Figure 4). On the positive side, GEE offers unparalleled access to vast datasets without the need for local storage, backed by Google's powerful computing infrastructure. Its advanced raster processing tools and ability to share codes and scripts make it a highly collaborative platform. However, the platform has some challenges, such as requiring programming skills, reliance on an internet connection, a primary focus on raster-based imagery, and limited user data uploads. Despite these drawbacks, GEE remains a valuable tool for researchers and practitioners in various fields.

1.3 How does it Work?

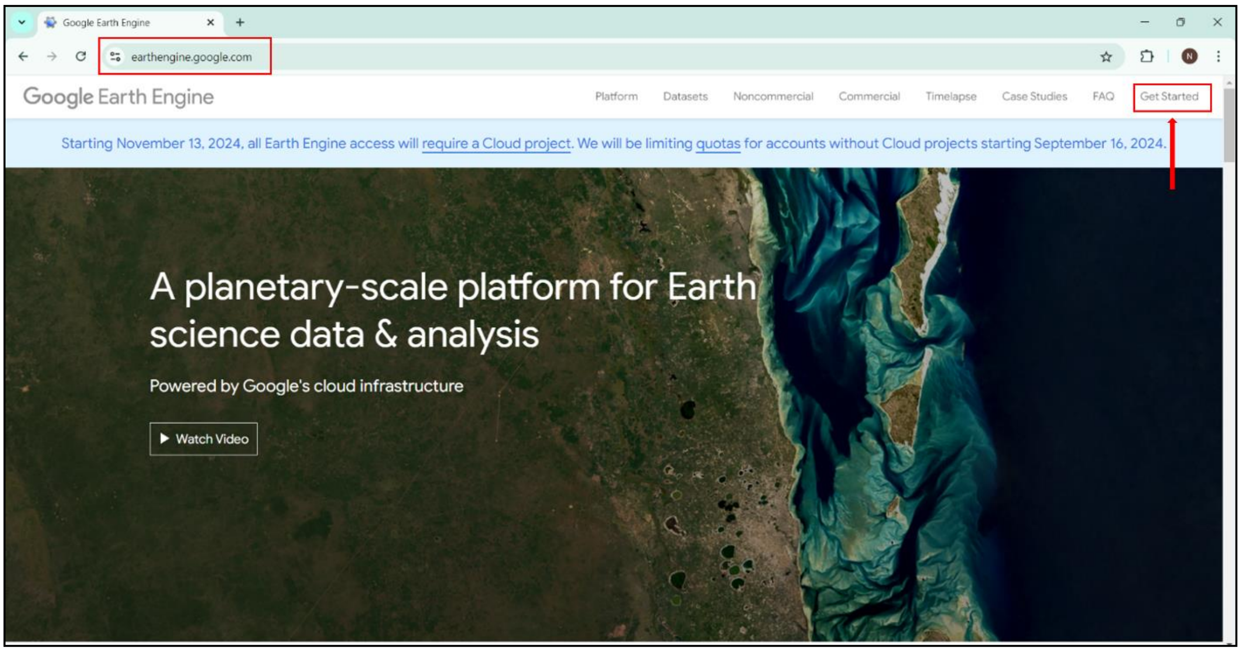
Google Earth Engine (GEE) is designed to democratize extensive-scale geospatial data analysis through its accessible, web-based platform. The registration process is straightforward, and once your account is verified, you gain access to GEE's powerful tools for satellite image analysis and visualization. With a simple sign-in process and free access, GEE is an ideal resource for researchers and professionals looking to explore spatiotemporal datasets, such as those used for analyzing Land Use and Land Cover (LULC) changes. To begin with Google Earth Engine, the steps are to be done:

Step 1: Visit the Google Earth Engine Website

- Go to the official GEE website at https://earthengine.google.org.

Step 2: Navigate to the Sign-Up Page

- On the homepage, click on **"Get Started"** or navigate to the **"Sign Up"** section.

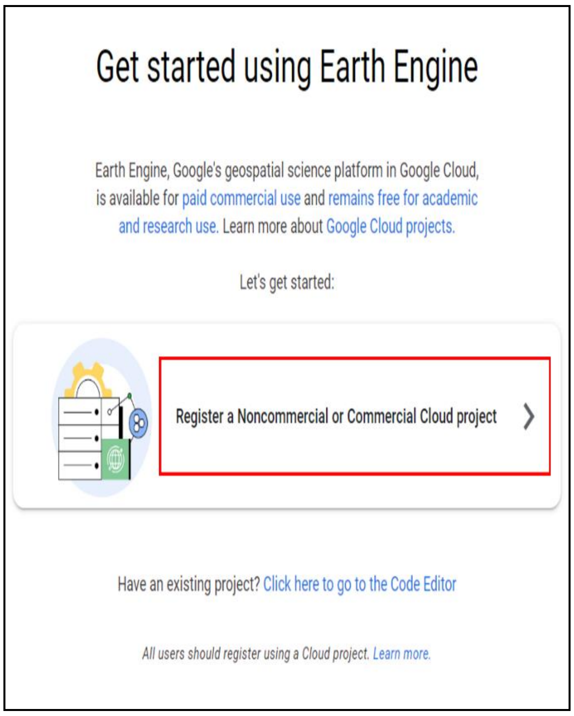


Starting November 13, 2024, all Earth Engine access will require a Cloud project. We will be limiting quotas for accounts without Cloud projects starting September 16, 2024.

Step 3: Get started using Earth Engine

• On the Product Registration page, select whether to register a Non-commercial or Commercial Cloud project.

• Next, specify your intended use of Earth Engine by selecting either Paid or Unpaid usage, depending on your requirements, and click "**Next**". For instance, we selected "**Unpaid usage**" under the "**Academia & Research**" category. Ensure that you choose the option that best aligns with your specific use case.



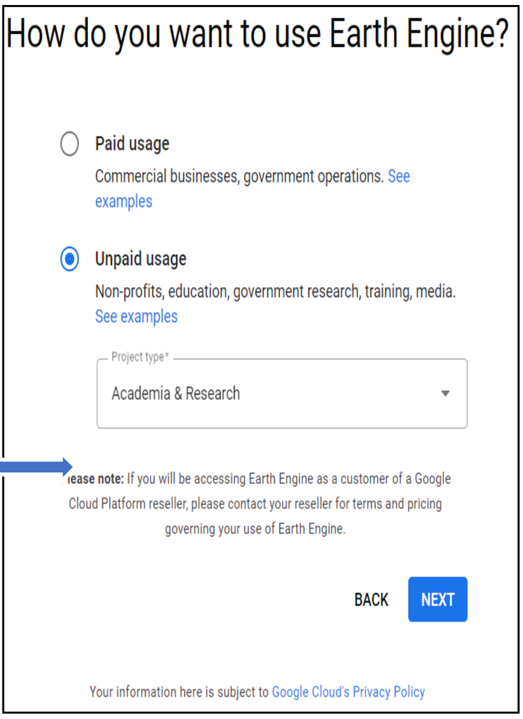
Get started using Earth Engine

Earth Engine, Google's geospatial science platform in Google Cloud, is available for paid commercial use and remains free for academic and research use. Learn more about Google Cloud projects.

Let's get started:

Have an existing project? Click here to go to the Code Editor

All users should register using a Cloud project. Learn more.



How do you want to use Earth Engine?

Paid usage
Commercial businesses, government operations. See examples

[✓ Unpaid usage]
Non-profits, education, government research, training, media. See examples

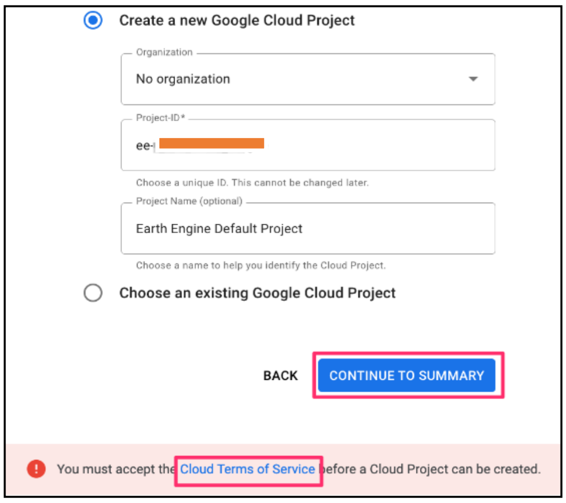
Project type*: Academia & Research

Please note: If you will be accessing Earth Engine as a customer of a Google Cloud Platform reseller, please contact your reseller for terms and pricing governing your use of Earth Engine.

Your information here is subject to Google Cloud's Privacy Policy

Step 4: Create a new Google Cloud Project

- In the following dialog, select "**Create a new Google Cloud Project**". For the "**Organization**" field, choose "**No organization**" and then provide a unique Project ID. After entering the ID, click "**Continue to Summary**".
- If you are leveraging Google Cloud for the first time, you may encounter an error message that will ask you to accept the agreements of Cloud Terms of Service before starting a new project. If prompted, click the provided link to open the Google Cloud Console and accept the terms.



[✓ Create a new Google Cloud Project]

Organization: No organization

Project-ID*: ee-[redacted]

Choose a unique ID. This cannot be changed later.

Project Name (optional): Earth Engine Default Project

Choose a name to help you identify the Cloud Project.

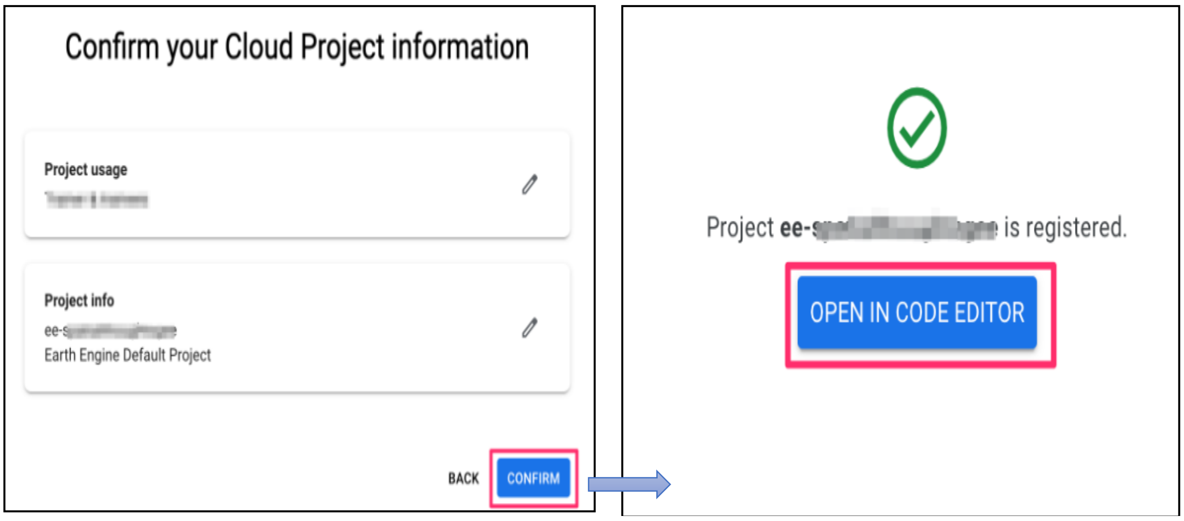
Choose an existing Google Cloud Project

You must accept the Cloud Terms of Service before a Cloud Project can be created.

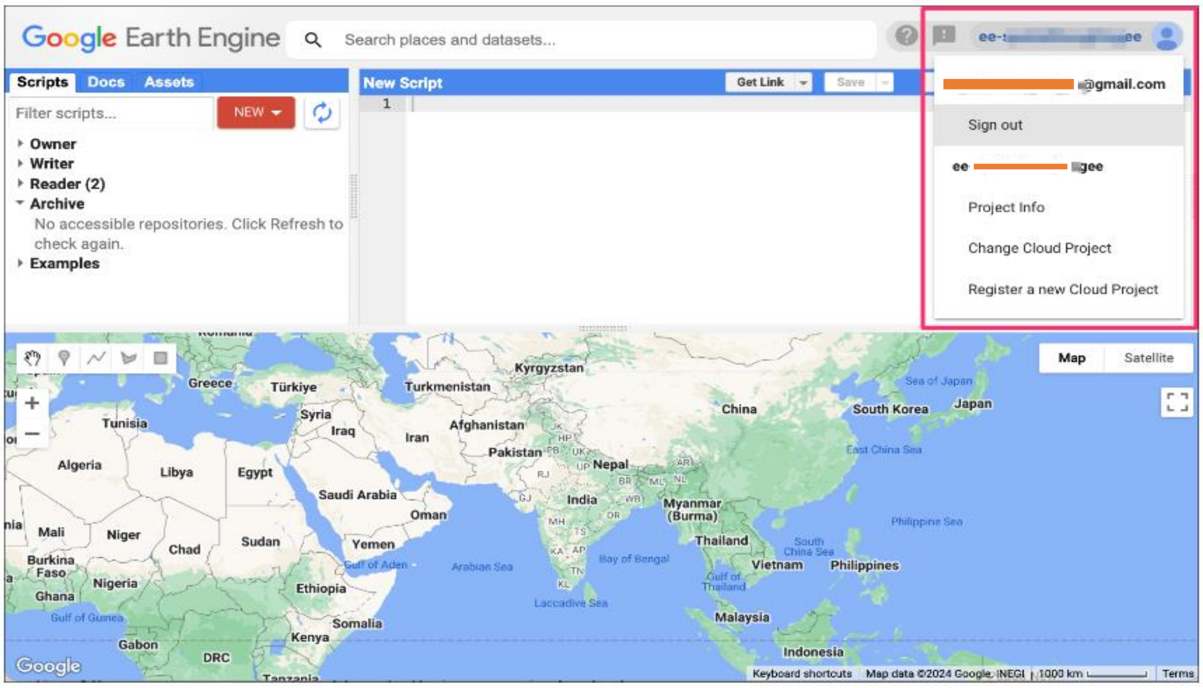
Step 5: Confirm your Cloud project information

- A summary of your project details will appear in the "**Confirm your Cloud project information**" dialog. Carefully review the information, then click "**Confirm**".

- Once the project is successfully registered, you will be redirected to the Code Editor. If this does not happen automatically, you can manually access the Code Editor by visiting https://code.earthengine.google.com/.


Confirm your Cloud Project information
Project usage
[redacted]
Project info
ee-[redacted]
Earth Engine Default Project
BACK
CONFIRM
Project ee-[redacted] is registered.
OPEN IN CODE EDITOR

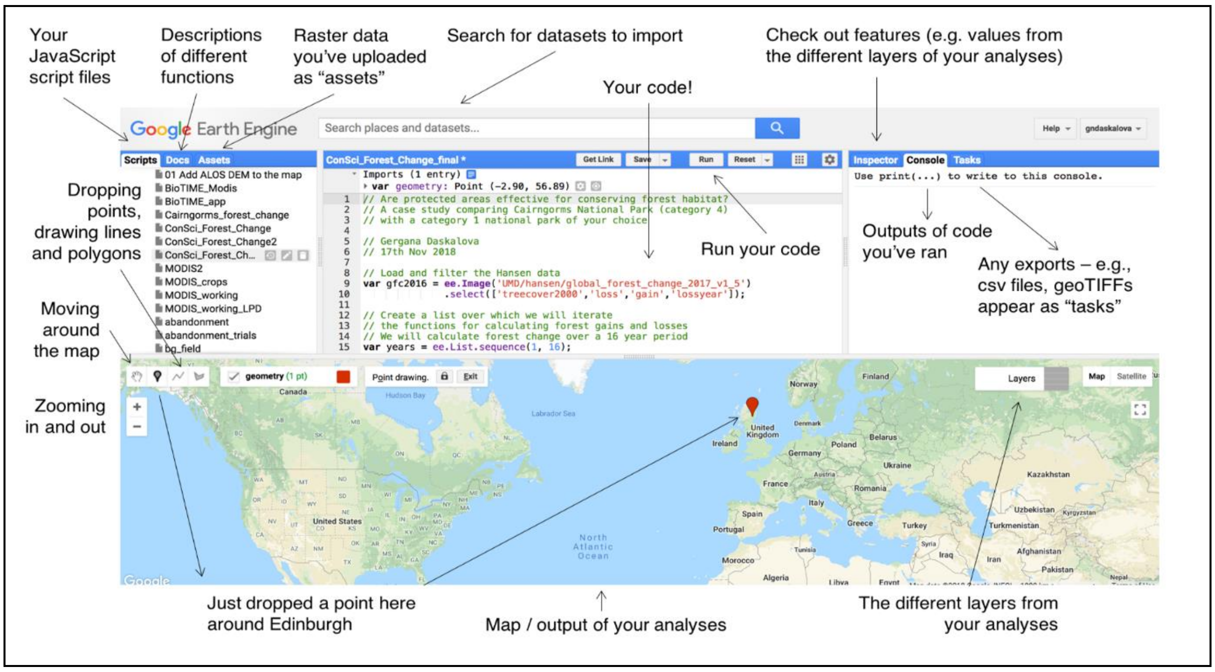
Step 6: The code editor will now display details about the linked project



Done! You are now set to begin utilizing your Earth Engine account with a Cloud Project.

1.4 Introduction to the code editor

The Google Earth Engine Code Editor is a robust web-based framework that enables users to write JavaScript code for the analysis of geospatial data. It facilitates seamless access to Google's extensive datasets, streamlines the processing of satellite imagery, and offers a range of tools for visualization and data export.


Your JavaScript script files
Descriptions of different functions
Raster data you've uploaded as "assets"
Search for datasets to import
Check out features (e.g. values from the different layers of your analyses)
Your code!
Google Earth Engine
Search places and datasets...
Help
gndaskalova
Scripts
Docs
Assets
ConSci_Forest_Change_final *
Get Link
Save
Run
Reset
Inspector
Console
Tasks
Dropping points, drawing lines and polygons
Run your code
Outputs of code you've ran
Any exports – e.g., csv files, geoTIFFs appear as "tasks"
Moving around the map
Zooming in and out
geometry (1 pt)
Point drawing.
Exit
Layers
Map
Satellite
Just dropped a point here around Edinburgh
Map / output of your analyses
The different layers from your analyses

01 Add ALOS DEM to the map
BioTIME_Modis
BioTIME_app
Cairngorms_forest_change
ConSci_Forest_Change
ConSci_Forest_Change2
ConSci_Forest_Ch...
MODIS2
MODIS_crops
MODIS_working
MODIS_working_LPD
abandonment
abandonment_trials
bg_field

```
Imports (1 entry)
  var geometry: Point (-2.90, 56.89)
// Are protected areas effective for conserving forest habitat?
// A case study comparing Cairngorms National Park (category 4)
// with a category 1 national park of your choice

// Gergana Daskalova
// 17th Nov 2018

// Load and filter the Hansen data
var gfc2016 = ee.Image('UMD/hansen/global_forest_change_2017_v1_5')
              .select(['treecover2000','loss','gain','lossyear']);

// Create a list over which we will iterate
// the functions for calculating forest gains and losses
// We will calculate forest change over a 16 year period
var years = ee.List.sequence(1, 16);
```

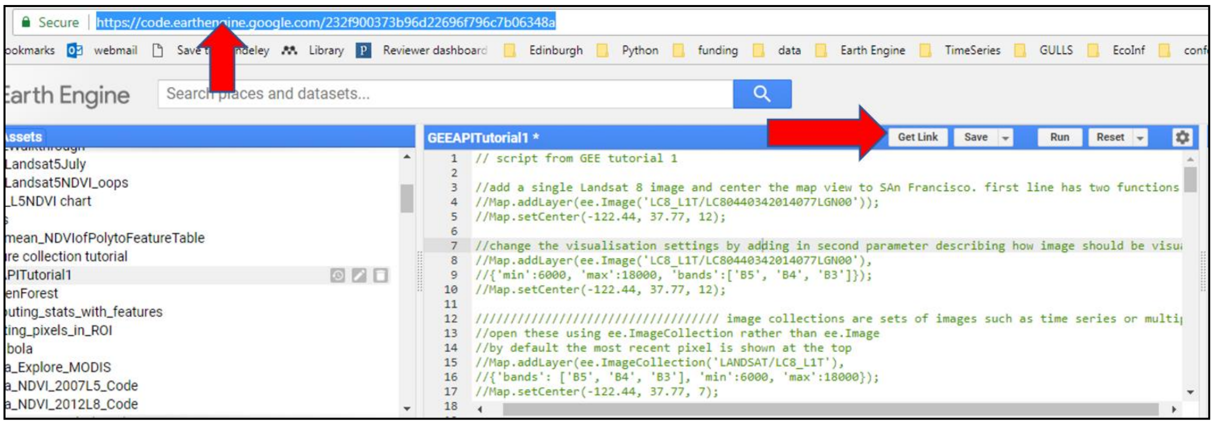
```
Use print(...) to write to this console.
```

As illustrated in the image, the left panel includes the Scripts, Docs, and Assets tabs, which facilitate the management of code and data. The central section is used for writing codes, where you may write and then execute scripts, while the results are shown in the Map Viewer placed below. On the right side, the Inspector/Console presents outputs from the executed code, and the Tasks tab oversees data exports. This web-based editor enhances the efficiency of spatiotemporal assessments, such as analyzing Land Use and Land Cover (LULC) changes within a specific region, by harnessing the cloud processing capabilities of Google Earth Engine.

1.4.1 Script Sharing

In Google Earth Engine, the process of sharing scripts is straightforward, fostering collaboration and providing easy access to geospatial analyses. By creating a shareable link, users can distribute their scripts, allowing others to execute, modify, or review the code directly within their own Code Editor. This functionality enhances teamwork and promotes knowledge exchange across various projects. To share a script in Google Earth Engine, follow these steps:

- **Open the script** you wish to share in the section in Code Editor.
- Click the "**Get Link**" button located at the top of the editor.
- A shareable link will be generated; **copy** this link.
- **Send** the link to anyone you would like to share the script with.

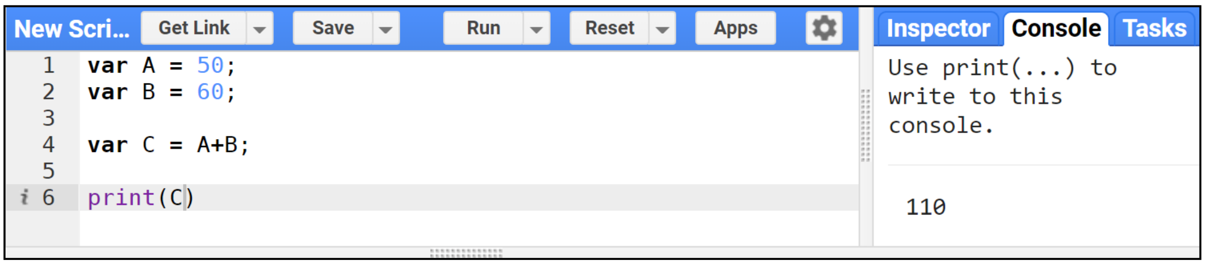

https://code.earthengine.google.com/232f900373b96d22696f796c7b06348a
Earth Engine
Search places and datasets...
Assets
GEEAPITutorial1 *
Get Link
Save
Run
Reset

```
// script from GEE tutorial 1

//add a single Landsat 8 image and center the map view to SAn Francisco. first line has two functions
//Map.addLayer(ee.Image('LC8_L1T/LC80440342014077LGN00'));
//Map.setCenter(-122.44, 37.77, 12);

//change the visualisation settings by adding in second parameter describing how image should be visu
//Map.addLayer(ee.Image('LC8_L1T/LC80440342014077LGN00'),
//{'min':6000, 'max':18000, 'bands':['B5', 'B4', 'B3']});
//Map.setCenter(-122.44, 37.77, 12);

////////////////////////////////// image collections are sets of images such as time series or multi
//open these using ee.ImageCollection rather than ee.Image
//by default the most recent pixel is shown at the top
//Map.addLayer(ee.ImageCollection('LANDSAT/LC8_L1T'),
//{'bands': ['B5', 'B4', 'B3'], 'min':6000, 'max':18000});
//Map.setCenter(-122.44, 37.77, 7);
```

1.4.2 Client vs. Server object

Earth Engine client libraries for Python and JavaScript facilitate the translation of intricate geospatial analyses into requests for Earth Engine. The code written for a client library may encompass a blend of references to client-side objects and variables that denote server-side objects.

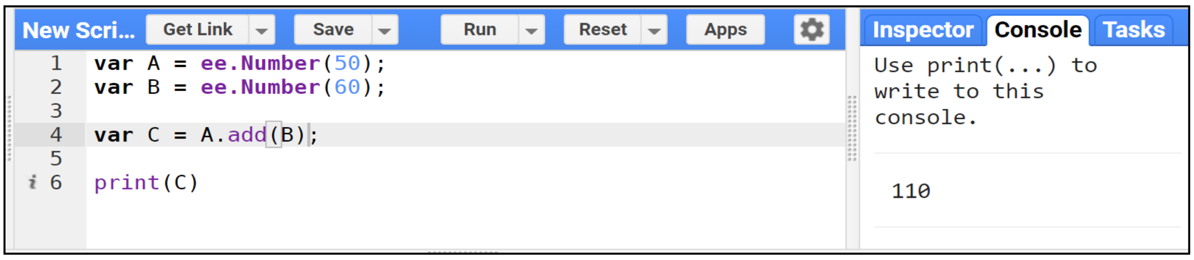
- "**Client-side**" refers to actions executed on the user's (the client's) computer.
- The user platform components within your Code Editor, such as the Drawing Tools and Map View, are classified as 'client-side' elements, as they operate within the browser.



```
var A = 50;
var B = 60;

var C = A+B;

print(C)
```

```
Use print(...) to
write to this
console.

110
```

- "**Server-side**" variables denotes that the action is performed on a web server.
- Conversely, image collections, feature collections, and calculations on Earth Engine objects are categorized as 'server-side' elements, executing within Google's data centres. It is important to note that these two types of objects cannot be mixed.



```
var A = ee.Number(50);
var B = ee.Number(60);

var C = A.add(B);

print(C)
```

```
Use print(...) to
write to this
console.

110
```

To alter client-side entities into server-side, you need to utilize the relevant API functions, which are prefixed with "ee.", like "ee.Date()" and "ee.Image()". Conversely, to transform server-side entities into client-side entities, you may invoke the "getInfo()" method on an Earth Engine entities.

1.4.3 What can you do in Earth Engine?

In Google Earth Engine (GEE), you can perform a wide array of advanced geospatial analyses as mentioned in the following (Table 1):

Table 1. Possible analysis processes and their functions in GEE

Analysis Process	Functions
Image Processing	Map Algebra, Kernels and Convolutions, Spectral Unmixing, Pan-sharpening, Gap Filling, Data Fusion
Vector Processing	Zonal Statistics, Spatial Joins, Spatial Query etc.
Terrain Processing	Slope, Aspect, Hillshade, Hill Shadow Analysis
Time Series Analysis	Extract Time-Series, Trend Analysis, Time-Series Smoothing, Temporal Segmentation etc.
Object-based Image Analysis	GLCM, Texture, Hotspots etc.
Change Detection	Spectral Distance, Change Classification, Class Transitions
Machine Learning	Supervised and Unsupervised Classification, Linear Regression, Principal Components Analysis etc.
Deep Learning	DNN, Object Detection etc. via TensorFlow

1.4.4 What you cannot do in Earth Engine?

While Google Earth Engine (GEE) is a robust framework for performing geospatial analysis, there are certain limitations to its capabilities (Table 2). Some tasks require external tools or software to complement GEE's functionality:

Table 2. Analysis processes cannot perform in GEE

Analysis Process	Functions
Create Cartographic Outputs	Have to use the GEE Plugin in QGIS to create maps
3D visualization and analysis	N/A
Run Hydrological models (e.g., Rainfall-runoff modelling) and analyses (e.g., watershed delineation, fill depression)	N/A
Photogrammetry (e.g., Orthorectification, Point-Clouds)	N/A
LIDAR processing	N/A
SAR Interferometry (Earth Engine does not support images with complex values)	N/A

1.5 The Earth engine public data catalog

The Earth Engine public data catalog is a huge collection of geospatial datasets, totaling multiple petabytes, that many users rely on. It primarily consists of Earth-observing remote sensing images, including the entire Landsat archive and all data from Sentinel-1 and Sentinel-2. Besides that, it offers climate forecasts, land cover information, and a variety of other datasets related to the environment, geophysics, and socio-economic factors (Table 3).

Earth Engine uses a simple data model based on 2D raster bands stored in lightweight "image" containers. Each pixel in a band needs to be consistent in data type, resolution, and projection. However, images can have multiple bands that don’t necessarily have to match in these aspects. Each image can also come with key/value metadata, which provides important details about where and when the image was captured and the conditions under which it was collected or processed.

Table 3. Commonly utilized datasets within the Earth Engine data catalog

Datasets	Spatial Resolution	Temporal Resolution	Data availability in GEE
Sentinel			
Sentinel-1	10	6 days	2014 - present
Sentinel-2	10/20m	5 days	2015 - present
Landsat			
Landsat-8	30 m	16 days	2013 - present
Landsat-7	30 m	16 days	2000 - present
Landsat-5	30 m	16 days	1984 - 2012
Landsat – 4 - 8	30 m	16 days	1984 - present
MODIS			
MOD08	1°	1 day	2000 - Present
MOD09	500 m	1day/8day	2000 - Present
MOD10	500 m	1day	2000 - Present
MOD11	1000 m	1day/8day	2000 - Present
MOD12	500 m	Annual	2000 - Present
Topography			
SRTM	30 m	Single	2000
GTOPO30	30″	Single	Multiple
Landcover			
GlobCover	300m	Non-periodic	2009
JRC global surface water	30m	Monthly	1984–2015
USGS National Landcover	30m	Non-periodic	1992–2011
USDA NASS cropland data	30m	Annual	1997–2015
Weather, Precipitation & atmosphere			
CHIRPS	5 km	5 days	1981 - Present
GPM	11 km	3 hours	2014 - present
TRMM	27 km	3 hours	1998 - 2015
GLDAS	0.25°	3 hours	2000 - Present
Population			
WorldPop	100m	5year	2010–2015

Source: Gorelick et al. (2017) & https://developers.google.com/earth-engine/datasets/

1.6 Filtering and Displaying Data

In this section, we will walk through a step-by-step tutorial on key operations in Google Earth Engine (GEE) for analyzing and visualizing satellite imagery. The tutorial will cover essential tasks such as filtering and displaying satellite images from Landsat datasets, importing raster and vector data, and performing calculations like clipping, reducing, and displaying data. Additionally, you will learn how to remove cloud cover from satellite images, calculate important spectral indices (e.g. NDVI, and NDWI), and enhance your maps with legends and titles to make your visualizations more informative and complete.

1.6.1 Data Importing Process in Google Earth Engine

In Google Earth Engine (GEE), you can import data using two main methods: via the **Data Catalog** or through **Assets**. Here's a brief explanation of both methods:

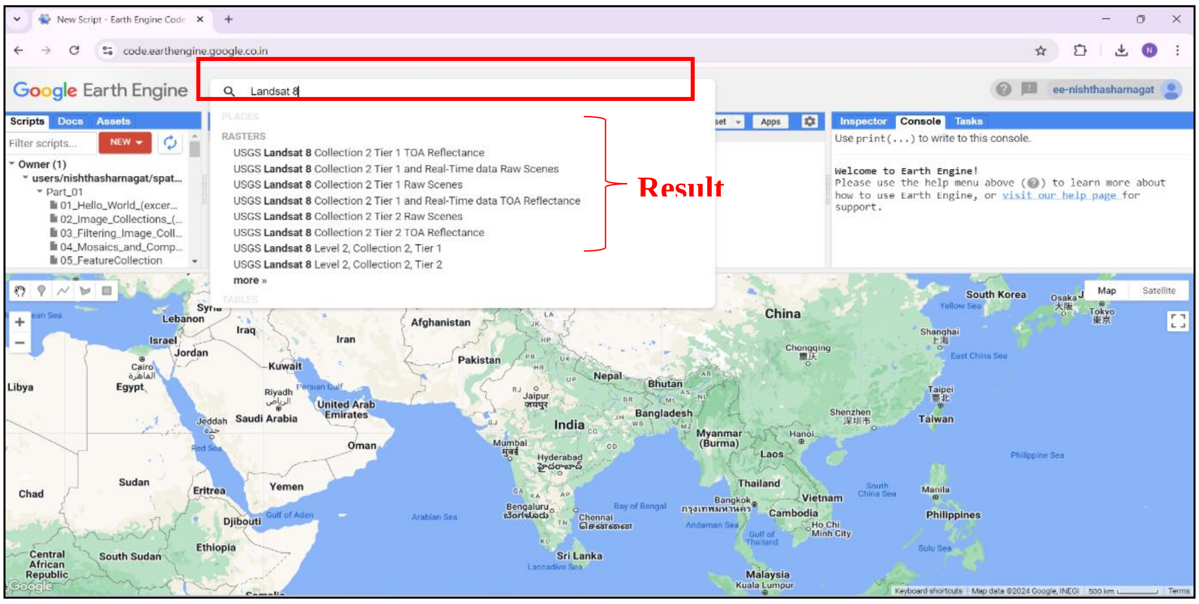
Method 1: Importing Data from the Data Catalog:
The **Data Catalog** in GEE contains numerous publicly available datasets, that include satellite data, atmosphere data, land use and land cover maps, and many more.

Step 1: Open the Data Catalog:

- In Code Editor, click on the "**Search**" option to browse the datasets that are already available in the Google Earth Engine public data catalog.

Step 2: Search for a Dataset:

- You can search for *Landsat*, *Sentinel*, *MODIS or any other dataset*, for example, Landsat 8 data is used here.

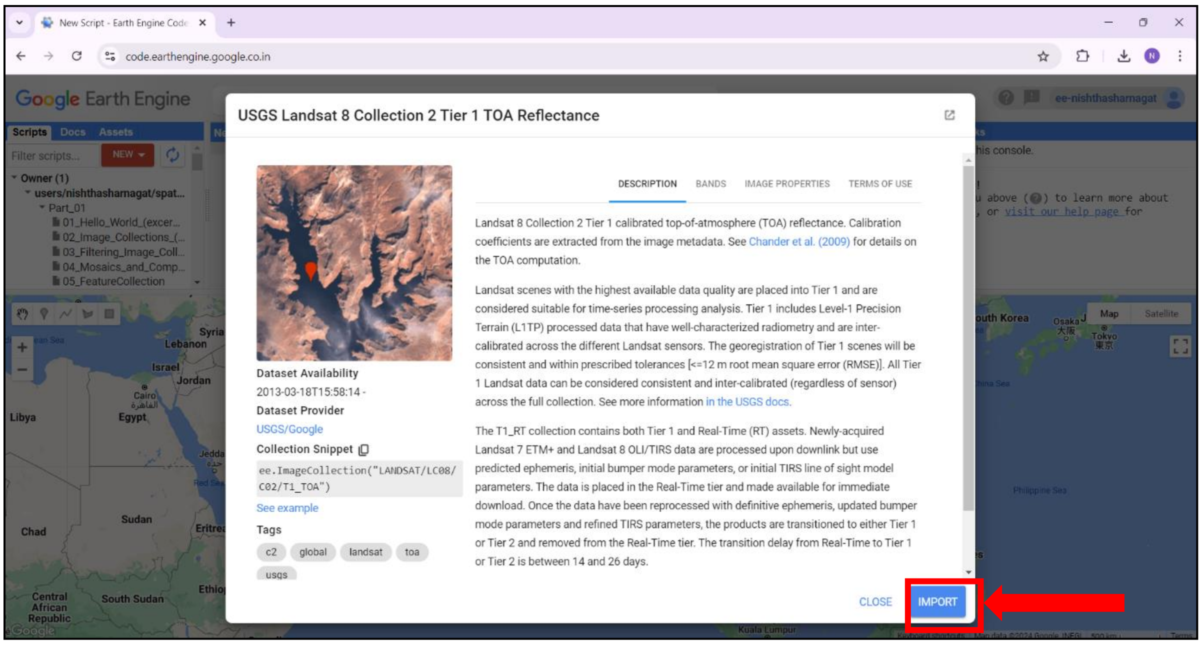


RASTERS
USGS **Landsat 8** Collection 2 Tier 1 TOA Reflectance
USGS **Landsat 8** Collection 2 Tier 1 and Real-Time data Raw Scenes
USGS **Landsat 8** Collection 2 Tier 1 Raw Scenes
USGS **Landsat 8** Collection 2 Tier 1 and Real-Time data TOA Reflectance
USGS **Landsat 8** Collection 2 Tier 2 Raw Scenes
USGS **Landsat 8** Collection 2 Tier 2 TOA Reflectance
USGS **Landsat 8** Level 2, Collection 2, Tier 1
USGS **Landsat 8** Level 2, Collection 2, Tier 2
more »

```
Use print(...) to write to this console.

Welcome to Earth Engine!
Please use the help menu above (?) to learn more about
how to use Earth Engine, or visit our help page for
support.
```

Step 3: Select and Import the Dataset:

- Click on the dataset you want and hit the "Import" button. This will load the dataset as an ee.ImageCollection in your code editor. Same process will be use to import any vector dataset.


USGS Landsat 8 Collection 2 Tier 1 TOA Reflectance

Landsat 8 Collection 2 Tier 1 calibrated top-of-atmosphere (TOA) reflectance. Calibration coefficients are extracted from the image metadata. See Chander et al. (2009) for details on the TOA computation.

Landsat scenes with the highest available data quality are placed into Tier 1 and are considered suitable for time-series processing analysis. Tier 1 includes Level-1 Precision Terrain (L1TP) processed data that have well-characterized radiometry and are inter-calibrated across the different Landsat sensors. The georegistration of Tier 1 scenes will be consistent and within prescribed tolerances [<=12 m root mean square error (RMSE)]. All Tier 1 Landsat data can be considered consistent and inter-calibrated (regardless of sensor) across the full collection. See more information in the USGS docs.

The T1_RT collection contains both Tier 1 and Real-Time (RT) assets. Newly-acquired Landsat 7 ETM+ and Landsat 8 OLI/TIRS data are processed upon downlink but use predicted ephemeris, initial bumper mode parameters, or initial TIRS line of sight model parameters. The data is placed in the Real-Time tier and made available for immediate download. Once the data have been reprocessed with definitive ephemeris, updated bumper mode parameters and refined TIRS parameters, the products are transitioned to either Tier 1 or Tier 2 and removed from the Real-Time tier. The transition delay from Real-Time to Tier 1 or Tier 2 is between 14 and 26 days.

Dataset Availability
2013-03-18T15:58:14 -
Dataset Provider
USGS/Google
Collection Snippet

```
ee.ImageCollection("LANDSAT/LC08/C02/T1_TOA")
```

See example
Tags
c2 global landsat toa usgs

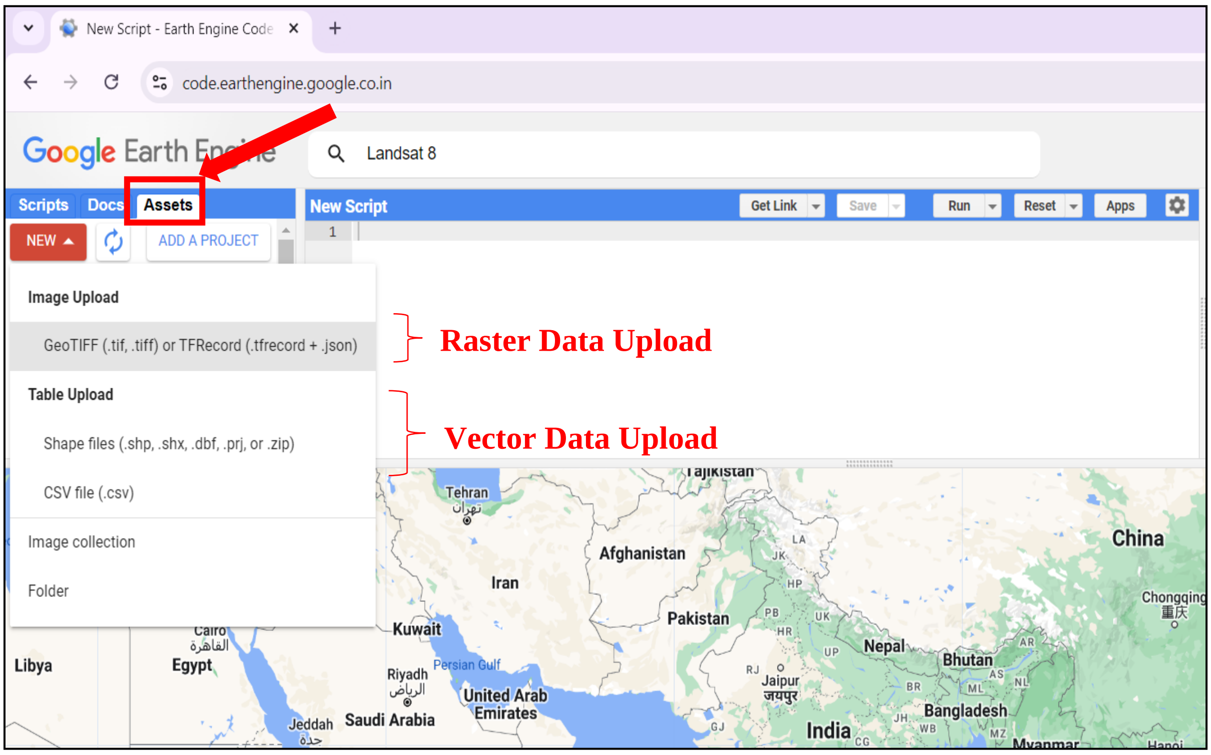
Method 2: Steps to importing Data from the Assets:

Step 1: Navigate “Assets” button in code editor

- Click on the "Assets" tab.

Step 2: Click “New” button to upload data

- Click the "New" button, then choose whether to upload an image (raster), table (vector), or other supported data types.

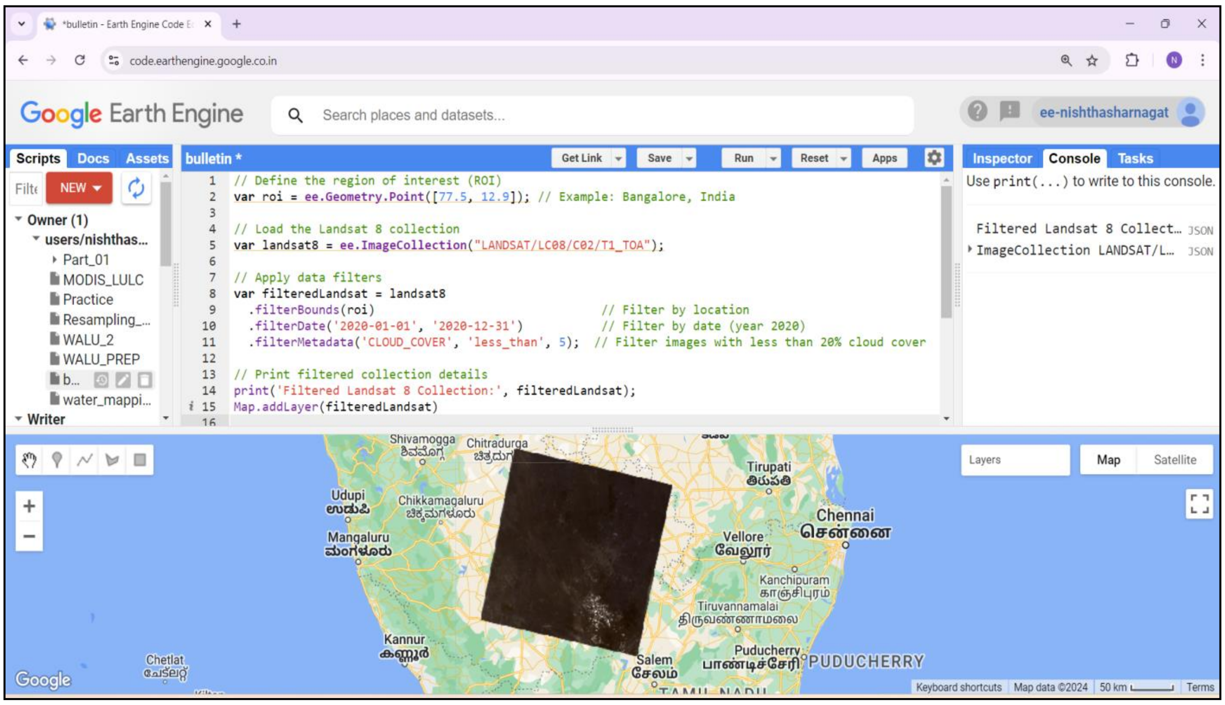


1.6.2 Data Filtering and visualization

In Google Earth Engine (GEE), data filtering and visualization are essential steps for narrowing down datasets and displaying them effectively. Here's a breakdown of key functions used for these purposes:

➢ **Data Filtering Functions**

- **filterBounds(geometry)**: Filters datasets based on a specific geographic boundary. The geometry argument can be a point, line, polygon, or other geometric shape.
- **filterDate(start, end)**: Filters datasets by date range. This function narrows down temporal datasets (like satellite imagery) within a specified period.
- **filter(ee.Filter.eq(property, value))**: Filters datasets based on a property and its value. For example, it can be used to filter images with specific cloud cover percentages or certain land use types.
- **filterMetadata(name, operator, value)**: Similar to filter(), but this is more specialized for filtering datasets based on metadata fields (e.g., cloud cover or land cover type).



```
// Define the region of interest (ROI)
var roi = ee.Geometry.Point([77.5, 12.9]); // Example: Bangalore, India

// Load the Landsat 8 collection
var landsat8 = ee.ImageCollection("LANDSAT/LC08/C02/T1_TOA");

// Apply data filters
var filteredLandsat = landsat8
  .filterBounds(roi)                              // Filter by location
  .filterDate('2020-01-01', '2020-12-31')          // Filter by date (year 2020)
  .filterMetadata('CLOUD_COVER', 'less_than', 5);  // Filter images with less than 20% cloud cover

// Print filtered collection details
print('Filtered Landsat 8 Collection:', filteredLandsat);
Map.addLayer(filteredLandsat)
```

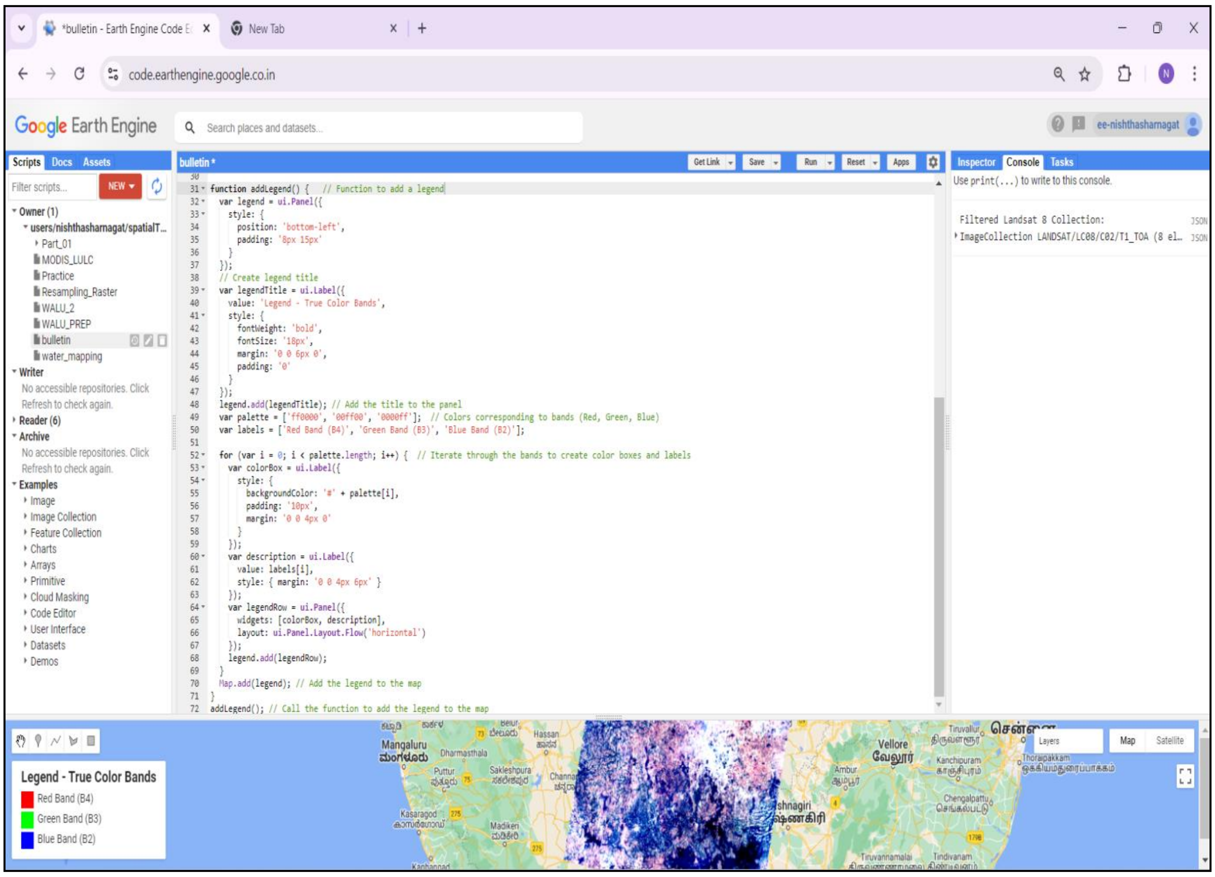
```
Use print(...) to write to this console.

Filtered Landsat 8 Collect…   JSON
▸ImageCollection LANDSAT/L…   JSON
```

➢ **Data Visualization Functions**

- **Map.addLayer(image, visParams, name)**: Adds an image layer to the map for visualization. The visParams argument controls the display settings (e.g., bands to display, color scales).
- **Map.centerObject(object, zoom)**: Centers the map view on a specific object, like a point or an image, at a certain zoom level.
- **Map.setCenter(lon, lat, zoom)**: Directly sets the map's center coordinates and zoom level.
- **image.visualize()**: Converts an image into an RGB or grayscale visualization, with parameters like min, max, palette, etc.

- **Map.addLegend() (custom)**: Though GEE does not have a built-in legend function, custom legends can be added using JavaScript/HTML elements. A legend provides context for map colours.

These functions help refine data selection and visually represent results for clearer analysis.



```
function addLegend() {   // Function to add a legend
  var legend = ui.Panel({
    style: {
      position: 'bottom-left',
      padding: '8px 15px'
    }
  });
  // Create legend title
  var legendTitle = ui.Label({
    value: 'Legend - True Color Bands',
    style: {
      fontWeight: 'bold',
      fontSize: '18px',
      margin: '0 0 6px 0',
      padding: '0'
    }
  });
  legend.add(legendTitle); // Add the title to the panel
  var palette = ['ff0000', '00ff00', '0000ff'];  // Colors corresponding to bands (Red, Green, Blue)
  var labels = ['Red Band (B4)', 'Green Band (B3)', 'Blue Band (B2)'];

  for (var i = 0; i < palette.length; i++) {  // Iterate through the bands to create color boxes and labels
    var colorBox = ui.Label({
      style: {
        backgroundColor: '#' + palette[i],
        padding: '10px',
        margin: '0 0 4px 0'
      }
    });
    var description = ui.Label({
      value: labels[i],
      style: { margin: '0 0 4px 6px' }
    });
    var legendRow = ui.Panel({
      widgets: [colorBox, description],
      layout: ui.Panel.Layout.Flow('horizontal')
    });
    legend.add(legendRow);
  }
  Map.add(legend); // Add the legend to the map
}
addLegend(); // Call the function to add the legend to the map
```

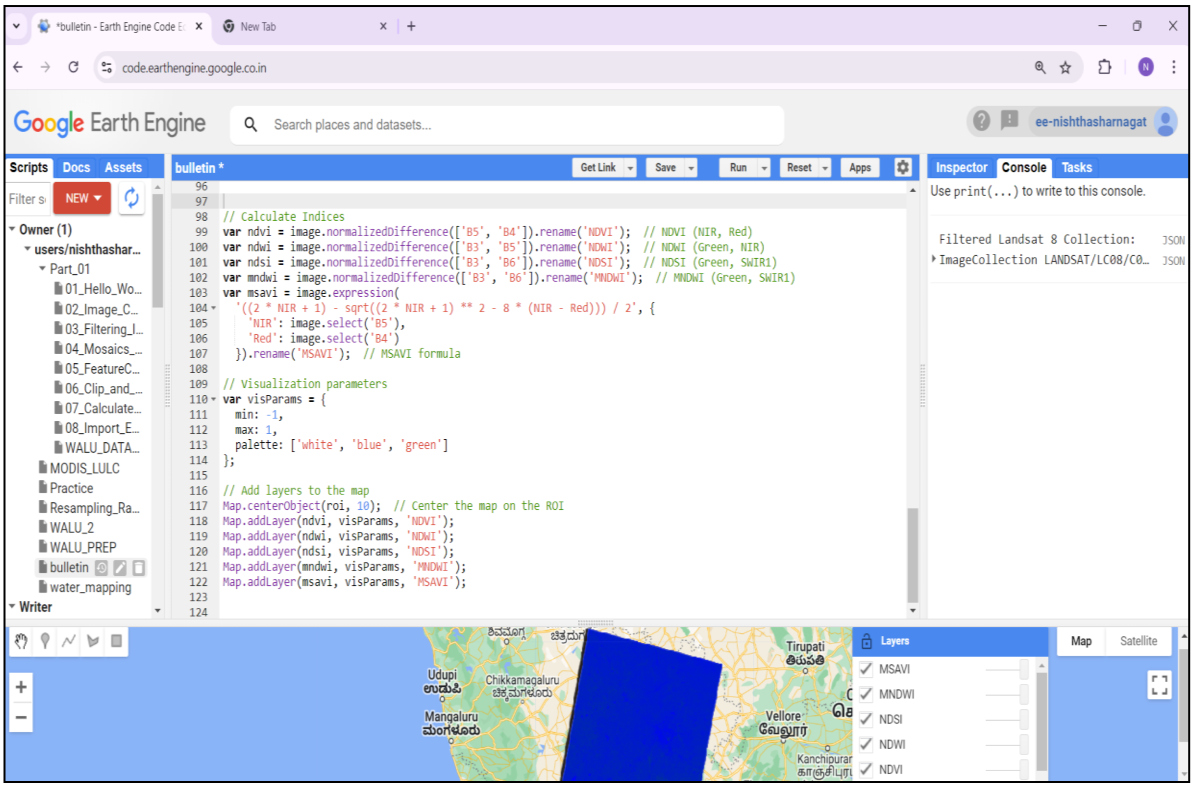
Use print(...) to write to this console.

```
Filtered Landsat 8 Collection:
▸ImageCollection LANDSAT/LC08/C02/T1_TOA (8 el…
```

Legend - True Color Bands
🟥 Red Band (B4)
🟩 Green Band (B3)
🟦 Blue Band (B2)

1.6.3 Calculation of Spectral Indices

Calculating various indices using remote sensing data is essential for using land cover, crop health, and water resources. In Earth Engine (EE), the application of normalized difference indices enables researchers to get useful and important information from satellite images. The provided code snippet illustrates the calculation of several important indices, including the Normalized Difference Vegetation Index (NDVI), Normalized Difference Snow Index (NDSI), Normalized Difference Water Index (NDWI), Modified Soil Adjusted Vegetation Index (MSAVI), and the Modified Normalized Difference Water Index (MNDWI). Each index is computed using specific spectral bands, which are normalized to highlight particular features within the study area. The resulting layers are then visualized on the map, providing insights into the spatial distribution of vegetation, water, and snow cover, which are crucial for environmental monitoring and land management practices.



```
// Calculate Indices
var ndvi = image.normalizedDifference(['B5', 'B4']).rename('NDVI');  // NDVI (NIR, Red)
var ndwi = image.normalizedDifference(['B3', 'B5']).rename('NDWI');  // NDWI (Green, NIR)
var ndsi = image.normalizedDifference(['B3', 'B6']).rename('NDSI');  // NDSI (Green, SWIR1)
var mndwi = image.normalizedDifference(['B3', 'B6']).rename('MNDWI');  // MNDWI (Green, SWIR1)
var msavi = image.expression(
  '((2 * NIR + 1) - sqrt((2 * NIR + 1) ** 2 - 8 * (NIR - Red))) / 2', {
    'NIR': image.select('B5'),
    'Red': image.select('B4')
  }).rename('MSAVI');  // MSAVI formula

// Visualization parameters
var visParams = {
  min: -1,
  max: 1,
  palette: ['white', 'blue', 'green']
};

// Add layers to the map
Map.centerObject(roi, 10);  // Center the map on the ROI
Map.addLayer(ndvi, visParams, 'NDVI');
Map.addLayer(ndwi, visParams, 'NDWI');
Map.addLayer(ndsi, visParams, 'NDSI');
Map.addLayer(mndwi, visParams, 'MNDWI');
Map.addLayer(msavi, visParams, 'MSAVI');
```

Use print(...) to write to this console.

```
Filtered Landsat 8 Collection:                JSON
▸ImageCollection LANDSAT/LC08/C0…              JSON
```

Chapter 2.0: Machine Learning

2.1 Overview of ML in GEE

ML that falls under one of the components of artificial intelligence (AI) highlights on developing algorithms that enable ML models to take decisions or predictions based on data (Huang and Jensen, 1997; Maxwell et al., 2018). These approaches have proven effective for processing remotely sensed data and constitute a fundamental component of the computational algorithms employed by the Google Earth Engine (GEE) platform. The integration of process-oriented computational tuning tools by Google aims to streamline workflows, reducing the need for offline processing (Schulz et al., 2018; Zhou et al., 2020).

ML algorithms are trained using labelled, unlabelled, or hybrid datasets, whereby the specific data requirements for a task inform the type of machine learning model developed (Alnuaimi & Albaldawi, 2024). Broadly, ML is categorized in three types: unsupervised learning, supervised learning, and reinforcement learning. Each of these approaches is applied in various contexts and with diverse datasets. Figure 5 illustrates the classification of machine learning algorithms.

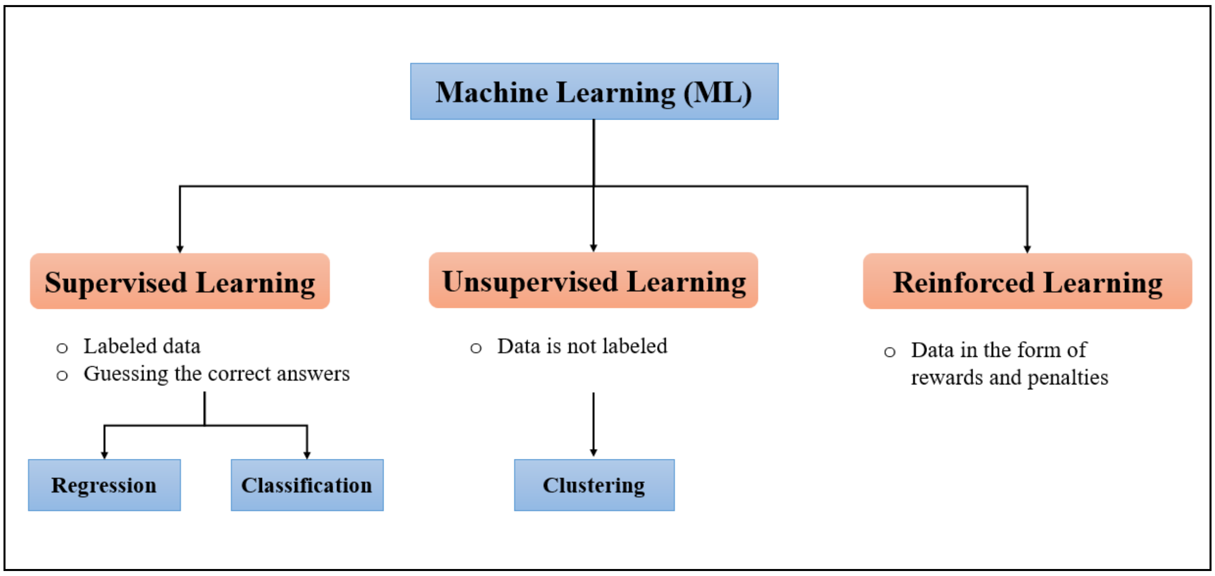


Fig. 5 Classification of Machine Learning Algorithm

2.1.1 Common terminology involved in Machine Learning (GIS)

Labeled data: This refers to datasets comprising a collection of training examples, where each example is represented as a pair containing an input and its corresponding desired output value, incorporating both attributes and labels.

Classification: The objective of many machine learning projects is to predict discrete values, such as True or False, 1 or 0, or categories like Forest or Not Forest.

Regression: This type of ML project aims to forecast continuous values, such as the percentage of land cover.

Clustering: This is an unsupervised ML technique designed to group unlabelled examples depending on their similarities. When examples are labeled, this process is referred to as classification.

2.2 What is a classifier?

A classifier is an algorithm that categorizes data points (pixels) into predefined classes based on their attributes. In Google Earth Engine (GEE), classifiers are utilized to classify satellite imagery into meaningful (LULC) types. Algorithms responsible for executing the classification of data are called classifiers. There are two primary types of classifiers:

- **Binary Classifier:** Used when the classification problem presents only two possible outcomes. Examples include YES or NO, MALE or FEMALE, CAT or DOG, SPAM or NOT SPAM, and.
- **Multi-class Classifier:** Used when a classification involves more than two possible outcomes. This may include the classification of crop types, the categorization of music genres, and LULC classification.

2.3 Supervised Learning

Supervised learning (SL) is an ML paradigm that utilizes labelled datasets to train a system to predict outcomes based on its prior training. This approach closely resembles human learning, where an instructor guides the learner using specific examples to derive broader principles (Alnuaimi & Albaldawi, 2024).

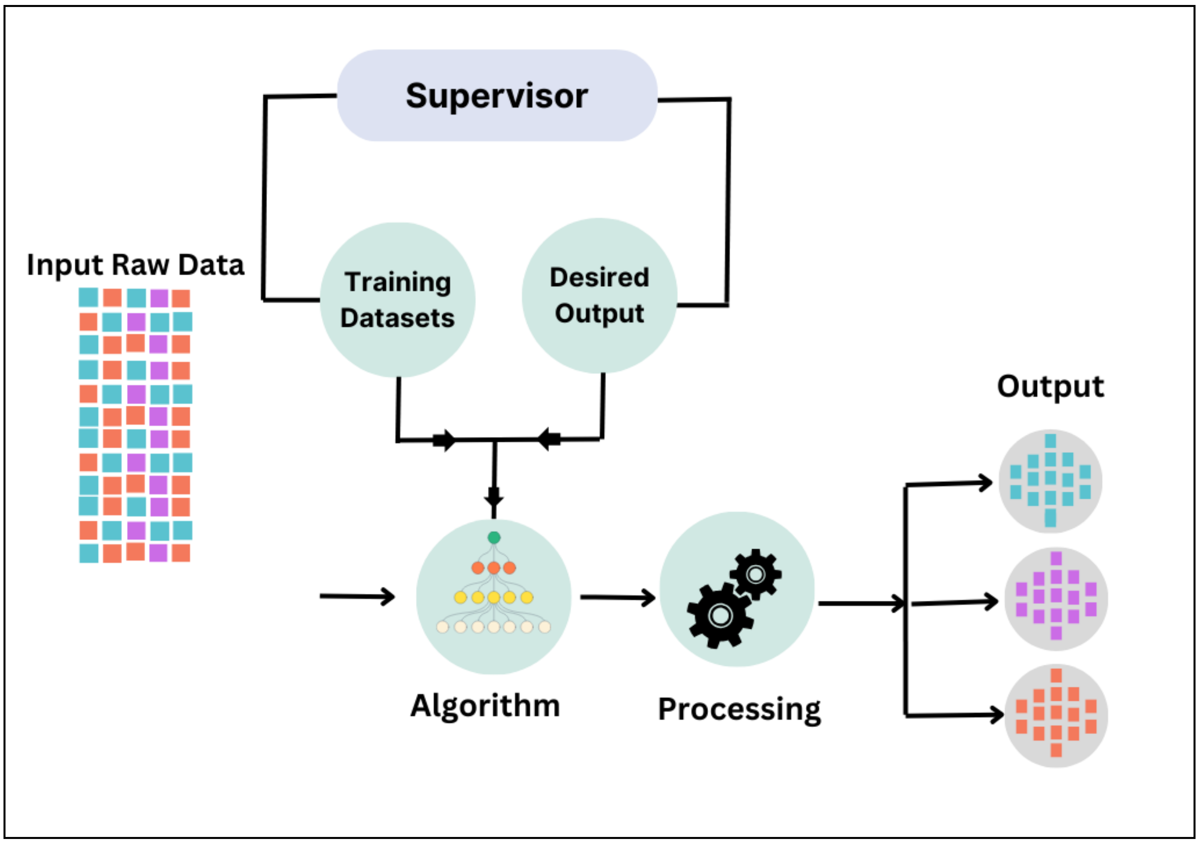


Fig. 6 General overview of supervised ML technique

In the context of land classification, this machine learning technique employs ground truth examples to train a model to distinguish between different classes. Google Earth Engine's integrated supervised classifiers facilitate this process. Figure 6 illustrates the general overview of the supervised ML technique. Here are some key points related to supervised ML technique:

- **Input Data**: The data in supervised learning includes both input features and labeled output. For example, in a house price prediction model, the features could be square footage, location, etc., while the output is the price.
- **When to Use**: Supervised learning is suitable when you have clear labels and outcomes you are trying to predict. It is used in scenarios where the output is known and can be used to train the model.
- **Nature of Problem**: Commonly used for **regression** (for predicting continuous outcomes) and **classification** (categorizing data into specific labels or classes).
- **Goal**: The goal is to train the model on labeled data that is used for making correct predictions for unseen data. For instance, predicting whether an email is spam based on training data labeled as "spam" or "not spam."
- **Output**: The model provides predicted labels based on the input data.
- **Associated Algorithms**: Algorithms mainly include Linear Regression, Support Vector Machines (SVM), Logistic Regression, and K-Nearest Neighbors (KNN), each suited for different types of regression and classification problems.
- **Proximity to AI**: Supervised learning is further from AI, as it relies more on human-labeled data, making it less autonomous in finding patterns compared to other learning types like unsupervised learning.
- **Drawbacks**: The major drawback is the requirement for a large number of labelled datasets, which can be time-consuming and expensive. Training can also be slow and resource-intensive.
- **Expected Accuracy**: Models trained using supervised learning can achieve high accuracy because they are provided with clear examples during training. This is why it is widely used in high-stakes tasks like medical diagnosis or financial predictions.

2.4 Unsupervised Learning

Unsupervised learning entails training a machine using only input samples or labels, without any prior knowledge of the output. This method enables the discovery of patterns within the data and the formation of its own data clusters. It is particularly beneficial for identifying unknown patterns, such as in recommendation engines utilized by online retailers, which often employ unsupervised machine learning techniques, specifically clustering (Alnuaimi & Albaldawi, 2024).

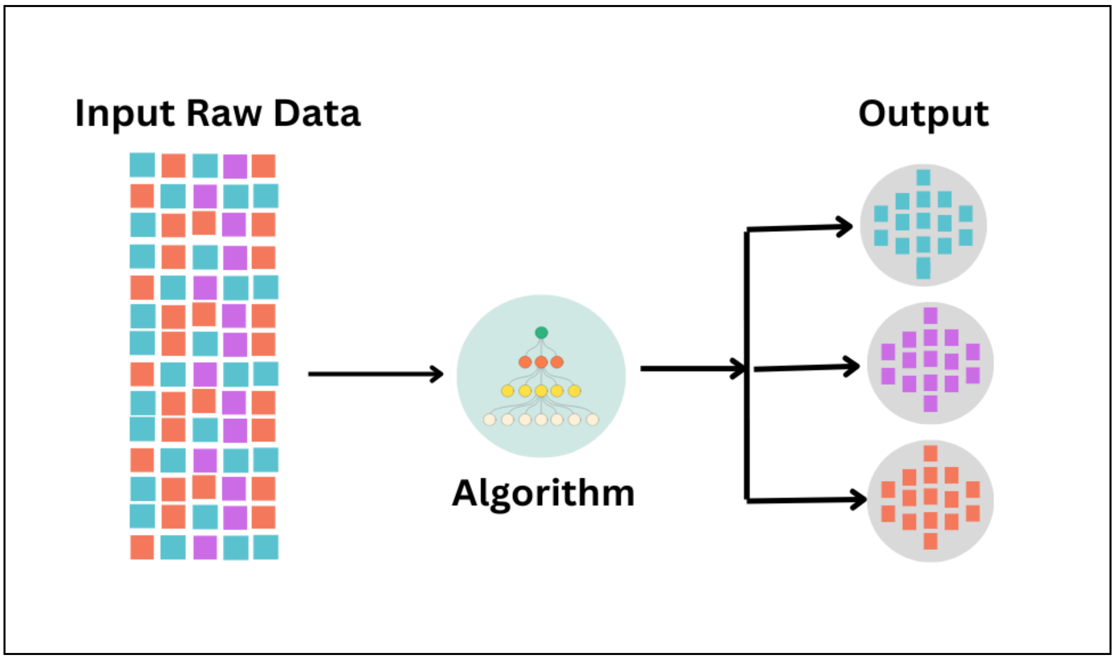


Fig. 7 General overview of unsupervised ML technique

In unsupervised classification, the training algorithm operates without any ground truth examples. Instead, it arranges the available data based on inherent differences among the data points into clusters. Google Earth Engine's unsupervised classifiers are particularly advantageous in situations where ground truth data is absent, when the final number of classes is unknown, or when rapid experimentation is required. The `ee.Clusterer` package facilitates unsupervised classification (or clustering) within Earth Engine, employing algorithms that align with those found in Weka. Figure 7 provides a general overview of the unsupervised machine learning technique. Below are some key points related to this approach:

- **Input Data**: The data used in unsupervised learning is unstructured and unlabeled, meaning the model has no predefined output to learn from. For example, a dataset containing customer purchase history without any categorization.
- **When to Use**: Unsupervised learning is of extreme use when you don't have clear labels or when you're looking to explore the underlying structure of the data. It's ideal when the outcome or the nature of the desired result is unknown.
- **Nature of Problems**: The most common problems tackled using unsupervised learning include clustering (grouping similar data), dimensionality reduction (simplifying data without losing essential information), and association (finding rules that describe relationships between variables).
- **Goal**: The goal is to find hidden patterns or data groupings that can provide insights or reveal underlying structures.
- **Output**: The output is generally clusters or association rules, revealing insights such as which items tend to be purchased together or which users exhibit similar behaviors.
- **Associated Algorithms**: Some well-known unsupervised learning algorithms are K-means, which groups similar items together, and DBSCAN, which identifies clusters based on how closely packed the data points are. Another common algorithm is

Principal Component Analysis (PCA), which simplifies data by reducing its dimensions while keeping the most important information. Finally, the Apriori Algorithm is used to find patterns and associations in data.

- **Drawbacks**: A key drawback is that human intervention is often required to interpret the model's results and ensure they are meaningful. The output may also be inaccurate or difficult to validate since there are no predefined labels to measure against.
- **Expected Accuracy**: The accuracy is generally lower compared to supervised learning because there are no labelled examples to guide the learning process, and results often require more subjective validation.

2.5 LULC Classification in (GEE)

GEE offers an extensive range of tools and functions for performing (LULC) classification, supporting various methodologies like change detection, multi-year classification, and model optimization. Below is a detailed look into the functions and techniques that can be applied within GEE for LULC analysis (Table 4).

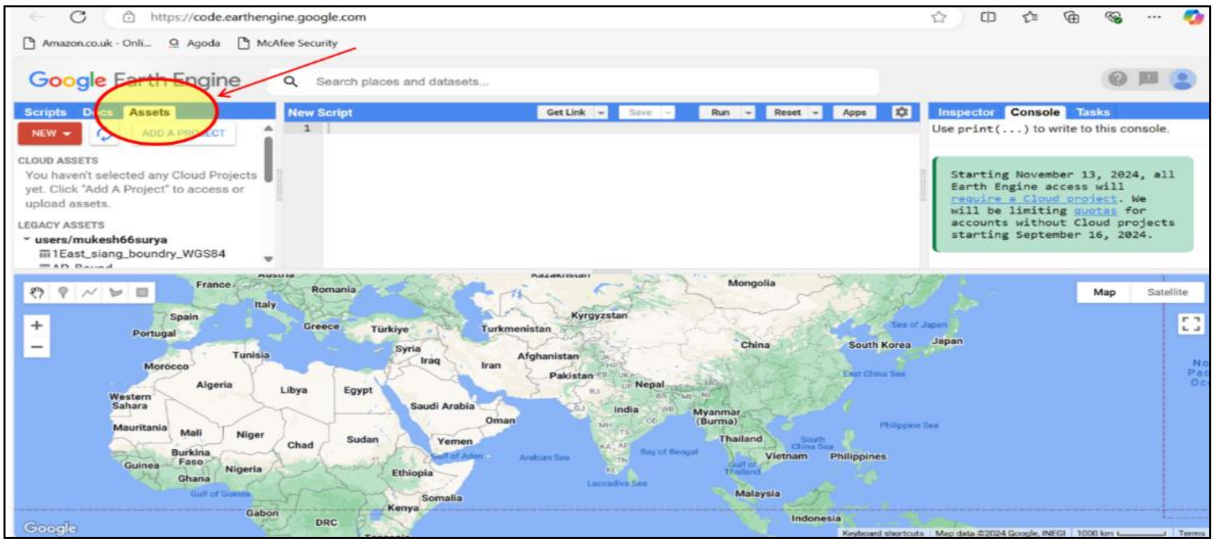
Table 4 Functions and techniques that can be applied within GEE for LULC analysis

Application in LULC	Description
LULC Change Detection	• LULC change detection identifies how land cover types change over time (Cui et al., 2022). • Helping researchers monitor environmental changes, deforestation, urbanization, and agricultural expansion.
Malti Year LULC Classification	• Multi-year LULC classification refers to the process of classifying satellite imagery over several years to analyze trends and monitor land use changes over time (Liu et al., 2020). • This helps in understanding how land cover transitions occur across different periods. • Useful in research related to climate change, ecosystem services, and land policy planning.
Class-wise LULC change detection in ONE layer	• Class-wise change detection focuses on tracking the changes in specific land cover classes within a single LULC map. • For example, it can help monitor how much forest has been converted to agricultural land or how water bodies have changed over time. • This method allows for focused analysis of land cover changes related to a specific class, which is tough for conservation, land management, and agricultural planning (Dubertret et al., 2022).
Hyperparameter Tuning for improving the accuracy of your machine-learning model	• Hyperparameter tuning is important for optimizing ML models, and improving the accuracy of classification results in GEE (Elgeldawi et al., 2021). • Hyperparameters are configuration settings for ML models that are not learned from the data but are set by the user (e.g., the number of trees in a Random Forest) (Ilemobayo et al., 2024).

Chapter 3.0: Tutorial (The process to make Landcover classification Map in GEE)

3.1 Step 1: Selecting Your ROI (Region of Interest) in Google Earth Engine (GEE)

1. **Open the GEE Platform**
   - Navigate to the GEE Code Editor by visiting: https://code.earthengine.google.com/.
2. **Go to the "Assets" Tab**
   - In the Code Editor interface, locate the **Assets** tab on the left-hand panel.



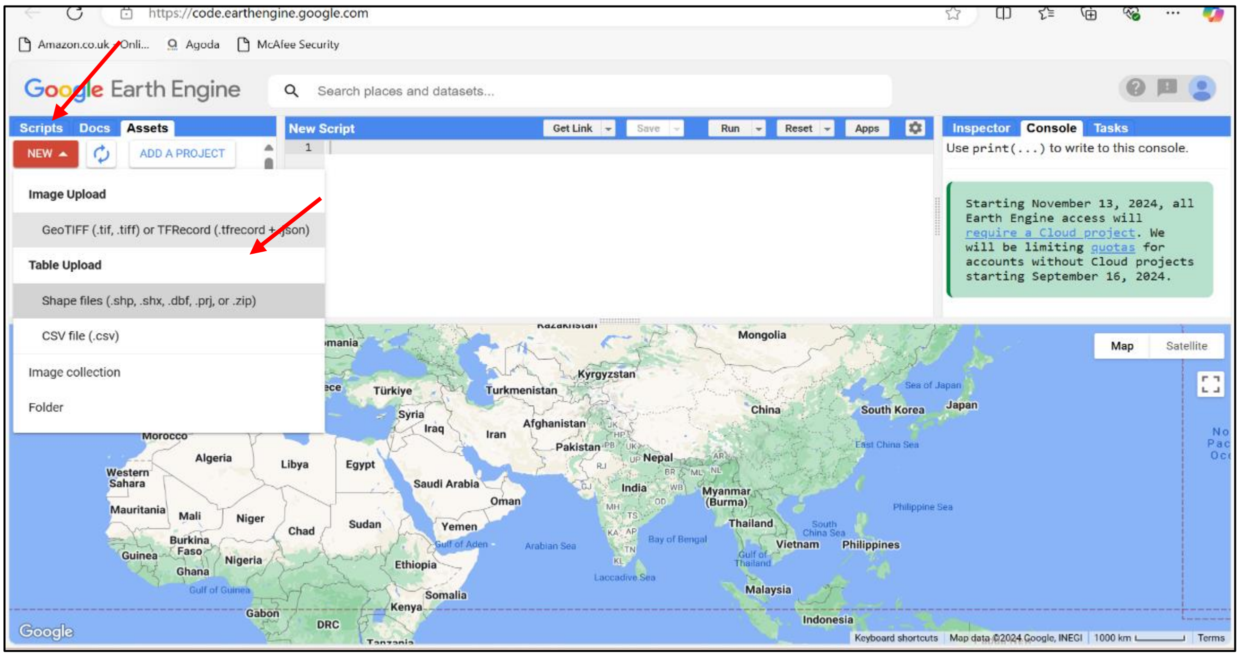
You haven't selected any Cloud Projects yet. Click "Add A Project" to access or upload assets.

Use print(...) to write to this console.

```
Starting November 13, 2024, all
Earth Engine access will
require a Cloud project. We
will be limiting quotas for
accounts without Cloud projects
starting September 16, 2024.
```

3. **Click on "NEW" to Upload a Shapefile**
   - Under the **Assets** tab, click the **NEW** button to upload your Region of Interest (ROI) as a shapefile.

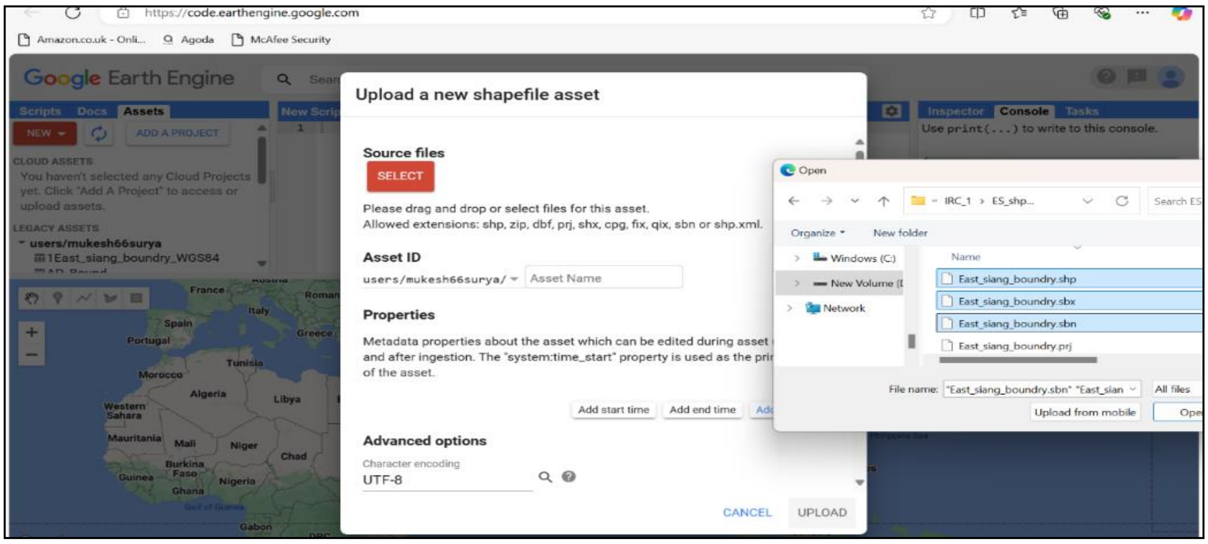
- A dropdown menu will appear, select **Shape files (.shp, .shx, .dbf, .prj, or .zip)** from the list of asset types.



Use print(...) to write to this console.

```
Starting November 13, 2024, all
Earth Engine access will
require a Cloud project. We
will be limiting quotas for
accounts without Cloud projects
starting September 16, 2024.
```

4. **Select Files from Your Folder**
   - You will be prompted to upload the shapefile. Choose the necessary files from your local system. You may either upload individual files or a compressed **.zip** folder containing the following file formats:
     - **Shapefile extensions allowed**: .shp, .dbf, .prj, .shx, .zip
     - **Additional optional extensions**: .cpg, .fix, .qix, .sbn, .shp.xml



Please drag and drop or select files for this asset.
Allowed extensions: shp, zip, dbf, prj, shx, cpg, fix, qix, sbn or shp.xml.

users/mukesh66surya/ Asset Name

Metadata properties about the asset which can be edited during asset and after ingestion. The "system:time_start" property is used as the pri of the asset.

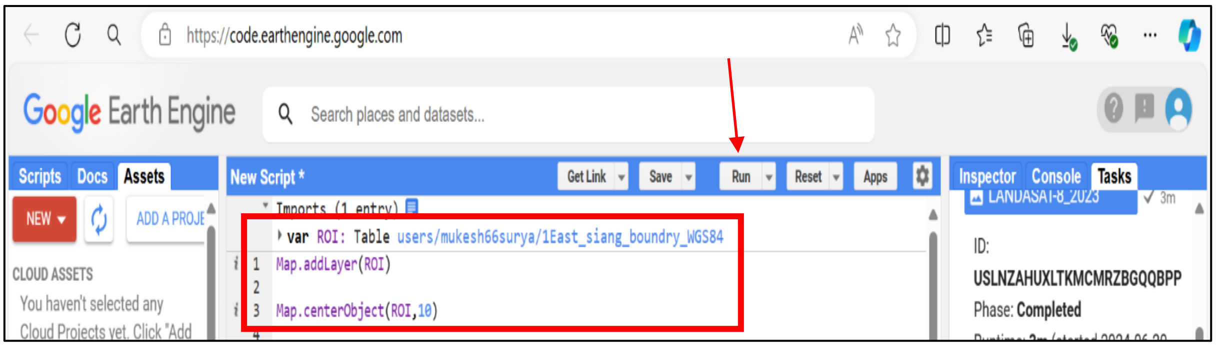
Character encoding: UTF-8

East_siang_boundry.shp
East_siang_boundry.sbx
East_siang_boundry.sbn
East_siang_boundry.prj

File name: "East_siang_boundry.sbn" "East_sian

5. **Assign an Asset Name**
   - After selecting the files, give your asset a meaningful name (e.g., "EastSiang_ROI"). This will make it easier to reference the shapefile in your analysis.

6. **Click on "UPLOAD"**
    - Once you've named your asset and selected the necessary files, click the **UPLOAD** button. Your files will be processed and added to your Earth Engine assets.
7. **Use the Uploaded Shapefile in Code**
    - After uploading, your shapefile will be available as an asset in your GEE account. You can now use it in your code by referencing it like this:



```
Imports (1 entry)
  var ROI: Table users/mukesh66surya/1East_siang_boundry_WGS84
Map.addLayer(ROI)

Map.centerObject(ROI,10)
```

ID:
USLNZAHUXLTKMCMRZBGQQBPP
Phase: Completed

This will allow you to use your custom Region of Interest (ROI) in further analysis and processing within Google Earth Engine.

8. **Centre the Map on the ROI and Set the Zoom Level**
    - **Map.addLayer(ROI)**; This line of code adds your Region of Interest (ROI) as a layer to the map.
    - **Map.centerObject(ROI, 10);** This line centres the map view on the defined ROI and sets the zoom level to 10.
    - Click **"Run".**

3.2 Step 2: Choosing and Filtering Image Collection in Google Earth Engine (GEE)

In this step, you'll select a specific image collection (e.g., Landsat 8) and apply filters based on your Region of Interest (ROI), date range, and cloud cover.

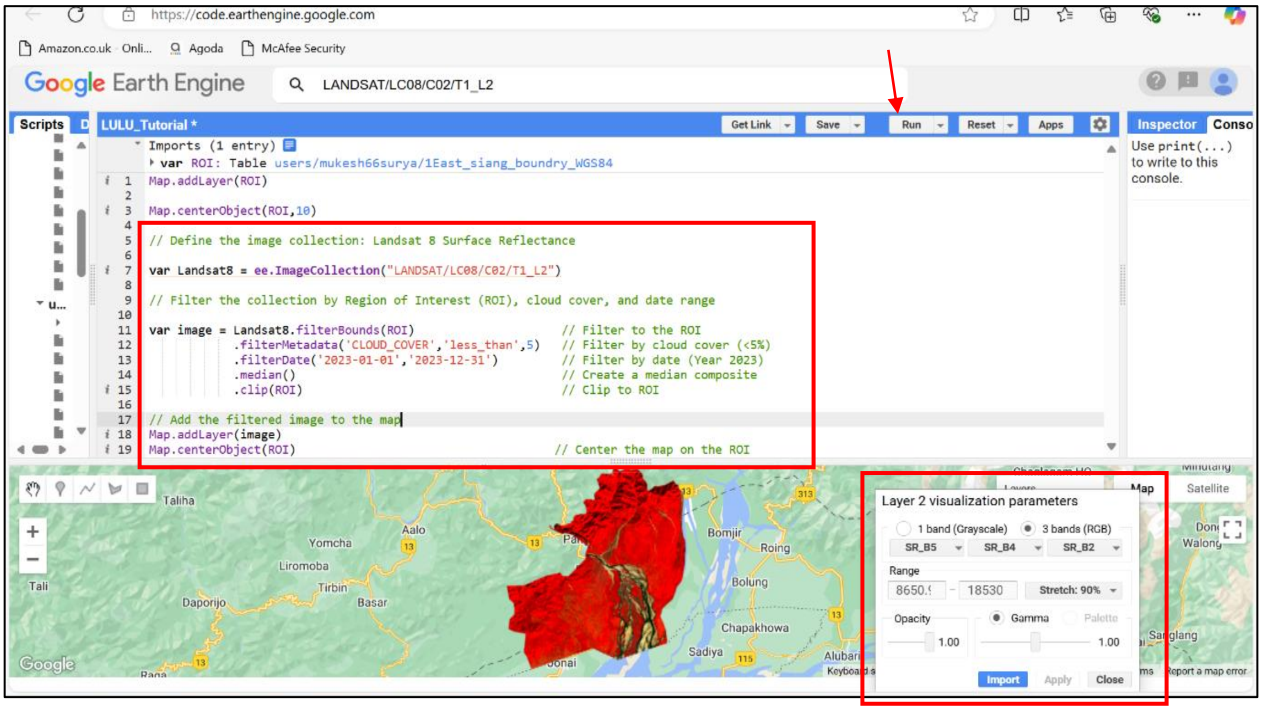
Steps to Filter Image Collection:

1. **Define the Image Collection**
    - Use the Landsat 8 image collection for the analysis. The collection is referenced as **ee.ImageCollection("LANDSAT/LC08/C02/T1_L2"),** which provides Level-2 Surface Reflectance data.
2. **Filter by Region (ROI)**
    - Apply the **filterBounds(ROI)** function to limit the images to your selected Region of Interest (ROI).
3. **Filter by Cloud Cover**
    - Use **filterMetadata('CLOUD_COVER','less_than', 5)** to exclude images with more than 5% cloud cover.
4. **Filter by Date**
    - Set a date range for the imagery by using the **filterDate('YYYY-MM-DD')** function to choose images from the year 2023.

5. **Take the Median Composite**
   - Use the **.median()** function to generate a median composite image, which reduces noise from multiple images.
6. **Clip the Image to ROI**
   - Clip the image to the boundaries of your ROI using **.clip(ROI)** to limit the data to your area of interest.
7. **Display the Image**
   - Finally, display the filtered image on the map using **Map.addLayer(image)**.
   - Click **"Run".**
   - Display the Image on the Map with Visualization Parameters-

     In this final step, you will display the filtered image on the map by selecting specific RGB bands and setting the visualization parameters. **Select RGB Bands**: For Landsat 8 imagery, the bands corresponding to Red, Green, and Blue (RGB) are: Red (SR_B4), Green (SR_B3) and Blue (SR_B2).

     Set Visualization Parameters: Define the minimum and maximum values for image stretching to adjust the contrast and brightness.



```
Imports (1 entry)
 var ROI: Table users/mukesh66surya/1East_siang_boundry_WGS84
Map.addLayer(ROI)

Map.centerObject(ROI,10)

// Define the image collection: Landsat 8 Surface Reflectance

var Landsat8 = ee.ImageCollection("LANDSAT/LC08/C02/T1_L2")

// Filter the collection by Region of Interest (ROI), cloud cover, and date range

var image = Landsat8.filterBounds(ROI)                   // Filter to the ROI
            .filterMetadata('CLOUD_COVER','less_than',5)  // Filter by cloud cover (<5%)
            .filterDate('2023-01-01','2023-12-31')        // Filter by date (Year 2023)
            .median()                                     // Create a median composite
            .clip(ROI)                                    // Clip to ROI

// Add the filtered image to the map
Map.addLayer(image)
Map.centerObject(ROI)                                    // Center the map on the ROI
```

Layer 2 visualization parameters

1 band (Grayscale) [✓ 3 bands (RGB)]

SR_B5 | SR_B4 | SR_B2

Range: 8650.! – 18530 Stretch: 90%

Opacity: 1.00

[✓ Gamma] Palette: 1.00

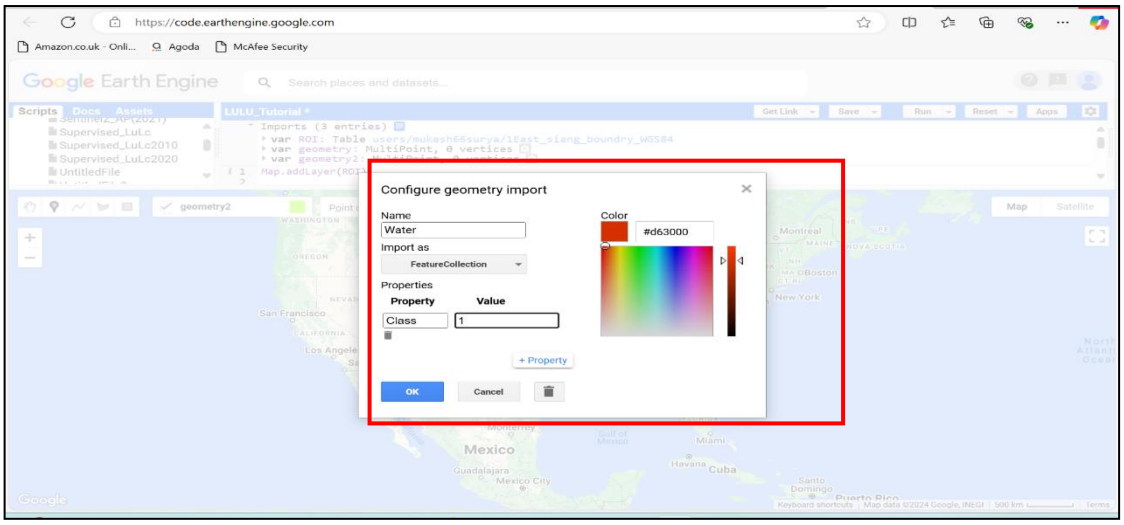
3.3 Step 3: Collecting Sample Points Using Geometry Import in GEE for Land Cover Classification

In this step, you'll collect sample points for each land cover type by manually selecting features such as points, lines, or polygons using the **Geometry Import** tool in Google Earth Engine. You will then assign a class label (e.g., 1 for water, 2 for eroded land, etc.) to each feature.

Steps to Collect Sample Points:

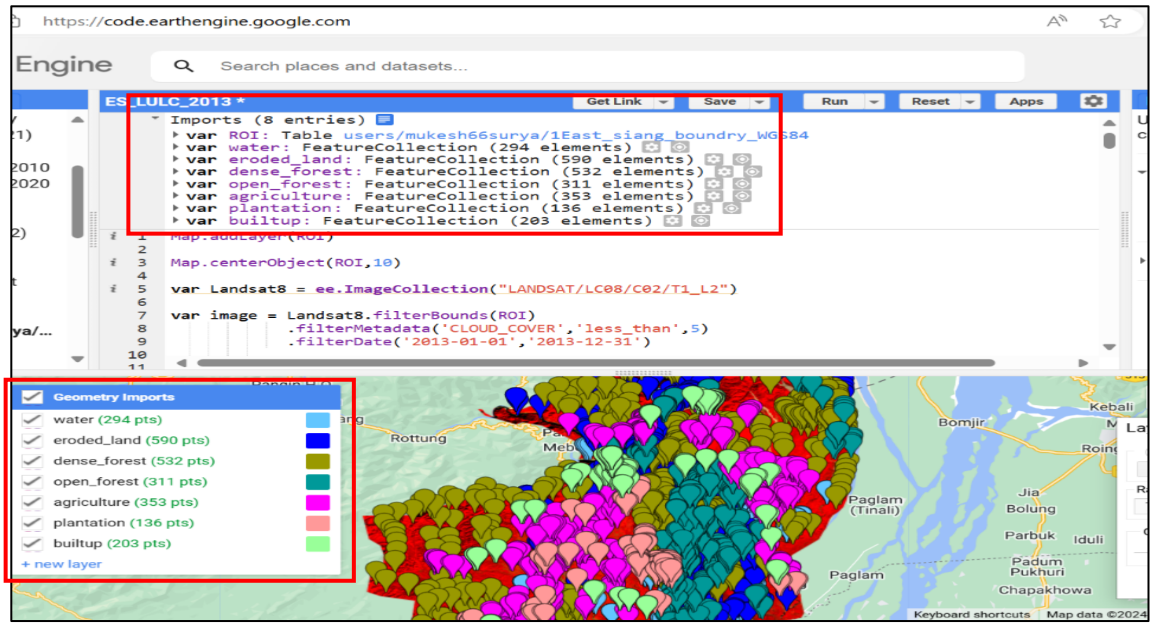
1. **Open the Geometry Tool**
    - In the left-hand panel of the GEE Code Editor, locate the **Geometry Tools** at the top left corner.
    - Click on the **Geometry Imports** dropdown to start drawing your sample points for different land cover types.
2. **Draw Features (Points, Lines, or Polygons)**
    - Select a geometry type based on your requirement:
        - **Point**: For individual sample points.
        - **Line**: To mark linear features like roads or rivers.
        - **Polygon**: To draw areas representing land cover types like forests or agricultural fields.
        - **Rectangle**: For rectangular areas.
    - Click on the map to draw your points or shapes. After each feature is drawn, it will appear in the geometry list on the left-hand panel.
3. **Configure Geometry Properties**
    - After drawing a feature, configure the **Geometry Import** settings.
        - **Name**: Give each feature a descriptive name, such as water, eroded_land, or dense_forest.
        - **Import As**: Choose **FeatureCollection** since you're collecting multiple points or areas.
        - **Property**: Set a property named Class, which will store the land cover class code.
        - **Value**: Assign a value to the property that corresponds to the land cover class. For example:
            - 1 for water/wetland
            - 2 for eroded land
            - 3 for dense forest
            - And so on for the other categories.
4. **Select the Colour for Each Class**
    - To visually distinguish between different land cover types, you can assign a specific colour to each class. You will see an option to choose a colour for the feature once you've drawn it on the map.
5. **Click "OK" to Save**

- Once the geometry is configured with the appropriate class property, click **OK** to save it as a feature collection.



Configure geometry import

Name: Water
Import as: FeatureCollection
Properties — Property: Class, Value: 1
Color: #d63000



```
ES_LULC_2013 *
Imports (8 entries)
▸ var ROI: Table users/mukesh66surya/1East_siang_boundry_WGS84
▸ var water: FeatureCollection (294 elements)
▸ var eroded_land: FeatureCollection (590 elements)
▸ var dense_forest: FeatureCollection (532 elements)
▸ var open_forest: FeatureCollection (311 elements)
▸ var agriculture: FeatureCollection (353 elements)
▸ var plantation: FeatureCollection (136 elements)
▸ var builtup: FeatureCollection (203 elements)
Map.addLayer(ROI)

Map.centerObject(ROI,10)

var Landsat8 = ee.ImageCollection("LANDSAT/LC08/C02/T1_L2")

var image = Landsat8.filterBounds(ROI)
              .filterMetadata('CLOUD_COVER','less_than',5)
              .filterDate('2013-01-01','2013-12-31')
```

Geometry Imports
- water (294 pts)
- eroded_land (590 pts)
- dense_forest (532 pts)
- open_forest (311 pts)
- agriculture (353 pts)
- plantation (136 pts)
- builtup (203 pts)
- + new layer

3.4 Step 4: Merge Feature Collections in GEE

To classify the entire region based on multiple land cover types, you need to merge these separate feature collections into one combined **FeatureCollection**. The merging process ensures that all the sample points, representing different land cover categories, are grouped into a single collection.

1. **Merging**:
   - The **.merge()** function is used to combine these individual collections into a single **FeatureCollection** called **merged_sample**. The function appends the features of one collection to another.

- The **water.merge(eroded_land)** command first combines the water and eroded land collections. Then **.merge(dense_forest)** and **.merge(open_forest)** are added in sequence.
- Click **"Run".**

2. **Result**:

- The resulting **merged_sample** contains all the features from the original feature collections, each with the assigned Class property.

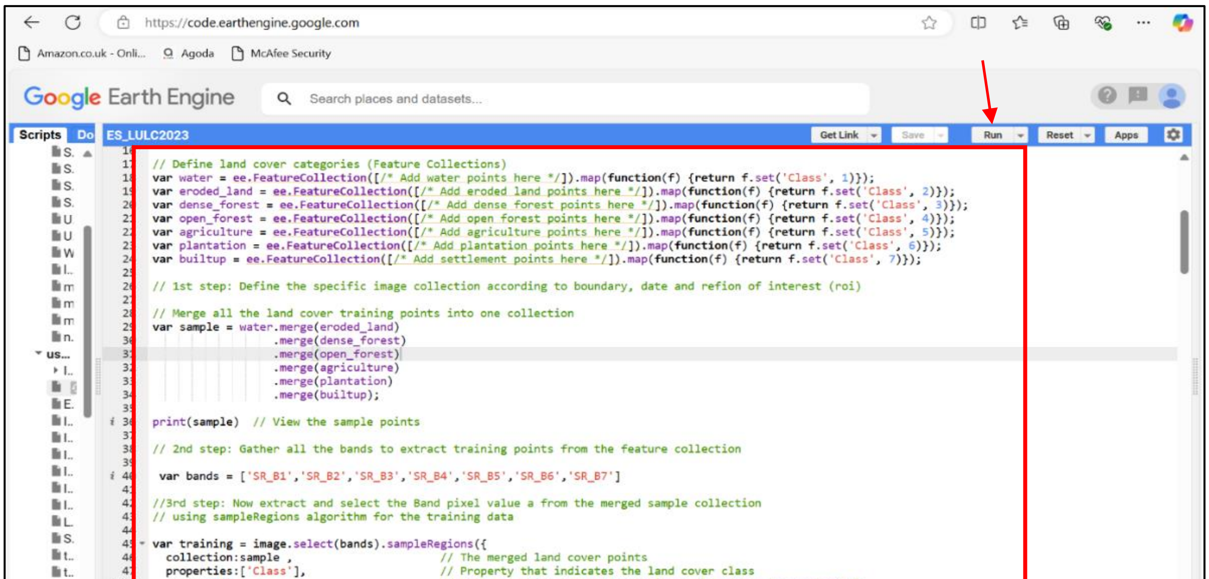
- This merged collection can now be used for further analysis, such as extracting training data or running a classification algorithm.

3.5 Step 5: Creating a Training Dataset by Sampling Regions

In this step, we will create a **training dataset** by extracting the pixel values from selected image bands for each point in the merged feature collection (representing various land cover classes). This is done using the **sampleRegions** function in GEE.

Explanation of the Steps:

1. **Selecting the Bands**:
   - We define the **bands** that we want to use for classification. In this case, we are using the seven spectral bands from Landsat 8 (SR_B1 to SR_B7), which capture information from different parts of the electromagnetic spectrum.
2. **Sampling the Image**:
   - The **sampleRegions()** function is used to extract the pixel values for each band at the locations of the sample points (in the **sample** feature collection). Each sample point represents a specific land cover type, and the pixel values at that location are extracted for the specified bands.
   - The properties argument specifies which property (in this case, **Class)** contains the land cover class for each point.
   - The scale argument specifies the resolution at which to sample the image, which is 30 meters for Landsat 8.
3. **Training Dataset**:
   - The resulting training variable is a **FeatureCollection** where each feature contains the pixel values for the selected bands as well as the corresponding land cover class. This is your training dataset, which will be used for training a machine learning classifier.



```
// Define land cover categories (Feature Collections)
var water = ee.FeatureCollection([/* Add water points here */]).map(function(f) {return f.set('Class', 1)});
var eroded_land = ee.FeatureCollection([/* Add eroded land points here */]).map(function(f) {return f.set('Class', 2)});
var dense_forest = ee.FeatureCollection([/* Add dense forest points here */]).map(function(f) {return f.set('Class', 3)});
var open_forest = ee.FeatureCollection([/* Add open forest points here */]).map(function(f) {return f.set('Class', 4)});
var agriculture = ee.FeatureCollection([/* Add agriculture points here */]).map(function(f) {return f.set('Class', 5)});
var plantation = ee.FeatureCollection([/* Add plantation points here */]).map(function(f) {return f.set('Class', 6)});
var builtup = ee.FeatureCollection([/* Add settlement points here */]).map(function(f) {return f.set('Class', 7)});

// 1st step: Define the specific image collection according to boundary, date and refion of interest (roi)

// Merge all the land cover training points into one collection
var sample = water.merge(eroded_land)
                  .merge(dense_forest)
                  .merge(open_forest)
                  .merge(agriculture)
                  .merge(plantation)
                  .merge(builtup);

print(sample)  // View the sample points

// 2nd step: Gather all the bands to extract training points from the feature collection

 var bands = ['SR_B1','SR_B2','SR_B3','SR_B4','SR_B5','SR_B6','SR_B7']

//3rd step: Now extract and select the Band pixel value a from the merged sample collection
// using sampleRegions algorithm for the training data

var training = image.select(bands).sampleRegions({
  collection:sample ,                    // The merged land cover points
  properties:['Class'],                  // Property that indicates the land cover class
```

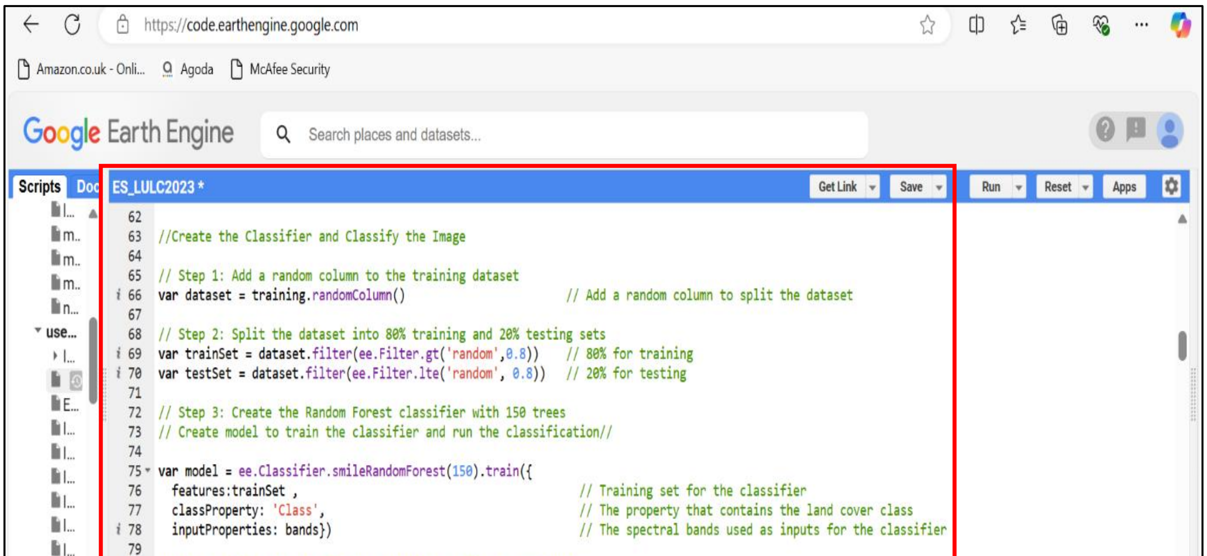
Once you have the training dataset, you can use it to train a classifier such as the **Random Forest (RF)** algorithm to classify all the images depending on the land cover classes. The training dataset contains all the necessary information to train the model for each land cover type based on their spectral characteristics.

3.6 Step 6: Randomly Splitting the Data and Create the classifier

In this step, we will create a **Random Forest classifier** and classify the image based on the training data. We'll first divide the data into training and testing. 80% of the data is used for training and 20% is used for testing and validation of the model. This ensures that the model's performance is evaluated on data it has not seen during training.

Explanation of the Steps:

1. **Randomly Splitting the Data**:
    - The **randomColumn()** function creates a column named **'random'** with values between 0 and 1, allowing us to split the data randomly.
    - The **trainSet** includes features where the value of the **'random'** column is greater than 0.2, which represents 80% of the data.
    - The **testSet** includes features where the value of the **'random'** column is less than or equal to 0.2, which represents 20% of the data.
2. **Creating the Classifier**:
    - The **ee.Classifier.smileRandomForest(150)** creates a **Random Forest classifier** with 150 trees. This is a powerful and commonly used machine learning algorithm for land cover classification.
    - The **train()** function is used to train the model with the **trainSet**, specifying the **'Class'** property as the target variable (land cover class) and the **bands** as the input features (spectral bands).
3. **Visualizing the Classified Image**:
   The classified image is displayed on the map, with each land cover class assigned a different colour using the palette argument.



```

//Create the Classifier and Classify the Image

// Step 1: Add a random column to the training dataset
var dataset = training.randomColumn()                    // Add a random column to split the dataset

// Step 2: Split the dataset into 80% training and 20% testing sets
var trainSet = dataset.filter(ee.Filter.gt('random',0.8))    // 80% for training
var testSet = dataset.filter(ee.Filter.lte('random', 0.8))   // 20% for testing

// Step 3: Create the Random Forest classifier with 150 trees
// Create model to train the classifier and run the classification//

var model = ee.Classifier.smileRandomForest(150).train({
  features:trainSet ,                                     // Training set for the classifier
  classProperty: 'Class',                                 // The property that contains the land cover class
  inputProperties: bands})                                // The spectral bands used as inputs for the classifier

```

3.7 Step 7: Hyperparameter tuning (best number of decision tree)

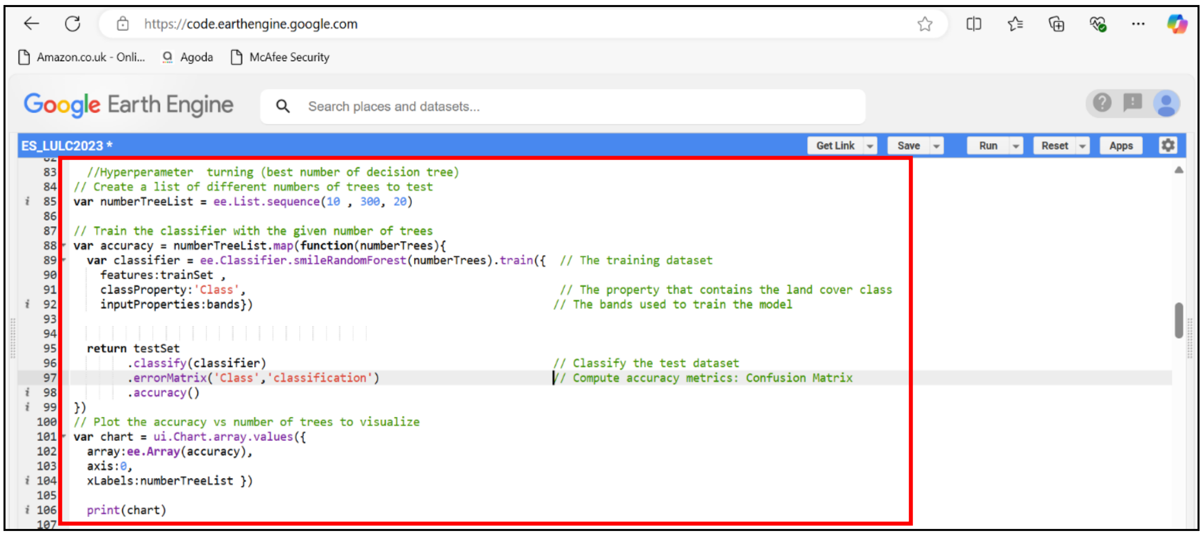
Hyperparameter tuning involves finding the best values for the parameters that are not learned from the data, such as the number of decision trees in a Random Forest classifier. The goal is to improve the model's accuracy by selecting the optimal number of trees.

In this step, we will:

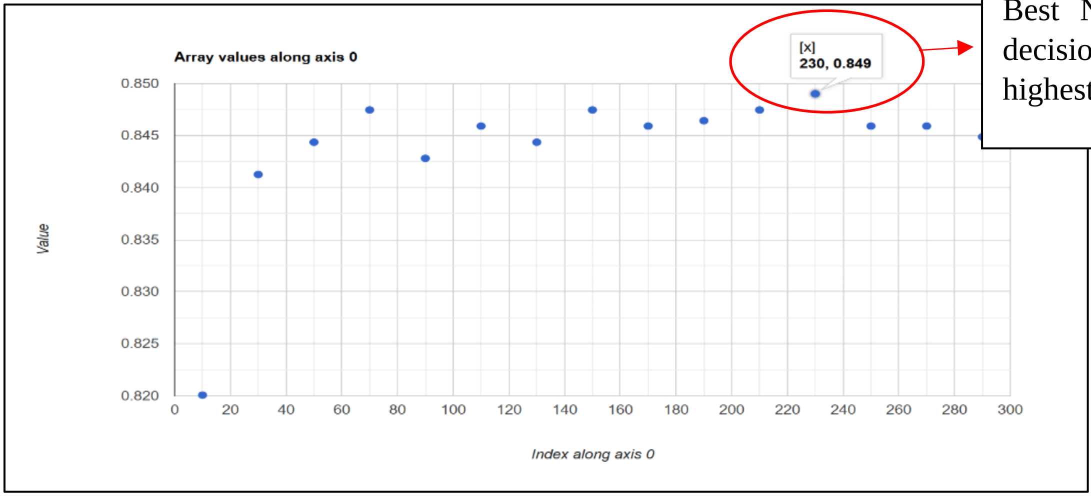
1. Test the classifier performance with different numbers of decision trees.
2. Evaluate the model’s accuracy for each value.
3. Choose the number of trees that results in the highest accuracy.

Explanation:

1. **Creating the List of Tree Numbers**:
    - We use **ee.List.sequence(10, 300, 20)** to create a sequence of values ranging from 10 to 300, incrementing by 20. These values represent the different numbers of trees we will test in the Random Forest classifier.
2. **Evaluating the Model**:
    - The function **evaluateModel** takes a number of trees as input, trains the Random Forest classifier with that number, and classifies the **test dataset**.
    - We use the **errorMatrix** function to calculate a confusion matrix and derive the **accuracy** of the classifier for that specific number of trees.
3. **Mapping Over Tree Numbers**:
    - The map function applies the **evaluateModel** function to each value in **numberTreeList**, resulting in a list of accuracies for the different numbers of trees.
4. **Visualizing Accuracy vs. Number of Trees**:
    - The accuracy results are plotted using the **ui.Chart.array.values** function, allowing you to visualize how the accuracy changes with the number of trees.
    - You can then choose the number of trees that maximizes the accuracy.



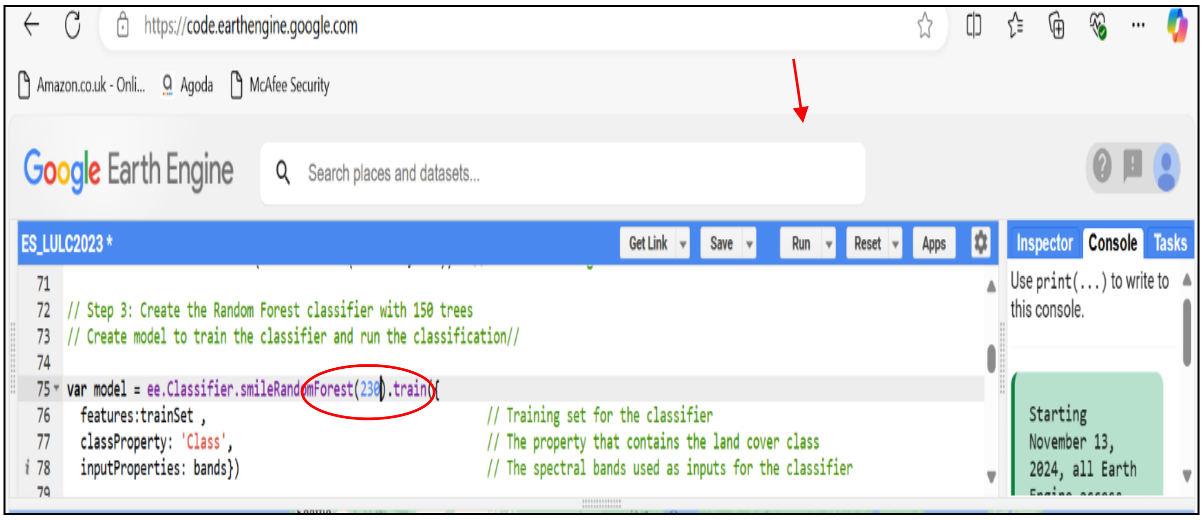
```
  //Hyperperameter  turning (best number of decision tree)
// Create a list of different numbers of trees to test
var numberTreeList = ee.List.sequence(10 , 300, 20)

// Train the classifier with the given number of trees
var accuracy = numberTreeList.map(function(numberTrees){
  var classifier = ee.Classifier.smileRandomForest(numberTrees).train({  // The training dataset
    features:trainSet ,
    classProperty:'Class',                                                // The property that contains the land cover class
    inputProperties:bands})                                              // The bands used to train the model


  return testSet
        .classify(classifier)                                            // Classify the test dataset
        .errorMatrix('Class','classification')                           // Compute accuracy metrics: Confusion Matrix
        .accuracy()
})
// Plot the accuracy vs number of trees to visualize
var chart = ui.Chart.array.values({
  array:ee.Array(accuracy),
  axis:0,
  xLabels:numberTreeList })

  print(chart)
```



Best Number of decision tree with highest accuracy

Final Step:

Once you've identified the number of trees that results in the highest accuracy, you can update the classifier in your main model and **Run**.



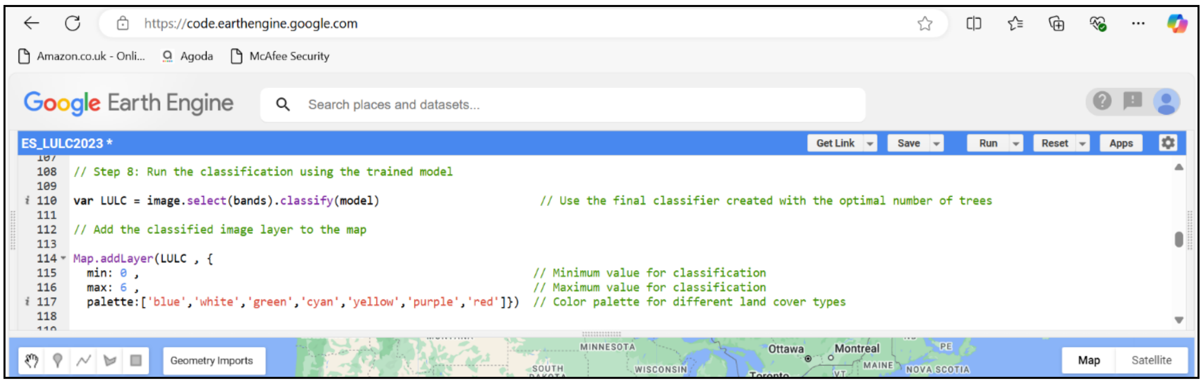
```
// Step 3: Create the Random Forest classifier with 150 trees
// Create model to train the classifier and run the classification//

var model = ee.Classifier.smileRandomForest(230).train({
  features:trainSet ,                           // Training set for the classifier
  classProperty: 'Class',                       // The property that contains the land cover class
  inputProperties: bands})                      // The spectral bands used as inputs for the classifier
```

Use print(...) to write to this console.

```
Starting
November 13,
2024, all Earth
```

3.8 Step 8: Run the Classification and Visualize the Results

In this step, we will classify the image using the trained Random Forest model and visualize the results on the map.



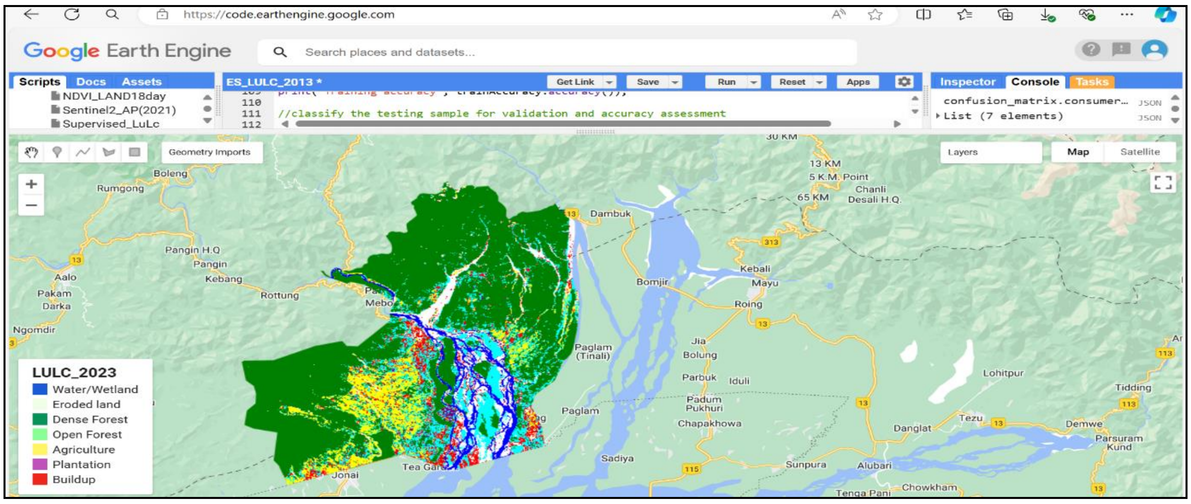
```
// Step 8: Run the classification using the trained model

var LULC = image.select(bands).classify(model)                 // Use the final classifier created with the optimal number of trees

// Add the classified image layer to the map

Map.addLayer(LULC , {
  min: 0 ,                                                      // Minimum value for classification
  max: 6 ,                                                      // Maximum value for classification
  palette:['blue','white','green','cyan','yellow','purple','red']})  // Color palette for different land cover types

```

Explanation:

1. **Running the Classification**:
    - The classification is performed using the **classify** method on the selected bands of the image.
    - The **model** is used here, which was trained with the optimal number of trees obtained from hyperparameter tuning.
2. **Visualizing the Results**:
    - The classified (LULC) image is added to the map with specific visualization parameters.
    - Min and max values define the range of classification values to be visualized.
    - The palette defines the colours assigned to different land cover classes, where:
        - 0 could represent water (blue),
        - 1 could represent eroded land (white),
        - 2 could represent dense forest (green),
        - 3 could represent open forest (cyan),
        - 4 could represent agriculture (yellow),
        - 5 could represent plantation (purple),
        - 6 could represent built-up areas (red).

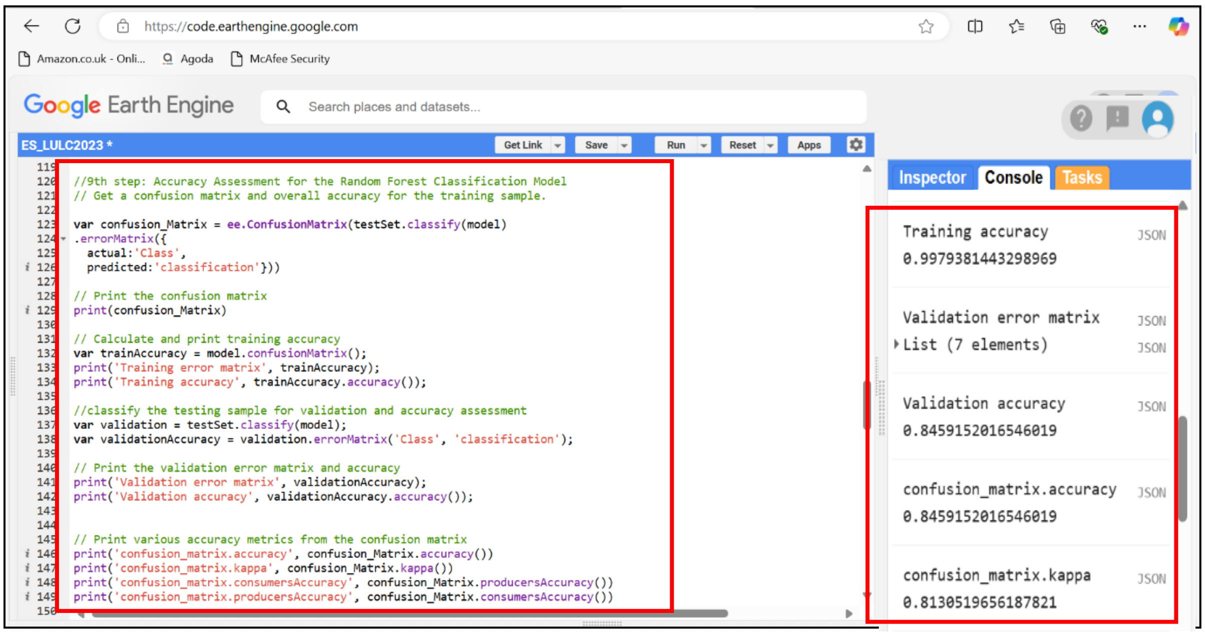
Final Visualization

Once you run this code, you should see the classified land use/land cover map overlaid on your selected region of interest in Google Earth Engine. This visualization will help you know about the spatial distribution of land cover types for the chosen study area.



3.9 Step 9: Print confusion matrix for accuracy assessment

In this step, we will assess the efficiency of the Random Forest by estimating the confusion matrix for both the training and testing data. This assessment helps us understand how well the model performs in classifying land cover types.



```
//9th step: Accuracy Assessment for the Random Forest Classification Model
// Get a confusion matrix and overall accuracy for the training sample.

var confusion_Matrix = ee.ConfusionMatrix(testSet.classify(model)
.errorMatrix({
  actual:'Class',
  predicted:'classification'}))

// Print the confusion matrix
print(confusion_Matrix)

// Calculate and print training accuracy
var trainAccuracy = model.confusionMatrix();
print('Training error matrix', trainAccuracy);
print('Training accuracy', trainAccuracy.accuracy());

//classify the testing sample for validation and accuracy assessment
var validation = testSet.classify(model);
var validationAccuracy = validation.errorMatrix('Class', 'classification');

// Print the validation error matrix and accuracy
print('Validation error matrix', validationAccuracy);
print('Validation accuracy', validationAccuracy.accuracy());


// Print various accuracy metrics from the confusion matrix
print('confusion_matrix.accuracy', confusion_Matrix.accuracy())
print('confusion_matrix.kappa', confusion_Matrix.kappa())
print('confusion_matrix.consumersAccuracy', confusion_Matrix.producersAccuracy())
print('confusion_matrix.producersAccuracy', confusion_Matrix.consumersAccuracy())
```

```
Training accuracy
0.9979381443298969

Validation error matrix
▸List (7 elements)

Validation accuracy
0.8459152016546019

confusion_matrix.accuracy
0.8459152016546019

confusion_matrix.kappa
0.8130519656187821
```

Explanation of Each Step:

1. **Confusion Matrix for the Test Dataset**:
    - The confusion matrix is created using **errorMatrix()** by comparing the actual land cover classes (**actual: 'Class'**) to the predicted classes (**predicted: 'classification'**).
    - The confusion matrix provides insight into the classification performance, showing how many instances were correctly or incorrectly classified for each class.
2. **Printing the Confusion Matrix**:

- The confusion matrix is printed to the **console** for examination, allowing you to see the true positives, false positives, true negatives, and false negatives for each class.

3. **Training Accuracy Calculation**:
   - The confusion matrix for the training dataset is obtained using **model.confusionMatrix().**
   - The training accuracy is calculated and printed using **trainAccuracy.accuracy().** This gives you an idea of how well the model fits the training data.
4. **Classifying the Testing Sample**:
   - The test dataset is classified again to validate the model's performance on unseen data. This is essential to evaluate the model's generalizability.
5. **Validation Accuracy Assessment**:
   - A confusion matrix is generated for the validation data using **validation.errorMatrix().**
   - The validation accuracy is printed, which indicates how well the model performs on the testing dataset.
6. **Additional Accuracy Metrics**:
   - **Overall Accuracy**: This metric reflects the percentage of correct predictions from the confusion matrix **(confusionMatrix.accuracy()).**
   - **Kappa Coefficient**: This metric assesses the accuracy between predicted and the actual classifications, adjusting for the agreement that could occur by chance **(confusionMatrix.kappa()).**
   - **Producers Accuracy**: This indicates the quantity of actual positive instances correctly identified for each class. It answers the question: "Of all the actual classes, how many did we predict correctly?" **(confusionMatrix.producersAccuracy()).**
   - **Consumers Accuracy**: This indicates the proportion of predicted positive instances that were correctly classified. It answers the question: "Of all the predicted classes, how many were actually correct?" **(confusionMatrix.consumersAccuracy()).**

3.10 Step 10: Calculating Area for Each Class Using the Chart Method

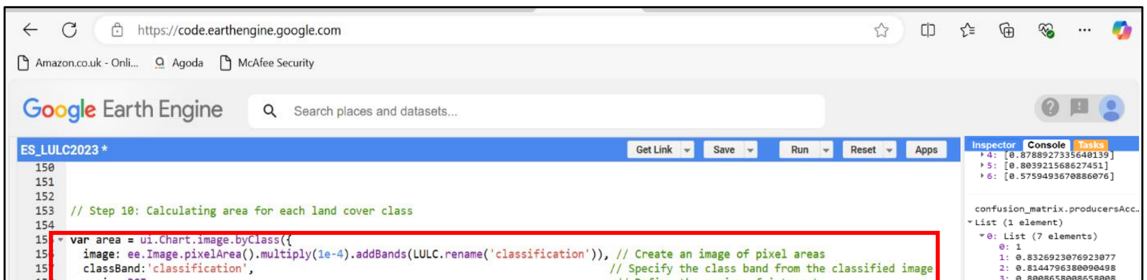
In this step, we will estimate the area of each land cover class using the **ui.Chart.image.byClass()** method in Google Earth Engine (GEE). This method allows us to visualize the area distribution of different land cover classes within the region of interest (ROI).

Explanation of Each Step:

1. **Creating an Image of Pixel Areas**:
   - **ee.Image.pixelArea()**: This function generates an image where each pixel's value represents the area of that pixel in square meters.

- **.multiply(1e-4)**: The pixel area is multiplied by 1×10^{-4} to convert square meters to hectares, since 1 ha=10,000 m^2.
- **.addBands(LULC.rename('classification'))**: The classified (LULC) image is added as a new band named 'classification'. This allows us to link the pixel area values with their corresponding land cover classes.

2. **Setting Class Band**:
   - **classBand: 'classification'**: This specifies which band in the image contains the class labels for each pixel. In this case, it's the band we added earlier, which contains the classified values for each land cover type.
3. **Defining the Region of Interest**:
   - **region: ROI**: The area over which the calculation will be performed is defined by the variable ROI, which represents the region of interest specified earlier in the workflow.
4. **Using the Sum Reducer**:
   - **reducer: ee.Reducer.sum()**: The sum reducer is applied to estimate the total area for each land cover class. This will sum the pixel area values corresponding to each class within the ROI.
5. **Setting the Scale**:
   - **scale: 30**: This parameter sets the scale at which to perform the calculations. For Landsat 8 imagery, a scale of 30 meters is appropriate, as it matches the pixel resolution of the data.
6. **Defining Class Labels**:
   - **classLabels**: A list of strings representing the names of the land cover classes. This helps in labelling the chart axes and interpreting the results.
7. **Configuring Chart Options**:
   - **setOptions({ ... })**: This method allows customization of the chart's appearance. The title of the chart, axis titles, and colour scheme are set to make the chart more informative and visually appealing.
8. **Printing the Area Chart**:
   - Finally, **print(area)**: This command outputs the area chart to the console in GEE, allowing you to visualize the area distribution of each land cover class.



```
// Step 10: Calculating area for each land cover class

var area = ui.Chart.image.byClass({
  image: ee.Image.pixelArea().multiply(1e-4).addBands(LULC.rename('classification')), // Create an image of pixel areas
  classBand:'classification',                                              // Specify the class band from the classified image
```

```
 ▸4: [0.8788927335640139]
 ▸5: [0.803921568627451]
 ▸6: [0.5759493670886076]

confusion_matrix.producersAcc…
▾List (1 element)
 ▾0: List (7 elements)
    0: 1
    1: 0.8326923076923077
    2: 0.8144796380090498
```

3.11 Step 11: Export Image and Create Legend in Google Earth Engine

Part 1: Exporting the Classified Image

The **Export.image.toDrive()** function is used to export the classified image (LULC) to Google Drive for downloading and further use.

Explanation:

1. **image: LULC**: Specifies the classified image that you want to export.
2. **description: 'LULC_2023_ES'**: The name of the file when it is exported to Google Drive.
3. **folder: 'LULC_folder'**: The folder in Google Drive where the image will be saved. You can change the folder name or leave it blank to export directly to the main drive.
4. **region: ROI:** The region of interest is defined earlier and specifies the boundary for the export.
5. **scale: 30**: The spatial resolution for export, set to 30 meters for Landsat 8 data.
6. **maxPixels: 1e13**: Sets the maximum number of pixels that can be exported (a large number to accommodate a large dataset).

Part 2: Creating a Legend for the Map

You can add a legend to your map to visually represent the LULC classes with colors.

1. **Creating the Legend Panel**:
   position: 'bottom-left': Positions the legend panel at the bottom-left of the map.
   padding: '8px 15px': Adds some spacing around the content inside the legend panel.

2**. Creating the Legend Title:**

The title of the legend, **"LULC_2023",** is created with styling to make it bold and larger.

3. **Adding the Title to the Legend Panel**:

4. **Creating Rows for Each Class in the Legend**:

   - **makeRow()**: A function that creates a row with a colored box and a description (land cover type name).
   - **colorBox**: A label representing the color of each land cover type.
   - **description**: A label representing the name of the land cover type.

- **ui.Panel.Layout.Flow('horizontal')**: Ensures that the color box and label are placed next to each other in a row.

5. **Defining the Colours and Labels for the Legend**:

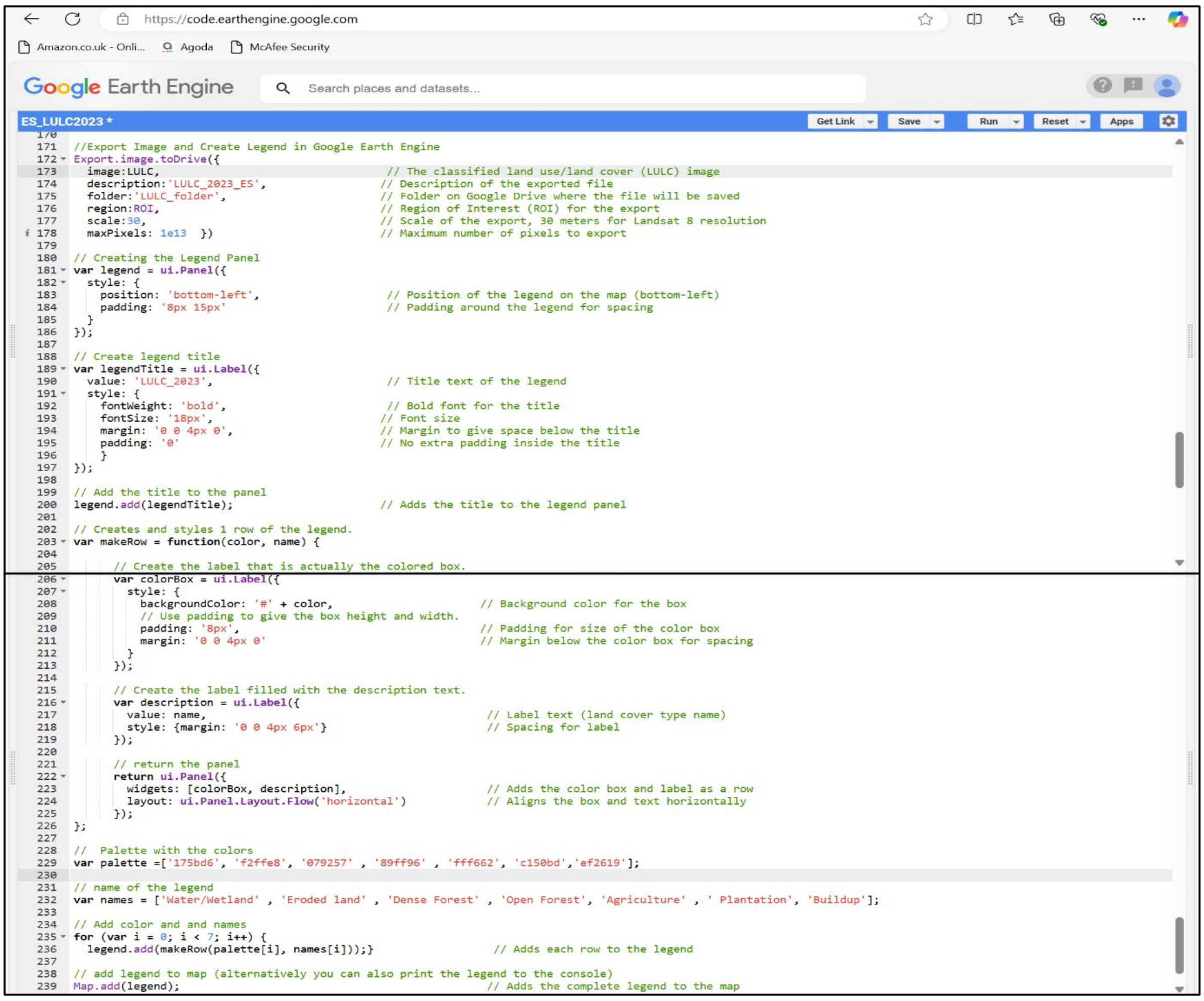
- **palette**: Defines the colours for each land cover class.
- **names**: Defines the names of each land cover class.

6. **Adding Rows for Each Land Cover Class:**

A loop iterates through each class, adding a row (colour + name) for each land cover category.

7. **Adding the Legend to the Map**:

This line adds the legend panel to the Google Earth Engine map interface.



```
//Export Image and Create Legend in Google Earth Engine
Export.image.toDrive({
  image:LULC,                                   // The classified land use/land cover (LULC) image
  description:'LULC_2023_ES',                  // Description of the exported file
  folder:'LULC_folder',                        // Folder on Google Drive where the file will be saved
  region:ROI,                                  // Region of Interest (ROI) for the export
  scale:30,                                    // Scale of the export, 30 meters for Landsat 8 resolution
  maxPixels: 1e13  })                          // Maximum number of pixels to export

// Creating the Legend Panel
var legend = ui.Panel({
  style: {
    position: 'bottom-left',                    // Position of the legend on the map (bottom-left)
    padding: '8px 15px'                         // Padding around the legend for spacing
  }
});

// Create legend title
var legendTitle = ui.Label({
  value: 'LULC_2023',                           // Title text of the legend
  style: {
    fontWeight: 'bold',                         // Bold font for the title
    fontSize: '18px',                          // Font size
    margin: '0 0 4px 0',                       // Margin to give space below the title
    padding: '0'                               // No extra padding inside the title
    }
});

// Add the title to the panel
legend.add(legendTitle);                       // Adds the title to the legend panel

// Creates and styles 1 row of the legend.
var makeRow = function(color, name) {

      // Create the label that is actually the colored box.
      var colorBox = ui.Label({
        style: {
          backgroundColor: '#' + color,                      // Background color for the box
          // Use padding to give the box height and width.
          padding: '8px',                                    // Padding for size of the color box
          margin: '0 0 4px 0'                                // Margin below the color box for spacing
        }
      });

      // Create the label filled with the description text.
      var description = ui.Label({
        value: name,                                          // Label text (land cover type name)
        style: {margin: '0 0 4px 6px'}                        // Spacing for label
      });

      // return the panel
      return ui.Panel({
        widgets: [colorBox, description],                     // Adds the color box and label as a row
        layout: ui.Panel.Layout.Flow('horizontal')            // Aligns the box and text horizontally
      });
};

//  Palette with the colors
var palette =['175bd6', 'f2ffe8', '079257' , '89ff96' , 'fff662', 'c150bd','ef2619'];

// name of the legend
var names = ['Water/Wetland' , 'Eroded land' , 'Dense Forest' , 'Open Forest', 'Agriculture' , ' Plantation', 'Buildup'];

// Add color and and names
for (var i = 0; i < 7; i++) {
  legend.add(makeRow(palette[i], names[i]));}                 // Adds each row to the legend

// add legend to map (alternatively you can also print the legend to the console)
Map.add(legend);                                             // Adds the complete legend to the map
```

Part 3: To export an image from GEE to Google Drive, follow these steps:

1. Go to the "Tasks" Tab:

- In the top-right corner of the GEE code editor, you'll see a tab labelled "Tasks" next to the Console, Inspector, and Layers tabs.

2. Run the Unsubmitted Task:

- Once you've defined your **Export.image.toDrive()** function in the script, it creates a task.
- Go to the "Tasks" tab, where you will see your unsubmitted export task.
- Click on "**Run**" next to the task.

3. Fill in the Export Details:

A dialog box will appear. Here you can fill in the necessary export details:

- **Task Name**: Enter a descriptive name for the task (e.g., **LULC_Export**).
- **Coordinate Reference System (CRS)**: This refers to the projection system in which you want the exported image. For example, you can use EPSG:4326 for geographic coordinates or EPSG:32643 for UTM Zone 43N (customize according to your project needs).
- **Scale**: This is the spatial resolution of the exported image. For Landsat 8, a common scale is **30 meters**.
- **Drive Folder Name**: Select or create a folder in your Google Drive where you want the image to be saved (e.g., **LULC_folder**).
- **Filename**: Choose a name for the exported image file (e.g., **LULC_2023_ES**).
- **File Format**: Choose the file format for the export, typically **GeoTIFF (.tif).**

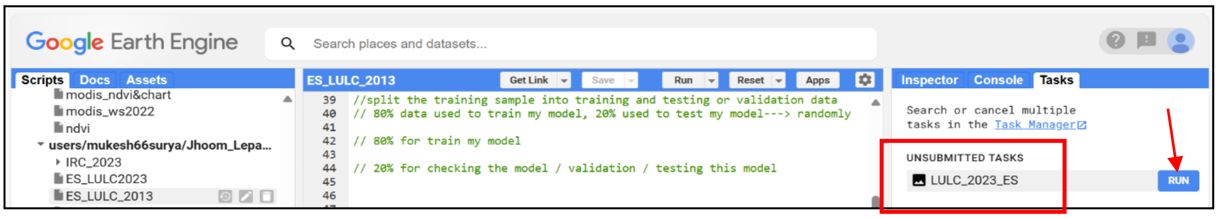
4. Click "Run":

- After entering all the required information, click the "**Run**" button.

5. Download the Image:

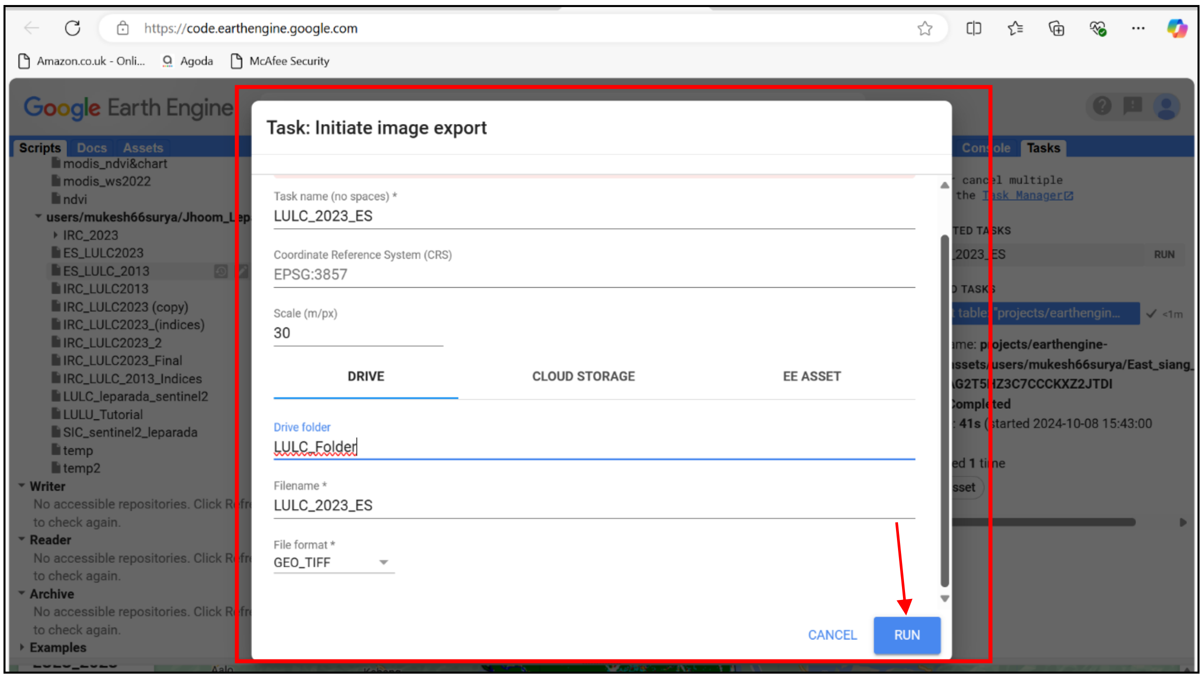
- The export process will start, and it may take some time that depends on the image size.
- Once the export is complete, the file will be available in your Google Drive under the folder name you specified.
- You can now download the image from your Drive folder to your local machine.

By following these steps, you will initiate the export process and can later download the classified LULC image from your Google Drive.



```
//split the training sample into training and testing or validation data
// 80% data used to train my model, 20% used to test my model---> randomly

// 80% for train my model

// 20% for checking the model / validation / testing this model
```

Search or cancel multiple tasks in the Task Manager



Task: Initiate image export

Task name (no spaces) *
LULC_2023_ES

Coordinate Reference System (CRS)
EPSG:3857

Scale (m/px)
30

DRIVE | CLOUD STORAGE | EE ASSET

Drive folder
LULC_Folder

Filename *
LULC_2023_ES

File format *
GEO_TIFF

CANCEL RUN

References:

Alnuaimi, A. F., & Albaldawi, T. H. (2024). An overview of machine learning classification techniques. *BIO Web of Conferences*, *97*, 00133. https://doi.org/10.1051/bioconf/20249700133

Amani, M., Ghorbanian, A., Ahmadi, S. A., Kakooei, M., Moghimi, A., Mirmazloumi, S. M., Moghaddam, S. H. A., Mahdavi, S., Ghahremanloo, M., Parsian, S., Wu, Q., & Brisco, B. (2020). Google Earth Engine Cloud Computing Platform for Remote Sensing Big Data Applications: A Comprehensive Review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *13*, 5326–5350. https://doi.org/10.1109/jstars.2020.3021052

Cui, J., Zhu, M., Liang, Y., Qin, G., Li, J., & Liu, Y. (2022). Land Use/Land Cover Change and Their Driving Factors in the Yellow River Basin of Shandong Province Based on Google Earth Engine from 2000 to 2020. *ISPRS International Journal of Geo-Information*, *11*(3), 163. https://doi.org/10.3390/ijgi11030163

Dubertret, F., Tourneau, F. L., Villarreal, M. L., & Norman, L. M. (2022). Monitoring Annual Land Use/Land Cover Change in the Tucson Metropolitan Area with Google Earth Engine (1986–2020). *Remote Sensing*, *14*(9), 2127. https://doi.org/10.3390/rs14092127

Elgeldawi, E., Sayed, A., Galal, A. R., & Zaki, A. M. (2021). Hyperparameter Tuning for Machine Learning Algorithms Used for Arabic Sentiment Analysis. *Informatics*, *8*(4), 79. https://doi.org/10.3390/informatics8040079

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, *202*, 18–27. https://doi.org/10.1016/j.rse.2017.06.031

Huang, X., Jensen, J.R., 1997. A machine-learning approach to automated knowledge base building for remote sensing image analysis with GIS data. Photogramm. Eng. Remote Sens. 63, 1185–1193.

Ilemobayo, J. A., Durodola, O., Alade, O., Awotunde, O. J., Olanrewaju, A. T., Falana, O., Ogungbire, A., Osinuga, A., Ogunbiyi, D., Ifeanyi, A., Odezuligbo, I. E., & Edu, O. E. (2024). Hyperparameter Tuning in Machine Learning: A Comprehensive Review. *Journal of Engineering Research and Reports*, *26*(6), 388–395. https://doi.org/10.9734/jerr/2024/v26i61188

Liu, C., Li, W., Zhu, G., Zhou, H., Yan, H., & Xue, P. (2020). Land Use/Land Cover Changes and Their Driving Factors in the Northeastern Tibetan Plateau Based on Geographical Detectors and Google Earth Engine: A Case Study in Gannan Prefecture. Remote Sensing, 12(19), 3139. https://doi.org/10.3390/rs12193139

Schulz, K., Hänsch, R., Sörgel, U., 2018. Machine learning methods for remote sensing applications: an overview. In: Earth Resources and Environmental Remote Sensing/ GIS Applications IX. International Society for Optics and Photonics, pp. 1079002.

Singh, V., Nema, A. K., Chouksey, A., & Suryawanshi, A. (2024). Assessment of Eco-Environmental Vulnerability Using Remote Sensing and GIS Tools in Maharashtra Region, India. *International Journal of Environment and Climate Change*, 14(4), 119–129. https://doi.org/10.9734/ijecc/2024/v14i44103

Suryawanshi, A., Dubey, S. & Sharma, M. (2023). Evaluating Soil Erosion through Geospatial Techniques: Difficulties and Prospects in the Context of the Central Indian Chambal River Basin. *International Journal of Environment and Climate Change*, 13(11): 4518-4533. http://dx.doi.org/10.9734/ijecc/2023/v13i113632

Suryawanshi, A., Nema, A. K., Jaiswal, R. K., Jain, S., & Kar, S. K. (2021). Identification of soil erosion prone areas of Madhya Pradesh using USLE/RUSLE. *Journal of Agricultural Engineering*, 58(2): 177-191. https://doi.org/10.9734/ijecc/2024/v14i44103

Tamiminia, H., Salehi, B., Mahdianpari, M., Quackenbush, L., Adeli, S., & Brisco, B. (2020). Google Earth Engine for geo-big data applications: A meta-analysis and systematic review. *ISPRS Journal of Photogrammetry and Remote Sensing*, *164*, 152–170. https://doi.org/10.1016/j.isprsjprs.2020.04.001

Velastegui-Montoya, A., Montalván-Burbano, N., Carrión-Mero, P., Rivera-Torres, H., Sadeck, L., & Adami, M. (2023). Google Earth Engine: A Global Analysis and Future Trends. *Remote Sensing*, *15*(14), 3675. https://doi.org/10.3390/rs15143675

Waleed, M., & Sajjad, M. (2023). On the emergence of geospatial cloud-based platforms for disaster risk management: A global scientometric review of google earth engine applications. *International Journal of Disaster Risk Reduction*, *97*, 104056. https://doi.org/10.1016/j.ijdrr.2023.104056

Wangchu, L., Angami, T., Jini, D., Bam, J., Singh, R., Tasung, A. & Suryawanshi, A. (2024). Natural Farming: Scope and Prospective in Arunachal Pradesh. *ICAR (Research Complex) for NEH Region, Umiam-793103, Meghalaya*