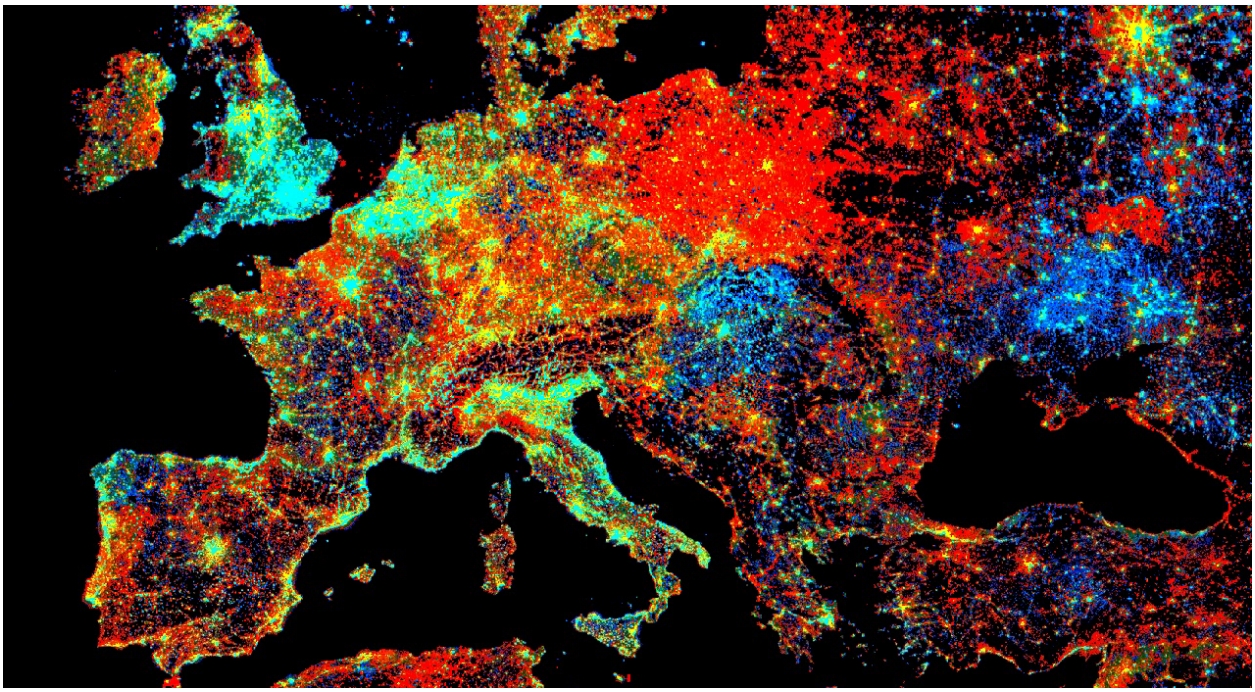
Cloud-Based Remote Sensing with Google Earth Engine



Fundamentals and Applications

or, click here to get back to the master document to access different sections)

Part A1: Human Applications

This Part covers some of the many Human Applications of Earth Engine. It includes demonstrations of how Earth Engine can be used in agricultural and urban settings, including for sensing the built environment and the effects it has on air composition and temperature. This Part also covers the complex topics of human health, illicit deforestation activity, and humanitarian actions.

Chapter A1.1: Agricultural Environments

Authors
Sherrie Wang and George Azzari

Overview

The purpose of this chapter is to introduce how datasets and functions available in Earth Engine can be used to map agriculture at scale. We will walk through an example of mapping crop types in the US Midwest, which is one of the main breadbaskets of the world. The skills learned in this chapter will equip you to go on to map other agricultural characteristics, such as yields and management practices.

Learning Outcomes

- Classifying crop type in a county in the US Midwest using the Cropland Data Layer (CDL) as labels.
- Using `ee.Reducer.linearRegression` to fit a second-order harmonic regression to a crop time series and extract harmonic coefficients.
- Using the Green Chlorophyll Vegetation Index (GCVI) for crop type classification.
- Training a random forest to classify crop type from harmonic coefficients.
- Applying the trained random forest classifier to the study region and assessing its performance.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Create a graph using `ui.Chart` (Chap. F1.3).
- Perform basic image analysis: select bands, compute indices, create masks, classify images (Part F2).
- Obtain accuracy metrics from classifications (Chap. F2.2)
- Recognize similarities and differences among satellite spectral bands (Part F1, Part F2, Part F3).
- Write a function and `map` it over an `ImageCollection` (Chap. F4.0).
- Mask cloud, cloud shadow, snow/ice, and other undesired pixels (Chap. F4.3).
- Fit linear and nonlinear functions with regression in an `ImageCollection` time series (Chap. F4.6).

- Filter a `FeatureCollection` to obtain a subset (Chap. F5.0, Chap. F5.1).

Introduction to Theory

Agriculture is one of the primary ways in which humans have altered the surface of our planet. Globally, about five billion hectares, or 38% of Earth's land surface, is devoted to agriculture (FAO 2020). About one-third of this is used to grow crops, while the other two-thirds are used to graze livestock.

In the face of a growing human population and changing climate, it is more important than ever to manage land resources effectively in order to grow enough food for all people while minimizing damage to the environment. Toward this end, remote sensing offers a critical source of data for monitoring agriculture. Since most agriculture occurs outdoors, sensors on satellites and airplanes can capture many crop characteristics. Research has shown that crops' spectral reflectance over time can be used to classify crop types (Wang et al. 2019), predict yield (Burke and Lobell 2017), detect crop stress (Ihuoma and Madramootoo 2017), identify irrigation (Deines et al. 2017), quantify species diversity (Duro et al. 2014), and pinpoint sowing and harvest dates (Jain et al. 2016). Remotely sensed imagery has also been key to the rise of precision agriculture, where management practices are varied at fine scales to respond to differences in crop needs within a field (Azzari et al. 2019, Jin et al. 2017, Seifert et al. 2018). Ultimately, the goal of measuring agricultural practices and outcomes is to improve yields and reduce environmental degradation.

Practicum

In this practicum, we will use Earth Engine to pull Landsat imagery and classify crop types in the US Midwest. The US is the world's largest producer of corn and second-largest producer of soybeans; therefore, maintaining high yields in the US is vital for global food security and price stability. Crop type mapping is an important prerequisite to using satellite imagery to estimate agricultural production. We will show how to obtain features from a time series by fitting a harmonic regression and extracting the coefficients. Then we will use a random forest to classify crop type, where the "ground truth" labels will come from the Cropland Data Layer (CDL) from the US Department of Agriculture. CDL is itself a product of a classifier using satellite imagery as input and survey-based ground truth as training labels (USDA 2021). We demo crop type classification in the US Midwest because CDL allows us to validate our predictions.

While using satellite imagery to map crop types is already operational in the US, mapping crop types remains an open challenge in much of the world.

Section 1. Pull All Landsat Imagery for the Study Area

In the Midwest, corn and soybeans dominate the landscape. We will map crop types in McLean County, Illinois. This is the county that produces the most corn and soybeans in the United States, at 62.9 and 19.3 million bushels, respectively. Let's start by defining the study area and visualizing it (Fig. A1.1.1). We obtain county boundaries from the United States Census Bureau's TIGER dataset, which can be found in the Earth Engine Data Catalog. We extract the McLean County feature from TIGER by name and by using Illinois' FIPS code of 17.

```
// Define study area.
var TIGER = ee.FeatureCollection('TIGER/2018/Counties');
var region = ee.Feature(TIGER
    .filter(ee.Filter.eq('STATEFP', '17'))
    .filter(ee.Filter.eq('NAME', 'McLean'))
    .first());
var geometry = region.geometry();
Map.centerObject(region);
Map.addLayer(region, {
    'color': 'red'
}, 'McLean County');
```

Fig. A1.1.1 Location and borders of McLean County, Illinois

Next, we import Landsat 7 and 8 imagery, as presented in Part 1. In particular, we use Level-2 data, which contains atmospherically corrected surface reflectance and land surface temperature. While top-of-atmosphere data will also yield good results for many agricultural applications, we prefer data that has been corrected for atmospheric conditions when available, since atmospheric variation is just one more source of noise in the inputs.

```
// Import Landsat imagery.
var landsat7 = ee.ImageCollection('LANDSAT/LE07/C02/T1_L2');
var landsat8 = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2');
```

We define a few functions that will make it easier to work with Landsat data. The first two functions, `renameL7` and `renameL8`, rename the bands of either a Landsat 7 or Landsat 8 image to more intuitive names. For example, instead of calling the first band of a Landsat 7 image `'B1'`, we rename it to `'BLUE'`, since band 1 captures light in the blue portion of the visible spectrum. Renaming also makes it easier to work with Landsat 7 and 8 images together, since the band numbering of their sensors (ETM+ and OLI) is different.

```
// Functions to rename Landsat 7 and 8 images.
function renameL7(img) {
    return img.rename(['BLUE', 'GREEN', 'RED', 'NIR', 'SWIR1',
        'SWIR2', 'TEMP1', 'ATMOS_OPACITY', 'QA_CLOUD',
        'ATRAN', 'CDIST',
        'DRAD', 'EMIS', 'EMSD', 'QA', 'TRAD', 'URAD',
        'QA_PIXEL',
        'QA_RADSAT'
    ]);
}

function renameL8(img) {
    return img.rename(['AEROS', 'BLUE', 'GREEN', 'RED', 'NIR',
        'SWIR1',
        'SWIR2', 'TEMP1', 'QA_AEROSOL', 'ATRAN', 'CDIST',
        'DRAD', 'EMIS',
```

```
        'EMSD', 'QA', 'TRAD', 'URAD', 'QA_PIXEL', 'QA_RADSAT'
    ]);
}
```

The Landsat 7 Level-2 product has seven surface reflectance and temperature bands, while the Landsat 8 Level-2 product has eight. Both have a number of other bands for image quality, atmospheric conditions, etc. For this chapter, we will mostly be concerned with the near-infrared (NIR), short-wave infrared (SWIR1 and SWIR2), and pixel quality (QA_PIXEL) bands. Many differences among vegetation types can be seen in the NIR, SWIR1, and SWIR2 bands of the electromagnetic spectrum. The pixel quality band is important for masking out clouds when working with optical imagery (Chap. F4.3). Below, we define two functions: the `addMask` function to turn the `QA_PIXEL` bitmask into multiple masking layers, and the `maskQAClear` function to remove all non-clear pixels from each image.

```
// Functions to mask out clouds, shadows, and other unwanted features.
function addMask(img) {
    // Bit 0: Fill
    // Bit 1: Dilated Cloud
    // Bit 2: Cirrus (high confidence) (L8) or unused (L7)
    // Bit 3: Cloud
    // Bit 4: Cloud Shadow
    // Bit 5: Snow
    // Bit 6: Clear
    //        0: Cloud or Dilated Cloud bits are set
    //        1: Cloud and Dilated Cloud bits are not set
    // Bit 7: Water
    var clear = img.select('QA_PIXEL').bitwiseAnd(64).neq(0);
    clear = clear.updateMask(clear).rename(['pxqa_clear']);

    var water = img.select('QA_PIXEL').bitwiseAnd(128).neq(0);
    water = water.updateMask(water).rename(['pxqa_water']);

    var cloud_shadow = img.select('QA_PIXEL').bitwiseAnd(16).neq(0);
    cloud_shadow = cloud_shadow.updateMask(cloud_shadow).rename([
        'pxqa_cloudshadow'
    ]);
```

```
    var snow = img.select('QA_PIXEL').bitwiseAnd(32).neq(0);
    snow = snow.updateMask(snow).rename(['pxqa_snow']);

    var masks = ee.Image.cat([
        clear, water, cloud_shadow, snow
    ]);

    return img.addBands(masks);
}

function maskQAClear(img) {
    return img.updateMask(img.select('pxqa_clear'));
}
```

In addition to the raw bands sensed by ETM+ and OLI, vegetation indices (VI) can also help distinguish different vegetation types. Prior work has found the Green Chlorophyll Vegetation Index (GCVI) particularly useful for distinguishing corn from soybeans in the Midwest (Wang et al. 2019). We will add it as a band to each Landsat image.

```
// Function to add GCVI as a band.
function addVIs(img){
  var gcvi = img.expression('(nir / green) - 1', {
      nir: img.select('NIR'),
      green: img.select('GREEN')
  }).select([0], ['GCVI']);

  return ee.Image.cat([img, gcvi]);
}
```

Now we are ready to pull Landsat 7 and 8 imagery for our study area and apply the above functions. We will access all images from January 1 to December 31, 2020, that intersect with McLean County. Because different crops have different growth characteristics, it is valuable to obtain a time series of images to distinguish when crops grow, senesce, and are harvested. Differences in spectral reflectance at individual points in time can also be informative.

```
// Define study time period.
var start_date = '2020-01-01';
var end_date = '2020-12-31';

// Pull Landsat 7 and 8 imagery over the study area between start and
end dates.
var landsat7coll = landsat7
    .filterBounds(geometry)
    .filterDate(start_date, end_date)
    .map(renameL7);

var landsat8coll = landsat8
    .filterDate(start_date, end_date)
    .filterBounds(geometry)
    .map(renameL8);
```

Next, we merge the Landsat 7 and Landsat 8 collections, mask out clouds, and add GCVI as a band.

```
// Merge Landsat 7 and 8 collections.
var landsat = landsat7coll.merge(landsat8coll)
    .sort('system:time_start');

// Mask out non-clear pixels, add VIs and time variables.
landsat = landsat.map(addMask)
    .map(maskQAClear)
    .map(addVIs);
```

We can use a `ui.Chart` object to visualize the combined Landsat GCVI time series at a particular point, which in this case falls in a corn field, according to CDL.

```
// Visualize GCVI time series at one location.
var point = ee.Geometry.Point([-88.81417685576481,
    40.579804398254005
]);
var landsatChart = ui.Chart.image.series(landsat.select('GCVI'),
```

```
        point)
    .setChartType('ScatterChart')
    .setOptions({
        title: 'Landsat GCVI time series',
        lineWidth: 1,
        pointSize: 3,
    });
print(landsatChart);
```

You should see the chart shown in Fig. A1.1.2 in the Earth Engine **Console**.

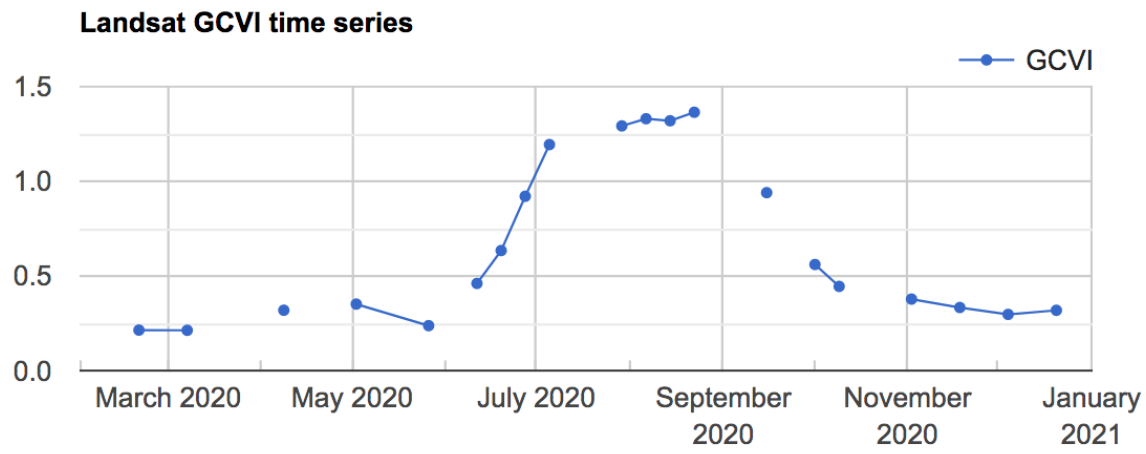


Fig. A1.1.2 The Landsat GCVI time series visualized at a corn point in McLean County, Illinois. A higher GCVI corresponds to greater leaf chlorophyll content; the crop can be seen greening up in June and senescing in September.

Finally, let's also take a look at the crop type dataset that we will be using to train and evaluate our classifier. We load the CDL image for 2020 and select the layer that contains crop type information.

```
// Get crop type dataset.
var cdl = ee.Image('USDA/NASS/CDL/2020').select(['cropland']);
Map.addLayer(cdl.clip(geometry), {}, 'CDL 2020');
```

Corn fields are visualized in yellow and soybean fields in green in CDL (Fig. A1.1.3).

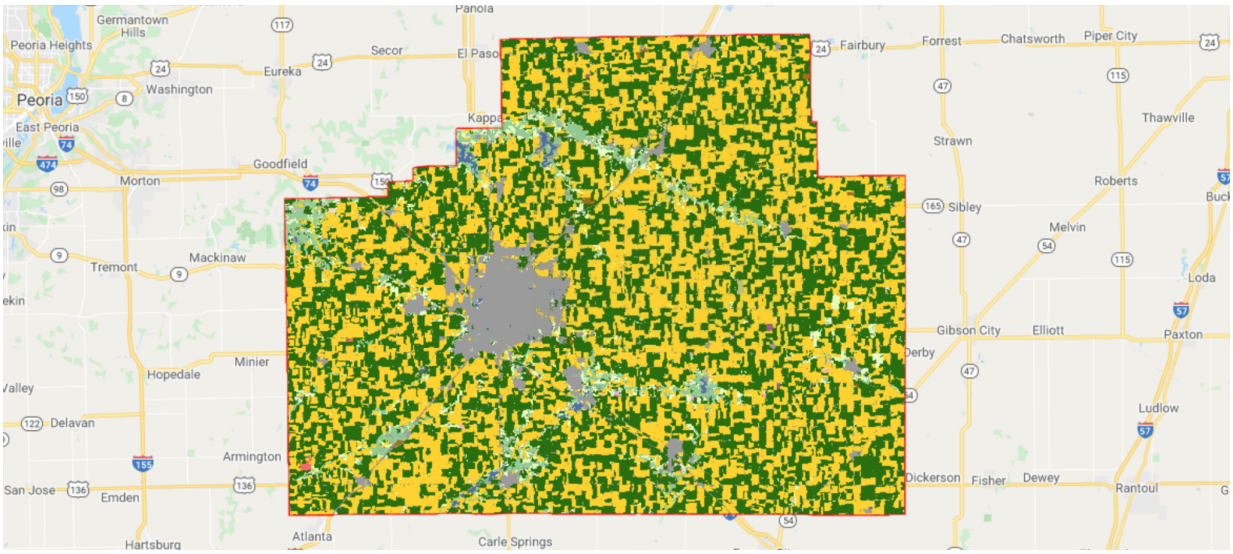


Fig. A1.1.3 The CDL visualized over McLean County, Illinois. Corn fields are shown in yellow, while soybean fields are shown in dark green. Other land cover types include urban areas in gray, deciduous forest in teal, and water in blue.

Code Checkpoint A11a. The book's repository contains a script that shows what your code should look like at this point.

Question 1. How many images were taken by Landsat 7 and Landsat 8 over McLean County in 2020?

Question 2. By using `ui.Chart`, can you qualitatively spot any difference between the time series of corn fields and soybean fields?

Section 2. Add Bands to Landsat Images for Harmonic Regression

Now that we have a collection that combines all available Landsat imagery over McLean County in 2020, we can extract features from the time series for crop type classification. Harmonic regression is one way to extract features by fitting sine and cosine functions to

a time series (Chap. F4.6). Also known as a Fourier transform, the harmonic regression is especially well suited for data patterns that reappear at regular intervals.

In particular, we fit a second-order harmonic regression, which takes the form:

$$f(t) = a_1 cos(2\pi\omega t) + b_1 sin(2\pi\omega t) + a_2 cos(4\pi\omega t) + b_2 sin(4\pi\omega t) + c \quad \text{(A1.1.1)}$$

Here, *t* is the time variable; *f(t)* is the observed variable to be fit; *a*, *b*, and *c* are coefficients found through regression; and ω is a hyperparameter that controls the periodicity of the harmonic basis. For this exercise, we will use ω = 1.

To prepare the collection, we define two functions that add a harmonic basis and intercept to the `ImageCollection`. The first function adds the acquisition time of each image as a band to the image in units of years. The function takes an image and a reference date as arguments; the new time band is computed relative to this reference date.

```
// Function that adds time band to an image.
function addTimeUnit(image, refdate) {
    var date = image.date();

    var dyear = date.difference(refdate, 'year');
    var t = image.select(0).multiply(0).add(dyear).select([0], ['t'])
        .float();

    var imageplus = image.addBands(t);

    return imageplus;
}
```

The second function calls `addTimeUnit`, then takes the sine and cosine of each image's acquisition time and adds them as new bands to each image. We are fitting a second-order harmonic regression, so we will add two sine and two cosine terms as bands, along with a constant term. The function also allows you to use any value of ω (omega). Note that, to create the `'constant'` band, we divide the time band by itself; this results in a band with value 1 and the same mask as the time band.

```
// Function that adds harmonic basis to an image.
function addHarmonics(image, omega, refdate) {
    image = addTimeUnit(image, refdate);
    var timeRadians = image.select('t').multiply(2 * Math.PI * omega);
    var timeRadians2 = image.select('t').multiply(4 * Math.PI *
    omega);

    return image
        .addBands(timeRadians.cos().rename('cos'))
        .addBands(timeRadians.sin().rename('sin'))
        .addBands(timeRadians2.cos().rename('cos2'))
        .addBands(timeRadians2.sin().rename('sin2'))
        .addBands(timeRadians.divide(timeRadians)
            .rename('constant'));
}
```

We have all the tools we need to add the sine and cosine terms to each image—we just use `map` to apply `addHarmonics` to each image in the collection. We use the `start_date` of January 1, 2020, as the reference date.

```
// Apply addHarmonics to Landsat image collection.
var omega = 1;
var landsatPlus = landsat.map(
    function(image) {
        return addHarmonics(image, omega, start_date);
    });
print('Landsat collection with harmonic basis: ', landsatPlus);
```

When we print the new `ImageCollection`, the **Console** tab should display a collection of images with bands that now include `cos`, `sin`, `cos2`, `sin2`, and `constant`—the terms we just added.

Code Checkpoint A11b. The book's repository contains a script that shows what your code should look like at this point.

Question 3. What assumption are we making about crop spectral reflectance when we fit a harmonic regression to a variable like GCVI using ω = 1? What types of time series might be better suited for a smaller value of ω? A larger ω?

Question 4. What happens to the harmonic fit if you change the reference date by a quarter of a year? By half of a year? By a full year?

Section 3. Fit a Harmonic Regression at Each Landsat Pixel

In the previous section, we added the harmonic basis (cosine, sine, and constant terms) to each image in our Landsat collection. Now we can run a linear regression using this basis as the independent variables, and Landsat bands and GCVI as the dependent variables. This section defines several functions that will help us complete this regression.

The key step is linear regression performed by the `ee.Reducer.linearRegression` reducer. This reducer takes in two parameters: the number of independent variables and the number of dependent variables. When applied to an `ImageCollection`, it expects each image in the collection to be composed of its independent variable bands followed by a dependent variable band or bands. The reducer returns an array image whose first band is an array of regression coefficients computed through linear least squares and whose second band is the regression residual.

The first function we define is called `arrayimgHarmonicRegr` and helps us apply the linear regression reducer to each image.

```
// Function to run linear regression on an image.
function arrayimgHarmonicRegr(harmonicColl, dependent, independents) {

    independents = ee.List(independents);
    dependent = ee.String(dependent);

    var regression = harmonicColl
        .select(independents.add(dependent))
        .reduce(ee.Reducer.linearRegression(independents.length(),
        1));
```

```
    return regression;
}
```

Next, we want to transform the returned array image into an image with each coefficient as its own band (Chap. F3.1). Furthermore, since we will be running harmonic regressions for multiple Landsat bands, we want to rename the harmonic coefficients returned by our regression to match their corresponding band. The second function, `imageHarmonicRegr`, performs these operations. The array image is transformed into a multiband image using the functions `arrayProject` and `arrayFlatten`, and the coefficients for the cosine, sine, and constant terms are renamed by adding the band name as a prefix (e.g., the first cosine coefficient for the NIR band becomes `NIR_cos`).

```
// Function to extract and rename regression coefficients.
function imageHarmonicRegr(harmonicColl, dependent, independents) {

    var hregr = arrayimgHarmonicRegr(harmonicColl, dependent,
        independents);

    independents = ee.List(independents);
    dependent = ee.String(dependent);

    var newNames = independents.map(function(b) {
        return dependent.cat(ee.String('_')).cat(ee.String(
        b));
    });

    var imgCoeffs = hregr.select('coefficients')
        .arrayProject([0])
        .arrayFlatten([independents])
        .select(independents, newNames);

    return imgCoeffs;
}
```

Finally, the third function is called `getHarmonicCoeffs` and performs the harmonic regression for all dependent bands by mapping the `imageHarmonicRegr` function over each band. It then creates a multiband image comprised of the regression coefficients.

```
// Function to apply imageHarmonicRegr and create a multi-band image.
function getHarmonicCoeffs(harmonicColl, bands, independents) {
    var coefficients = ee.ImageCollection.fromImages(bands.map(
        function(band) {
            return imageHarmonicRegr(harmonicColl, band,
                independents);
        }));
    return coefficients.toBands();
}
```

Now that we have defined all the helper functions, we are ready to apply them to our Landsat `ImageCollection` with the harmonic basis. We are using GCVI, NIR, SWIR1, and SWIR2 as the bands for crop type mapping. For each band, we have five coefficients from the regression corresponding to the `cos`, `sin`, `cos2`, `sin2`, and `constant` terms. In total, then, we will end up with 4 x 5 = 20 harmonic coefficients. After applying `getHarmonicCoeffs`, we clip the coefficients to the study area and apply a transformation to change the coefficients from a 64-bit double data type to a 32-bit integer. This last step allows us to save space on storage.

```
// Apply getHarmonicCoeffs to ImageCollection.
var bands = ['NIR', 'SWIR1', 'SWIR2', 'GCVI'];
var independents = ee.List(['constant', 'cos', 'sin', 'cos2',
'sin2']);
var harmonics = getHarmonicCoeffs(landsatPlus, bands, independents);

harmonics = harmonics.clip(geometry);
harmonics = harmonics.multiply(10000).toInt32();
```

Let's take a look at the harmonics that we've fit. Following the same logic as in Chap. F4.6, we can compute the fitted values by multiplying the regression coefficients with the independent variables. We'll do this just for GCVI, as we did in Sect. 1 above.

```
// Compute fitted values.
var gcviHarmonicCoefficients = harmonics
    .select([
```

```
        '3_GCVI_constant', '3_GCVI_cos',
        '3_GCVI_sin', '3_GCVI_cos2', '3_GCVI_sin2'
    ])
    .divide(10000);

var fittedHarmonic = landsatPlus.map(function(image) {
    return image.addBands(
        image.select(independents)
        .multiply(gcviHarmonicCoefficients)
        .reduce('sum')
        .rename('fitted')
    );
});
```

Now we can plot the fitted values along with the original Landsat GCVI time series in one chart (Fig. A1.1.4).

```
// Visualize the fitted harmonics in a chart.
var harmonicsChart = ui.Chart.image.series(
        fittedHarmonic.select(
            ['fitted', 'GCVI']), point, ee.Reducer.mean(), 30)
    .setSeriesNames(['GCVI', 'Fitted'])
    .setOptions({
        title: 'Landsat GCVI time series and fitted harmonic
regression values',
        lineWidth: 1,
        pointSize: 3,
    });

print(harmonicsChart);
```

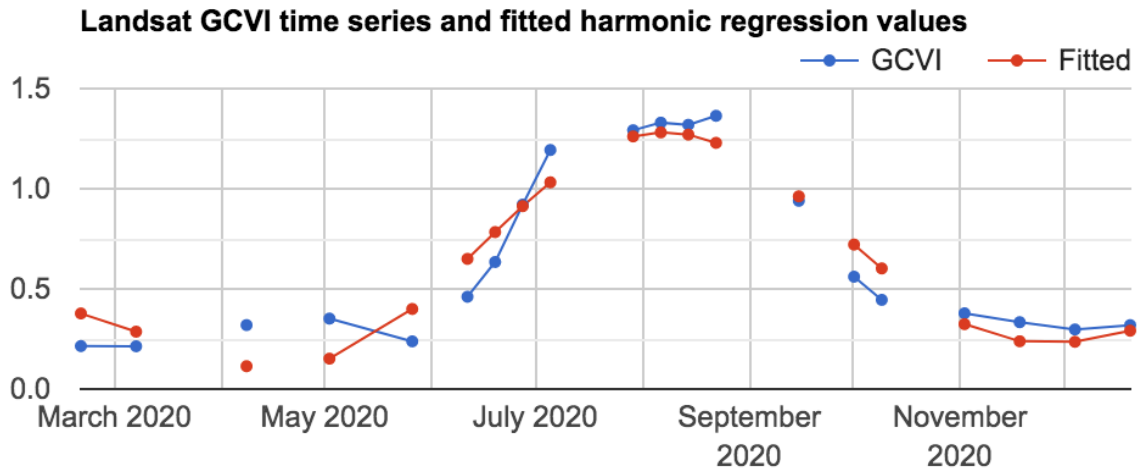


Fig. A1.1.4 The Landsat GCVI time series (blue line) and fitted harmonic regression (red line) curve visualized at a corn point in McLean County, Illinois. While the fit is not perfect, the fitted harmonic captures the general characteristics of crop green-up and senescence.

Before we move on to classifying crop type, we will add CDL as a band to the harmonics and export the final image as an asset. Recall that CDL will be the source of crop type labels for model training; adding this band now is not necessary but makes the labels more accessible at training time.

```
// Add CDL as a band to the harmonics.
var harmonicsPlus = ee.Image.cat([harmonics, cdl]);
```

We are ready to export the harmonics we've computed to an asset. Exporting to an asset is an optional step but can make it easier in the next step to train a model. For a larger study area, the operations may time out without the export step because computing harmonics is computationally intensive. You should replace `'your_asset_path_here/'` and `filename` below with the place you want to save your harmonics.

```
// Export image to asset.
var filename = 'McLean_County_harmonics';
```

```
Export.image.toAsset({
    image: harmonicsPlus,
    description: filename,
    assetId: 'your_asset_path_here/' + filename,
    dimensions: null,
    region: region,
    scale: 30,
    maxPixels: 1e12
});
```

We visualized the fitted harmonics at one point in the study area; let's also take a look at the coefficients over the entire study area using `Map.addLayer`. Since we can only visualize three bands at a time, we will have to pick three and visualize them in false color. Sometimes, it takes a bit of tweaking to visualize regression coefficients in false color; below, we transform three fitted GCVI bands via division and addition by constants (obtained through trial and error) to obtain a nice image for visualization.

```
// Visualize harmonic coefficients on map.
var visImage = ee.Image.cat([
    harmonics.select('3_GCVI_cos').divide(7000).add(0.6),
    harmonics.select('3_GCVI_sin').divide(7000).add(0.5),
    harmonics.select('3_GCVI_constant').divide(7000).subtract(
        0.6)
]);

Map.addLayer(visImage, {
    min: -0.5,
    max: 0.5
}, 'Harmonic coefficient false color');
```

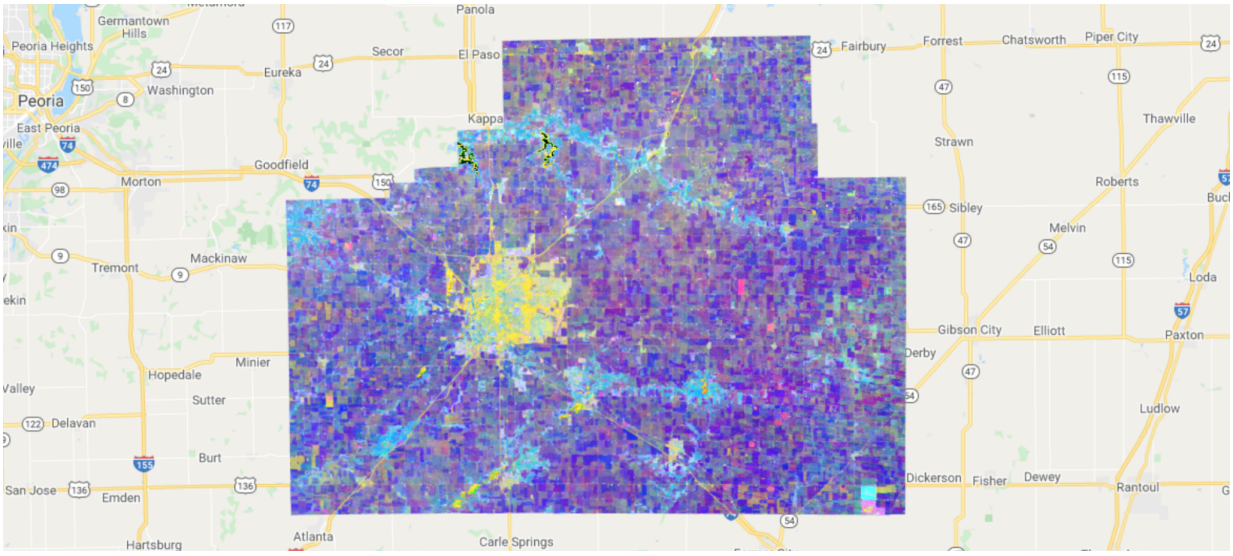


Fig. A1.1.5 False-color image of harmonic coefficients fitted to GCVI time series in McLean County, Illinois

These harmonic features allow us to see differences between urban land cover, water, and what we are currently interested in, crop types. Looking closely at the false-color image shown in Fig. A1.1.5, we can also see striping unrelated to land cover throughout the image. This striping, most visible in the bottom right of the image, is due to the failure of Landsat 7's Scan Line Corrector in 2003, which resulted in data gaps that occur in a striped pattern across Landsat 7 images. It is worth noting that these artifacts can affect classification performance, but exactly how much depends on the task.

Code Checkpoint A11c. The book's repository contains a script that shows what your code should look like at this point.

Question 5. What color do forested areas appear as in Fig. A1.1.5? Urban areas? Water?

Question 6. Comparing the map layers in Fig. A1.1.5 and Fig. A1.1.3, can you tell whether the GCVI cosine, sine, and constant terms capture differences between corn and soybean time series? What color do corn fields appear as in the false-color visualization? What about soybean fields?

Section 4. Train and Evaluate a Random Forest Classifier

We have our features, and we are ready to train a random forest in McLean County to classify crop types. As explained in Chap. F2.1, random forests are a classifier available in Earth Engine that can achieve high accuracies while being computationally efficient. We define a `ee.Classifier` object with 100 trees and a minimum leaf population of 10.

```
// Define a random forest classifier.
var rf = ee.Classifier.smileRandomForest({
    numberOfTrees: 50,
    minLeafPopulation: 10,
    seed: 0
});
```

The bands that we will use as features in the classification are the harmonic coefficients fitted to the NIR, SWIR1, SWIR2, and GCVI time series at each pixel. This may be different from previous classification examples you have seen, where band values at one time step are used as dependent variables. Using harmonic coefficients is one way to provide information from the temporal dimension to a classifier. Temporal information is very useful for distinguishing crop types.

We can obtain the band names by calling `bandNames` on the harmonics image and removing the CDL band and the `system:index` band.

```
// Get harmonic coefficient band names.
var bands = harmonicsPlus.bandNames();
bands = bands.remove('cropland').remove('system:index');
```

To prepare the CDL band to be the output of classification, we remap the crop-type values in CDL (corn = 1, soybeans = 5, and over a hundred other crop types) to corn taking on a value of 1, soybeans a value of 2, and everything else a value of 0. We focus on classifying corn and soybeans for this exercise.

```
// Transform CDL into a 3-class band and add to harmonics.
var cornSoyOther = harmonicsPlus.select('cropland').eq(1)
    .add(harmonicsPlus.select('cropland').eq(5).multiply(2));
```

```
var dataset = ee.Image.cat([harmonicsPlus.select(bands),
    cornSoyOther]);
```

Next, we sample 100 points from McLean County to serve as the training sample for our model. The classifier will learn to differentiate among corn, soybeans, and everything else using this set of points.

```
// Sample training points.
var train_points = dataset.sample(geometry, 30, null, null, 100, 0);
print('Training points', train_points);
```

Training the model is as simple as calling `train` on the classifier and feeding it the training points, the name of the label band, and the names of the input bands. To assess model performance on the training set, we can then compute the confusion matrix (Chap. F.2.2) by calling `confusionMatrix` on the model object. Finally, we can compute the accuracy by calling `accuracy` on the confusion matrix object.

```
// Train the model!
var model = rf.train(train_points, 'cropland', bands);
var trainCM = model.confusionMatrix();
print('Training error matrix: ', trainCM);
print('Training overall accuracy: ', trainCM.accuracy());
```

You should see the confusion matrix shown in Table A1.1.1 on the training set.

Table A1.1.1 The confusion matrix obtained for the training set

		Predicted crop type		
		Other (class 0)	Corn (class 1)	Soy (class 2)
Actual crop type	Other (class 0)	27	1	1
	Corn (class 1)	2	34	0
	Soy (class 2)	3	4	28

Table A1.1.1 tells us that the classifier is achieving 89% accuracy on the training set. Given that the model is successful on the training set, we next want to explore how well it will generalize to the entire study region. To estimate generalization performance, we can sample a test set (also called a "validation set") and apply the model to that feature collection using `classify(model)`. To keep the amount of computation manageable and avoid exceeding Earth Engine's memory quota, we sample 50 test points.

```
// Sample test points and apply the model to them.
var test_points = dataset.sample(geometry, 30, null, null, 50, 1);
var tested = test_points.classify(model);
```

Next, we can compute the confusion matrix and accuracy on the test set by calling `errorMatrix('cropland', 'classification')` followed by `accuracy` on the classified points. The `'classification'` argument refers to the property where the model predictions are stored.

```
// Compute the confusion matrix and accuracy on the test set.
var testAccuracy = tested.errorMatrix('cropland', 'classification');
print('Test error matrix: ', testAccuracy);
print('Test overall accuracy: ', testAccuracy.accuracy());
```

You should see a test set confusion matrix matching (or perhaps almost matching) Table A1.1.2.

Table A1.1.2 The confusion matrix obtained for the test set

		Predicted crop type		
		Other (class 0)	Corn (class 1)	Soy (class 2)
Actual crop type	Other (class 0)	12	1	0
	Corn (class 1)	1	15	2
	Soy (class 2)	1	5	13

The test set accuracy of the model is lower than the training set accuracy, which is indicative of some overfitting and is common when the model is expressive. We also see

from Table A1.1.2 that most of the error comes from confusing corn pixels with soybean pixels and vice versa. Understanding the errors in your classifier can help you build better models in the next iteration.

Beyond estimating the generalization error, we can actually apply the model to the entire study region and see how well the model performs across space. We do this by calling `classify(model)` again on the harmonic image for McLean County. We add it to the map to visualize our predictions (Fig. A1.1.6).

```
// Apply the model to the entire study region.
var regionClassified = harmonicsPlus.select(bands).classify(model);
var predPalette = ['gray', 'yellow', 'green'];
Map.addLayer(regionClassified, {
    min: 0,
    max: 2,
    palette: predPalette
}, 'Classifier prediction');
```

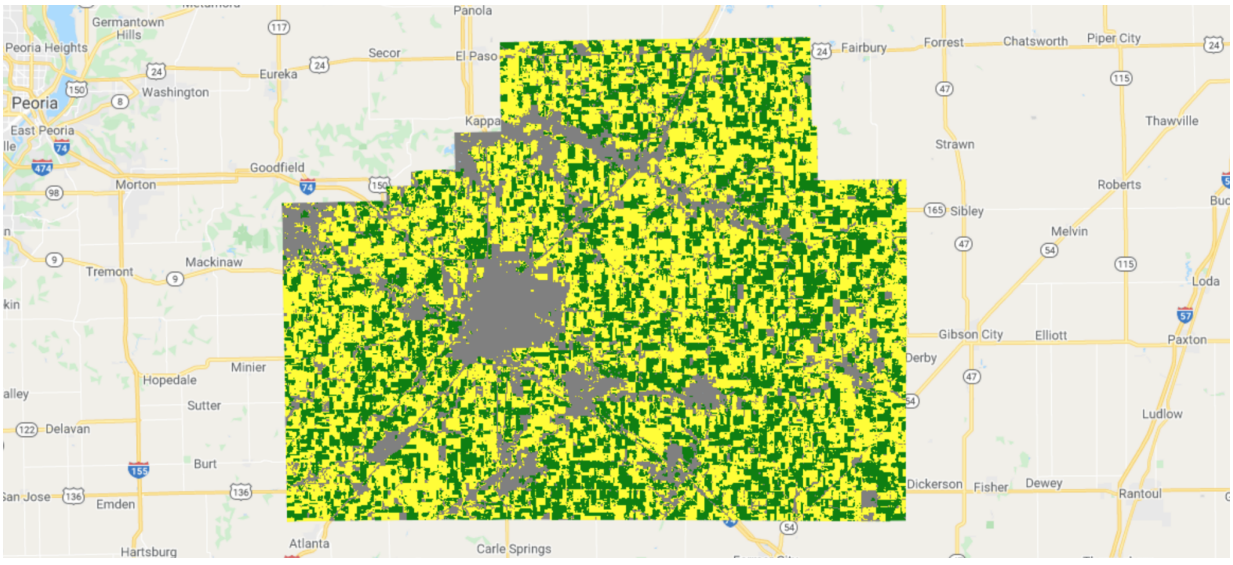


Fig. A1.1.6 Model predictions of crop type in McLean County, Illinois. Yellow is corn, green is soybeans, and gray is other land cover.

How do our model predictions compare to the CDL labels? We can see where the two are equal and where they differ by calling `eq` on the two images.

```
// Visualize agreements/disagreements between prediction and CDL.
Map.addLayer(regionClassified.eq(cornSoyOther), {
    min: 0,
    max: 1
}, 'Classifier agreement');
```

Fig. A1.1.7 shows agreement between our predictions and CDL labels. When visualized this way, some patterns become clear: Many errors occur along field boundaries, and sometimes entire fields are misclassified.

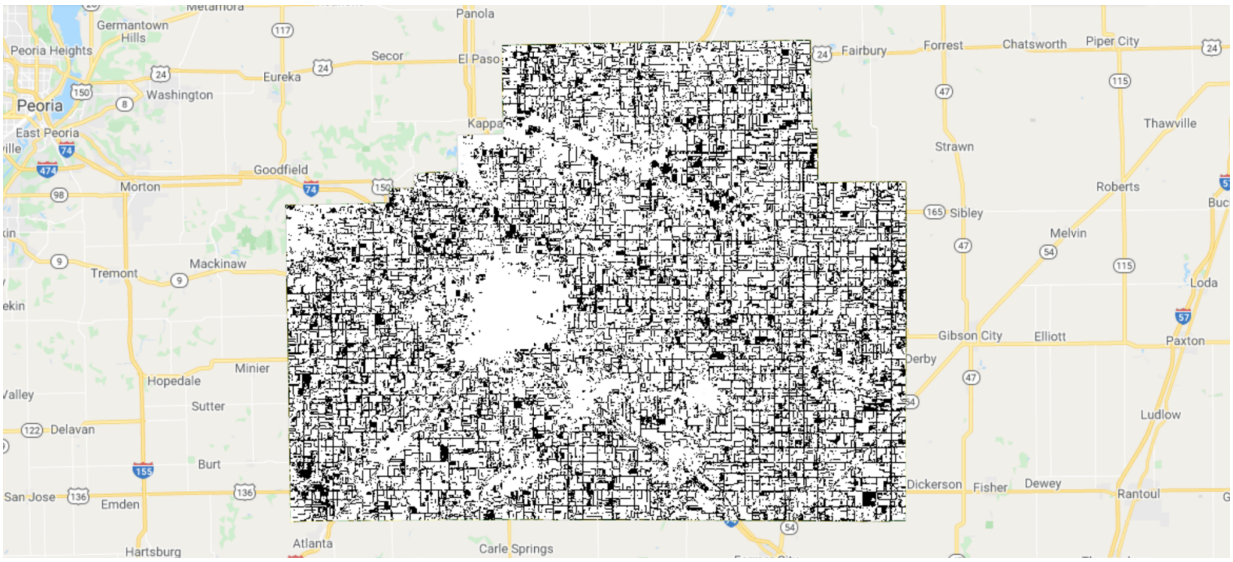


Fig. A1.1.7 Prediction agreement with CDL as ground truth labels. White pixels are where our predictions and CDL agree; black pixels are where they disagree. Errors happen on field boundaries and sometimes inside fields as well.

Code Checkpoint A11d. The book's repository contains a script that shows what your code should look like at this point.

Question 7. As important as accuracy is for summarizing a classifier's overall performance, we often want to know class-specific performance as well. What are the

producer's and user's accuracies (Chap. F2.2) for the corn class? For the soybean class? For the "other" class?

Question 8. Investigate the labels of field boundaries. What class is our model classifying them as, and what class does CDL classify them as? Now investigate the GCVI time series and fitted harmonic regressions at field boundaries (Fig. A1.1.4). Do they look different from time series inside fields? Why do you think our model is having a hard time classifying field boundaries as the CDL class?

Synthesis

When we trained the above classifier to identify crop types, we made many choices, such as which type of classifier to use (random forest), which Landsat 7 bands to use (NIR, SWIR1, SWIR2, GCVI), how to extract features (harmonic regression), and how to sample training points (100 uniformly random points). In practice, any machine learning task will involve choices like these, and it is important to understand how these choices affect the performance of the classifier we end up with. To gain an understanding of how some of these choices affect crop classification accuracy, try the following:

Assignment 1. Instead of using only three Landsat bands plus GCVI, add the blue, green, and red bands as well. You should end up with 35 harmonic coefficients per pixel instead of 20. What happens to the out-of-sample classification accuracy when you add these three additional bands as features?

Assignment 2. Instead of a second-order harmonic regression, try a third-order harmonic regression. You should end up with 28 harmonic coefficients per pixel instead of 20. What happens to the out-of-sample classification accuracy?

Assignment 3. Using the original set of features (20 harmonic coefficients), train a random forest using 10, 20, 50, 200, 500, and 1000 training points. How does the out-of-sample classification accuracy change as the training set size increases?

Assignment 4. Extra challenge: Instead of using Landsat 7 and 8 as the input imagery, perform the same crop type classification using Sentinel-2 data and report the out-of-sample classification accuracy.

Conclusion

In this chapter, we showed how to use datasets and functions available in Earth Engine to classify crop types in the US Midwest. In particular, we used Landsat 7 and 8 time

series as inputs, extracted features from the time series using harmonic regression, obtained labels from the CDL, and predicted crop type with a random forest classifier. Remote sensing data can be used to understand many more aspects of agriculture beyond crop type, such as crop yields, field boundaries, irrigation, tillage, cover cropping, soil moisture, biodiversity, and pest pressure. While the exact imagery source, classifier type, and classification task can all differ, the general workflow is often similar to what has been laid out in this chapter. Since farming sustainably while feeding 11 billion people by 2100 is one of the great challenges of this century, we encourage you to continue exploring how Earth Engine and remote sensing data can help us understand agricultural environments around the world.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Azzari G, Grassini P, Edreira JIR, et al (2019) Satellite mapping of tillage practices in the North Central US region from 2005 to 2016. Remote Sens Environ 221:417–429. https://doi.org/10.1016/j.rse.2018.11.010

Burke M, Lobell DB (2017) Satellite-based assessment of yield variation and its determinants in smallholder African systems. Proc Natl Acad Sci USA 114:2189–2194. https://doi.org/10.1073/pnas.1616919114

Deines JM, Kendall AD, Hyndman DW (2017) Annual irrigation dynamics in the U.S. Northern High Plains derived from Landsat satellite data. Geophys Res Lett 44:9350–9360. https://doi.org/10.1002/2017GL074071

Duro DC, Girard J, King DJ, et al (2014) Predicting species diversity in agricultural environments using Landsat TM imagery. Remote Sens Environ 144:214–225. https://doi.org/10.1016/j.rse.2014.01.001

Food and Agriculture Organization of the United Nations (2020) Land use in agriculture by the numbers. In: Sustain. Food Agric. https://www.fao.org/sustainability/news/detail/en/c/1274219/. Accessed 11 Nov 2021

Ihuoma SO, Madramootoo CA (2017) Recent advances in crop water stress detection. Comput Electron Agric 141:267–275. https://doi.org/10.1016/j.compag.2017.07.026

Jain M, Srivastava AK, Balwinder-Singh, et al (2016) Mapping smallholder wheat yields and sowing dates using micro-satellite data. Remote Sens 8:860. https://doi.org/10.3390/rs8100860

Jin Z, Prasad R, Shriver J, Zhuang Q (2017) Crop model- and satellite imagery-based recommendation tool for variable rate N fertilizer application for the US Corn system. Precis Agric 18:779–800. https://doi.org/10.1007/s11119-016-9488-z

Seifert CA, Azzari G, Lobell DB (2018) Satellite detection of cover crops and their effects on crop yield in the Midwestern United States. Environ Res Lett 13:64033 https://doi.org/10.1088/1748-9326/aaf933

USDA (2022) CropScape - Cropland Data Layer. In: Crop. - NASS CDL Progr. https://nassgeodata.gmu.edu/CropScape/. Accessed 12 Nov 2021

Wang S, Azzari G, Lobell DB (2019) Crop type mapping without field-level labels: Random forest transfer and unsupervised clustering techniques. Remote Sens Environ 222:303–317. https://doi.org/10.1016/j.rse.2018.12.026

Chapter A1.2: Urban Environments

Authors
Michelle Stuhlmacher and Ran Goldblatt

Overview

Urbanization has dramatically changed Earth's surface. This chapter starts with a qualitative look at the impact urban expansion has on the landscape, covering three existing urban classification schemes that have been created by other remote sensing scientists. We look at how these classifications can be used to quantify urban areas, and close with instructions on how to perform per-pixel supervised image classification to map built-up land cover at any location on Earth and at any point in time using Landsat 7 imagery.

Learning Outcomes

- Creating an animated GIF.
- Implementing quantitative and qualitative analyses with pre-existing urban classifications.
- Running your own classification of built-up land cover.
- Quantifying and mapping the extent of urbanization across space and time.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Use drawing tools to create points lines and polygons (Chap. F2.1).
- Perform a supervised image classification (Chap. F2.1).
- Use `reduceRegions` to summarize an image with zonal statistics in irregular shapes (Chap. F5.0, Chap. F5.2).

Introduction to Theory

Urbanization and its consequences have expanded rapidly over the last several decades. In 1950, only 30% of the world's population lived in urban areas, but that figure is expected to be 68% by 2050 (United Nations 2018). Already, over 50% of the world's population lives in urban areas (United Nations 2018). This shift to urban dwelling has helped lift millions of people out of poverty, but it is also creating complicated

socio-environmental challenges, such as habitat loss, urban heat islands, flooding, and greater greenhouse gas emissions (Bazaz et al. 2018, United Nations 2018).

An important part of addressing these challenges is understanding urbanization trajectories and their consequences. Satellite imagery provides a rich source of historic and current information on urbanization all over the globe, and it is increasingly being leveraged to contribute to research on urban sustainability (Goldblatt et al. 2018, Prakash et al. 2020). In this chapter, we will cover several methods for visualizing and quantifying urbanization's impact on the landscape.

Practicum

Section 1. Time Series Animation

To start our examination of urban environments, we'll create a time series animation (GIF) for a qualitative depiction of the impact of urbanization. We'll use Landsat imagery to visualize how the airport northwest of Bengaluru, India, grew between 2010 and 2020.

First, search for "Gangamuthanahalli" in the Code Editor search bar and click on the city name to navigate there. Gangamuthanahalli is home to Kempegowda International Airport. Use the **Geometry Tools**, as shown in Chap. F2.1, to draw a rectangle around the airport.

Now search for "landsat 8 level 2" and import the "USGS Landsat 8 Level 2, Collection 2, Tier 1" `ImageCollection`. Name the import `L8`. Filter the collection by the geometry you created and the date range of interest, 2010–2020. Limit cloud cover on land to 3%.

```
// Filter collection.
var collection = L8
    .filterBounds(geometry)
    .filterDate('2010-01-01', '2020-12-31')
    .filter(ee.Filter.lte('CLOUD_COVER_LAND', 3));
```

Next, set up the parameters for creating a GIF. We want to visualize the three visible bands over the airport region and cycle through the images at 15 frames per second.

```
// Define GIF visualization arguments.
```

```
var gifParams = {
    bands: ['SR_B4', 'SR_B3', 'SR_B2'],
    min: 0.07 * 65536,
    max: 0.3 * 65536,
    region: geometry,
    framesPerSecond: 15,
    format: 'gif'
};
```

Last, render the GIF in the **Console**.

```
// Render the GIF animation in the console.
print(ui.Thumbnail(collection, gifParams));
```

Code Checkpoint A12a. The book's repository contains a script that shows what your code should look like at this point.

The GIF gives a qualitative sense of the way the airport has changed. To quantify urban environment change (i.e., summing the area of total new impervious surfaces), we often turn to classifications. Section 2 covers pre-existing classifications you can leverage to quantify urban extent.

Section 2. Pre-Existing Urban Classifications

There are several publicly accessible land cover classifications that include urban areas as one of their classes. These classifications are a quick way to qualify urban extent. In this section, we'll cover three from the Earth Engine Data Catalog and show you how to visualize and quantify the urban classes: (1) the MODIS Land Cover Type Yearly Global; (2) the Copernicus CORINE Land Cover; and (3) the United States Geological Survey (USGS) National Land Cover Database (NLCD). These three classifications vary in their spatial and temporal resolution. For example, MODIS is a global dataset, while the NLCD covers only the United States, and CORINE covers only the European Union. We'll explore more of these differences later.

First, start a new code and search for "MODIS land cover." Find the `ImageCollection` titled "MCD12Q1.006 MODIS Land Cover Type Yearly Global 500m," import it, and

rename it to `MODIS`. We will first look at the MODIS land cover classification over Accra, Ghana.:

```
// MODIS (Accra)
// Center over Accra.
Map.setCenter(-0.2264, 5.5801, 10);
```

The Land Cover Type Yearly Global dataset contains multiple classifications (Fig. A1.2.1). We will use `LC_Type1`, the annual classification using the International Geosphere-Biosphere Programme (IGBP) categories, in this example.

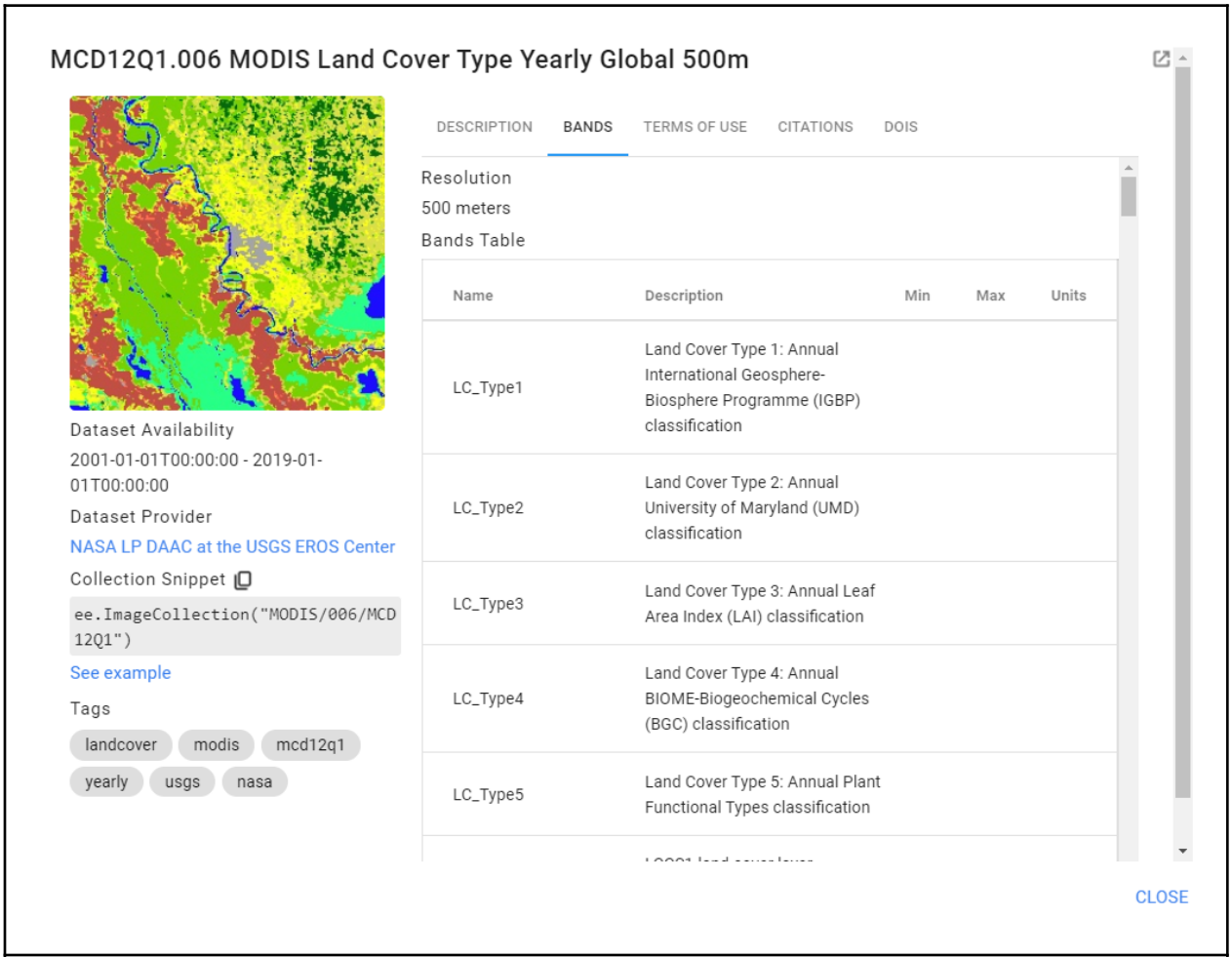


Fig. A1.2.1 Metadata of the MODIS Land Cover classification types

Select the `LC_Type1` band, copy and paste the visualization parameters for the classification, and visualize the full classification.

```
// Visualize the full classification.
var MODIS_lc = MODIS.select('LC_Type1');
var igbpLandCoverVis = {
    min: 1.0,
    max: 17.0,
    palette: ['05450a', '086a10', '54a708', '78d203', '009900',
        'c6b044', 'dcd159', 'dade48', 'fbff13', 'b6ff05',
        '27ff87', 'c24f44', 'a5a5a5', 'ff6d4c', '69fff8',
        'f9ffa4', '1c0dff'
    ],
};
Map.addLayer(MODIS_lc, igbpLandCoverVis, 'IGBP Land Cover');
```

Press **Run** and view the output with the classification visualization. Use the **Inspector** tab to click on the different colors and determine the land cover classes they represent.

Next, we're going to visualize only the urban class at points almost two decades apart: from 2001 and 2019. We'll start by filtering for 2019 dates.

```
// Visualize the urban extent in 2001 and 2019.
// 2019
var MODIS_2019 = MODIS_lc.filterDate(ee.Date('2019-01-01'));
```

LC_Type1 Class Table

Value	Color	Description
1	05450a	Evergreen Needleleaf Forests: dominated by evergreen conifer trees (canopy >2m). Tree cover >60%.
2	086a10	Evergreen Broadleaf Forests: dominated by evergreen broadleaf and palmate trees (canopy >2m). Tree cover >60%.
3	54a708	Deciduous Needleleaf Forests: dominated by deciduous needleleaf (larch) trees (canopy >2m). Tree cover >60%.
4	78d203	Deciduous Broadleaf Forests: dominated by deciduous broadleaf trees (canopy >2m). Tree cover >60%.
5	009900	Mixed Forests: dominated by neither deciduous nor evergreen (40-60% of each) tree type (canopy >2m). Tree cover >60%.
6	c6b044	Closed Shrublands: dominated by woody perennials (1-2m height) >60% cover.
7	dcd159	Open Shrublands: dominated by woody perennials (1-2m height) 10-60% cover.
8	dade48	Woody Savannas: tree cover 30-60% (canopy >2m).
9	fbff13	Savannas: tree cover 10-30% (canopy >2m).
10	b6ff05	Grasslands: dominated by herbaceous annuals (<2m).
11	27ff87	Permanent Wetlands: permanently inundated lands with 30-60% water cover and >10% vegetated cover.
12	c24f44	Croplands: at least 60% of area is cultivated cropland.
13	a5a5a5	Urban and Built-up Lands: at least 30% impervious surface area including building materials, asphalt and vehicles.
14	ff6d4c	Cropland/Natural Vegetation Mosaics: mosaics of small-scale cultivation 40-60% with natural tree, shrub, or herbaceous vegetation.
15	69fff8	Permanent Snow and Ice: at least 60% of area is covered by snow and ice for at least 10 months of the year.
16	f9ffa4	Barren: at least 60% of area is non-vegetated barren (sand, rock, soil) areas with less than 10% vegetation.
17	1c0dff	Water Bodies: at least 60% of area is covered by permanent water bodies.

Fig. A1.2.2 Classes for the `LC_Type1` classification

Looking at the metadata shows us that the urban class has a value of 13 (Fig. A1.2.2), so we'll select only the pixels with this value and visualize them for 2019. To show only the urban pixels, mask the urban image with itself in the `Map.addLayer` command.

```
var M_urb_2019 = MODIS_2019.mosaic().eq(13);
Map.addLayer(M_urb_2019.mask(M_urb_2019), {
    'palette': 'FF0000'
}, 'MODIS Urban 2019');
```

Repeat the same steps for 2001, but give it a gray palette to contrast with the red palette of 2019.

```
var MODIS_2001 = MODIS_lc.filterDate(ee.Date('2001-01-01'));
var M_urb_2001 = MODIS_2001.mosaic().eq(13);
Map.addLayer(M_urb_2001.mask(M_urb_2001), {
    'palette': 'a5a5a5'
}, 'MODIS Urban 2001');
```

The result is a visualization showing the extent of Accra in 2002 in gray and the new urbanization between 2001 and 2019 in red (Fig. A1.2.3).

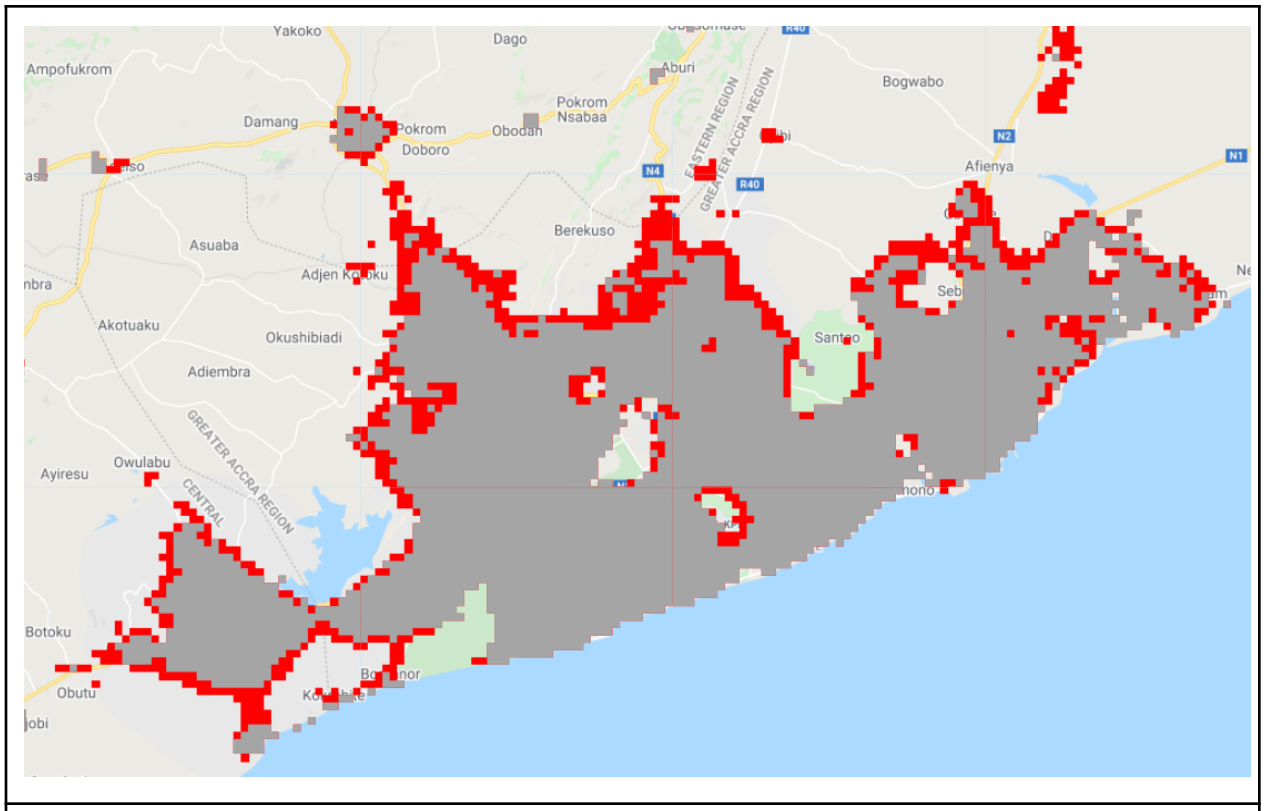


Fig. A1.2.3 Output of MODIS visualization

Code Checkpoint A12b. The book's repository contains a script that shows what your code should look like at this point.

Next, we'll look at CORINE. Open a new script, search for "CORINE," select the Copernicus CORINE Land Cover dataset to import, rename it to `CORINE`, and center it over London, England.

```
// CORINE (London)
// Center over London
Map.setCenter(-0.1795, 51.4931, 10);
```

Conducting a similar process to the one we used with MODIS, first select the image for 2018, the most recent classification year.

```
// Visualize the urban extent in 2000 and 2018.
// 2018 (2017-2018)
var CORINE_2018 = CORINE.select('landcover').filterDate(ee.Date(
    '2017-01-01'));
```

Why filter by 2017 and not 2018? The description section of CORINE's metadata shows that the time period covered by the 2018 asset includes both 2017 and 2018. To select the 2018 asset in the code above, we filter by the date of the first year. (If you're curious, you can test what happens when you filter using 2018 as the date.)

The metadata for CORINE show that the built-up land cover classes range from 111 to 133, so we'll select all classes less than or equal to 133 and visualize them in red.

```
var C_urb_2018 = CORINE_2018.mosaic().lte(133); //Select urban areas
Map.addLayer(C_urb_2018.mask(C_urb_2018), {
    'palette': 'FF0000'
}, 'CORINE Urban 2018');
```

Repeat the same steps for 2000, but visualize the pixels in gray.

```
// 2000 (1999-2001)
var CORINE_2000 = CORINE.select('landcover').filterDate(ee.Date(
    '1999-01-01'));
var C_urb_2000 = CORINE_2000.mosaic().lte(133); //Select urban areas
Map.addLayer(C_urb_2000.mask(C_urb_2000), {
    'palette': 'a5a5a5'
}, 'CORINE Urban 2000');
```

The output shows the extent of London's built-up land in 2000 in gray, and the larger extent in 2018 in red (Fig. A1.2.4).

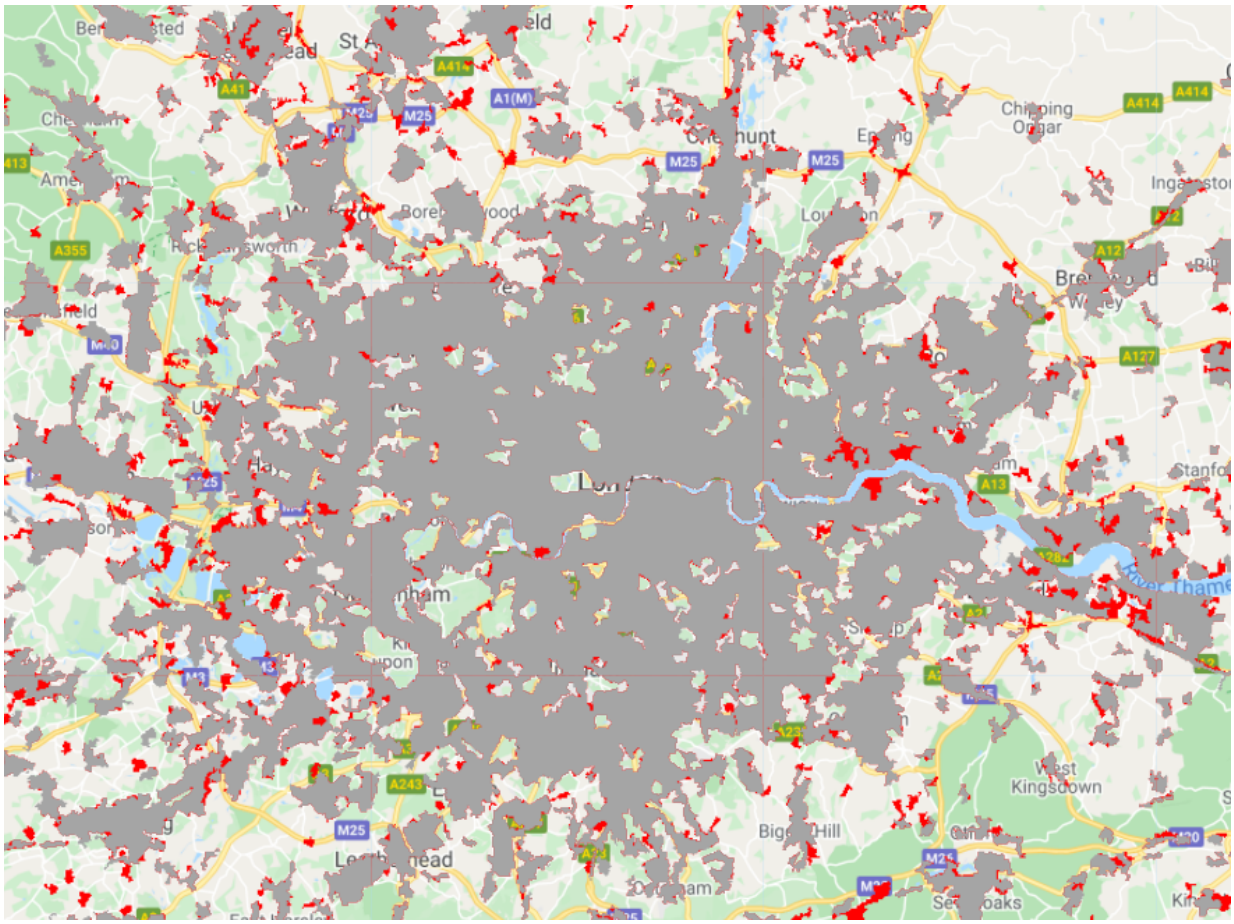


Fig. A1.2.4 Output of CORINE visualization

Code Checkpoint A12c. The book's repository contains a script that shows what your code should look like at this point.

Last, we'll look at NLCD for 2016. Open a new script, search for "NLCD," import the NLCD: USGS National Land Cover Database, rename it to `NLCD`, and center it over Chicago, Illinois, USA.

```
// NLCD (Chicago)
// Center over Chicago.
```

```
Map.setCenter(-87.6324, 41.8799, 10);
```

Select the `landcover` classification band and filter it to the 2016 classification. Previously, we have done so using the `filterDate` command, but you can also filter on the `system:index`.

```
// Select the land cover band.
var NLCD_lc = NLCD.select('landcover');

// Filter NLCD collection to 2016.
var NLCD_2016 = NLCD_lc.filter(ee.Filter.eq('system:index', '2016'))
    .first();
Map.addLayer(NLCD_2016, {}, 'NLCD 2016');
```

Hit **Run** and view the classification visualization. Use the **Inspector** tab to explore the classification values. The shades of red represent different levels of development. Dark red is "Developed, High Intensity" and the lightest pink is "Developed, Open Space." You can look at the metadata description of this dataset to learn about the rest of the classes and their corresponding colors.

One of the benefits of classifications is that you can use them to calculate quantitative information such as the area of a specific land cover. Now we'll cover how to do so by summing the high-intensity urban development land cover in Chicago. Import a boundary of the city of Chicago and clip the NLCD classification to its extent.

```
// Calculate the total area of the 'Developed high intensity' class
(24) in Chicago.
var Chicago = ee.FeatureCollection(
    'projects/gee-book/assets/A1-2/Chicago');

// Clip classification to Chicago
var NLCD_2016_chi = NLCD_2016.clip(Chicago);
```

Select the "Developed, High Intensity" class (24), and mask the class with itself (to ensure that we calculate the area of only the highly developed pixels and no surrounding ones), then visualize the output.

```
// Set class 24 pixels to 1 and mask the rest.
var NLCD_2016_chi_24 = NLCD_2016_chi.eq(24).selfMask();
Map.addLayer(NLCD_2016_chi_24, {},
    'Chicago developed high intensity');
```

Multiply the pixels by their area, and use `reduceRegions` to sum the area of the pixels.

```
// Area calculation.
var areaDev = NLCD_2016_chi_24.multiply(ee.Image.pixelArea())
    .reduceRegion({
        reducer: ee.Reducer.sum(),
        geometry: Chicago.geometry(),
        scale: 30
    })
    .get('landcover');
print(areaDev);
```

Code Checkpoint A12d. The book's repository contains a script that shows what your code should look like at this point.

If everything worked correctly, you should see "203722704.70588234" printed to the **Console**. This means that highly developed land covered about 204,000,000 m^2 (204 km^2) in the city of Chicago in 2016. Let's roughly check this answer by looking up the area of the city of Chicago. Chicago is about 234 square miles (~606 km^2), which means highly developed areas cover about a third of the city, which visually matches what we see in the **Map** viewer.

Question 1. Return to your MODIS classification visualization of Accra and look at the pixels that became urban between 2002 and 2019. What land cover classes were they before they became urbanized? Hint: you will need to change the date of your full classification visualization to 2002 in order to answer this question.

Question 2. Return to your NLCD area calculation. Write new code to calculate the area of the highly developed class in Chicago in 2001. How much did the highly developed class change between 2001 and 2016? Report your answer in square meters.

Question 3. Fill out Table A1.2.1 on the spatial and temporal extents of the three classifications that we've just experimented with (some entries are filled in as an example). Based on your table, what are the benefits and the limitations of each of the classifications for studying urban areas?

Table A1.2.1 Spatial and temporal extents of three classifications

	Pixel Size (m)	**Years Available**	**Coverage**	**Number of Urban Classes**
MODIS	500 m			
CORINE			European Union	
NLCD				4

Section 3. Classifying Urban Areas

In the previous section, we relied on several LULC classifications to estimate the distribution of built-up land cover and its change over time. While for many applications these products would be sufficient, for others their temporal or spatial granularity may be a limiting factor. For example, your research question may require tracking monthly changes in the distribution of built-up land cover or depicting granular intra-city spatial patterns of built-up areas.

Luckily, with emerging and more accessible machine learning tools—like those available in Earth Engine—you can now create your own classification maps, including maps that depict the distribution of built-up land cover (Goldblatt et al. 2016). Machine learning deals with how machines learn rules from examples and generalize these rules to new examples. In this section, we will use one of the most common supervised machine learning classifiers: Random Forests (Breiman 2001).

You will use Landsat 7 imagery to classify built-up land cover in the city of Ahmedabad, Gujarat, India, at two points in time, 2020 and 2010. You will hand draw polygons (rectangles) representing areas of built-up and not built-up land cover, and then build a Random Forest classifier and train it to predict whether a pixel is "built-up" or not.

First, start a new script. In the search bar, search for "USGS Landsat 7 Collection 2 Tier 1". Of the similarly named datasets that come up, select the one with the data set specifier "LANDSAT/LE07/C02/T1_L2". The metadata shows that this is an

`ImageCollection` of images collected since 1999 (Fig. A1.2.5, left) and that each pixel is characterized by 11 spectral bands (Fig. A1.2.5, right) . In this exercise, we will work only with bands 1–6. Import this `ImageCollection` and change the name of the variable to `L7`.

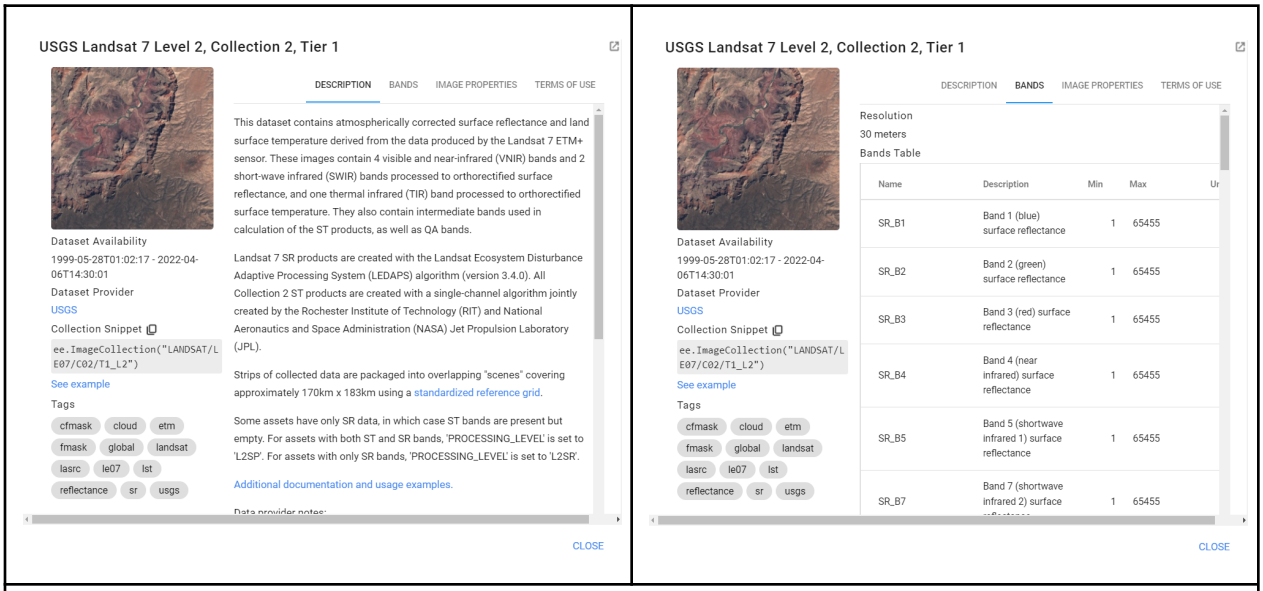


Fig. A1.2.5 Metadata for the "USGS Landsat 7 Collection 2 Tier 1" dataset

Next, you will create an annual composite for the year 2020 using the Google example **Landsat457 Surface Reflectance** script, located in the **Cloud Masking Examples** folder at the bottom of your Scripts section (Fig. A1.2.6).

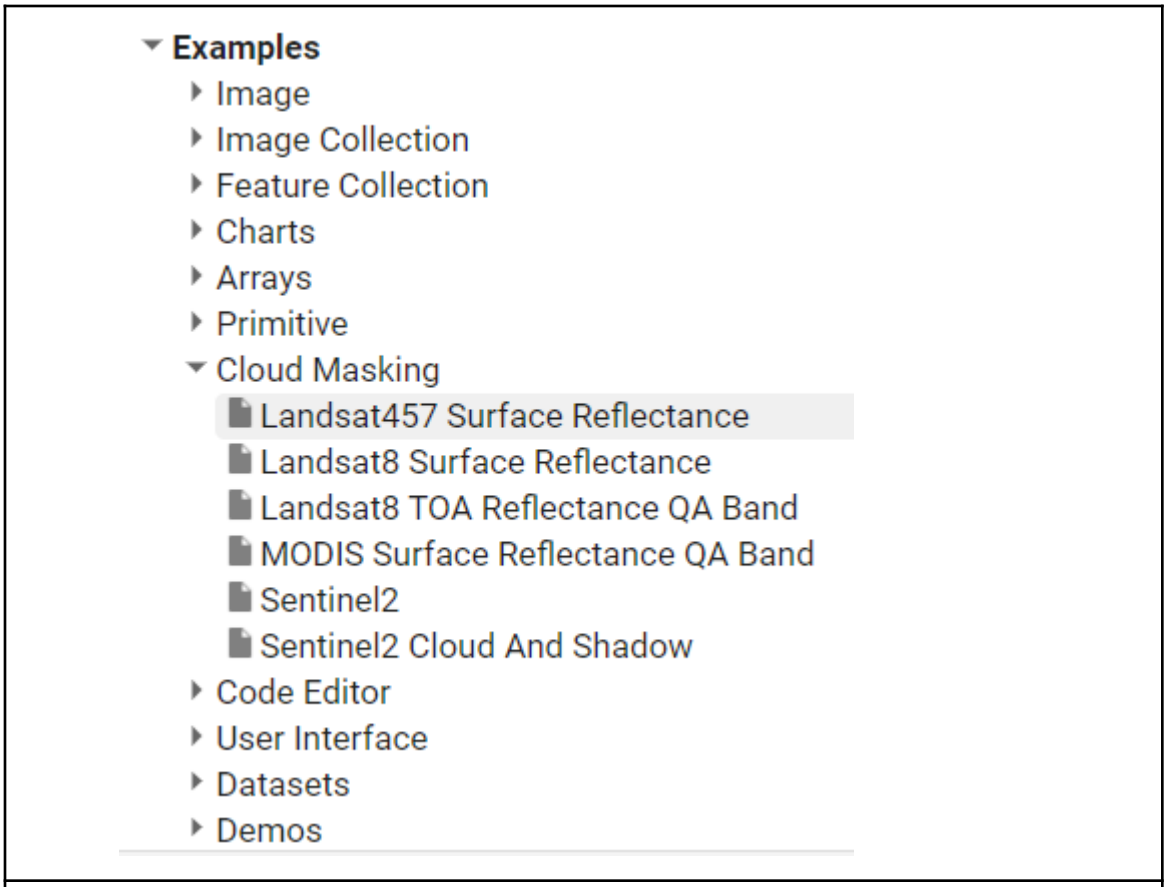


Fig. A1.2.6 Location of the surface reflectance code in the **Examples** repository

This function computes surface reflectance. To create a composite for the year 2020, use the `ee.FilterDate` method to identify the images captured between January 1, 2020 and December 31, 2020, then map the function over that collection.

```
// Surface reflectance function from example:
function maskL457sr(image) {
    var qaMask = image.select('QA_PIXEL').bitwiseAnd(parseInt('11111',
        2)).eq(0);
    var saturationMask = image.select('QA_RADSAT').eq(0);

    // Apply the scaling factors to the appropriate bands.
    var opticalBands = image.select('SR_B.').multiply(0.0000275).add(-
        0.2);
    var thermalBand = image.select('ST_B6').multiply(0.00341802).add(
```

```
        149.0);

    // Replace the original bands with the scaled ones and apply the
masks.
    return image.addBands(opticalBands, null, true)
        .addBands(thermalBand, null, true)
        .updateMask(qaMask)
        .updateMask(saturationMask);
}

// Map the function over one year of data.
var collection = L7.filterDate('2020-01-01', '2021-01-01').map(
    maskL457sr);
var landsat7_2020 = collection.median();
```

Now, in the search bar, search for and navigate to Ahmedabad, India. Add the `landsat7_2020` composite to the map. Specify the red, green, and blue bands to achieve a natural color visualization, noting that in Landsat 7, band 3 (B3) is red, band 2 (B2) is green, and band 1 (B1) is blue. Also specify the colors' stretch parameters. In this case, we stretch the values between 10 and 120, but feel free to experiment with other stretches.

```
Map.addLayer(landsat7_2020, {
    bands: ['SR_B3', 'SR_B2', 'SR_B1'],
    min: 0,
    max: 0.3
}, 'landsat 7, 2020');
```

Code Checkpoint A12e. The book's repository contains a script that shows what your code should look like at this point.

The next step is probably the most time consuming in this exercise. In supervised classification, a classifier is trained with labeled examples to predict the labels of new examples. In our case, each training example will be a pixel, labeled as either built-up or not-built-up. Rather than selecting pixels for the training set one at a time as was done in Chap. F2.2, you will hand-draw polygons (rectangles) and label them as built-up or not-built-up, and assign to each pixel with the label of the spatially overlapping polygon. You will create two feature collections: one with 50 rectangles over built-up areas

(labeled as "1") and one with 50 examples of rectangles over not-built-up areas (labeled as "0"). To do this, follow these steps:

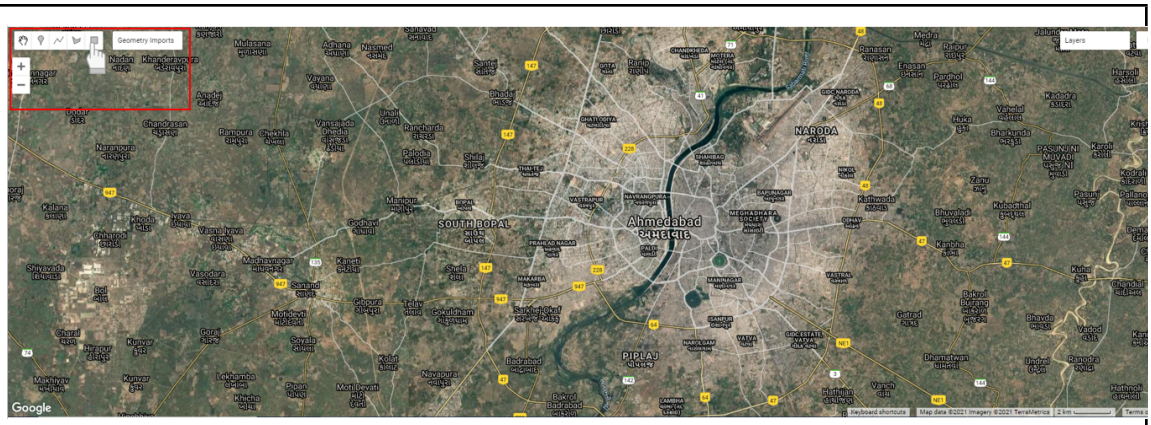


Fig. A1.2.7 To hand-draw examples of not-built-up rectangles, click on the rectangle icon in the upper-left corner of the screen

Start by hand-drawing the not-built-up examples. Click on the rectangle icon in the upper-left corner of the screen (Fig. A1.2.7). Clicking on this icon will create a new variable (called `geometry`). Click on the gear-shaped icon of this imported variable. Change the name of the variable to `nbu` (stands for "not-built-up") (Fig. A1.2.8a). Change the **Import as** parameter from **Geometry** to **FeatureCollection** (Fig. A1.2.8b). Click on the **+Property** button (Fig. A1.2.8c), to create a new property called "class," and assign it with a value of 0 (Fig. A1.2.8d). You have just created a new empty `FeatureCollection`, where each feature will have a property called "class", with a value of 0 assigned to it.

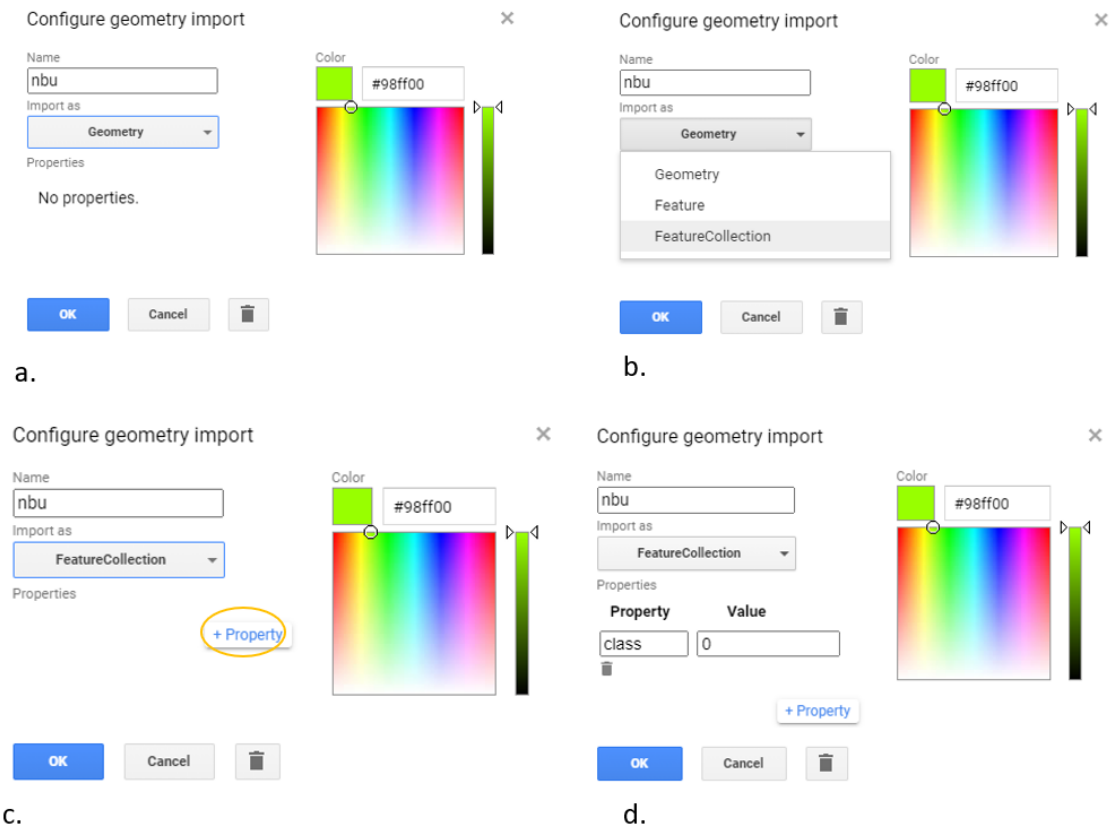


Fig. A1.2.8 Creating a new `FeatureCollection` for examples of not-built-up areas: (a), change the name of the variable to `nbu`; (b), change the parameter **Import as** parameter from **Geometry** to **FeatureCollection**; (c), click on the **+Property** button; (d) to create a new property called "class," and assign to it the value 0.

The `nbu` `FeatureCollection` is now initialized, and empty. You now need to populate it with features:. In this exercise, you will draw 50 rectangles drawn over areas representing not-built-up locations. To identify these areas, you can use either the 2020 Landsat 7 scene (`landsat7_2020`) as the background reference, or use the high-resolution satellite basemap (provided in Earth Engine) as reference. Note, however, that you can use the basemap as reference only if you classify the current land cover, since Earth Engine's satellite background is continually updated.

For this exercise, we define "not-built-up" land cover as any type of land cover that is not built-up (this includes vegetation, bodies of water, bare land, etc.). Click

on the `nbu` layer you created under **Geometry Imports** (Fig. A1.2.9a) to select it (the rectangle button will be highlighted; see Fig. A1.2.9b) and draw 50 rectangles representing examples of areas that are not-built-up (Fig. A1.2.9c). It is important to select diverse and representative examples. To avoid exceeding the user memory limit, draw relatively small rectangles (see examples in Fig. A1.2.9).

Tip: To delete a feature, simply click on **Esc**, select the feature you want to delete, and click **Delete**. You can also click on **Esc** if you want to navigate (zoom in, zoom out, pan). If you'd like to re-draw a rectangle, click again on the name **nbu** layer under **Geometry Imports** and modify the geometry. When done, click on the **Exit** button or **Esc**.

Note that while adding more examples may improve the accuracy of your classification, creating too many examples could potentially improve the accuracy of the classification of your specific area of interest—but it may not fit well in other geographical areas. That is an example of a type of "overfitting," in which samples used to train a model are overly specific to one area and not generalizable to other locations.

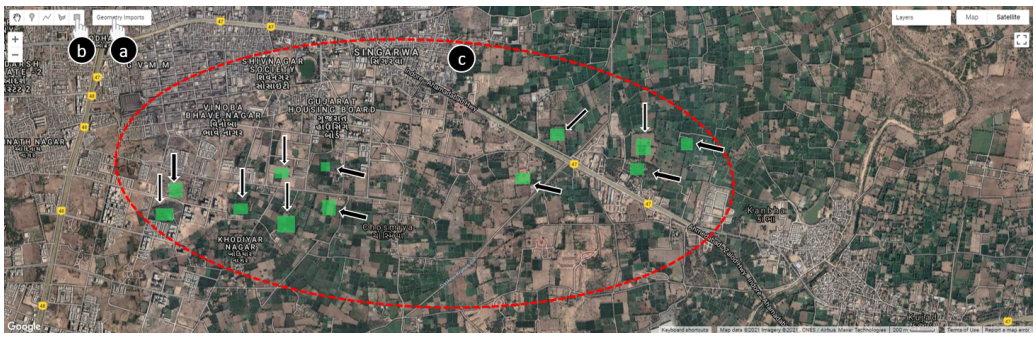


Fig. A1.2.9 Rectangles representing example locations of not-built-up areas

Now that you created 50 examples of not-built-up areas, it is time to create a second additional `FeatureCollection` of 50 examples of built-up areas. As with the not-built-up examples, you will first create an empty `FeatureCollection` and then populate it with example features.

Important: you will need to create a new layer. Make sure that you do not add these new examples to the not-built-up `FeatureCollection`. To create a new

layer, click on **+new layer** (Fig. A1.2.10). This will create a new variable called `geometry`.

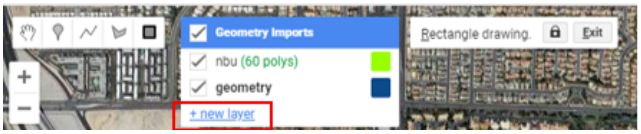


Fig. A1.2.10 Creating a new layer for the built-up `FeatureCollection` To create a new empty feature collection of built up examples, click on the **+new layer** (at the bottom of the **Geometry Imports**)

Click on the gear-shaped icon (⚙) next to the new variable you just created. Change the name of the variable from `geometry` to `bu` (stands for built-up) (Fig. A1.2.11a). Change the **Import as** parameter from **geometry** to **FeatureCollection** (Fig. A1.2.11b). Click on **+Property** (Fig. A1.2.11c), and create a property called `class` (Fig. A1.2.11d), and assign to it a value of 1. (Fig. A1.2.11e). Recall that this is the same property name you used for the `nbu` examples. Assign the features with a value "1" (Fig. A1.2.11e). Also change the color of the `bu` examples to red (Fig. A1.2.11f), representing built-up areas. Click **OK**.

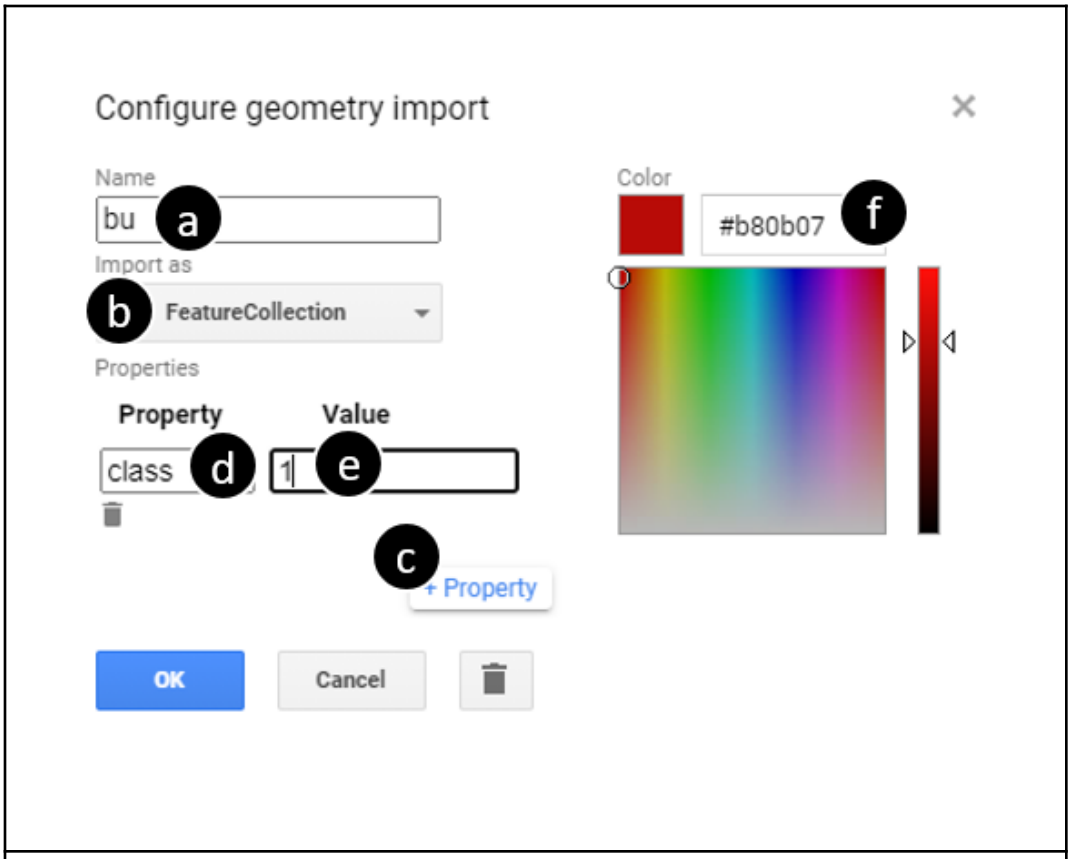


Fig. A1.2.11 Creating a new `FeatureCollection` for examples of built-up areas: (a), change the name of the variable to "bu"; (b), change the parameter **Import as** parameter from **Geometry** to **FeatureCollection**; (c), click on the **+Property** button; (d) to create a new property called "class," and assign to it the value 1.

The `bu` `FeatureCollection` is now initialized, and empty. To populate it with examples of built-up areas, hover over the **bu** `FeatureCollection` (Fig. A1.2.12a), select the rectangle tool (Fig. A1.2.12b), and draw 50 rectangles over built-up areas. Keep (keep the size of the rectangles similar to the not-built-up examples) (Fig. A1.2.12, location 'c').

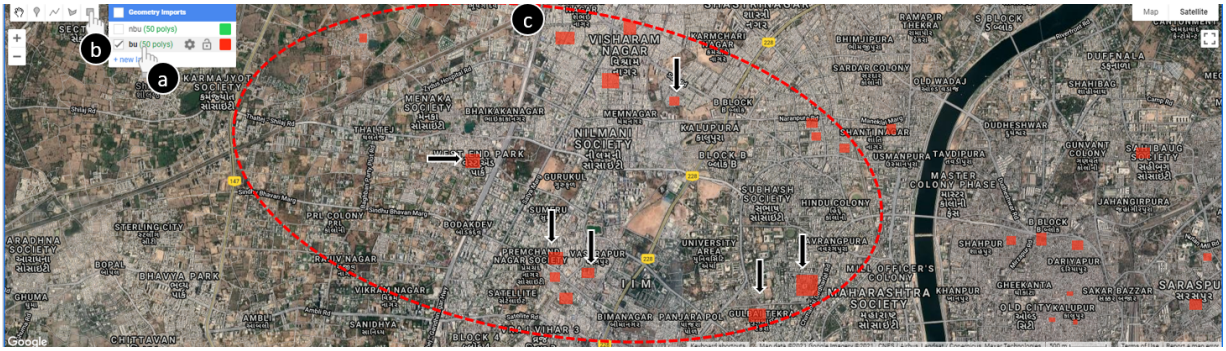


Fig. A1.2.12 Rectangles representing examples of built-up areas. To create these examples, hover over the new "bu" feature collection you just created (a), select the rectangle tool (b), and draw 50 examples of rectangles over built-up areas (c).

You've now created two feature collections: 50 features representing built-up areas (the variable is called `bu`), and 50 features representing not-built-up areas (the variable is called `nbu`). Next, you will merge these two feature collections to create one collection of 100 features. Remember that in both collections you added a property called `class`; features in the `FeatureCollection` `nbu` were assigned a value of 0 and features in the `FeatureCollection` `bu` were assigned a value of 1. The merged `FeatureCollection` will also include the property `class`: 50 features with a value of 1 and 50 with a value of 0. To merge the feature collections, use the method `merge`. Call the merged `FeatureCollection` with the variable `lc`.

```
var lc = nbu.merge(bu);
```

Code Checkpoint A12f. The book's repository contains a script that shows what your code should look like at this point.

List the properties that the classifier will use to determine whether a pixel is built-up or not-built-up. Recall that Landsat 7 has 11 bands; two of them are QA Bitmask bands, which likely do not vary by land cover. Create a new variable called `bands` with a list of the bands to be used by the classifier.

```
var bands = ['SR_B1', 'SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'ST_B6',
'SR_B7'];
```

Next, you will create the labeled examples (pixels) that will serve as the training set of the classifier. Remember that, at this point, the pixels are not yet labeled; you labeled only the rectangles (stored in the `lc` variable), which are not associated with the bands of the Landsat pixels. To associate the Landsat pixels with a label, you will sample the pixels that overlap with these rectangles and assign them with the class of the overlapping rectangle.

This can be done with the method `sampleRegions`. Define a new variable and call it `training`. Under this variable, call the 2020 Landsat composite (`landsat7_2020`), select the relevant bands (the variable `bands`), and sample the overlapping pixels. You will sample the `landsat7_2020` pixels that overlap with the `FeatureCollection` `lc` and copy the property `class` from each feature to the overlapping pixel (each pixel will now include the 9 properties and a class, either 1 or 0). Set the argument `scale` to `30` (30 m— Landsat 7's spatial resolution—is the scale of the projection to sample in). These steps are shown in the code block below and were described in Section F2.1.

```
var training = landsat7_2020.select(bands).sampleRegions({
    collection: lc,
    properties: ['class'],
    scale: 30
});
```

Now it is time to create the classifier. Create an empty Random Forest classifier using the method `ee.Classifier.smileRandomForest`. You can keep most of the arguments' values as default and only set the number of decision trees in the forest (the argument `numberOfTrees`) to 20. Use the method `train` to train the classifier. This method trains a classifier on a collection of features, using the specified numeric properties of each feature as training data. The collection to train on is called `training` and the name of the property containing the class value is called `'class'`. The list of properties (`inputProperties`) to be considered by the classifier is stored in the `bands` variable. Note that, by default, the classifier predicts the class of the pixel (in this case, "1," built-up, or "0," not-built-up). You could also set the mode of the classifier to `PROBABILITY`, which will result in the probability that a pixel has the value "1". In this example, we used the default `CLASSIFIER` mode).

```
// Create a random forest classifier with 20 trees.
```

```
var classifier = ee.Classifier.smileRandomForest({
    numberOfTrees: 20
}).train({ // Train the classifier.
    // Use the examples we got when we sampled the pixels.
    features: training,
    // This is the class that we want to predict.
    classProperty: 'class',
    // The bands the classifier will use for training and
classification.
    inputProperties: bands
});
```

In the previous step, you trained a classifier. Next, you will use the trained model to predict that class on new pixels, using the method `classify`. You'll define a new variable, `classified20`, which will take the Landsat composite; select the relevant bands (the variable `bands`); and classify this image using the trained classifier (called `'classifier'`).

```
// Apply the classifier on the 2020 image.
var classified20 = landsat7_2020.select(bands).classify(classifier);
```

You can visualize the classification by adding the classified image to the map. In our case, we have only two classes (built-up and not-built-up); thus, the classifier's prediction is binary (either "1" for built-up, or "0" for not-built-up). To show only the pixels with a value of 1, use the `mask` method to mask out the pixels with a value of 0. Set the visualization parameters to visualize the remaining pixels as red, with a transparency of 60%.

```
Map.addLayer(classified20.mask(classified20), {
    palette: ['#ff4218'],
    opacity: 0.6
}, 'built-up, 2020');
```

Code Checkpoint A12g. The book's repository contains a script that shows what your code should look like at this point.

Lastly, many applications require mapping the extent of built-up land cover at more than one point in time (e.g., to understand urbanization processes). As you recall, Landsat 7 has been collecting imagery from every location on Earth since 1999. We can use our trained classifier—which was trained based on Landsat 7 2020 imagery—to predict the extent of built-up land cover in any year collected (with the assumption that the characteristics of a built-up pixel did not change over time). In this exercise, you will use the trained classifier to map the built-up land cover in 2010.

First, create a composite for 2010 using the `simpleComposite` method.:

```
var landsat7_2010 = L7.filterDate('2010-01-01', '2010-12-31')
    .map(maskL457sr).median();
```

Now, you can use the classifier to classify the 2010 image using the `classify` method. Classify the 2010 image (`landsat7_2010`) and add it to the map, this time in a yellow color.

```
// Apply the classifier to the 2010 image.
var classified10 = landsat7_2010.select(bands).classify(
    classifier);
Map.addLayer(classified10.mask(classified10), {
    palette: ['#f1ff21'],
    opacity: 0.6
}, 'built-up, 2010');
```

The result should be something like what is presented in Fig. A1.2.13. Built-up land cover in 2020 is presented in red, and in 2010 in yellow.

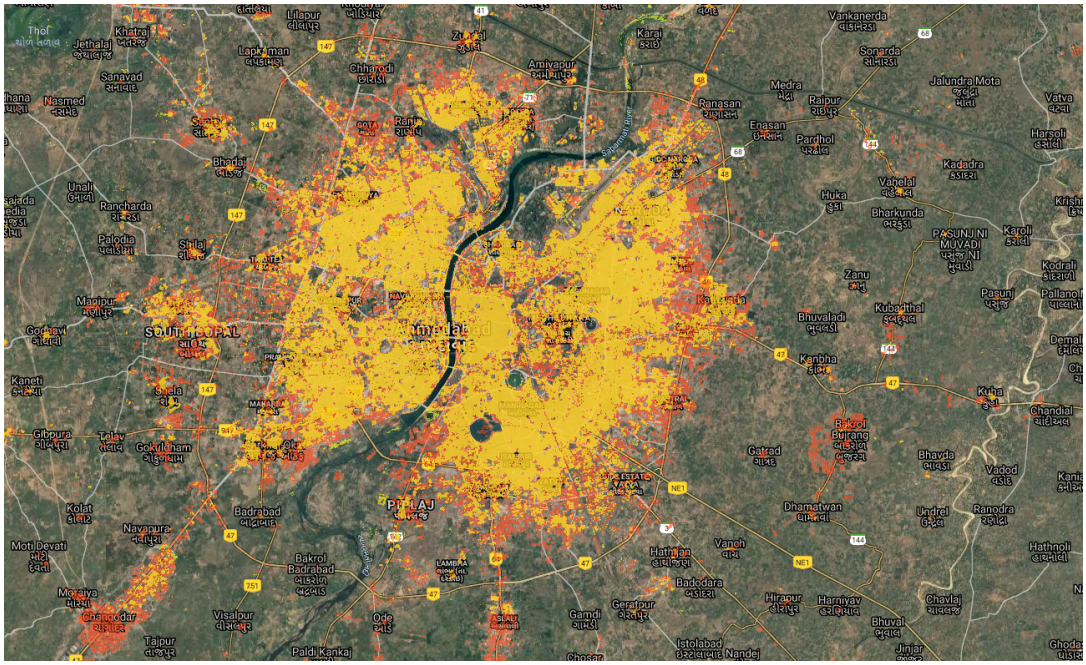


Fig. A1.2.13 The classified built-up land cover in 2020 (red) and in 2010 (yellow)

Want to see which areas were developed in the period between 2010 and 2020? Create a new variable and call it `difference`. In this variable, subtract the values of 2010 from 2020. Any pixel that changed from "0" in 2010 to "1" in 2020 will be assigned with a value of 1. You can then visualize this difference on the map in blue.

```
var difference = classified20.subtract(classified10);

Map.addLayer(difference.mask(difference), {
    palette: ['#315dff'],
    opacity: 0.6
}, 'difference');
```

Code Checkpoint A12h. The book's repository contains a script that shows what your code should look like at this point.

If your code worked as expected, you should be able to see the three layers you classified: the built-up land cover in 2010, in 2020, and the difference between the two.

Question 4. Calculate the total built-up land cover in Ahmedabad in 2010 and in 2020. How much built-up area was added to the city? What is the percent increase of this growth?

Note: Use this `FeatureCollection`, filtered to Ahmedabad, to bound your area calculation.:

```
var indiaAdmin3 = ee.FeatureCollection(
    'projects/gee-book/assets/A1-2/IndiaAdmin3');
var ahmedabad = indiaAdmin3.filterMetadata('VARNAME_3', 'equals',
    'Ahmadabad city');
Map.addLayer(ahmedabad, {}, 'Ahmedabad city');
```

Question 5. Add two lines to your code to visualize the built-up area that was added between 2010 and 2020.

Synthesis

Assignment 1. In this exercise, we mapped built-up land cover with Landsat 7 imagery. Now try to repeat this exercise using Sentinel-2 as an input to the classifier. Calculate the total area of built-up land cover in Ahmedabad in 2020, and answer the following questions.

Note, that you can create an annual Sentinel-2 composite by calculating the median value of all cloud-free scenes captured in a given year (see an example of how to filter by cloud values in Sect. 1 of this chapter). Note that the property that stores the cloudy pixel percentage in the Sentinel-2 `ImageCollection` is `'CLOUDY_PIXEL_PERCENTAGE'`.

Question 6. How do the area totals differ from each other? (Use the Ahmedabad `FeatureCollection` boundary from Question 4)?

Question 7. Why do you think that is the case? Think about spatial and spectral resolutions.

Question 8. What is the area of the MODIS urban classification in 2019?

Question 9. Is it larger or smaller than Landsat/Sentinel? Why?

1. How do the area totals differ from each other? (Use the Ahmedabad FeatureCollectionfeature collection boundary from Question 4 above.)

2. Why do you think that is the case? Think about spatial and spectral resolutions.

3. What is the area of the MODIS urban classification in 2019?

4. Is it larger or smaller than Landsat/Sentinel? Why?

Conclusion

In this chapter, you have learned how to visualize and quantify the magnitude and pace of urbanization anywhere in the world. Combined with your knowledge from other chapters, you're now equipped to look at urban heat islands, food and water system stresses, and many other socio-environmental impacts of urbanization (Bazaz et al. 2018). As more and higher-resolution remote sensing data become available, the scope of questions we can answer about urban areas will widen and the accuracy of our predictions will improve. This knowledge can be used for policy and decision-making, in particular in the context of the United Nations Sustainable Development Goals (United Nations 2020, Prakash et al. 2020).

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Bazaz A, Bertoldi P, Buckeridge M, et al (2018) Summary for Urban Policymakers – What the IPCC Special Report on 1.5°C Means for Cities. IIHS Indian Institute for Human Settlements

Breiman L (2001) Random forests. Mach Learn 45:5–32. https://doi.org/10.1023/A:1010933404324

Goldblatt R, Stuhlmacher MF, Tellman B, et al (2018) Using Landsat and nighttime lights for supervised pixel-based image classification of urban land cover. Remote Sens Environ 205:253–275. https://doi.org/10.1016/j.rse.2017.11.026

Goldblatt R, You W, Hanson G, Khandelwal AK (2016) Detecting the boundaries of urban areas in India: A dataset for pixel-based image classification in Google Earth Engine. Remote Sens 8:634. https://doi.org/10.3390/rs8080634

Prakash M, Ramage S, Kavvada A, Goodman S (2020) Open Earth observations for sustainable urban development. Remote Sens 12:1646. https://doi.org/10.3390/rs12101646

United Nations Department of Economic and Social Affairs (2019) World Urbanization Prospects: The 2018 Revision

United Nations Department of Economic and Social Affairs (2020) The Sustainable Development Goals Report 2020

Chapter A1.3: Built Environments

Author
Erin Trochim

Overview

The built environment consists of the human-made physical parts of the environment, including homes, buildings, streets, open spaces, and infrastructure. This chapter will focus on analyzing global infrastructure datasets.

Learning Outcomes

- Quantifying road characteristics.
- Comparing road and transmission line distributions.
- Contrasting changes in impervious surfaces with flooding.
- Understanding vector-based versus raster-based approaches.

Helps if you know how to:

- Create a function for code reuse (Chap. F1.0).
- Summarize an `ImageCollection` with reducers (Chap. F4.0, Chap. F4.1).
- Filter a `FeatureCollection` to obtain a subset (Chap. F5.0, Chap. F5.1).
- Convert between raster and vector data (Chap. F5.1).
- Join elements of vector datasets together (Chap. F5.3).

Introduction to Theory

Until the recent past, data about the built environment were mostly produced and managed at local and regional levels, and global datasets were very rare. Municipal, state, and national governments require easy access to data in order to plan, build, and maintain assets, which can be either privately or publicly owned. In the last decade, there has been a push to produce globally consistent data to support economic assessment and environmental transitions. The existence of this data makes it possible to explore the impact of factors including weather, climate, and growth over time.

Built environment datasets currently available in the Earth Engine Data Catalog include the Global Power Plant Database, Open Buildings V1 Polygons (which currently

includes over half of Africa), and Global Artificial Impervious Area. Complementary environmental information includes the Global Flood Database. In addition, the Awesome GEE Community Datasets hosts the Global Roads Inventory Project, Global Power System, and Global Fixed Broadband and Mobile (Cellular) Network Performance collections.

As will be seen in the rest of this chapter, information on the built environment is most often stored as vector data: that is, as points, lines, and polygons. This format readily lends itself to representing, for example, bridges as points, roads as lines, and buildings as polygons. Many of the built environment datasets listed above now have both raster and vector components. It is useful to understand the utility of each of these formats and how to extract information between them and ancillary datasets.

Practicum

Section 1. Road Characteristics

We will start by calculating road length using the Global Roads Inventory Project (GRIP) dataset (Meijer et al. 2018). This dataset was created to provide a consistent global roadways dataset for environmental and biodiversity assessments. For this exercise, we will focus on examining the largest countries in Africa, North America, and Europe.

Start by importing the `grip4` datasets into Earth Engine using the following code, and examine how they look in the Code Editor.

```
// Import roads data.
var grip4_africa = ee.FeatureCollection(
        'projects/sat-io/open-datasets/GRIP4/Africa'),
    grip4_north_america = ee.FeatureCollection(
        'projects/sat-io/open-datasets/GRIP4/North-America'),
    grip4_europe = ee.FeatureCollection(
        'projects/sat-io/open-datasets/GRIP4/Europe');

// Check the roads data sizes.
print('Grip4 Africa size', grip4_africa.size());
print('Grip4 North America size', grip4_north_america.size());
print('Grip4 Europe size', grip4_europe.size());
```

```
// Display the roads data.
Map.addLayer(ee.FeatureCollection(grip4_africa).style({
    color: '413B3A',
    width: 1
}), {}, 'Grip4 Africa');
Map.addLayer(ee.FeatureCollection(grip4_north_america).style({
    color: '413B3A',
    width: 1
}), {}, 'Grip4 North America');
Map.addLayer(ee.FeatureCollection(grip4_europe).style({
    color: '413B3A',
    width: 1
}), {}, 'Grip4 Europe');
```

Using the Large Scale International Boundary dataset from the United States Office of the Geographer, below we will import the simplified country boundaries. After calculating the area enclosed by the boundary of each country, we will select Algeria, a quite large country with many roads.

```
// Import simplified countries.
var countries = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017');

// Add a function to calculate the feature's geometry area.
// Add the function as a property.
var addArea = function(feature) {
    return feature.set({
        areaKm: feature.geometry().area().divide(1000 *
            1000)
    }); // km2 squared
};

// Map the area getting function over the FeatureCollection.
var countriesArea = countries.map(addArea);

// Filter to the largest country in Africa.
var Algeria = countriesArea.filter(ee.Filter.inList('country_na', [
    'Algeria'
```

```
]));

// Display the selected countries.
Map.addLayer(Algeria.style({
    fillColor: 'b5ffb4',
    color: '00909F',
    width: 1.0
}), {}, 'Algeria');
```

Next, we will calculate the road density for Algeria. For this example, we will implement a function to join the roads to each of the countries. The approach below joins the roads to the countries if there is spatial overlap between the features. Then, for each joined road in the specified country, an intersection is performed to keep only the portion of the road within the country. The length of the road is calculated and added as a property per feature, as is the road density per country.

```
// This function calculates the road length per country for the
associated GRIP dataset.
var roadLength4Country = function(country, grip4) {

    // Join roads to countries.
    var intersectsFilter = ee.Filter.intersects({
        leftField: '.geo',
        rightField: '.geo',
        maxError: 10
    });

    var grip4Selected = grip4.filterBounds(country);

    var countriesWithRds = ee.Join.saveAll('roads').apply({
        primary: country,
        secondary: grip4Selected,
        condition: intersectsFilter
    }).filter(ee.Filter.neq('roads', null));

    // Return country with calculation of roadLength and roadsPerArea.
    return countriesWithRds.map(function(country) {
```

```
        var roadsList = ee.List(country.get('roads'));
        var roadLengths = roadsList.map(function(road) {
            return ee.Feature(road).intersection(
                country, 10).length(10);
        });
        var roadLength = ee.Number(roadLengths.reduce(ee
            .Reducer.sum()));
        return country.set({
            roadLength: roadLength.divide(
                1000), // Convert to km.
            roadsPerArea: roadLength.divide(ee
                .Number(country.get('areaKm'))
            )
        });
    }).select(['country_na', 'areaKm', 'roadLength',
        'roadsPerArea'
    ]);
};

// Apply the road length function to Algeria.
var roadLengthAlgeria = roadLength4Country(Algeria, grip4_africa);
```

Print the roads per area in Algeria.

```
// Print the road statistics for Algeria.
print('Roads statistics in Algeria', roadLengthAlgeria);
```

Question 1. How many roads are there in Africa? Hint: Check the size of the `grip4` Africa dataset.

Question 2. Which continent has the most roads: Africa, Europe, or North America?

Question 3. How many roads per square kilometer are there in Algeria?

Question 4. What is the total road length in kilometers in Algeria?

Code Checkpoint A13a. The book's repository contains a script that shows what your code should look like at this point.

Earth Engine is powerful, but its processing power is not infinite. If you try applying the same code to countries such as Canada and France, it doesn't run very well interactively due to the enormous size of the request. To successfully execute long-running, complex tasks, you can export results. Exporting forces Earth Engine to continue to run for a very long time; in that case, you can't see the results interactively, but you will be able to get an answer for even very large problems, and then load the exported data. An example of this is shown below.

```
// Export feature collection to drive.
Export.table.toDrive({
    collection: roadLengthAlgeria,
    description: 'RoadStatisticsforAlgeria',
    selectors: ['country_na', 'roadLength', 'roadsPerArea']
});
```

Let's think about how to simplify this analysis, to see if we can optimize the computation in an interactive environment. Print the first feature of the `grip4` Africa feature collection and display it using the following commands.

```
// Print the first feature of the grip4 Africa feature collection.
print(grip4_africa.limit(1));

Map.setCenter(-0.759, 9.235, 6);
Map.addLayer(grip4_africa.limit(1),
    {},
    'Road length comparison');
```

You can find the road in northern Ghana as shown in Fig. A1.3.1.

Fig. A1.3.1 Exploring the GRIP datasets by visualizing the first feature in the Africa region. According to the feature properties, the road is 0.76, but the unit is not specified. Compare this to the scale at the bottom of the **Map** panel in the Code Editor.

In the **Console**, examine the properties of the road. Notice there is an existing column called `Shape_Leng`. The length of each road appears to have already been precalculated and included as a value. The length of this road is given as 0.76, but the unit is not specified. This looks suspicious in comparison to the scale—the road is obviously longer than 0.76 meters, kilometers, or miles. It is easy to recalculate the length of the lines: use the following code to do so and compare the results.

```
// This function adds line length in km.
var addLength = function(feature) {
    return feature.set({
        lengthKm: feature.length().divide(1000)
    });
};

// Calculate the line lengths for all roads in Africa.
var grip4_africaLength = grip4_africa.map(addLength);

// Compare with other values.
print('Calculated road length property', grip4_africaLength.limit(1));
```

The road in Ghana has a new calculated length of about 84 km, which looks accurate.

Repeat the road calculation analysis, but examine each step. First, filter the road data in Africa for Algeria. Visualize the result, taking note of whether any roads are included that originate in Algeria but extend into other countries. Then, reduce the `lengthKm` column and calculate the sum in the filtered Algeria road data.

```
// Repeat the analysis to calculate the length of all roads.
// Filter the table geographically: only keep roads in Algeria.
var grip4_Algeria = grip4_africaLength.filterBounds(Algeria);

// Visualize the output.
Map.addLayer(grip4_Algeria.style({
    color: 'green',
    width: 2.0
}), {}, 'Algeria roads');

// Sum the lengths for roads in Algeria.
var sumLengthKmAlgeria = ee.Number(
    // Reduce to get the sum.
    grip4_Algeria.reduceColumns(ee.Reducer.sum(), ['lengthKm'])
    .get('sum')
);

// Print the result.
print('Length of all roads in Algeria', sumLengthKmAlgeria);
```

Question 5. What is the total recalculated road length in kilometers in Algeria?

Question 6. Is this higher or lower than the first value?

Code Checkpoint A13b. The book's repository contains a script that shows what your code should look like at this point.

The difference in values when recalculating is due to extra roads being included in the second calculation. This was due to skipping the joining and intersection of the Algeria

feature collection. Using only `filterBounds` to approximate the road limits was not as accurate.

Let's try another method to make this more computationally efficient. Rasterize the roads by interpolating the current feature collection into an image, and use `reduceRegions` to calculate the area of the road pixels. For this exercise, set the scale as 100 m. In order to convert the area into the appropriate dimension and units, divide the results by 1000 and apply a square root to transform the length into kilometers.

```
// Repeat the analysis again to calculate length of all roads using
rasters.
// Convert to raster.
var empty = ee.Image().float();

var grip4_africaRaster = empty.paint({
    featureCollection: grip4_africaLength,
    color: 'lengthKm'
}).gt(0);

Map.addLayer(grip4_africaRaster, {
    palette: ['orange'],
    max: 1
}, 'Rasterized roads');

// Add reducer output to the features in the collection.
var AlgeriaRoadLength = ee.Image.pixelArea()
    .addBands(grip4_africaRaster)
    .reduceRegions({
        collection: Algeria,
        reducer: ee.Reducer.sum(),
        scale: 100,
    }).map(function(feature) {
        var num = ee.Number.parse(feature.get('area'));
        return feature.set('length', num.divide(1000).sqrt()
            .round());
    });
```

```
// Print the first feature to illustrate the result.
print('Length of all roads in Algeria calculated via rasters', ee
    .Number(AlgeriaRoadLength.first().get('length')));
```

Notice this value is about 13%, or 7000 km, lower than the first estimate.

Take a look at the rasterized roads visualization. With no scale set, it looks very similar to the vectors. Zoom in to an area like the example in Fig. A1.3.2 and examine the structure of the vectors, rasters, and roads themselves. Note that, in some areas, divided roads are represented by two separate features (vectors). The initial rasters match the scale of the vector. One 100 m pixel, though, would cover both roads and represent only a single unit length. Understanding how the data represents the actual built environment is critical for accuracy and precision estimates. There is also a tradeoff in computation time and whether the calculations can be performed on the fly.

Fig. A1.3.2 Examining roads from the GRIP dataset as features (in green) versus rasters (in orange) overlaid with transparency on the **Map** view. There can be offset between datasets and different spatial shapes (lines, polygons, and rasters).

The advantage to this approach is that the analysis can be performed interactively across much larger areas. We will test this by calculating total road length for the largest countries in Africa, North America, and Europe, which are Algeria, Canada, and France.

```
// Calculate line lengths for all roads in North America and Europe.
var grip4_north_americaLength = grip4_north_america.map(addLength);
var grip4_europeLength = grip4_europe.map(addLength);

// Merge all vectors.
var roadLengthMerge = grip4_africaLength.merge(
    grip4_north_americaLength).merge(grip4_europeLength);

// Convert to raster.
var empty = ee.Image().float();

var roadLengthMergeRaster = empty.paint({
    featureCollection: roadLengthMerge,
    color: 'roadsPerArea'
}).gt(0);

// Filter to largest countries in Africa, North America and Europe.
var countriesSelected = countries.filter(ee.Filter.inList(
    'country_na', ['Algeria', 'Canada', 'France']));

// Clip image to only countries of analysis.
var roadLengthMergeRasterClipped = roadLengthMergeRaster
    .clipToCollection(countriesSelected);

// Add reducer output to the features in the collection.
var countriesRoadLength = ee.Image.pixelArea()
    .addBands(roadLengthMergeRasterClipped)
    .reduceRegions({
        collection: countriesSelected,
        reducer: ee.Reducer.sum(),
        scale: 100,
    }).map(function(feature) {
        var num = ee.Number.parse(feature.get('area'));
        return feature.set('length', num.divide(1000).sqrt()
            .round());
    });
```

```
// Compute totaled road lengths in km, grouped by country.
print('Length of all roads in Canada', countriesRoadLength.filter(ee
    .Filter.equals('country_na', 'Canada')).aggregate_sum(
    'length'));
print('Length of all roads in France', countriesRoadLength.filter(ee
    .Filter.equals('country_na', 'France')).aggregate_sum(
    'length'));
```

Question 7. Which country has the highest total length of roads?

Question 8. Explore the effect of the scale value by reprojecting the rasterized roads (for example, `Map.addLayer(grip4_africaRaster.reproject({crs: 'EPSG:4326', scale: 100}), {palette: ['orange'], max: 1}, 'Rasterized roads 100 m')`). Does increasing the scale result in an over- or underestimation of roads? What is an optimal scale to still allow the large calculations to run in real time?

Code Checkpoint A13c. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Road and Transmission Line Comparison

Next, let's compare the overlap between roads and transmission lines, which often are found in the same vicinity. We will test this concept by examining the Global Power System data (Arderne et al. 2020).

Let's start by creating a new script and importing data into the Code Editor. As in the first exercise, import the GRIP road datasets. We will reuse the same `addLength` function from the previous exercise and apply it to roads in Africa. Then, convert the roads to a raster using length in kilometers as the pixel value.

```
// Import roads data.
var grip4_africa = ee.FeatureCollection(
        'projects/sat-io/open-datasets/GRIP4/Africa'),
    grip4_europe = ee.FeatureCollection(
        'projects/sat-io/open-datasets/GRIP4/Europe'),
    grip4_north_america = ee.FeatureCollection(
```

```
        'projects/sat-io/open-datasets/GRIP4/North-America');

// Add a function to add line length in km.
var addLength = function(feature) {
    return feature.set({
        lengthKm: feature.length().divide(1000)
    }); // km;
};

// Calculate line lengths for all roads in Africa.
var grip4_africaLength = grip4_africa.map(addLength);

// Convert the roads to raster.
var empty = ee.Image().float();

var grip4_africaRaster = empty.paint({
    featureCollection: grip4_africaLength,
    color: 'lengthKm'
});
```

Next, import the simplified countries again and filter the feature collection, this time to all countries on the continent of Africa.

Import the global power system data that represents transmission lines. For this exercise, we will use both the OpenStreetMap and predicted values in this dataset. Validation of this dataset included several parts of Africa, so it tends to have higher accuracy here than in other parts of the world like the Arctic.

Apply a filter to the transmission lines data to limit the analysis to Africa. Calculate the length of the lines and convert it to rasters, as with the roads data.

```
// Import simplified countries.
var countries = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017');

// Filter to Africa.
var Africa = countries.filter(ee.Filter.eq('wld_rgn', 'Africa'));
```

```
// Import global power transmission lines.
var transmission = ee.FeatureCollection(

'projects/sat-io/open-datasets/predictive-global-power-system/distribu
tion-transmission-lines'
);

// Filter transmission lines to Africa.
var transmissionAfrica = transmission.filterBounds(Africa);

// Calculate line lengths for all transmission lines in Africa.
var transmissionAfricaLength = transmissionAfrica.map(addLength);

// Convert the transmission lines to raster.
var transmissionAfricaRaster = empty.paint({
    featureCollection: transmissionAfricaLength,
    color: 'lengthKm'
});
```

Question 9. In this exercise, we are rasterizing our feature collections to calculate spatial association indicators. If the data were left as a feature collection, what would be the simplest way of comparing the two datasets in a similar way?

Code Checkpoint A13d. The book's repository contains a script that shows what your code should look like at this point.

In order to compare the road and transmission line rasters, they need to be stacked together into a single image. It is a good idea to rename the bands to keep track of the input values. Clip this dataset to the Africa feature collection in order to minimize computation in the following steps.

```
// Add roads and transmission lines together into one image.
// Clip to Africa feature collection.
var stack = grip4_africaRaster
    .addBands(transmissionAfricaRaster)
    .rename(['roads', 'transmission'])
```

```
    .clipToCollection(Africa);
```

Next, use the following code to calculate the spatial statistics' local Geary's C (Anselin 2010). This will indicate clustering due to spatial autocorrelation. In this case, we are comparing whether roads and transmission lines are located together. Geary's C values of −1 indicate dispersion, while those of 1 indicate clustering.

```
// Calculate spatial statistics: local Geary's C.
// Create a list of weights for a 9x9 kernel.
var list = [1, 1, 1, 1, 1, 1, 1, 1, 1];

// The center of the kernel is zero.
var centerList = [1, 1, 1, 1, 0, 1, 1, 1, 1];

// Assemble a list of lists: the 9x9 kernel weights as a 2-D matrix.
var lists = [list, list, list, list, centerList, list, list, list,
    list
];

// Create the kernel from the weights.
// Non-zero weights represent the spatial neighborhood.
var kernel = ee.Kernel.fixed(9, 9, lists, -4, -4, false);

// Use the max among bands as the input.
var maxBands = stack.reduce(ee.Reducer.max());

// Convert the neighborhood into multiple bands.
var neighs = maxBands.neighborhoodToBands(kernel);

// Compute local Geary's C, a measure of spatial association
// - 0 indicates perfect positive autocorrelation/clustered
// - 1 indicates no autocorrelation/random
// - 2 indicates perfect negative autocorrelation/dispersed
var gearys = maxBands.subtract(neighs).pow(2).reduce(ee.Reducer.sum())
    .divide(Math.pow(9, 2));

// Convert to a -/+1 scale by: calculating C* = 1 - C
```

```
// - 1 indicates perfect positive autocorrelation/clustered
// - 0 indicates no autocorrelation/random
// - -1 indicates perfect negative autocorrelation/dispersed
var gearysStar = ee.Image(1).subtract(gearys);
```

Examine the output by creating a custom palette and using a high-contrast basemap. The example illustrated in Fig. A1.3.3 shows how to add a focal max to the final image in order to better visualize the results as a raster. The color blue indicates values toward −1 and therefore dispersion, while red represents values toward 1 and therefore spatially clustered.

```
// Import palettes.
var palettes = require('users/gena/packages:palettes');

// Create custom palette, blue is negative while red is positive
autocorrelation/clustered.
var palette = palettes.colorbrewer.Spectral[7].reverse();

// Normalize the image and add it to the map.
var visParams = {
    min: -1,
    max: 1,
    palette: palette
};

// Display the image.
Map.setCenter(19.8638, -34.5705, 10);
Map.addLayer(gearysStar.focalMax(1), visParams, 'local Gearys C*');
```

Fig. A1.3.3 Results of Geary's C* spatial autocorrelation between roads and transmission lines in South Africa. Red indicates clustering, while blue indicates dispersion. Shorter stretches in this example appear to have more co-located linear infrastructure.

Question 10. What are the characteristics of the roads and transmission lines that are clustered?

Question 11. In this exercise, we are rasterizing our feature collections to calculate local indicators of spatial association. If the data were left as a feature collection, what would be the simplest way of comparing the two datasets?

Code Checkpoint A13e. The book's repository contains a script that shows what your code should look like at this point.

Save your script for your own future use, as outlined in Chap. F1.0. Then, refresh the Code Editor to begin with a new script for the next section.

Section 3. Impervious Surfaces and Flooding

For our final exercise, we will compare impervious surfaces and flooding over time in a new script. The Global Artificial Impervious Area dataset captures annual change

information at a 30 m resolution of impervious surface area from 1985 to 2018 (Gong et al. 2020). Impervious surfaces can include a variety of built environment surfaces, including anything with pavement or water-resistant materials—roads, sidewalks, airports, parking lots, ports, distribution centers, rooftops, etc. Artificial impervious areas are important because they are a clear representation of human settlements (Gong et al. 2020).

We will compare these impervious areas with the Global Flood Database, which describes flood extent and population characteristics for 913 large flood events from 2000 to 2018 at 250 m resolution (Tellman et al. 2021).

In this example, let's compare the area of impervious surfaces in 2000 versus 2018 over the satellite-observed historical floodplain. Below, we will load the Global Artificial Impervious Area dataset, to display the images and compare the data for the two dates.

```
// Import Tsinghua FROM-GLC Year of Change to Impervious Surface
var impervious = ee.Image('Tsinghua/FROM-GLC/GAIA/v10');

// Use the change year values found in the band.
// The change year values is described here:
//
https://developers.google.com/earth-engine/datasets/catalog/Tsinghua_F
ROM-GLC_GAIA_v10#bands
// Select only those areas which were impervious by 2000.
var impervious2000 = impervious.gte(19);

// Select only those areas which were impervious by 2018.
var impervious2018 = impervious.gte(1);

Map.setCenter(-98.688, 39.134, 5);

// Display the images.
Map.addLayer(
    impervious2000.selfMask(),
    {
        min: 0,
        max: 1,
```

```
        palette: ['014352', '856F96']
    },
    'Impervious Surface 2000');

Map.addLayer(
    impervious2018.selfMask(),
    {
        min: 0,
        max: 1,
        palette: ['014352', '1A492C']
    },
    'Impervious Surface 2018');
```

Subtract the two images to find the change between 2018 and 2000, and again display the results:

```
// Calculate the difference between impervious areas in 2000 and 2018.
var imperviousDiff = impervious2018.subtract(impervious2000);

Map.addLayer(
    imperviousDiff.selfMask(),
    {
        min: 0,
        max: 1,
        palette: ['014352', 'FFBF00']
    },
    'Impervious Surface Diff 2000-18');
```

Import the Global Flood Database. Select the `'flooded'` band and sum all values to create the satellite-observed historical floodplain. Mask out the areas of permanent water in the floodplain using the JRC Global Surface Water dataset included in the Global Flood Database as the `'jrc_perm_water'` band. The JRC data uses the dataset's original `'transition'` band. Display the final floodplain results:

```
// Import the Global Flood Database v1 (2000-2018).
```

```
var gfd = ee.ImageCollection('GLOBAL_FLOOD_DB/MODIS_EVENTS/V1');

// Map all floods to generate the satellite-observed historical flood
plain.
var gfdFloodedSum = gfd.select('flooded').sum();

// Mask out areas of permanent water.
var gfdFloodedSumNoWater = gfdFloodedSum.updateMask(gfd.select(
    'jrc_perm_water').sum().lt(1));

var durationPalette = ['C3EFFE', '1341E8', '051CB0', '001133'];

Map.addLayer(
    gfdFloodedSumNoWater.selfMask(),
    {
        min: 0,
        max: 10,
        palette: durationPalette
    },
    'GFD Satellite Observed Flood Plain');
```

Now we'll calculate which states have been developing the greatest amounts of impervious surfaces in floodplain areas. Start by masking out areas in the `imperviousDiff` image that are not in the floodplains (`gfdFloodedSumNoWater` greater than or equal to 1). Then, we will import the first-order administrative Global Administrative Unit Layers from the UN Food and Agriculture Organization. Filter this feature collection to administrative names (`'ADM0_NAME'`) equal to `'United States of America'`. Create an area image by multiplying the impervious difference flood image by pixel area. Apply a `reduceRegions` reducer to the area image using the United States feature collection and an initial scale of 100 m. Sort the output sum by descending order and get only the five highest states. Print the output.

```
// Mask areas in the impervious difference image that are not in flood
plains.
var imperviousDiffFloods = imperviousDiff
    .updateMask(gfdFloodedSumNoWater.gte(1));
```

```
// Which state has built the most area in the flood plains?
// Import FAO countries with first level administrative units.
var countries = ee.FeatureCollection('FAO/GAUL/2015/level1');

// Filter to the United States.
var unitedStates = countries.filter(ee.Filter.eq('ADM0_NAME',
    'United States of America'));

// Calculate the image area.
var areaImage = imperviousDiffFloods.multiply(ee.Image.pixelArea());

// Sum the area image for each state.
var unitedStatesImperviousDiffFlood = areaImage.reduceRegions({
        collection: unitedStates,
        reducer: ee.Reducer.sum(),
        scale: 100,
    }) // Sort descending.
    .sort('sum', false)
    // Get only the 5 highest states.
    .limit(5);

// Print state statistics for change in impervious area in flood
plain.
print('Impervious-flood change statistics for states in US',
    unitedStatesImperviousDiffFlood);
```

Question 12. Which state built the highest amount of impervious surfaces on satellite-derived floodplains between 2000 and 2018?

Question 13. Which state built the lowest amount of impervious surfaces on satellite-derived floodplains between 2000 and 2018?

Code Checkpoint A13f. The book's repository contains a script that shows what your code should look like at this point.

Synthesis

Assignment 1. Rerun the first exercise using the first-order administrative Global Administrative Unit Layers. Try selecting different countries to explore which ones have the highest road density. You will likely need to save the output in order to run this analysis.

Assignment 2. Compute local Geary's C for roads and transmission lines on other continents. Which continent appears to have the strongest clustering?

Assignment 3. Examine the Global Flood Database in relation to roads in an area of your choice. Try country-level analysis. Which country in your area has the highest proportion of roads potentially affected by flooding?

Conclusion

In this chapter, we have examined data from the built environment. We started by looking at characteristics of roads, then examined the interactions between roads and transmission lines, and finished by examining where built environments were most commonly developed in floodplains. Data in the built environment can be found in both vector and raster forms. We examined analysis using both forms, including rasterizing vector data. Our analysis also focused on larger-scale spatial analysis at the state, country, and continental level. Understanding how to do this type of analysis allows us to develop a better understanding of these systems as more seamless data across temporal and spatial dimensions continues to become available.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Anselin L (1995) Local indicators of spatial association—LISA. Geogr Anal 27:93–115. https://doi.org/10.1111/j.1538-4632.1995.tb00338.x

Arderne C, Zorn C, Nicolas C, Koks EE (2020) Predictive mapping of the global power system using open data. Sci Data 7:1–12. https://doi.org/10.1038/s41597-019-0347-4

Gong P, Li X, Wang J, et al (2020) Annual maps of global artificial impervious area (GAIA) between 1985 and 2018. Remote Sens Environ 236:111510. https://doi.org/10.1016/j.rse.2019.111510

Meijer JR, Huijbregts MAJ, Schotten KCGJ, Schipper AM (2018) Global patterns of current and future road infrastructure. Environ Res Lett 13:64006. https://doi.org/10.1088/1748-9326/aabd42

Tellman B, Sullivan JA, Kuhn C, et al (2021) Satellite imaging reveals increased proportion of population exposed to floods. Nature 596:80–86. https://doi.org/10.1038/s41586-021-03695-w

Chapter A1.4: Air Pollution and Population Exposure

Authors
Zander Venter and Sourangsu Chowdhury

Overview

After high blood pressure and smoking, air pollution is the third-largest risk factor for death globally (Murray et al. 2020). Air pollution can therefore be described as a global "pandemic" that should arguably be monitored and addressed with the same intensity with which the COVID-19 pandemic has been. Remote sensing and cloud computing technologies allow us to do so.

The purpose of this chapter is to explore and analyze gridded air pollution data from Sentinel-5P in the context of changes brought about by COVID-19 lockdowns. Practical components will include analyzing changes in nitrogen dioxide (NO_2) concentrations over time and quantifying population-weighted NO_2 concentrations for selected administrative units.

Learning Outcomes

- Understanding Sentinel-5P data.
- Quantifying changes in air pollutant concentrations over time.
- Generating a split-panel map to compare two time epochs.
- Calculating population-weighted air pollutant concentrations.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Create a graph using `ui.Chart` (Chap. F1.3).
- Perform basic image analysis: select bands, compute indices, create masks (Part F2).
- Use `ee.Reducer` functions to summarize pixels over an area (Chap. F3.0, Chap. F3.1).
- Write a function and `map` it over an `ImageCollection` (Chap. F4.0).
- Mask cloud, cloud shadow, snow/ice, and other undesired pixels (Chap. F4.3).

- Design user interfaces for an Earth Engine App (Chap. F6.3).

Introduction to Theory

Air pollution can be generally defined as any chemical, physical, or biological agent that alters the natural composition of the atmosphere. Pollutants that are of primary concern for public health include particulate matter with diameter less than 2.5 μm ($PM_{2.5}$), carbon monoxide (CO), ozone (O_3), NO_2, and sulfur dioxide (SO_2). Globally, chronic exposure to air pollution results in greater loss of life than HIV/AIDS, malaria, and tuberculosis combined, and more than an order of magnitude more deaths than all forms of violence (Lelieveld et al. 2020). Exposure to $PM_{2.5}$ and O_3 is estimated to result in ~4.7 million excess deaths annually across the globe (Murray et al. 2020), although these estimates range between 3 and 10 million excess deaths per year, based on the disease categories considered and the exposure-response function used (Burnett et al. 2018, Chowdhury et al. 2022). Exposure to NO_2 may result in 4 million new pediatric asthma cases annually (Achakulwisut et al. 2019).

Knowledge about the global distribution of these air pollutants and their sources has improved over the last decade, with the expansion of networks of ground-based monitors in many countries, the evolution of satellite products, and the advancement of complex atmospheric chemistry models. Studies have found that more than 70% of the global health burden from air pollution is attributable to anthropogenic emissions (Chowdhury et al. 2022, Lelieveld et al. 2019). The main anthropogenic sources of air pollution are industries, motor vehicles, power generation, agricultural activities, and household combustion, while non-anthropogenic sources include desert dust, biogenic emissions, forest fires, and even volcanoes. The reduction in transport and industrial activity during the COVID-19 lockdowns significantly reduced global air pollution levels, thereby highlighting the significance of anthropogenic emissions (Venter et al. 2020). In fact, it is estimated that the decline in air pollution during the first five months of 2020 resulted in 49,900 avoided deaths and 89,000 fewer pediatric asthma emergency room visits (Venter et al. 2021).

Despite the recent growth in monitoring networks, the air in most regions of Earth is insufficiently monitored, limiting air quality management. Given the paucity of ground-based monitoring, alternative monitoring approaches such as satellite remote sensing are gaining popularity and becoming more accurate (e.g., Griffin et al. 2019). Over the past few decades, we have had increasing access to a range of satellite sensors that monitor the contents of Earth's atmosphere. However, it is important to note that satellites measure pollutant concentrations in the troposphere and stratosphere,

which extend for many kilometers above the Earth's surface. As a result, satellite measurements are not necessarily representative of the concentrations humans are exposed to on the ground, and consequently, relying on satellite data alone for human health applications is not advised. However, more sophisticated methods combine information from satellite remote sensing data, complex atmospheric chemistry models, and ground-based monitors to provide ground-level concentrations of pollutants with high confidence (Dey et al. 2020, Donkelar et al. 2021).

Practicum

Section 1. Data Importing and Cleaning

There are a range of satellite-based datasets on air pollution to choose from in the Earth Engine Data Catalog. The main datasets relevant to air pollution include the Moderate Resolution Imaging Spectroradiometer and Advanced Very-High-Resolution Radiometer for monitoring aerosol optical depth (a proxy for $PM_{2.5}$); the Total Ozone Mapping Spectrometer Ozone Monitoring Instrument for monitoring O_3; and more recently the TROPOspheric Monitoring Instrument (TROPOMI) on board the Sentinel-5 Precursor (Sentinel-5P), which monitors a range of air pollutants. We will use Sentinel-5P in this practicum, but the methods covered here are easily transferable to the datasets mentioned above.

Now, let's load the satellite data for this practicum. If you search "tropomi" in the Earth Engine Data Catalog, you will see a range of datasets from Sentinel-5P, which can all be of value in quantifying air quality (Fig. A1.4.1).

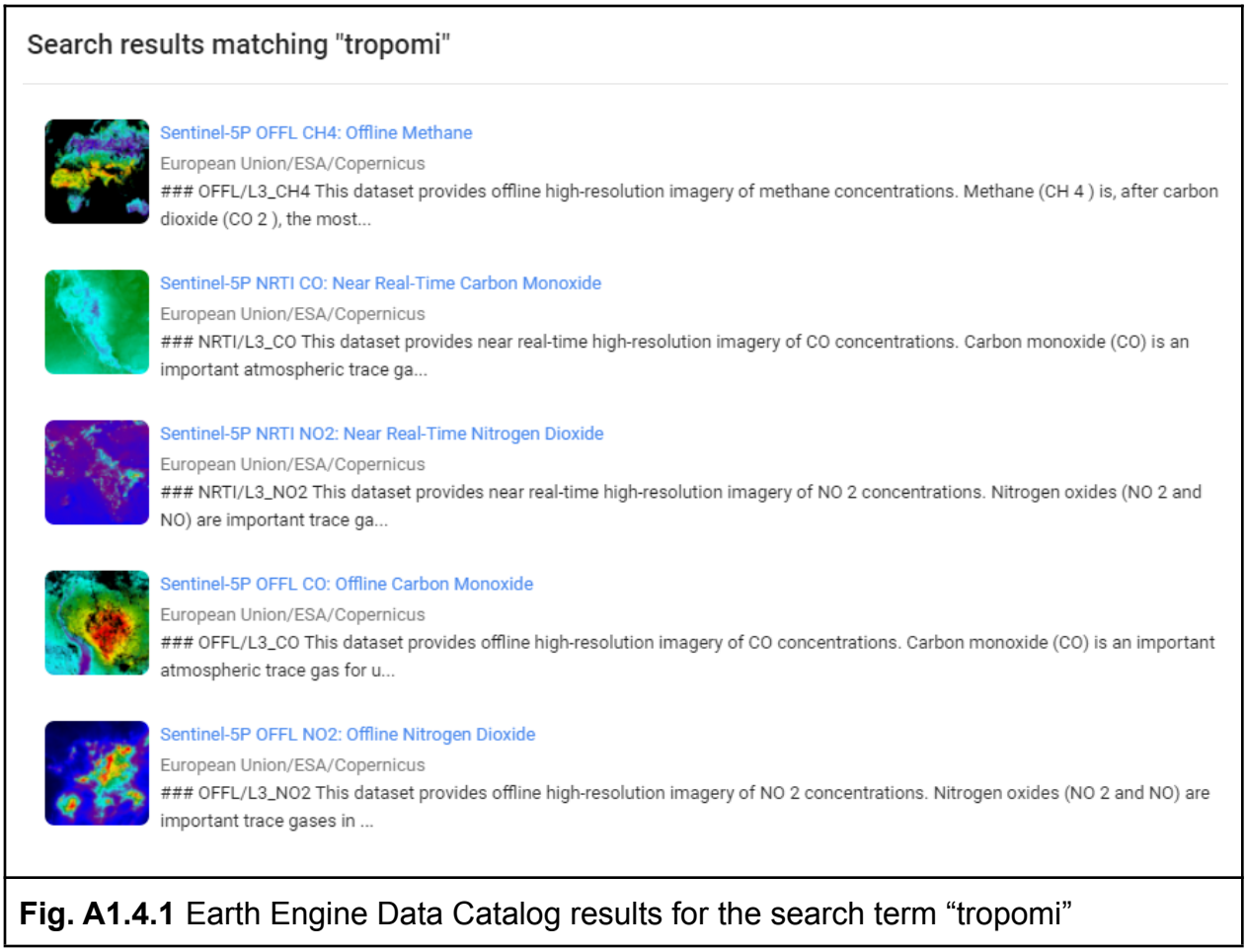


Fig. A1.4.1 Earth Engine Data Catalog results for the search term "tropomi"

Although Sentinel-5 was launched in October 2017, the data available for analysis in Earth Engine are from July 2018 onward. TROPOMI, the sensor on board the satellite, is a spectrometer sensing ultraviolet, visible, near-infrared, and shortwave infrared wavelengths to monitor NO_2, O_3, aerosol, methane (CH_4), formaldehyde, CO, and SO_2 in the atmosphere. The swath width of TROPOMI is approximately 2,600 km on the ground, resulting in a global daily coverage with a spatial resolution of 7 x 7 km. All of the Sentinel-5P datasets, except CH_4, have two versions: Near Real-Time (NRTI) and Offline (OFFL); CH_4 is available as OFFL only. The NRTI assets cover a smaller area than the OFFL assets but appear more quickly after acquisition. The OFFL assets have a delayed availability, but each asset contains data from an entire orbit and is arguably easier to work with for retrospective analyses. We will use the OFFL NO_2 product in this practicum.

First we need to define an area of interest. Wuhan is infamous for being the epicenter of the COVID-19 pandemic and witnessed severe lockdowns. In the next section of this practicum, we will test to see if we can detect a reduction in NO_2 during the early 2020 lockdowns in the surrounding province, Hubei. To start, in the code below, we import a global dataset of administrative boundaries and filter them for intersection with an `ee.Geometry.Point` object, which appears under the **Imports** section at the top of your script. This `geometry` has to be drawn with the drawing tool and can be moved to a new location to rerun the analysis for that administrative boundary.

After centering the **Map** on Hubei Province, we will import a population dataset, which is necessary for calculating population-weighted exposures in Sect. 3 of this practicum. We will use the Gridded Population of the World dataset for 2020, which includes a total population count per ~1 x 1 km grid (Fig. A1.4.2).

```
// Import a global dataset of administrative units level 1.
var adminUnits = ee.FeatureCollection(
    'FAO/GAUL_SIMPLIFIED_500m/2015/level1');

// Filter for the administrative unit that intersects
// the geometry located at the top of this script.
var adminSelect = adminUnits.filterBounds(geometry);

// Center the map on this area.
Map.centerObject(adminSelect, 8);

// Make the base map HYBRID.
Map.setOptions('HYBRID');

// Add it to the map to make sure you have what you want.
Map.addLayer(adminSelect, {}, 'selected admin unit');

// Import the population count data from Gridded Population of the
World Version 4.
var population = ee.ImageCollection(
        'CIESIN/GPWv411/GPW_Population_Count')
    // Filter for 2020 using the calendar range function.
    .filter(ee.Filter.calendarRange(2020, 2020, 'year'))
```

```
    // There should be only 1 image, but convert to an image using
.mean().
    .mean();

// Clip it to your area of interest (only necessary for visualization
purposes).
var populationClipped = population.clipToCollection(adminSelect);

// Add it to the map to see the population distribution.
var popVis = {
    min: 0,
    max: 4000,
    palette: ['black', 'yellow', 'white'],
    opacity: 0.55
};
Map.addLayer(populationClipped, popVis, 'population count');
```

Question 1. There are two other datasets of gridded population in the Earth Engine Data Catalog, namely WorldPop and Global Human Settlement Layers. Use the search bar to find them and add them to the map to compare them with the Gridded Population of the World dataset. Which one looks more realistic in your opinion, and why?

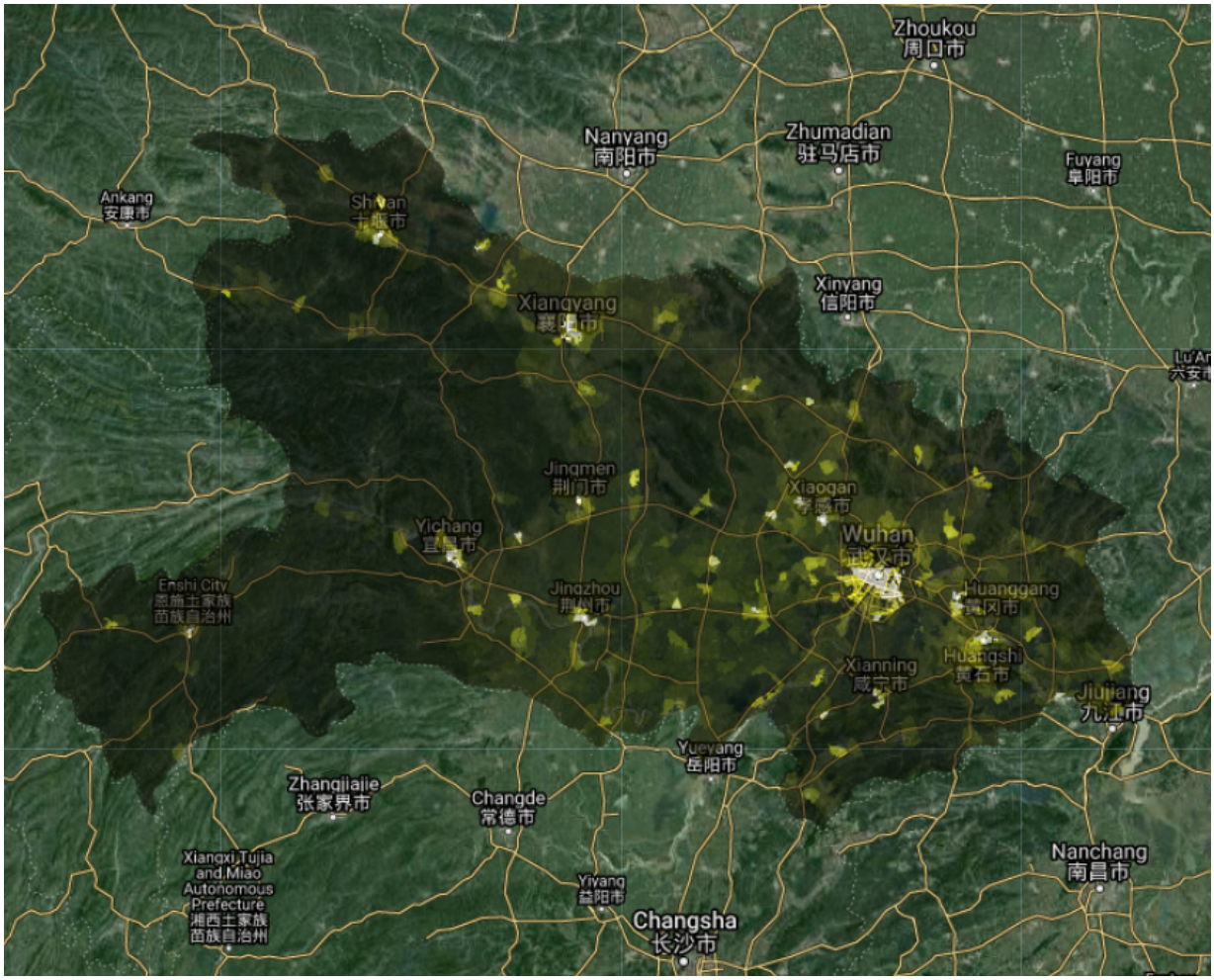


Fig. A1.4.2 Population density over Hubei Province. Brighter areas have higher population counts.

Now it is time to import the NO_2 data. As with most optical satellite data, there can be things in the atmosphere that contaminate the signal from the object or chemical you want to measure. Clouds are a common issue for land surface reflectance products (Chap. F4.3), and they are also an issue when trying to measure air pollutant concentrations. In the code below, we create a function to mask out pixels with a cloud fraction above 0.3 (i.e., 30% cloud cover). You can test different masking thresholds to see what suits your use case best. After masking out cloudy pixels, we create a median composite from images during March 2021. It is important to note that we are working with the band that gives measurements for the tropospheric vertical column of NO_2 and not the stratospheric or total vertical column. The troposphere is the closest we can get

to ground-level measurements with Sentinel-5P. The median image for March 2021 should look like the map shown in Fig. A1.4.3.

```
// Import the Sentinel-5P NO2 offline product.
var no2Raw = ee.ImageCollection('COPERNICUS/S5P/OFFL/L3_NO2');

// Define function to exclude cloudy pixels.
function maskClouds(image) {
    // Get the cloud fraction band of the image.
    var cf = image.select('cloud_fraction');
    // Create a mask using 0.3 threshold.
    var mask = cf.lte(0.3); // You can play around with this value.
    // Return a masked image.
    return image.updateMask(mask).copyProperties(image);
}

// Clean and filter the Sentinel-5P NO2 offline product.
var no2 = no2Raw
    // Filter for images intersecting our area of interest.
    .filterBounds(adminSelect)
    // Map the cloud masking function over the image collection.
    .map(maskClouds)
    // Select the tropospheric vertical column of NO2 band.
    .select('tropospheric_NO2_column_number_density');

// Create a median composite for March 2021
var no2Median = no2.filterDate('2021-03-01', '2021-04-01').median();

// Clip it to your area of interest (only necessary for visualization
purposes).
var no2MedianClipped = no2Median.clipToCollection(adminSelect);

// Visualize the median NO2.
var no2Viz = {
    min: 0,
    max: 0.00015,
```

```
    palette: ['black', 'blue', 'purple', 'cyan', 'green',
        'yellow', 'red'
    ]
};
Map.addLayer(no2MedianClipped, no2Viz, 'median no2 Mar 2021');
```

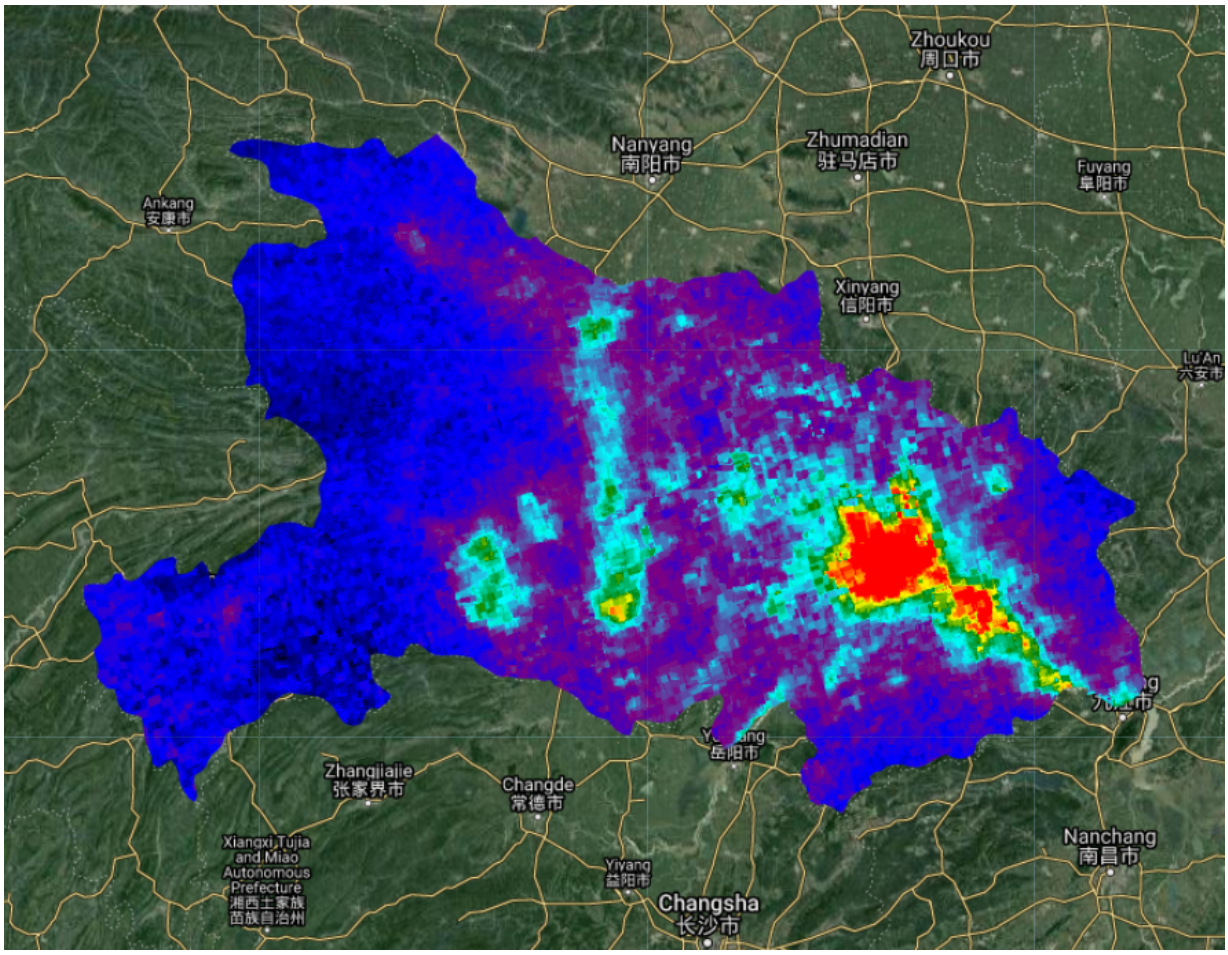


Fig. A1.4.3 Tropospheric NO_2 concentrations over Hubei Province. Hotter colors have higher concentrations, while cooler colors have lower concentrations.

Code Checkpoint A14a. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Quantifying and Visualizing Changes

Next we will test to see if we can visualize a change in NO_2 concentrations during the 2020 COVID-19 lockdowns. We will compare the median NO_2 concentration during March 2020 (during which Hubei Province was in lockdown) with the median value during March 2019.

Weather can significantly affect air pollutant concentrations (e.g., wind causing long-range transport of smoke), and therefore differences between 2020 and 2019 could be an artifact of differences in weather. By comparing the same month in different years, we partly control for the effects of seasonal weather patterns, but not completely. If you would like to control for weather effects more thoroughly, see Venter et al. (2020) for details. In the code below, we calculate and visualize median composite images for March 2019 and March 2020. The visualization makes use of Earth Engine's comprehensive library of user-interface widgets (see Chap. F6.3 for more details). Specifically, we use the `ui.SplitPanel` widget to compare the two median composites side by side (Fig. A1.4.4). This widget can be set to have a wiping effect where maps are overlaid on top of one another, or a side-by-side comparison.

```
// Define a lockdown NO2 median composite.
var no2Lockdown = no2.filterDate('2020-03-01', '2020-04-01')
    .median().clipToCollection(adminSelect);

// Define a baseline NO2 median using the same month in the previous
year.
var no2Baseline = no2.filterDate('2019-03-01', '2019-04-01')
    .median().clipToCollection(adminSelect);

// Create a ui map widget to hold the baseline NO2 image.
var leftMap = ui.Map().centerObject(adminSelect, 8).setOptions(
    'HYBRID');

// Create ta ui map widget to hold the lockdown NO2 image.
var rightMap = ui.Map().setOptions('HYBRID');

// Create a split panel widget to hold the two maps.
```

```
var sliderPanel = ui.SplitPanel({
    firstPanel: leftMap,
    secondPanel: rightMap,
    orientation: 'horizontal',
    wipe: true,
    style: {
        stretch: 'both'
    }
});
var linker = ui.Map.Linker([leftMap, rightMap]);

// Make a function to add a label with fancy styling.
function makeMapLab(lab, position) {
    var label = ui.Label({
        value: lab,
        style: {
            fontSize: '16px',
            color: '#ffffff',
            fontWeight: 'bold',
            backgroundColor: '#ffffff00',
            padding: '0px'
        }
    });
    var panel = ui.Panel({
        widgets: [label],
        layout: ui.Panel.Layout.flow('horizontal'),
        style: {
            position: position,
            backgroundColor: '#00000057',
            padding: '0px'
        }
    });
    return panel;
}

// Create baseline map layer, add it to the left map, and add the
label.
```

```
var no2BaselineLayer = ui.Map.Layer(no2Baseline, no2Viz);
leftMap.layers().reset([no2BaselineLayer]);
leftMap.add(makeMapLab('Baseline 2019', 'top-left'));

// Create lockdown map layer, add it to the right map, and add the
label.
var no2LockdownLayer = ui.Map.Layer(no2Lockdown, no2Viz);
rightMap.layers().reset([no2LockdownLayer]);
rightMap.add(makeMapLab('Lockdown 2020', 'top-right'));

// Reset the map interface (ui.root) with the split panel widget.
// Note that the Map.addLayer() calls earlier on in Section 1
// will no longer be shown because we have replaced the Map widget
// with the sliderPanel widget.
ui.root.widgets().reset([sliderPanel]);
```

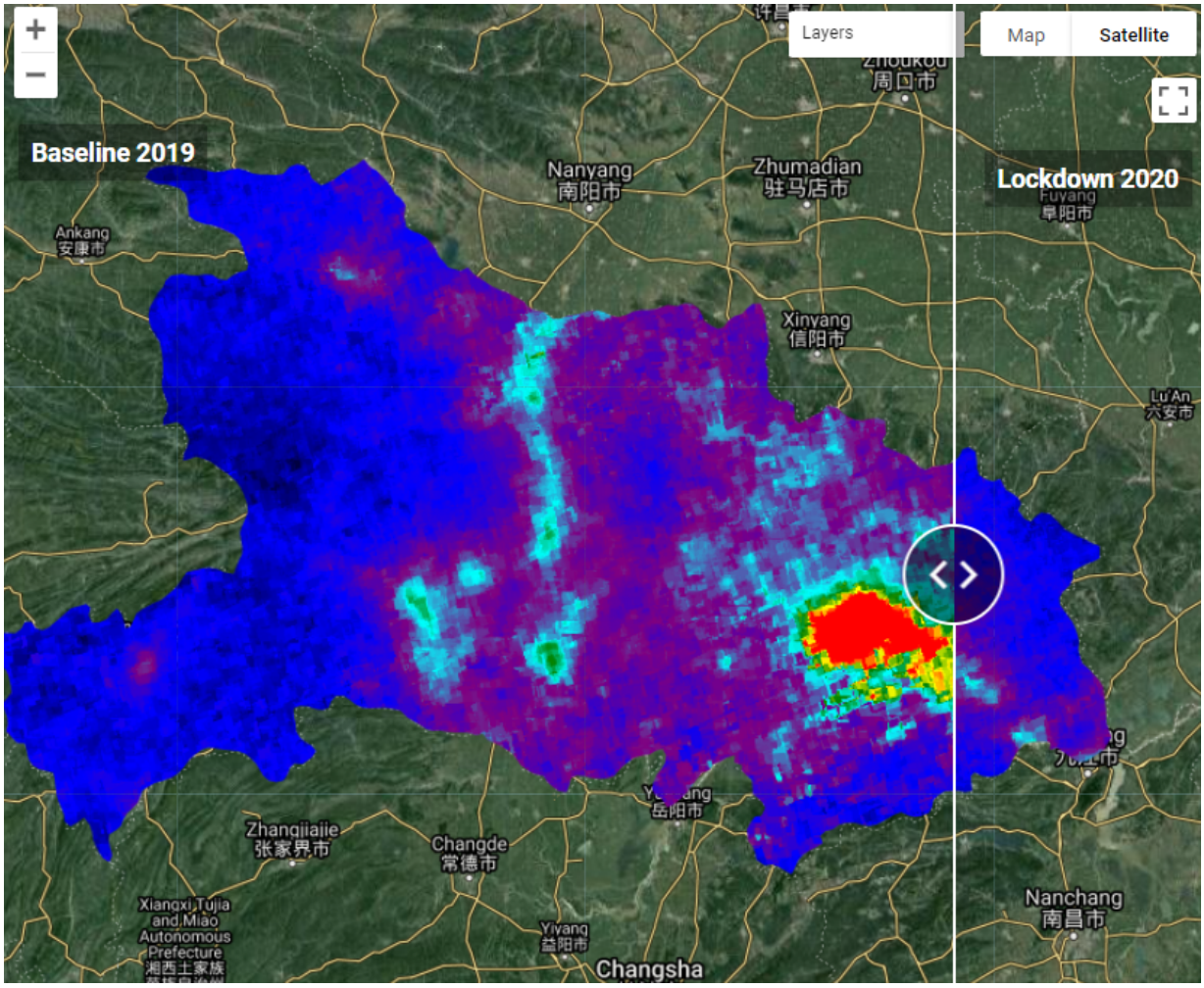


Fig. A1.4.4 Split-panel map showing tropospheric NO_2 concentrations over Hubei Province for March 2019 (left) and March 2020 (right). Hotter colors have higher concentrations, while cooler colors have lower concentrations.

Question 2. Comparing the two maps in the split-panel map, do you find a reduction in NO_2 concentrations during the lockdown? Where is the change in NO_2 concentrations most significant?

Question 3. How are changes in NO_2 concentrations related to population density? To help answer this question, you can (1) create a difference image by subtracting the `no2Lockdown` image from the `no2Baseline` image, (2) create a new `ui.Map.Layer` for the difference image and the population image created in Sect. 1, and (3) add these to

the left or right map. Hint: You can change the opacity of the NO_2 layers to aid interpretability.

Exploring the differences in NO_2 concentrations as `ee.Image` objects can be visually informative, but quantifying the changes for specific regions requires further work. In the code below, we calculate the mean NO_2 concentrations for Hubei Province by applying a `reduceRegion` function to each image in the March 2019 and March 2020 collections. The resulting time series are visualized in the chart shown in Fig. A1.4.5.

```
// Create a function to get the mean NO2 for the study region
// per image in the NO2 collection.
function getConc(collectionLabel, img) {
    return function(img) {
        // Calculate the mean NO2.
        var no2Mean = img.reduceRegion({
            reducer: ee.Reducer.mean(),
            geometry: adminSelect.geometry(),
            scale: 7000
        }).get('tropospheric_NO2_column_number_density');

        // Get the day-of-year of the image.
        var doy = img.date().getRelative('day', 'year');

        // Return a feature with NO2 concentration and day-of-year
properties.
        return ee.Feature(null, {
            'conc': no2Mean,
            'DOY': doy,
            'type': collectionLabel
        });
    };
}

// Get the concentrations for a baseline and lockdown collection
// and merge for plotting.
var no2AggChange_forPlotting = no2
    .filterDate('2020-03-01', '2020-04-01')
```

```
    .map(getConc('lockdown'))
    .merge(no2.filterDate('2019-03-01', '2019-04-01')
        .map(getConc('baseline')));
no2AggChange_forPlotting = no2AggChange_forPlotting
    .filter(ee.Filter.notNull(['conc']));

// Make a chart.
var chart1 = ui.Chart.feature.groups(
        no2AggChange_forPlotting, 'DOY', 'conc', 'type')
    .setChartType('LineChart')
    .setOptions({
        title: 'DOY time series for mean [NO2] during ' +
            'March 2019 (baseline) and 2020 (lockdown)'
    });

// Print it to the console.
print('Baseline vs lockdown NO2 for the study region by DOY', chart1);
```

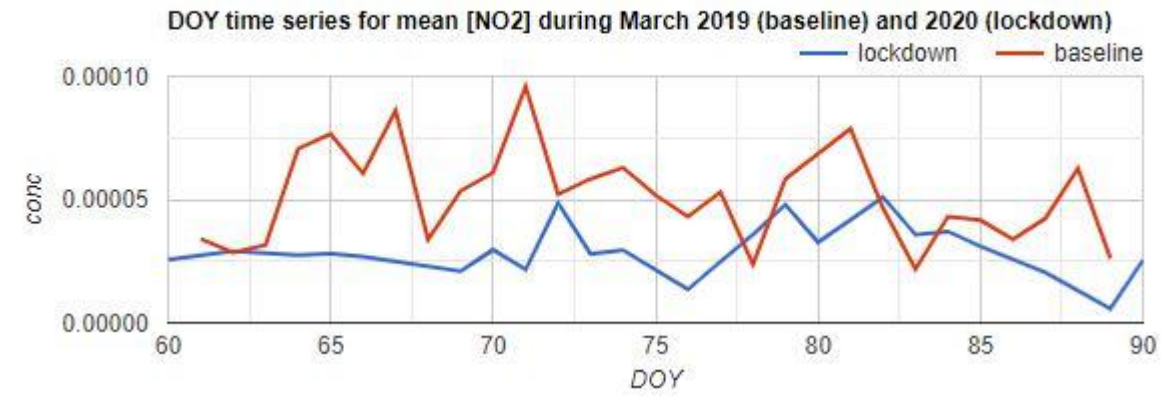


Fig. A1.4.5 Time-series graph showing average NO_2 concentrations for Hubei Province during March 2019 and March 2020

Code Checkpoint A14b. The book's repository contains a script that shows what your code should look like at this point.

Section 3. Calculating Population-Weighted Concentrations

In Sect. 2, we used the `ee.Reducer.mean` reducer in the `reduceRegion` function to get the average NO_2 concentration over Hubei Province. However, when aggregating pollutant concentrations to define population exposure, we need a different approach. Imagine there was a large concentration of NO_2 in a rural area in the east of Hubei Province where very few people live. If we simply calculated the average of all pixels, this rural NO_2 anomaly would skew our representation of population exposure. Using the population number dataset imported in Sect. 1, we can calculate the population-weighted exposure (Exp) aggregated across n pixels in the area of interest (in this case, Hubei Province) using Eq. A1.4.1 below, where C_i is the NO_2 concentration and P_i is the subpopulation in pixel i.

$$Exp = \sum_{i}^{n} \frac{P_i}{\sum_{i}^{n}(P)} \cdot C_i \qquad \text{(Eq. A1.4.1)}$$

In the code below, we map a function to calculate population-weighted exposure over all the images in the NO_2 `ImageCollection`. Remember that in Sect. 1 we masked out pixels from images that had a cloud cover value greater than 30%. Therefore, an important step in this function is to calculate the percentage of available Sentinel-5P pixels within Hubei Province per image. We need to decide what percentage pixel coverage is enough to calculate a representative average for the province. Here we choose 25% for illustrative purposes, but depending on your research question, you may want to calculate averages only when you have 100% coverage by/from Sentinel-5P that is free of clouds. The contrast between the simple average and population-weighted average is shown in Fig. A1.4.6. The difference may appear small in this case, but when aggregating over larger areas with greater variation in population density, population-weighted averages can be very different from simple averages.

```
// Define the spatial resolution of the population data.
var scalePop = 927.67; // See details in GEE Catalogue.

// Now we define a function that will map over the NO2 collection
// and calculate population-weighted concentrations.
// We will use the formula Exp = SUM {(Pi/P)*Ci}.
// We can calculate P outside of the function
// so that it is not computed multiple times for each NO2 image.
var P = population.reduceRegion({
```

```
    reducer: ee.Reducer.sum(),
    geometry: adminSelect.geometry(),
    scale: scalePop
}).get('population_count');

// And here is the function.
function getPopWeightedConc(P, region, regionName, img) {
    return function(img) {
        var Ci = img;
        var Pi = population;
        // Calculate the percentage of valid pixels in the region.
        // (masked pixels will not be counted).
        var pixelCoverPerc = Ci.gte(0).unmask(0).multiply(100)
            .reduceRegion({
                reducer: ee.Reducer.mean(),
                geometry: region.geometry(),
                scale: scalePop // Add in the scale of the population
raster.
            }).get('tropospheric_NO2_column_number_density');

        // Calculate the per-pixel EXP (see formula above).
        var exp = Pi.divide(ee.Image(ee.Number(P))).multiply(Ci);

        // Sum the exp over the region.
        var expSum = exp.reduceRegion({
            reducer: ee.Reducer.sum(),
            geometry: region.geometry(),
            scale: scalePop
        }).get('population_count');

        // Calculate the mean NO2 - the approach that would usually
        // be taken without population weighting.
        var no2Mean = Ci.reduceRegion({
            reducer: ee.Reducer.mean(),
            geometry: region.geometry(),
            scale: scalePop
        }).get('tropospheric_NO2_column_number_density');
```

```
        // Return a feature with properties
        var featOut = ee.Feature(null, {
            'system:time_start': img.get(
                'system:time_start'),
            'dateString': img.date().format('YYYY-MM-DD'),
            'regionName': regionName,
            'no2ConcPopWeighted': expSum,
            'no2ConcRaw': no2Mean,
            'pixelCoverPerc': pixelCoverPerc
        });

        return featOut;
    };

}

// Filter the NO2 collection for March 2020 and map the function over
it.
var no2Agg_popWeighted = no2.filterDate('2020-03-01', '2020-04-01')
    .map(getPopWeightedConc(P, adminSelect, 'Wuhan'));
no2Agg_popWeighted = ee.FeatureCollection(no2Agg_popWeighted);

// Define the percentage of valid pixels you want in your region per
time point.
// Here we choose 25; i.e. only images with at least 25% valid NO2
pixels.
var validPixelPerc = 25; // you can play around with this value

// Filter the feature collection based on your pixel criteria.
no2Agg_popWeighted = no2Agg_popWeighted
    .filter(ee.Filter.greaterThanOrEquals('pixelCoverPerc',
        validPixelPerc));
print('Population weighted no2 feature collection:',
    no2Agg_popWeighted);

// Create a feature collection for plotting the mean [NO2]
```

```
// and the mean pop-weighted [NO2] on the same graph.
var no2Agg_forPlotting = no2Agg_popWeighted.map(function(ft) {
    return ft.set('conc', ft.get('no2ConcPopWeighted'),
        'type', 'no2ConcPopWeighted');
}).merge(no2Agg_popWeighted.map(function(ft) {
    return ft.set('conc', ft.get('no2ConcRaw'), 'type',
        'no2ConcRaw');
}));

// Make a chart
var chart2 = ui.Chart.feature.groups(
        no2Agg_forPlotting, 'system:time_start', 'conc', 'type')
    .setChartType('LineChart')
    .setOptions({
        title: 'Time series for mean [NO2] and the pop-weighted [NO2]'
    });

// Print it to the console
print('Raw vs population-weighted NO2 for the study region', chart2);
```

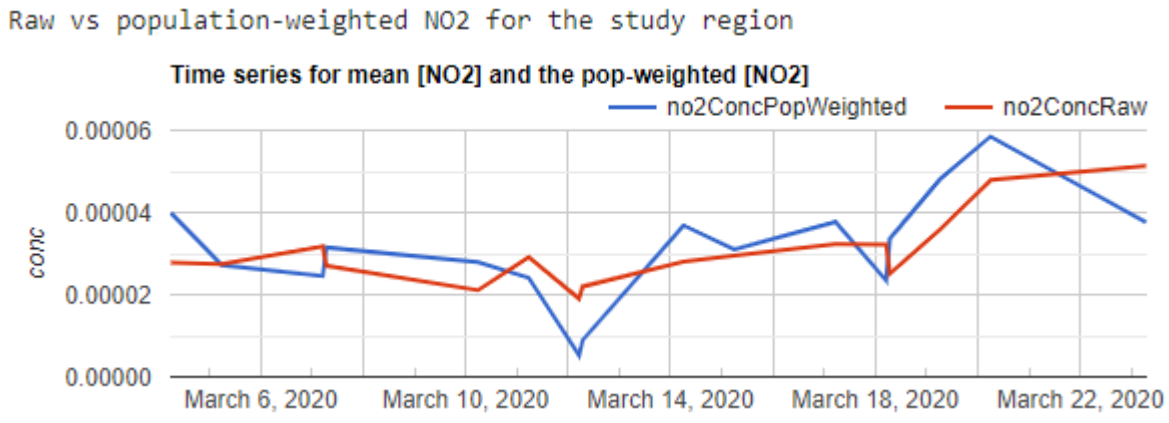


Fig. A1.4.6 Time-series graph showing average (no2ConcRaw) and population-weighted average (no2ConcPopWeighted) NO_2 concentrations for Hubei Province in March 2020

Finally, although we can plot this data in Earth Engine, it is often easier to process with other statistical software, such as R or Python. So, to conclude, let us code for exporting time series of population-weighted averages for more than one area of interest (in this case, administrative units). In the code below, we map the function over two regions and then export the resulting table as a CSV file to Google Drive.

```
// Export population-weighted data for multiple regions.
// First select the regions. This can also be done with
// .filterBounds() as in Line 9 above.
var regions = adminUnits
    .filter(ee.Filter.inList('ADM1_NAME', ['Chongqing Shi',
        'Hubei Sheng'
    ]));
// Map a function over the regions that calculates population-weighted
[NO2].
var No2AggMulti_popWeighted = regions.map(function(region) {
    var P = population.reduceRegion({
        reducer: ee.Reducer.sum(),
        geometry: region.geometry(),
        scale: scalePop
    }).get('population_count');
    var innerTable = no2.filterDate('2020-03-01',
            '2020-04-01')
        .map(getPopWeightedConc(P, region, region.get(
            'ADM1_NAME')));
    return innerTable;
}).flatten();
// Remember to filter out readings that have pixel percentage cover
// below your threshold
No2AggMulti_popWeighted = No2AggMulti_popWeighted
    .filter(ee.Filter.greaterThanOrEquals('pixelCoverPerc',
        validPixelPerc));

// Run the export under the 'Tasks' tab on the right
// and find your CSV file in Google Drive later on.
Export.table.toDrive({
    collection: No2AggMulti_popWeighted,
```

```
    description: 'no2_popWeighted',
    fileFormat: 'CSV'
});
```

Code Checkpoint A14c. The book's repository contains a script that shows what your code should look like at this point.

Synthesis

In this practicum, we focused on a particular pollutant (NO_2), region (Hubei), and time period (March 2019 and March 2020). To reinforce your comprehension and understanding, consider the following assignments.

Assignment 1. How would you run this analysis for a different pollutant? Try substituting the NO_2 collection with the Sentinel-5P NRTI SO_2 collection. Hint: The main emission source for SO_2 is electricity generation, for which coal is the most significant fuel. Use this information to inform your selection of a location and time period so that you can detect interesting changes.

Assignment 2. How would you run this analysis for a different geographic area? Try deleting the `ee.Geometry.Point` at the top of your script and using the **Geometry Tools** to digitize your own point on which to focus the analysis. If you are running the latter part of the script, you can also change the list of named administrative units. Hint: Add the `adminUnits` object from Sect. 1 of the code to the map. You can use the **Inspector** tab to click on polygons and get the name of the administrative unit under the 'ADM1_NAME' property.

Assignment 3. Finally, try changing the dates in the script so that you are comparing two different time periods. Remember that the Sentinel-5P data are available from July 2018 onward; defining dates before this will cause the script to throw an error.

Conclusion

In this chapter, we covered the basics of importing Sentinel-5P air pollution data, comparing changes over time, and calculating population-weighted averages for spatial units. Satellite detection of air pollutants is an important tool for monitoring air quality from local to global scales, but ground-station measurements and atmospheric modeling are often necessary to draw conclusions about human health risk. The fusion of ground-level and satellite data with advanced machine learning models to map and

forecast air pollution is a growing research field with important societal applications (e.g., https://www.iqair.com/). Earth Engine is a well-suited and currently underutilized resource to advance this field.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Achakulwisut P, Brauer M, Hystad P, Anenberg SC (2019) Global, national, and urban burdens of paediatric asthma incidence attributable to ambient NO2 pollution: Estimates from global datasets. Lancet Planet Heal 3:e166–e178. https://doi.org/10.1016/S2542-5196(19)30046-4

Benedetti A, Morcrette J-J, Boucher O, et al (2009) Aerosol analysis and forecast in the European centre for medium-range weather forecasts integrated forecast system: 2. Data assimilation. J Geophys Res Atmos 114. https://doi.org/10.1029/2008JD011235.

Burnett R, Chen H, Szyszkowicz M, et al (2018) Global estimates of mortality associated with long-term exposure to outdoor fine particulate matter. Proc Natl Acad Sci USA 115:9592–9597. https://doi.org/10.1073/pnas.1803222115

Chowdhury S, Pozzer A, Haines A, et al (2022) Global health burden of ambient PM2.5 and the contribution of anthropogenic black carbon and organic aerosols. Environ Int 159:107020. https://doi.org/10.1016/j.envint.2021.107020

Dey S, Purohit B, Balyan P, et al (2020) A satellite-based high-resolution (1-km) ambient PM2.5 database for India over two decades (2000–2019): Applications for air quality management. Remote Sens 12:1–22. https://doi.org/10.3390/rs12233872

Griffin D, Zhao X, McLinden CA, et al (2019) High-resolution mapping of nitrogen dioxide with TROPOMI: First results and validation over the Canadian oil sands. Geophys Res Lett 46:1049–1060. https://doi.org/10.1029/2018GL081095

Lelieveld J, Klingmüller K, Pozzer A, et al (2019) Effects of fossil fuel and total anthropogenic emission removal on public health and climate. Proc Natl Acad Sci USA 116:7192–7197. https://doi.org/10.1073/pnas.1819989116

Lelieveld J, Pozzer A, Pöschl U, et al (2020) Loss of life expectancy from air pollution compared to other risk factors: A worldwide perspective. Cardiovasc Res 116:1910–1917. https://doi.org/10.1093/cvr/cvaa025

Murray CJL, Aravkin AY, Zheng P, et al (2020) Global burden of 87 risk factors in 204 countries and territories, 1990–2019: A systematic analysis for the Global Burden of Disease Study 2019. Lancet 396:1223–1249. https://doi.org/10.1016/S0140-6736(20)30752-2

Van Donkelaar A, Hammer MS, Bindle L, et al (2021) Monthly global estimates of fine particulate matter and their uncertainty. Environ Sci Technol 55:15287–15300. https://doi.org/10.1021/acs.est.1c05309

Venter ZS, Aunan K, Chowdhury S, Lelieveld J (2020) COVID-19 lockdowns cause global air pollution declines. Proc Natl Acad Sci USA 117:18984–18990. https://doi.org/10.1073/pnas.2006853117

Venter ZS, Aunan K, Chowdhury S, Lelieveld J (2021) Air pollution declines during COVID-19 lockdowns mitigate the global health burden. Environ Res 192:110403. https://doi.org/10.1016/j.envres.2020.110403

Chapter A1.5: Heat Islands

Author
TC Chakraborty

Overview

In this chapter, you will learn about urban heat islands and how they can be calculated from satellite measurements of thermal radiation from the Earth's surface.

Learning Outcomes

- Understanding how to derive land surface temperature.
- Understanding how to generate urban and rural references.
- Knowing how to calculate the surface urban heat island intensity.

Helps if you know how to:

- Import, filter, and visualize images (Part F1).
- Perform basic image analysis: select bands, compute indices, create masks (Part F2).
- Use expressions to perform calculations on image bands (Chap. F3.1).
- Write a function and map it over an `ImageCollection` (Chap. F4.0).
- Mask cloud, cloud shadow, snow/ice, and other undesired pixels (Chap. F4.3).
- Conduct basic vector analyses: vectorizing and buffering (Part F5).
- Write a function and map it over a `FeatureCollection` (Chap. F5.1, Chap. F5.2).

Introduction to Theory

Urbanization involves replacement of natural landscapes with built-up structures such as buildings, roads, and parking lots. This land cover modification also changes the properties of the land surface. These changes can range from how much radiation is reflected and absorbed by the surface, to how the heat is dissipated from the surface (for example, removal of vegetation for urban development reduces evaporative cooling). The changes in surface properties can modify local weather and climate (Kalnay and Cai 2003). The most-studied local climate modification due to urbanization is the urban heat island (UHI) effect (Arnfield 2003; Qian et al. 2022). The UHI is the

phenomenon in which a city is warmer than either its surroundings or an equivalent surface that is not urbanized. We have known about the UHI effect for almost 200 years (Howard 1833).

Traditionally, the UHI was defined as the difference in air temperature, measured by weather stations, between a city and some rural reference outside the city (Oke 1982). One issue with this method is that different parts of the city can have different air temperatures, making it difficult to capture the UHI for the entire city. Using satellite observations in the thermal bands allows us to get another measure of temperature: the radiometric skin temperature, often known as the land surface temperature (LST). We can use LST to calculate a surface UHI (SUHI) intensity, including how it varies within cities at the pixel scale (Ngie et al. 2014). It is important to stress here that the UHI values observed by satellites and those calculated using air temperature measurements can be very different (Chakraborty et al. 2017, Hu et al. 2019, Venter et al. 2021).

Practicum

Section 1. Deriving Land Surface Temperature

Section 1.1. Deriving Land Surface Temperature from MODIS

Land surface temperature can either be extracted from derived products, such as the MODIS Terra and Aqua satellite products (Wan 2006), or estimated directly from measurements in the thermal band. We will explore both options using the city of New Haven, Connecticut, USA, as the region of interest (Fig. A1.5.1). We will start with the MODIS LST.

We start by loading the feature collection, which, being a census tract-level aggregation, we dissolve to get the overall boundary using the `union` operation. The `FeatureCollection` is added to the map for demonstration:

```
// Load feature collection of New Haven's census tracts from user
assets.
var regionInt = ee.FeatureCollection(
    'projects/gee-book/assets/A1-5/TC_NewHaven');

// Get dissolved feature collection using an error margin of 50
meters.
```

```
var regionInt = regionInt.union(50);

// Set map center and zoom level (Zoom level varies from 1 to 20).
Map.setCenter(-72.9, 41.3, 12);

// Add layer to map.
Map.addLayer(regionInt, {}, 'New Haven boundary');
```

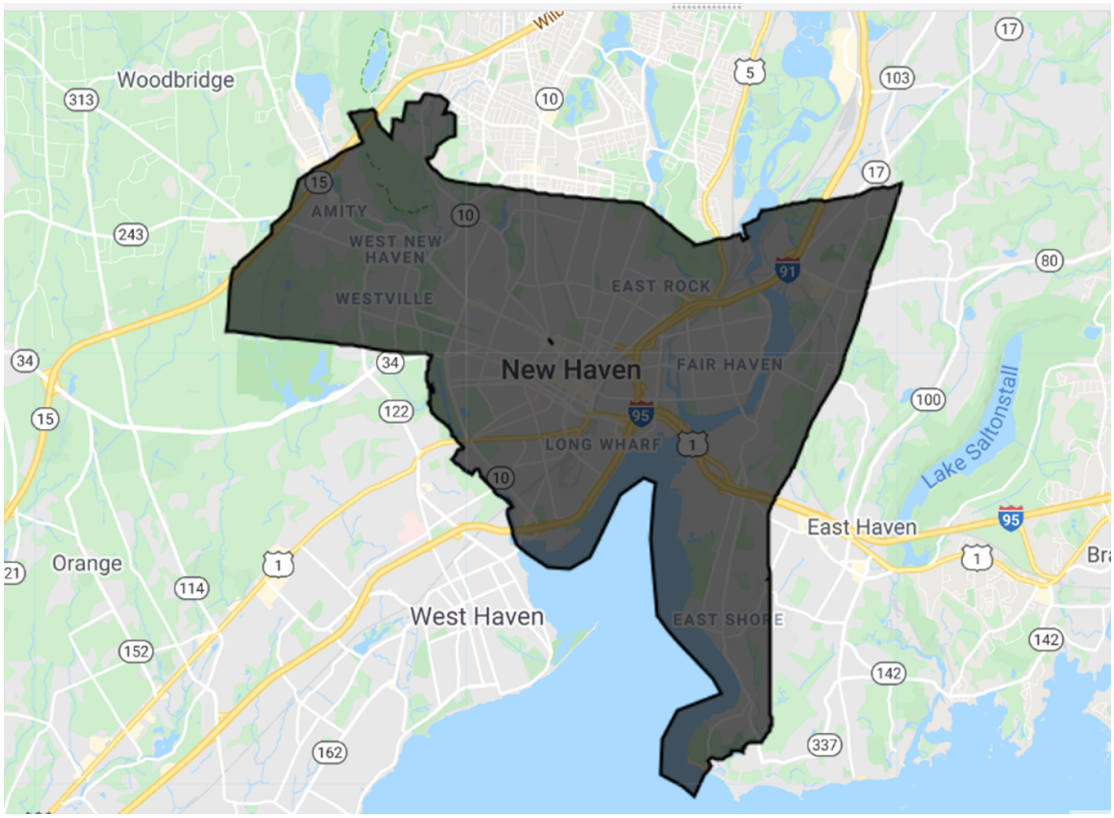


Fig. A1.5.1 Boundary of New Haven, Connecticut

Next we load in the MODIS MYD11A2 version 6 product, which provides eight-day composites of LST from the Aqua satellite. This corresponds to an equatorial crossing time of roughly 1:30 p.m. during daytime and 1:30 a.m. at night. In contrast, the MODIS sensor onboard the Terra platform (MOD11A2 version 6) has an overpass of roughly 10:30 a.m. and 10:30 p.m.

```
// Load MODIS image collection from the Earth Engine data catalog.
var modisLst = ee.ImageCollection('MODIS/006/MYD11A2');
```

```
// Select the band of interest (in this case: Daytime LST).
var landSurfTemperature = modisLst.select('LST_Day_1km');
```

We want to focus on only summertime SUHI, so we will create a five-year summer composite of LST using a day-of-year filter assembling images from June 1 (day 152) to August 31 (day 243) in each year:

```
// Create a summer filter.
var sumFilter = ee.Filter.dayOfYear(152, 243);

// Filter the date range of interest using a date filter.
var lstDateInt = landSurfTemperature
    .filterDate('2014-01-01', '2019-01-01').filter(sumFilter);

// Take pixel-wise mean of all the images in the collection.
var lstMean = lstDateInt.mean();
```

We now convert this image into LST in degrees Celsius and mask out all the water pixels (the high specific heat capacity of water would affect LST, and we are focused on land pixels). For the water mask, we use the Global Surface Water dataset (Pekel et al. 2016); to convert the pixel values, we use the scaling factor for the band from the data provider and then subtract by 273.15 to convert from Kelvin to degrees Celsius. The scaling factor can be found in the Earth Engine data summary page (Fig. A1.5.2).

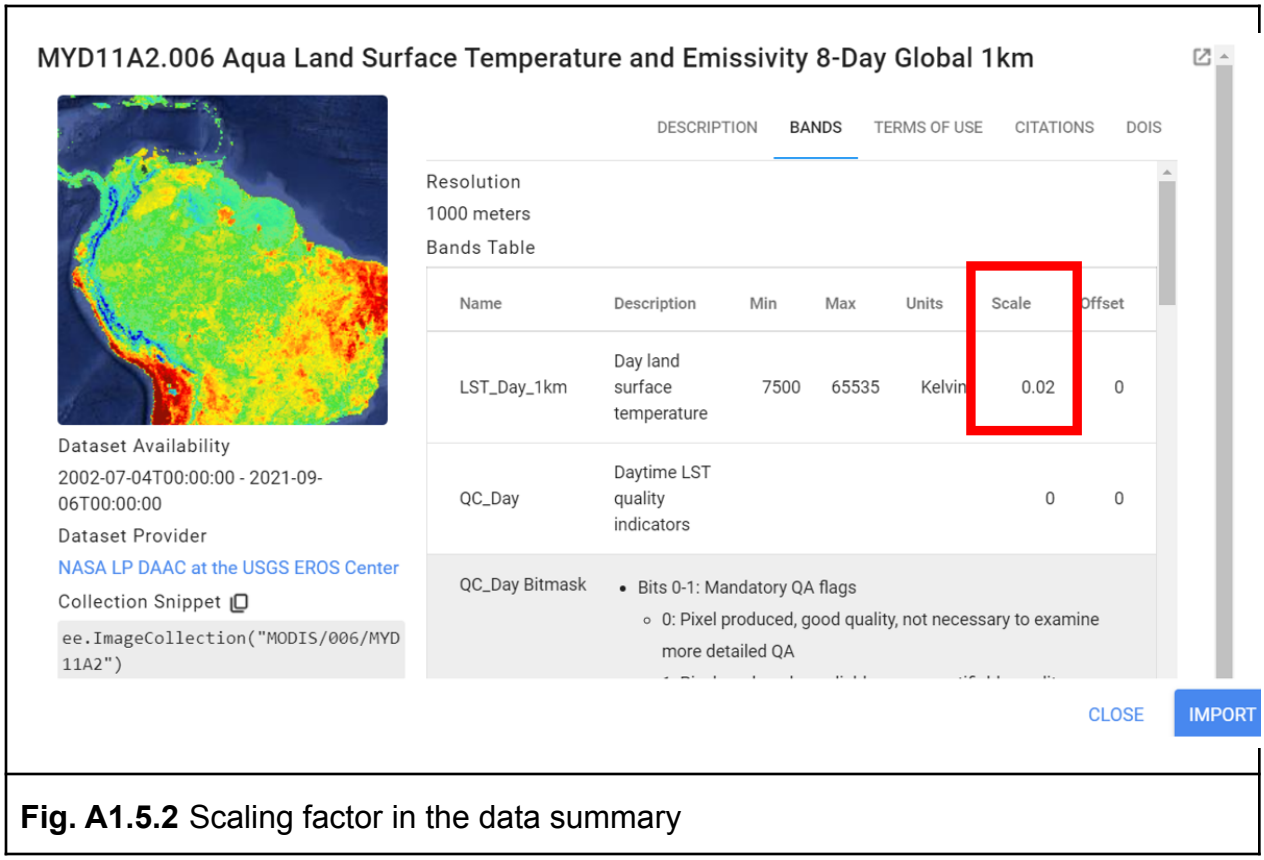


Fig. A1.5.2 Scaling factor in the data summary

Finally, we clip the image using the city boundary and add the layer to the map.

```
// Multiply each pixel by scaling factor to get the LST values.
var lstFinal = lstMean.multiply(0.02);

// Generate a water mask.
var water = ee.Image('JRC/GSW1_0/GlobalSurfaceWater').select(
    'occurrence');
var notWater = water.mask().not();

// Clip data to region of interest, convert to degree Celsius, and
mask water pixels.
var lstNewHaven = lstFinal.clip(regionInt).subtract(273.15)
    .updateMask(notWater);

// Add layer to map.
Map.addLayer(lstNewHaven, {
```

```
        palette: ['blue', 'white', 'red'],
        min: 25,
        max: 38
    },
    'LST_MODIS');
```

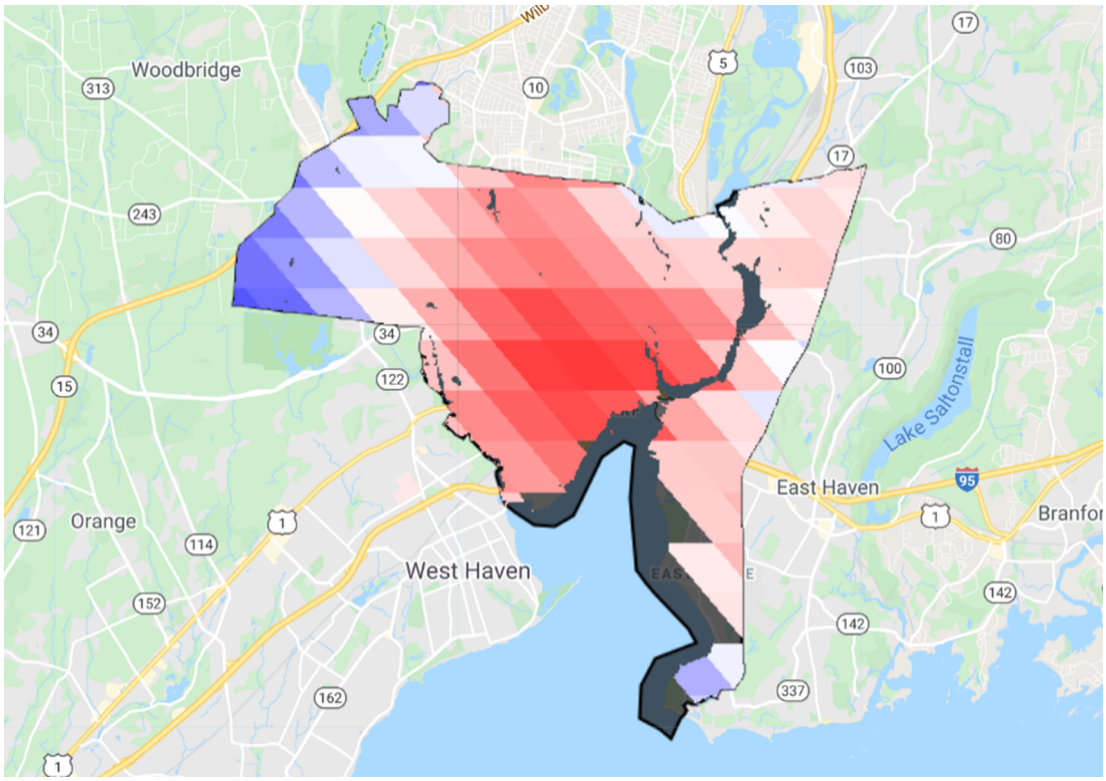


Fig. A1.5.3 Five-year summer composite of daytime MODIS Aqua LST over New Haven, Connecticut. Red pixels show higher LST values and blue pixels have lower values.

Code Checkpoint A15a. The book's repository contains a script that shows what your code should look like at this point.

Section 1.2. Deriving Land Surface Temperature from Landsat

Working with MODIS LST is relatively simple because the data are already processed by the NASA team. You can also derive LST from Landsat, which has a much finer native

resolution (~30 m) than the ~1 km MODIS pixels. However, you need to derive LST yourself from the measurements in the thermal bands, which also usually involves some estimate of surface emissivity (Li et al. 2013). The surface emissivity (ε) of a material is the effectiveness with which it can emit thermal radiation compared to a black body at the same temperature and can range from 0 (for a perfect reflector) to 1 (for a perfect absorber and emitter). Since the thermal radiation captured by satellites is a function of both LST and ε, you need to accurately prescribe or estimate ε to get to the correct LST. Let's consider one such simple method using Landsat 8 data.

We will start by loading in the Landsat data, cloud screening and then filtering to a time and region of interest. Continuing in the same script, add the following code:

```
// Function to filter out cloudy pixels.
function cloudMask(cloudyScene) {
    // Add a cloud score band to the image.
    var scored = ee.Algorithms.Landsat.simpleCloudScore(cloudyScene);

    // Create an image mask from the cloud score band and specify
threshold.
    var mask = scored.select(['cloud']).lte(10);

    // Apply the mask to the original image and return the masked
image.
    return cloudyScene.updateMask(mask);
}

// Load the collection, apply coud mask, and filter to date and region
of interest.
var col = ee.ImageCollection('LANDSAT/LC08/C02/T1_TOA')
    .filterBounds(regionInt)
    .filterDate('2014-01-01', '2019-01-01')
    .filter(sumFilter)
    .map(cloudMask);

print('Landsat collection', col);
```

After creating a median composite as a simple way to further reduce the influence of clouds, we mask out the water pixels, and select the brightness temperature band.

```
// Generate median composite.
var image = col.median();

// Select thermal band 10 (with brightness temperature).
var thermal = image.select('B10')
    .clip(regionInt)
    .updateMask(notWater);

Map.addLayer(thermal, {
        min: 295,
        max: 310,
        palette: ['blue', 'white', 'red']
    },
    'Landsat_BT');
```

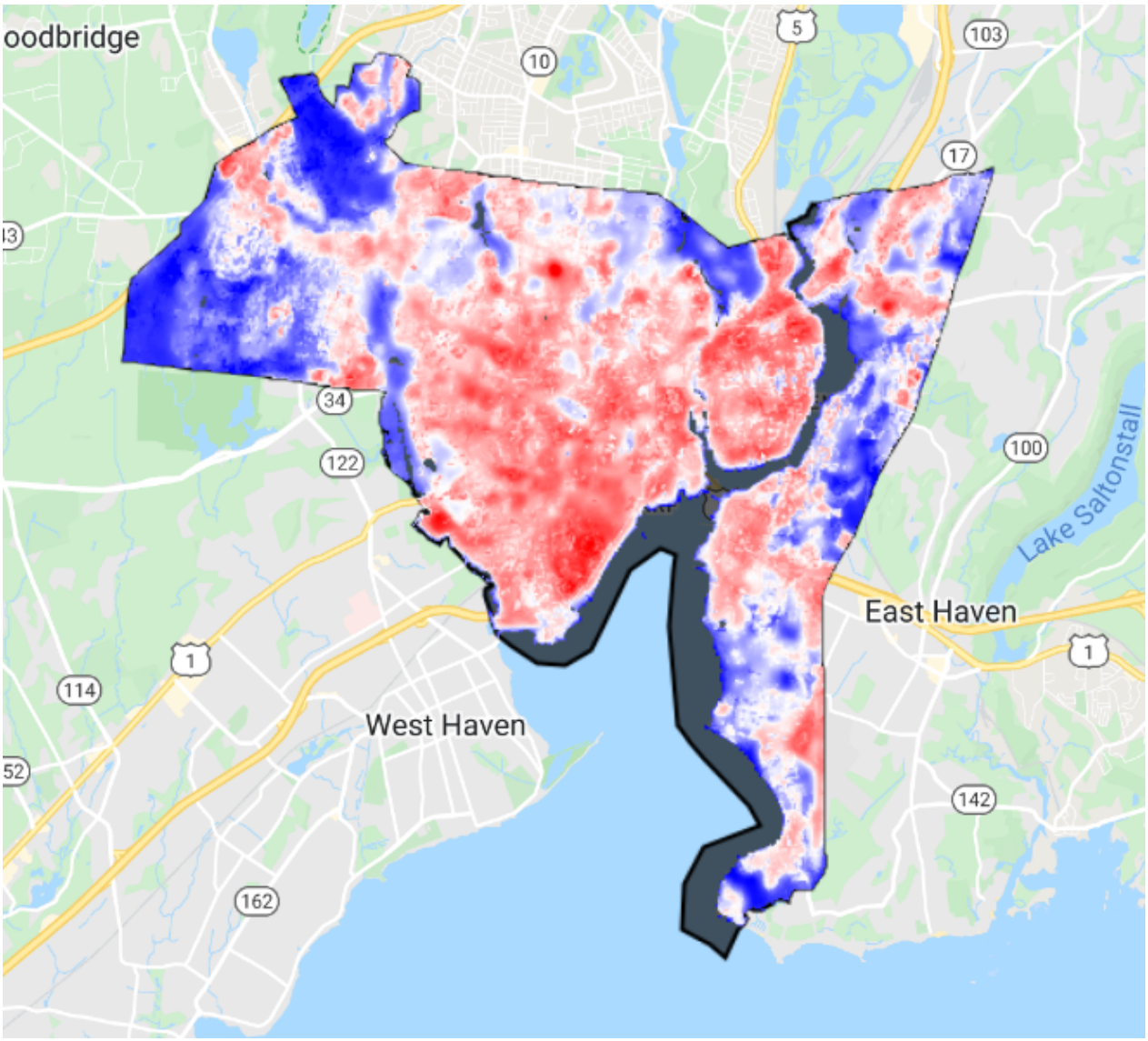


Fig. A1.5.4 Five-year summer median composite of Landsat brightness temperature over New Haven, Connecticut. Red pixels show higher values, and blue pixels have lower values.

Brightness temperature (Fig. A1.5.4) is the temperature equivalent of the infrared radiation escaping the top of the atmosphere, assuming the Earth to be a black body. It is not the same as the LST, which requires accounting for atmospheric absorption and re-emission, as well as the emissivity of the land surface. One way to derive pixel-level emissivity is as a function of the vegetation fraction of the pixel (Malakar et al. 2018). For

this, we start by calculating the Normalized Difference Vegetation Index (NDVI) from the Landsat surface reflectance data (see Fig. A1.5.5).

```
// Calculate Normalized Difference Vegetation Index (NDVI)
// from Landsat surface reflectance.
var ndvi = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2')
    .filterBounds(regionInt)
    .filterDate('2014-01-01', '2019-01-01')
    .filter(sumFilter)
    .median()
    .normalizedDifference(['SR_B5', 'SR_B4']).rename('NDVI')
    .clip(regionInt)
    .updateMask(notWater);

Map.addLayer(ndvi, {
        min: 0,
        max: 1,
        palette: ['blue', 'white', 'green']
    },
    'ndvi');
```

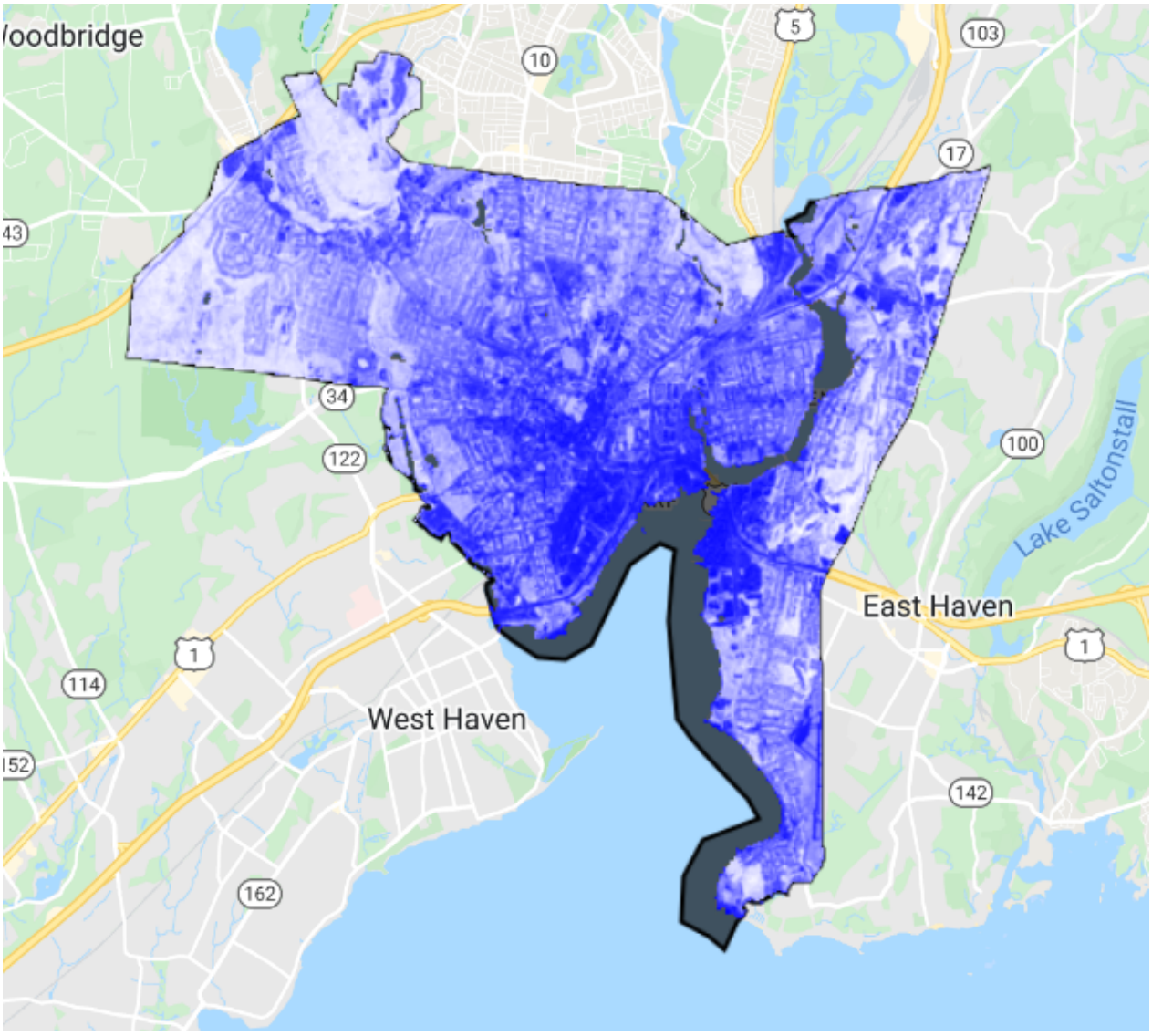


Fig. A1.5.5 Five-year summer median composite of Landsat-derived NDVI over New Haven, Connecticut. White pixels show higher NDVI values, and blue pixels have lower values.

To map NDVI for each pixel to the actual fraction of the pixel with vegetation (fractional vegetation cover), we next use a relationship based on the range of NDVI values for each pixel.

```
// Find the minimum and maximum of NDVI.  Combine the reducers
// for efficiency (single pass over the data).
var minMax = ndvi.reduceRegion({
    reducer: ee.Reducer.min().combine({
        reducer2: ee.Reducer.max(),
        sharedInputs: true
    }),
    geometry: regionInt,
    scale: 30,
    maxPixels: 1e9
});
print('minMax', minMax);

var min = ee.Number(minMax.get('NDVI_min'));
var max = ee.Number(minMax.get('NDVI_max'));

// Calculate fractional vegetation.
var fv = ndvi.subtract(min).divide(max.subtract(min)).rename('FV');
Map.addLayer(fv, {
    min: 0,
    max: 1,
    palette: ['blue', 'white', 'green']
}, 'fv');
```

Now we use an empirical model of emissivity based on this fractional vegetation cover (Sekertekin and Bonafoni 2020).

```
// Emissivity calculations.
var a = ee.Number(0.004);
var b = ee.Number(0.986);
var em = fv.multiply(a).add(b).rename('EMM').updateMask(notWater);

Map.addLayer(em, {
        min: 0.98,
        max: 0.99,
        palette: ['blue', 'white', 'green']
```

```
    },
    'EMM');
```

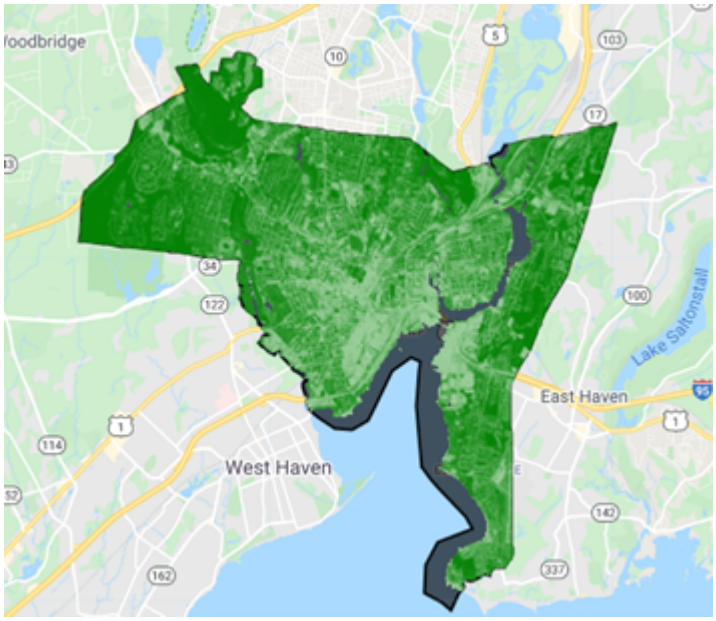


Fig. A1.5.6 Surface emissivity over New Haven, Connecticut, based on vegetation fraction. Green pixels show higher values, and white pixels have lower values.

As seen in Fig. A1.5.6, emissivity is lower over the built-up structures compared to over vegetation, which is expected. Note that different models of estimating emissivity would lead to some differences in LST values as well as the SUHI intensity (Sekertekin and Bonafoni 2020, Chakraborty et al. 2021a).

Then we combine this emissivity with the brightness temperature to calculate the LST for each pixel using a simple single-channel algorithm, which is a linearized approximation of the radiation transfer equation (Ermida et al. 2020).

```
// Calculate LST from emissivity and brightness temperature.
var lstLandsat = thermal.expression(
    '(Tb/(1 + (0.001145* (Tb / 1.438))*log(Ep)))-273.15', {
        'Tb': thermal.select('B10'),
        'Ep': em.select('EMM')
    }).updateMask(notWater);
```

```
Map.addLayer(lstLandsat, {
        min: 25,
        max: 35,
        palette: ['blue', 'white', 'red'],
    },
    'LST_Landsat');
```

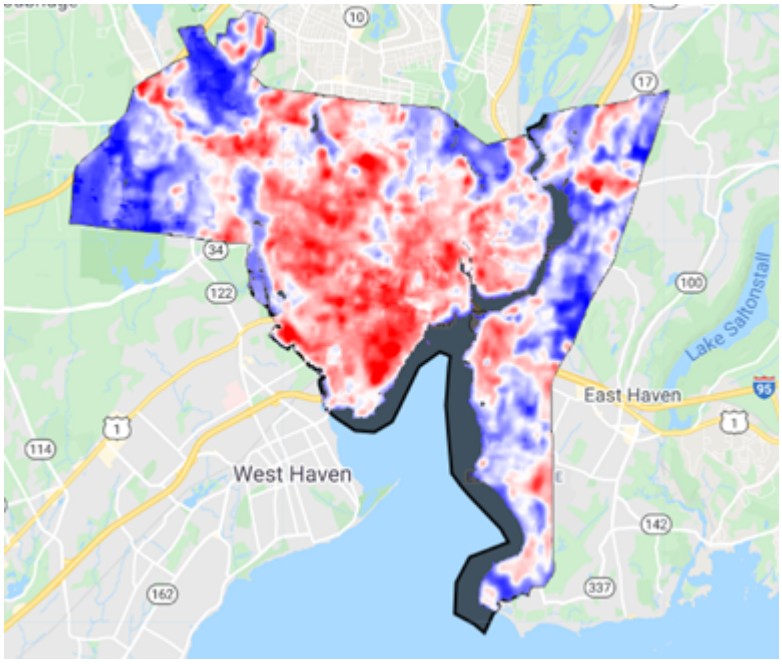


Fig. A1.5.7 Five-year summer median composite of Landsat-derived LST over New Haven, Connecticut. Red pixels show higher LST values, and blue pixels have lower values.

The Landsat-derived values correspond to those of the MODIS Terra daytime overpass. Overall, you do see similar patterns in Figs. A1.5.3 and A1.5.7, but Landsat picks up a lot more heterogeneity than MODIS due to its finer resolution.

Code Checkpoint A15b. The book's repository contains a script that shows what your code should look like at this point.

Section 1.3. Deriving Land Surface Temperature Using the Earth Engine Landsat LST Toolbox

In the previous section, we explored an LST retrieval algorithm to give an example of the standard steps to get to LST from the satellite measurements in the thermal bands. In this section, we will use an Earth Engine module developed for this purpose to calculate LST (Fig. A1.5.8).

```
// Link to the module that computes the Landsat LST.
var landsatLST = require(
    'projects/gee-edu/book:Part A - Applications/A1 - Human
Applications/A1.5 Heat Islands/modules/Landsat_LST.js');

// Select region of interest, date range, and Landsat satellite.
var geometry = regionInt.geometry();
var satellite = 'L8';
var dateStart = '2014-01-01';
var dateEnd = '2019-01-01';
var useNdvi = true;

// Get Landsat collection with additional necessary variables.
var landsatColl = landsatLST.collection(satellite, dateStart, dateEnd,
    geometry, useNdvi);

// Create composite, clip, filter to summer, mask, and convert to
degree Celsius.
var landsatComp = landsatColl
    .select('LST')
    .filter(sumFilter)
    .median()
    .clip(regionInt)
    .updateMask(notWater)
    .subtract(273.15);

Map.addLayer(landsatComp, {
        min: 25,
        max: 38,
```

```
        palette: ['blue', 'white', 'red']
    },
    'LST_SMW');
```

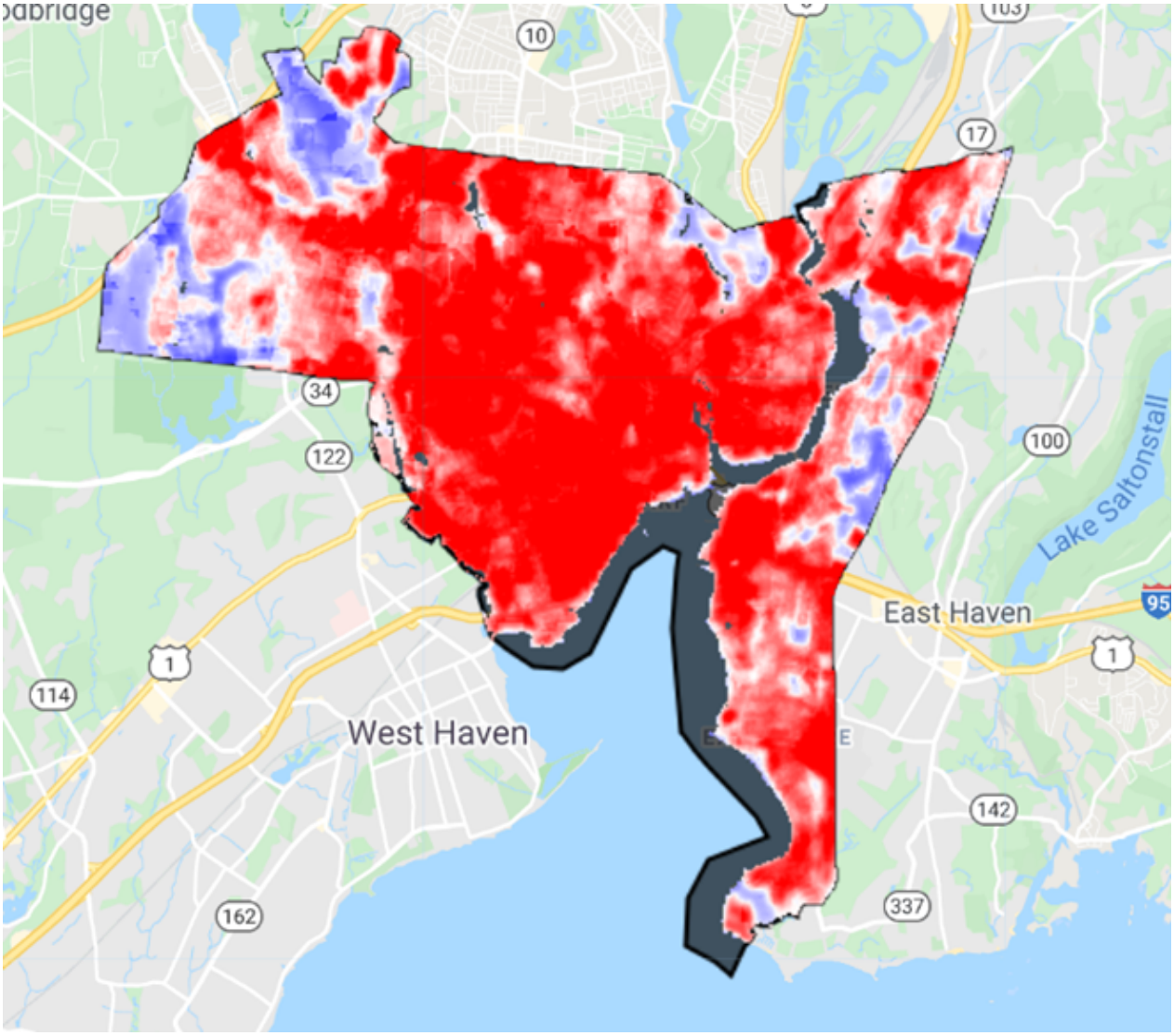


Fig. A1.5.8 Five-year summer median composite of Landsat-derived LST over New Haven, Connecticut, using the Statistical Mono-Window algorithm. Red pixels show higher LST values, and blue pixels have lower values.

As an aside, the Landsat Collection 2 products have recently incorporated LST bands, which can be processed similar to the MODIS data, but with the bands' own specific offsets and scaling factors.

Code Checkpoint A15c. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Defining Urban and Rural References

Now that we have estimates of LST using various products and algorithms, we can calculate the rural LST and subtract from the urban LST to get the SUHI intensity. There are many ways to estimate the rural reference temperature (Li et al. 2022), and we will explore a few of them in this section.

The simplest and probably the most commonly used method to get the rural reference when calculating the SUHI is to generate a buffered area around the urban boundary. The exact width of the buffer varies across studies, with buffers of 2–30 km in width being used in previous studies (Clinton and Gong 2013, Venter et al. 2021, Yao et al. 2019). In Earth Engine, generating such a buffer is simple:

```
// Function to subtract the original urban cluster from the buffered
cluster
// to generate rural references.
function bufferSubtract(feature) {
    return ee.Feature(feature.geometry()
        .buffer(2000)
        .difference(feature.geometry()));
}

var ruralRef = regionInt.map(bufferSubtract);

Map.addLayer(ruralRef, {
    color: 'green'
}, 'Buffer_ref');
```

In the script above, a buffered polygon with a 2 km width is generated around the urban boundary, and the original urban boundary is subtracted from the buffered polygon. The result is shown in Fig. A1.5.9.

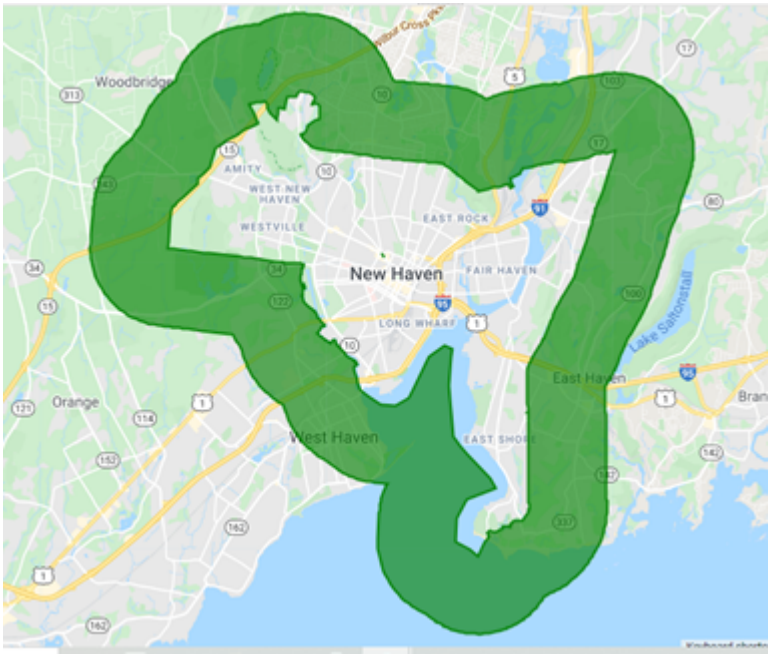


Fig. A1.5.9 A 2 km buffer around the original city boundary to serve as the rural reference

The use of a constant buffer assumes that all urban areas, regardless of size, have a similar influence around the city. This may not be true for large cities. In fact, there is some evidence that there is a footprint of the SUHI that is dependent on the size of the city (Yang et al. 2019, Zhou et al. 2015). As such, another way to define the buffered region is to normalize its area by the area of the urban cluster it surrounds (Chakraborty et al. 2021b, Peng et al. 2011). One way to do so in Earth Engine is by using an iterative method. This method (see code block below) uses functions to first calculate buffers of different widths around a geometry and then select the buffered region that is closest in size to the original geometry.

```
// Define sequence of buffer widths to be tested.
var buffWidths = ee.List.sequence(30, 3000, 30);

// Function to generate standardized buffers (approximately comparable
to area of urban cluster).
function bufferOptimize(feature) {
    function buff(buffLength) {
        var buffedPolygon = ee.Feature(feature.geometry()
                .buffer(ee.Number(buffLength)))
```

```
            .set({
                'Buffer_width': ee.Number(buffLength)
            });
        var area = buffedPolygon.geometry().difference(feature
            .geometry()).area();
        var diffFeature = ee.Feature(
            buffedPolygon.geometry().difference(feature
                .geometry()));
        return diffFeature.set({
            'Buffer_diff': area.subtract(feature.geometry()
                .area()).abs(),
            'Buffer_area': area,
            'Buffer_width': buffedPolygon.get('Buffer_width')
        });
    }

    var buffed = ee.FeatureCollection(buffWidths.map(buff));
    var sortedByBuffer = buffed.sort({
        property: 'Buffer_diff'
    });
    var firstFeature = ee.Feature(sortedByBuffer.first());
    return firstFeature.set({
        'Urban_Area': feature.get('Area'),
        'Buffer_width': firstFeature.get('Buffer_width')
    });
}

// Map function over urban feature collection.
var ruralRefStd = regionInt.map(bufferOptimize);

Map.addLayer(ruralRefStd, {
    color: 'brown'
}, 'Buffer_ref_std');

print('ruralRefStd', ruralRefStd);
```

Note how mapping the `buff` function over a sequence of pre-defined values, as done here, does not require loops, which are best avoided when using Earth Engine The same is true of mapping the `bufferOptimize` function: here it is mapped over a `FeatureCollection` with a single feature, but it would work even if `regionInt` contained multiple features. In this way, nested `map` functions in Earth Engine have the utility of nested loops in other languages.

Check the printed value on the **Console**. According to the result, within an uncertainty of 30 m, a buffer of 1170 m in width creates a polygon that is roughly equal to the area of the city. This function is best run via export when working with large feature collections.

The final way to define a rural reference does not use a buffer at all, but relies on land cover classes to select pixels that are urban versus non-urban (Chakraborty et al. 2020, Chakraborty and Lee 2019). For this, we will rely on the NLCD 2016 land cover data (Wickham et al. 2021) and create masks for urban and non-urban pixels (Fig. A1.5.10).

```
// Select the NLCD land cover data.
var landCover = ee.Image('USGS/NLCD/NLCD2016').select('landcover');
var urban = landCover;

// Select urban pixels in image.
var urbanUrban = urban.updateMask(urban.eq(23).or(urban.eq(24)));

// Select background reference pixels in the image.
var nonUrbanVals = [41, 42, 43, 51, 52, 71, 72, 73, 74, 81, 82];
var nonUrbanPixels = urban.eq(ee.Image(nonUrbanVals)).reduce('max');
var urbanNonUrban = urban.updateMask(nonUrbanPixels);

Map.addLayer(urbanUrban.clip(regionInt), {
    palette: 'red'
}, 'Urban pixels');
Map.addLayer(urbanNonUrban.clip(regionInt), {
    palette: 'blue'
}, 'Non-urban pixels');
```

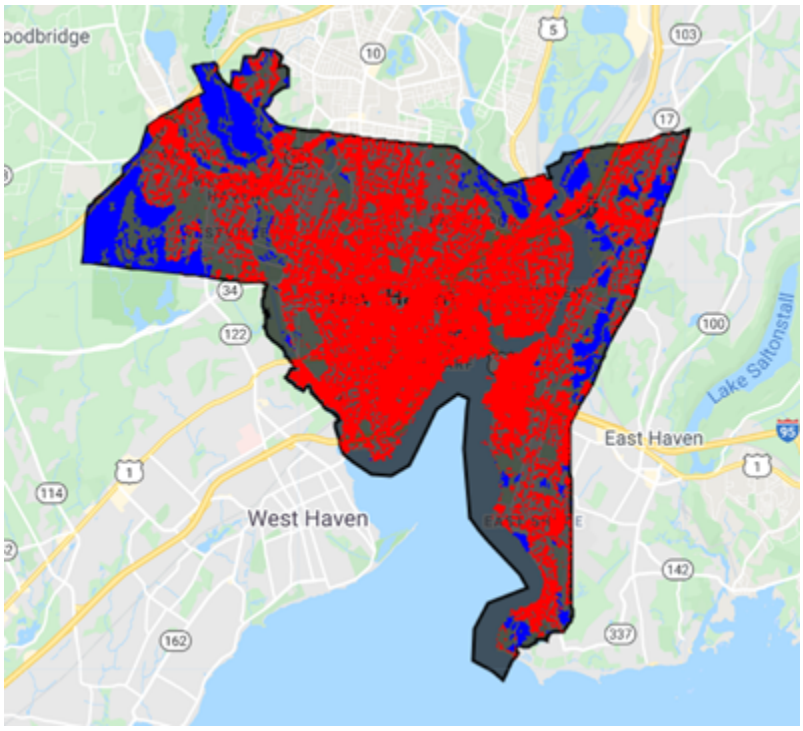


Fig. A1.5.10 Urban (red) and rural (blue) pixels in New Haven, Connecticut

We can then subsequently use these as masks to select urban versus rural LST pixels. You will find more about this in the next section.

Code Checkpoint A15d. The book's repository contains a script that shows what your code should look like at this point.

Section 3. Calculating the Surface Urban Heat Island Intensity

Since the SUHI is the temperature difference between the urban area and the rural reference, we will calculate summary temperature values for the urban boundary and the different versions of rural reference using the Landsat and MODIS LST.

```
// Define function to reduce regions and summarize pixel values
// to get mean LST for different cases.
function polygonMean(feature) {

    // Calculate spatial mean value of LST for each case
    // making sure the pixel values are converted to °C from Kelvin.
```

```
var reducedLstUrb = lstFinal.subtract(273.15).updateMask(notWater)
    .reduceRegion({
        reducer: ee.Reducer.mean(),
        geometry: feature.geometry(),
        scale: 30
    });
var reducedLstUrbMask = lstFinal.subtract(273.15).updateMask(
        notWater)
    .updateMask(urbanUrban)
    .reduceRegion({
        reducer: ee.Reducer.mean(),
        geometry: feature.geometry(),
        scale: 30
    });
var reducedLstUrbPix = lstFinal.subtract(273.15).updateMask(
        notWater)
    .updateMask(urbanUrban)
    .reduceRegion({
        reducer: ee.Reducer.mean(),
        geometry: feature.geometry(),
        scale: 500
    });
var reducedLstLandsatUrbPix = landsatComp.updateMask(notWater)
    .updateMask(urbanUrban)
    .reduceRegion({
        reducer: ee.Reducer.mean(),
        geometry: feature.geometry(),
        scale: 30
    });
var reducedLstRurPix = lstFinal.subtract(273.15).updateMask(
        notWater)
    .updateMask(urbanNonUrban)
    .reduceRegion({
        reducer: ee.Reducer.mean(),
        geometry: feature.geometry(),
        scale: 500
    });
```

```
    var reducedLstLandsatRurPix = landsatComp.updateMask(notWater)
        .updateMask(urbanNonUrban)
        .reduceRegion({
            reducer: ee.Reducer.mean(),
            geometry: feature.geometry(),
            scale: 30
        });

    // Return each feature with the summarized LSY values as
properties.
    return feature.set({
        'MODIS_LST_urb': reducedLstUrb.get('LST_Day_1km'),
        'MODIS_LST_urb_mask': reducedLstUrbMask.get(
            'LST_Day_1km'),
        'MODIS_LST_urb_pix': reducedLstUrbPix.get(
            'LST_Day_1km'),
        'MODIS_LST_rur_pix': reducedLstRurPix.get(
            'LST_Day_1km'),
        'Landsat_LST_urb_pix': reducedLstLandsatUrbPix.get(
            'LST'),
        'Landsat_LST_rur_pix': reducedLstLandsatRurPix.get(
            'LST')
    });
}

// Map the function over the urban boundary to get mean urban and
rural LST
// for cases without any explicit buffer-based boundaries.
var reduced = regionInt.map(polygonMean);
```

As you know from the code above, we extract urban temperature from MODIS (`'MODIS_LST_urb`) and from Landsat and MODIS (`'MODIS_LST_urb_pix'` and `'Landsat_LST_urb_pix'`) after considering only the urban pixels within the boundary.

Corresponding values are also extracted from the rural reference (including using only rural reference pixels within the urban boundary). For the buffered regions (both using constant width and variable width), we define and call another function.

```
// Define a function to reduce region and summarize pixel values
// to get mean LST for different cases.
function refMean(feature) {
    // Calculate spatial mean value of LST for each case
    // making sure the pixel values are converted to °C from Kelvin.
    var reducedLstRur = lstFinal.subtract(273.15).updateMask(notWater)
        .reduceRegion({
            reducer: ee.Reducer.mean(),
            geometry: feature.geometry(),
            scale: 30
        });
    var reducedLstRurMask = lstFinal.subtract(273.15).updateMask(
            notWater)
        .updateMask(urbanNonUrban)
        .reduceRegion({
            reducer: ee.Reducer.mean(),
            geometry: feature.geometry(),
            scale: 30
        });
    return feature.set({
        'MODIS_LST_rur': reducedLstRur.get('LST_Day_1km'),
        'MODIS_LST_rur_mask': reducedLstRurMask.get(
            'LST_Day_1km'),
    });
}

// Map the function over the constant buffer rural reference boundary
one.
var reducedRural = ee.FeatureCollection(ruralRef).map(refMean);

// Map the function over the standardized rural reference boundary.
var reducedRuralStd = ruralRefStd.map(refMean);

print('reduced', reduced);
print('reducedRural', reducedRural);
print('reducedRuralStd', reducedRuralStd);
```

We can print the newly created feature collections to go through these values for the different cases. Even though absolute MODIS and Landsat urban LSTs are different (29 for Landsat and 34 for MODIS), the SUHI is similar (3.6 C for MODIS and 4.8 C from Landsat). As one might expect, when only urban pixels are considered within the boundary, the average LST is higher (and lower for rural LST).

The SUHI variability within the city (Fig. A1.5.11) can then be displayed by subtracting the rural LST from the total LST:

```
// Display SUHI variability within the city.
var suhi = landsatComp
    .updateMask(urbanUrban)
    .subtract(ee.Number(ee.Feature(reduced.first())
        .get('Landsat_LST_rur_pix')));

Map.addLayer(suhi, {
    palette: ['blue', 'white', 'red'],
    min: 2,
    max: 8
}, 'SUHI');
```

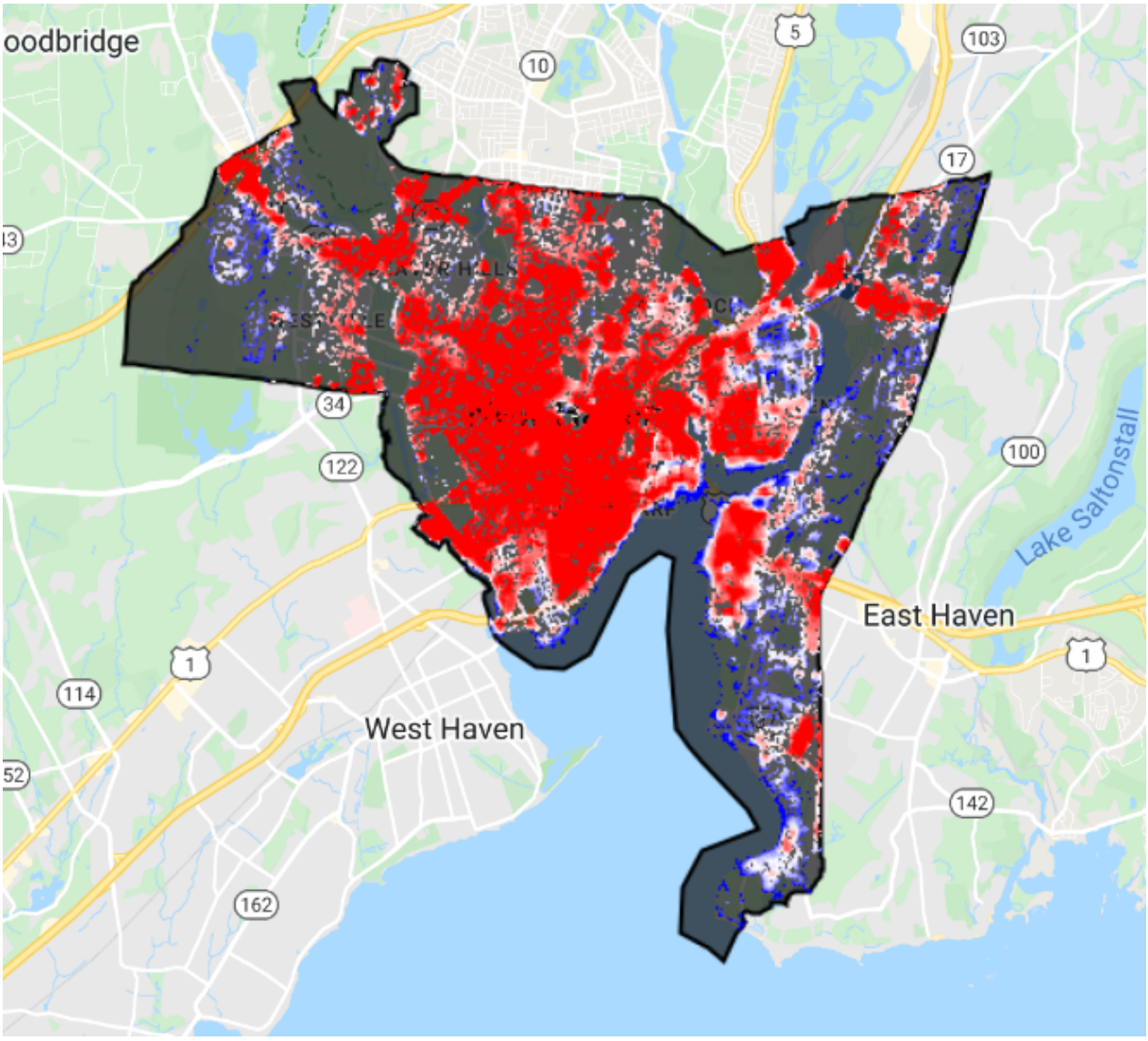


Fig. A1.5.11 Spatial variability in SUHI for the period of interest for New Haven, Connecticut. Red pixels show higher values and blue pixels show lower values.

Code Checkpoint A15e. The book's repository contains a script that shows what your code should look like at this point.

Synthesis

Now that you know the different ways to calculate the SUHI and estimate LST using Earth Engine, load in your own city's feature collection and compare the different methods.

Question 1. What are the SUHI values during summer and winter from the two products?

Question 2. How does a pixel-based urban-rural delineation method compare to a buffer-based method for SUHI estimation?

Conclusion

You should now have a good understanding of satellite measurements in the thermal bands, how they can be used to estimate LST, and how we can calculate the SUHI using these measurements.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Arnfield AJ (2003) Two decades of urban climate research: A review of turbulence, exchanges of energy and water, and the urban heat island. Int J Climatol 23:1–26. https://doi.org/10.1002/joc.859

Chakraborty TC, Lee X, Ermida S, Zhan W (2021) On the land emissivity assumption and Landsat-derived surface urban heat islands: A global analysis. Remote Sens Environ 265:112682. https://doi.org/10.1016/j.rse.2021.112682

Chakraborty TC, Sarangi C, Lee X (2021) Reduction in human activity can enhance the urban heat island: Insights from the COVID-19 lockdown. Environ Res Lett 16:54060. https://doi.org/10.1088/1748-9326/abef8e

Chakraborty T, Hsu A, Manya D, Sheriff G (2020) A spatially explicit surface urban heat island database for the United States: Characterization, uncertainties, and possible applications. ISPRS J Photogramm Remote Sens 168:74–88. https://doi.org/10.1016/j.isprsjprs.2020.07.021

Chakraborty T, Lee X (2019) A simplified urban-extent algorithm to characterize surface urban heat islands on a global scale and examine vegetation control on their spatiotemporal variability. Int J Appl Earth Obs Geoinf 74:269–280. https://doi.org/10.1016/j.jag.2018.09.015

Chakraborty T, Sarangi C, Tripathi SN (2017) Understanding diurnality and inter-seasonality of a sub-tropical urban heat island. Boundary-Layer Meteorol 163:287–309. https://doi.org/10.1007/s10546-016-0223-0

Clinton N, Gong P (2013) MODIS detected surface urban heat islands and sinks: Global locations and controls. Remote Sens Environ 134:294–304. https://doi.org/10.1016/j.rse.2013.03.008

Ermida SL, Soares P, Mantas V, et al (2020) Google Earth Engine open-source code for land surface temperature estimation from the Landsat series. Remote Sens 12:1471. https://doi.org/10.3390/RS12091471

Howard L (1833) The Climate of London: Deduced from Meteorological Observations Made in the Metropolis and at Various Places Around it. Harvey and Darton, J. and A. Arch, Longman, Hatchard, S. Highley and R. Hunter

Hu Y, Hou M, Jia G, et al (2019) Comparison of surface and canopy urban heat islands within megacities of eastern China. ISPRS J Photogramm Remote Sens 156:160–168. https://doi.org/10.1016/j.isprsjprs.2019.08.012

Kalnay E, Cai M (2003) Impact of urbanization and land-use change on climate. Nature 425:102. https://doi.org/10.1038/nature01952

Li K, Chen Y, Gao S (2022) Uncertainty of city-based urban heat island intensity across 1112 global cities: Background reference and cloud coverage. Remote Sens Environ 271:112898. https://doi.org/10.1016/j.rse.2022.112898

Li ZL, Wu H, Wang N, et al (2013) Land surface emissivity retrieval from satellite data. Int J Remote Sens 34:3084–3127. https://doi.org/10.1080/01431161.2012.716540

Malakar NK, Hulley GC, Hook SJ, et al (2018) An operational land surface temperature product for Landsat thermal data: Methodology and validation. IEEE Trans Geosci Remote Sens 56:5717–5735. https://doi.org/10.1109/TGRS.2018.2824828

Ngie A, Abutaleb K, Ahmed F, et al (2014) Assessment of urban heat island using satellite remotely sensed imagery: A review. South African Geogr J 96:198–214. https://doi.org/10.1080/03736245.2014.924864

Oke TR (1982) The energetic basis of the urban heat island. Q J R Meteorol Soc 108:1–24. https://doi.org/10.1002/qj.49710845502

Pekel JF, Cottam A, Gorelick N, Belward AS (2016) High-resolution mapping of global surface water and its long-term changes. Nature 540:418–422. https://doi.org/10.1038/nature20584

Peng S, Piao S, Ciais P, et al (2012) Surface urban heat island across 419 global big cities. Environ Sci Technol 46:6889–6890. https://doi.org/10.1021/es301811b

Qian Y, Chakraborty TC, Li J, et al (2022) Urbanization impact on regional climate and extreme weather: Current understanding, uncertainties, and future research directions. Adv Atmos Sci 39:819–860. https://doi.org/10.1007/s00376-021-1371-9

Sekertekin A, Bonafoni S (2020) Sensitivity analysis and validation of daytime and nighttime land surface temperature retrievals from Landsat 8 using different algorithms and emissivity models. Remote Sens 12:2776. https://doi.org/10.3390/RS12172776

Venter ZS, Chakraborty T, Lee X (2021) Crowdsourced air temperatures contrast satellite measures of the urban heat island and its mechanisms. Sci Adv 7:eabb9569. https://doi.org/10.1126/sciadv.abb9569

Wan Z (2006) MODIS land surface temperature products users' guide. Inst Comput Earth Syst Sci Univ Calif St Barbar CA, USA 805

Wickham J, Stehman SV, Sorenson DG, et al (2021) Thematic accuracy assessment of the NLCD 2016 land cover for the conterminous United States. Remote Sens Environ 257:112357. https://doi.org/10.1016/j.rse.2021.112357

Yang Q, Huang X, Tang Q (2019) The footprint of urban heat island effect in 302 Chinese cities: Temporal trends and associated factors. Sci Total Environ 655:652–662. https://doi.org/10.1016/j.scitotenv.2018.11.171

Yao R, Wang L, Huang X, et al (2019) Greening in rural areas increases the surface urban heat island intensity. Geophys Res Lett 46:2204–2212. https://doi.org/10.1029/2018GL081816

Zhou D, Zhao S, Zhang L, et al (2015) The footprint of urban heat island effect in China. Sci Rep 5:1–11. https://doi.org/10.1038/srep11160

Chapter A1.6: Health Applications

Author
Dawn Nekorchuk

Overview

The purpose of this chapter is to demonstrate how Google Earth Engine may be used to support modeling and forecasting of vector-borne infectious diseases such as malaria. In doing so, the chapter will also show how Earth Engine may be used to gather data for subsequent analyses outside of Earth Engine, the results of which can then also be brought back into Earth Engine.

We will be calculating and exporting data of remotely-sensed environmental variables: precipitation, temperature, and a vegetation water index. These factors can impact mosquito life cycles, malaria parasites, and transmission dynamics. These data can then be used in R for modeling and forecasting malaria in the Amhara region of Ethiopia, using the Epidemic Prognosis Incorporating Disease and Environmental Monitoring for Integrated Assessment (EPIDEMIA) system, developed by the EcoGRAPH research group at the University of Oklahoma.

Learning Outcomes

- Extracting and calculating malaria-relevant variables from existing data sets: precipitation, temperature, and wetness.
- Importing satellite data and filtering for images over a region and time period.
- Joining two data products to get additional quality information.
- Computing zonal summaries of the calculated variables for elements in a `FeatureCollection`.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Perform basic image analysis: select bands, compute indices, create masks (Part F2).
- Use expressions to perform calculations on image bands (Chap. F3.1).
- Write a function and `map` it over an `ImageCollection` (Chap. F4.0).
- Mask cloud, cloud shadow, snow/ice, and other undesired pixels (Chap. F4.3).

- Flatten a table for export to CSV (Chap. F5.0).
- Use `reduceRegions` to summarize an image with zonal statistics in irregular shapes (Chap. F5.0, Chap. F5.2).
- Write a function and `map` it over a `FeatureCollection` (Chap. F5.1, Chap. F5.2).

Introduction to Theory

Vector-borne diseases cause more than 700,000 deaths per year, of which approximately 400,000 are due to malaria, a parasitic infection spread by *Anopheles* mosquitoes (World Health Organization 2018, 2020). The WHO estimates that there were around 229 million clinical cases of malaria worldwide in 2019 (WHO 2020). Environmental factors including temperature, humidity, and rainfall are known to be important determinants of malaria risk as these affect mosquito and parasite development and life cycles, including larval habitats, mosquito fecundity, growth rates, mortality, and *Plasmodium* parasite development rates within the mosquito vector (Franklinos et al. 2019, Jones 2008, Wimberly et al. 2021).

Data from Earth-observing satellites can be used to monitor spatial and temporal changes in these environmental factors (Ford et al. 2009). These data can be incorporated into disease modeling, usually as lagged functions, to help develop early warning systems for forecasting outbreaks (Wimberly et al. 2021, 2022). Accurate forecasts would allow limited resources for prevention and control to be more efficiently and effectively targeted at appropriate locations and times (WHO 2018).

To implement near-real-time forecasting, meteorological and climatic data must be acquired, processed, and integrated on a regular and frequent basis. Over the past 10 years, the Epidemic Prognosis Incorporating Disease and Environmental Monitoring for Integrated Assessment (EPIDEMIA) project has developed and tested a malaria forecasting system that integrates public health surveillance with monitoring of environmental and climate conditions. Since 2018 the environmental data has been acquired using Earth Engine scripts and apps (Wimberly et al. 2022). In 2019 a local team at Bahir Dar University in Ethiopia had been using EPIDEMIA with near-real-time epidemiological data to generate weekly malaria early warning reports in the Amhara region of Ethiopia.

In this example, we are looking at near-real-time environmental conditions that affect disease vectors and human transmission dynamics. On longer time scales, issues such as climate change can alter vector-borne disease transmission cycles and the geographic distributions of various vector and host species (Franklinos 2019). More

broadly, health applications involving Earth Engine data likely align with a One Health approach to complex health issues. Under One Health, a core assumption is that environmental, animal, and human health are inextricably linked (Mackenzie and Jeggo 2019).

Practicum

The goal of the practicum is to create a download of three environmental variables:

1. Precipitation
2. Mean land surface temperature (LST)
3. Normalized Difference Water Index (NDWI) spectral index.

These downloads will be zonal summaries based on our uploaded shapefile of *woredas* (districts) in the Amhara region of Ethiopia.

The practicum is an extract from the longer Retrieving Environmental Analytics for Climate and Health (REACH) Earth Engine script (developed by Dr. Michael C. Wimberly and Dr. Dawn Nekorchuk) used in the EPIDEMIA project (Dr. Michael C. Wimberly, PI). This script also has a more advanced user interface for the user to request date ranges for the download of data. Links to this script and related apps can be found in the "For Further Reading" section of this book.

Section 1. Data Import

To start, we need to import the data we will be working with. The first item is an external asset of our study area—these are *woredas* in the Amhara region of Ethiopia. The four that follow are remotely sensed data that we will be processing:

- The Integrated Multi-satellite Retrievals for GPM (IMERG) rainfall estimates from Global Precipitation Measurement (GPM) v6
- Terra Land Surface Temperature and Emissivity 8-Day Global 1km
- MODIS Nadir BRDF (Bidirectional Reflectance Distribution Function) Adjusted Reflectance Daily 500m
- MODIS BRDF-Albedo Quality Daily 500m

```
// Section 1: Data Import
var woredas = ee.FeatureCollection(
    'projects/gee-book/assets/A1-6/amhara_woreda_20170207');
```

```
// Create region outer boundary to filter products on.
var amhara = woredas.geometry().bounds();
var gpm = ee.ImageCollection('NASA/GPM_L3/IMERG_V06');
var LSTTerra8 = ee.ImageCollection('MODIS/061/MOD11A2')
    // Due to MCST outage, only use dates after this for this script.
    .filterDate('2001-06-26', Date.now());
var brdfReflect = ee.ImageCollection('MODIS/006/MCD43A4');
var brdfQa = ee.ImageCollection('MODIS/006/MCD43A2');
```

We can take a look at the *woreda* boundaries by adding the following code to draw it onto the map (Fig. A1.6.1). See Chap. F5.3 for more information on visualizing feature collections.

```
// Visualize woredas with black borders and no fill.
// Create an empty image into which to paint the features, cast to
byte.
var empty = ee.Image().byte();
// Paint all the polygon edges with the same number and width.
var outline = empty.paint({
    featureCollection: woredas,
    color: 1,
    width: 1
});
// Add woreda boundaries to the map.
Map.setCenter(38, 11.5, 7);
Map.addLayer(outline, {
    palette: '000000'
}, 'Woredas');
```

Code Checkpoint A16a. The book's repository contains a script that shows what your code should look like at this point.

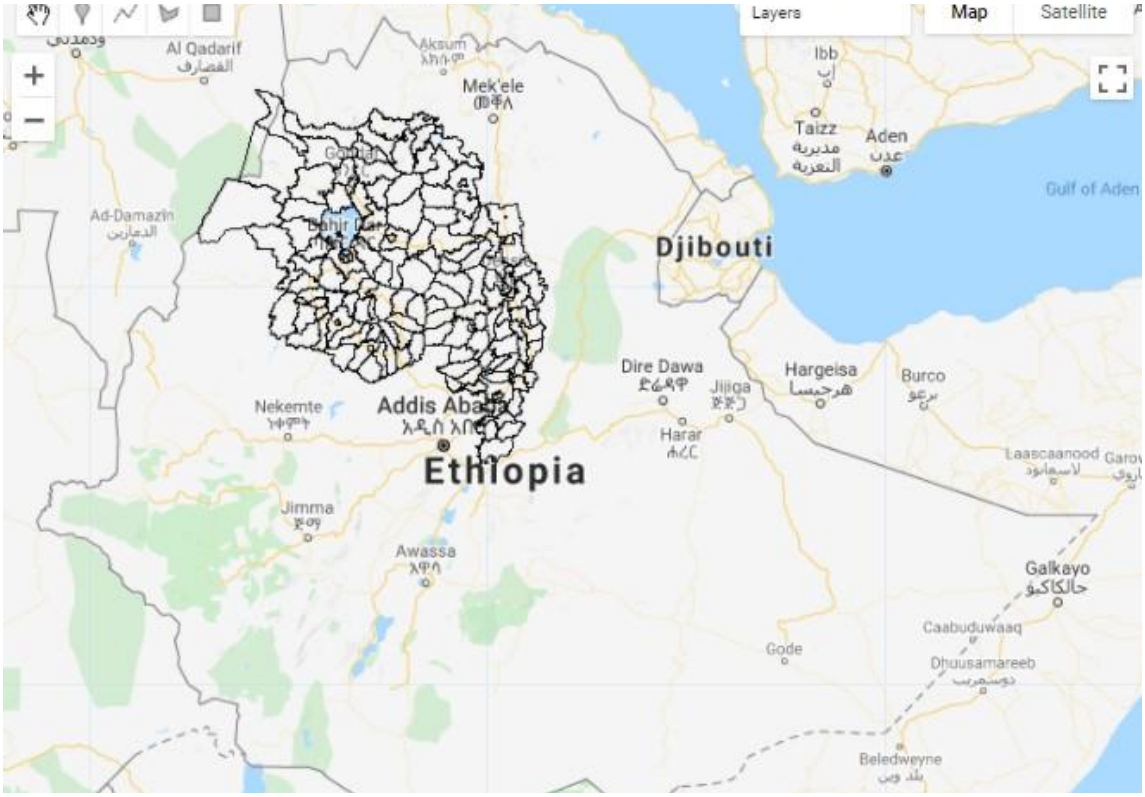


Fig. A1.6.1 *Woreda* (district) boundaries in the Amhara region of Ethiopia

Section 2. Date Preparation

The user will be requesting the date range for the summarized data, and it is expected that they will be looking for near-real-time data. Different data products that we are using have different data lags, and some data may not be available in the user-requested date range. We will want to get the last available data date so we can properly create and name our export datasets.

We need daily data, but the LST data are in 8-day composites. For this, we will assign the 8-day composite value to each of the eight days in the range. This means we also need to acquire the 8-day composite value that covers the requested start date (i.e., the previous image).

```
// Section 2: Handling of dates

// 2.1 Requested start and end dates.
var reqStartDate = ee.Date('2021-10-01');
var reqEndDate = ee.Date('2021-11-30');

// 2.2 LST Dates
// LST MODIS is every 8 days, and a user-requested date will likely
not match.
// We want to get the latest previous image date,
//  i.e. the date the closest, but prior to, the requested date.
// We will filter later.
// Get date of first image.
var LSTEarliestDate = LSTTerra8.first().date();
// Filter collection to dates from beginning to requested start date.
var priorLstImgCol = LSTTerra8.filterDate(LSTEarliestDate,
    reqStartDate);
// Get the latest (max) date of this collection of earlier images.
var LSTPrevMax = priorLstImgCol.reduceColumns({
    reducer: ee.Reducer.max(),
    selectors: ['system:time_start']
});
var LSTStartDate = ee.Date(LSTPrevMax.get('max'));
print('LSTStartDate', LSTStartDate);

// 2.3 Last available data dates
// Different variables have different data lags.
// Data may not be available in user range.
// To prevent errors from stopping script,
//  grab last available (if relevant) & filter at end.

// 2.3.1 Precipitation
// Calculate date of most recent measurement for gpm (of all time).
var gpmAllMax = gpm.reduceColumns(ee.Reducer.max(), [
    'system:time_start'
]);
var gpmAllEndDateTime = ee.Date(gpmAllMax.get('max'));
```

```
// GPM every 30 minutes, so get just date part.
var gpmAllEndDate = ee.Date.fromYMD({
    year: gpmAllEndDateTime.get('year'),
    month: gpmAllEndDateTime.get('month'),
    day: gpmAllEndDateTime.get('day')
});

// If data ends before requested start, take last data date,
// otherwise use requested date.
var precipStartDate = ee.Date(gpmAllEndDate.millis()
    .min(reqStartDate.millis()));
print('precipStartDate', precipStartDate);

// 2.3.2 BRDF
// Calculate date of most recent measurement for brdf (of all time).
var brdfAllMax = brdfReflect.reduceColumns({
    reducer: ee.Reducer.max(),
    selectors: ['system:time_start']
});
var brdfAllEndDate = ee.Date(brdfAllMax.get('max'));
// If data ends before requested start, take last data date,
// otherwise use the requested date.
var brdfStartDate = ee.Date(brdfAllEndDate.millis()
    .min(reqStartDate.millis()));
print('brdfStartDate', brdfStartDate);
print('brdfEndDate', brdfAllEndDate);
```

Code Checkpoint A16b. The book's repository contains a script that shows what your code should look like at this point.

Question 1. Explore the earliest date of LST images you get if you do not specifically acquire the previous image. The following code may be useful:

```
var naiveLstFilter = LSTTerra8.filterDate(reqStartDate, reqEndDate);
var naiveLstStart = naiveLstFilter.reduceColumns({
    reducer: ee.Reducer.min(),
```

```
    selectors: ['system:time_start']
});
var naiveLstStartDate = ee.Date(naiveLstStart.get('min'));
print('naiveLstStartDate', naiveLstStartDate);
```

Question 2. Try changing the requested dates to closer to the current date to see how the dates for the different data products adjust. If you have a narrow window (1–2 weeks), you may find that some data products do not have any data available for the requested time period yet.

Section 3. Precipitation

Now we will calculate our precipitation variable for the appropriate date range and then perform a zonal summary (see Chap. F5.2) of our *woredas*.

Section 3.1. Precipitation Filtering and Dates

Using the dates when data actually exists in the user-requested date range, we create a list of dates for which we will calculate our variable.

```
// Section 3: Precipitation

// Section 3.1: Precipitation filtering and dates

// Filter gpm by date, using modified start if necessary.
var gpmFiltered = gpm
    .filterDate(precipStartDate, reqEndDate.advance(1, 'day'))
    .filterBounds(amhara)
    .select('precipitationCal');

// Calculate date of most recent measurement for gpm
// (in the modified requested window).
var gpmMax = gpmFiltered.reduceColumns({
    reducer: ee.Reducer.max(),
    selectors: ['system:time_start']
});
```

```
var gpmEndDate = ee.Date(gpmMax.get('max'));
var precipEndDate = gpmEndDate;
print('precipEndDate ', precipEndDate);

// Create a list of dates for the precipitation time series.
var precipDays = precipEndDate.difference(precipStartDate, 'day');
var precipDatesPrep = ee.List.sequence(0, precipDays, 1);

function makePrecipDates(n) {
    return precipStartDate.advance(n, 'day');
}
var precipDates = precipDatesPrep.map(makePrecipDates);
```

Section 3.2. Calculate Daily Precipitation

In this section, we will map a function over our filtered `FeatureCollection` (`gpmFiltered`) to calculate the total daily rainfall per day. In this product, precipitation in millimeters per hour is recorded every half hour, so we will sum the day and divide by two.

```
// Section 3.2: Calculate daily precipitation

// Function to calculate daily precipitation:
function calcDailyPrecip(curdate) {
    curdate = ee.Date(curdate);
    var curyear = curdate.get('year');
    var curdoy = curdate.getRelative('day', 'year').add(1);
    var totprec = gpmFiltered
        .filterDate(curdate, curdate.advance(1, 'day'))
        .select('precipitationCal')
        .sum()
        //every half-hour
        .multiply(0.5)
        .rename('totprec');

    return totprec
        .set('doy', curdoy)
        .set('year', curyear)
```

```
        .set('system:time_start', curdate);
}
// Map function over list of dates.
var dailyPrecipExtended =
    ee.ImageCollection.fromImages(precipDates.map(calcDailyPrecip));

// Filter back to the original user requested start date.
var dailyPrecip = dailyPrecipExtended
    .filterDate(reqStartDate, precipEndDate.advance(1, 'day'));
```

***Section 3.3. Summarize Daily Precipitation by* Woreda**

In the last section for precipitation, we will calculate a zonal summary, a mean, of the rainfall per *woreda* and flatten for export as a CSV. The exports (of all variables) will be all done in Sect. 7.

```
// Section 3.3: Summarize daily precipitation by woreda

// Filter precip data for zonal summaries.
var precipSummary = dailyPrecip
    .filterDate(reqStartDate, reqEndDate.advance(1, 'day'));

// Function to calculate zonal statistics for precipitation by woreda.
function sumZonalPrecip(image) {
    // To get the doy and year,
    // convert the metadata to grids and then summarize.
    var image2 = image.addBands([
        image.metadata('doy').int(),
        image.metadata('year').int()
    ]);
    // Reduce by regions to get zonal means for each county.
    var output = image2.select(['year', 'doy', 'totprec'])
        .reduceRegions({
            collection: woredas,
            reducer: ee.Reducer.mean(),
            scale: 1000
        });
    return output;
```

```
}
// Map the zonal statistics function over the filtered precip data.
var precipWoreda = precipSummary.map(sumZonalPrecip);
// Flatten the results for export.
var precipFlat = precipWoreda.flatten();
```

Code Checkpoint A16c. The book's repository contains a script that shows what your code should look like at this point.

Section 4. Land Surface Temperature

We will follow a similar pattern of steps for land surface temperatures, though first we will calculate the variable (mean LST). Then we will calculate the daily values and summarize them by *woreda*.

Section 4.1. Calculate LST Variables

We will use the daytime and nighttime observed values to calculate a mean value for the day. We will use the quality layers to mask out poor-quality pixels. Working with the bitmask below, we are taking advantage of the fact that bits 6 and 7 are at the end, so the `rightShift(6)` just returns these two. Then we check if they are less than or equal to 2, meaning average LST error <= 3k (see MODIS documentation for the meaning of each element in the bit sequence). For more information on how to use bitmasks in other situations, see Chap. F4.3. To convert the pixel values, we will use the scaling factor in the data product (0.2) and convert from Kelvin to Celsius values (−273.15). See Chap. F1.5, about Heat Islands, for another example using LST data.

```
// Section 4: Land surface temperature

// Section 4.1: Calculate LST variables

// Filter Terra LST by altered LST start date.
// Rarely, but at the end of the year if the last image is late in the
year
//  with only a few days in its period, it will sometimes not grab
//  the next image. Add extra padding to reqEndDate and
//  it will be trimmed at the end.
var LSTFiltered = LSTTerra8
```

```
    .filterDate(LSTStartDate, reqEndDate.advance(8, 'day'))
    .filterBounds(amhara)
    .select('LST_Day_1km', 'QC_Day', 'LST_Night_1km', 'QC_Night');

// Filter Terra LST by QA information.
function filterLstQa(image) {
    var qaday = image.select(['QC_Day']);
    var qanight = image.select(['QC_Night']);
    var dayshift = qaday.rightShift(6);
    var nightshift = qanight.rightShift(6);
    var daymask = dayshift.lte(2);
    var nightmask = nightshift.lte(2);
    var outimage = ee.Image(image.select(['LST_Day_1km',
        'LST_Night_1km'
    ]));
    var outmask = ee.Image([daymask, nightmask]);
    return outimage.updateMask(outmask);
}
var LSTFilteredQa = LSTFiltered.map(filterLstQa);

// Rescale temperature data and convert to degrees Celsius (C).
function rescaleLst(image) {
    var LST_day = image.select('LST_Day_1km')
        .multiply(0.02)
        .subtract(273.15)
        .rename('LST_day');
    var LST_night = image.select('LST_Night_1km')
        .multiply(0.02)
        .subtract(273.15)
        .rename('LST_night');
    var LST_mean = image.expression(
        '(day + night) / 2', {
            'day': LST_day.select('LST_day'),
            'night': LST_night.select('LST_night')
        }
    ).rename('LST_mean');
    return image.addBands(LST_day)
```

```
        .addBands(LST_night)
        .addBands(LST_mean);
}
var LSTVars = LSTFilteredQa.map(rescaleLst);
```

Section 4.2. Calculate Daily LST

Now, using a mapped function over our filtered collection, we will calculate a daily value from the 8-day composite value by assigning each of the eight days the value of the composite. We will also filter to our user-requested dates, as data exists in that range.

```
// Section 4.2: Calculate daily LST

// Create list of dates for time series.
var LSTRange = LSTVars.reduceColumns({
    reducer: ee.Reducer.max(),
    selectors: ['system:time_start']
});
var LSTEndDate = ee.Date(LSTRange.get('max')).advance(7, 'day');
var LSTDays = LSTEndDate.difference(LSTStartDate, 'day');
var LSTDatesPrep = ee.List.sequence(0, LSTDays, 1);

function makeLstDates(n) {
    return LSTStartDate.advance(n, 'day');
}
var LSTDates = LSTDatesPrep.map(makeLstDates);

// Function to calculate daily LST by assigning the 8-day composite
summary
// to each day in the composite period:
function calcDailyLst(curdate) {
    var curyear = ee.Date(curdate).get('year');
    var curdoy = ee.Date(curdate).getRelative('day', 'year').add(1);
    var moddoy = curdoy.divide(8).ceil().subtract(1).multiply(8).add(
        1);
    var basedate = ee.Date.fromYMD(curyear, 1, 1);
    var moddate = basedate.advance(moddoy.subtract(1), 'day');
    var LST_day = LSTVars
```

```
        .select('LST_day')
        .filterDate(moddate, moddate.advance(1, 'day'))
        .first()
        .rename('LST_day');
    var LST_night = LSTVars
        .select('LST_night')
        .filterDate(moddate, moddate.advance(1, 'day'))
        .first()
        .rename('LST_night');
    var LST_mean = LSTVars
        .select('LST_mean')
        .filterDate(moddate, moddate.advance(1, 'day'))
        .first()
        .rename('LST_mean');
    return LST_day
        .addBands(LST_night)
        .addBands(LST_mean)
        .set('doy', curdoy)
        .set('year', curyear)
        .set('system:time_start', curdate);
}
// Map the function over the image collection
var dailyLstExtended =
    ee.ImageCollection.fromImages(LSTDates.map(calcDailyLst));

// Filter back to original user requested start date
var dailyLst = dailyLstExtended
    .filterDate(reqStartDate, LSTEndDate.advance(1, 'day'));
```

***Section 4.3. Summarize Daily LST by* Woreda**

In the final section for LST, we will perform a zonal mean of the temperature to our *woredas* and flatten in preparation for export as CSV. The exports (of all variables) will be all done in Sect. 7.

```
// Section 4.3: Summarize daily LST by woreda

// Filter LST data for zonal summaries.
```

```
var LSTSummary = dailyLst
    .filterDate(reqStartDate, reqEndDate.advance(1, 'day'));
// Function to calculate zonal statistics for LST by woreda:
function sumZonalLst(image) {
    // To get the doy and year, we convert the metadata to grids
    //  and then summarize.
    var image2 = image.addBands([
        image.metadata('doy').int(),
        image.metadata('year').int()
    ]);
    // Reduce by regions to get zonal means for each county.
    var output = image2
        .select(['doy', 'year', 'LST_day', 'LST_night', 'LST_mean'])
        .reduceRegions({
            collection: woredas,
            reducer: ee.Reducer.mean(),
            scale: 1000
        });
    return output;
}
// Map the zonal statistics function over the filtered LST data.
var LSTWoreda = LSTSummary.map(sumZonalLst);
// Flatten the results for export.
var LSTFlat = LSTWoreda.flatten();
```

Code Checkpoint A16d. The book's repository contains a script that shows what your code should look like at this point.

Section 5. Spectral Index: NDWI

We will follow a similar pattern of steps for our spectral index, NDWI, as we did for precipitation and land surface temperatures: first, calculate the variable(s), then calculate the daily values, and finally summarize by *woreda*.

Section 5.1. Calculate NDWI

Here we will focus on NDWI, which we actively used in forecasting malaria. For examples on other indices, see Chap. F3.1.

The MODIS MCD43A4 product contains simplified band quality information, and it is recommended to use the additional quality information in the MCD43A2 product for your particular application. We will join these two products to apply our selected quality information. (Note that we do not have to worry about snow in our study area.) For more information on joining image collections, see Chap. F4.9.

```
// Section 5: Spectral index NDWI

// Section 5.1: Calculate NDWI

// Filter BRDF-Adjusted Reflectance by date.
var brdfReflectVars = brdfReflect
    .filterDate(brdfStartDate, reqEndDate.advance(1, 'day'))
    .filterBounds(amhara)
    .select([
            'Nadir_Reflectance_Band1', 'Nadir_Reflectance_Band2',
            'Nadir_Reflectance_Band3', 'Nadir_Reflectance_Band4',
            'Nadir_Reflectance_Band5', 'Nadir_Reflectance_Band6',
            'Nadir_Reflectance_Band7'
        ],
        ['red', 'nir', 'blue', 'green', 'swir1', 'swir2', 'swir3']);

// Filter BRDF QA by date.
var brdfReflectQa = brdfQa
    .filterDate(brdfStartDate, reqEndDate.advance(1, 'day'))
    .filterBounds(amhara)
    .select([
            'BRDF_Albedo_Band_Quality_Band1',
            'BRDF_Albedo_Band_Quality_Band2',
            'BRDF_Albedo_Band_Quality_Band3',
            'BRDF_Albedo_Band_Quality_Band4',
            'BRDF_Albedo_Band_Quality_Band5',
            'BRDF_Albedo_Band_Quality_Band6',
            'BRDF_Albedo_Band_Quality_Band7',
            'BRDF_Albedo_LandWaterType'
        ],
        ['qa1', 'qa2', 'qa3', 'qa4', 'qa5', 'qa6', 'qa7', 'water']);
```

```
// Join the 2 collections.
var idJoin = ee.Filter.equals({
    leftField: 'system:time_end',
    rightField: 'system:time_end'
});
// Define the join.
var innerJoin = ee.Join.inner('NBAR', 'QA');
// Apply the join.
var brdfJoined = innerJoin.apply(brdfReflectVars, brdfReflectQa,
    idJoin);

// Add QA bands to the NBAR collection.
function addQaBands(image) {
    var nbar = ee.Image(image.get('NBAR'));
    var qa = ee.Image(image.get('QA')).select(['qa2']);
    var water = ee.Image(image.get('QA')).select(['water']);
    return nbar.addBands([qa, water]);
}
var brdfMerged = ee.ImageCollection(brdfJoined.map(addQaBands));

// Function to mask out pixels based on QA and water/land flags.
function filterBrdf(image) {
    // Using QA info for the NIR band.
    var qaband = image.select(['qa2']);
    var wband = image.select(['water']);
    var qamask = qaband.lte(2).and(wband.eq(1));
    var nir_r = image.select('nir').multiply(0.0001).rename('nir_r');
    var swir2_r = image.select('swir2').multiply(0.0001).rename(
        'swir2_r');
    return image.addBands(nir_r)
        .addBands(swir2_r)
        .updateMask(qamask);
}
var brdfFilteredVars = brdfMerged.map(filterBrdf);

// Function to calculate spectral indices:
```

```
function calcBrdfIndices(image) {
    var curyear = ee.Date(image.get('system:time_start')).get('year');
    var curdoy = ee.Date(image.get('system:time_start'))
        .getRelative('day', 'year').add(1);
    var ndwi6 = image.normalizedDifference(['nir_r', 'swir2_r'])
        .rename('ndwi6');
    return image.addBands(ndwi6)
        .set('doy', curdoy)
        .set('year', curyear);
}
// Map function over image collection.
brdfFilteredVars = brdfFilteredVars.map(calcBrdfIndices);
```

Section 5.2. Calculate Daily NDWI

Similar to the other variables, we will calculate a daily value and filter to our user-requested dates, as data exists in that range.

```
// Section 5.2: Calculate daily NDWI

// Create list of dates for full time series.
var brdfRange = brdfFilteredVars.reduceColumns({
    reducer: ee.Reducer.max(),
    selectors: ['system:time_start']
});
var brdfEndDate = ee.Date(brdfRange.get('max'));
var brdfDays = brdfEndDate.difference(brdfStartDate, 'day');
var brdfDatesPrep = ee.List.sequence(0, brdfDays, 1);

function makeBrdfDates(n) {
    return brdfStartDate.advance(n, 'day');
}
var brdfDates = brdfDatesPrep.map(makeBrdfDates);

// List of dates that exist in BRDF data.
var brdfDatesExist = brdfFilteredVars
    .aggregate_array('system:time_start');
```

```
// Get daily brdf values.
function calcDailyBrdfExists(curdate) {
    curdate = ee.Date(curdate);
    var curyear = curdate.get('year');
    var curdoy = curdate.getRelative('day', 'year').add(1);
    var brdfTemp = brdfFilteredVars
        .filterDate(curdate, curdate.advance(1, 'day'));
    var outImg = brdfTemp.first();
    return outImg;
}
var dailyBrdfExtExists =
    ee.ImageCollection.fromImages(brdfDatesExist.map(
        calcDailyBrdfExists));

// Create empty results, to fill in dates when BRDF data does not
exist.
function calcDailyBrdfFiller(curdate) {
    curdate = ee.Date(curdate);
    var curyear = curdate.get('year');
    var curdoy = curdate.getRelative('day', 'year').add(1);
    var brdfTemp = brdfFilteredVars
        .filterDate(curdate, curdate.advance(1, 'day'));
    var brdfSize = brdfTemp.size();
    var outImg = ee.Image.constant(0).selfMask()
        .addBands(ee.Image.constant(0).selfMask())
        .addBands(ee.Image.constant(0).selfMask())
        .addBands(ee.Image.constant(0).selfMask())
        .addBands(ee.Image.constant(0).selfMask())
        .rename(['ndvi', 'evi', 'savi', 'ndwi5', 'ndwi6'])
        .set('doy', curdoy)
        .set('year', curyear)
        .set('system:time_start', curdate)
        .set('brdfSize', brdfSize);
    return outImg;
}
// Create filler for all dates.
```

```
var dailyBrdfExtendedFiller =
    ee.ImageCollection.fromImages(brdfDates.map(calcDailyBrdfFiller));
// But only used if and when size was 0.
var dailyBrdfExtFillFilt = dailyBrdfExtendedFiller
    .filter(ee.Filter.eq('brdfSize', 0));
// Merge the two collections.
var dailyBrdfExtended = dailyBrdfExtExists
    .merge(dailyBrdfExtFillFilt);

// Filter back to original user requested start date.
var dailyBrdf = dailyBrdfExtended
    .filterDate(reqStartDate, brdfEndDate.advance(1, 'day'));
```

***Section 5.3. Summarize Daily Spectral Indices by* Woreda**

Lastly in our NDWI section, we will use the mean to summarize the values for each of the *woredas* and prepare for export by flattening the dataset. The exports (of all variables) will be all done in Sect. 7.

```
// Section 5.3: Summarize daily spectral indices by woreda

// Filter spectral indices for zonal summaries.
var brdfSummary = dailyBrdf
    .filterDate(reqStartDate, reqEndDate.advance(1, 'day'));

// Function to calculate zonal statistics for spectral indices by
woreda:
function sumZonalBrdf(image) {
    // To get the doy and year, we convert the metadata to grids
    //  and then summarize.
    var image2 = image.addBands([
        image.metadata('doy').int(),
        image.metadata('year').int()
    ]);
    // Reduce by regions to get zonal means for each woreda.
    var output = image2.select(['doy', 'year', 'ndwi6'])
        .reduceRegions({
            collection: woredas,
```

```
            reducer: ee.Reducer.mean(),
            scale: 1000
        });
    return output;
}

// Map the zonal statistics function over the filtered spectral index
data.
var brdfWoreda = brdfSummary.map(sumZonalBrdf);
// Flatten the results for export.
var brdfFlat = brdfWoreda.flatten();
```

Code Checkpoint A16e. The book's repository contains a script that shows what your code should look like at this point.

Question 3. Here we are only calculating NDWI, which is calculated from the near infrared (NIR) and shortwave infrared 2 (SWIR2) bands. If we wanted to calculate a vegetation index like the Normalized Difference Vegetation Index (NDVI), which bands would we need to add? Where in Sects. 5.1 through 5.3 would we need to add or select the raw bands and/or our new calculated band? Note: Fully implementing this is one of the synthesis challenges, so this is a good head start!

Section 6. Map Display

Here we will take a look at our calculated variables but prior to zonal summary (Fig. A1.6.2). The full user interface restricts the date to display within the requested range, so be mindful in the code below which date you choose to view (we set our time range here in Sect. 2.1).

```
// Section 6: Map display of calculated environmental variables
var displayDate = ee.Date('2021-10-01');

var precipDisp = dailyPrecip
    .filterDate(displayDate, displayDate.advance(1, 'day'));
var brdfDisp = dailyBrdf
    .filterDate(displayDate, displayDate.advance(1, 'day'));
var LSTDisp = dailyLst
```

```
    .filterDate(displayDate, displayDate.advance(1, 'day'));

// Select the image (should be only one) from each collection.
var precipImage = precipDisp.first().select('totprec');
var LSTmImage = LSTDisp.first().select('LST_mean');
var ndwi6Image = brdfDisp.first().select('ndwi6');

// Palettes for environmental variable maps:
var palettePrecip = ['f7fbff', '08306b'];
var paletteLst = ['fff5f0', '67000d'];
var paletteSpectral = ['ffffe5', '004529'];

// Add layers to the map.
// Show precipitation by default,
//  others hidden until users picks them from layers drop down.
Map.addLayer({
    eeObject: precipImage,
    visParams: {
        min: 0,
        max: 20,
        palette: palettePrecip
    },
    name: 'Precipitation',
    shown: true,
    opacity: 0.75
});
Map.addLayer({
    eeObject: LSTmImage,
    visParams: {
        min: 0,
        max: 40,
        palette: paletteLst
    },
    name: 'LST Mean',
    shown: false,
    opacity: 0.75
});
```

```
Map.addLayer({
    eeObject: ndwi6Image,
    visParams: {
        min: 0,
        max: 1,
        palette: paletteSpectral
    },
    name: 'NDWI6',
    shown: false,
    opacity: 0.75
});
```

Code Checkpoint A16f. The book's repository contains a script that shows what your code should look like at this point.

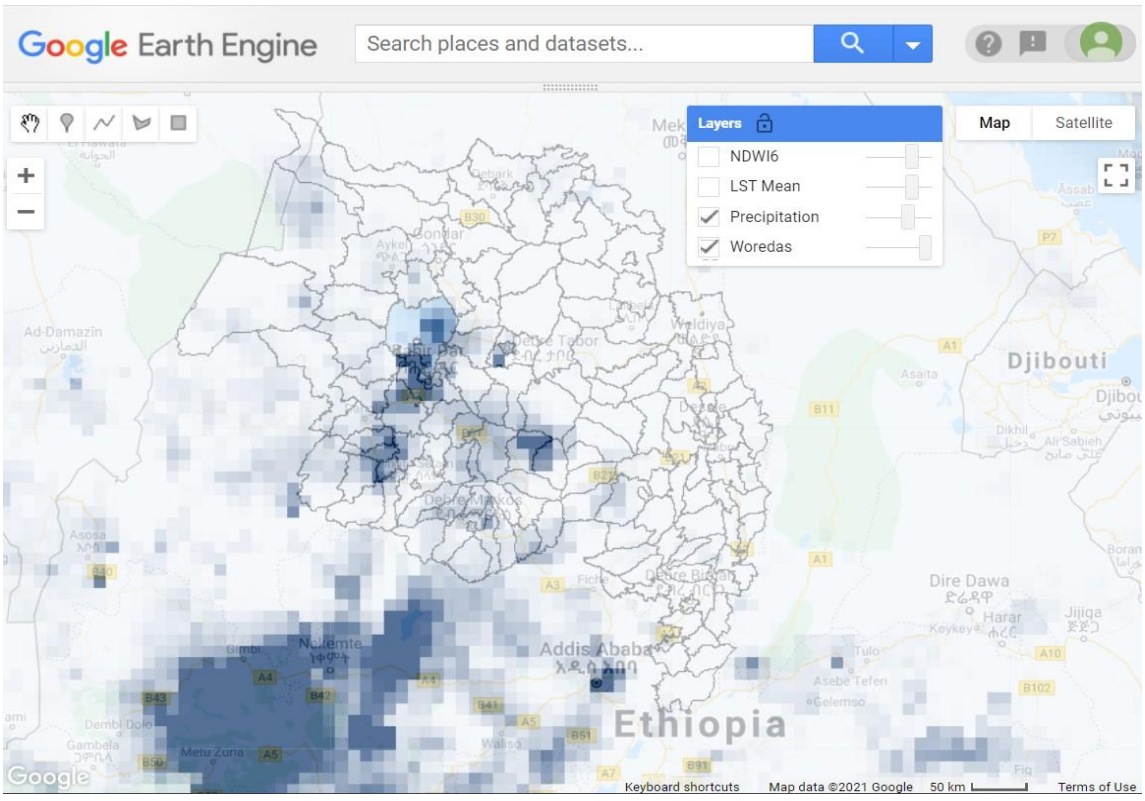


Fig. A1.6.2 Calculated total daily precipitation overlaid on *woreda* boundaries in the Amhara region of Ethiopia

Section 7. Exporting

Two important strengths of Google Earth Engine are the ability to gather and process the remotely sensed data all in the cloud, and to have the only download be a small text file ready to use in the forecasting software. Most of our partners on this project were experts in public health and did not have a remote sensing or programming background. We also had partners in areas of limited or unreliable internet connectivity. We needed something that could be easily usable by our users in these types of situations.

In this section, we will create small text CSV downloads for each of our three environmental factors prepared earlier. Each factor may have different data availability within the user's requested range, and these dates will be added to the file name to indicate the actual date range of the downloaded data (Fig. A1.6.3).

```
// Section 7: Exporting

// 7.1 Export naming
var reqStartDateText = reqStartDate.format('yyyy-MM-dd').getInfo();

// Precipitation
var precipPrefix = 'Export_Precip_Data';
var precipLastDate = ee.Date(reqEndDate.millis()
    .min(precipEndDate.millis()));
var precipSummaryEndDate = precipLastDate
    .format('yyyy-MM-dd').getInfo();
var precipFilename = precipPrefix
    .concat('_', reqStartDateText,
        '_', precipSummaryEndDate);
// LST
var LSTPrefix = 'Export_LST_Data';
var LSTLastDate = ee.Date(reqEndDate.millis()
    .min(LSTEndDate.millis()));
var LSTSummaryEndDate = LSTLastDate
    .format('yyyy-MM-dd').getInfo();
var LSTFilename = LSTPrefix
    .concat('_', reqStartDateText,
```

```
        '_', LSTSummaryEndDate);
// BRDF
var brdfPrefix = 'Export_Spectral_Data';
var brdfLastDate = ee.Date(reqEndDate.millis()
    .min(brdfEndDate.millis()));
var brdfSummaryEndDate = brdfLastDate
    .format('yyyy-MM-dd').getInfo();
var brdfFilename = brdfPrefix
    .concat('_', reqStartDateText,
        '_', brdfSummaryEndDate);

// 7.2 Export flattened tables to Google Drive
// Need to click 'RUN in the Tasks tab to configure and start each
export.
Export.table.toDrive({
    collection: precipFlat,
    description: precipFilename,
    selectors: ['wid', 'woreda', 'doy', 'year', 'totprec']
});
Export.table.toDrive({
    collection: LSTFlat,
    description: LSTFilename,
    selectors: ['wid', 'woreda', 'doy', 'year',
        'LST_day', 'LST_night', 'LST_mean'
    ]
});
Export.table.toDrive({
    collection: brdfFlat,
    description: brdfFilename,
    selectors: ['wid', 'woreda', 'doy', 'year', 'ndwi6']
});
```

Code Checkpoint A16g. The book's repository contains a script that shows what your code should look like at this point.

In the Earth Engine **Tasks** tab, click **Run** to configure and start each export to Google Drive.

	A	B	C	D	E
1	wid	woreda	doy	year	totprec
2	74	Kewet	274	2021	0.164364
3	133	Jilie Timuga	274	2021	0.390325
4	70	Efratana Gidim	274	2021	0.311525
5	134	Artuma Fursi	274	2021	0.00951
6	131	Dewa Chefa	274	2021	0.002063
7	135	Dewa Harewa	274	2021	0.294391
8	132	Bati	274	2021	0.12475
9	138	Aregoba Sp. Wo.	274	2021	0.221236
10	49	Kalu	274	2021	0.031743

Export_Precip_Data_2021-10-01_2

	A	B	C	D	E	F	G
1	wid	woreda	doy	year	lst_day	lst_night	lst_mean
2	74	Kewet	274	2021	28.68206	15.83887	20.81999
3	133	Jilie Timuga	274	2021	32.34133	18.60174	25.96135
4	70	Efratana Gidim	274	2021	23.99423	11.27908	16.5801
5	134	Artuma Fursi	274	2021	32.20547	17.27566	24.15875
6	131	Dewa Chefa	274	2021	27.4921	14.66733	20.32833
7	135	Dewa Harewa	274	2021	32.06023	15.22292	21.1516
8	132	Bati	274	2021	30.5469	18.90438	22.68315
9	138	Aregoba Sp. Wo.	274	2021	30.11613	16.51357	23.15889
10	49	Kalu	274	2021	26.07411	13.13784	19.56946

Export_LST_Data_2021-10-01_2021

	A	B	C	D	E
1	wid	woreda	doy	year	ndwi6
2	74	Kewet	274	2021	0.215263
3	133	Jilie Timuga	274	2021	0.228306
4	70	Efratana Gidim	274	2021	0.248442
5	134	Artuma Fursi	274	2021	0.240127
6	131	Dewa Chefa	274	2021	0.267753
7	135	Dewa Harewa	274	2021	0.221924
8	132	Bati	274	2021	0.202291
9	138	Aregoba Sp. Wo.	274	2021	0.226092
10	49	Kalu	274	2021	0.27609

Export_Spectral_Data_2021-10-01

Fig. A1.6.3 Examples of the three CSV files returned from the script

Section 8. Importing and Viewing External Analyses Results

As mentioned at the start of the chapter, the environmental data obtained from Earth Engine can be used for infectious disease modeling and forecasting. The above Earth Engine code was written in support of EPIDEMIA, a software system based in the R

language and computing environment for forecasting malaria, and was actively used in certain study pilot *woredas* in the Amhara region of Ethiopia. The R system consists of an R package—*epidemiar*—for generic functions and a companion R project for handling all the location-specific data and settings.

One of the main outputs of EPIDEMIA is the forecasted incidence of malaria in each *woreda* by week from one to eight (or more) weeks in advance. Using our publicly available demo project that uses synthetic data (not for use in epidemiological study), we created forecasts for week 32 of 2018 made eight weeks prior ("knowing" data up to week 24), and also added the observed incidence for comparison. (Note: dates and weeks follow International Organization for Standardization [ISO] standard 8601). These new data can be re-uploaded to Earth Engine for further analyses or exploration.

Starting a new script, you can use the Sect. 8 code that follows to visualize the pre-generated demo 2018W32 results (Fig. A1.6.4).

```
// Section 8: Viewing external analyses results

// This is using *synthetic* malaria data.
// For demonstration only, not to be used for epidemiological
purposes.
var epidemiaResults = ee.FeatureCollection(
    'projects/gee-book/assets/A1-6/amhara_pilot_synthetic_2018W32'
);
// Filter to only keep pilot woredas with forecasted values.
var pilot = epidemiaResults
    .filter(ee.Filter.neq('inc_n_fc', null));
var nonpilot = epidemiaResults
    .filter(ee.Filter.eq('inc_n_fc', null));

Map.setCenter(38, 11.5, 7);

// Paint the pilot woredas with different colors for forecasted*
incidence
// fc_n_inc here is the forecasted incidence (cut into factors)
// made on (historical) 2018W24 (i.e. 8 weeks in advance).
// * based on synthetic data for demonstration only.
// Incidence per 1000
```

```
// 1 : [0 - 0.25)
// 2 : [0.25 - 0.5)
// 3 : [0.5 - 0.75)
// 4 : [0.75 - 1)
// 5 : > 1

var empty = ee.Image().byte();
var fill_fc = empty.paint({
    featureCollection: pilot,
    color: 'inc_n_fc',
});
var palette = ['fee5d9', 'fcae91', 'fb6a4a', 'de2d26', 'a50f15'];
Map.addLayer(
    fill_fc, {
        palette: palette,
        min: 1,
        max: 5
    },
    'Forecasted Incidence'
);

// Paint the woredas with different colors for the observed*
incidence.
// * based on synthetic data for demonstration only
var fill_obs = empty.paint({
    featureCollection: pilot,
    color: 'inc_n_obs',
});
var palette = ['fee5d9', 'fcae91', 'fb6a4a', 'de2d26', 'a50f15'];
// Layer is off by default, users change between the two in the map
viewer.
Map.addLayer(
    fill_obs, {
        palette: palette,
        min: 1,
        max: 5
    },
```

```
    'Observed Incidence',
    false
);

// Add gray fill for nonpilot woredas (not included in study).
var fill_na = empty.paint({
    featureCollection: nonpilot
});
Map.addLayer(
    fill_na, {
        palette: 'a1a9a8'
    },
    'Non-study woredas'
);

// Draw borders for ALL Amhara region woredas.
var outline = empty.paint({
    featureCollection: epidemiaResults,
    color: 1,
    width: 1
});
// Add woreda boundaries to map.
Map.addLayer(
    outline, {
        palette: '000000'
    },
    'Woredas'
);
```

Code Checkpoint A16h. The book's repository contains a script that shows what your code should look like at this point.

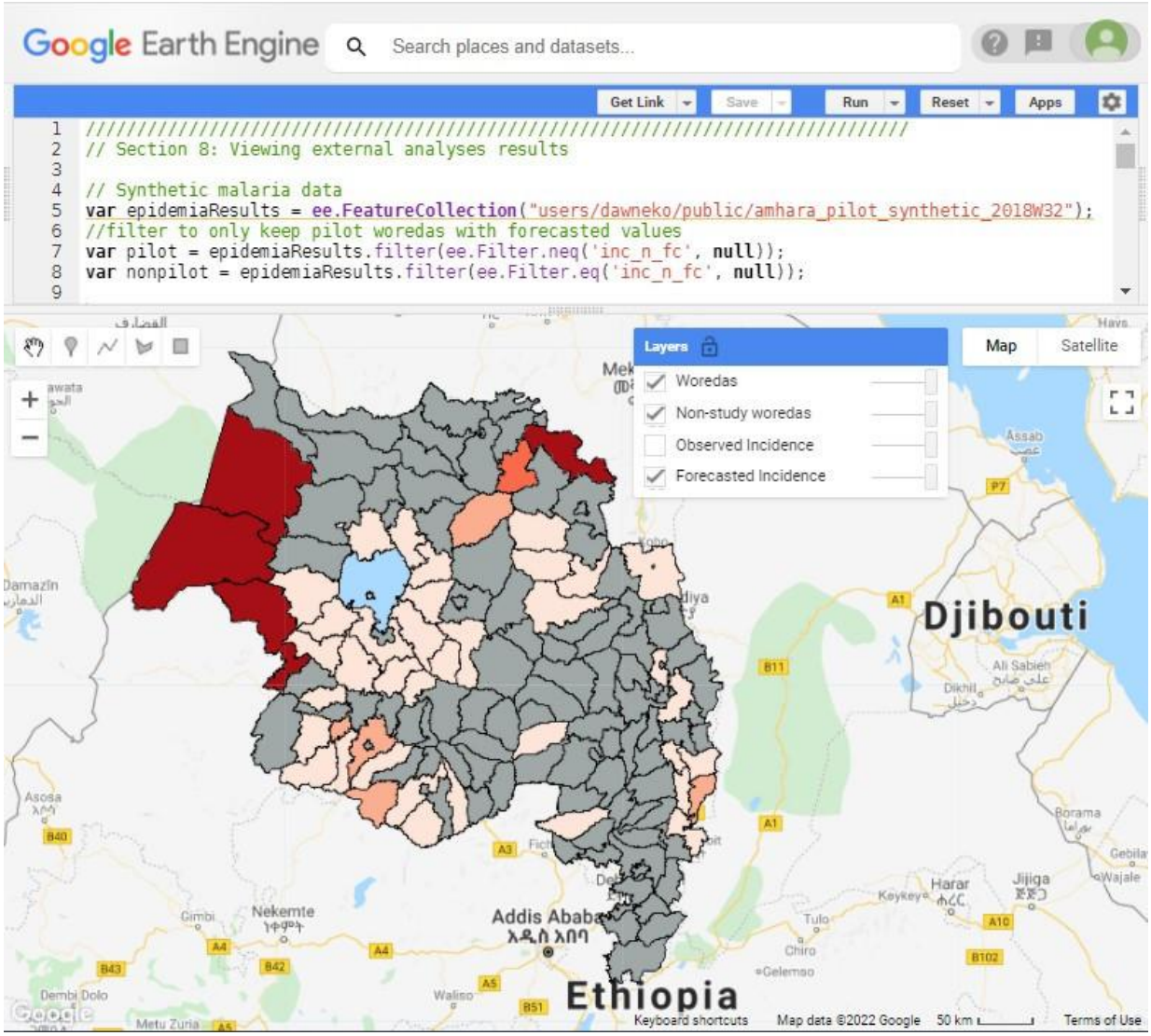


Fig. A1.6.4 Visualization of forecasted malaria incidence for week 32 of 2018 made during week 24 (an eight-week lead time). Malaria data is synthetic, for demonstration purposes only. The incidence has been categorized into five categories (from lighter to dark red): 0–0.25, 0.25–0.5, 0.5–0.75, 0.75–1, and greater than 1. Only *woredas* in the pilot project have values; the rest of the Amhara region is marked in gray fill. Another layer available to view is the observed (synthetic) incidence rate for 2018W32.

Synthesis

Assignment 1. Calculate other spectral indices: In this chapter, we only calculate and export the NDWI from the spectral data. Calculate another index, such as a vegetation index like NDVI, Soil Adjusted Vegetation Index (SAVI), or Enhanced Vegetation Index (EVI) to the calculations. Think about what bands you will need, how to calculate the index, and how to propagate the band through all the remaining processing steps (including exporting).

Assignment 2. Change location: In this chapter we obtained data for *woredas* in the Amhara region of Ethiopia. Upload or import a new shapefile of different locations and acquire environmental data for there instead. Remember that you will need to adjust any references to asset-specific fields (as we did here for "*woreda*"). See Chap. F5.0 for help with uploading assets, if needed.

Conclusion

In this chapter, we saw how Earth Engine can be used to acquire environmental data to support external analyses, such as forecasting of malaria, a vector-borne disease. An understanding of the biology of the vector (e.g., mosquito, tick), and how different environmental conditions can affect the disease system and transmission risk, will help identify environmental variables to investigate for use in mathematical modeling.

In this chapter we obtained data from three different satellite-based datasets: rainfall from IMERG/GPM, land surface temperature 8-day composite values from MODIS, and the calculation of spectral indices from MODIS bands. We saw how to perform zonal summaries to our location of interest, and download CSV files that are suitable for import into other programs for additional analyses.

This chapter shows the value of cloud computation and generating small downloads for use by professionals who may not have expertise in remote sensing or the computing resources that would otherwise be needed. Finally, we saw that the results of intermediate processing and work outside of Earth Engine can be re-imported for additional analyses within Earth Engine.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Ford TE, Colwell RR, Rose JB, et al (2009) Using satellite images of environmental changes to predict infectious disease outbreaks. Emerg Infect Dis 15:1341–1346. https://doi.org/10.3201/eid/1509.081334

Franklinos LHV, Jones KE, Redding DW, Abubakar I (2019) The effect of global change on mosquito-borne disease. Lancet Infect Dis 19:e302–e312. https://doi.org/10.1016/S1473-3099(19)30161-6

Jones KE, Patel NG, Levy MA, et al (2008) Global trends in emerging infectious diseases. Nature 451:990–993. https://doi.org/10.1038/nature06536

Mackenzie JS, Jeggo M (2019) The one health approach–why is it so important? Trop. Med. Infect. Dis. 4:88. https://doi.org/10.3390/tropicalmed4020088

Wimberly MC, de Beurs KM, Loboda T V., Pan WK (2021) Satellite observations and malaria: New opportunities for research and applications. Trends Parasitol 37:525–537. https://doi.org/10.1016/j.pt.2021.03.003

Wimberly MC, Nekorchuk DM, Kankanala RR (2022) Cloud-based applications for accessing satellite Earth observations to support malaria early warning. Sci Data 9:1–11. https://doi.org/10.1038/s41597-022-01337-y

World Health Organization (2018) Malaria surveillance, monitoring and evaluation: a reference manual. World Health Organization

World Health Organization (2020) World Malaria Report 2020: 20 years of global progress and challenges. World Health Organization

Chapter A1.7: Humanitarian Applications

Authors

Jamon Van Den Hoek and Hannah K. Friedrich

Overview

The global refugee population has never been as large as it is today, with at least 26 million refugees living in more than 100 countries. Refugees are international migrants who have been forcibly displaced from their home countries due to violence or persecution and who cross an international border and settle elsewhere, most often in a neighboring country. Remote sensing can help refugee leaders, humanitarian agencies, and refugee-hosting countries gain new insights into refugee settlement, population, and land cover change dynamics (Maystadt et al. 2020, Van Den Hoek et al. 2021). In this chapter, we will examine the value of using satellite imagery and satellite-derived data to map a refugee settlement in Uganda, estimate its population, and gauge land cover changes in and around the settlement.

Learning Outcomes

- Using a range of techniques—maps, videos, and charts—to visualize and measure land cover changes before and after the establishment of a refugee settlement.
- Understanding the considerations and limitations involved in automated detection of refugee settlement boundaries using unsupervised classification.
- Becoming familiar with satellite-derived human settlement and population datasets and their application in a refugee settlement context.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Perform basic image analysis: select bands, compute indices, create masks, classify images (Part F2).
- Create a graph using `ui.Chart` (Chap. F1.3).
- Use `normalizedDifference` to calculate vegetation indices (Chap. F2.0).

- Perform pixel-based supervised or unsupervised classification (Chap. F2.1).
- Use `ee.Reducer` functions to summarize pixels over an area (Chap. F3.0, Chap. F3.1).
- Perform image morphological operations (Chap. F3.2).
- Write a function and `map` it over an `ImageCollection` (Chap. F4.0).
- Use `reduceRegions` to summarize an image with zonal statistics in irregular shapes (Chap. F5.0, Chap. F5.2).
- Converting from a vector to a raster representation with `reduceToImage` (Chap. F5.1).
- Write a function and `map` it over a `FeatureCollection` (Chap. F5.1, Chap. F5.2).

Introduction to Theory

In a humanitarian context, remote sensing data and analysis have become essential tools for monitoring refugee settlement dynamics both immediately after refugee arrival and over the long term. Nonetheless, there remain important challenges to characterizing refugee settlement conditions. First, dwellings, roadways, and agricultural plots tend to be small in size within refugee settlements and generally difficult to detect without the use of very high resolution satellite imagery. Second, dwellings and other structures within refugee settlements may be diffusely distributed and intermixed with vegetation and bare earth. Third, data on settlement features, boundaries, and refugee populations are often out of date or otherwise inappropriate for detailed geospatial analysis. In this chapter, we will examine these challenges and do our best to document refugee settlement dynamics through analysis of multi-date Landsat imagery.

Practicum

The study area for this chapter is Pagirinya Refugee Settlement in northwestern Uganda (Fig. A1.7.1a). As of 2020, Uganda was home to 1.4 million refugees, the fourth-largest refugee population in the world and the largest in Africa (UNHCR 2021). Refugees living in Uganda primarily fled violence in South Sudan and the Democratic Republic of the Congo, and most live in rural refugee settlements. Pagirinya in particular is home to 36,000 South Sudanese refugees and was established in mid-2016.

In this practicum, we will visualize and document the land cover changes that have taken place in Pagirinya (Fig. A1.7.1b and c), use satellite data to estimate the settlement's boundary and compare it to the official boundary laid out by the United Nations High Commissioner for Refugees (UNHCR), and use satellite-derived demographic products to estimate the refugee population within Pagirinya.

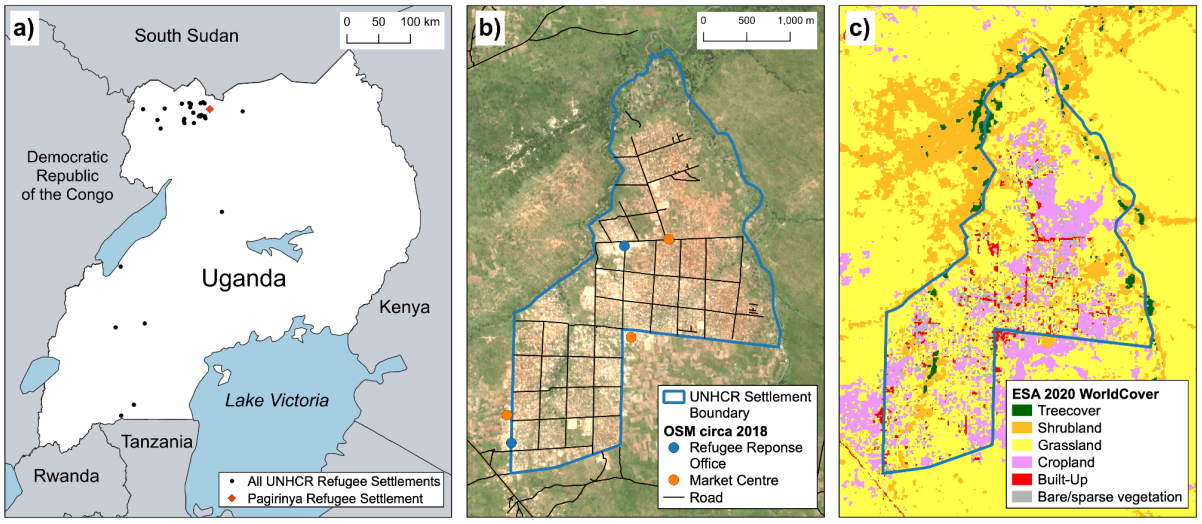


Fig. A1.7.1 Maps of a) UNHCR refugee settlements in Uganda, b) OpenStreetMap features, roadways, and the UNHCR settlement boundary for Pagirinya, and c) European Space Agency 2020 WorldCover land cover at Pagirinya

Section 1. Seeing Refugee Settlements from Above

In preparation for the arrival of refugees, humanitarian actors and refugee settlement planners are often interested in analyzing local land cover conditions before a refugee settlement is established. The goal of this first section is to use Landsat satellite imagery to characterize initial land cover conditions and land cover changes at Pagirinya Refugee Settlement in the years before and following the settlement's establishment in 2016.

Let's begin by adding the refugee settlement's boundary to the **Map** by loading the `FeatureCollection` of refugee settlement boundaries in Uganda and filtering to Pagirinya Refugee Settlement. We'll also initialize the **Map** to center on Pagirinya and default to showing the satellite basemap for visual reference.

```
Map.setOptions('SATELLITE');

// Load UNHCR settlement boundary for Pagirinya Refugee Settlement.
var pagirinya = ee.Feature(ee.FeatureCollection(
    'projects/gee-book/assets/A1-7/pagirinya_settlement_boundary'
).first());
```

```
Map.addLayer(pagirinya, {}, 'Pagirinya Refugee Settlement');
Map.centerObject(pagirinya, 14);
```

Next, let's create annual Landsat composites using the Landsat 8 surface reflectance `ImageCollection`. We'll spatially filter the `ImageCollection` to a buffered settlement boundary and temporally filter to 2015–2020, which includes the full year before the settlement was established and the four years that followed. We'll also apply a cloud filter of less than or equal to 40% to help ensure that our annual composites are cloud free.

For better legibility, we'll rename the Landsat bands and add three new spectral index bands to each image in the `ImageCollection` using the `addIndices` function, which calculates the Normalized Difference Vegetation Index (NDVI), Normalized Difference Building Index (NDBI), and Normalized Burn Ratio (NBR) using `normalizedDifference`. Each of these metrics offers a different approach to characterizing land cover conditions and change over time. NDVI is commonly used for monitoring vegetation health; NDBI helps to characterize impervious and built-up surfaces; and NBR helps to identify land that has been cleared with fire, a common practice in our study region. Note that other spectral metrics or remote sensing platforms may be better suited for identifying refugee settlements in other regions.

```
// Create buffered settlement boundary geometry.
// 500 meter buffer size is arbitrary but large enough
// to capture area outside of the study settlement.
var bufferSize = 500; // (in meters)

// Buffer and convert to Geometry for spatial filtering and clipping.
var bufferedBounds = pagirinya.buffer(bufferSize)
    .bounds().geometry();

function addIndices(img) {
    var ndvi = img.normalizedDifference(['nir', 'red'])
        .rename('NDVI'); // NDVI = (nir-red)/(nir+red)
    var ndbi = img.normalizedDifference(['swir1', 'nir'])
        .rename(['NDBI']); // NDBI = (swir1-nir)/(swir1+nir)
    var nbr = img.normalizedDifference(['nir', 'swir2'])
```

```
        .rename(['NBR']); // NBR = (nir-swir2)/(nir+swir2)
    var imgIndices = img.addBands(ndvi).addBands(ndbi).addBands(nbr);
    return imgIndices;
}

// Create L8 SR Collection 2 band names and new names.
var landsat8BandNames = ['SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'SR_B6',
    'SR_B7'
];
var landsat8BandRename = ['blue', 'green', 'red', 'nir', 'swir1',
'swir2'];

// Create image collection.
var landsat8Sr = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2');
var ic = ee.ImageCollection(landsat8Sr.filterDate('2015-01-01',
        '2020-12-31')
    .filterBounds(bufferedBounds)
    .filter(ee.Filter.lt('CLOUD_COVER', 40))
    .select(landsat8BandNames, landsat8BandRename)
    .map(addIndices));
```

To build annual composites from before and after Pagirinya's establishment in 2016, let's create two temporal subsets of the `ImageCollection`—one from 2015 and one from 2017—and use the `median` function to composite images for each time frame (Fig. A1.7.2). We'll also clip our image collections to the buffered region around Pagirinya. To visualize the NDVI composites, we'll use true-color and false-color visualizations and color palettes, which should help us identify and interpret features within and surrounding the settlement boundary.

```
// Make annual pre- and post-establishment composites.
var preMedian = ic.filterDate('2015-01-01', '2015-12-31').median()
    .clip(bufferedBounds);
var postMedian = ic.filterDate('2017-01-01', '2017-12-31').median()
    .clip(bufferedBounds);

// Import visualization palettes
https://github.com/gee-community/ee-palettes.
```

```
var palettes = require('users/gena/packages:palettes');
var greenPalette = palettes.colorbrewer.Greens[9];
var prGreenPalette = palettes.colorbrewer.PRGn[9];

// Set-up "true color" visualization parameters.
var TCImageVisParam = {
    bands: ['red', 'green', 'blue'],
    gamma: 1,
    max: 13600,
    min: 8400,
    opacity: 1
};

// Set-up "false color" visualization parameters.
var FCImageVisParam = {
    bands: ['nir', 'red', 'green'],
    gamma: 1,
    min: 9000,
    max: 20000,
    opacity: 1
};

// Display true-color composites.
Map.addLayer(preMedian, TCImageVisParam,
    'Pre-Establishment Median TC');
Map.addLayer(postMedian, TCImageVisParam,
    'Post-Establishment Median TC');

// Display false-color composites.
Map.addLayer(preMedian, FCImageVisParam,
    'Pre-Establishment Median FC');
Map.addLayer(postMedian, FCImageVisParam,
    'Post-Establishment Median FC');

// Display median NDVI composites.
Map.addLayer(preMedian, {
    min: 0,
```

```
    max: 0.7,
    bands: ['NDVI'],
    palette: greenPalette
}, 'Pre-Establishment Median NDVI');
Map.addLayer(postMedian, {
    min: 0,
    max: 0.7,
    bands: ['NDVI'],
    palette: greenPalette
}, 'Post-Establishment Median NDVI');

// Create an empty byte Image into which we'll paint the settlement
boundary.
var empty = ee.Image().byte();

// Convert settlement boundary's geometry to an Image for overlay.
var pagirinyaOutline = empty.paint({
    featureCollection: pagirinya,
    color: 1,
    width: 2
});

// Display Pagirinya boundary in blue.
Map.addLayer(pagirinyaOutline,
    {
        palette: '0000FF'
    },
    'Pagirinya Refugee Settlement boundary');
```

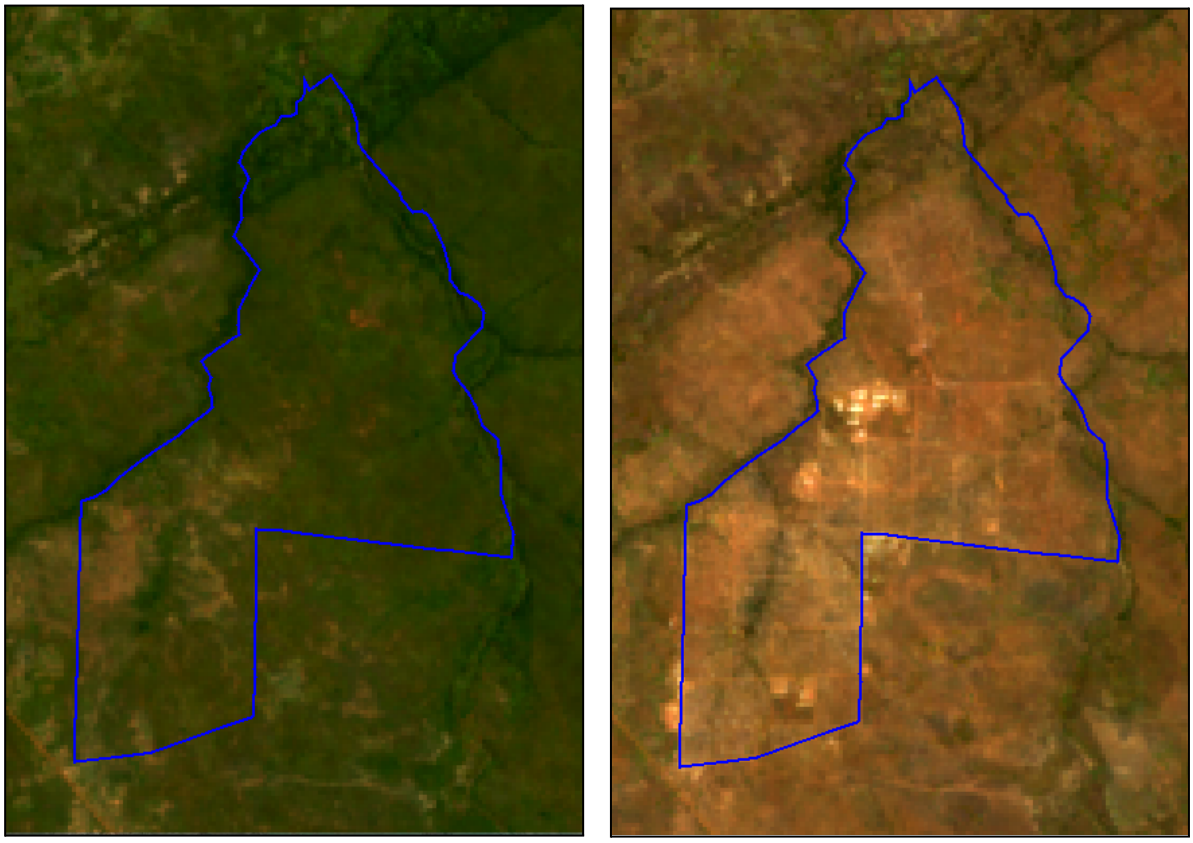


Fig. A1.7.2 Pre-establishment (left) and post-establishment (right) true-color composites with Pagirinya Refugee Settlement boundary overlaid in blue

Now that we have pre- and post-establishment composites to support a visual qualitative assessment, let's make a complementary quantitative assessment by measuring pre- and post-establishment differences in median NDVI and plotting the distribution of NDVI from both periods.

```
// Compare pre- and post-establishment differences in NDVI.
var diffMedian = postMedian.subtract(preMedian);
Map.addLayer(diffMedian,
    {
        min: -0.1,
        max: 0.1,
        bands: ['NDVI'],
        palette: prGreenPalette
```

```
    },
    'Difference Median NDVI');

// Chart the NDVI distributions for pre- and post-establishment.
var combinedNDVI = preMedian.select(['NDVI'], ['pre-NDVI'])
    .addBands(postMedian.select(['NDVI'], ['post-NDVI']));

var prePostNDVIFrequencyChart =
    ui.Chart.image.histogram({
        image: combinedNDVI,
        region: bufferedBounds,
        scale: 30
    }).setSeriesNames(['Pre-Establishment', 'Post-Establishment'])
    .setOptions({
        title: 'NDVI Frequency Histogram',
        hAxis: {
            title: 'NDVI',
            titleTextStyle: {
                italic: false,
                bold: true
            },
        },
        vAxis:
        {
            title: 'Count',
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        colors: ['cf513e', '1d6b99']
    });
print(prePostNDVIFrequencyChart);
```

In addition to the pre- and post-establishment annual composites, let's create an annotated video time series of the full 2015–2020 Landsat 8 surface reflectance `ImageCollection`. We'll be able to use this video to view changes at our study refugee settlement image by image.

```
// Import package to support text annotation.
var text = require('users/gena/packages:text');
var rgbVisParam = {
    bands: ['red', 'green', 'blue'],
    gamma: 1,
    max: 12011,
    min: 8114,
    opacity: 1
};

// Define arguments for animation function parameters.
var videoArgs = {
    region: bufferedBounds,
    framesPerSecond: 3,
    scale: 10
};

var annotations = [{
    position: 'left',
    offset: '5%',
    margin: '5%',
    property: 'label',
    scale: 30
}];

function addText(image) {
    var date = ee.String(ee.Date(image.get('system:time_start'))
        .format(' YYYY-MM-dd'));
    // Set a property called label for each image.
    var image = image.clip(bufferedBounds).visualize(rgbVisParam)
        .set({
            'label': date
        });
    // Create a new image with the label overlaid using gena's
package.
    var annotated = text.annotateImage(image, {}, bufferedBounds,
```

```
        annotations);
    return annotated;
}

// Add timestamp annotation to all images in video.
var tempCol = ic.map(addText);

// Click the URL to watch the time series video.
print('L8 Time Series Video', tempCol.getVideoThumbURL(videoArgs));
```

Code Checkpoint A17a. The book's repository contains a script that shows what your code should look like at this point.

Question 1. How would you describe the land cover type in the area in 2015, before the establishment of the refugee settlement? Is the land cover consistent within the settlement's boundary in the pre-establishment period? Does the settlement boundary conform to land cover type or condition in any meaningful way?

Question 2. What features (dwellings, roadways, agricultural plots, etc.) present the greatest visual difference between the pre- and post-establishment periods? Comparing the visual differences in true color, false color, and NDVI with the satellite image basemap may be helpful here.

Question 3. Which of the annual composite visualizations (true color, false color, or NDVI) do you prefer for distinguishing the refugee settlement in the post-establishment period, and why?

Question 4. How do the range and mode of NDVI values change from pre- to post-establishment? How might the changes in NDVI distribution correlate to overall changes in land cover type in the post-establishment period?

Question 5. Beyond the rapid establishment of the settlement's dwellings and roads, what changes do you observe in the time-series video? Do these changes occur within or outside the settlement boundary? What kinds of changes do you see in the imagery from 2019 or 2020, well after the settlement was established in 2016?

Section 2. Mapping Features Within the Refugee Settlement

In Sect. 1, we used Landsat data to gauge changes in land cover conditions and types, but we can also draw upon data products derived from satellite imagery. For instance, satellite-derived building footprints, which represent geometries of individual structures and dwellings, are often used to estimate human populations and population density in humanitarian contexts and to support the planning and delivery of food and other kinds of aid. In this section, we'll identify different features within Pagirinya, which we'll use to create a satellite image-based settlement boundary map in Sect. 3.

Let's add to our script from Sect. 1 by first loading the Open Buildings V1 Polygons dataset from the Earth Engine Data Catalog. This dataset includes satellite-derived building footprints based on very high resolution (0.5 m) satellite imagery, and each footprint has a confidence score. Let's visualize building footprints with a confidence score above 75% as orange and building footprints with a 75% or lower confidence score as purple.

```
// Visualize Open Buildings dataset.
var footprints = ee.FeatureCollection(
    'GOOGLE/Research/open-buildings/v1/polygons');
var footprintsHigh = footprints.filter('confidence > 0.75');
var footprintsLow = footprints.filter('confidence <= 0.75');

Map.addLayer(footprintsHigh, {
    color: 'FFA500'
}, 'Buildings high confidence');
Map.addLayer(footprintsLow, {
    color: '800080'
}, 'Buildings low confidence');
```

With a map of building footprints in place, let's turn to examining other features of interest that we identified in Sect. 1. Let's load a `FeatureCollection` of sample locations of infrastructure, forest, and agriculture visible on the satellite basemap as well as a sample of building footprint locations. Note in the `print` output that each feature has a `value`, which represents the feature type. Let's write a function to use this `value` property to automatically assign a unique color to each feature as part of a style (Fig. A1.7.3).

```
// Load land cover samples.
var lcPts = ee.FeatureCollection(
    'projects/gee-book/assets/A1-7/lcPts');
print('lcPts', lcPts);

// Create a function to set Feature properties based on value.
var setColor = function(f) {
    var value = f.get('class');
    var mapDisplayColors = ee.List(['#13a1ed', '#7d02bf',
        '#f0940a', '#d60909'
    ]);
    // Use the class as an index to lookup the corresponding display
color.
    return f.set({
        style: {
            color: mapDisplayColors.get(value)
        }
    });
};

// Apply the function and view the results.
var styled = lcPts.map(setColor);
Map.addLayer(styled.style({
    styleProperty: 'style'
}), {}, 'Land cover samples');
```

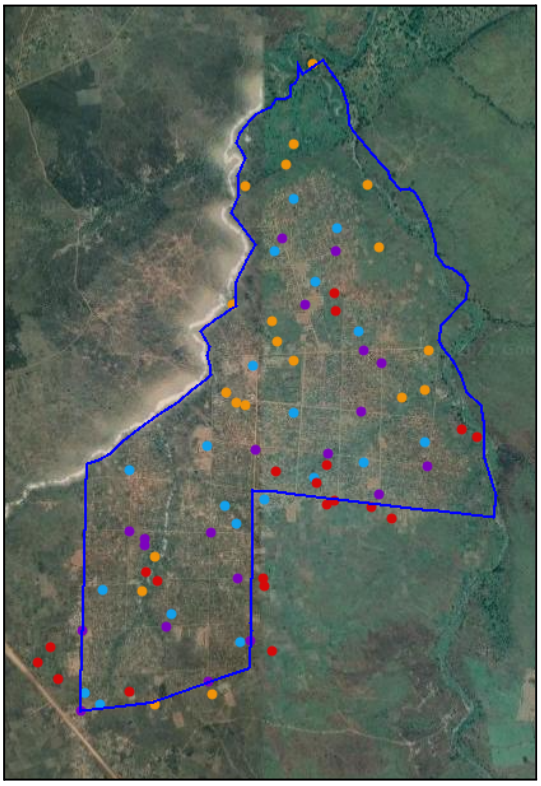


Fig. A1.7.3 Feature samples across Pagirinya Refugee Settlement (boundary shown in blue)

Since we want to use these sample land cover locations to help delineate the refugee settlement boundary, these different land cover types should be spectrally distinguishable from each other. To see how the spectral values vary among different features, let's create spectral signature plots for the post-establishment period. We first need to add the land cover class to the post-establishment composites that we made in Sect. 1 so that the class and spectral value information can be referenced together in our spectral signature plots. To do that, let's use `reduceToImage` to convert our `lcPts FeatureCollection` to an image, `lcBand`, and then add that image to the post-establishment composite.

```
// Convert land cover sample FeatureCollection to an Image.
var lcBand = lcPts.reduceToImage({
    properties: ['class'],
    reducer: ee.Reducer.first()
```

```
}).rename('class');

// Add lcBand to the post-establishment composite.
postMedian = postMedian.addBands(lcBand);
```

Now we have a `postMedian` image that we can sample at specific sample locations and identify not only the spectral values but also the class type. Let's plot the spectral values by class type. Note that since band names are sorted alphabetically on the x-axis, `nir` values are plotted in between `green` and `red` and are therefore out of order with respect to band wavelengths.

```
// Define bands that will be visualized in chart.
var chartBands = ['blue', 'green', 'red', 'nir', 'swir1', 'swir2',
    'class'
];

print(postMedian, 'postMedian');

// Plot median band value for each land cover type.
var postBandsChart = ui.Chart.image
    .byClass({
        image: postMedian.select(chartBands),
        classBand: 'class',
        region: lcPts,
        reducer: ee.Reducer.median(),
        scale: 30,
        classLabels: ['Settlement', 'Road', 'Forest',
            'Agriculture'
        ],
        xLabels: chartBands
    })
    .setChartType('ScatterChart')
    .setOptions({
        title: 'Band Values',
        hAxis: {
            title: 'Band Name',
            titleTextStyle: {
```

```
                italic: false,
                bold: true
            },
        },
        vAxis: {
            title: 'Reflectance (x1e4)',
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        colors: ['#13a1ed', '#7d02bf', '#f0940a', '#d60909'],
        pointSize: 0,
        lineSize: 5,
        curveType: 'function'
    });
print(postBandsChart);
```

Remember that we also calculated NDVI, NDBI, and NBR spectral indices in Sect. 1. Since these bands range from −1 to 1, we have to plot their values separately from the Landsat band spectral signature plots above, which use scaled reflectance values.

```
// Define spectral indices that will be visualized in the chart.
var indexBands = ['NDVI', 'NDBI', 'NBR', 'class'];

// Plot median index value for each land cover type.
var postIndicesChart = ui.Chart.image
    .byClass({
        image: postMedian.select(indexBands),
        classBand: 'class',
        region: lcPts,
        reducer: ee.Reducer.median(),
        scale: 30,
        classLabels: ['Settlement', 'Road', 'Forest',
            'Agriculture'
        ],
        xLabels: indexBands
```

```
    })
    .setChartType('ScatterChart')
    .setOptions({
        title: 'Index Values',
        hAxis: {
            title: 'Index Name',
            titleTextStyle: {
                italic: false,
                bold: true
            },
            //viewWindow: {min: wavelengths[0], max: wavelengths[2]}
            scaleType: 'string'
        },
        vAxis: {
            title: 'Value',
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        colors: ['#13a1ed', '#7d02bf', '#f0940a', '#d60909'],
        pointSize: 5
    });
print(postIndicesChart);

// Create an empty image into which to paint the features, cast to
byte.
var empty = ee.Image().byte();

// Paint all the polygon edges with the same number and width,
display.
var pagirinyaOutline = empty.paint({
    featureCollection: pagirinya,
    color: 1,
    width: 2
});
```

```
// Map outline of Pagirinya in blue.
Map.addLayer(pagirinyaOutline,
    {
        palette: '0000FF'
    },
    'Pagirinya Refugee Settlement boundary');
```

Code Checkpoint A17b. The book's repository contains a script that shows what your code should look like at this point.

Question 6. How would you describe the coverage of the footprints within the settlement? Are there sections of the settlement visible in the basemap or the post-establishment composite that are missing footprints?

Question 7. How do NDVI, NDBI, and NBR change from the pre- to post-establishment period at building footprint locations?

Question 8. Are the spectral profiles of the four feature types distinct from each other? Which profiles are the most similar overall?

Question 9. Which bands or indices provide the greatest separation between the four feature types?

Section 3. Delineating Refugee Settlement Boundaries

Now that we've become familiar with the different land cover types and the changes that can occur as a refugee settlement is established, let's turn to formally delineating the refugee settlement from its surroundings by mapping a settlement boundary. Having information on refugee settlement boundaries is helpful for the basic accounting of refugee settlement extent and for confidently attributing land cover or land use changes to a specific refugee settlement (Friedrich and Van Den Hoek 2020, Van Den Hoek and Friedrich 2021). In this section, we'll use a *k*-means unsupervised classifier to generate a settlement/non-settlement map that represents land that has been transformed by the refugee settlement's establishment or subsequent use. Note that the settlement boundary that we used in Sect. 1 is a settlement planning boundary established by the UNHCR and so represents the land within the formal boundary that *potentially* could be accessed or used by refugees.

To start making a binary classification that separates settlement from non-settlement, let's create a random sample of 500 NDVI values from across the post-establishment composite. Remember that the `postMedian` composite was clipped to the 500-meter-buffered extent of the UNHCR settlement boundary geometry, so these sample sites should be dispersed inside and outside of the UNHCR boundary's geometry. For parameterization, we only need two values output from the classifier (`numClusters` = 2) and can set the maximum number of iterations to a low value of 5 (`maxIter` = 5) and the seed value to an arbitrary value of 21. Now let's apply the classifier to the post-establishment composite, view the coverage of settlement (pixel value of 1) and non-settlement (pixel value of 0), and visually compare the result with the UNHCR settlement boundary.

```
// Create samples to input to a K-means classifier.
var numPx = 500;
var samples = postMedian.select('NDVI').sample({
    scale: 30, // Landsat resolution
    numPixels: numPx,
    geometries: true
});

Map.addLayer(samples, {}, 'K-means samples');

// Set-up the parameters for K-means.
var numClusters = 2;
var maxIter = 5;
var seedValue = 21;

// Seed the classifier using land cover samples.
var clusterer = ee.Clusterer.wekaKMeans({
    nClusters: numClusters,
    maxIterations: maxIter,
    seed: seedValue
}).train(samples);

// Apply the K-means classifier.
var kmeansResult = postMedian.cluster(clusterer);
Map.addLayer(kmeansResult, {
```

```
    bands: ['cluster'],
    max: 1,
    min: 0
}, 'K-means output');
```

The resulting *k*-means classification looks promising for separating settlement from non-settlement pixels, but it has plenty of gaps in settlement coverage as well as isolated settlement patches and pixels. To produce a single contiguous settlement coverage, let's apply spatial morphological operations of dilation and erosion on the *k*-means output. Dilation incrementally expands the boundary of a raster dataset, filling gaps and connecting patches along the way. Conversely, erosion chips away at the outermost pixels, thereby removing the surplus pixels that were added during the dilation step but still maintaining the filled-in gaps.

We'll apply these in sequence, first dilation and then erosion, using `focal_max` and `focal_min`, respectively; `focal_max` works as a dilation since it outputs the maximum value detected within the kernel, which will always be a settlement pixel because the settlement pixel value of 1 is always greater than the non-settlement pixel value of 0. Since we just need to do some fine-tuning on the boundary of the settlement coverage, we can use a kernel with a small radius of 3. Finally, let's convert the output of the dilation and erosion to a polygon `FeatureCollection` where each contiguous patch of pixels becomes its own polygon (Fig. A1.7.4). Feel free to map the outline of Pagirinya in blue, as above, for a helpful visual reference.

```
// Define the kernel used for morphological operations.
var kernel = ee.Kernel.square({
    radius: 3
});

// Perform a dilation followed by an erosion.
var kMeansCleaned = kmeansResult
    .focal_max({
        kernel: kernel,
        iterations: 1
    }) // Dilation
    .focal_min({
        kernel: kernel,
        iterations: 1
```

```
    }); // Erosion
Map.addLayer(kMeansCleaned, {
    bands: ['cluster'],
    max: 1,
    min: 0
}, 'K-means cleaned');

// Convert cleaned K-means settlement and non-settlement coverages to
polygons.
var kMeansCleanedPolygon = kMeansCleaned.reduceToVectors({
    scale: 30,
    eightConnected: true
});

Map.addLayer(kMeansCleanedPolygon, {}, 'K-Means cleaned polygon');

// Map outline of Pagirinya in blue.
Map.addLayer(pagirinyaOutline,
    {
        palette: '0000FF'
    },
    'Pagirinya Refugee Settlement boundary');
```

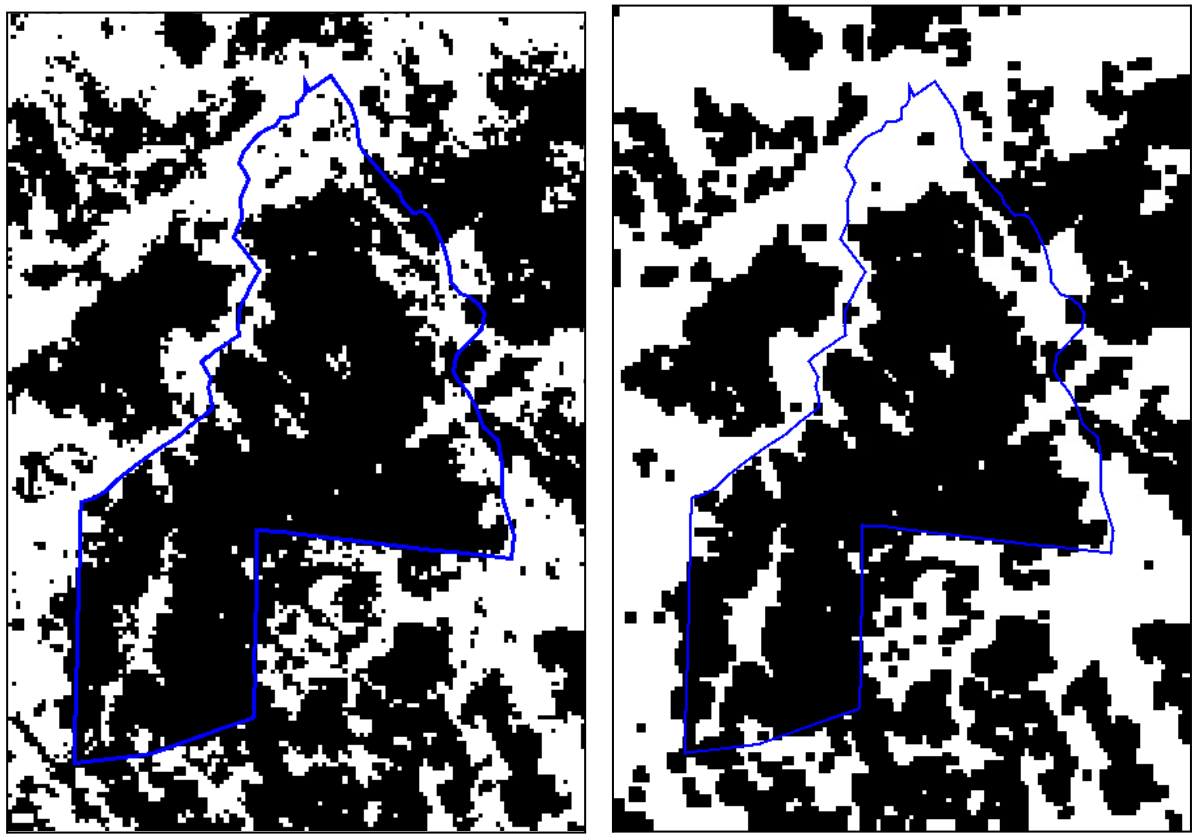


Fig. A1.7.4 *K*-means output before (left) and after (right) dilation and erosion, with Pagirinya Refugee Settlement boundary overlaid in blue

We've created a usable vector map of settlement and non-settlement polygons, but we are aiming for a single polygon that represents the settlement boundary. To filter these polygons to a single polygon that represents the refugee settlement's boundary, let's use a simple logic rule and select the polygon that has the largest overlap (i.e., intersected area) with the UNHCR boundary (Fig. A1.7.5).

```
// Intersect K-means polygons with UNHCR settlement boundary and
// return intersection area as a feature property.
var kMeansIntersect = kMeansCleanedPolygon.map(function(feat) {
    var boundaryIsect = pagirinya.intersection(feat, ee
        .ErrorMargin(1));
    return ee.Feature(feat).set({
        'isectArea': boundaryIsect.area()
```

```
    });
});

// Sort to select the polygon with largest overlap with the UNHCR
settlement boundary.
var kMeansBoundary = ee.Feature(kMeansIntersect.sort('isectArea',
    false).first());
Map.addLayer(kMeansBoundary, {}, 'K-Means Settlement Boundary');
```

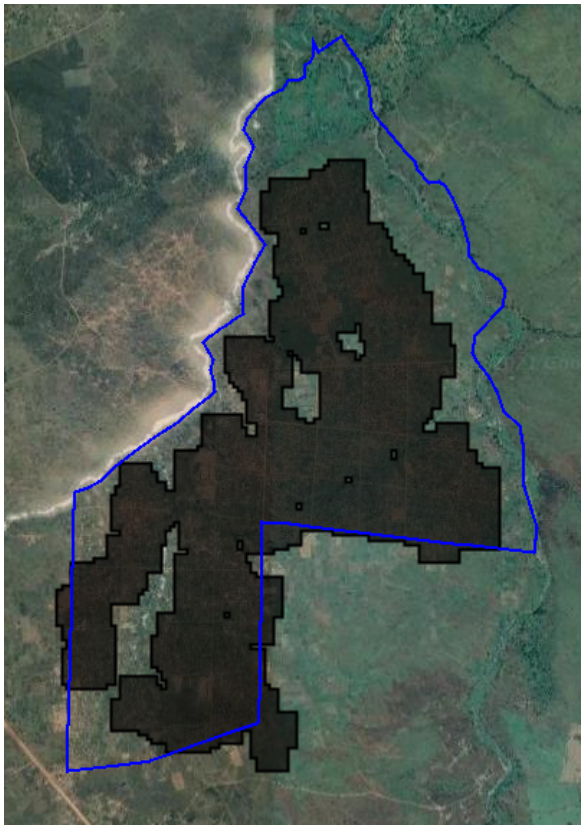


Fig. A1.7.5 *K*-means settlement boundary (black) overlaid by UNHCR settlement boundary (blue)

Code Checkpoint A17c. The book's repository contains a script that shows what your code should look like at this point.

Question 10. In your opinion, does the *k*-means boundary accurately separate the settlement from its surroundings? Considering differences between the UNHCR boundary and the *k*-means boundary, comment on potential errors of commission (areas that are inaccurately included in the *k*-means boundary) and of omission (areas that are inaccurately excluded).

Question 11. Rather than just collecting samples for input to *k*-means based on NDVI in `postMedian`, adjust the script above to sample from all bands in `postMedian`. How does the resulting settlement polygon differ? Does increasing the amount of spectral information available to the classifier improve the result?

Question 12. Rerun the *k*-means classifier based on the `diffMedian` image from Sect. 1 rather than the `postMedian` image while keeping the other parameters the same. How does the resulting settlement boundary polygon differ?

Section 4. Estimating Refugee Population Within the Settlement

Thus far, we have looked at land cover conditions and land cover changes at Pagirinya and used that information to help map the extent of the settlement. Let's turn towards using satellite-derived data to estimate the size of the refugee population living at Pagirinya. Knowing how many refugees are at a settlement is essential for gauging the need for food aid and for guiding sustainable development and disaster risk reduction efforts. Satellite-informed population estimates can be useful for these purposes, especially if no other data are available.

In this final section, we will work with several datasets used to estimate the geographic distribution of human populations, each of which is based in part on remote-sensing detection of buildings. We will analyze population estimates at Pagirinya Refugee Settlement from the Global Human Settlement Layer (GHSL), High Resolution Settlement Layer (HRSL), and WorldPop data products. To gauge the accuracy of these products, we will compare the population estimates with UNHCR-recorded refugee population data from September 2020.

HRSL and WorldPop data are from 2020, and GHSL is available for multiple years, most recently 2015. Let's filter the GHSL `ImageCollection` to only the dataset. We'll also rename all relevant bands to `'population'` for consistency, and map all population maps using the same visualization approach to better support a direct comparison. Use the **Inspector** tool to identify the different pixel-level values for each population dataset

within and around Pagirinya. These values represent the human population estimated to be present at each pixel in 2015.

```
Map.centerObject(pagirinya, 14);

var ghslPop = ee.ImageCollection('JRC/GHSL/P2016/POP_GPW_GLOBE_V1')
    .filter(ee.Filter.date('2015-01-01', '2016-01-01')).first()
    .select(['population_count'], ['population']);
var hrslPop = ee.Image('projects/gee-book/assets/A1-7/HRSL')
    .select(['b1'], ['population']);
var worldPop = ee.ImageCollection(
        'WorldPop/GP/100m/pop_age_sex_cons_unadj')
    .filterMetadata('country', 'equals', 'UGA')
    .first()
    .select(['population']);

// Set-up visualization to be shared by all population datasets.
var visualization = {
    bands: ['population'],
    min: 0.0,
    max: 50.0,
    palette: ['24126c', '1fff4f', 'd4ff50']
};

// Map population datasets.
Map.addLayer(ghslPop, visualization, 'GHSL Pop');
Map.addLayer(hrslPop, visualization, 'HRSL Pop');
Map.addLayer(worldPop, visualization, 'WorldPop');
```

You'll notice that each dataset has a different spatial resolution (also commonly referred to as the "scale"). We'll need to know these different spatial resolutions when we summarize each dataset's population estimate across Pagirinya using `reduceRegion`. Once we have a population estimate, we'll add it to the Pagirinya feature as a new property.

```
// Collect spatial resolution of each dataset.
var ghslPopProjection = ghslPop.projection();
```

```
var ghslPopScale = ghslPopProjection.nominalScale();
print(ghslPopScale, 'GHSL scale');

var hrslPopProjection = hrslPop.projection();
var hrslPopScale = hrslPopProjection.nominalScale();
print(hrslPopScale, 'HRSL scale');

var worldPopProjection = worldPop.projection();
var worldPopScale = worldPopProjection.nominalScale();
print(worldPopScale, 'WorldPop scale');

// Summarize population totals for each population product at each
settlement and
// assign as new properties to the UNHCR boundary Feature.
pagirinya = pagirinya.set(ghslPop.select(['population'], ['GHSL'])
    .reduceRegion({
        reducer: 'sum',
        scale: ghslPopScale,
        geometry: pagirinya.geometry(),
        maxPixels: 1e9,
    }));

pagirinya = pagirinya.set(hrslPop.select(['population'], ['HRSL'])
    .reduceRegion({
        reducer: 'sum',
        scale: hrslPopScale,
        geometry: pagirinya.geometry(),
        maxPixels: 1e9,
    }));

pagirinya = pagirinya.set(worldPop.select(['population'], [
    'WorldPop'])
    .reduceRegion({
        reducer: 'sum',
        scale: worldPopScale,
        geometry: pagirinya.geometry(),
        maxPixels: 1e9,
```

```
    }));

print(pagirinya, 'Pagirinya with population estimates');
```

Now we have three very different population estimates for Pagirinya based on the three population datasets. Let's see how they compare to the population recorded in 2020 by UNHCR, which is also stored as a property of the Pagirinya feature.
To do so, we'll simply subtract the UNHCR population total from each dataset's estimated population total and store each difference as a new property. A negative difference indicates an underestimation of the UNHCR population, and a positive difference indicates an overestimation.

```
// Measure difference between settlement product and UNHCR-recorded
population values.
var unhcrPopulation = ee.Number(pagirinya.get('UNHCR_Pop'));
var ghslDiff = ee.Number(pagirinya.get('GHSL')).subtract(
    unhcrPopulation);
var hrslDiff = ee.Number(pagirinya.get('HRSL')).subtract(
    unhcrPopulation);
var worldPopDiff = ee.Number(pagirinya.get('WorldPop')).subtract(
    unhcrPopulation);

// Update UNHCR boundary Feature with population difference
properties.
pagirinya = pagirinya.set(ee.Dictionary.fromLists(['GHSL_diff',
        'HRSL_diff', 'WorldPop_diff'
    ],
    [ghslDiff, hrslDiff, worldPopDiff]));

print('Pagirinya Population Estimations', pagirinya);
```

Code Checkpoint A17d. The book's repository contains a script that shows what your code should look like at this point.

Question 13. Visually interpret the coverage of each population dataset alongside the building footprint data from Sect. 2. Which population dataset seems to better capture population density at hot spots of building footprints?

Question 14. Many buildings in Pagirinya are not household dwellings but rather administrative offices, shops, food market buildings, etc., and such differences in building use are not necessarily considered in generating the population estimates. How would the inclusion of non-dwellings in population datasets bias settlement-level population estimates?

Question 15. Note that the coverage of the WorldPop population data at Pagirinya is not wholly contained within the UNHCR settlement boundary. Is this "spillover" better captured by the *k*-means boundary from Sect. 3?

Synthesis

You may have noticed that we showed a 2020 land cover map from the European Space Agency (ESA) based on Sentinel-1 and Sentinel-2 data in Fig. A1.7.1c but did not make use of those land cover data in the practicum. How would your settlement boundary detection approach and results change if you used Sentinel-2 instead of Landsat data and sampled land cover sites from this ESA dataset? As a homework challenge, please complete the following assignment.

Assignment 1. Use Sentinel-2 surface reflectance data collected in 2020. Collect 20 samples of each land cover class in the ESA land cover product within Pagirinya using `ee.Image.stratifiedSample`. Assess the spectral separability between land cover classes. Then, run a modified *k*-means classifier that makes use of Sentinel-2 NDVI values collected across the ESA land cover map.

Conclusion

This chapter introduced approaches for characterizing land cover dynamics within and surrounding Pagirinya Refugee Settlement using a range of open-access satellite data and geospatial products. We saw that satellite remote sensing approaches are effective for characterizing land cover changes before and following the establishment of Pagirinya in 2016, and for delineating a refugee settlement boundary that represents land directly affected by the settlement's establishment and use. We also noted wide disagreement and pronounced inaccuracies in Pagirinya refugee population estimates based on satellite-informed human population datasets. This chapter shows the value of remote sensing for long-term monitoring of refugee settlements as well as the need for deeper integration of humanitarian data and scenarios in remote sensing applications.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Friedrich HK, Van Den Hoek J (2020) Breaking ground: Automated disturbance detection with Landsat time series captures rapid refugee settlement establishment and growth in North Uganda. Comput Environ Urban Syst 82:101499. https://doi.org/10.1016/j.compenvurbsys.2020.101499

Maystadt JF, Mueller V, Van Den Hoek J, Van Weezel S (2020) Vegetation changes attributable to refugees in Africa coincide with agricultural deforestation. Environ Res Lett 15:44008. https://doi.org/10.1088/1748-9326/ab6d7c

UNHCR (2020) Global Trends: Forced Displacement in 2020

Van Den Hoek J, Friedrich HK, Wrathall D (2021) A primer on refugee-environment relationships. In: PERN Cyberseminar on Refugee and internally displaced populations, environmental impacts and climate risks

Van Den Hoek J, Friedrich HK (2021) Satellite-based human settlement datasets inadequately detect refugee settlements: A critical assessment at thirty refugee settlements in Uganda. Remote Sens 13:3574. https://doi.org/10.3390/rs13183574

Chapter A1.8: Monitoring Gold Mining Activity Using SAR

Authors
Lucio Villa, Sidney Novoa, Milagros Becerra, Andréa Puzzi Nicolau, Karen Dyson, Karis Tenneson, John Dilger

Overview

The expansion of gold mining has had a large impact on the rainforests of the Amazon over the last decades. To take just one example, it has affected both the biodiversity and the lives of local people in the Madre de Dios region of southeastern Peru. In this chapter, we will review a methodology developed to generate early warnings of deforestation based on the use of synthetic aperture radar (SAR) images. First, we will identify the Sentinel-1 images suitable for the construction of a time series of preprocessed datasets. Second, we will run a change detection analysis based on a statistical analysis of the Sentinel-1 images (Canty et al. 2019). Finally, we will show the steps to follow in the post-processing stage by filtering information with forest / non-forest and bodies of water datasets.

Learning Outcomes

- Selecting and creating a multitemporal SAR mosaic.
- Generating SAR change detection based on a statistical analysis of Sentinel-1 images.
- Post-processing of alerts generated by filtering maximum area patches and information of forest/non-forest and water bodies.

Helps if you know how to:

- Import, filter, and visualize images (Part F1).
- Work with time-series data in Earth Engine (Part F4).
- Write a function and `map` it over an ImageCollection (Chap. F4.0).
- Use the `require` function to load code from existing modules (Chap. F6.1).
- Export results to assets (Chap. F6.2).

Introduction to Theory

Accelerated demand for natural resources has transformed the Amazon rainforest into a new economic frontier that generates commodities such as agricultural products, livestock, and more recently, minerals, especially gold (RAISG 2020). In Peru, illegal gold mining is a serious problem that affects local populations in the southeastern region of Madre de Dios (Yard et al. 2012, Asner and Tupayachi 2017, Alvarez-Berrios et al. 2021). According to Caballero et al. (2018), it has led to the deforestation of about 1,000 km^2 of rainforest, affecting protected areas, Indigenous communities, and sustainable management areas. Gold mining is carried out throughout the year, even during the rainy season.

Optical Earth-observation satellites have played a very important role in monitoring gold-mining deforestation in recent years. Madre de Dios is one of the few tropical regions in the world for which there is well-documented information on annual forest loss due to this deforestation driver (Asner et al. 2013, Asner and Tupayachi 2017, Caballero et al. 2018, Nicolau et al. 2019, Csillick and Asner 2020, Alarcón Aguirre et al. 2021). However, these resources do not reveal the progress of illegal mining during the rainy season and at other times when the optical images are obscured by clouds. This reduces the window of opportunity to know the dynamics of the activity and guide actions to control the advance of deforestation.

Satellite-based SAR sensors obtain information throughout the entire year thanks to the ability of microwaves to penetrate cloud cover (Ballere et al. 2021, Nicolau et al. 2021). Therefore, we are able to capture changes due to gold mining expansion with SAR data from the European Space Agency (ESA) Sentinel-1 C-band satellite. In this chapter, we will look at a successful example of how to process radar images to generate early warnings of gold-mining deforestation.

Practicum

SAR sensors transmit microwave signals at an oblique angle and measure the backscattered portion of the signal in order to analyze features on the surface. Unlike optical sensors, which are passive, SAR is an active instrument with its own source of illumination, and it is one of the few sensing instruments that allows full control of the signal polarization on both the transmit and receive paths. The majority of today's SAR sensors are linearly polarized, and transmit horizontally and/or vertically polarized wave forms (i.e., SAR bands can be VV, VH, HV, or HH). While interpreting optical imagery is similar to interpreting a photograph, interpreting SAR data requires a different way of

thinking, in that the signal is instead responsive in complex ways to surface characteristics such as structure and moisture. More information about the theory and concepts behind SAR is available in the SAR Handbook (Flores-Anderson et al. 2019).

The area of study for this chapter is the region surrounding a mining corridor located in Madre de Dios in the southern Peruvian Amazon (Fig. A1.8.1).

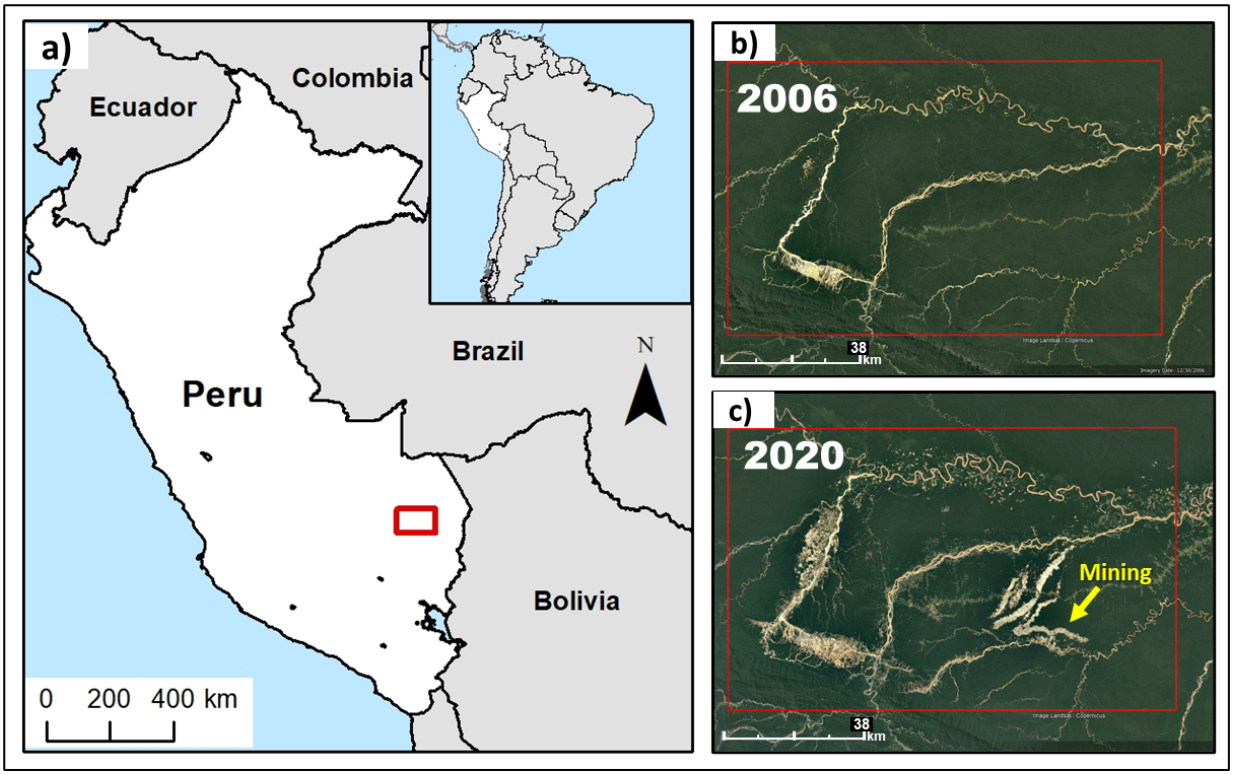


Fig. A1.8.1 Maps of (a) the area of study in the southern Peruvian Amazon, and the expansion of alluvial mining activity between (b) 2006 and (c) 2020

Alluvial gold mining exploitation generates changes in the radar backscattering mechanism over forested areas, since it involves deforestation, removal of topsoil, excavation, and the use of water for the extraction of gold from the loose sediment. Fig. A1.8.2 shows the changes in SAR Sentinel-1 images and a radar backscattering mechanism before (Fig. A1.8.2a) and after (Fig. A1.8.2b) the impact of alluvial gold mining activity.

Therefore, you will learn how to use SAR Sentinel-1 time series to detect changes generated by alluvial gold mining activities in forested areas.

The first step is to create a SAR mosaic for a given period of time and orbit pass. Then, we will apply the omnibus Q-test algorithm to generate change alerts from the SAR Sentinel-1 time series. Finally, we will filter and eliminate potential false positive alerts coming from other activities with the same temporal pattern as the mining activity (e.g., natural forest loss by river expansion or water over bare soil during the rainy season).

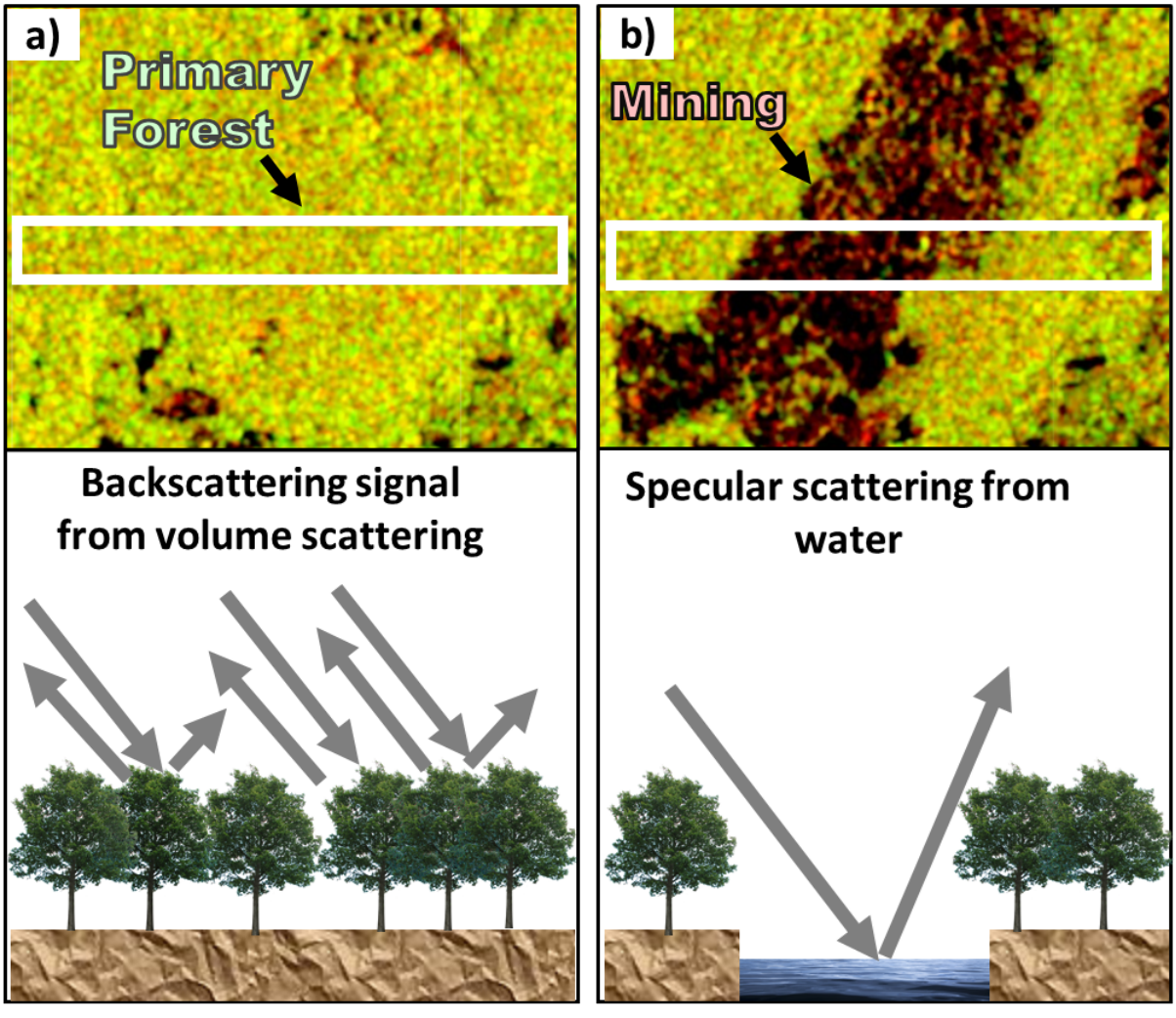


Fig. A1.8.2 Radar backscattering mechanism from an area affected by alluvial mining activity a) before: backscattering signal from primary forest (volume scattering);

b) after: specular scattering from bearing soil and water as a result of clearing forest cover, removing topsoil, digging pits and using water in the extraction process of gold from the loose sediment

Section 1. Creating a Single SAR Mosaic

We will use ESA's Sentinel-1 dual-polarized (VV+VH) in descending orbit pass to create a single SAR mosaic over the area of study. Sentinel-1 SAR Ground Range Detected (GRD) data (Fig. A1.8.3) is stored in Earth Engine in two formats: logarithmic scale (dB) and the original values (power scale named as `FLOAT`). We will use the latter since mathematical operations should not be applied into data on a logarithmic scale. The Sentinel-1 dataset is composed of Level-1 SAR amplitude multi-look images preprocessed according to the following steps: orbit metadata update, removal of border and thermal noise, radiometric calibration and terrain correction.

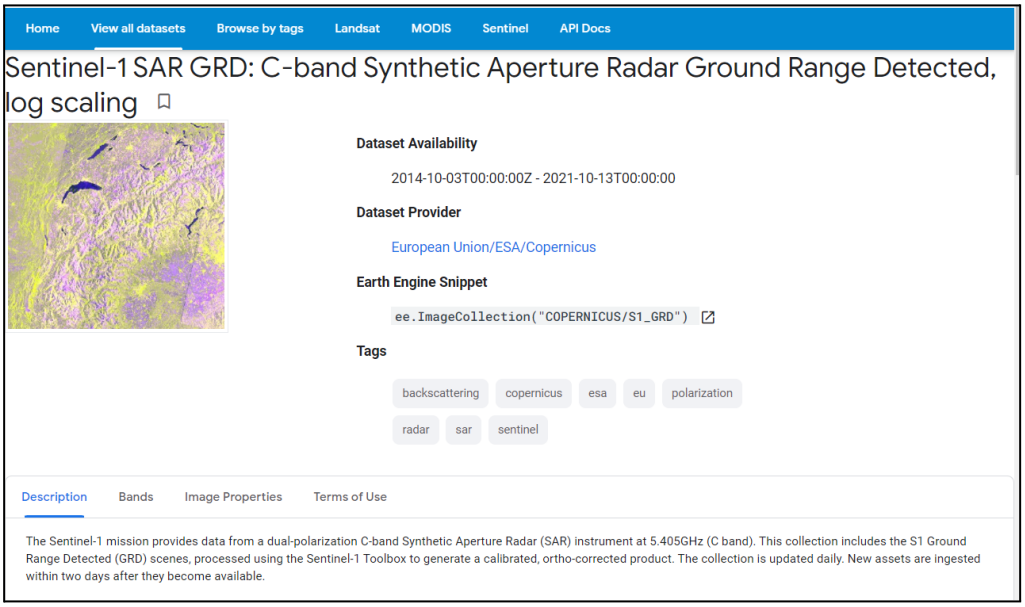


Fig. A1.8.3 Sentinel-1 imagery available in the Earth Engine data catalog

Copy and paste the code below to define the area of study (Fig. A1.8.1), convert this vector to a boundary image, and add it to the map (Fig. A1.8.4).

```
////////////////////////////////////////////////////
/// Section One
////////////////////////////////////////////////////

// Define the area of study.
var aoi = ee.FeatureCollection('projects/gee-book/assets/A1-8/mdd');

// Center the map at the aoi.
Map.centerObject(aoi, 9);

// Create an empty image.
var empty = ee.Image().byte();

// Convert the area of study to an EE image object
// so we can visualize only the boundary.
var aoiOutline = empty.paint({
    featureCollection: aoi,
    color: 1,
    width: 2
});

// Select the satellite basemap view.
Map.setOptions('SATELLITE');

// Add the area of study boundary to the map.
Map.addLayer(aoiOutline, {
    palette: 'red'
}, 'Area of Study');
```

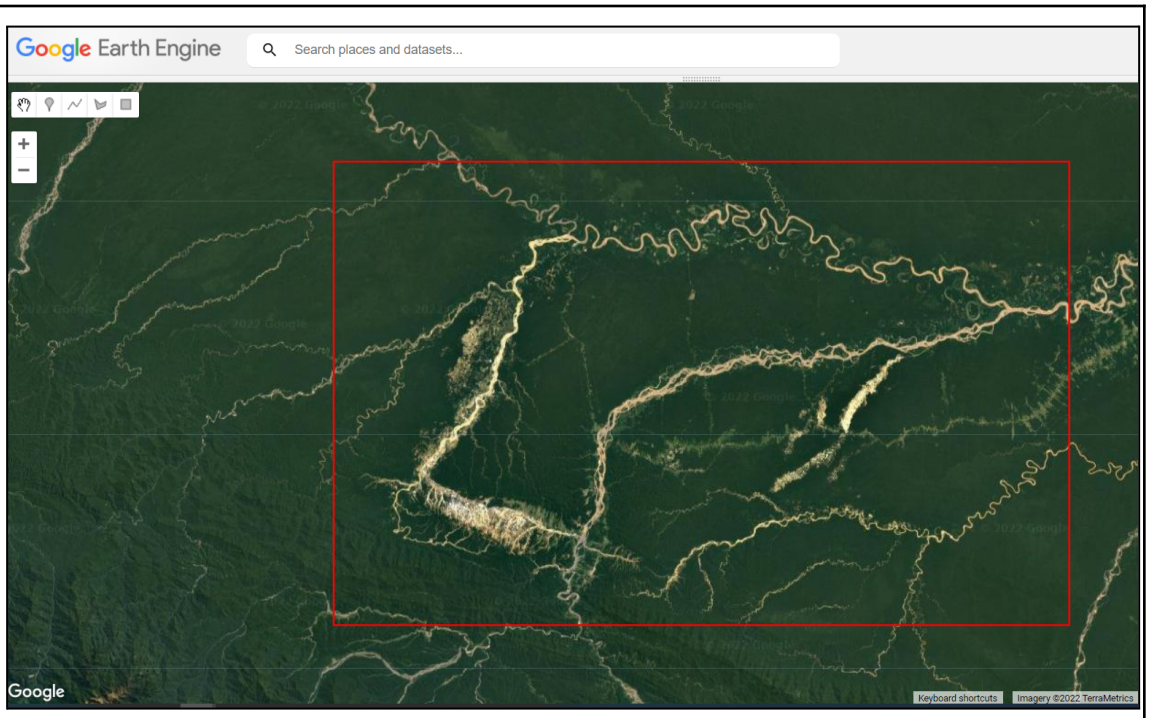


Fig. A1.8.4 The area of study around the mining corridor located in the Madre de Dios region

Next, copy and paste the code below to define two functions, `maskAngle` and `getCollection`. The first function masks sections of SAR images acquired at an incidence angle less than 31 or greater than 45 degrees. The second function filters the Sentinel-1 imagery to a specific period of time, region of interest, and orbit pass. Note that the Sentinel-1 GRD dataset is imported inside the second function.

```
// Function to mask the SAR images acquired with an incidence angle
// lower or equal to  31 and greater or equal to 45 degrees.
function maskAngle(image) {
    var angleMask = image.select('angle');
    return image.updateMask(angleMask.gte(31).and(angleMask.lte(45)));
}

// Function to get the SAR Collection.
function getCollection(dates, roi, orbitPass0) {
    var sarCollFloat = ee.ImageCollection('COPERNICUS/S1_GRD_FLOAT')
```

```
        .filterBounds(roi)
        .filterDate(dates[0], dates[1])
        .filter(ee.Filter.eq('orbitProperties_pass', orbitPass0));
    return sarCollFloat.map(maskAngle).select(['VV', 'VH']);
}
```

Copy and paste the code below to import the Sentinel-1 collection, define time variables (a list of dates) and the orbit pass variable, apply the functions, create a mosaic by using the `mosaic` function, and clip the mosaic to the study area.

```
// Define variables: the period of time and the orbitpass.
var listOfDates = ['2021-08-01', '2021-08-12'];
var orbitPass = 'DESCENDING';

// Apply the function to get the SAR mosaic.
var sarImageColl = getCollection(listOfDates, aoi, orbitPass)
    .mosaic()
    .clip(aoi);
print('SAR Image Mosaic', sarImageColl);
```

Before adding the mosaic to the map, it's important to scale the values to a logarithmic scale (`log10().multiply(10.0)`). The parameters of visualization (`sarVis`) should be taken between 3 and -23 decibels (dB). Copy and paste the code below to do so and to add the mosaic to the map (Fig. A1.8.5). The code creates a scaled image and draws it using visualization parameters set through trial and error.

```
// Apply logarithmic scale.
var sarImageScaled = sarImageColl.log10().multiply(10.0);

// Visualize results.
var sarVis = {
    bands: ['VV', 'VH', 'VV'],
    min: [-18, -23, 3],
    max: [-4, -11, 15]
};
Map.addLayer(sarImageScaled, sarVis, 'Sentinel-1 / SAR Mosaic');
```

Code Checkpoint A18a. The book's repository contains a script that shows what your code should look like at this point.

Question 1. How many bands (polarizations) does Sentinel-1 have over the area of study?

Question 2. Using the **Inspector** tool, explore the values from the VV and VH bands over different land covers. Which one do you think is better able to detect forested areas?

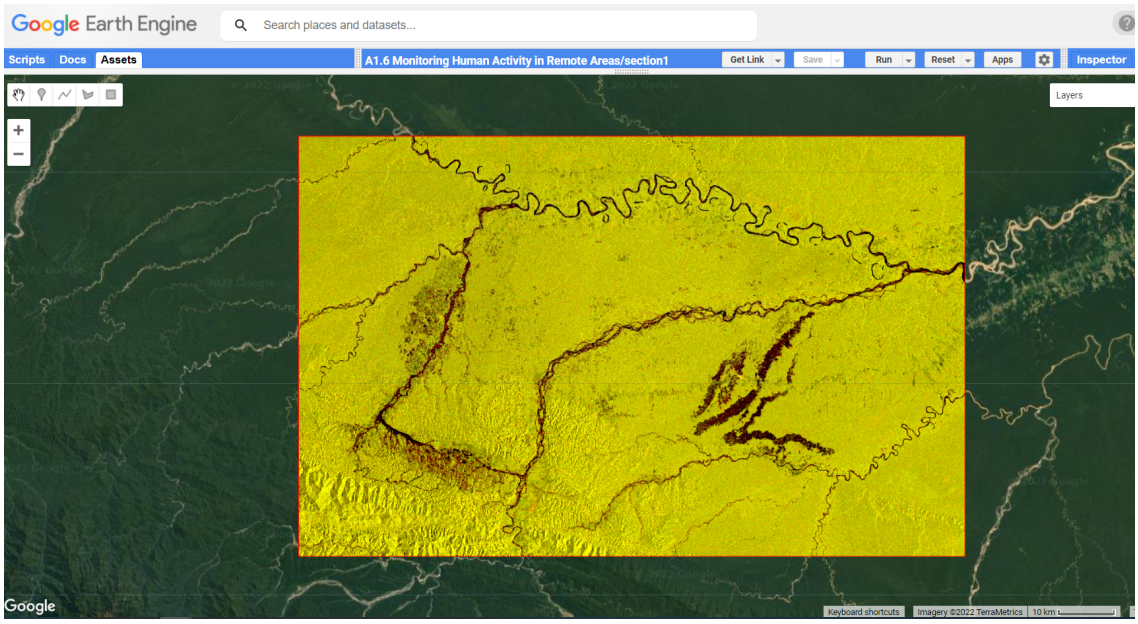


Fig. A1.8.5 SAR Sentinel-1 mosaic generated in previous section

Section 2. Creating a SAR Mosaic Time Series

We will reuse most of the code from Section 1, only changing the period of time and not applying the mosaic function just yet. Start this section by opening the following code checkpoint.

Code Checkpoint A18b. The book's repository contains a script to use to begin this section. You will need to start with that script and paste code below into it.

Expand the SAR `ImageCollection` in the **Console** and note that it is composed of 30 elements. To create a time series of mosaics, we need to define two additional functions (`getDates` and `mosaicSAR`). The first function converts the format of the date from milliseconds to format `'YYYY-MM-dd'`. The second function filters the SAR ImageCollection using the list of dates and generates a mosaic. The result is an `ImageCollection` of mosaics per date. Copy and paste the code below to define these two functions.

```
// Function to get dates in 'YYYY-MM-dd' format.
function getDates(dd) {
    return ee.Date(dd).format('YYYY-MM-dd');
}

// Function to get a SAR Mosaic clipped to the study area.
function mosaicSAR(dates1) {
    dates1 = ee.Date(dates1);
    var imageFilt = sarImageColl
        .filterDate(dates1, dates1.advance(1, 'day'));
    return imageFilt.mosaic()
        .clip(aoi)
        .set({
            'system:time_start': dates1.millis(),
            'dateYMD': dates1.format('YYYY-MM-dd')
        });
}
```

Now, copy and paste the code below to generate a list of dates without duplicate elements (i.e., where there are images from the same dates in the collection, we only keep one). We avoid duplicates by using the `ee.List.distinct` and the `getDates` functions, and output an `ImageCollection` of mosaics per date.

```
// Function to get a SAR Collection of mosaics by date.
var datesMosaic = ee.List(sarImageColl
        .aggregate_array('system:time_start'))
    .map(getDates)
    .distinct();
```

```
// Get a SAR List and Image Collection of mosaics by date.
var getMosaicList = datesMosaic.map(mosaicSAR);
var getMosaicColl = ee.ImageCollection(getMosaicList);
print('get Mosaic Collection', getMosaicColl);
```

Finally, copy and paste the code below to set the visualization parameter (`sarVis`) and add two SAR mosaics filtered by the date of acquisition as an example (one from 2021-01-04 and the other from 2021-12-18; Fig. A1.8.6).

```
// Visualize results.
var sarVis = {
    bands: ['VV', 'VH', 'VV'],
    min: [-18, -23, 3],
    max: [-4, -11, 15]
};

var image1 = getMosaicColl
    .filter(ee.Filter.eq('dateYMD', '2021-01-04'))
    .first().log10().multiply(10.0);
var image2 = getMosaicColl
    .filter(ee.Filter.eq('dateYMD', '2021-12-18'))
    .first().log10().multiply(10.0);

Map.addLayer(image1, sarVis, 'Sentinel-1 | 2021-01-04');
Map.addLayer(image2, sarVis, 'Sentinel-1 | 2021-12-18');
```

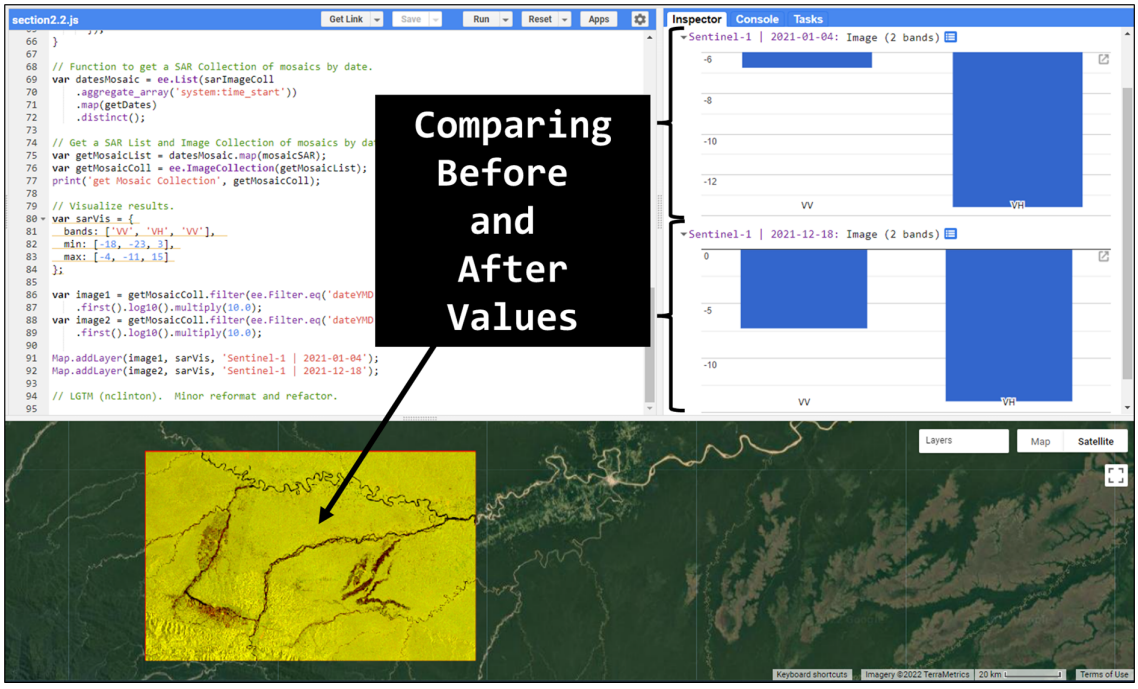


Fig. A1.8.6 SAR Sentinel-1 mosaics generated in Section 2. The **Inspector** tool shows a comparison between before and after values of VV and VH SAR bands.

Note that we applied the logarithmic scale for visualization purposes. Zoom in and switch between the layers to note the differences between the images.

Code Checkpoint A18c. The book's repository contains a script that shows what your code should look like at this point.

Question 3. How many images were taken by Sentinel-1 over the area of study between 2018-01-01 and 2020-01-01?

Question 4. Using the **Inspector** tool, explore the temporal changes of the values from VV and VH bands over new mining areas.

Section 3. Generate SAR Change Detection

There are different methods for detecting changes using SAR data. In this case, we will use a SAR change detection method based on Canty et al. (2020).

This methodology allows us to identify changes for a series of "*k*" uncorrelated SAR images using a pixel-based omnibus likelihood ratio test statistic Q for covariance matrices ($\sum_i$, i = 1. . . *k*), based on a Wishart distribution. The Q test is defined by

$$\ln Q = n\left(pk\ln k + \sum_{i=1}^{k}\ln|X_i| - k\ln\left|\sum_{i=1}^{k}X_i\right|\right)$$

Where $X_i = n\Sigma^*$ (with Σ^* being the maximum likelihood estimate of the covariance matrices Σ_i), "n" the number of looks and "*p*" the dimensionality of the covariance matrices (with p = 2 for dual polarización SAR data). The | · | denotes the determinant.

The Q test is an omnibus test statistic because it evaluates the equality of several covariance matrices simultaneously. Thus, this test statistic tests the null hypothesis (no change, H_0) against alternative hypothesis (change, H_1) using SAR time series pixel-based data and the level of significance (probability that the null hypothesis H_0 is true, also known as *p*-value) estimated in each iteration.

In the first iteration, the first two first images in the time series are tested (null hypothesis of no change against the alternative of change):

1) $H_0: \sum_1 = \sum_2$ against $H_1: \sum_1 \neq \sum_2$ ⟶ Null hypothesis rejected?

If the Null hypothesis (H_0) is not rejected, then the test continues including the values of the next image in the series:

2) $H_0: \sum_1 = \sum_2 = \sum_3$ against $H_1: \sum_1 = \sum_2 \neq \sum_3$ ⟶ Null hypothesis rejected?

However, if the null hypothesis is rejected in this iteration, then the interval of time for this change is labeled and the test is restarted from there.

The explanation of the Omnibus likelihood ratio test statistic is beyond the scope of this chapter but more details can be found in Canty et al. (2020).

The next steps correspond to obtaining a single change detection output image using the time series of mosaics generated in Section 2.

To do so, we will use modules adapted from the original JavaScript libraries by Canty et al (2020). Beginning from the last code checkpoint from Section 2, copy and paste the code below to import the adapted modules and define a variable that stores the number of images in our collection. This number will be used later on for visualization purposes.

```
// Libraries of SAR Change Detection (version modified).
// The original version can be found in:
// users/mortcanty/changedetection
var omb = require(
    'projects/gee-edu/book:Part A - Applications/A1 - Human
Applications/A1.8 Monitoring Gold Mining Activity Using
SAR/modules/omnibusTest_v1.1'
);
var util = require(
    'projects/gee-edu/book:Part A - Applications/A1 - Human
Applications/A1.8 Monitoring Gold Mining Activity Using
SAR/modules/utilities_v1.1'
);

// Count the length of the list of dates of the time-series.
var countDates = datesMosaic.size().getInfo();
```

Before applying the SAR change algorithm, we need to define the input parameters such as the `significance` and the reducer to be applied (`median` in this case). The result is an `ee.Dictionary` that contains several images: among these are `cmap`, `smap`, `fmap`, `bmap`. The `cmap` image shows the occurrence of the most recent significant change, `smap` shows the first significant change, `fmap` shows the frequency of significant changes, and `bmap` shows the interval in which each significant change occurred.
Copy and paste the code below to define such variables, apply the algorithm to the list of SAR mosaics (`getMosaicList`), and extract the results.

```
// Run the algorithm and print the results.
var significance = 0.0001;
var median = true;
var result = ee.Dictionary(omb.omnibus(getMosaicList, significance,
    median));
```

```
print('result', result);

// Change maps generated (cmap, smap, fmap and bmap)
// are detailed in the next commented lines.

// cmap: the interval in which the most recent significant change
occurred (single-band).
// smap: the interval in which the first significant change occurred
(single-band).
// fmap: the frequency of significant changes (single-band).
// bmap: the interval in which each significant change occurred ((k -
1)-band).

// Extract and print the images result
// (cmap, smap, fmap and bmap) from the ee.Dictionary.
var cmap = ee.Image(result.get('cmap')).byte();
var smap = ee.Image(result.get('smap')).byte();
var fmap = ee.Image(result.get('fmap')).byte();
var bmap = ee.Image(result.get('bmap')).byte();
```

The values for `cmap`, `smap`, and `bmap` are numbers that correspond to dates. These are the dates that are stored in the `datesMosaic` list. For example, the pixel value of 0 corresponds to the first date (2021-01-04), the pixel value of 1 corresponds to the second date (2021-01-16), and so on (Fig. A1.8.7).

If we want to export the resulting images we need to also export the list of dates. To do so, we need to create a `FeatureCollection` since we can't currently export lists directly in Earth Engine. Copy and paste the code below to create a `FeatureCollection` where each feature contains the date information as a property. We are also printing the dates in order to visualize the pixel-date association (expand the list on the **Console** to see it.)

```
// Build a Feature Collection from Dates.
var fCollectionDates = ee.FeatureCollection(datesMosaic
    .map(function(element) {
        return ee.Feature(null, {
            prop: element
        });
```

```
    }));
print('Dates', datesMosaic);
```

system:index	prop
0	2021-01-04
1	2021-01-16
2	2021-01-28
3	2021-02-09
4	2021-02-21
5	2021-03-05
6	2021-03-17
7	2021-03-29
8	2021-04-10
9	2021-04-22
10	2021-05-04
11	2021-05-16
12	2021-05-28
13	2021-06-09
14	2021-06-21
15	2021-07-03
16	2021-07-15
17	2021-07-27
18	2021-08-08
19	2021-08-20
20	2021-09-01
21	2021-09-13
22	2021-09-25
23	2021-10-07
24	2021-10-19
25	2021-10-31
26	2021-11-12
27	2021-11-24
28	2021-12-06
29	2021-12-18

Fig. A1.8.7 List of dates to be exported as a CSV file. Each value (index) is associated with a specific date of change of the raster file (smap).

Now we can add the results to the map. Copy and paste the code below to define visualizations parameters, make a legend that associates date numbers with colors, and add the resulting images (Fig. A1.8.8).To load results faster, change the

`Map.centerObject` function at the top of the script to `Map.setCenter(-70.003, -12.849, 12)`—we are zooming in to a specific area—and leave only the `smap` layer checked under the **Layers** panel.

```
// Visualization parameters.
var jet = ['black', 'blue', 'cyan', 'yellow', 'red'];
var vis = {
    min: 0,
    max: countDates,
    palette: jet
};

// Add resulting images and legend to the map.
Map.add(util.makeLegend(vis));
Map.addLayer(cmap, vis, 'cmap - recent change (unfiltered)');
Map.addLayer(smap, vis, 'smap - first change (unfiltered)');
Map.addLayer(fmap.multiply(2), vis, 'fmap*2 - frequency of changes');
```

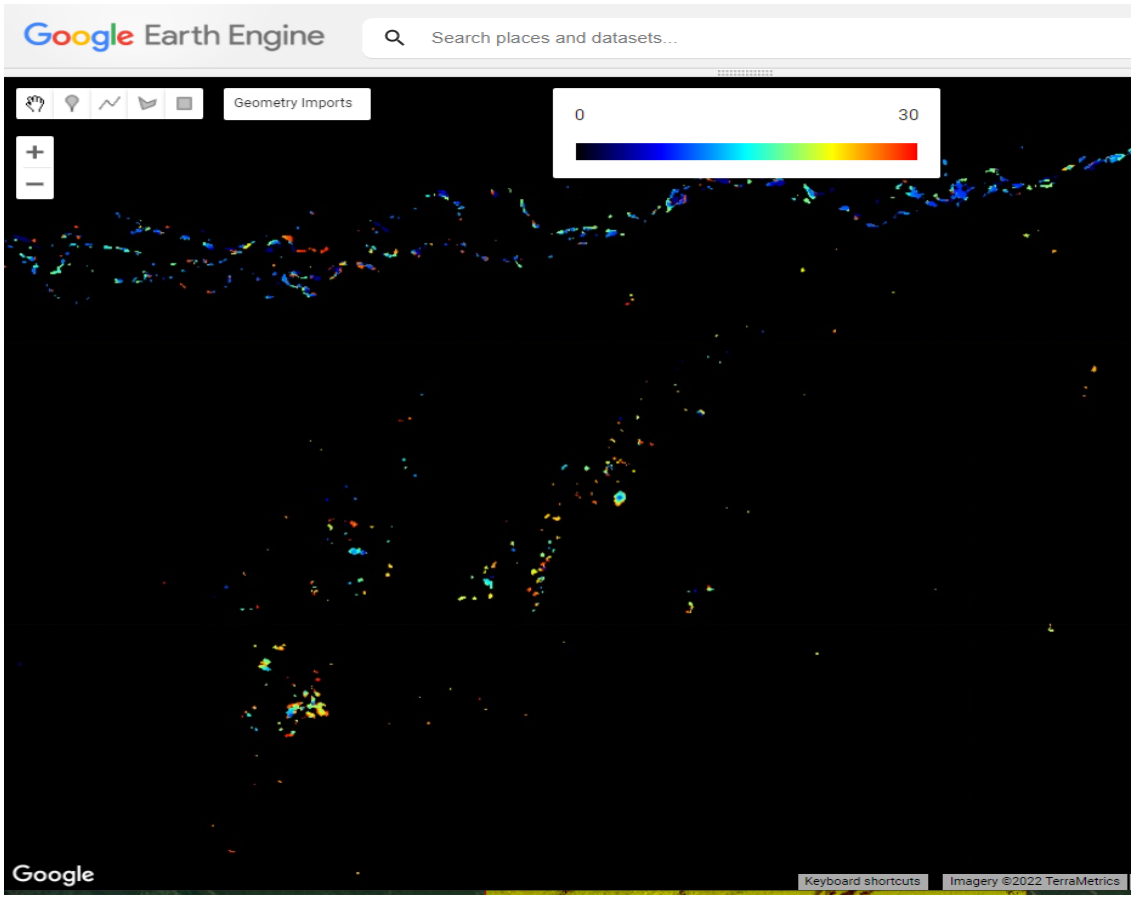


Fig. A1.8.8 SAR change detection results from applying the Q-test algorithm

Now, copy and paste the code below to export two items to Google Drive: `fCollectionDates`, the dates of SAR images processed (Fig. A1.8.7); and `smap`, the image of the first significant change (Fig. A1.8.8).

```
//  Export the Feature Collection with the dates of change.
var exportDates = Export.table.toDrive({
    collection: fCollectionDates,
    folder: 'datesChangesDN',
    description: 'dates',
    fileFormat: 'CSV'
});
// Export the image of the first significant changes.
```

```
var exportImgChanges = Export.image.toAsset({
    image: smap,
    description: 'smap',
    assetId: 'your_asset_path_here/' + 'smap',
    region: aoi,
    scale: 10,
    maxPixels: 1e13
});
```

Code Checkpoint A18d. The book's repository contains a script that shows what your code should look like at this point.

Question 5. What is the difference between the `smap` and `cmap` images?

Question 6. How does the `FeatureCollection` `fCollectionDate` relate to the change maps `smap`, `cmap`, and `fmap`?

Section 4. Filtering and Postprocessing Alerts

As explained earlier in the Practicum, the `smap` results need to be filtered in order to eliminate possible false positives. The false positives are associated with the forest loss due to river morphology and the presence of muddy water bodies (Fig. A1.8.2).

In this section, we will explore different options to filter false positives and, therefore, postprocess the results generated.

Like we did in previous sections, copy and paste the code below into a new script to import the study area and the exported `smap` image. Remember that our analysis covered 30 Sentinel-1 images from distinct dates.

```
////////////////////////////////////////////////////
/// Section Four
////////////////////////////////////////////////////

// Define the area of study.
var aoi = ee.FeatureCollection('projects/gee-book/assets/A1-8/mdd');
```

```
// Center the map.
Map.centerObject(aoi, 10);

// Create an empty image.
var empty = ee.Image().byte();

// Convert the area of study to an EE image object so we can visualize
// only the boundary.
var aoiOutline = empty.paint({
    featureCollection: aoi,
    color: 1,
    width: 2
});

// Select the satellite basemap view.
Map.setOptions('SATELLITE');

// Add the area of study boundary to the map.
Map.addLayer(aoiOutline, {
    palette: 'red'
}, 'Area of Study');

// Import the smap result from section 3.
var changeDetect = ee.Image('projects/gee-book/assets/A1-8/smap');

// Visualization parameters.
var countDates = 30;
var jet = ['black', 'blue', 'cyan', 'yellow', 'red'];
var vis = {
    min: 0,
    max: countDates,
    palette: jet
};

// Add results to the map.
Map.addLayer(changeDetect, vis, 'Change Map Unfiltered');
```

Next, copy and paste the code below to import from the Earth Engine data catalog and add to the map all the sources of information for filtering false positives: Shuttle Radar Topography Mission (SRTM) digital elevation data, Hansen Global Forest Change data, and JRC Global Surface Water data (Fig. A1.8.9).

```
// Digital Elevation Model SRTM.
//
https://developers.google.com/earth-engine/datasets/catalog/USGS_SRTMG
L1_003
var srtm = ee.Image('USGS/SRTMGL1_003').clip(aoi);
var slope = ee.Terrain.slope(srtm);
var srtmVis = {
    min: 0,
    max: 1000,
    palette: ['black', 'blue', 'cyan', 'yellow', 'red']
};
Map.addLayer(srtm, srtmVis, 'SRTM Elevation');
var slopeVis = {
    min: 0,
    max: 15,
    palette: ['black', 'blue', 'cyan', 'yellow', 'red']
};
Map.addLayer(slope, slopeVis, 'SRTM Slope');

// Hansen Global Forest Change v1.8 (2000-2020)
//
https://developers.google.com/earth-engine/datasets/catalog/UMD_hansen
_global_forest_change_2020_v1_8
var gfc = ee.Image('UMD/hansen/global_forest_change_2020_v1_8').clip(
    aoi);
var forest2020 = gfc.select('treecover2000')
    .gt(0)
    .updateMask(gfc.select('loss')
        .neq(1))
    .selfMask();
Map.addLayer(forest2020,
    {
```

```
        min: 0,
        max: 1,
        palette: ['black', 'green']
    },
    'Forest cover 2020');

// JRC Yearly Water Classification History, v1.3 (Updated until Dec
2020).
//
https://developers.google.com/earth-engine/datasets/catalog/JRC_GSW1_3
_GlobalSurfaceWater
var waterJRC = ee.Image('JRC/GSW1_3/GlobalSurfaceWater').select(
    'max_extent');
var waterVis = {
    min: 0,
    max: 1,
    palette: ['blue', 'black']
};
Map.addLayer(waterJRC.eq(0), waterVis, 'Water Bodies until 2020');
```

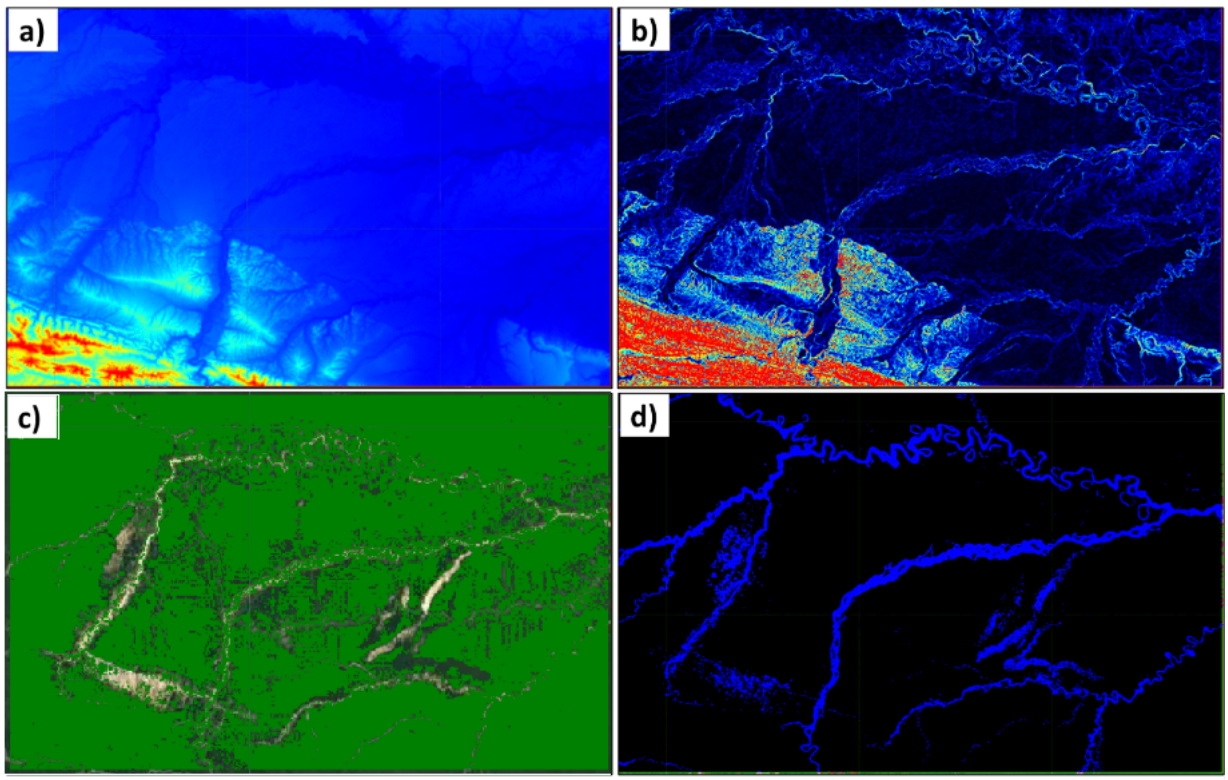


Fig. A1.8.9 Layers used to filter false positives alerts: a) SRTM elevation, red color shows areas over 1000 meters above sea level; b) SRTM slope, red color shows areas with slope over 15 degrees; c) Hansen Global Forest Change, green color shows forested areas updated to 2020; d) JRC Yearly Water Classification History, blue color shows the maximum extent of water surface detected from 1984 to 2020.

You can toggle these layers on and off, zoom in and out, and inspect pixel values to analyze them.

The SRTM elevation layer (`'SRTM Elevation'`) and the slope (`'SRTM Slope'`) derived from it with `ee.Terrain.slope` show red areas that correspond to an altitude over 1000 meters above sea level and a slope over 15 degrees, compared to blue areas, which are closer to sea level or to flat terrain. We chose these options because mining activity in this region is located in lowlands. Furthermore, the SAR Sentinel-1 data provided by Earth Engine are not radiometric-terrain corrected. So, steep slopes (>15 degrees) generate distortions in SAR images, and therefore, potential false changes between two or more images taken at different times.

The Hansen Global Forest Change data are composed of forested areas in 2000 (the `'treecover2000'` band) and the forest loss between 2001 and 2020. In this sense, the binary layer `'Forest cover 2020'` previously generated and added shows in green the forested areas updated until 2020, with all the non-forested and the forest loss between 2001-2020 areas masked.

The JRC Yearly Water Classification History data shows surface water extent and change between 1984 and 2020. The binary layer `'Water Bodies until 2020'` previously added shows in blue the water bodies' maximum extent between 1984 and 2020. Non-water bodies are shown in black.

Note that alluvial mining expansion pattern in the study area is associated with primary forest loss, and the appearance of new surface water patches (Fig. A1.8.2).

Now, we will filter the false positives based on thresholds. We will mask any pixel in `smap` marked as change over areas greater than 1000 meters above sea level, slope greater than 15 degrees according to the SRTM data (classified as forest until 2020 by the Hansen data), and that are not classified as water bodies by the JRC dataset. Copy and paste below to add the filtered results to the map.

```
// Apply filters through masks.
var alertsFiltered = changeDetect
    .updateMask(srtm.lt(1000))
    .updateMask(slope.lt(15))
    .updateMask(forest2020.eq(1))
    .updateMask(waterJRC.eq(0))
    .selfMask();

// Add filtered results to the map.
Map.addLayer(alertsFiltered,
    {
        min: 0,
        max: countDates,
        palette: jet
    },
```

```
    'Change Map Filtered',
    1);
```

We can still improve the results a bit more. Copy and paste the code below to define and apply a function that eliminates small pixel patches and isolated pixels. We do this because we know that in this area, mining activities occur in areas of at least 0.5 hectares.

```
// Function to filter small patches and isolated pixels.
function filterMinPatchs(alerts0, minArea0, maxSize0) {
    var pixelCount = alerts0.gt(0).connectedPixelCount(maxSize0);
    var minPixelCount = ee.Image(minArea0).divide(ee.Image
    .pixelArea());
    return alerts0.updateMask(pixelCount.gte(minPixelCount));
}

// Apply the function and visualize the filtered results.
var alertsFiltMinPatchs = filterMinPatchs(alertsFiltered, 10000, 200);

Map.addLayer(alertsFiltMinPatchs, vis,
    'Alerts Filtered - Minimum Patches');
```

Turn off all the other layers to visualize the filtered result. By analyzing the results without the filters and with the filters, we can see that we eliminated most of the false positives (Fig. A1.8.10).

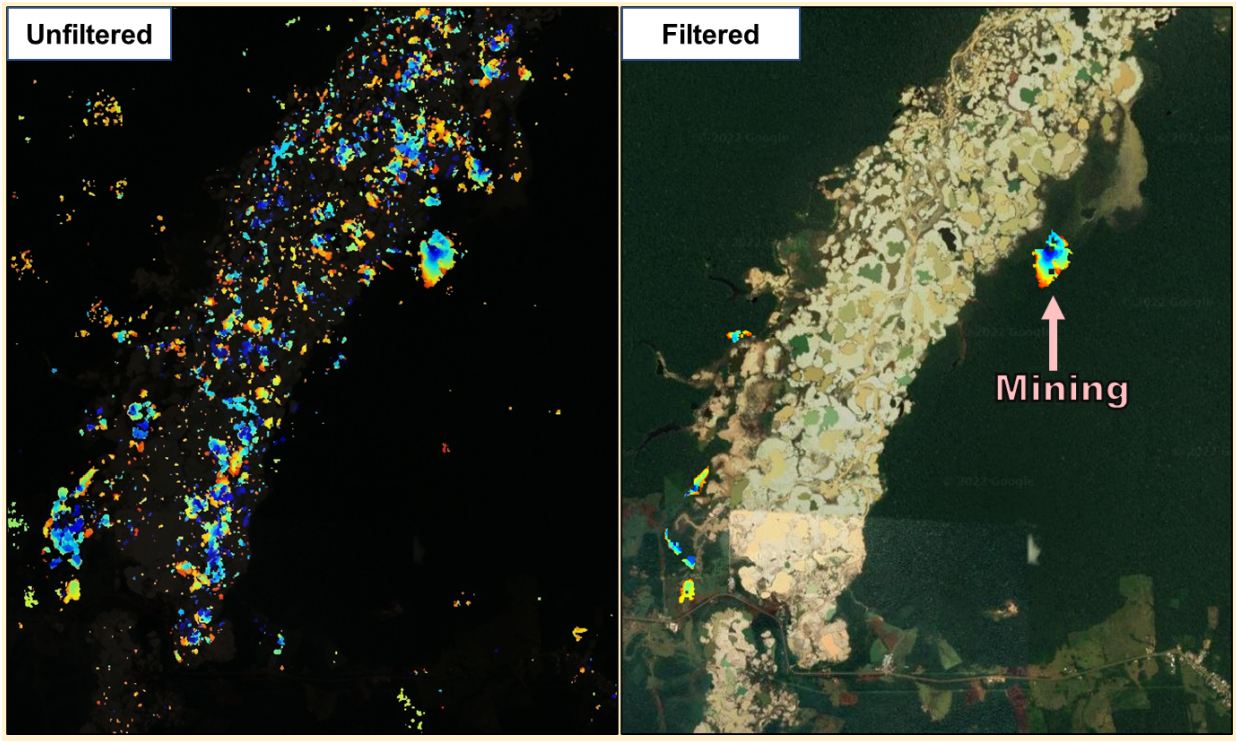


Fig. A1.8.10 A comparison of results, after (`'Alerts Filtered - Minimum Patches'`) and before filtering false positives (`'Change Map Unfiltered'`).

Finally, we can export the outcome. Copy and paste the code below to export to the Drive.

```
// Export filtered results to the Drive.
Export.image.toDrive({
    image: alertsFiltMinPatchs,
    description: 'alertsFiltered',
    folder: 'alertsFiltered',
    region: aoi,
    scale: 10,
});
```

Code Checkpoint A18e. The book's repository contains a script that shows what your code should look like at this point.

Synthesis

In this chapter, we mapped the changes generated by the alluvial mining activity over a forested area and between a period of time using Sentinel-1 SAR time series. For this, we separated the methodology into three steps. First, we were able to build a time series from Sentinel-1 mosaics. Second, we estimate all the changes based on the omnibus Q-test change detection algorithm. Finally, we filter the detected changes based on existing forest/non-forest, water bodies, and elevation data, and a minimum mapping unit in order to retrieve the changes generated by the alluvial mining activity only.

Now, it's your turn to explore the use of the methodology.

Assignment 1. In this chapter we applied the methodology for a given SAR orbit. Describe how we could identify the different SAR orbits over a specific area of study.

Assignment 2. Describe whether these alerts, which are generated by a change detection algorithm, are different for the ascending or descending orbit over our area of study.

Conclusion

In this chapter, you have learned how to analyze the changes generated by alluvial mining activity over forested areas based on the application of a SAR change detection methodology. This is possible because of the significant impact generated by this activity over the environment (that is, the deforestation and the use of water in the alluvial gold wash machine) that is reflected in the backscatter signal of SAR data. This methodology can be applied to other study cases since a good understanding of the principles of change detection has been achieved in this chapter and complemented by chapters in part F4.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

Aguirre GA, Robles RRC, Duarez FMG, et al (2021) Dinámica de la pérdida de bosques en el sureste de la Amazonia peruana: Un estudio de caso en Madre de Dios. Ecosistemas 30:2175. https://doi.org/10.7818/ECOS.2175

Álvarez-Berríos N, L'Roe J, Naughton-Treves L (2021) Does formalizing artisanal gold mining mitigate environmental impacts? Deforestation evidence from the Peruvian Amazon. Environ Res Lett 16:64052. https://doi.org/10.1088/1748-9326/abede9

Asner GP, Llactayo W, Tupayachi R, Luna ER (2013) Elevated rates of gold mining in the Amazon revealed through high-resolution monitoring. Proc Natl Acad Sci USA 110:18454–18459. https://doi.org/10.1073/pnas.1318271110

Asner GP, Tupayachi R (2017) Accelerated losses of protected forests from gold mining in the Peruvian Amazon. Environ Res Lett 12:94004. https://doi.org/10.1088/1748-9326/aa7dab

Ballère M, Bouvet A, Mermoz S, et al (2021) SAR data for tropical forest disturbance alerts in French Guiana: Benefit over optical imagery. Remote Sens Environ 252:112159. https://doi.org/10.1016/j.rse.2020.112159

Bourbigot M, Johnsen H, Piantanida R, Hajduch G (2021) Sentinel-1 Product Specification for products generated with IPF 3.4.0. CLS S-1 Mission Perform. Cent. pp. 186

Caballero Espejo J, Messinger M, Román-Dañobeytia F, et al (2018) Deforestation and forest degradation due to gold mining in the Peruvian Amazon: A 34-year perspective. Remote Sens 10:1903. https://doi.org/10.20944/preprints201811.0113.v1

Canty MJ, Nielsen AA, Conradsen K, Skriver H (2020) Statistical analysis of changes in Sentinel-1 time series on the Google Earth Engine. Remote Sens 12:46. https://doi.org/10.3390/rs12010046

Csillik O, Asner GP (2020) Near-real time aboveground carbon emissions in Peru. PLoS One 15:e0241418. https://doi.org/10.1371/journal.pone.0241418

Flores-Anderson AI, Herndon KE, Thapa RB, Cherrington E (2019) The SAR Handbook: Comprehensive Methodologies for Forest Monitoring and Biomass Estimation

Fröjse L (2014) Unsupervised change detection using multi-temporal SAR data: A case study of Arctic sea ice. KTH: Royal Institute of Technology

Gorelick N, Hancher M, Dixon M, et al (2017) Google Earth Engine: Planetary-scale geospatial analysis for everyone. Remote Sens Environ 202:18–27. https://doi.org/10.1016/j.rse.2017.06.031

Kellndorfer J, Flores-Anderson AI, Herndon KE, Thapa RB (2019) Using SAR data for mapping deforestation and forest degradation. SAR Handbook Compr Methodol For Monit Biomass Estim ServirGlobal Hunstville, AL, USA 65–79. https://doi.org/10.25966/68c9-gw82

Nicolau AP, Flores-Anderson A, Griffin R, et al (2021) Assessing SAR C-band data to effectively distinguish modified land uses in a heavily disturbed Amazon forest. Int J Appl Earth Obs Geoinf 94:102214. https://doi.org/10.1016/j.jag.2020.102214

Nicolau AP, Herndon K, Flores-Anderson A, Griffin R (2019) A spatial pattern analysis of forest loss in the Madre de Dios region, Peru. Environ Res Lett 14:124045. https://doi.org/10.1088/1748-9326/ab57c3

Proisy C, Mougin E, Fromard F, Rudant JP (1998) Télédétection radar des mangroves de Guyane Française. Séminaire Télédétection Végétation, Montpellier, Fr 1996-11-26, Photo interprétation 36:26

RAISG (2020) Amazonia Under Pressure. https://atlas2020.amazoniasocioambiental.org/en. Accessed 25 Feb 2022

Richards JA (2009) Remote Sensing with Imaging Radar. Springer

Rignot EJM, van Zyl JJ (1993) Change detection techniques for ERS-1 SAR data. IEEE Trans Geosci Remote Sens 31:896–906. https://doi.org/10.1109/36.239913

Yard EE, Horton J, Schier JG, et al (2012) Mercury exposure among artisanal gold miners in Madre de Dios, Peru: A cross-sectional study. J Med Toxicol 8:441–448. https://doi.org/10.1007/s13181-012-0252-0

Outline

Below is an outline of the entire section, including every section header.