Using Google Earth Engine With Colab

Eric Xia, GIS and Data Asst., Brown University

January 30, 2024

https://libguides.brown.edu/gis_data_tutorials/google_earth_engine

---

1 Introduction

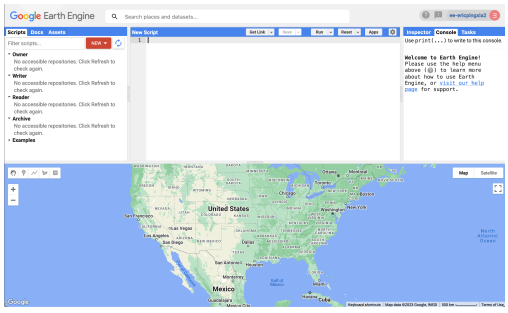
Google Earth Engine is a planetary-scale platform for satellite imagery and related datasets, available at no cost for academic and personal use. The most commonly used datasets includes satellite imagery from the LANDSAT, SENTINEL, and MODIS programs. However, there are hundreds of other geospatial datasets accessible, including atmospheric, weather and climate data, geological classifications, vegetation indices, and DEMs.

While Earth Engine provides an invaluable source of data, it can be difficult to integrate the custom Javascript editor into the research process. This tutorial uses the Python Google Earth Engine API in conjunction with Google Colab, an online code editor that makes it easy to create and share notebooks. Using Colab, we will demonstrate how to access and display satellite imagery from Google Earth Engine. Then, we will perform some basic geoprocessing on the resulting images. Finally, we will export the data to Google Drive.

In order to use the Google Earth Engine API, your account need to have permission to create Google Cloud Projects, which some organizations (like Brown University) do not permit. In these cases, it is recommended that you use a personal non-organizational account. Otherwise, you can also obtain noncommercial access without creating a Cloud Project by completing this signup form.

Registering for GEE

Go to `https://earthengine.google.com/`, and click *Get Started*. Register your project, selecting *Unpaid Usage*, and *Academia & Research*. Create a new Google Cloud Project. You can configure it however you want, but make a note of the Project ID. Confirm your Cloud Project information; you should be taken to a scripting editor under the Cloud Project you have created.



Although we won't be using the online code editor, the setup here is necessary for notebook authentication and initialization.

GEE Authentication in Google Colab

If you want to follow along, go to `https://colab.research.google.com` and make a new notebook. If you want to just run the code, open this notebook and select File > Save a Copy in Drive. Each section of code below is a separate block.

Our code will mainly use two libraries. `ee` is the Python API for Google Earth Engine, while `geemap` provides essential add-ons for working with Google Earth Engine. Among other things, this includes a version of the Google Earth Engine map we saw in the editor above. Run a code cell with the following contents:

```
import ee
import geemap
```

Now, go ahead and authenticate Earth Engine.

```
ee.Authenticate()
ee.Initialize(project="GOOGLE-CLOUD-PROJECT-ID")
```

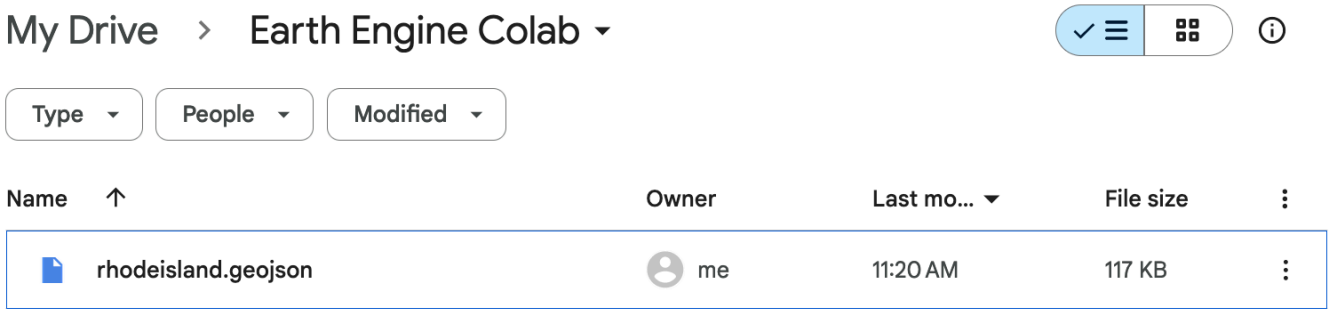
With the ID for the project you set up previously, you should be able to run through the authentication and initialization process. If Google alerts you that the app is unverified, click *continue*: you'll be authoring the client app in the notebook! Also, make sure to leave read-only scope unchecked.

Import Data from Google Drive

Let's now mount Google Drive and display some vector data with `geemap`. At the bottom of the right toolbar, expand the Files menu and then click Mount Drive, or run the following cell.

```
from google.colab import drive
drive.mount('/content/drive')
```

Now, we can upload any kind of geospatial data and interact with it in Colab. In My Drive, make a new folder named 'Earth Engine Colab'. Upload this GeoJSON file of Rhode Island to the newly created folder.

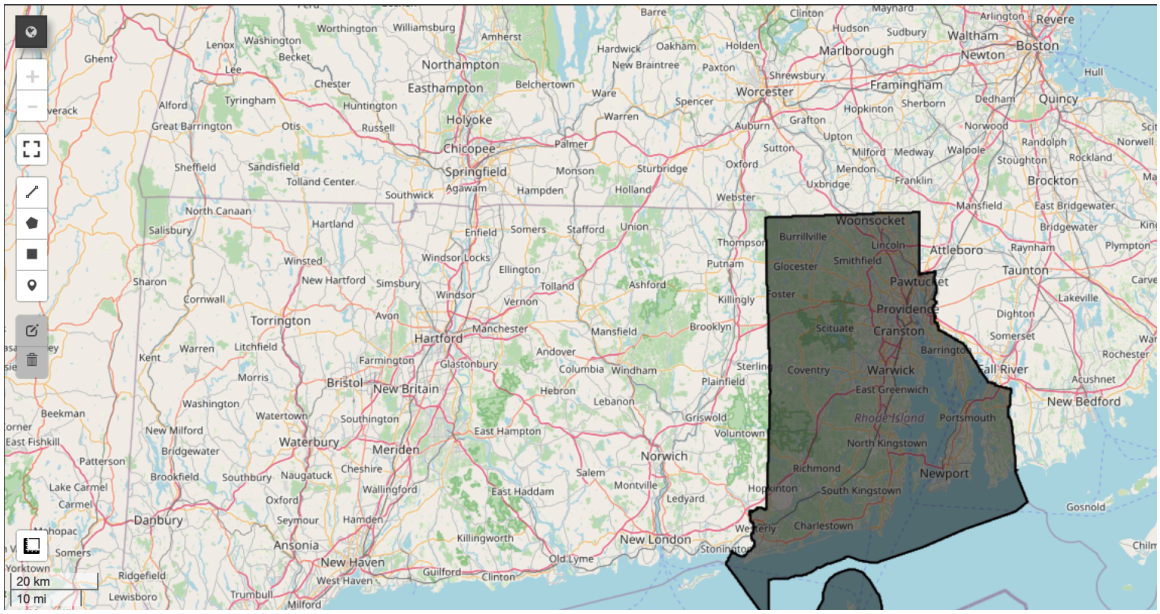


We can now view the polygon as a layer on the geemap map:

```
import json
path = "../content/drive/MyDrive/Earth Engine Colab/rhodeisland.geojson"
region = geemap.geojson_to_ee(path)

map = geemap.Map()
map.centerObject(region, 9)
map.addLayer(region, {}, 'region')
map
```

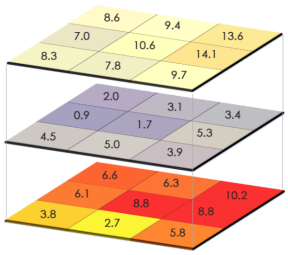
You should see the following display in the notebook:

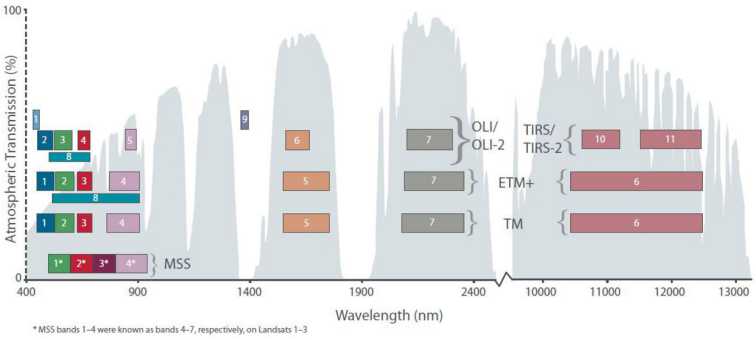


Retrieve Raster Data

The raster images on Google Earth Engine are often in several bands. A band is a matrix of values for a certain variable, where each variable corresponds to a pixel (square area). These variables might correspond to the average rainfall, or surface temperature at a given time. More routinely, RGB images are composed of red, blue, and green bands, which are the visible portion of the electromagnetic spectrum. On the left, you can see how a raster is made of several bands.

Satellites like LANDSAT take images by sensing reflected light from the Earth's surface. On the right, you can see the reflected light by wavelength, and the bands which the satellite captures. The acronyms are for sensors on different Landsat satellites; the numbers are for the bands gathered by respective sensors. Note that the lower bands are RGB, while the higher ones are infrared.





The data we're using here is 1:30M scale, which means each pixel in a band represents a 30 x 30 meter square area. We'll be using the LANDSAT 8 Tier 1 Surface Reflectance dataset. In Google Earth Engine, a set of images is known as an ImageCollection. By initializing an ImageCollection, we import the associated data:

```
l8 = ee.ImageCollection('LANDSAT/LC08/C02/T1_TOA');
```

We've successfully imported the ImageCollection. However, l8 is a set of images; there's no way to visualize them all at once. We want to display a single image. Let's first filter by location, using the centroid of the state region:

```
centroid = region.geometry().centroid(maxError=1)
spatialFiltered = l8.filterBounds(centroid)
```

And then filter by date:

```
startDate = '2015-01-01';
endDate = '2015-12-31'
temporalFiltered = spatialFiltered.filterDate(startDate, endDate);
```

We've filtered our collection for location and date, but still can't be displayed, as our data varies by time. To get a single image out of the ImageCollection, we can use `first()` or `reduce()`. For example, we can sort by cloud cover and take the least cloudy one.

```
cloudless = temporalFiltered.sort('CLOUD_COVER').first();
```

Or, we can get composites of all of the images in the time range by running functions over our collection. Think about the differences between taking the pixel-wise mean, median, and the minimum/maximum across a time range. Which one is best for avoiding clouds?

```
mean = temporalFiltered.reduce(ee.Reducer.mean());
median = temporalFiltered.reduce(ee.Reducer.median());
min = temporalFiltered.reduce(ee.Reducer.min());
```

Displaying Rasters

When visualizing map data, Google Earth Engine can't guess what you want. For any multi-band image, it will default to displaying B1, B2, and B3 as red, blue, and green. It will also stretch them to a [0,1] range. When applied to the LANDSAT data, which has different band numbering, this results in a bad image. To display an RGB composite – what you and I would recognize as satellite imagery – we need to tell it the actual corresponding bands in the LANDSAT data. To figure this out, we read the metadata on the Google Earth Engine Catalog page:

Name	Pixel Size	Wavelength	Description
B1	30 meters	0.43 - 0.45 µm	Coastal aerosol
B2	30 meters	0.45 - 0.51 µm	Blue
B3	30 meters	0.53 - 0.59 µm	Green
B4	30 meters	0.64 - 0.67 µm	Red
B5	30 meters	0.85 - 0.88 µm	Near infrared
B6	30 meters	1.57 - 1.65 µm	Shortwave infrared 1
B7	30 meters	2.11 - 2.29 µm	Shortwave infrared 2

It tells us we should display bands B4, B3, and B2 as red, blue, green, and to set the max value to 0.3. With the following code, our image displays as expected:

```
visParams = {'bands': ['B4', 'B3', 'B2'], 'max': 0.3};
map.addLayer(cloudless, visParams, 'true-color composite');
map
```

Perform Operations on Raster Data

Let's go ahead and create a NDVI band for the data. NDVI stands for Normalized Difference Vegetation Index, and is defined as follows, where NIR is the near-infrared band and RED is the red band:

$$\text{NDVI} = \frac{\text{NIR} - \text{RED}}{\text{NIR} + \text{RED}}$$

NDVI is an important estimate of the health and density of vegetation in a particular region. Verdant, green land reflect much less red light and much more near-infrared light than barren land, so the difference NIR − RED varies with the quantity of vegetation present. However the total intensity NIR + RED can vary, so we normalize the index by dividing this quantity out.

We first define a NDVI function, which takes an image and calculates the per-pixel NDVI.

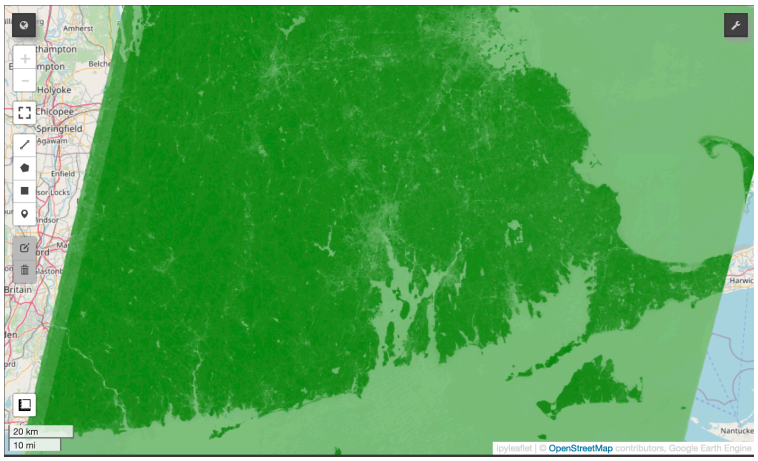
Then, we map the function over our ImageCollection with Google Earth Engine's 'map()' function. This time, we reduce the collection with 'max()', to create a greenest pixel composite. There are other ways of compositing NDVI over time, such as using qualityMosaic or taking 95 percentile values, which we won't cover here.

```
def addNDVI(image):
  ndvi = image.normalizedDifference(['B5', 'B4']).rename('NDVI');
  return image.addBands(ndvi);

withNDVI = temporalFiltered.map(addNDVI)
max = withNDVI.reduce(ee.Reducer.max());
```

Displaying the `max` Image should display the following maximum NDVI for the entire year:

```
vizParams = {'bands': ['NDVI_max'], 'min': -1, 'max': 1,
'palette': ['white', 'green']};
map.addLayer(max, vizParams, 'Greenest pixel composite');
map
```



Export Data to Drive

We've created an NDVI band for our ImageCollection. The final step is to export the data to Google Drive. To do this, we use `batch.Export.image.toDrive`. We only want the NDVI band, and we want to clip the polygon exported to the region we defined.

The parameters are the image, the coordinate reference system, a region to clip to, a description (the filename of the output file), and a destination folder. There is also a scale parameter where you should specify the meters per pixel.

```
task = ee.batch.Export.image.toDrive(
  image=max.select('NDVI_max'),
  crs='EPSG:4326',
  region=region.geometry(),
  description=f'NDVI_max_{startDate}_{endDate}',
  folder='../content/drive/MyDrive/Earth Engine Colab',
  scale=30);

task.start()
```

Make sure to specify the region parameter! If you don't specify a region, the output TIFF file can be very large, and the task will take a very long time to complete.

To check on the status of the task, you can use `task.status()`:

```
task.status()
```

References

To summarize, we registered a Cloud Project for Google Earth Engine and set up Google Colab to work with it. Then, we imported vector data from Google Drive, retrieved raster data filtered by spatial and temporal data, and did some basic processing on the data. Finally, we exported new raster data back to the Drive.

Google Earth Engine can be tricky to work with. To protect datasets from unauthorized uses, client-side operations (like `print()`) within `map()` functions are prohibited, which can make understanding what's being retrieved difficult. Below, I list some helpful resources. Some use Javascript instead of Python, but for scripting purposes switching between the two is usually straightforward.

Official Introduction to Google Earth Engine's Python API
Official Earth Engine JavaScript API Tutorials
A collection of 300+ examples for using Earth Engine and the geemap Python package
Google Earth Engine Beginner's Cookbook (Javascript)