1. Introduction

Google Earth Engine (GEE) combines a multi-petabyte catalog of satellite imagery and geospatial datasets with planetary-scale analysis capabilities and makes it available for scientists, researchers, and developers to detect changes, map trends, and quantify differences on the Earth's surface.

2. For GEE Tutorials

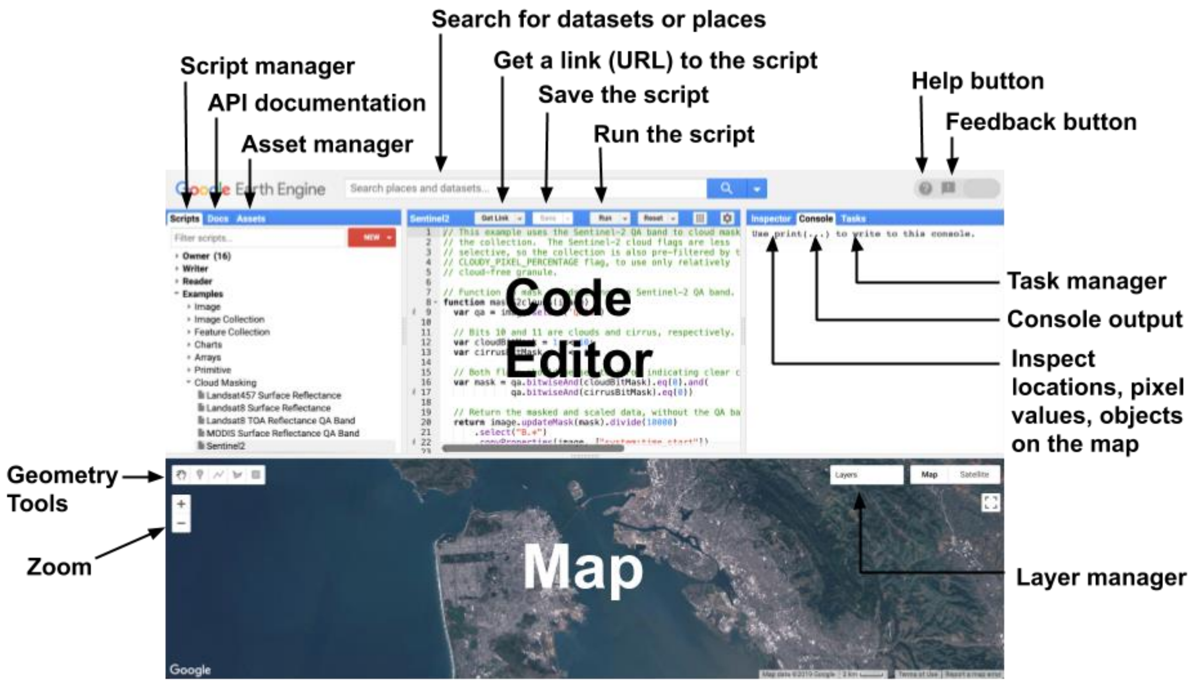
https://developers.google.com/earth-engine/tutorials/tutorials

3. Setting up the Environment

Before getting started with earth engine you must have an account

- **Step one** is to create a Gmail account and sign in the account to move further (Ignore this step if already had)
- **Step two**: https://signup.earthengine.google.com/ and sign-up with your Google account. (It will take 1-2 days for approval.)
- **Step Three**: Login to google earth engine - https://code.earthengine.google.com/

4. Earth Engine Code Editor - Components



5. Introduction to JavaScript

What is JavaScript and how it works?

- JavaScript is a lightweight scripting language mostly used in web development but there are also other application which are using for development purpose like earth engine and many more.
- JavaScript is an object oriented language – It uses objects and classes as a reusable pieces of code which makes it easy for the programmer and saves lot of time in rewriting of the same code again and again.
- You can use any available IDE for JavaScript to write the code but for now we will be using earth engine to get our work done. let's get started with writing some basics
- JavaScript is a **case sensitive language.**

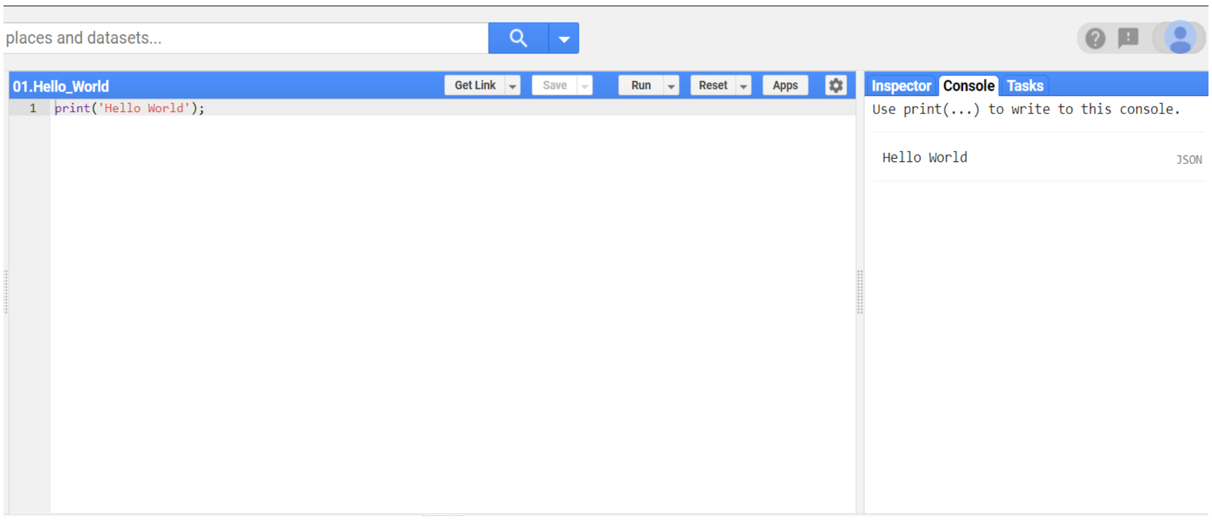
6. JavaScript Basics

Let's start with WELLKNOWN code

//type the below code in code editor

print ('Hello World');

// Gives output as follows



```
print('Hello World');
```

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

Hello World
```

- **Variables declaration**

// Variables

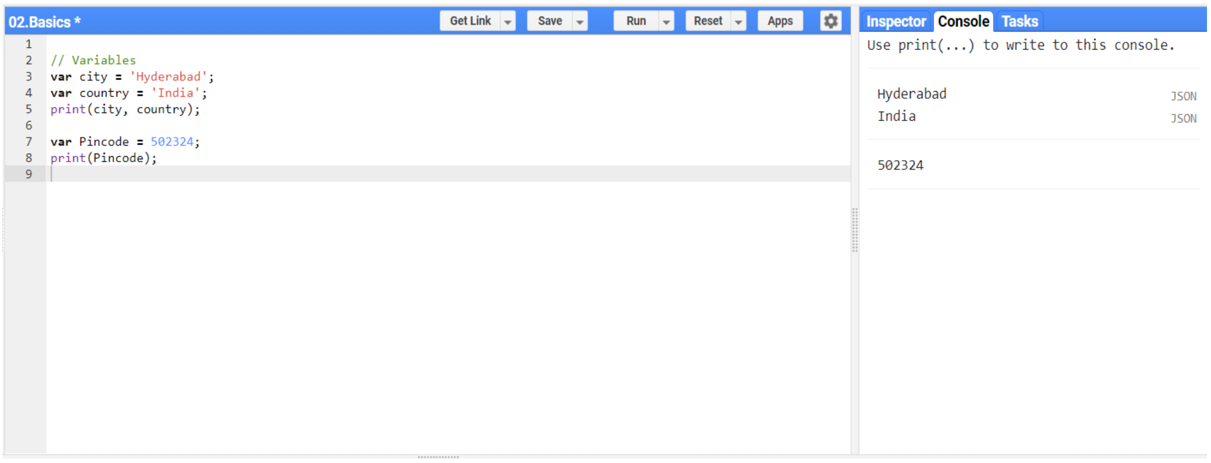
var city = 'Hyderabad';

var country = 'India';

print(city, country);

var Pincode = 502324;

print(Pincode);



```
// Variables
var city = 'Hyderabad';
var country = 'India';
print(city, country);

var Pincode = 502324;
print(Pincode);
```

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

Hyderabad                                   JSON
India                                       JSON

502324
```

- **List, Dictionary and Functions:**
- *// List and Dictionary*

var majorCities = ['Hyderabad', 'Mumbai', 'Delhi', 'Chennai', 'Kolkata'];

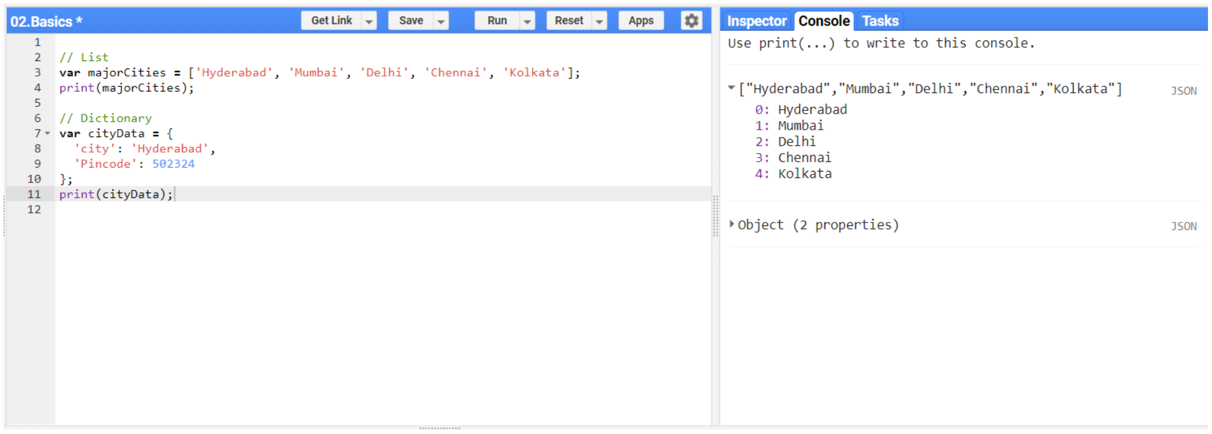
print(majorCities);

// Dictionary

var cityData = { 'city': 'Hyderabad', 'Pincode': 502324

};

print(cityData);



```
// List
var majorCities = ['Hyderabad', 'Mumbai', 'Delhi', 'Chennai', 'Kolkata'];
print(majorCities);

// Dictionary
var cityData = {
  'city': 'Hyderabad',
  'Pincode': 502324
};
print(cityData);
```

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

▾["Hyderabad","Mumbai","Delhi","Chennai","Kolkata"]     JSON
    0: Hyderabad
    1: Mumbai
    2: Delhi
    3: Chennai
    4: Kolkata

▸Object (2 properties)                                   JSON
```

- **Functions**

// Function

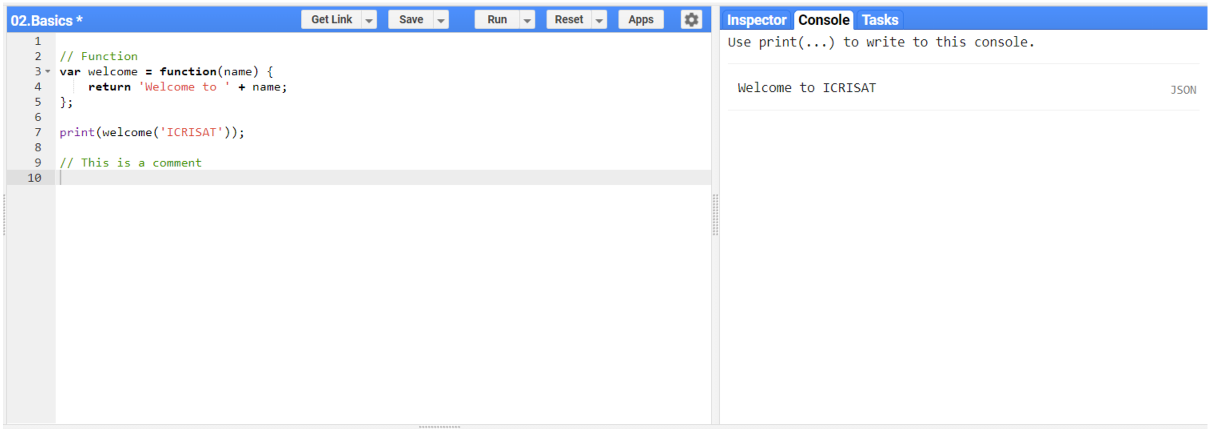
var welcome = function(name) {

return 'Welcome to ' + name;

};

print(welcome('ICRISAT'));

"//" is used to write comments about the code or line of code to make it more clear about what you have written.

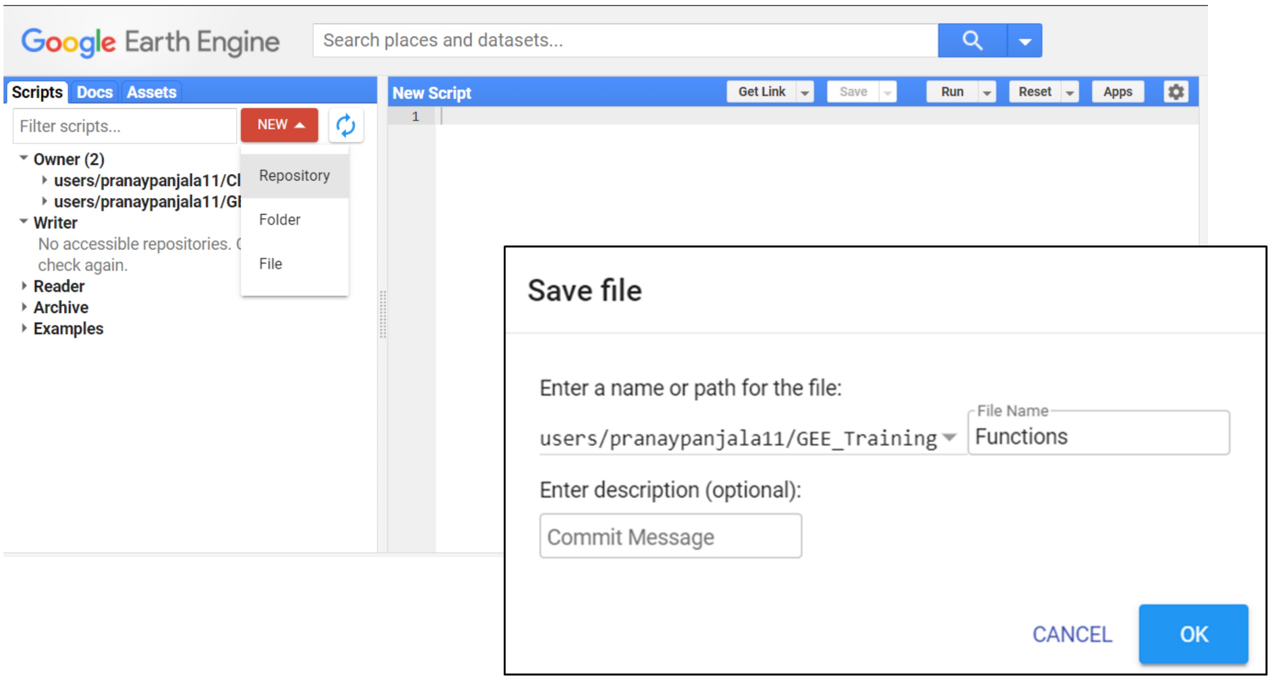


```
// Function
var welcome = function(name) {
    return 'Welcome to ' + name;
};

print(welcome('ICRISAT'));

// This is a comment
```

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

Welcome to ICRISAT
```

7. Save your Work

- Save your work by clicking save button on above code editor in any folder.
- Create Repository and folder
- Save your work as file in your folder



Save file

Enter a name or path for the file:

users/pranaypanjala11/GEE_Training

File Name: Functions

Enter description (optional):

Commit Message

CANCEL OK

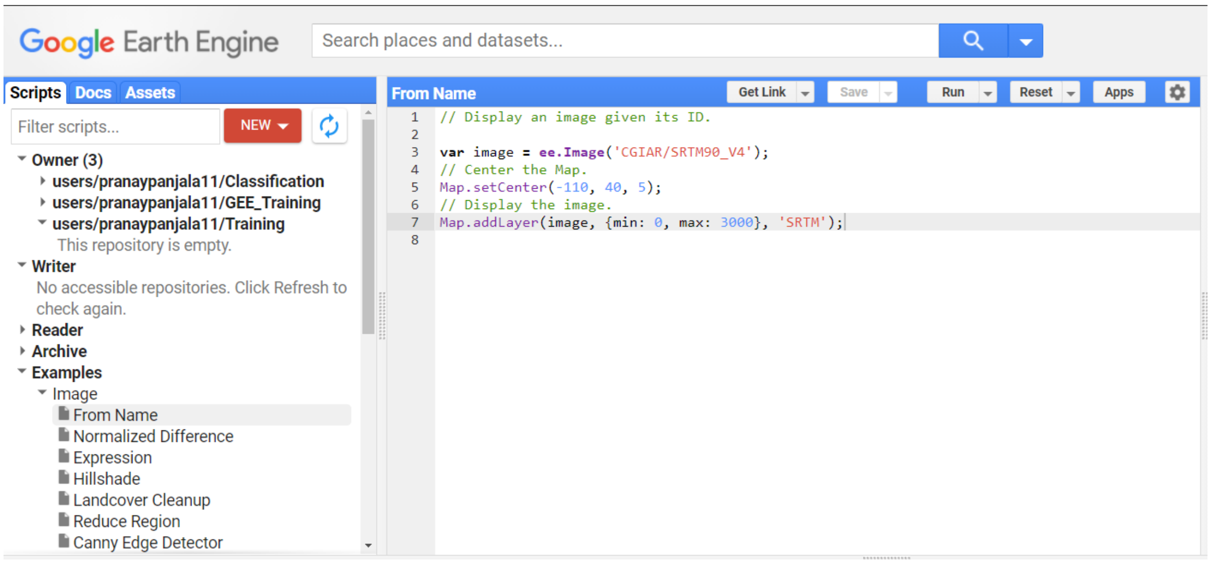
8. Hands on Training – Google Earth Engine

- Image, the fundamental raster data type in Earth Engine.
- ImageCollection, a stack or time-series of images.
- Geometry, the fundamental vector data type in Earth Engine.
- Feature, or a Geometry with attributes.
- FeatureCollection, or a set of features.
- Reducer, an object used to compute statistics or perform aggregations.
- Join, or how to combine datasets (Image or Feature collections) based on time, location, or an attribute property.
- Array, for multi-dimensional analyses.

9. Image

- Import images by search from available datasets
- Also import the images by using search datasets

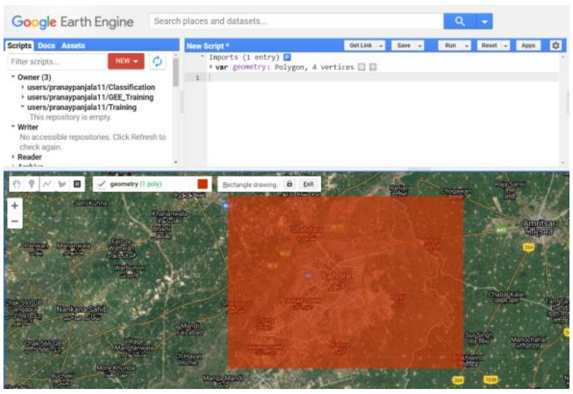
Example: https://code.earthengine.google.com/2ae789bbd115d7a4e9aa3ef3a210346c



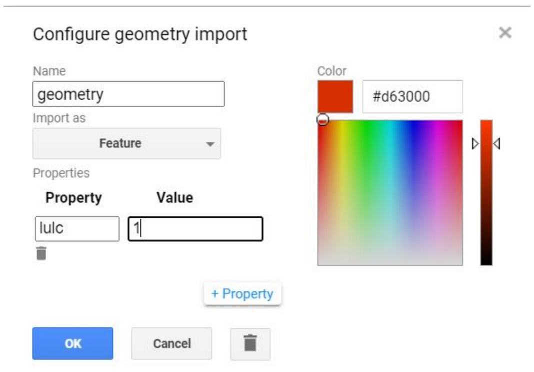
```
// Display an image given its ID.

var image = ee.Image('CGIAR/SRTM90_V4');
// Center the Map.
Map.setCenter(-110, 40, 5);
// Display the image.
Map.addLayer(image, {min: 0, max: 3000}, 'SRTM');
```

10. Geometry and Feature

The illustration of drawing geometry and defining feature with property


Google Earth Engine
Search places and datasets...
Scripts
Docs
Assets
New Script *
Get Link
Save
Run
Reset
Apps
Filter scripts...
NEW
Owner (3)
users/pranaypanjala11/Classification
users/pranaypanjala11/GEE_Training
users/pranaypanjala11/Training
This repository is empty.
Writer
No accessible repositories. Click Refresh to check again.
Reader
geometry (1 poly)
Rectangle drawing.
Exit
Amritsar
Lahore

```
Imports (1 entry)
  var geometry: Polygon, 4 vertices
```



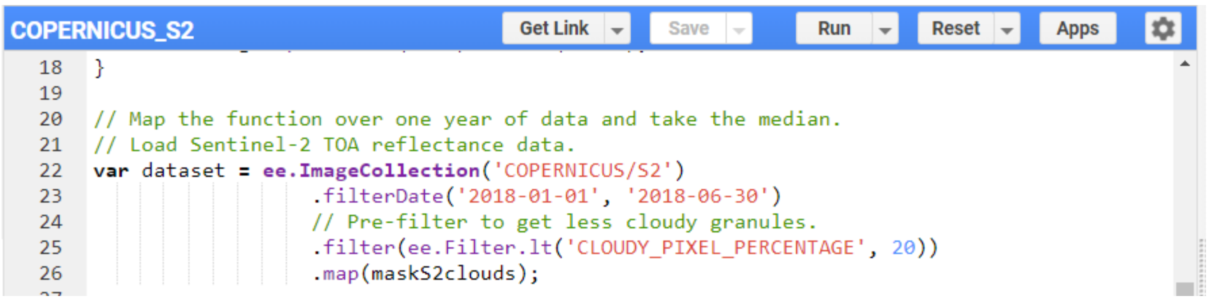
Name: geometry
Import as: Feature
Color: #d63000
Properties

Property	Value
lulc	1

11. ImageCollection – FeatureCollection

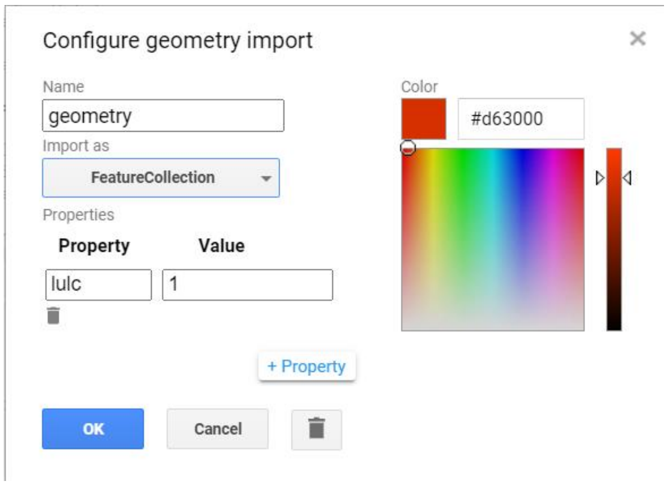
- ImageCollection, a stack or time-series of images.

Example: https://code.earthengine.google.com/44264679da5eef579263023818cd54f1



```
}

// Map the function over one year of data and take the median.
// Load Sentinel-2 TOA reflectance data.
var dataset = ee.ImageCollection('COPERNICUS/S2')
                  .filterDate('2018-01-01', '2018-06-30')
                  // Pre-filter to get less cloudy granules.
                  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 20))
                  .map(maskS2clouds);
```

- FeatureCollection, or a set of features.



Name: geometry
Import as: FeatureCollection
Properties

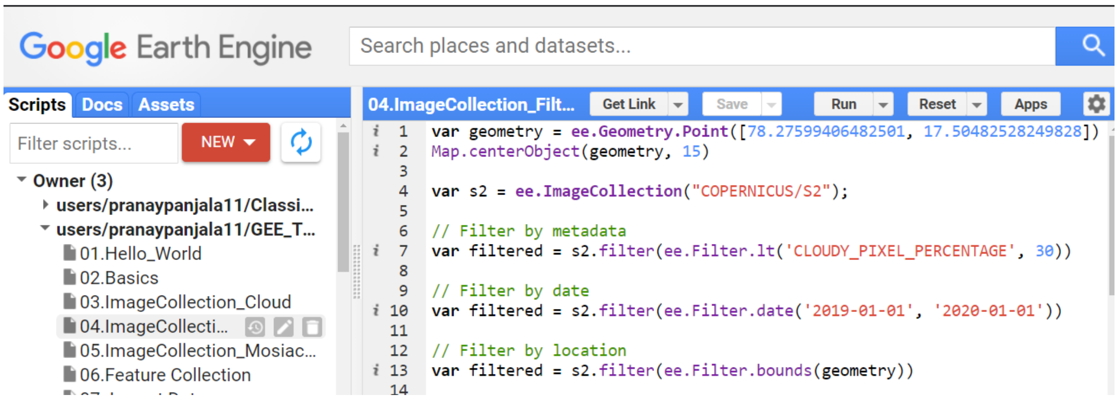
Property	Value
lulc	1

Color: #d63000

12. ImageCollection – Filters

(https://code.earthengine.google.com/7f9e8198e591d4b7318e80121d1d0580)

- .filterDate – filter used to get images between range of dates
- .filterBounds –Geometry or study area of selection
- . filter(ee.Filter.lt(‘attribute', 30)) – for selection of any specified attributes and ranges
- .select – for selection of bands

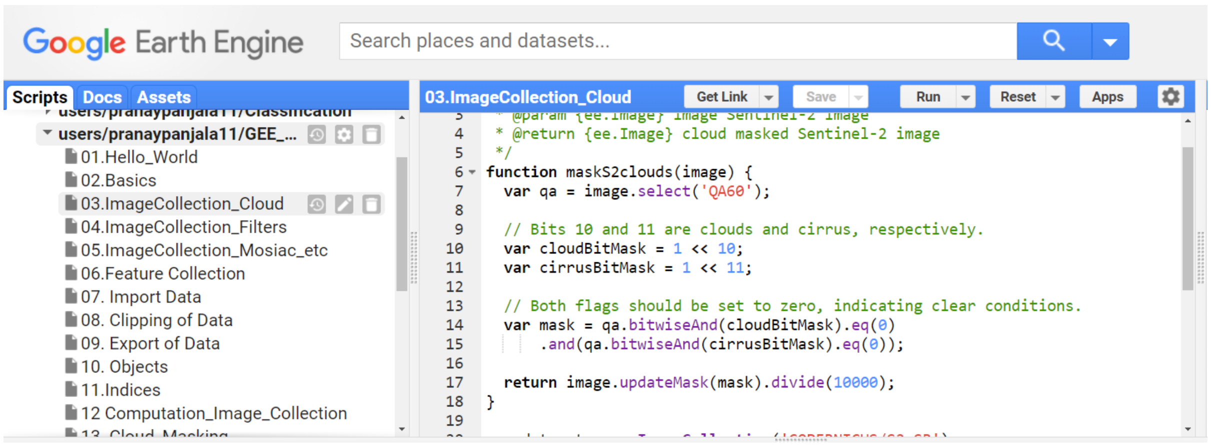


```
var geometry = ee.Geometry.Point([78.27599406482501, 17.50482528249828])
Map.centerObject(geometry, 15)

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

// Filter by metadata
var filtered = s2.filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30))

// Filter by date
var filtered = s2.filter(ee.Filter.date('2019-01-01', '2020-01-01'))

// Filter by location
var filtered = s2.filter(ee.Filter.bounds(geometry))
```

13. ImageCollection – Cloud Thresold

- Function used to remove cloud pixels by applying mask with QA60 band

(https://code.earthengine.google.com/d514cc4a2045e1fc67fb1af282bf16c2)



```
 * @param {ee.Image} image Sentinel-2 image
 * @return {ee.Image} cloud masked Sentinel-2 image
 */
function maskS2clouds(image) {
  var qa = image.select('QA60');

  // Bits 10 and 11 are clouds and cirrus, respectively.
  var cloudBitMask = 1 << 10;
  var cirrusBitMask = 1 << 11;

  // Both flags should be set to zero, indicating clear conditions.
  var mask = qa.bitwiseAnd(cloudBitMask).eq(0)
      .and(qa.bitwiseAnd(cirrusBitMask).eq(0));

  return image.updateMask(mask).divide(10000);
}

```

14. ImageCollection – Composite – Mosaic -Reducing

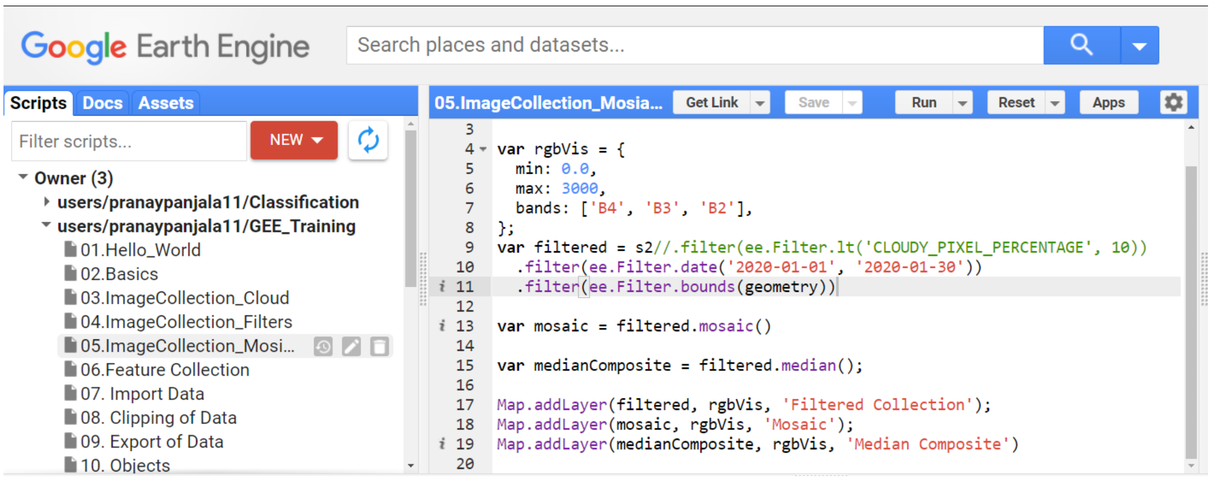
We can reduce the set of images into single image using reducers like

- Median
- Mean
- Max

And

- Mosaic for mosaicking images (last images)

(https://code.earthengine.google.com/4de07cad1369e037ec3c20169bb823a4)

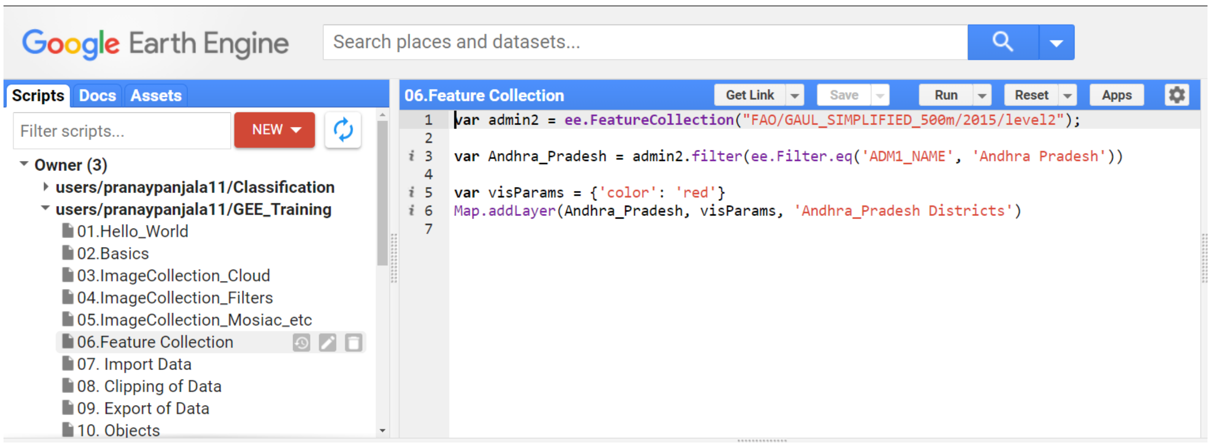


```

var rgbVis = {
  min: 0.0,
  max: 3000,
  bands: ['B4', 'B3', 'B2'],
};
var filtered = s2//.filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10))
  .filter(ee.Filter.date('2020-01-01', '2020-01-30'))
  .filter(ee.Filter.bounds(geometry))

var mosaic = filtered.mosaic()

var medianComposite = filtered.median();

Map.addLayer(filtered, rgbVis, 'Filtered Collection');
Map.addLayer(mosaic, rgbVis, 'Mosaic');
Map.addLayer(medianComposite, rgbVis, 'Median Composite')

```

15. Feature Collection - Filters

We can filter the filter collection using attributes

(https://code.earthengine.google.com/d1fbb18ba0bb08d0a198a77843b89705)

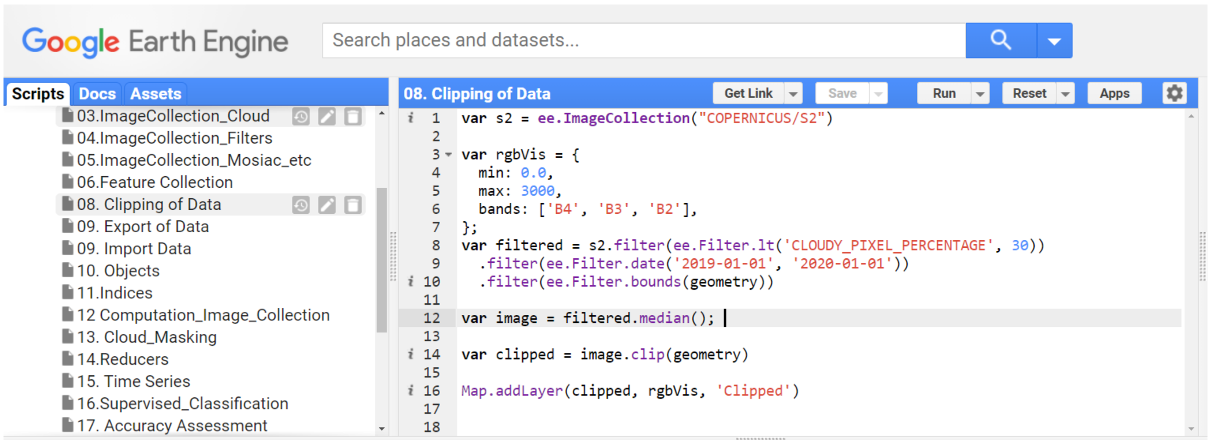


```
var admin2 = ee.FeatureCollection("FAO/GAUL_SIMPLIFIED_500m/2015/level2");

var Andhra_Pradesh = admin2.filter(ee.Filter.eq('ADM1_NAME', 'Andhra Pradesh'))

var visParams = {'color': 'red'}
Map.addLayer(Andhra_Pradesh, visParams, 'Andhra_Pradesh Districts')
```

16. Clipping and Mask of Data

- We can clip the data at required geometry or study area using .clip and Mask using .updateMask

(https://code.earthengine.google.com/a543374c3b6262f324b04484a5802173)



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

var rgbVis = {
  min: 0.0,
  max: 3000,
  bands: ['B4', 'B3', 'B2'],
};
var filtered = s2.filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30))
  .filter(ee.Filter.date('2019-01-01', '2020-01-01'))
  .filter(ee.Filter.bounds(geometry))

var image = filtered.median();

var clipped = image.clip(geometry)

Map.addLayer(clipped, rgbVis, 'Clipped')

```

17. Calculating Indices – Image & ImageCollection

- We can calculate the indices using Bands of image with mathematical expressions

For example:

var savi = image.expression(

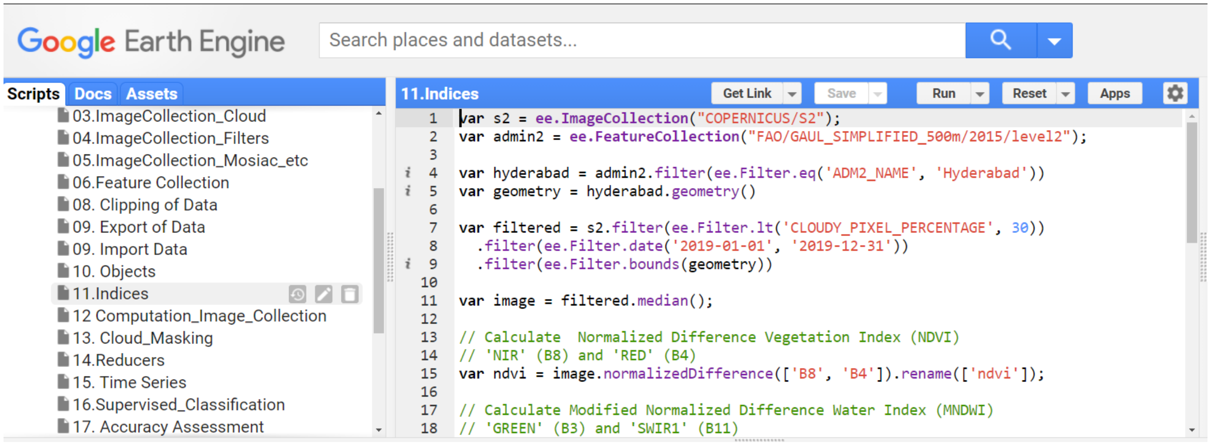
*'1.5 * ((NIR - RED) / (NIR + RED + 0.5))', {*

'NIR': image.select('B8').multiply(0.0001),

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

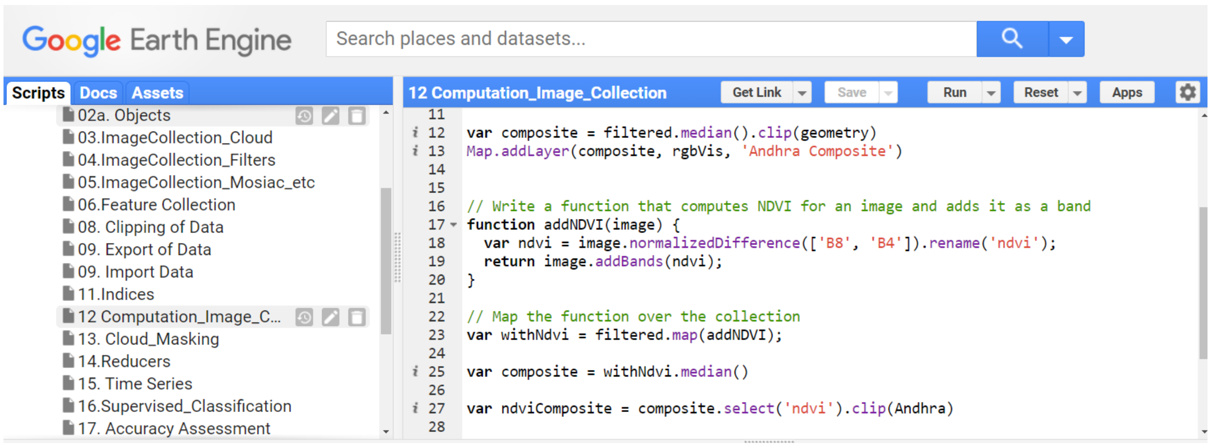
}).rename('savi');

- On image: (https://code.earthengine.google.com/fe767d1b7e488f14bcbf7229abe18b79)



```
var s2 = ee.ImageCollection("COPERNICUS/S2");
var admin2 = ee.FeatureCollection("FAO/GAUL_SIMPLIFIED_500m/2015/level2");

var hyderabad = admin2.filter(ee.Filter.eq('ADM2_NAME', 'Hyderabad'))
var geometry = hyderabad.geometry()

var filtered = s2.filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30))
  .filter(ee.Filter.date('2019-01-01', '2019-12-31'))
  .filter(ee.Filter.bounds(geometry))

var image = filtered.median();

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

// Calculate Modified Normalized Difference Water Index (MNDWI)
// 'GREEN' (B3) and 'SWIR1' (B11)
```

- On imageCollection: using function (https://code.earthengine.google.com/e5d39e7a0648c9b0a922149b26b610bd)



```

var composite = filtered.median().clip(geometry)
Map.addLayer(composite, rgbVis, 'Andhra Composite')


// Write a function that computes NDVI for an image and adds it as a band
function addNDVI(image) {
  var ndvi = image.normalizedDifference(['B8', 'B4']).rename('ndvi');
  return image.addBands(ndvi);
}

// Map the function over the collection
var withNdvi = filtered.map(addNDVI);

var composite = withNdvi.median()

var ndviComposite = composite.select('ndvi').clip(Andhra)

```

18. **<u>Reducers</u>**

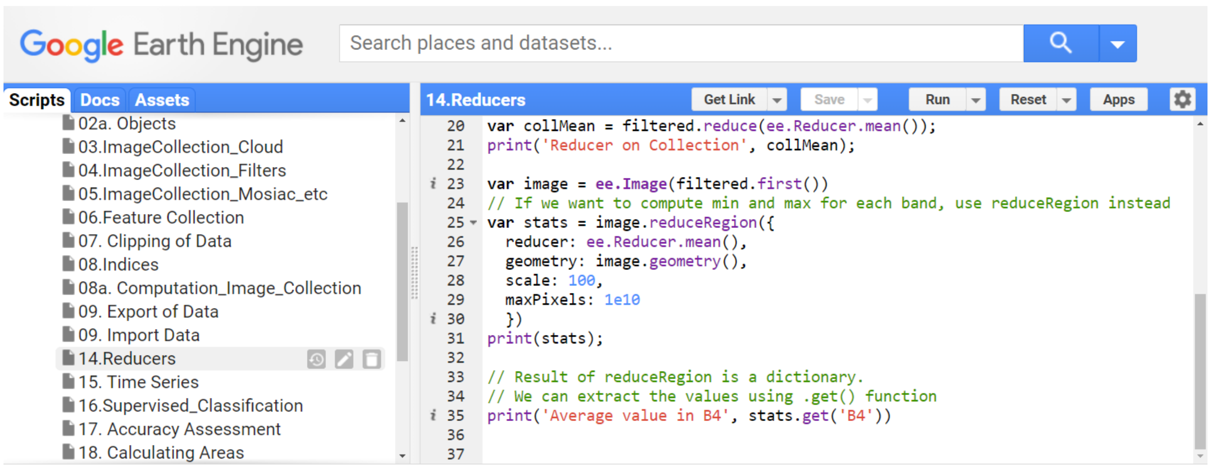
- Used to calculated mean of all pixels in specified geometry – ee.Reducer.mean()

Maximum - ee.Reducer.max()

Minimum - ee.Reducer.max()

Median - ee.Reducer.median()

Reducers: (https://code.earthengine.google.com/0535376ac6fd6f58aaee819921ff23ee)

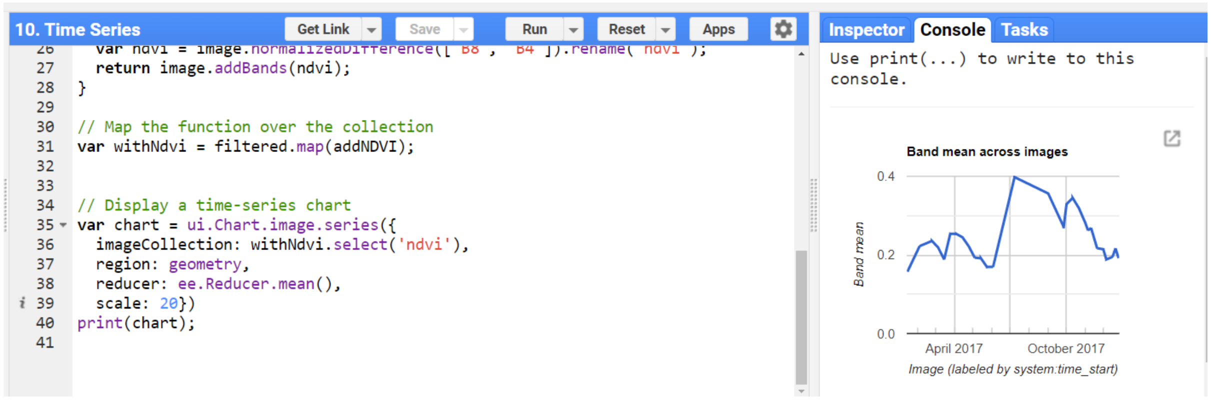


```
var collMean = filtered.reduce(ee.Reducer.mean());
print('Reducer on Collection', collMean);

var image = ee.Image(filtered.first())
// If we want to compute min and max for each band, use reduceRegion instead
var stats = image.reduceRegion({
  reducer: ee.Reducer.mean(),
  geometry: image.geometry(),
  scale: 100,
  maxPixels: 1e10
  })
print(stats);

// Result of reduceRegion is a dictionary.
// We can extract the values using .get() function
print('Average value in B4', stats.get('B4'))
```

19. Time Series –View

- Applying indices for image collection and reducing the image using reducer
- Plotting TimeSeries using ui.Chart.image.series

Time Series: (https://code.earthengine.google.com/adee70cddffe525c5ab5b115cd819e00)



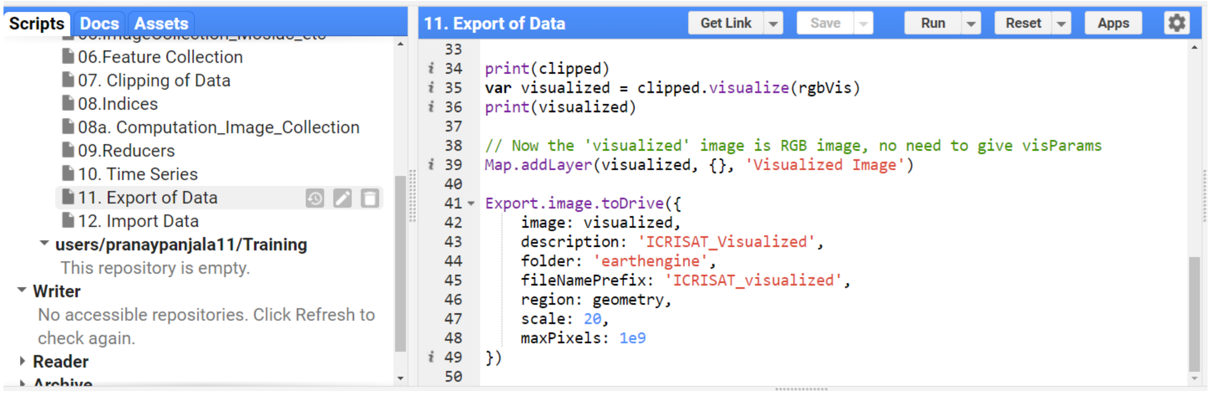
```
  var ndvi = image.normalizedDifference(['B8', 'B4']).rename('ndvi');
  return image.addBands(ndvi);
}

// Map the function over the collection
var withNdvi = filtered.map(addNDVI);


// Display a time-series chart
var chart = ui.Chart.image.series({
  imageCollection: withNdvi.select('ndvi'),
  region: geometry,
  reducer: ee.Reducer.mean(),
  scale: 20})
print(chart);
```

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

20. Import and Export of Data

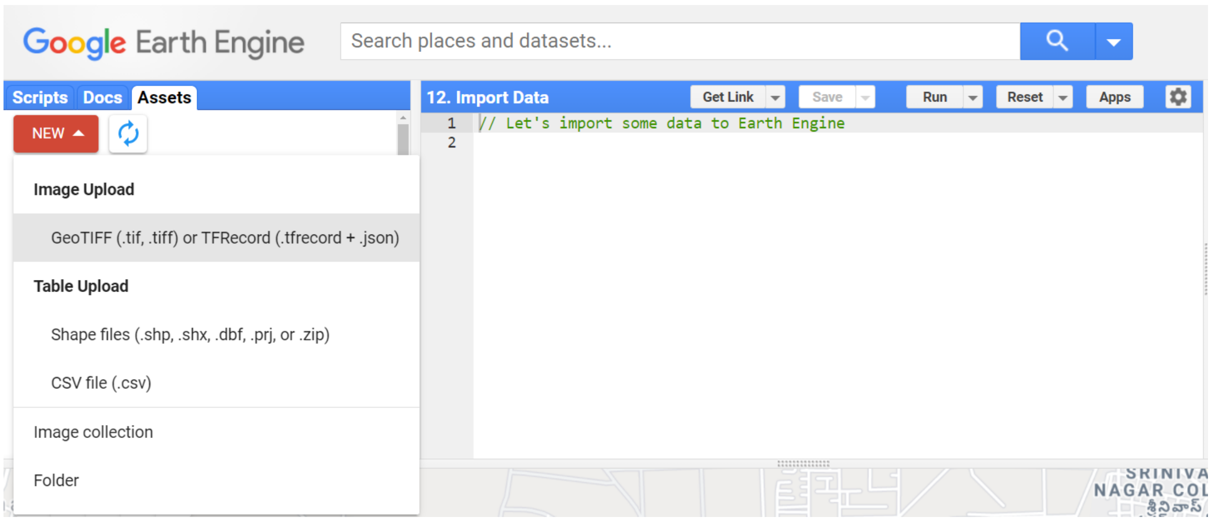
- Export of image using Export.image.toDrive() and we can also export feature collection to Google Drive, to asset, to cloud

(https://code.earthengine.google.com/e48705fdbb697b9ac5fdd4358f8115e1)



```

print(clipped)
var visualized = clipped.visualize(rgbVis)
print(visualized)

// Now the 'visualized' image is RGB image, no need to give visParams
Map.addLayer(visualized, {}, 'Visualized Image')

Export.image.toDrive({
    image: visualized,
    description: 'ICRISAT_Visualized',
    folder: 'earthengine',
    fileNamePrefix: 'ICRISAT_visualized',
    region: geometry,
    scale: 20,
    maxPixels: 1e9
})
```

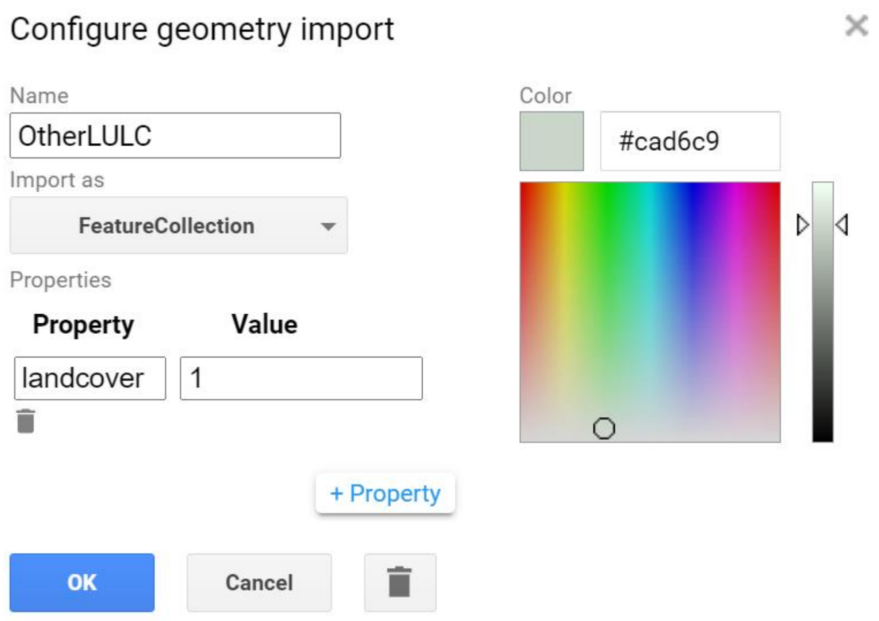
- Import of data either through search or through asset, we can upload image (.tif) and shapefile as per our requirement



```
// Let's import some data to Earth Engine
```

21. Supervised Classification – Random Forest Algorithm – Accuracy Assessment – Calculating Areas (https://code.earthengine.google.com/432fa9196cc9981b0c6c0e1d7d0e2816)

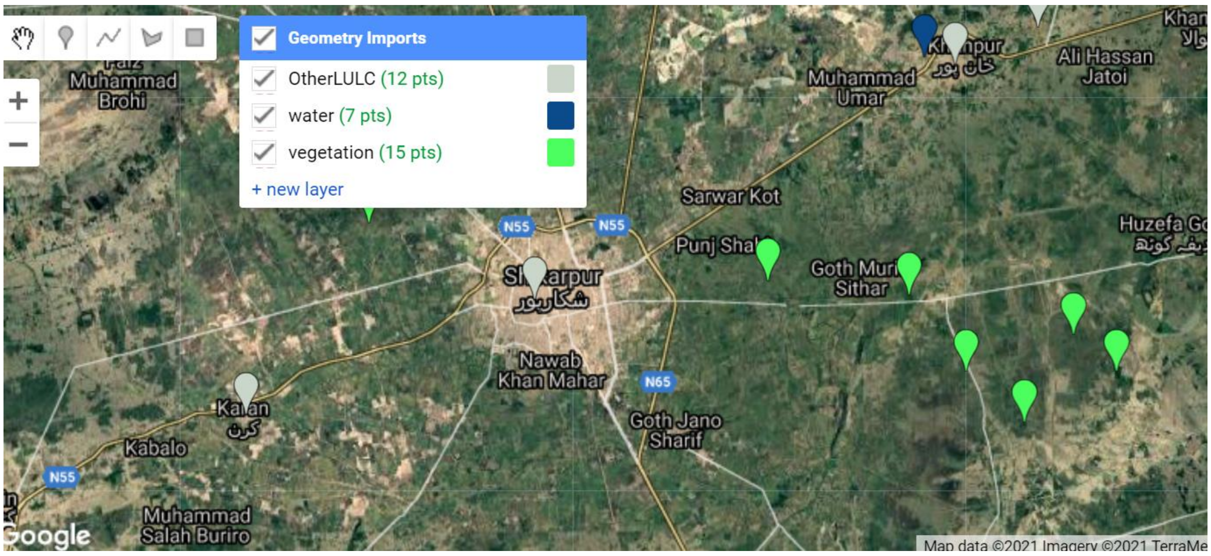
- we have taken Sindh district as study area
- First, we have to collect some training data (feature Collection) for classification
    i. Other LULC - property name – landcover - value 1
    ii. Water – property name – landcover - value 2
    iii. Vegetation - property name – landcover - value 3



Name: OtherLULC
Import as: FeatureCollection
Properties

Property	Value
landcover	1

Color: #cad6c9


Geometry Imports

OtherLULC (12 pts)

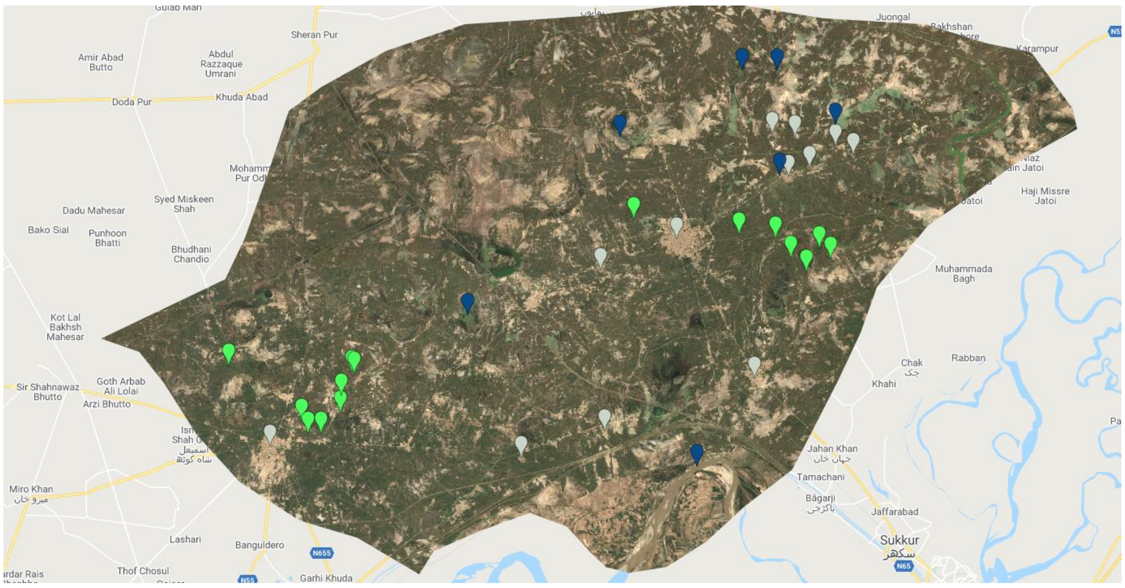
water (7 pts)

vegetation (15 pts)

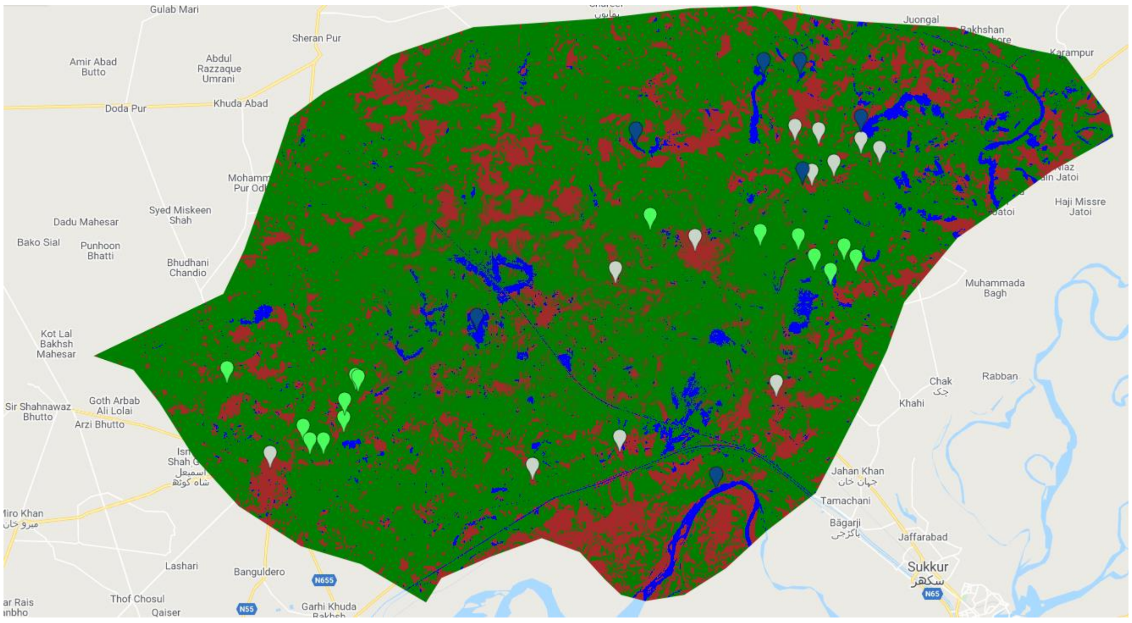
+ new layer

After Collection of Data, we will perform Random Forest Algorithm on band stack

```
    // Overlay the point on the image to get training data.
var training = composite.sampleRegions({
  collection: gcps,
  properties: ['landcover'],
  scale: 10
});
// Train a classifier.
var classifier = ee.Classifier.smileRandomForest(50).train({
  features: training,
  classProperty: 'landcover',
  inputProperties: composite.bandNames()
});
// // Classify the image.
var classified = composite.classify(classifier);
Map.addLayer(classified, {min: 1, max: 3, palette: [ 'brown', 'blue', 'green']}, '2019');
```



Before Classification



After Classification

- **For Accuracy Assessment,** we will divide the collected into training as well as validation points

// Add a random column and split the GCPs into training and validation set

var gcp = gcps.randomColumn()

// This being a simpler classification, we take 60% points

// for validation. Normal recommended ratio is

// 70% training, 30% validation

var trainingGcp = gcp.filter(ee.Filter.lt('random', 0.6));

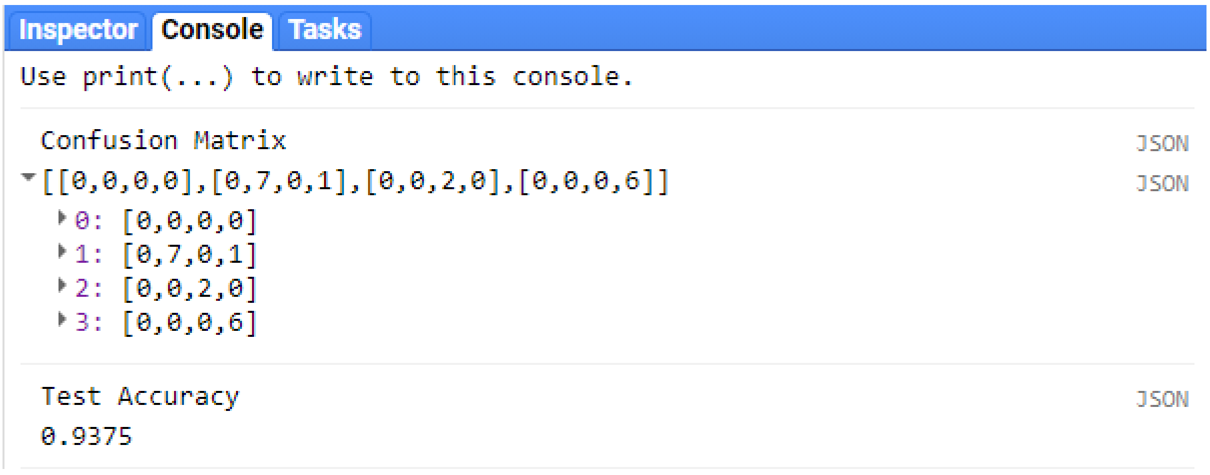
var validationGcp = gcp.filter(ee.Filter.gte('random', 0.6));

var testConfusionMatrix = test.errorMatrix('landcover', 'classification')

// Printing of confusion matrix may time out. Alternatively, you can export it as CSV

print('Confusion Matrix', testConfusionMatrix);

print('Test Accuracy', testConfusionMatrix.accuracy());



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

Confusion Matrix
[[0,0,0,0],[0,7,0,1],[0,0,2,0],[0,0,0,6]]
  0: [0,0,0,0]
  1: [0,7,0,1]
  2: [0,0,2,0]
  3: [0,0,0,6]

Test Accuracy
0.9375
```

- **For Calculating Areas**

// Area Calculation for Images

var vegetation = classified.eq(3)

Map.addLayer(vegetation, {min:0, max:1, palette: ['red', 'green']}, 'Green Cover')

// pixels will have values equal to their area

var areaImage = vegetation.multiply(ee.Image.pixelArea())

// Now that each pixel for vegetation class in the image has the value

var area = areaImage.reduceRegion({

reducer: ee.Reducer.sum(),

geometry: Sind.geometry(),

tileScale: 16,

scale: 10,

```
  maxPixels: 1e10
  })
// The result of the reduceRegion() function is a dictionary with the key
// being the band name. We can extract the area number and convert it to
// square kilometers
var vegetationAreaSqKm = ee.Number(area.get('classification')).divide(1e6).round()
print(vegetationAreaSqKm)
```

Hope, you got a glance on how GEE works and its applications, if you want to learn further level i.e. advance level. You can visit the Google Earth Engine Tutorials

https://developers.google.com/earth-engine/tutorials/tutorials

Big Thanks to Google for this.

/* Thanks for Participating, Have a great learning */