Open the Tutorial in your Google Colab!

Open in Colab

Introduction to Google Earth Engine

Google Earth Engine is a cloud-based platform for geospatial analysis, enabling users to process large-scale datasets using Google's infrastructure. It offers multiple interfaces for interaction:

- Explorer
- Code Editor
- JavaScript API
- Python API

Installing Google Earth Engine

Use the following command to install Google Earth Engine (GEE) in your environment. It may already be pre-installed.

```
!pip install earthengine-api
```

```
Requirement already satisfied: earthengine-api in /usr/local/lib/python3.12/dist-packages (1.5.24)
Requirement already satisfied: google-cloud-storage in /usr/local/lib/python3.12/dist-packages (from earthengine-
Requirement already satisfied: google-api-python-client>=1.12.1 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: google-auth>=1.4.1 in /usr/local/lib/python3.12/dist-packages (from earthengine-ap
Requirement already satisfied: google-auth-httplib2>=0.0.3 in /usr/local/lib/python3.12/dist-packages (from earth
Requirement already satisfied: httplib2<1dev,>=0.9.2 in /usr/local/lib/python3.12/dist-packages (from earthengine
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from earthengine-api) (2.32.4
Requirement already satisfied: google-api-core!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.0,<3.0.0,>=1.31.5 in /usr/local/lib/p
Requirement already satisfied: uritemplate<5,>=3.0.1 in /usr/local/lib/python3.12/dist-packages (from google-api-
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from google-aut
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from google-auth
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.12/dist-packages (from google-auth>=1.4.1-
Requirement already satisfied: pyparsing<4,>=3.0.4 in /usr/local/lib/python3.12/dist-packages (from httplib2<1dev
Requirement already satisfied: google-cloud-core<3.0dev,>=2.3.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: google-resumable-media>=2.7.2 in /usr/local/lib/python3.12/dist-packages (from goo
Requirement already satisfied: google-crc32c<2.0dev,>=1.0 in /usr/local/lib/python3.12/dist-packages (from google
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->earthengin
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->eart
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->eart
Requirement already satisfied: googleapis-common-protos<2.0.0,>=1.56.2 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<7
Requirement already satisfied: proto-plus<2.0.0,>=1.22.3 in /usr/local/lib/python3.12/dist-packages (from google-
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.12/dist-packages (from pyasn1-modul
```

Double-click (or enter) to edit

Authenticating to Earth Engine

To access Google Earth Engine (GEE), first sign up at GEE Signup.

Authenticate yourself with Google Cloud to gain access to storage and other resources. Run the following code, which will open an authentication page in your browser.

If this is your first time using GEE, you'll need to create a new project. In the final step, you may also need to register for GEE.

- Select **Unpaid → Organization**
- After completing the process, click **Generate Token**
- Copy and paste the token into the output field below.


```
!earthengine authenticate
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1761836060.227134     289 cuda_dnn.cc:8579] Unable to register cuDNN factory: Attempting to register
E0000 00:00:1761836060.252861     289 cuda_blas.cc:1407] Unable to register cuBLAS factory: Attempting to registe
W0000 00:00:1761836060.325645     289 computation_placer.cc:177] computation placer already registered. Please ch
W0000 00:00:1761836060.325696     289 computation_placer.cc:177] computation placer already registered. Please ch
W0000 00:00:1761836060.325707     289 computation_placer.cc:177] computation placer already registered. Please ch
W0000 00:00:1761836060.325717     289 computation_placer.cc:177] computation placer already registered. Please ch
Authenticate: Limited support in Colab. Use ee.Authenticate() or --auth_mode=notebook instead.
W1030 14:54:30.280470 132247714938880 _default.py:711] No project ID could be determined. Consider running `gclou
```

```
To authorize access needed by Earth Engine, open the following URL in a web browser and follow the instructions.

    https://code.earthengine.google.com/client-auth?scopes=https%3A//www.googleapis.com/auth/earthengine%20https%

The authorization workflow will generate a code, which you should paste in the box below.
Enter verification code: 4/1Ab32j91TiKUYn7vvVGZFawHNPov95stMwPkRJMCLWypNv8sBYSHyNg-H6fQ

Successfully saved authorization token.
```

Authenticating to Earth Engine

To access Google Earth Engine (GEE), first sign up at GEE Signup.

Authenticate yourself with Google Cloud to gain access to storage and other resources. Run the following code, which will open an authentication page in your browser.

If this is your first time using GEE, you'll need to create a new project. In the final step, you may also need to register for GEE.

- Select **Unpaid → Organization**
- After completing the process, click **Generate Token**
- Copy and paste the token into the output field below.

Importing and Initializing Earth Engine

Once authentication is complete, you can import the `ee` module and initialize Google Earth Engine in your environment. This step establishes a connection to the Earth Engine servers, allowing you to run geospatial computations.

Use the following code to import and initialize Earth Engine:

```
import ee
ee.Initialize()
```

Exploring GEE Assets and Datasets

Now that we have initialized EE, lets explore various assets in Google Earth Engine (GEE) to familiarize ourselves with the different types of datasets available.

You can browse GEE's entire data catalog here: **GEE Data Catalog**

Common GEE Datasets

Some of the key datasets available in Google Earth Engine (GEE) include:

- **SRTM Digital Elevation Data** View Dataset
- **ESA World Cover Map** View Dataset
- **Sentinel-2 Multispectral Data** View Dataset
- **Landsat-9** View Dataset

When selecting **Landsat 9 Collection 2** data, you can view details such as availability, data provider, and a sample snippet.

Further down the page, you'll find:

- **Description** of the dataset
- **Bands** and how they can be accessed
- **Properties** (metadata)
- **Terms of Use**

At the bottom, sample code is available for accessing the data using **JavaScript** and **Python** (if applicable).

ImageCollection Object

An **ImageCollection** is a time series or stack of images. You can load it using an Earth Engine collection ID or create one using:

- `ee.ImageCollection()`
- `ee.ImageCollection.fromImages()` (from a list of images)

You can also merge existing collections to create new ones.

Getting Information from an ImageCollection

ImageCollections information can be printed to the console, but the output is limited to 5,000 elements. Below is an example of retrieving ImageCollection details programmatically.

```
# Load the Landsat 9 Collection 2 TOA dataset
####### COMPLETE ME! GO TO GEE DATA CATALOG AND FIND THE COLLECTION ID FOR SENTINEL-2 SURFACE REFLECTANCE DATASE
collection = ee.ImageCollection("LANDSAT/LC09/C02/T2_TOA")

# Get the total number of images in the collection
total_count = collection.size().getInfo()
print(f"Total images in collection: {total_count}")

# Filter the collection for a specific date range
####### COMPLETE ME! PASS IN A DATE RANGE FOR START AND END DATE, IN YYYY-MM-DD FORMAT
filtered_collection = collection.filterDate('2022-01-01', '2022-02-01')

# Get the number of images in the filtered collection
filtered_count = filtered_collection.size().getInfo()
print(f"Images between 2022-01-01, 2022-02-01 of the earth: {filtered_count}")

# Filter for a single day to see how many scenes were captured globally
single_day_collection = collection.filterDate('2022-01-01', '2022-01-02')

# Get the count of images for that day
single_day_count = single_day_collection.size().getInfo()
print(f"Images captured on 2022-01-01 of the earth: {single_day_count}")
```

```
Total images in collection: 424512
Images between 2022-01-01, 2022-02-01 of the earth: 11276
Images captured on 2022-01-01 of the earth: 365
```

.getInfo() converts an Earth Engine object (e.g., ee.Number, ee.Image, ee.Feature, ee.Dictionary) into a Python-native object (e.g., int, dict, list). It retrieves the object's value from the Earth Engine server, making it accessible in Python.

Viewing Available Metadata

Each image in an ImageCollection contains metadata (properties) that can be accessed. To check the metadata of an image, we can print its properties:

```
# Grabbing the first image of the collection
first_image = collection.first()

# Print the metadata (properties) of the image
print(first_image.propertyNames().getInfo())

# Could also do. Works the same way
print(first_image.getInfo())
```

```
['system:version', 'system:id', 'RADIANCE_MULT_BAND_5', 'RADIANCE_MULT_BAND_6', 'RADIANCE_MULT_BAND_3', 'RADIANCE
{'type': 'Image', 'bands': [{'id': 'B1', 'data_type': {'type': 'PixelType', 'precision': 'float'}, 'dimensions':
```

Viewing property keys in GEE website

You can also view property keys in the GEE Data Catalog under the "Image Properties" section of the dataset page in GEE. example: https://developers.google.com/earth-engine/datasets/catalog/LANDSAT_LC09_C02_T1_TOA#image-properties

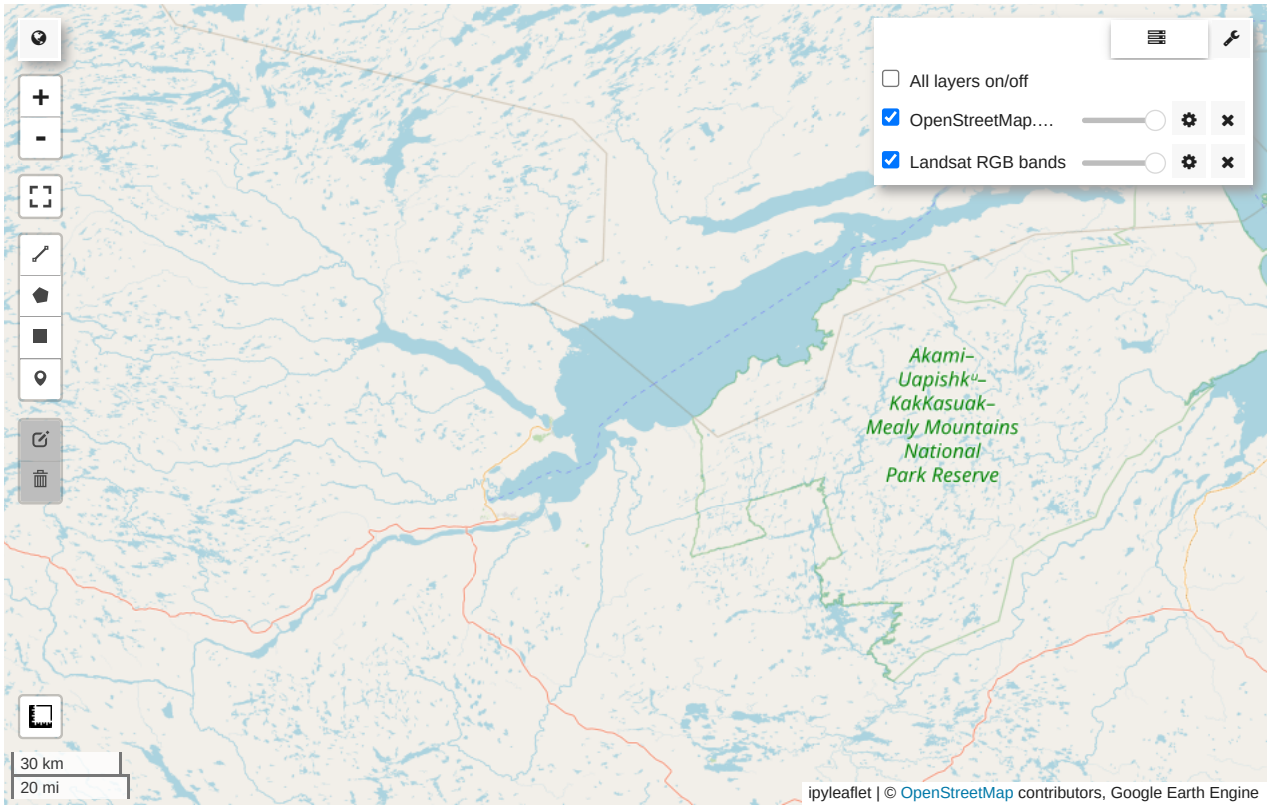
Visualizing an Image Using Geemap

Now lets pick a range date and a speicifc bounds and use **Geemap**, a Python package for interactive visualization of Earth Engine datasets, to display a sample image.

```
!pip install geemap
import geemap
```

```
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from ipywi
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2>=2.9->fo
Requirement already satisfied: google-cloud-core<3.0dev,>=2.3.0 in /usr/local/lib/python3.12/dist-packages (fro
Requirement already satisfied: google-resumable-media>=2.7.2 in /usr/local/lib/python3.12/dist-packages (from g
Requirement already satisfied: google-crc32c<2.0dev,>=1.0 in /usr/local/lib/python3.12/dist-packages (from goog
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from reques
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->earthengi
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->ea
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->ea
Requirement already satisfied: decorator in /usr/local/lib/python3.12/dist-packages (from ratelim->geocoder->ge
Requirement already satisfied: googleapis-common-protos<2.0.0,>=1.56.2 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<
Requirement already satisfied: proto-plus<2.0.0,>=1.22.3 in /usr/local/lib/python3.12/dist-packages (from google
Requirement already satisfied: debugpy>=1.0 in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.5.1->i
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.12/dist-packages (from ipykernel
Requirement already satisfied: matplotlib-inline>=0.1 in /usr/local/lib/python3.12/dist-packages (from ipykernel
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.5.1->i
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.5.1->ipywidg
Requirement already satisfied: pyzmq>=17 in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.5.1->ipyw
Requirement already satisfied: tornado>=6.1 in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.5.1->i
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.12/dist-packages (from ipython>=4.0.0
Collecting jedi>=0.16 (from ipython>=4.0.0->ipywidgets->ipyfilechooser>=0.6.0->geemap)
  Downloading jedi-0.19.2-py2.py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.12/dist-packages (from ipython>=4.0.0->ipyw
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: pygments in /usr/local/lib/python3.12/dist-packages (from ipython>=4.0.0->ipywidg
Requirement already satisfied: backcall in /usr/local/lib/python3.12/dist-packages (from ipython>=4.0.0->ipywidg
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.12/dist-packages (from ipython>=4.0.0->ipyw
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.12/dist-packages (from pyasn1-modu
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.12/dist-packages (from widgetsnbextensi
Requirement already satisfied: parso<0.9.0,>=0.8.4 in /usr/local/lib/python3.12/dist-packages (from jedi>=0.16->
Requirement already satisfied: entrypoints in /usr/local/lib/python3.12/dist-packages (from jupyter-client>=6.1
Requirement already satisfied: jupyter-core>=4.9.2 in /usr/local/lib/python3.12/dist-packages (from jupyter-clie
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.12/dist-packages (from notebook>=4.4.1->wid
Requirement already satisfied: nbformat in /usr/local/lib/python3.12/dist-packages (from notebook>=4.4.1->widget
Requirement already satisfied: nbconvert>=5 in /usr/local/lib/python3.12/dist-packages (from notebook>=4.4.1->wi
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.12/dist-packages (from notebook>=4.4
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.12/dist-packages (from notebook>=4.4.1
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.12/dist-packages (from notebook>=4.4
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.12/dist-packages (from notebook>=4.4.1
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.12/dist-packages (from pexpect>4.3->ipy
Requirement already satisfied: wcwidth in /usr/local/lib/python3.12/dist-packages (from prompt-toolkit!=3.0.0,!=
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.12/dist-packages (from jupyter-core>=
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python3.12/dist-packages (from nbclassic>=
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.12/dist-packages (from nbconvert>=5->not
Requirement already satisfied: bleach!=5.0.0 in /usr/local/lib/python3.12/dist-packages (from bleach[css]!=5.0.0
Requirement already satisfied: defusedxml in /usr/local/lib/python3.12/dist-packages (from nbconvert>=5->notebo
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.12/dist-packages (from nbconvert>=5
Requirement already satisfied: mistune<4,>=2.0.3 in /usr/local/lib/python3.12/dist-packages (from nbconvert>=5->
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.12/dist-packages (from nbconvert>=5->no
```

```
my_map = geemap.Map()

####### COMPLETE ME! PASS IN LANDSAT 9 IMAGE COLLECITON OF OF LEVEL 1 TOP OF ATMOSPHERE, THE DATE RANGE OF THE 1
image = ee.ImageCollection("LANDSAT/LC09/C02/T1_TOA").filterDate('2022-12-02', '2022-12-05').first() # We are gr

my_map.addLayer(image, {"bands": ["B4", "B3", "B2"]}, "Landsat RGB bands")
my_map
```



Filtering Using Metadata

In addition to filtering by date, we can filter an **ImageCollection** based on specific metadata properties. Earth Engine provides several filtering options, including:

- **Equal to (`ee.Filter.eq`)**
- **Less than (`ee.Filter.lt`)**
- **Greater than (`ee.Filter.gt`)**
- **Less than or equal to (`ee.Filter.lte`)**
- **Greater than or equal to (`ee.Filter.gte`)**

```
# Filter images with cloud cover less than 10%
cloud_lt_10 = collection.filter(ee.Filter.lt('CLOUD_COVER', 20)) # CLOUD_COVER IS A PROPERTY IN LANDAT 9 COLLECT

count = cloud_lt_10.size().getInfo() # WE USE GET INFO TO GET THE VALUE FROM THE SERVER
print(count)
```

```
132711
```

Sorting an ImageCollection

In Earth Engine, we can **sort** an ImageCollection based on metadata properties using `.sort()`.

```
# Sort the filtered collection by acquisition date (oldest to newest)
sorted_by_date = collection.sort('system:time_start')

# setting ascending=False, to get descending (big to small values)
sorted_by_cloud_desc = collection.sort('CLOUD_COVER', False)
first_image = sorted_by_cloud_desc.first()

####### COMPLETE ME!
# your geemap code for displaying the first image in the sorted collection with the highest cloud cover, in desc
Map = geemap.Map()
Map.addLayer(first_image, {"bands": ["B4", "B3", "B2"]}, "Most Cloudy Landsat RGB Image")
Map
```

Filtering Images for a Specific Bounding Box

To extract all images within a specific region, we use a **bounding box** defined by **longitude and latitude in Google Earth Engine (GEE). (NOT LAT, LON)**

In GEE, we can create a bounding box using a **polygon**. A polygon is a vector geometry that defines an area of interest. We will talk more about other types of vector data in a bit.

Creating a Bounding Box

We define a polygon using the `ee.Geometry.Polygon()` function by specifying corner coordinates **(longitude, latitude)** and `.filterBounds(region)` command

```
# Define a bounding box (longitude, latitude)
# COMPLETE ME: GO TO GOOGLE EARTH WEBSITE, FLY TO THE AN INTERESTING AREA OF YOUR CHOICE, RIGHT CLICK ON THE MAP
region = ee.Geometry.Polygon([
    [83.949, 28.211],  # Bottom-left: (longitude, latitude)
     [83.937, 28.218],  # Top-left: (longitude, latitude)
     [83.939, 28.223], # Top-right: (longitude, latitude)
     [83.953, 28.217] # Bottom-right: (longitude, latitude)
    ])

# Filter the Landsat 9 collection for this region
collection = ee.ImageCollection('LANDSAT/LC09/C02/T1_L2') \
    .filterBounds(region)

# COMPLETE ME! filter for the month of july in 2024
collection = collection.filterDate("2024-07-01", "2024-07-30")

# Get the number of images in the filtered collection
count = collection.size().getInfo()
print(f"Images in the specified region and date range: {count}")

####### COMPLETE ME!
# your geemap code for displaying the first image in this collection
first_image = collection.first()
Map = geemap.Map()
Map.addLayer(first_image, {"bands": ["SR_B4", "SR_B3", "SR_B2"]}, "RGB Image")
Map
```

```
Images in the specified region and date range: 3
```

Iterating Through an ImageCollection in Google Earth Engine

`ImageCollection` does not support direct iteration in Python. To process images one by one, we convert the `ImageCollection` into a list using `.toList()`, which allows access to individual images sequentially.

Steps to Iterate Through an ImageCollection:

1. **Filter the ImageCollection**: Use `.filterDate()` and `.filterBounds()` to define a specific subset of images.
2. **Convert to a List**: Use `.toList(image_collection.size())` to create a list of images.
3. **Iterate Over the List**: Use a loop to extract each image, get its date, and optionally visualize it.

```
collection = ee.ImageCollection('LANDSAT/LC09/C02/T1_TOA') \
    .filterDate('2022-03-01', '2022-04-03') \
    .filterBounds(ee.Geometry.Polygon([
        [[-120.0, 35.0],  # Bottom-left
         [-120.0, 37.0],  # Top-left
         [-118.0, 37.0],  # Top-right
         [-118.0, 35.0]]  # Bottom-right
    ]))

# Converting the collection to an ee.List (not a Python list) and determining its size
# You can't have it as a python list because python list is local and EE is on server.
image_list = collection.toList(collection.size())

for index in range(image_list.size().getInfo()):  # Get size in Python
    image = ee.Image(image_list.get(index))  # Extract image from list

    # grab the image cloud cover
    cc = image.get("CLOUD_COVER").getInfo()  # Get cloud cover property

    # COMPLETE ME: GRAB THE Product Id property: you can see the exact property name you need to search from her
    id = image.get('LANDSAT_PRODUCT_ID').getInfo() # 'LXSS_LLLL_PPPRRR_YYYYMMDD_yyyymmdd_CC_TX', "COMPLETE ME: G

    # printing the cloud cover and property id pair
    print(f"Image {index + 1}: ID = {id}, Cloud Cover = {cc}%")
```

```
Image 1: ID = LC09_L1TP_041034_20220311_20230425_02_T1, Cloud Cover = 3.18%
Image 2: ID = LC09_L1TP_041034_20220327_20230423_02_T1, Cloud Cover = 56.13%
Image 3: ID = LC09_L1TP_041035_20220311_20230425_02_T1, Cloud Cover = 2.89%
Image 4: ID = LC09_L1TP_041035_20220327_20230423_02_T1, Cloud Cover = 27.4%
Image 5: ID = LC09_L1TP_041036_20220311_20230425_02_T1, Cloud Cover = 0.24%
Image 6: ID = LC09_L1TP_041036_20220327_20230423_02_T1, Cloud Cover = 29.22%
Image 7: ID = LC09_L1TP_042034_20220302_20230426_02_T1, Cloud Cover = 40.37%
Image 8: ID = LC09_L1TP_042035_20220302_20230426_02_T1, Cloud Cover = 9.66%
Image 9: ID = LC09_L1TP_042036_20220302_20230426_02_T1, Cloud Cover = 0.08%
Image 10: ID = LC09_L1TP_043034_20220309_20230425_02_T1, Cloud Cover = 16.24%
Image 11: ID = LC09_L1TP_043034_20220325_20230424_02_T1, Cloud Cover = 2.08%
Image 12: ID = LC09_L1TP_043035_20220309_20230425_02_T1, Cloud Cover = 0.05%
Image 13: ID = LC09_L1TP_043035_20220325_20230424_02_T1, Cloud Cover = 30.81%
Image 14: ID = LO09_L1TP_042034_20220318_20230424_02_T1, Cloud Cover = 10.88%
Image 15: ID = LO09_L1TP_042035_20220318_20230424_02_T1, Cloud Cover = 1.63%
Image 16: ID = LO09_L1TP_042036_20220318_20230424_02_T1, Cloud Cover = 0.06%
```

Image Object

An `ee.Image` represents a single image in Google Earth Engine, which contains multiple bands. It allows us to perform various operations such as:

- **Selecting Bands:** `.select()` – Extract specific bands from an image.
- **Viewing Projection Information:** `.projection()` – Check an image's coordinate reference system and projection details.
- **Reprojecting & Rescaling:** `.reproject()` **and** `.reduceResolution()` – Convert to different coordinate systems and change the scale (ground sampling distance).
- **Performing Operations Between Bands:** `.expression()` – Compute band-based mathematical operations.
- **Computing Normalized Difference:** `.normalizedDifference()` – Calculate NDVI or other indices.
- **Applying Kernels:** `.convolve()` **with** `ee.Kernel` – Perform edge detection or smoothing.
- **Exporting to NumPy:** `geemap.export_to_numpy()` – Convert image data to a NumPy array (limited pixels).
- **Downloading Images:** `geemap.download_ee_image()` – Download full images for local processing.

Retrieving the Projection of an Image in Google Earth Engine

What is a Projection? A **projection** defines how spatial data is represented on a flat surface. It includes:

- **Coordinate Reference System (CRS)** → Specifies the spatial reference (e.g., EPSG:4326 for WGS84).
- **Affine Transform** → Describes how pixel coordinates relate to geographic coordinates.
- **Scale (Ground Sampling Distance - GSD)** → Defines the pixel resolution in meters.

Understanding the Output • CRS (EPSG:32628) → The spatial reference system (UTM Zone 28N, WGS84). • Transform: [30, 0, 340185, 0, -30, 8808615] • 30 → Pixel width (X resolution in meters) • 0 → Shear in the X direction (0 means no skew) • 340185 → Upper-left corner X coordinate (Easting) • 0 → Shear in the Y direction • -30 → Pixel height (negative means north-up) • 8808615 → Upper-left corner Y coordinate (Northing)

```
### SELECTING A BAND AND GETTING PROJECTION
# selecting a band and getting the projection
image = ee.Image('LANDSAT/LC08/C01/T1_SR/LC08_044034_20140318')

# Select the Red band: you can find the band names in GEE catalog: https://developers.google.com/earth-engine/data
red_band = image.select(['B4', 'B3', 'B2'])

# Get and print the projection information
projection = red_band.projection()

# the output represents the CRS (coordiante Reference system), transform which includes
# x_gds (pixel width), _ , upper left corner easting, _, - y_gsd (pixel height), upper left corner northing. All
print("X_GSD, _, UL Easting, _, Y_GSD, UL NORTHING", projection.getInfo())

### getting the native rsolution (scale) in meters
scale = image.projection().nominalScale()
print(scale)
```

Reprojection and Rescaling in Google Earth Engine

Reprojection and rescaling using **Nearest Neighbor interpolation** can be done directly with the `.reproject()` function.

If a **different resampling method** is required, it must be specified using `.resample()` **before** applying `.reproject()`.

✅ .reproject() alone defaults to Nearest Neighbor resampling. ✅ To use a different method, apply .resample() before .reproject(). ✅ Supported resampling methods: "nearest" (default), "bilinear", "bicubic".

```
# grabbing an image
image = ee.ImageCollection('LANDSAT/LC09/C02/T1_TOA').filterDate('2022-03-01', '2022-04-03').filterBounds(ee.Geo
    [[-120.0, 35.0],  # Bottom-left
     [-120.0, 37.0],  # Top-left
     [-118.0, 37.0],  # Top-right
     [-118.0, 35.0]]  # Bottom-right
])).first()

### REPROJECTION + RESAMPLING using Nearest Neighbour
# use Reproject to change CRS and scale USING NEAREST
# COMPLETE ME: SET THE PROJECTION SYSTEM TO GEODETIC LAT AND LONG COORDINATES (NEED EPSG CODE) AND DOWNSAMPLE TH
image_wgs = image.reproject(crs='EPSG:4326', scale=100)

### REPROJECTION + RESAMPLING using "bilinear" or "bicubic"
# if you want to resample to a specific method use resample before reproject
# USE BICUBIC RESAMPLING
image = image.resample("bicubic")
#  AND DOWNSAMPLE THE IMAGE TO 100 METERS BY PASSING THE CORRECT SCALE
image_wgs_bicubic = image.reproject(crs='EPSG:4326', scale=100)

# Compute the difference image
diff_image = image_wgs.select("B4").subtract(image_wgs_bicubic.select("B4"))

# Compute mean difference over the entire image using reduce: this will get us a number
mean_difference = diff_image.reduceRegion(
    reducer=ee.Reducer.mean(),
    geometry=image.geometry())

# Print the mean difference
print("The mean difference between Nearest Neighbor and Bicubic resampling is:", mean_difference.getInfo())
```

```
The mean difference between Nearest Neighbor and Bicubic resampling is: {'B4': -9.283578913669018e-06}
```

Using Expression to Calculate NDVI in Google Earth Engine

Using `.expression()` for Band Math In Google Earth Engine (GEE), `.expression()` allows defining formulas directly on image bands, similar to **Band Math in ENVI**. Make sure to rename the created image after.

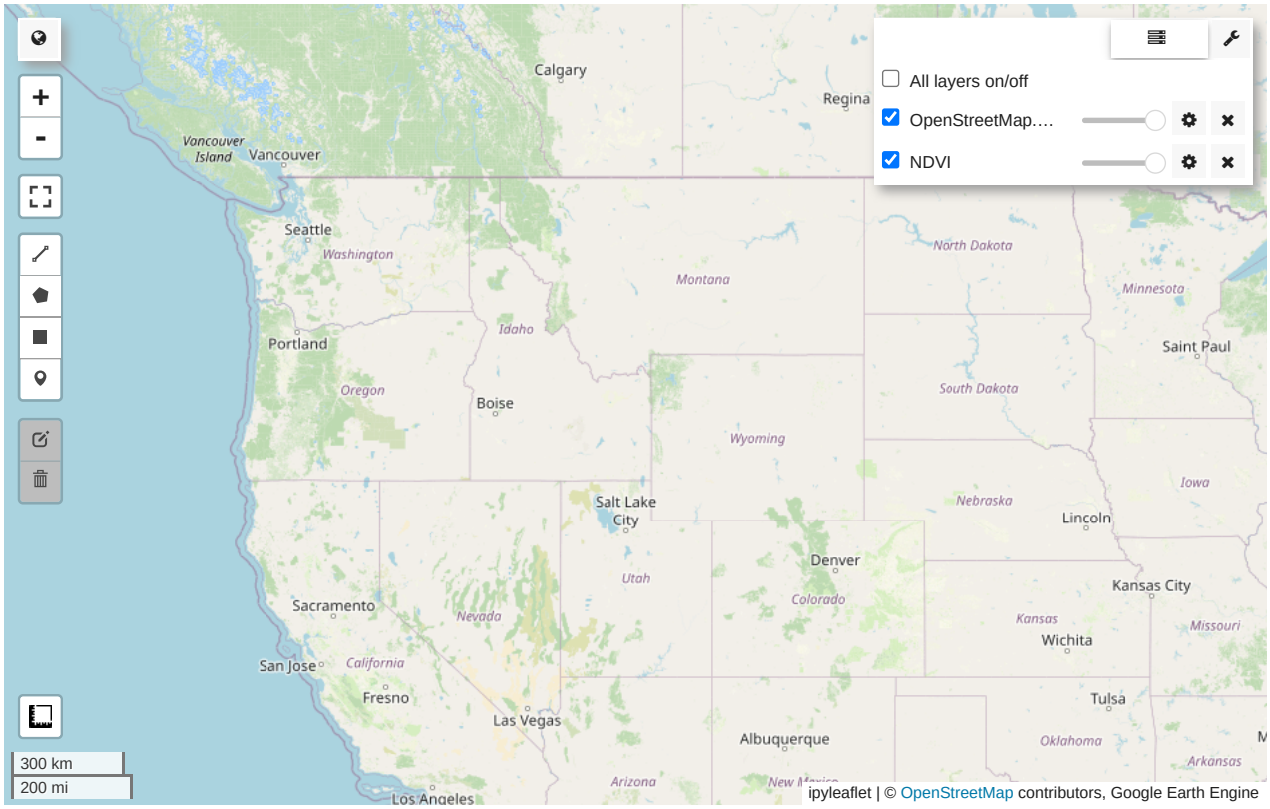
What is NDVI? Normalized Difference Vegetation Index (NDVI) is a commonly used index for analyzing vegetation health. It is calculated using the **Near-Infrared (NIR) and Red bands** of satellite imagery:

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

- **NIR (Near-Infrared, Band 5 in Landsat 9)** → Strongly reflected by healthy vegetation.
- **RED (Red, Band 4 in Landsat 9)** → Absorbed by chlorophyll in vegetation.

A higher NDVI value indicates **healthier vegetation**, while lower values suggest **water, bare soil, or unhealthy vegetation**.

```
# Load a Landsat 9 image
image = ee.ImageCollection('LANDSAT/LC09/C02/T1_TOA').filterDate('2022-03-01', '2022-04-03').filterBounds(ee.Geo
    [[-120.0, 35.0],  # Bottom-left
     [-120.0, 37.0],  # Top-left
     [-118.0, 37.0],  # Top-right
     [-118.0, 35.0]]  # Bottom-right
])).first()

# COMPLETE ME: CALCULATE NORMALIZED DIFFERENCE VEGETATION INDEX (NDVI) USING THE near infrared and the red band
ndvi = image.expression(
    '(NIR - RED) / (NIR + RED)',
    {
        'NIR': image.select("B5"),  # Near-Infrared Band
        'RED': image.select("B4")   # Red Band
    }
)

ndvi = ndvi.rename('NDVI')  # Rename the output band to 'NDVI'

# Create an interactive map
my_map = geemap.Map()

# Add the NDVI layer to the map
my_map.addLayer(ndvi, {"bands": ["NDVI"]}, "NDVI")

# display map
my_map
```



Downloading Tips using Geemap,

Data can be downloaded using `geemap.ee_to_numpy()` or `geemap.download_ee_image()`, each with its own advantages and limitations.

1. `ee_to_numpy()`
   - Converts the image to a **NumPy array**.
   - **Faster**, but has a **pixel limit**, making it unsuitable for large datasets.
2. `download_ee_image()`
   - Downloads the image **directly to the computer**.
   - Can handle **larger images**, but **slower** due to file transfer.
   - Requires a **separate library (e.g., `rasterio`)** to load the image after downloading, adding extra processing time.

```
import geemap
# Load a Landsat 9 image within a specific date range and geographic area
roi = ee.Geometry.Polygon([
        [[-120.0, 35.0],  # Bottom-left
         [-120.0, 35.1],  # Top-left
         [-119.9, 35.1],  # Top-right
         [-119.9, 35.0]]  # Bottom-right
    ])
landsat_image = ee.ImageCollection('LANDSAT/LC09/C02/T1_TOA') \
    .filterDate('2022-03-01', '2022-04-03') \
    .filterBounds(roi).first().select(["B4"])

# Export image as a NumPy array
# COMPLET ME: USE THE EE_TO_NUMPY FUNCTION TO EXPORT THE IMAGE AS A NUMPY ARRAY, PASS IN THE "region" argument f
array = geemap.ee_to_numpy(landsat_image, region=roi, bands=['B4'])

## Print the shape of the NumPy array
print("the shape of the numpy array is:")
print(array.shape)
```

```
the shape of the numpy array is:
(379, 316, 1)
```

Geometries and Features

Google Earth Engine supports various geometry types for spatial analysis:

- **Point** → A single coordinate in a specified projection.

- **LineString** → A sequence of connected points forming a line.
- **LinearRing** → A closed **LineString**, where the start and end points are the same.
- **Polygon** → A list of **LinearRings**, where:
  - The **first ring** defines the **outer boundary (shell)**.
  - Subsequent rings define **inner holes**.

    These geometry types enable **spatial operations** such as buffering, intersections, and spatial filtering in Earth Engine.

```
# Define a Point geometry (longitude, latitude)
point = ee.Geometry.Point([1.5, 1.5])  # Single coordinate location

# Define a LineString (a sequence of connected points forming a line)
lineString = ee.Geometry.LineString([
    [-35, -10],  # Start point (longitude, latitude)
    [35, -10],   # Second point
    [35, 10],    # Third point
    [-35, 10]    # End point
])

# Define a LinearRing (a closed LineString where the start and end points are the same)
linearRing = ee.Geometry.LinearRing([
    [-35, -10],  # Start point
    [35, -10],   # Second point
    [35, 10],    # Third point
    [-35, 10],   # Fourth point
    [-35, -10]   # Closing the ring (same as the start point)
])

# Define a Rectangle (bounding box using min long, min lat, max long, max lat)
rectangle = ee.Geometry.Rectangle([-40, -20, 40, 20])  # Defines a rectangular bounding box

# Define a Polygon (a closed shape with multiple points)
polygon = ee.Geometry.Polygon(  [
    [ [100.0, 0.0],
      [103.0, 0.0],
      [103.0, 3.0],
      [100.0, 3.0],
      [100.0, 0.0]
    ],
    [
      [101.0, 1.0],
      [102.0, 2.0],
      [102.0, 1.0]
    ]
  ])

# Print the geometries to verify
print("Point:", point.getInfo())
print("LineString:", lineString.getInfo())
print("LinearRing:", linearRing.getInfo())
print("Rectangle:", rectangle.getInfo())
print("Polygon:", polygon.getInfo())


# lets visualize them too
my_map = geemap.Map()
my_map.addLayer(point, {'color': 'red'}, "Point")
my_map.addLayer(lineString, {'color': 'blue'}, "LineString") # Add LineString to the map and assign color and na
my_map.addLayer(linearRing, {'color': 'green'}, "LinearRing") # Add LinearRing to the map and assign color and n
my_map.addLayer(rectangle, {'color': 'orange'}, "Rectangle") # Add Rectangle to the map and assign color and nam
my_map.addLayer(polygon, {'color': 'purple'}, "Polygon") # Add Polygon to the map and assign color and name

# Display the map
my_map
```

```
Point: {'type': 'Point', 'coordinates': [1.5, 1.5]}
LineString: {'type': 'LineString', 'coordinates': [[-35, -10], [35, -10], [35, 10], [-35, 10]]}
LinearRing: {'type': 'LinearRing', 'coordinates': [[-35, -10], [35, -10], [35, 10], [-35, 10], [-35, -10]]}
Rectangle: {'type': 'Polygon', 'coordinates': [[[-40, -20], [40, -20], [40, 20], [-40, 20], [-40, -20]]]}
Polygon: {'type': 'Polygon', 'coordinates': [[[100, 0], [103, 0], [103, 3], [100, 3], [100, 0]], [[101, 1], [102,
```

Extracting Information from Geometries in Google Earth Engine

Once a geometry is defined in Earth Engine, various properties can be retrieved, such as:

- **Area (`.area()`)** → Computes the area of a polygon in square meters.
- **Perimeter (`.length()`)** → Computes the perimeter of a polygon or the length of a LineString.
- **Convert to GeoJSON (`.toGeoJSON()`)** → Exports the geometry in GeoJSON format.
- **Print Coordinates (`.coordinates()`)** → Retrieves the list of coordinates that define the geometry.

```
print('Polygon printout: ', polygon.getInfo())

# Print polygon area in square kilometers.
print('Polygon area: ', polygon.area().divide(1000 * 1000).getInfo())

# Print polygon perimeter length in kilometers.
print('Polygon perimeter: ', polygon.perimeter().divide(1000).getInfo())

# Print the coordinates as lists
print('Polygon coordinates: ', polygon.coordinates().getInfo())
```

```
Polygon printout:  {'type': 'Polygon', 'coordinates': [[[100, 0], [103, 0], [103, 3], [100, 3], [100, 0]], [[101,
Polygon area:  105072.02394942634
Polygon perimeter:  1709.6573628964913
Polygon coordinates:  [[[100, 0], [103, 0], [103, 3], [100, 3], [100, 0]], [[101, 1], [102, 2], [102, 1], [101, 1
```

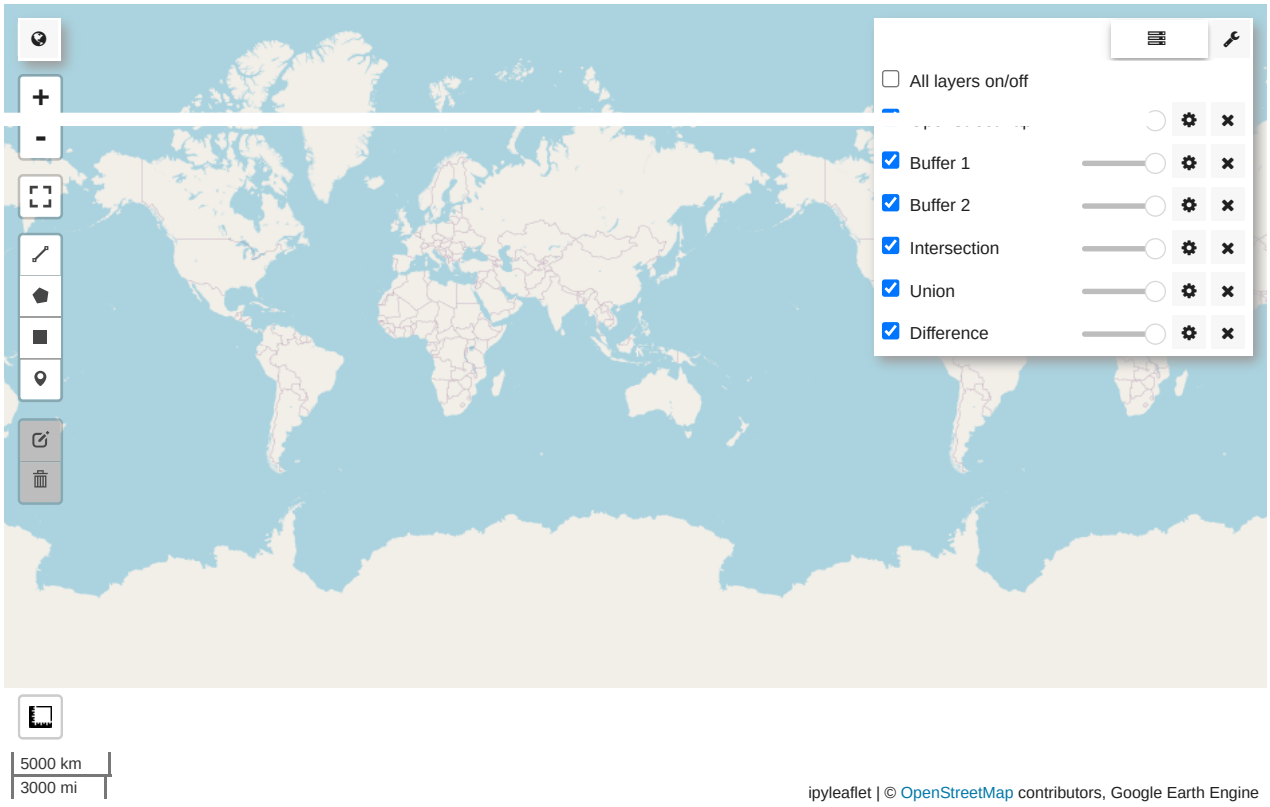
Geometric Operations in Google Earth Engine

Google Earth Engine allows performing various **geometric operations** on features, such as:

- **Buffering (`.buffer()`)** → Expands a point, line, or polygon by a specified distance.
- **Intersection (`.intersection()`)** → Computes the overlapping area between two geometries.
- **Union (`.union()`)** → Merges two geometries into a single shape.
- **Difference (`.difference()`)** → Subtracts one geometry from another.

```
import geemap
# Create a map
my_map = geemap.Map()

# Define two circular geometries (buffers around points)
```

```
poly1 = ee.Geometry.Point([-50, 30]).buffer(1e6)  # 1 million meter buffer
poly2 = ee.Geometry.Point([-40, 30]).buffer(1e6)  # 1 million meter buffer

# Perform geometric operations
buffer1 = poly1  # Original buffer
buffer2 = poly2  # Original buffer

# Intersection of two geometries
### COMPLETE ME INTERSECT THE POLY1 WITH POLY2
intersection = poly1.intersection(poly2)


# Union (merge both geometries)
### COMPLETE ME UNION THE POLY1 WITH POLY2
union = poly1.union(poly2)

# Difference (part of poly1 that is not in poly2)
### COMPLETE ME DIFFEENCE THE POLY1 WITH POLY2
difference = poly1.difference(poly2)

# Add geometries to the map
my_map.addLayer(buffer1, {'color': 'red'}, "Buffer 1")
my_map.addLayer(buffer2, {'color': 'blue'}, "Buffer 2") # Add buffer2 to the map and assign color and name
my_map.addLayer(intersection, {'color': 'green'}, "Intersection") # Add intersection to the map and assign color
my_map.addLayer(union, {'color': 'orange'}, "Union") # Add union to the map and assign color and name
my_map.addLayer(difference, {'color': 'purple'}, "Difference") # Add difference to the map and assign color and

# Display the map
my_map
```



Converting a Geometry to a Feature in Google Earth Engine

In Google Earth Engine, you can **convert a geometry into a feature** by adding **attributes (properties)** to it. This means the geometry is no longer just spatial data—it now contains **additional information**.

Key Concept

- A **feature** is essentially a **dictionary** where the **geometry** is just one part of it.
- Features allow **storing metadata** such as species names, land cover types, or other relevant properties alongside the geometry.

```
from pprint import pprint

# Create a feature with a Point geometry and assign properties
feature = ee.Feature(ee.Geometry.Point([-120, 35])) \
    .set('genus', 'Sequoia') \
    .set('year identified', 1999)
```

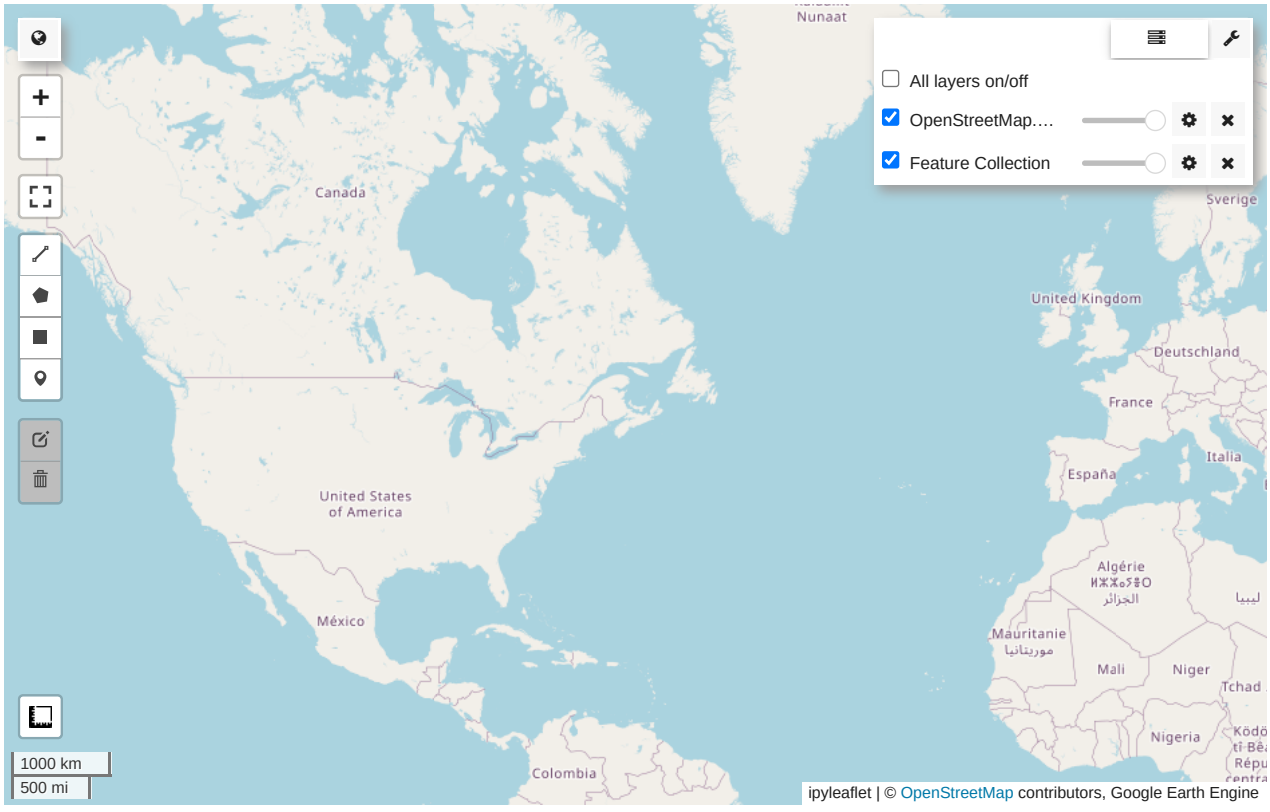
```
# Retrieve and print a property from the feature
species = feature.get('genus')
print(species.getInfo())  # Output: 'sempervirens'

# Add a new property to the feature called species
feature = feature.set('species', 1)

# Display the updated feature information
pprint(feature.getInfo())
```

```
Sequoia
{'geometry': {'coordinates': [-120, 35], 'type': 'Point'},
 'properties': {'genus': 'Sequoia', 'species': 1, 'year identified': 1999},
 'type': 'Feature'}
```

Creating a FeatureCollection in Google Earth Engine

In Google Earth Engine, a **FeatureCollection** is a collection of multiple **features**, where each feature consists of **a geometry and associated attributes**. This is useful for **grouping spatial data** and performing analysis on multiple features at once.

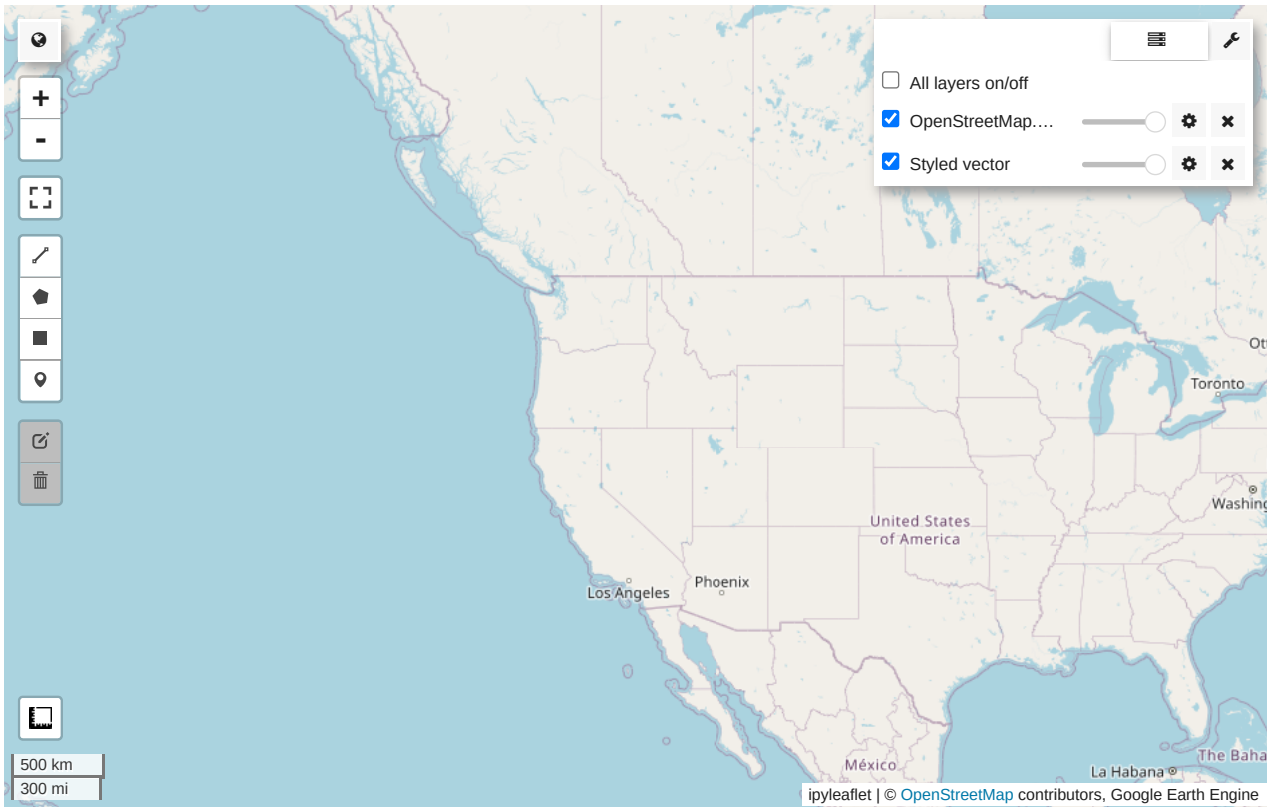
```
# Make a list of Features.
features = [
  ee.Feature(ee.Geometry.Rectangle(30.01, 59.80, 30.59, 60.15), {'name': 'Voronoi'}),
  ee.Feature(ee.Geometry.Point(-73.96, 40.781), {'name': 'Thiessen'}),
  ee.Feature(ee.Geometry.Point(6.4806, 50.8012), {'name': 'Dirichlet'})
]

# Create a FeatureCollection from the list and print it.
fromList = ee.FeatureCollection(features)
fromList.getInfo()

# DISPLAY THE FEATURE COLLECITON IN GEEMAP
# YOU SHOULD SEE POINTS IN GERMANY, NEW YORK AND RUSSIA
my_map = geemap.Map()

my_map.addLayer(fromList, {'color': 'red'}, 'Feature Collection')

my_map.setCenter(10, 50, 2)  # (longitude, latitude, zoom level)

my_map
```



Visualizing an Entire FeatureCollection in Google Earth Engine

You can also **visualize an entire FeatureCollection** on a map to display all its features at once. Below is an example of all the states in the CONUS as features and we are accessing this through the TIGER dataset.

```
Map = geemap.Map(center=[40, -100], zoom=4)
states = ee.FeatureCollection("TIGER/2018/States")
vis_params = {
    'color': '000000',
    'colorOpacity': 1,
    'pointSize': 3,
    'pointShape': 'circle',
    'width': 2,
    'lineType': 'solid',
    'fillColorOpacity': 0.66,
}
palette = ['006633', 'E5FFCC', '662A00', 'D8D8D8', 'F5F5F5']
Map.add_styled_vector(
    states, column="NAME", palette=palette, layer_name="Styled vector", **vis_params
)
Map
```



Reducers

Now that we have covered **image collections** and **individual images**, let's move on to **reducers**.

What Are Reducers? Reducers in Google Earth Engine are used to **aggregate data** across **time, space, bands, arrays, and other data structures**. They allow us to summarize large datasets into meaningful statistics.

Using the `ee.Reducer` Class The `ee.Reducer` class defines how data is aggregated. Reducers can compute **simple statistical summaries** such as:

- **Minimum, Maximum, Mean, Median, Standard Deviation**

Or **more complex summaries**, including:

- **Histograms, Linear Regression, Lists of values**

Reducers can be applied in different ways depending on what data needs to be aggregated:

Reduction Type	Method Used	Description
Over Time	`imageCollection.reduce()`	Aggregates images over a time series (e.g., mean of NDVI over time).
Over Space	`image.reduceRegion()`	Aggregates values over a specific region (e.g., mean temperature in a polygon).
Neighborhoods	`image.reduceNeighborhood()`	Aggregates values within a moving window (e.g., smoothing with local mean).
Across Bands	`image.reduce()`	Aggregates multiple spectral bands within a single image (e.g., mean of selected bands).

Statistical Reducers in Google Earth Engine

Statistical reducers in Google Earth Engine allow you to compute **summary statistics** over images, image collections, or regions. These reducers can be **combined** using the `.combine()` function, enabling multiple statistical calculations in a single operation.

Common Statistical Reducers Google Earth Engine provides several built-in reducers, including:

- `ee.Reducer.mean()` → Computes the mean (average) value.
- `ee.Reducer.median()` → Computes the median value.
- `ee.Reducer.min()` → Finds the minimum value.
- `ee.Reducer.max()` → Finds the maximum value.
- `ee.Reducer.stdDev()` → Computes the standard deviation.
- `ee.Reducer.sum()` → Computes the sum of values.
- `ee.Reducer.count()` → Counts the number of non-null values.
- `ee.Reducer.histogram()` → Creates a histogram of values.
- `ee.Reducer.percentile([percentiles])` → Computes specific percentiles.

Multiple reducers can be combined using `.combine()`.

```
from pprint import pprint

# Load a Landsat 9 image within a specific date range and geographic area
landsat_image = ee.ImageCollection('LANDSAT/LC09/C02/T1_TOA') \
    .filterDate('2022-03-01', '2022-04-03') \
    .filterBounds(ee.Geometry.Polygon([
    [[-120.0, 35.0],  # Bottom-left
     [-120.0, 37.0],  # Top-left
     [-118.0, 37.0],  # Top-right
     [-118.0, 35.0]]  # Bottom-right
])).first()

# Define a reducer that calculates both mean and standard deviation
combined_reducer = ee.Reducer.mean().combine(
    reducer2=ee.Reducer.stdDev(),
    sharedInputs=True
)

# Apply the reducer to compute statistical summaries for each band
image_statistics = landsat_image.reduceRegion(
    reducer=combined_reducer,
    bestEffort=True,
)

# Print the computed mean and standard deviation values
pprint(image_statistics.getInfo())
```

```
{'B10_mean': 290.213501706812,
 'B10_stdDev': 7.262484084904942,
 'B11_mean': 290.5839054535367,
 'B11_stdDev': 7.319421165423086,
 'B1_mean': 0.19007203841034984,
 'B1_stdDev': 0.09218701230877821,
 'B2_mean': 0.1839798329626056,
 'B2_stdDev': 0.09677390372457362,
 'B3_mean': 0.18399505891234447,
 'B3_stdDev': 0.09562728618135112,
 'B4_mean': 0.20962732341314796,
 'B4_stdDev': 0.1006840701914223,
 'B5_mean': 0.25362516345494746,
 'B5_stdDev': 0.09934639505389671,
 'B6_mean': 0.24337163699448405,
 'B6_stdDev': 0.0812573836551734,
 'B7_mean': 0.20926099989695715,
 'B7_stdDev': 0.07218661204055662,
 'B8_mean': 0.1902488331244534,
 'B8_stdDev': 0.09591643066503616,
 'B9_mean': 0.006106519682822471,
 'B9_stdDev': 0.002860328481271117,
 'QA_PIXEL_mean': 22208.145911827036,
 'QA_PIXEL_stdDev': 1435.098625766076,
 'QA_RADSAT_mean': 0,
 'QA_RADSAT_stdDev': 0,
 'SAA_mean': 14719.988291539583,
 'SAA_stdDev': 80.23799674650802,
 'SZA_mean': 4624.894700123634,
 'SZA_stdDev': 51.31518169626332,
 'VAA_mean': 817.2725174958596,
 'VAA_stdDev': 9637.115156217953,
 'VZA_mean': 436.0808367702174,
 'VZA_stdDev': 240.45955864400003}
```

Reducing an ImageCollection Using Reducers

To compute statistics over a time series of images in an **ImageCollection**, use `.reduce()`, which collapses the collection into a single image by applying a specified reducer **pixel-wise**. Each pixel in the output represents the computed statistic across all images

at that location.

Example Usage The following code demonstrates how to:

- Compute the **median** of an image collection.
- Use different reducers (**mean, sum, variance, percentiles**).
- Apply shortcut methods for basic statistics (**min, max, mean**).

```
# Load an ImageCollection (example: Landsat 9)
imageCollection = ee.ImageCollection('LANDSAT/LC09/C02/T1_TOA') \
    .filterDate('2022-03-01', '2022-04-03') \
    .filterBounds(ee.Geometry.Polygon([
        [[-120.0, 35.0], [-120.0, 37.0], [-118.0, 37.0], [-118.0, 35.0]]
    ]))

# Reduce the collection to a single image using different reducers
median_image = imageCollection.reduce(ee.Reducer.median())  # Compute median
mean_image = imageCollection.reduce(ee.Reducer.mean())  # Compute mean
```