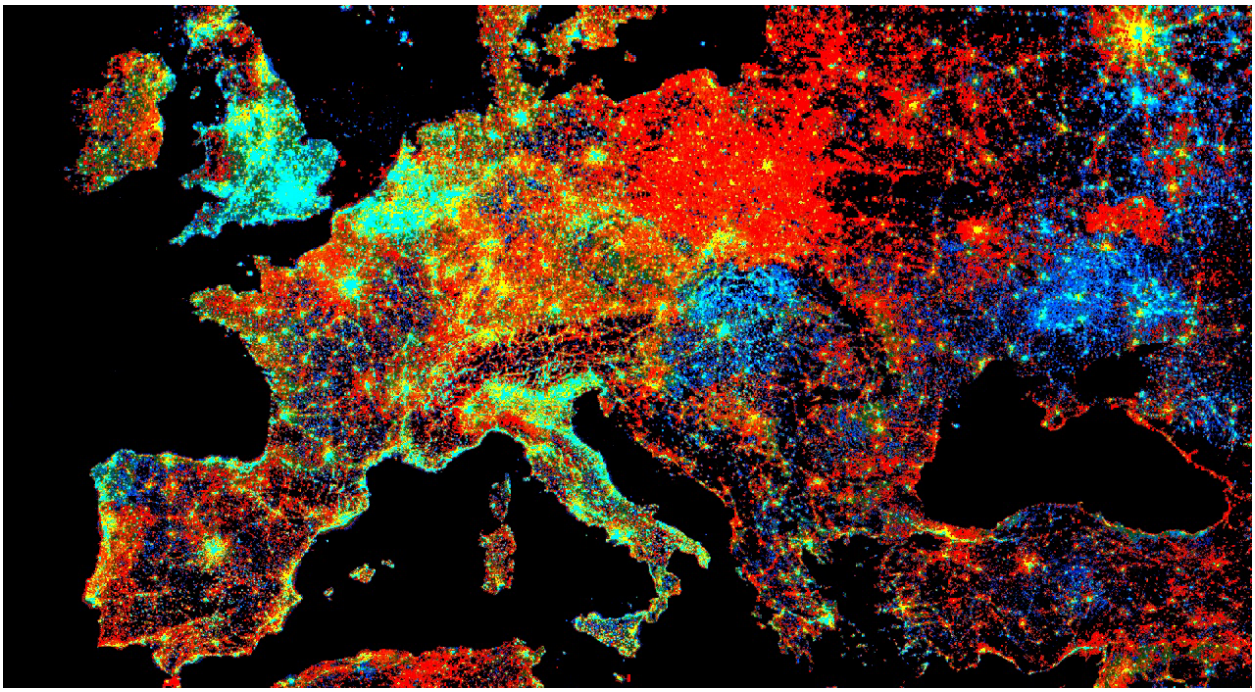
Cloud-Based Remote Sensing with Google Earth Engine



Fundamentals and Applications

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

Part A3: Terrestrial Applications

For chapters A3.1 to A 3.5,
go to this document

Chapter A3.6: Working With GPS and Weather Data

Authors
Peder Engelstad, Daniel Carver, Nicholas E. Young

Overview

The purpose of this chapter is to demonstrate how to use Google Earth Engine as a means of associating remotely sensed data (weather observations) with open-source GPS point locations. These methods will provide a quick and easy way to access and analyze large amounts of information relative to your own research and efficiently move your data outside Earth Engine.

Learning Outcomes

- Pairing values from remotely sensed data with uploaded data.
- Exporting features from Earth Engine.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Exporting calculated data to tables with Tasks (Chap. F5.0).

Introduction to Theory

Knowing how animals interact with their environment is critical to understanding how to manage them. The choices animals make are influenced by basic survival needs (e.g., food, shelter, water) and dynamic factors such as local weather conditions. Without direct observation, it can be difficult to understand the relationship between animal movement and weather conditions.

In this chapter, we will explore information retrieved from a GPS collar worn by a cougar, and explore its relationship to daily temperature estimates from the Daymet climatological dataset available in Google Earth Engine. This will require us to bring an asset into Earth Engine, connect the weather values to the point locations, and bring those value-added data back out of Earth Engine for further analysis.

Practicum

Section 1. GPS Location Data

Using GPS collars, Mahoney et al. (2017) tracked the movement of 2 cougars and 16 coyotes in central Utah. These data were used to understand some of the behavioral patterns of the individuals. These data have been freely shared with the broader research community and the public through Movebank, an online repository for animal movement datasets from across the globe. While some Movebank datasets list only the contact information of the authors, others (like those from the Mahoney study) allow you to display and download the information via an interactive web map.

We will pair the data on cougar movement with Daymet weather data. According to the Daymet website, the dataset "provides gridded estimates of daily weather parameters. Seven surface weather parameters are available at a daily time step, 1 km x 1 km spatial resolution, with a North American spatial extent allowing a rich resource of daily surface meteorology."

With data for every day at a 1 km^2 spatial resolution, the Daymet data are a great resource for the temporal and spatial scale at which a cougar would interact with the landscape. There are seven measured values in total, allowing us to check multiple aspects of the weather to assess how it may be affecting behavior (Fig. A3.6.1).

Parameter	Abbr	Units	Description
Day length	dayl	s/day	Duration of the daylight period in seconds per day. This calculation is based on the period of the day during which the sun is above a hypothetical flat horizon
Precipitation	prcp	mm/day	Daily total precipitation in millimeters per day, sum of all forms converted to water-equivalent. Precipitation occurrence on any given day may be ascertained.
Shortwave radiation	srad	W/m2	Incident shortwave radiation flux density in watts per square meter, taken as an average over the daylight period of the day. NOTE: Daily total radiation (MJ/m2/day) can be calculated as follows: ((srad (W/m2) * dayl (s/day)) / l,000,000)
Snow water equivalent	swe	kg/m2	Snow water equivalent in kilograms per square meter. The amount of water contained within the snowpack.
Maximum air temperature	tmax	degrees C	Daily maximum 2-meter air temperature in degrees Celsius.
Minimum air temperature	tmin	degrees C	Daily minimum 2-meter air temperature in degrees Celsius.
Water vapor pressure	vp	Pa	Water vapor pressure in pascals. Daily average partial pressure of water vapor.

Fig. A3.6.1 Metadata for Daymet imagery within Earth Engine

Section 2. Bringing Data into Earth Engine

In this chapter, we will discuss how to import assets into Earth Engine, extract values from a dataset, and export those values out of Earth Engine. The processes by which you can bring data into Earth Engine change frequently, so it is best to go directly to the documentation to view the latest updates.

Section 2.1. Bringing in an Asset

Using the following code, start a fresh script and import Mahoney's cougar movement data from Movebank (which has already been uploaded for you in the assets of this book). Here, we will focus on a single animal (cougar ID F53). Because the original dataset was in csv format, it has been converted into a shapefile outside of GEE. During this process, it is important to note that data has been projected to the WGS 1984 (EPSG: 4326) coordinate reference system. Projecting to EPSG:4326 is recommended in Earth Engine to minimize the reprojection errors during the uploading process.

```
// Import the data and add it to the map and print.
var cougarF53 = ee.FeatureCollection(
```

```
    'projects/gee-book/assets/A3-6/cougarF53');

Map.centerObject(cougarF53, 10);

Map.addLayer(cougarF53, {}, 'cougar presence data');

print(cougarF53, 'cougar data');
```

You can use the **Inspector** tool to look at the attribute data associated with the new asset. With the points visualized, make a geometry feature that encompasses our area of interest by selecting the square geometry tool and drawing a box that encompasses the points (Fig. A3.6.2). We will use the geometry feature to filter our climate data.

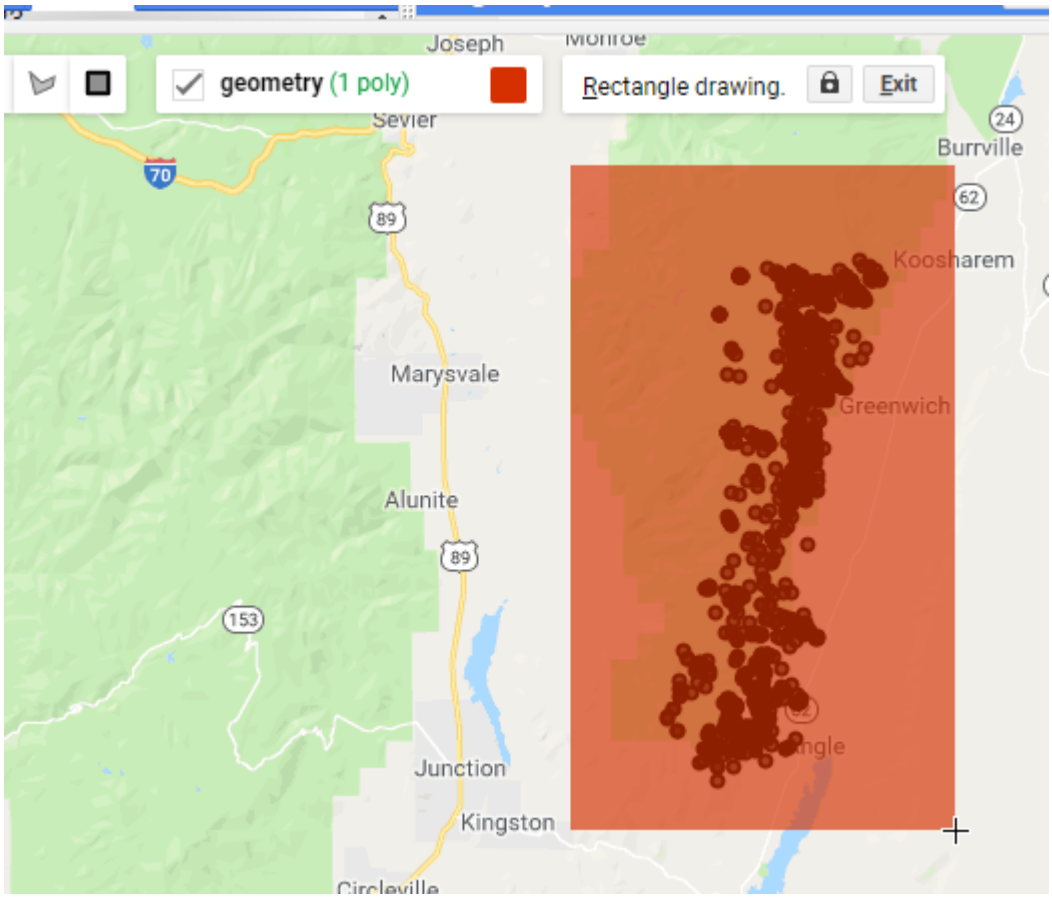


Fig. A3.6.2 Draw a geometry feature around the points to spatially filter the climate

data

Section 2.2. Defining Weather Variables

In this chapter, we are using Earth Engine as a means of associating remotely sensed data (i.e., our rasters) with our point locations. While the process is conceptually straightforward, it does take some work to accomplish. With our points loaded, the next step is to import the Daymet weather variables.

We are using the NASA-derived dataset Daymet V4 due to its 1 km^2 spatial resolution and the fact that it measures the environmental variables we think might be related to cougar movement or behavior. We will import these variables by calling the unique ID of the dataset, filtering it to our bounding box geometry and the dates during which the data were collected.

```
// Call in image collection and filter.
var Daymet = ee.ImageCollection('NASA/ORNL/DAYMET_V4')
    .filterDate('2014-02-11', '2014-11-02')
    .filterBounds(geometry)
    .map(function(image) {
        return image.clip(geometry);
    });

print(Daymet, 'Daymet');
```

```
▼ ImageCollection NASA/ORNL/DAYMET_V4 (264 elements)
    type: ImageCollection
    id: NASA/ORNL/DAYMET_V4
    version: 1633091282089416
    bands: []
  ▼ features: List (264 elements)
      ▸ 0: Image NASA/ORNL/DAYMET_V4/20140211 (7 bands)
      ▸ 1: Image NASA/ORNL/DAYMET_V4/20140212 (7 bands)
      ▸ 2: Image NASA/ORNL/DAYMET_V4/20140213 (7 bands)
      ▸ 3: Image NASA/ORNL/DAYMET_V4/20140214 (7 bands)
      ▸ 4: Image NASA/ORNL/DAYMET_V4/20140215 (7 bands)
      ▸ 5: Image NASA/ORNL/DAYMET_V4/20140216 (7 bands)
      ▸ 6: Image NASA/ORNL/DAYMET_V4/20140217 (7 bands)
      ▸ 7: Image NASA/ORNL/DAYMET_V4/20140218 (7 bands)
      ▸ 8: Image NASA/ORNL/DAYMET_V4/20140219 (7 bands)
      ▸ 9: Image NASA/ORNL/DAYMET_V4/20140220 (7 bands)
      ▸ 10: Image NASA/ORNL/DAYMET_V4/20140221 (7 bands)
```

Fig. A3.6.3 A view of the structure of the Daymet V4 data from the print statement

From the print statement (Fig. A3.6.3), we can see that this is an `ImageCollection` with 264 images (though your total number of images may be different, as the dataset changes over time). Each image has seven bands relating to specific weather measurements (see Fig. A3.6.1). Now that both datasets are loaded, we will associate the cougar occurrences data with the weather data.

Section 2.3. Extracting Values

With our points and imagery loaded, we can call a function to extract values from the underlying raster based on the known locations of the cougar. We will do this using the `ee.Image.sampleRegions` function. Search for the `ee.Image.sampleRegions` function under the **Docs** tab to familiarize yourself with the parameters it requires.

If we tried to call this function on the Daymet `ImageCollection` we would get an error, because `ee.Image.sampleRegions` is a function of an image. To get around this, we will convert the Daymet `ImageCollection` into a multiband image (Fig. A3.6.4). Each of the seven measurements for each day will become a specific band in our multiband image. This process will help us in the end, because each band is defined by the date it was

collected and the variable it shows. We can use this information to determine which data connects to the positions of the cougar on a specific day.

It is important to note that, with many images in the `ImageCollection`, we are going to create a single image with a large number of bands. Because Earth Engine is good at data manipulations, it can handle this type of request.

```
// Convert to a multiband image.
var DaymetImage = Daymet.toBands();

print(DaymetImage, 'DaymetImage');
```

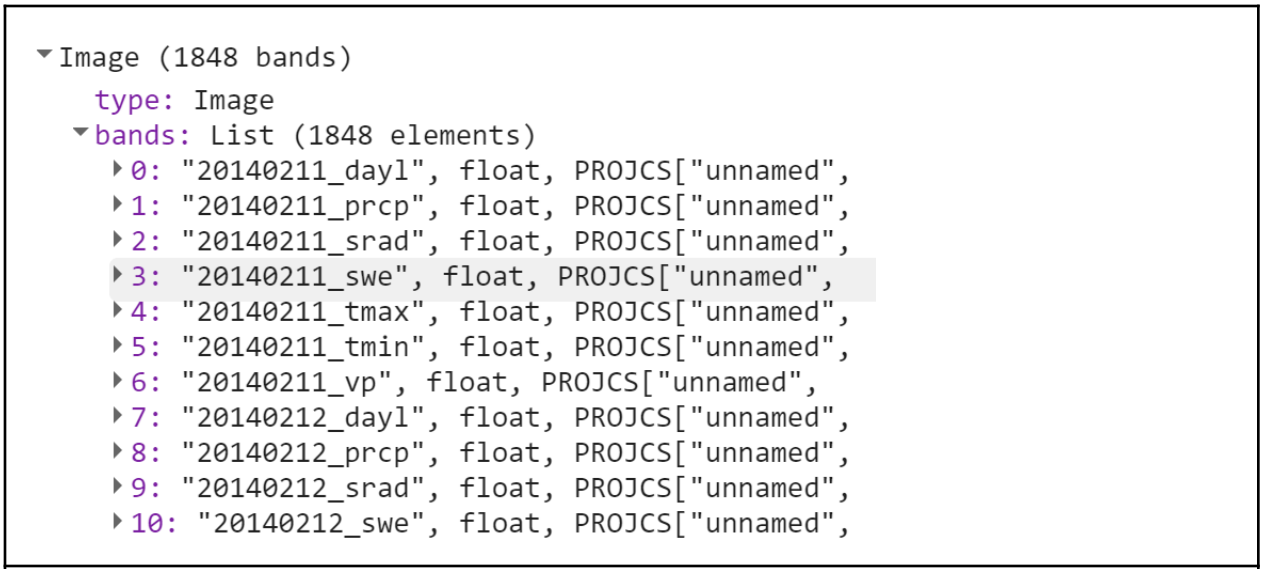


Fig. A3.6.4 A print statement showing the result of converting the Daymet `ImageCollection` into a multiband image

Now that we have a single multiband image, we can use the `sampleRegions` function to extract information across multiple dates at each of our GPS point locations (Fig. A3.6.5). There are three parameters of this function that should be considered:

- **Collection** This is the vector dataset that the sampled data will be associated with.
- **Properties** This defines which columns of the vector dataset will remain. In this case, we want to keep the ID column, because that is what we will use to join this dataset back to the original outside of Earth Engine.

- **Scale** This refers to the spatial resolution of the dataset. The scale parameter should always match the spatial resolution of your raster data. If you are not sure what the resolution of a raster is, you can search for the dataset using the search bar and locate that information in the documentation. (If you want to get the number for use in code, the `nominalScale` function, as described in Chap. F1.3, can extract it from an image.)

```
// Call the sample regions function.
var samples = DaymetImage.sampleRegions({
    collection: cougarF53,
    properties: ['id'],
    scale: 1000
});

print(samples, 'samples');
```

```
▼FeatureCollection (1361 elements, 0 columns)
   type: FeatureCollection
   columns: Object (0 properties)
  ▼features: List (1361 elements)
    ▼0: Feature 000000000000000002ef_0
       type: Feature
       id: 000000000000000002ef_0
       geometry: null
      ▼properties: Object (1849 properties)
          id: 1902296798
          20140211_dayl: 37471.94140625
          20140211_prcp: 0
          20140211_srad: 398.3299865722656
          20140211_swe: 40.790000915527344
          20140211_tmax: 4.760000228881836
          20140211_tmin: -9.079999923706055
          20140211_vp: 307.3999938964844
          20140212_dayl: 37604.12109375
          20140212_prcp: 0
          20140212_srad: 425.2900085449219
          20140212_swe: 40.790000915527344
          20140212_tmax: 8.430000305175781
```

Fig. A3.6.5 The print statement from the `sampleRegions` function showing that our GPS point locations now have weather measurements associated with them

Code Checkpoint A36a. The book's repository contains a script that shows what your code should look like at this point.

Section 2.4. Exporting

Exporting Points

We now have a series of daily weather data associated with the known locations of the cougar known as F53. While we could work more with these data in Earth Engine, it will be easy to bring them into R, Python, or Excel for further analysis. There are a few options to define where the exported data will be created. Generally speaking, saving the data to a Google Drive account provides an easy way to access the data with

another program. Here, we will use a dictionary, denoted by the curly brackets `{}`, to define the parameters of the `Export.table.toDrive` function.

We would have preferred to export to create a shapefile, but a shapefile can contain only 255 columns. The `samples` variable has 1361 columns, so we will export the data as a CSV file.

```
// Export value added data to your Google Drive.
Export.table.toDrive({
    collection: samples,
    description: 'cougarDaymetToDriveExample',
    fileFormat: 'csv'
});
```

When you export something, the **Tasks** panel will light up. To run the task, click the **Run** button (Fig. A3.6.6).

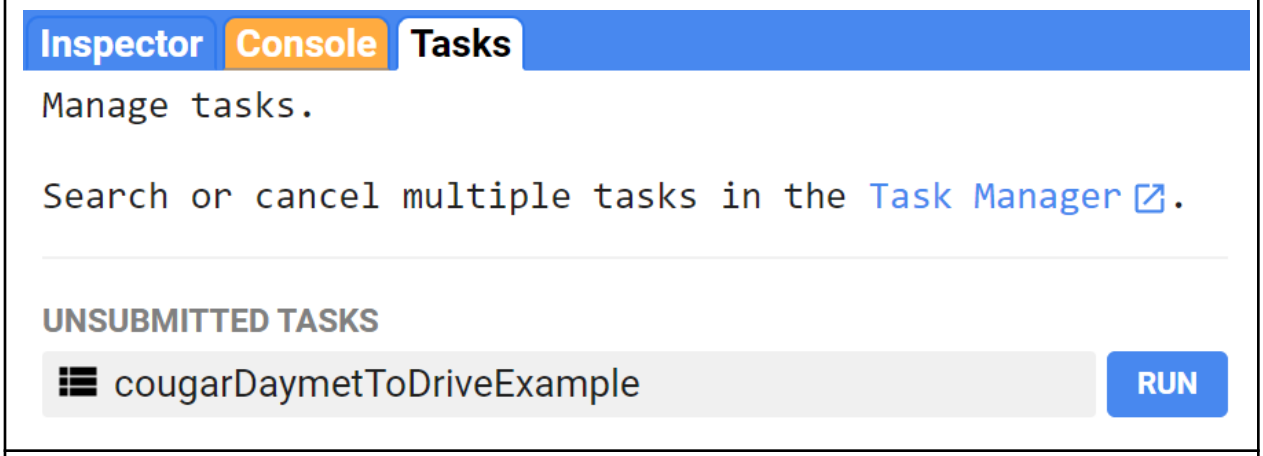


Fig. A3.6.6 An example of the **Tasks** bar once a script with an export function has been run

When you click the **Run** button, the pop-up shown in Fig. A3.6.7 will appear. This allows you to edit the details of the export.

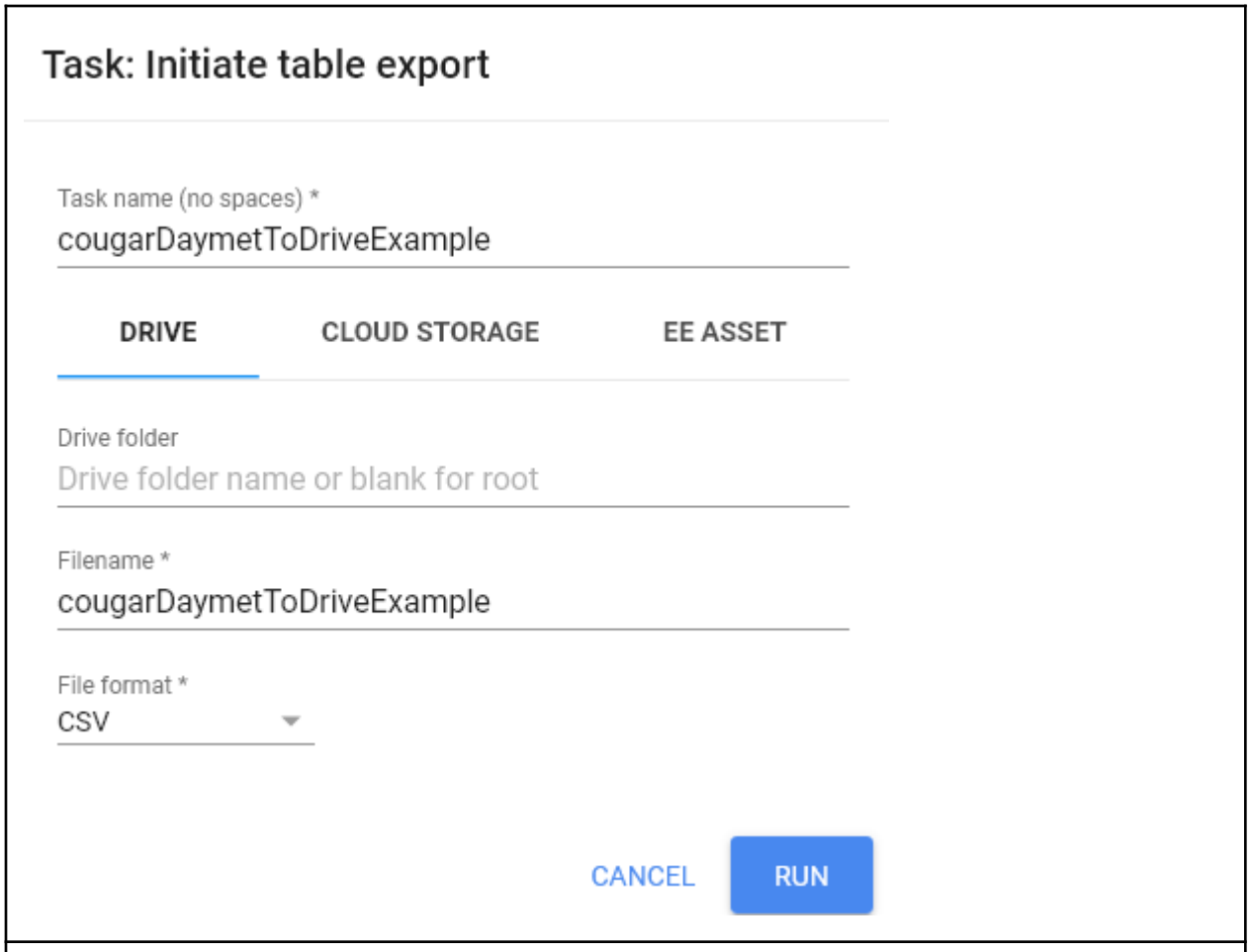


Fig. A3.6.7 An example of the user-defined parameters present when exporting a feature from Earth Engine

Exporting a Raster

While working with these spatial data, you may have realized that a raster showing the median values over the time period when data were collected on the cougar could be useful information to have. To do this, we will apply a `median` reducer function to the Daymet `ImageCollection` to generate a median value for each parameter in each cell. As with the tabular data, we will export this multiband image to Google Drive. Once we convert the `ImageCollection` to an image using the `median` function, we can clip it to the geometry feature object. This feature will be exported as a multiband raster.

```
// Apply a median reducer to the dataset.
var daymet1 = Daymet
```

```
    .median()
    .clip(geometry);

print(daymet1);

// Export the image to drive.
Export.image.toDrive({
    image: daymet1,
    description: 'MedianValueForStudyArea',
    scale: 1000,
    region: geometry,
    maxPixels: 1e9
});
```

In Earth Engine, there are many options for exporting images. One of the most important options when exporting data is the `maxPixels` parameter. Generally speaking, Earth Engine will not allow you to export a raster with more than 10^9 pixels. With the `maxPixels` parameter, you can bump this up to around 10^{12} pixels per image. If you are exporting data for an area larger than 10^{12} pixels, you will need to be creative about how to get the information out of Earth Engine. Sometimes this involves splitting the image into smaller pieces (as described in Chap. F6.2), or even reevaluating the usefulness of such a large image outside of Earth Engine.

Code Checkpoint A36b. The book's repository contains a script that shows what your code should look like at this point.

Question 1. With the `maxPixels` count set to 1e9, what square area would you expect to be able to export for imagery with 1000, 250, 30, 10 m cell size? Determine how this area measure changes when using 10^{12} pixels for one of the cell sizes to determine the percent increase.

Question 2. We selected all available variables from the Daymet collection. What specific bands from Daymet might be most relevant to cougars? How would you edit the code to select just one or a few bands?

Question 3. When we extracted the raster data, we set the scale parameter to 1000 to match the known spatial resolution of the Daymet dataset. Would it be reasonable to

change that scale parameter to 500 to get final resolution data? What does Earth Engine tell you when you attempt to change the scale of an export?

Synthesis

Assignment 1. Utilize the chapter's script with a different dataset to process tracking information from a different animal from Movebank or another occurrence database of your choice (e.g., GBIF.org). Evaluate whether the temperature ranges between the two animals are different and make a hypothesis about why this might be.

Conclusion

While Google Earth Engine can be used for planetary-scale analyses, it is also an effective resource for quickly accessing and analyzing large amounts of information across time at a scale relative to your own data. The method presented in this chapter is a great way to add value to your own research. Here, we worked with weather data, but that is by no means the only option. You can connect your data to many other datasets within Earth Engine. It is up to you to explore and find what is relevant to your work.

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

Mahoney PJ, Young JK (2017) Uncovering behavioural states from animal activity and site fidelity patterns. Methods Ecol Evol 8:174–183. https://doi.org/10.1111/2041-210X.12658

Chapter A3.7: Creating Presence and Absence Points

Authors
Peder Engelstad, Daniel Carver, Nicholas E. Young

Overview
The purpose of this chapter is to demonstrate a method to generate your own presence and absence data and distribute those samples using specific ecological characteristics found in remotely sensed imagery. You will see that even when field data is unavailable, you can still digitally sample a landscape and gather information on current or past ecological conditions.

Learning Outcomes
- Generating presence and absence data manually using high-resolution imagery.
- Generating randomly distributed points automatically within a feature class layer to use as field sampling locations.
- Filtering your points to refine your sampling locations.

Helps if you know how to:
- Import images and image collections, filter, and visualize (Part F1).
- Visualize images with a variety of false-color band combinations (Chap. F1.1).
- Perform basic image analysis: select bands, compute indices, create masks (Part F2).
- Use `normalizedDifference` to calculate vegetation indices (Chap. F2.0).
- Use drawing tools to create points, lines, and polygons (Chap. F2.1).
- Filter a `FeatureCollection` to obtain a subset (Chap. F5.0, Chap. F5.1).

Introduction to Theory

Herbivore grazing can have a negative effect on aspen regeneration in some areas, as aspens tend to grow in large monotypic stands (Halofsky and Ripple 2008). Excluding elk, deer, and cow grazing from an area has observable effects on aspen regrowth. In a hypothetical study, we may be interested in understanding the effect of herbivory from these species on various ecological aspects of aspen stands. But how can we monitor aspen stands without setting foot in the field? In this chapter, we will use multiple datasets and high-resolution resolution imagery (1 m^2) to generate sampling locations for this theoretical field survey. We will also build a presence/absence dataset that could be used to train a spatial model of aspen coverage.

Practicum

The National Land Cover Database (NLCD) is a Landsat-derived land cover database with 30 m spatial resolution, available at multiple time periods. Data from 1992 are primarily based on an unsupervised classification of Landsat data. The other years rely on a decision-tree classification to identify the land cover classes.

The National Agriculture Imagery Program (NAIP) acquires aerial imagery during the agricultural growing seasons across the continental United States. NAIP projects are contracted each year based upon available funding and the Farm Service Agency imagery acquisition cycle. NAIP imagery is acquired at a one-meter ground sample distance with a horizontal accuracy that matches within six meters of photo-identifiable ground control points. NAIP imagery is often collected in four bands: blue, green, red, and near infrared. The near infrared band is helpful in distinguishing between different types of vegetation.

The USGS National Elevation Dataset (NED) 1/3 arc-second is an elevation dataset produced by the USGS. This coast-to-coast dataset is available at 0.33 arc-second spatial resolution (30 m) across the lower 48 states, parts of Alaska, Hawaii, and US Territories.

Section 1. Developing Your Own Sampling Locations

We will start by developing potential field sampling locations based on the relative physical and ecological conditions.

Section 1.1. Region of Interest

The geographic region for this chapter is the Grand Mesa in western Colorado. Considered to be the largest mesa in the world by area, the Grand Mesa rises from 4000 feet near Grand Junction, Colorado, to over 10000 feet at its highest point. Historically, the Grand Mesa was glaciated, leaving its top flattened and spotted with lakes. While heavily forested at higher elevation, there are distinct transitions between shrubland, aspen, and conifer forest along the steep slopes. As one of the westernmost extensions of the montane environment in Colorado, the Grand Mesa represents important habitat for many ungulate species.

Our first step is to open a new script in Earth Engine. First, create a region of interest that encompasses the Grand Mesa (you can search for it by name in the search bar at the top). Do so using the geometry tool. Once you create the feature, rename it `roi` (Fig. A3.7.1).

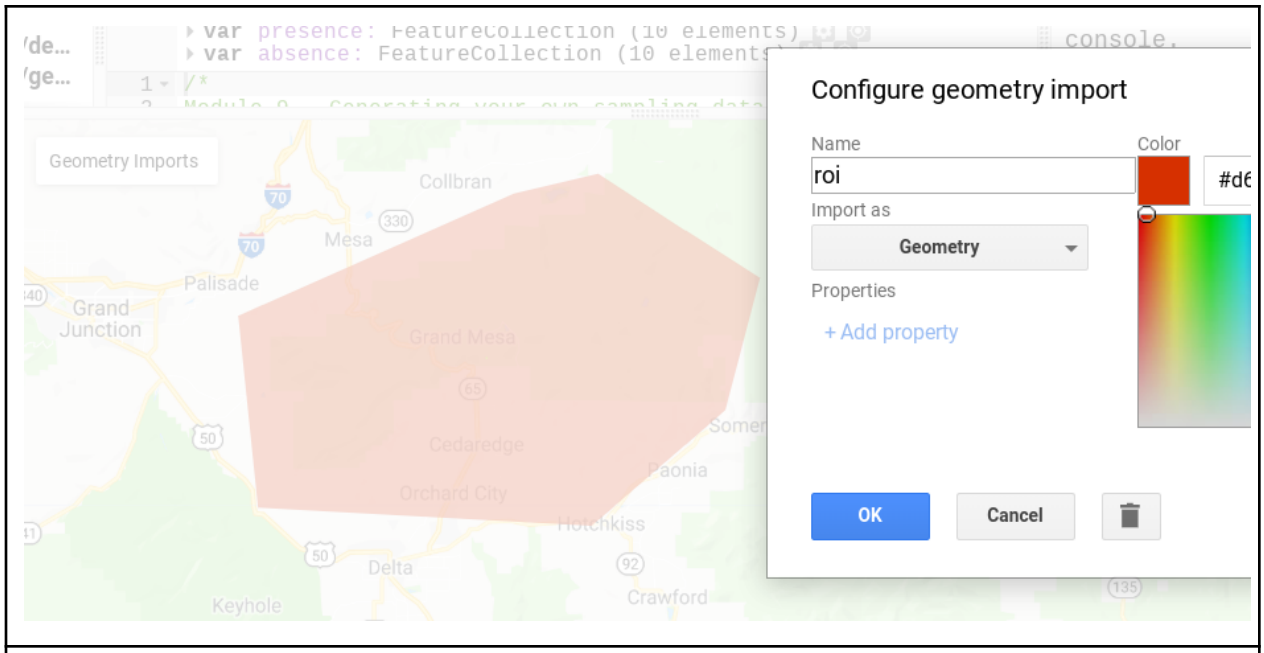


Fig. A3.7.1 An example of the general region of interest to use for this module

Section 1.2. Working with NAIP

We will rely on NAIP imagery for multiple steps in this process. NAIP imagery is not collected every year, so it makes sense to load in multiple years to determine what time

frame is available. We will use a print statement to see what years are available for our area from 2015 and 2017. Copy the following code to start your script.

```
// Call in NAIP imagery as an image collection.
var naip = ee.ImageCollection('USDA/NAIP/DOQQ')
    .filterBounds(roi)
    .filterDate('2015-01-01', '2017-12-31');

Map.centerObject(naip);

print(naip);
```

When you run the script you will see a `print` statement similar to the one shown in Fig. A3.7.2.

```
▾ ImageCollection USDA/NAIP/DOQQ (16 elements)
    type: ImageCollection
    id: USDA/NAIP/DOQQ
    version: 1646157264379446
    bands: []
  ▾ features: List (16 elements)
    ▾ 0: Image USDA/NAIP/DOQQ/m_3810701_ne_13_1_20150910 (4 bands)
        type: Image
        id: USDA/NAIP/DOQQ/m_3810701_ne_13_1_20150910
        version: 1494276705943000
      ▸ bands: List (4 elements)
      ▸ properties: Object (5 properties)
    ▾ 1: Image USDA/NAIP/DOQQ/m_3810701_ne_13_1_20171024 (4 bands)
        type: Image
        id: USDA/NAIP/DOQQ/m_3810701_ne_13_1_20171024
        version: 1526753219896697
      ▸ bands: List (4 elements)
      ▸ properties: Object (5 properties)
    ▸ 2: Image USDA/NAIP/DOQQ/m_3810701_nw_13_1_20150910 (4 bands)
    ▸ 3: Image USDA/NAIP/DOQQ/m_3810701_nw_13_1_20170826 (4 bands)
    ▸ 4: Image USDA/NAIP/DOQQ/m_3810702_nw_13_1_20150910 (4 bands)
    ▸ 5: Image USDA/NAIP/DOQQ/m_3810702_nw_13_1_20171008 (4 bands)
```

Fig. A3.7.2 A print statement identifying the years of available imagery for this region

of interest

The `print` statement shows us that imagery is available for both 2015 and 2017, yet it is difficult to determine the extent of coverage present without visualizing the data. For now, we will add both collections to the map to see what the image availability looks like. Add the following code to your existing script.

```
// Filter the data based on date.
var naip2017 = naip
    .filterDate('2017-01-01', '2017-12-31');

var naip2015 = naip
    .filterDate('2015-01-01', '2015-12-31');

// Define viewing parameters for multi band images.
var visParamsFalse = {
    bands: ['N', 'R', 'G']
};
var visParamsTrue = {
    bands: ['R', 'G', 'B']
};

// Add both sets of NAIP imagery to the map to compare coverage.
Map.addLayer(naip2015, visParamsTrue, '2015_true', false);
Map.addLayer(naip2017, visParamsTrue, '2017_true', false);
```

Fig. A3.7.3 NAIP imagery from 2015 has captured the aspen forest across the extent of our study area

Compared to the 2015 imagery (Fig. A3.7.3), the 2017 imagery (Fig. A3.7.4) has been captured later in the season and we can start to see some yellow in the aspen. A challenge with this image is that there were some distinct time lags between neighboring flight paths during the initial image collection. Notice the vertical band of green in the 2017 image. While it is usually best practice to use images that closely match the time that you will be in the field, in this case the consistency of the 2015 imagery outweighs those temporal concerns.

Fig. A3.7.4 NAIP imagery from 2017 has captured the yellowing of the aspen forest but shows a distinct difference in vegetation cover due to variability in the time at which the given images were captured

We will also add a false color image to the 2015 data because this band combination visualization is very helpful for distinguishing between deciduous and coniferous forests. We do this by adding the `naip2015` object with a different set of visualization parameters to the map. Add the following code to your existing script.

```
// Add 2015 false color imagery.
Map.addLayer(naip2015, visParamsFalse, '2015_false', false);
```

Section 1.3. Aspen Exclosures

In our hypothetical study, let us imagine land managers have established some aspen exclosures on the southern extent of the Grand Mesa. Let us also say that the land managers did not have specific shapefiles of the exclosures but did have GPS locations of the four corners. We will use these data to add the exclosures to the map by creating

a geometry feature within our script. In this case we are creating an `ee.Geometry.MultiPolygon` feature. Add the following code to your existing script. If your `roi` object does not encompass the exclosures, edit its bounds or redraw it.

```
// Creating a geometry feature.
var exclosure = ee.Geometry.MultiPolygon([
    [
        [-107.91079184, 39.012553345],
        [-107.90828129, 39.012553345],
        [-107.90828129, 39.014070552],
        [-107.91079184, 39.014070552],
        [-107.91079184, 39.012553345]
    ],
    [
        [-107.9512176, 39.00870162],
        [-107.9496834, 39.00870162],
        [-107.9496834, 39.00950196],
        [-107.95121765, 39.00950196],
        [-107.95121765, 39.00870162]
    ]
]);

print(exclosure);

Map.addLayer(exclosure, {}, 'exclosures');
```

The structure of the `ee.Geometry.MultiPolygon` feature is a bit complex, but it is effectively a set of nested lists. There are three tiers of lists present:

```
// Nested list example.
['tier 1' ['tier 2' ['tier 3']]]
```

- **Tier 1**: A single list that holds all the data. The `ee.Geometry.MultiPolygon` function requires the input to be a list.
- **Tier 2**: A list for each polygon, containing a unique set of coordinates.
- **Tier 3**: A list for each x,y coordinate pair. Each polygon is composed of a series of x,y points with a point that overlaps exactly with the first coordinate pair. This last coordinate pair is the same as the first and is essential to completing the feature.

If you are trying to make a geometry feature and are having trouble, you can always create one with the draw shape tool and look at the values in a print statement as a template (Fig. A3.7.5).

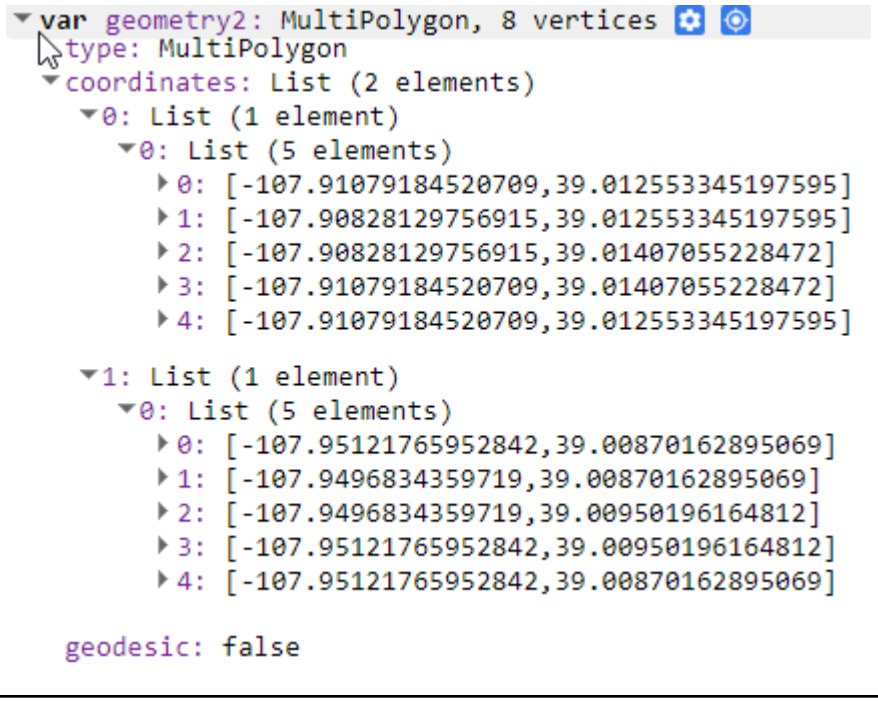


Fig. A3.7.5 A view of the exclosure object coordinate details from the print statement

Section 1.4. Determining Similar Areas for Sampling

Now that we have our aspen exclosures loaded, we are going to bring in some additional layers to help quantify the landscape characteristics of the exclosures. We will use those values to find similar areas nearby to use as sampling sites outside of the exclosures. By keeping the environmental conditions similar, we can make stronger statements about the comparative effects of herbivory on aspen stands.

We will use three datasets to describe the conditions within the sampled areas:

1. NED**:** Select areas in a similar elevation range. Elevation is correlated with many environmental conditions, so we are using it as a proxy for features such as temperature, precipitation, and solar radiation.
2. NAIP**:** Calculate an NDVI index to get a measure of vegetation productivity.

3. NLCD**:** Select the deciduous forest class as a way to limit the location of sampling sites.

Section 1.5. Loading in the Data

First, we will call the NED by its unique ID. We can gather these details by searching for the feature and reading through the metadata. Add the following code to your existing script.

```
// Load in elevation dataset; clip it to general area.
var elev = ee.Image('USGS/NED')
    .clip(roi);

Map.addLayer(elev, {
    min: 1500,
    max: 3300
}, 'elevation', false);
```

We already have NAIP imagery loaded but we need to convert it to an image and calculate NDVI. We have already filtered our NAIP imagery to a single year, so there is only one image per area. We could apply a reducer to convert the `ImageCollection` to an image, but a reducer is not necessary for a single layer. A more logical option for this is to apply the `mosaic` function to convert the `ImageCollection` to an image. Add the following code to your existing script.

```
// Apply mosaic, clip, then calculate NDVI.
var ndvi = naip2015
    .mosaic()
    .clip(roi)
    .normalizedDifference(['N', 'R'])
    .rename('ndvi');

Map.addLayer(ndvi, {
    min: -0.8,
    max: 0.8
}, 'NDVI', false);

print(ndvi, 'ndvi');
```

The last dataset we are bringing in is the NLCD. Add the following code to your existing script.

```
// Add National Land Cover Database (NLCD).
var dataset = ee.ImageCollection('USGS/NLCD');

print(dataset, 'NLCD');
```

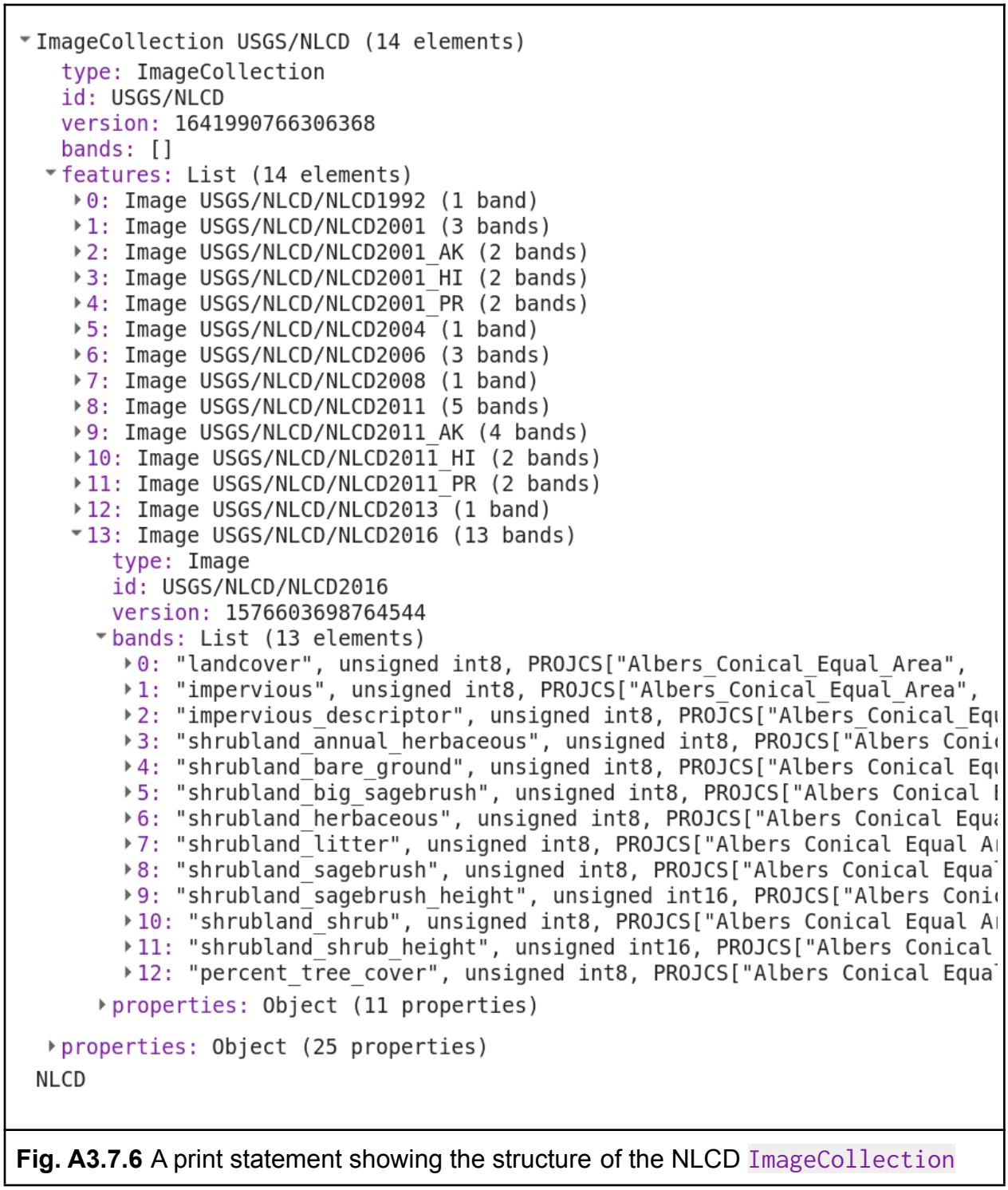


Fig. A3.7.6 A print statement showing the structure of the NLCD `ImageCollection`

From the print statement (Fig. A3.7.6), we can see that some of the images have a band called `'landcover'`. Rather than trying to pull it out from the feature collection, we will call the 2016 NLCD image directly by its unique ID and select the `'landcover'` band.

```
// Load the selected NLCD image.
var landcover = ee.Image('USGS/NLCD/NLCD2016')
    .select('landcover')
    .clip(roi);

Map.addLayer(landcover, {}, 'Landcover', false);
```

Question 1. Investigate the other available years in the NLCD collection. What parts of the study area have land cover classes that change? What might be driving that change?

Section 1.6. Generating Random Points

With our three datasets loaded we can now generate a series of potential survey sites. We will do this by generating random points within a given area (Fig. A3.7.7). We want these sites to be accessible, near the two exclosures, and within the public land boundary. We will create another geometry feature that we will use to contain the randomly generated points. Hover over the **Geometry imports** box and click **+new layer**. Be sure to name this second geometry feature `sampleArea`.

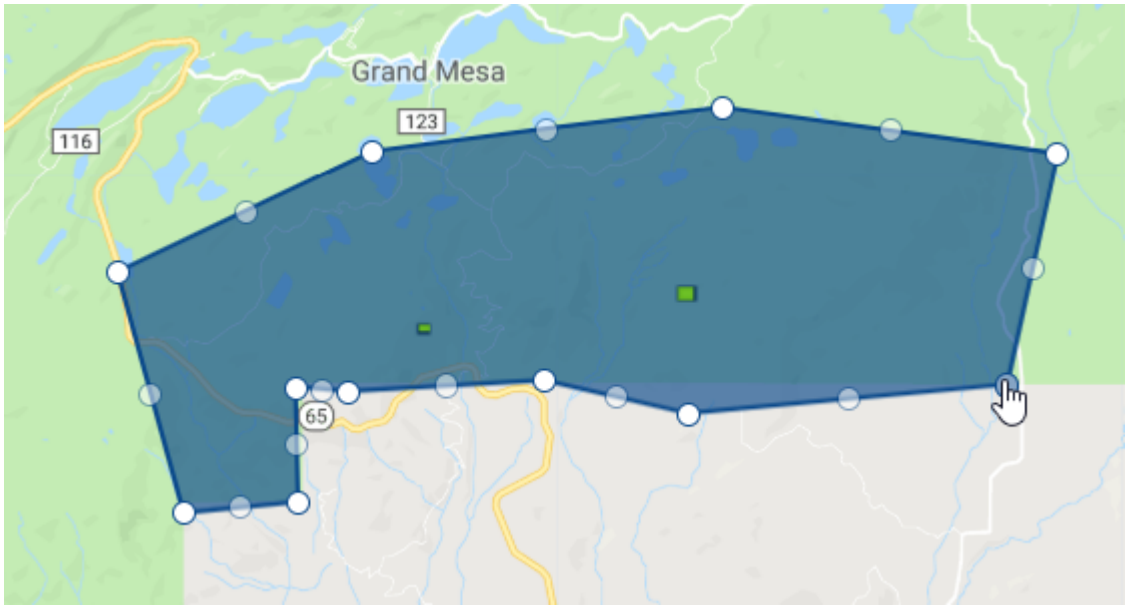


Fig. A3.7.7 An example of what your second sample area geometry feature might look like

With the geometry feature in place, we can create points using the `randomPoints` function. Add the following code to your existing script.

```
// Generate random points within the sample area.
var points = ee.FeatureCollection.randomPoints({
    region: sampleArea,
    points: 1000,
    seed: 1234
});

print(points, 'points');

Map.addLayer(points, {}, 'Points', false);
```

Here, we are using a dictionary to define the parameters of the function. The `region` parameter is the area in which the points are created. In our case, we are going to set this to `sampleArea` polygon. The `points` parameter specifies the total number of points to generate. The `seed` parameter is used to indicate specific random values. Think of this as a unique ID for a set of random values. The seed number (1234 in this example)

refers to an existing random list of values. Setting the seed is very helpful because you are still using random values, but the code will place the points in the same locations every time until the seed is changed.

Section 1.7. Extracting Values to Points

To associate the landscape characteristics with the point locations, we are going to call the `ee.Image.sampleRegions` function. This function requires a single image. Rather than calling three times on our three unique images (elevation, NDVI, and NLCD), we are going to add all those images together to create a multiband image so we can call this function a single time. Add the following code to your existing script.

```
// Add bands of elevation and NAIP.
var ndviElev = ndvi
    .addBands(elev)
    .addBands(landcover);

print(ndviElev, 'Multi band image');
```

With the multiband image set, we will call the `sampleRegions` function. Add the following code to your existing script.

```
// Extract values to points.
var samples = ndviElev.sampleRegions({
    collection: points,
    scale: 30,
    geometries: true
});

print(samples, 'samples');

Map.addLayer(samples, {}, 'samples', false);
```

Within this function, the `collection` parameter is set to the feature collection where the extracted values will be added. In this example it is a point dataset. The `scale` parameter refers to the spatial resolution (pixel size) of the data. The `geometries` parameter indicates whether or not you want to maintain the x,y coordinate pairs for each element in the collection. The default is false, but we set it to true here because we

eventually want to show these points on the map, as they will represent suitable sampling site locations.

Segue on Scale

The idea of *scale* is very important in remote sensing, but you will often encounter multiple definitions of it. *Map scale*, the relationship between a measured distance on a map and the actual distance on the landscape, is the most common in remote sensing. The image *extent* can also be referred to as the *scale of the image* or the *spatial scale*. Confusing, right?

In the case of the `scale` argument in the function above, it is referring to yet another definition of that word—the pixel size of the image. In our example, the multiband image has two bands with a pixel size of 30 m, and one with a pixel size of 1 m. It is best practice to use the largest pixel size when working with data having different spatial resolutions. Here, this means you are upscaling the 1 m image to 30 m. This means a potential loss of precision in your data. However, you can be confident that the number in that upscaled cell is representative of the mean value across all the cells. If you go in the opposite direction and downscale an image, you effectively make up the data to fill the gap. Because NAIP imagery has a 1 m pixel size, it contains 30 x 30 more pixels than a single 30 m pixel. Take a look at the examples below to help visualize these processes.

Upscaling takes the available data and finds the mean value.

| 3 | 7 | 8 |

| 4 | 2 | 2 |

| 1 | 3 | 2 |

Question 2. Calculate the mean for the matrix above. How would this value change your understanding of the pixel? Is it still representative of the overall "story" the data is telling, or would a different reducer (i.e., median) be more appropriate?

Downscaling takes a single value and places the value for all locations in the grid.

| 7 | 7 | 7 |

| 7 | 7 | 7 |

| 7 | 7 | 7 |

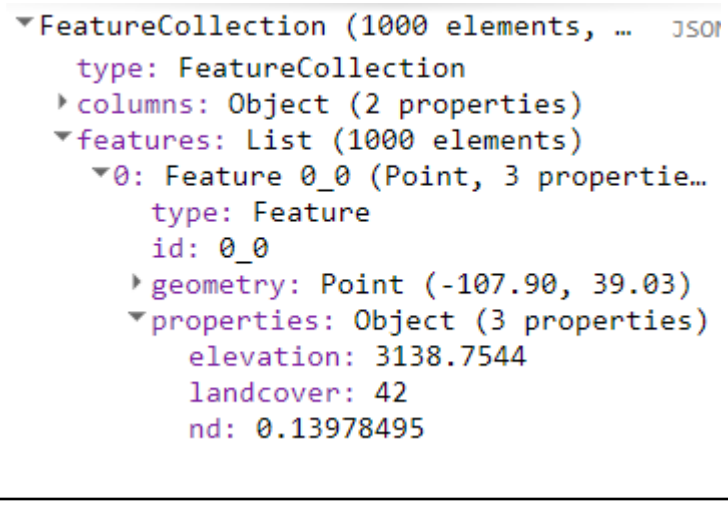


Fig. A3.7.8 The result of our sampling function has given us 1000 potential sites to choose from. We will limit this pool by comparing the measurable data we have at these sites to the mean values of those data from our exclosures.

From our print statement (Fig. A3.7.8) we can see that each of our 1000 point locations have three properties: elevation, land cover, and NDVI. We want to use these values to filter out sites that do not match the conditions of the exclosures. The land cover data are categorical, so it is easy to filter. We know from the metadata on NLCD that the land cover class for deciduous forest is a single value (41).

We will use the `filter` function to select all sites that are within aspen forests. Add the following code to your existing script.

```
// Filter metadata for sites in the NLCD deciduous forest layer.
var aspenSites = samples.filter(ee.Filter.equals('landcover', 41));

print(aspenSites, 'Sites');
```

From the print statement, we can see this reduces the total number of potential sites (1000) by about 25%, depending on your study area shape. However, to get down to roughly 10 potential new monitoring sites, there is still a lot of trimming to do.

Filtering based on elevation and NDVI is a bit trickier because both of these variables are continuous data. You want to find sites that have similar values to those in the exclosures, but they do not need to be exactly the same. For this example, we will say that if a value is within ±10% of the mean value found within the exclosure, we will group it as similar. There are a couple of things that need to be calculated before we can filter our potential sampling sites:

1. Mean values within exclosures.
2. 10% above and below the mean.

Question 3. The above use of ±10% is somewhat arbitrary. How might the range change given the inclusion of additional exclosures in our study area? What other methods could be used to generate this range?

We will work with the NDVI image first and then apply this process to the elevation dataset.

- Step 1: To find the mean value, we are going to apply a mean reducer over the area that is inside of the exclosure. Add the following code to your existing script.

```
// Set the NDVI range.
var ndvi1 = ndvi
    .reduceRegion({
        reducer: ee.Reducer.mean(),
        geometry: exclosure,
        scale: 1,
        crs: 'EPSG:4326'
    });

print(ndvi1, 'Mean NDVI');
```

The `reduceRegion` function takes in an image and returns a dictionary where the key is the name of the band, and the value is the output of the reducer.

- Step 2: Determine the acceptable variability around the mean.

We are relying on simple mathematical functions to find the ±10% values. Add the following code to your existing script.

```
// Generate a range of acceptable NDVI values.
var ndviNumber = ee.Number(ndvi1.get('ndvi'));
var ndviBuffer = ndviNumber.multiply(0.1);
var ndviRange = [
    ndviNumber.subtract(ndviBuffer),
    ndviNumber.add(ndviBuffer)
];

print(ndviRange, 'NDVI Range');
```

The first step in this process is all about understanding data types. The `ndvi1` object is a dictionary so we call the `get` function to pull a value based on a known key. We then convert that value to an `ee.Number` type object so we can apply the math functions. Our output is a list with the minimum and maximum values (±10% of the mean NDVI values).

Section 1.8. Create Your Own Function

We now have our range for NDVI, but we also need to apply the same process to elevation. In this case we are only applying this process twice, but it seems like a useful chunk of code that might be handy down the road. Rather than just copying and pasting the code again and again, we are going to create a function with flexible parameters so we can apply this useful bit of code efficiently.

A function has the following requirements: parameters, statements, and a return value. Here is an example of the structure of a function from the official Earth Engine documentation. The following pseudocode demonstrates the syntax and structure of a function in JavaScript. *Do not add this code to your script.*

```
var myFunction = function(parameter1, parameter2, parameter3) {
    statement;
    statement;
    var value = statement;
    return value;
};
```

If we apply this structure to our goal of reducing the elevation by area (like we did for NDVI with the code we created above), we would need to consider the following:

- Parameters : an image, a geometry object, and `pixelSize`.
- Statements: `reduceRegion` function.
- A return value: output of the `reduceRegion` function.

When using functions, it is important to use informative names within your parameters that give some indication of the data type that is required. If we want our function to be reproducible, we can provide some more information as a longer comment when we define the function. Add the following code to your existing script.

```
/*
This function is used to determine the mean value of an image within a
given area.
image: an image with a single band of ordinal or interval level data
geom: geometry feature that overlaps with the image
pixelSize: a number that defines the cell size of the image
Returns a dictionary with the median values of the band, the key is
the band name.
*/
var reduceRegionFunction = function(image, geom, pixelSize) {
    var dict = image.reduceRegion({
        reducer: ee.Reducer.mean(),
        geometry: geom,
        scale: pixelSize,
        crs: 'EPSG:4326'
    });
    return (dict);
};
```

Let's check our function definition by verifying that it gives the same answer for the NDVI range as when we did it step by step:

```
// Call function on the NDVI dataset to compare.
var ndvi_test = reduceRegionFunction(ndvi, exclosure, 1);

print(ndvi_test, 'ndvi_test');
```

This is a very clean method for coding when you need to apply a process multiple times.The function has been defined and now we can call it on the elevation dataset. Add the following code to your existing script.

```
// Call function on elevation dataset.
var elev1 = reduceRegionFunction(elev, exclosure, 30);

print(elev1, 'elev1');
```

We will define a second function to determine ±10% around a mean value.

- Parameters: an image, band name, proportion.
- Statements: multiple steps.
- A return value: list.

Add the following code to your existing script.

```
/*
Generate a range of acceptable values.
dictionary: a dictionary object
key: key to the value of interest, must be a string
proportion: a percentile to define the range of the values around the
mean
Returns a list with a min and max value for the given range.
*/
var effectiveRange = function(dictionary, key, proportion) {
    var number = ee.Number(dictionary.get(key));
    var buffer = number.multiply(proportion);
    var range = [
        number.subtract(buffer),
        number.add(buffer)
    ];
    return (range);
};
```

We will call the `effectiveRange` function on the output of the `reduceRegionFunction` function. Add the following code to your existing script.

```
// Call function on elevation data.
var elevRange = effectiveRange(elev1, 'elevation', 0.1);

print(elevRange);
```

Now that we have an effective range for both our NDVI and elevation values, we can apply an additional set of filters to thin the list of potential sample sites. We can do this by chaining multiple `ee.Filter` calls together. Add the following code to your existing script.

```
// Apply multiple filters to get at potential locations.
var combinedFilter = ee.Filter.and(
    ee.Filter.greaterThan('ndvi', ndviRange[0]),
    ee.Filter.lessThan('ndvi', ndviRange[1]),
    ee.Filter.greaterThan('elevation', elevRange[0]),
    ee.Filter.lessThan('elevation', elevRange[1])
);

var aspenSites2 = aspenSites.filter(combinedFilter);

print(aspenSites2, 'aspenSites2');

Map.addLayer(aspenSites2, {}, 'aspenSites2', false);
```

Depending on how you have drawn your study area, this filtering process should reduce the 1000 original sites by about 90%. From the ~100 remaining sites, we can manually select 10 either by something we already know about the landscape features of our study area, or by using a more restrictive range in our function (e.g., ±5%). This approach to site selection can be a good first step in ensuring you are sampling similar conditions in the field.

Code Checkpoint A37a. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Generating Your Own Training Dataset

As you have been examining this landscape, you may have noticed some misclassifications within the NLCD land cover layer (e.g., forests in non-forested areas).

Some misclassifications are expected in any land cover dataset. While the NLCD is trained to produce classifications of specific land cover assemblages across the United States, the aspen forest class that we are examining is included within a much larger grouping ("Deciduous forest"). Also, the particular NLCD image we selected shows land cover as it was detected in 2011. While forests are fairly stable over time, we can expect that some level of change has occurred. This might get you thinking about the possibility of generating your own aspen land cover product based on a remote sensing model for this specific region. There is a lot that goes into this process that we are not going to cover here. However, we are going to take the first step, which is generating our own presence/absence training dataset.

Section 2.1. Ocular Sampling

Generating your own training data relies on the assumption that you can confidently identify your species of interest using high-resolution imagery. We are using NAIP for this process because it is freely available and has a known collection date, allowing us to mark areas of aspen forest as "presence" and areas without aspens as "absence." These data could then be used later to train a model of aspen occurrence on the landscape.

Section 2.2. Adding Presence and Absence Points

First, we will need to create specific layers that will hold our new sampling points. Adding in presence and absence layers is a straightforward process accomplished by manually creating and placing geometry features on representative locations on the map. Hover over the **Geometry imports** box and click **+new layer (Fig. A3.7.9)**. A geometry feature named "geometry" will be added. Select the gear icon next to **geometry** and a pop-up will open. Change the **Import as** type to **FeatureCollection**, then press the **+Add property** button. Fill in the **Properties** values with "presence" and "1" and press **OK** to save your feature.

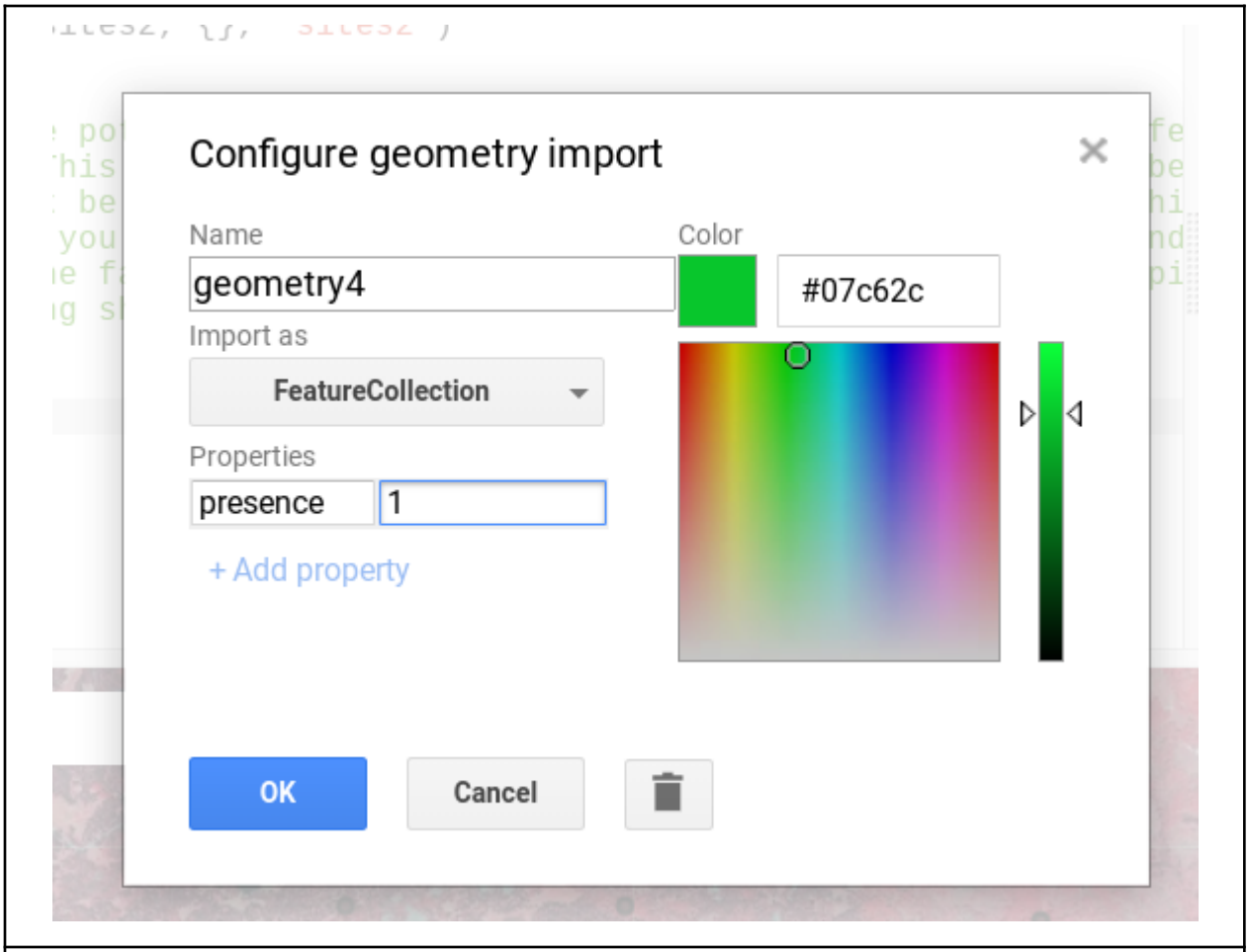


Fig. A3.7.9 An example of changing the parameters for the creation of the presence geometry feature

Change the feature collection name to `presence` and select a color you enjoy. Repeat this process to create an `absence` feature collection where the added property values are presence and "0". We will use the binary value in the presence column of both datasets to define what that location is referring to: 1 = Yes, this is aspen; 0 = No, this is not aspen.

Once the feature collections are created, we select the specific feature collection (`presence` or `absence`) and use the marker tool to drop points on the imagery. The sampling methodology you use will depend on your study. In this example, green presence points represent aspen forest, and blue points are not aspen (absence).

Use the 2015 false-color imagery in combination with the NDVI or NLCD to distinguish aspen from other land cover types. Aspen stands are brighter red than most other

vegetation types and tend to have a more complex texture than herbaceous vegetation in the imagery. Drop some points in what you perceive to be aspen forest (Fig. A3.7.10).

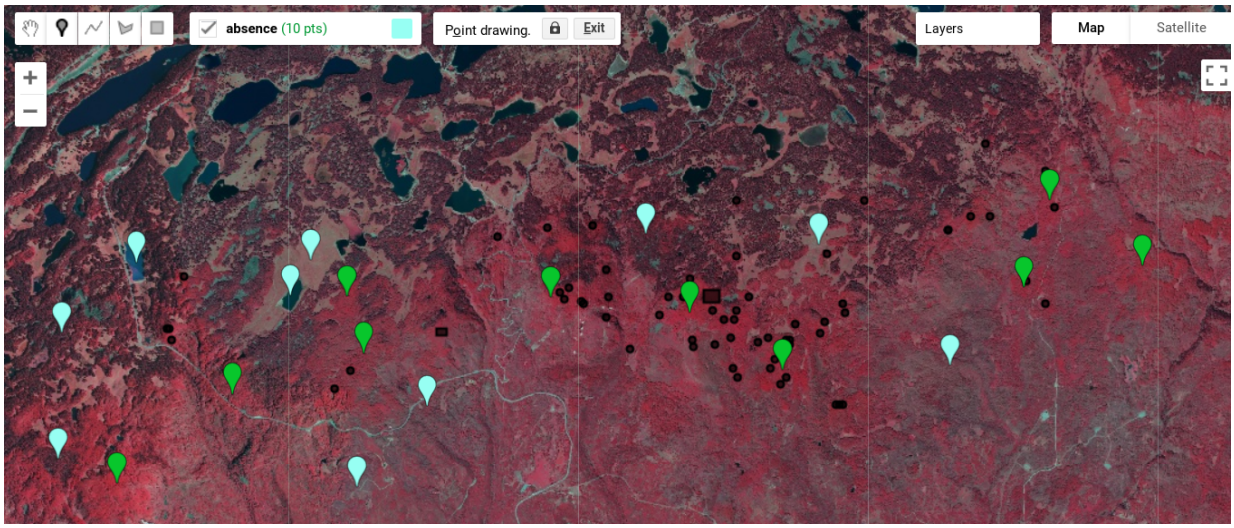


Fig. A3.7.10 Examples of presence and absence locations on the NAIP imagery created using the marker tool. Do your best to select locations that look correct to you.

Feel free to sample as many locations as you would like. Again, the quality of this data will depend on your ability to differentiate the multiple land cover classes present.

Section 2.3. Exporting Points

Currently our point locations are stored in two different features classes. We will merge these features into one feature class before exporting the data. We can merge the layers straightforwardly because they share the same data type (point geometry feature) and the same attribute data ("presence" with a numeric data value). Add the following code to your existing script.

```
// Merge presence and absence datasets.
var samples = presence.merge(absence);

print(samples, 'Samples');
```

Now that the sampling features classes are merged, we will export the features to our Google Drive. When you run the code below, the **Tasks** tab in the upper right-hand panel will light up. Earth Engine does not run tasks without you directing it to execute the

task from the **Tasks** tab. Add the following code to your existing script and run it to export your completed dataset.

```
Export.table.toDrive({
    collection: samples,
    description: 'presenceAbsencePointsForForest',
    fileFormat: 'csv'
});
```

Code Checkpoint A37b. The book's repository contains a script that shows what your code should look like at this point.

Synthesis

Assignment 1. Compare samples of NDVI ranges for aspen presence versus absence. Are the ranges significantly different? Create samples using the code presented here but for a coniferous tree of your choice. How different are these values from the values of deciduous forest?

Conclusion

In this module, we identified aspen locations with similar environmental characteristics and generated our own sampling data from those locations. Both processes are simple in concept but can be somewhat complex to implement without access to all your data in a single place. In both cases, we are generating value-added products that are informed by remote sensing but are not inherently remote sensing processes. This ability to be creative regarding how you use remotely sensed data is part of the beauty of the Earth Engine platform.

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

Halofsky J, Ripple W (2008) Linkages between wolf presence and aspen recruitment in the Gallatin elk winter range of southwestern Montana, USA. Forestry 81:195–207. https://doi.org/10.1093/forestry/cpm044

Chapter A3.8: Detecting Land Cover Change in Rangelands

Authors
Ginger Allington, Natalie Kreitzer

Overview

The purpose of this chapter is to familiarize you with the unique challenges of detecting land cover change in arid rangeland systems, and to introduce you to an approach for classifying such change that provides us with a better understanding of these systems. You will learn how to extract meaningful data about changes in vegetation cover from satellite imagery, and how to create a classification based on trajectories over time.

Learning Outcomes

- Visualizing and explaining the challenges of utilizing established land cover data products in arid rangelands.
- Applying a temporal segmentation algorithm to a time series of information about vegetation productivity.
- Classifying pixels based on similarities in their temporal trajectories.
- Extracting and visualizing data on the new trajectory classes.
- Comparing trajectory classes to information from traditional land cover data.

Helps if you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Perform pixel-based supervised or unsupervised classification (Chap. F2.1).
- Use expressions to perform calculations on image bands (Chap. F3.1).
- Write a function and `map` it over an `ImageCollection` (Chap. F4.0).
- Interpret the outputs from the LandTrendr algorithm implementation in Earth Engine (Chap. F4.5).
- Write a function and `map` it over a `FeatureCollection` (Chap. F5.1, Chap. F5.2).
- Use the `require` function to load code from existing modules (Chap. F6.1).

Introduction to Theory

Arid and semi-arid rangelands cover approximately 41% of the global land surface (Asner et al. 2004) and provide livelihoods for 38% of the human population (Millennium Ecosystem Assessment 2005) and forage for three-quarters of the world's livestock (Derner et al. 2017). Rangelands are located in regions of the world that are experiencing some of the most rapid changes in climate (Huang et al. 2015, Melillo et al. 2014), which can affect ecosystem productivity and resilience (Briske 2017). Land conversion to agriculture (Lambin and Meyfroidt 2014), urbanization and development (Fan et al. 2016, Sleeter et al. 2013), and afforestation (Cao et al. 2011) are increasing in many rangeland regions globally. Uncertainties about socioeconomic and sociopolitical forces such as land tenure security (Campbell et al. 2005, Li et al. 2007, Liu et al. 2015, Reid et al. 2000), rural out-migrations and urbanization (Lang et al. 2013), and changes in human demography and dietary preferences (Alexandratos and Bruinsma 2012) limit our ability to predict the future sustainability of rangeland systems. In order to understand the causes and consequences of land cover change in rangelands, we must first classify and quantify the change.

There are several readily available datasets commonly used to assess changes in land cover over large regions. The three most prominent global inventories are the annual MODIS 12Q land cover product (500 m) (Friedl et al. 2010), the 300 m GlobCover data from the European Space Agency (ESA) (Arino et al. 2008, Bontemps et al. 2011), and the new 10 m WorldCover data, also from the ESA (Zanaga et al. 2020), the latter two of which are available for single nominal dates. National-level data products typically are available at finer spatial resolutions and tend to use more categories to classify the surface (e.g., NLCD in the US, Homer et al. 2015)-- though they are not available for all countries. While these global and national land cover data products have been useful for documenting a number of important surface phenomena such as forest loss, urbanization, and habitat conversion in a variety of ecosystems (Schneider et al. 2010, Presetele et al. 2016), they generally have poor accuracy in arid portions of the globe (Friedl et al. 2002, Ganguly et al. 2010, García-Mora et al. 2012).

Additionally, traditional methods for change detection are derived from categorical land cover classifications, which can identify change only when a pixel crosses a threshold to a new state (e.g., from Grassland to Barren), and therefore can identify change only after it has occurred, not when it begins. In systems with long time lags in vegetation response (such as has been shown for recovery in arid grasslands), this can create uncertainty about the efficacy of environmental policies and the response of landscapes to large-scale management changes (Fig. A3.8.1).

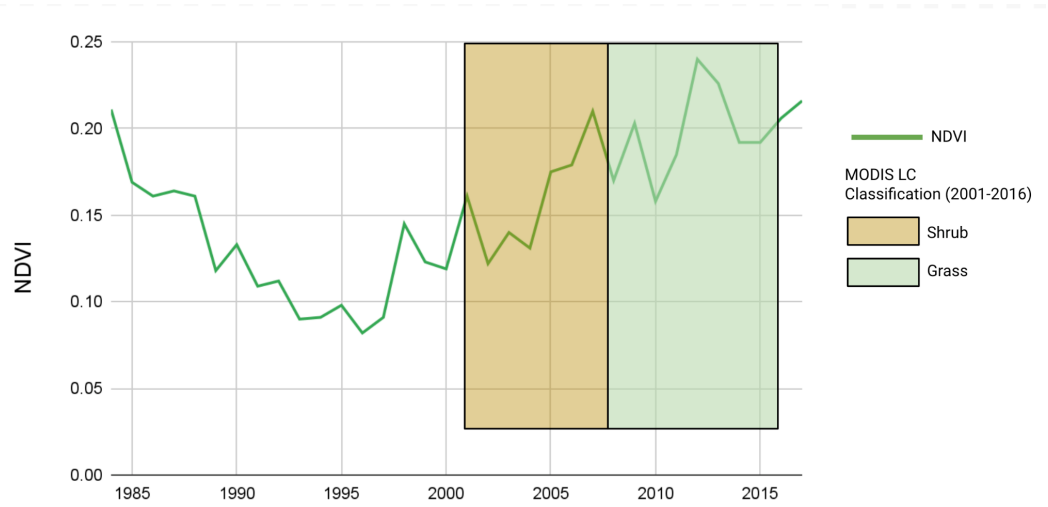


Fig. A3.8.1 Time series of NDVI in Naiman Banner, Inner Mongolia. Line represents the median value calculated over the region. Overlaid colored bars represent the corresponding land cover classes for the years 2001–2016, according to the MODIS-derived MCD12Q1 data.

As an alternative to classification-based change detection, new methods have emerged that identify unique features in time series of remotely sensed data to pinpoint disturbance events such as logging, wildfires, and flooding. These time series-based methods, such as CCDC (see Chap. F4.7) and LandTrendr (see Chap. F4.5), are typically calibrated and executed to detect abrupt changes in spectral reflectance or derived index values associated with those disturbances. This is extremely useful for detecting change events like deforestation or the defoliation resulting from an insect outbreak where the disturbance is punctuated in time and where the difference in spectral index is significant, and regreening occurs on the order of years (Fig. A3.8.2).

For these reasons, contemporary classification approaches are extremely limited in their ability to provide reliable information about landscape change and dynamics in rangeland systems. Current pressures from climate change, land use intensification, and urban expansion create an urgent need for methods to more accurately map and track land cover status in a way that is useful for arid-systems research, land use change detection, and the monitoring and management of rangelands.

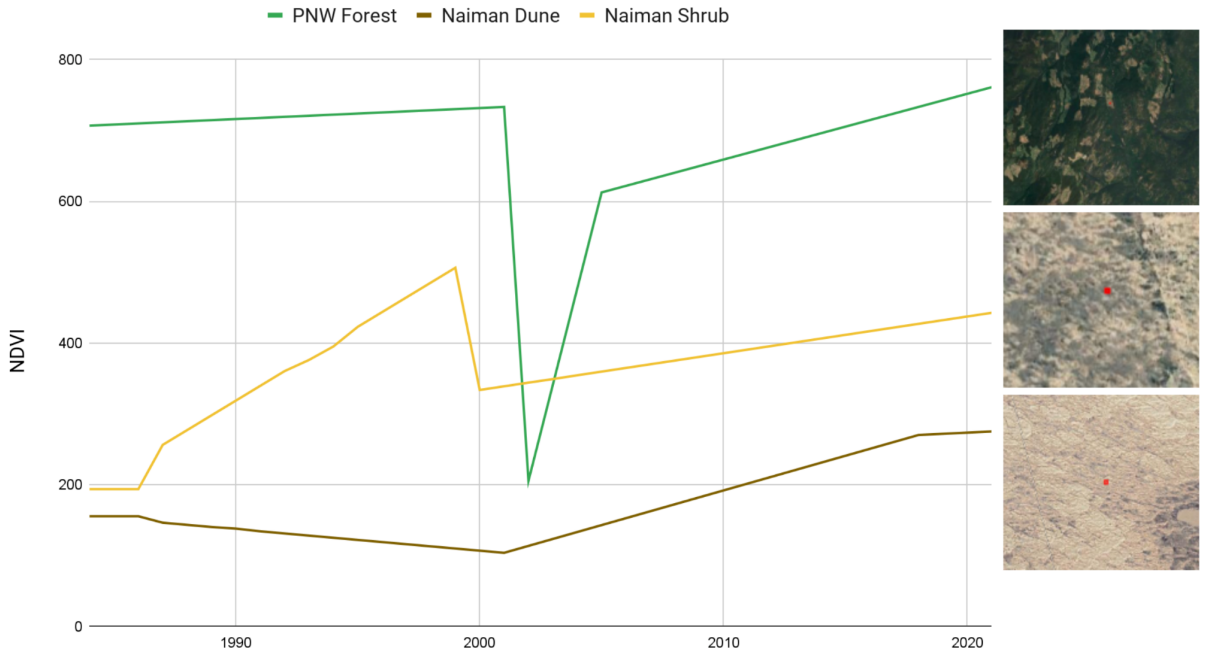


Fig. A3.8.2 Comparison of predicted NDVI for representative pixels from three regions: a forested area in southern British Columbia and two regions in Naiman Banner, Inner Mongolia, one dominated by woody shrubs and perennial forbs, the other by annual grasses and forbs

Forest land cover changes are stark in terms of the change in indices such as NDVI and NBR (Kennedy et al. 2016). A forest pixel might suddenly drop from an NDVI over 0.6 to 0.15, which is easily detectable and a relatively straightforward transition to select for when coding a to-from change detection algorithm. Additionally, the visual change is stark, so a visual interpreter can easily create a training dataset without ancillary information to identify conversion from forest to burned, cleared, or developed.

In contrast, vegetation degradation and recovery in rangelands are typically characterized by more gradual changes over longer periods of time. Unlike with forest change, a trained interpreter using only satellite imagery cannot visually identify a rangeland land cover class transition until years after it has begun, if at all.

However, there is potential to utilize the information generated from temporal segmentation to characterize other aspects of change. Here, we will employ the LandTrendr time series segmentation algorithm (as described in Chap. F4.5) to derive

information about how pixels are changing over time, and use that to generate a new land cover classification for a region of northern China.

To truly understand how rangelands are changing in space and time, and to provide adequate monitoring to support sustainable rangeland management, we need tools to help us capture and quantify those changes at landscape scales. Further, we need new ways of measuring changes within land cover types and interpreting those changes in ways that reflect rangeland dynamics and ecological processes. This necessitates rethinking the framework we use to categorize these lands in the first place. Approaches making use of greater temporal information on forest (Healey et al. 2018) and wetland systems (Dronova et al. 2015) show promise for deployment on rangeland systems.

We present a hybrid approach to rangeland classification that categorizes observable dynamic patterns in vegetation cover to classify pixels based on similarities in trajectory history. These new classes reveal much more meaningful information about the current status of a given pixel than a class based on cover type, and also yield information about the potential of a given pixel to respond to stressors in the future.

Practicum

Section 1. Inspecting Information about the Study Area

In this section, we will load and explore data sets that illustrate the area, and form the basis for the analysis.

Section 1.1 Inspect the Study Area

Naiman Banner (Fig. A3.8.3) is located in the southeastern portion of the Inner Mongolia Autonomous Region of northern China. Historically, this region was occupied by ethnic Mongolian pastoralists, and the primary vegetation was perennial grasses with some shrub cover. The region has undergone significant intensification of land uses over the past 60 years. Heavy grazing and conversion to crops has removed vegetation, exposing soils to erosion and resulting in extensive desertification. Since the early 1990s, a series of environmental policies and restoration programs, including grazing restriction and afforestation, have been implemented to halt the spread of desertification and promote revegetation of the rangelands. At the same time, cropland has continued to expand, and the use of irrigation has eliminated almost all of the surface water sources and severely lowered the groundwater table.

In this exercise, we will focus on a portion of central Naiman Banner that spans from the extensive Horqin Sandy Lands in the west to the dense agricultural area in the east.

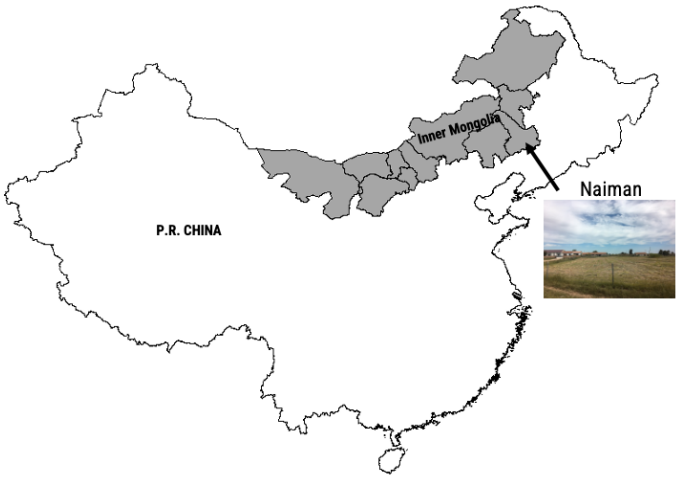


Fig. A3.8.3 Location of the Horqin Sandy Lands within Naiman Banner, Inner Mongolia, People's Republic of China

Our first step is to load a shapefile for our area of interest (AOI) and explore the landscape using the default satellite basemap.

```
// Load the shapefile asset for the AOI as a Feature Collection
var aoi = ee.FeatureCollection(
    'projects/gee-book/assets/A3-8/GEE_Ch_AOI');
Map.centerObject(aoi, 11);
Map.addLayer(aoi, {}, 'Subset of Naiman Banner');
```

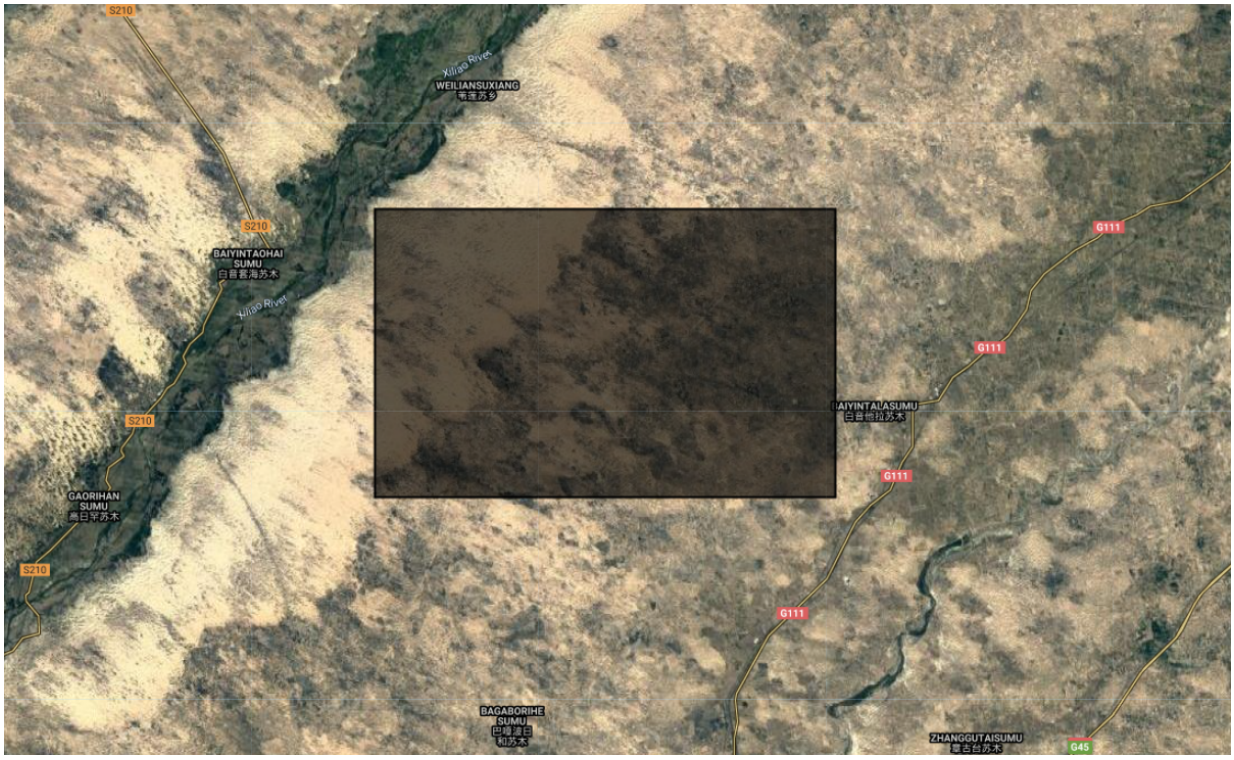


Fig. A3.8.4 Location of the study region, within the Horqin Sandy Lands

Switch the basemap to **Satellite** (Fig. A3.8.4). Turn the layer with the AOI boundary on and off and pan around the study area. Inspect the difference in land uses and cover types between the far western and eastern edges of the AOI. Mark them with pointer markers so you can revisit them later.

Question 1. List four potential land use or land cover classes that you observe in the image.

Section 1.2 Inspect Existing Land Use Data

Next, we will explore two different sources of land cover data to get a sense of how this landscape is typically categorized by these kinds of classified products.

First, we will explore the MODIS MCD12Q1 Land Cover Type dataset. These global layers are available annually from 2001 to the present, at a resolution of 500 m. We will

filter and visualize data for 2001, 2009, and 2016 in order to compare how MODIS captures change over this period.

Next, we will define and execute several different functions in this lesson. These functions will help us to execute the same logic across multiple images within different image collections.

```
// Filter the MODIS Collection
var MODIS_LC = ee.ImageCollection('MODIS/006/MCD12Q1').select(
    'LC_Type1');

// Function to clip an image from the collection and set the year
var clipCol = function(img) {
    var date = ee.String(img.get('system:index'));
    date = date.slice(0, 4);
    return img.select('LC_Type1').clip(aoi) // .clip(aoi)
        .set('year', date);
};

// Generate images for diff years you want to compare
var modis01 = MODIS_LC.filterDate('2001-01-01', '2002-01-01').map(
    clipCol);
var modis09 = MODIS_LC.filterDate('2009-01-01', '2010-01-01').map(
    clipCol);
var modis16 = MODIS_LC.filterDate('2016-01-01', '2017-01-01').map(
    clipCol);
// Create an Image for each of the years
var modis01 = modis01.first();
var modis09 = modis09.first();
var modis16 = modis16.first();
```

Now that we have loaded all three datasets, let's take a look at them and see how they compare. The `randomVisualizer` code below assigns random colors to each class in each layer.

```
Map.addLayer(modis01.randomVisualizer(), {}, 'modis 2001', false);
Map.addLayer(modis09.randomVisualizer(), {}, 'modis 2009', false);
```

```
Map.addLayer(modis16.randomVisualizer(), {}, 'modis 2016', false);
```

You should end up with something that looks like Fig. A3.8.5. (The colors assigned to classes in your map may differ.)

Fig. A3.8.5 Land cover classes identified in 2001 from MODIS MCD12Q1

Use the slider bar in the **Layer** menu to turn the data on and off to compare classifications across the three years. Use the **Inspector tool** to select a few pixels around the AOI to pull information on the specific classes. Compare the pixel values reported in the **Inspector** to the land cover codes and descriptions listed in Table A3.8.1.

Table A3.8.1 Subset of the 17 IGBP Classes that appear within the study area in the MODIS MCD12Q1 land cover dataset.

Value	Description
7	Open Shrublands: dominated by woody perennials (1-2m height) 10-60% cover.
10	Grasslands: dominated by herbaceous annuals (<2m).
12	Croplands: at least 60% of area is cultivated cropland.
16	Barren: at least 60% of area is non-vegetated barren (sand, rock, soil) areas with less than 10% vegetation.

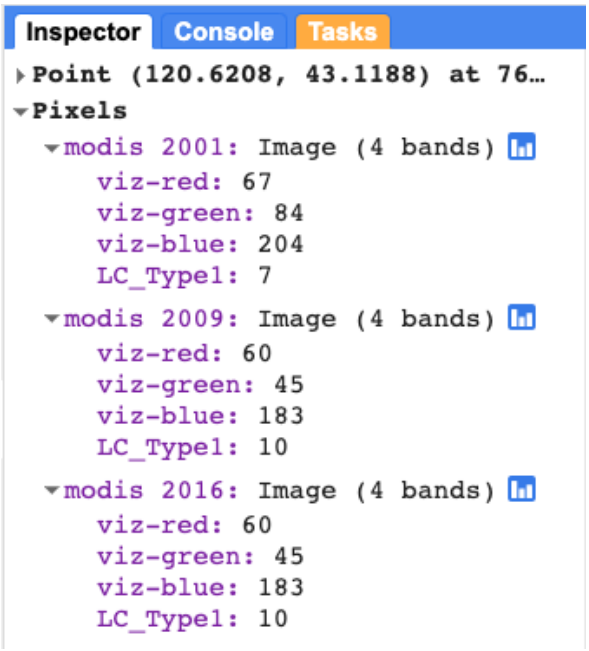


Fig. A3.8.6 Land cover classes assigned to a single pixel in the study area in 2001, 2009, and 2016 from MODIS MCD12Q1

Question 2. What are the four land cover types identified in this region, as classified by the MODIS data? Be sure to check all three time periods.

Next, we will add the WorldCover dataset from the ESA. This dataset was generated for 2020 only, and has a resolution of 10 m.

```
// Add and clip the WorldCover data
```

```
var wCov = ee.ImageCollection('ESA/WorldCover/v100').first();
var landcover20 = wCov.clip(aoi);
Map.addLayer(landcover20, {}, 'Landcover 2020');
```

The WorldCover dataset includes palette information in the `'Map'` band, so you should end up with something that looks like Fig. A3.8.7. (The colors assigned to classes in your map may differ.)

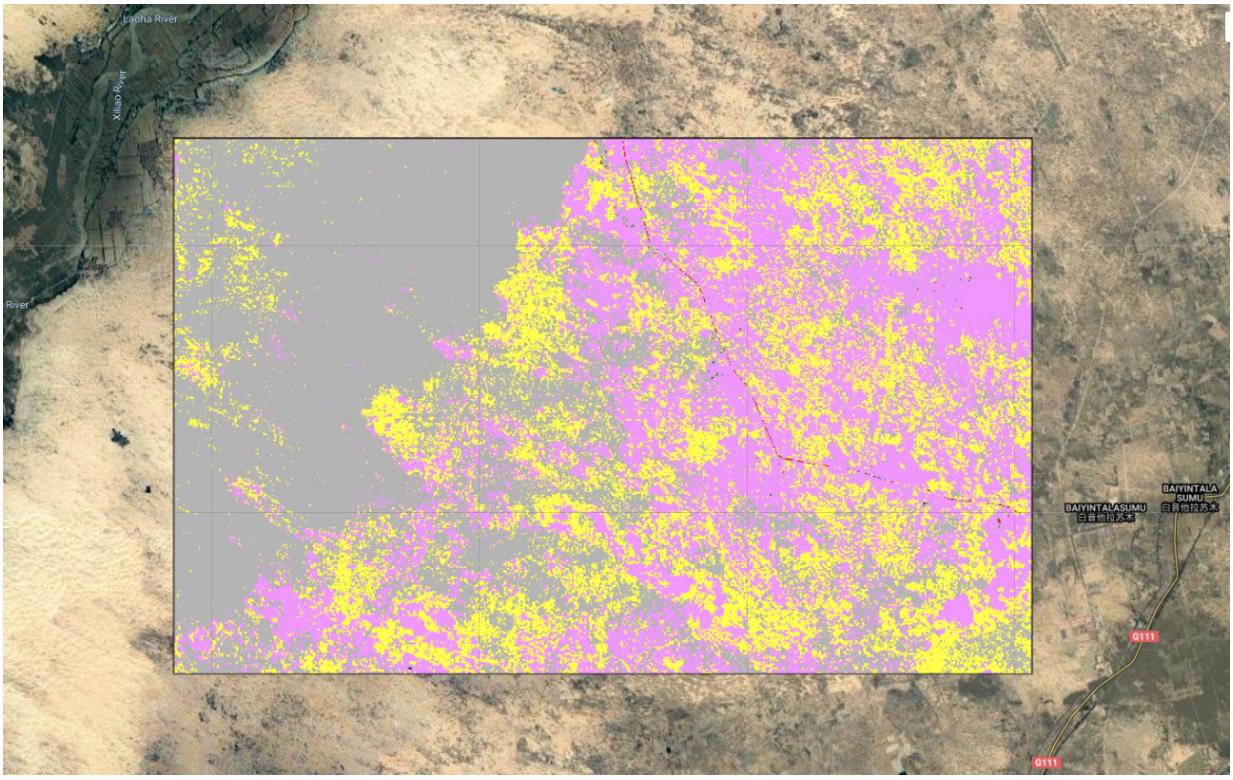


Fig. A3.8.7 Land cover classes identified in 2020 in the ESA WorldCover dataset

Table A3.8.2 Subset of the 11 land cover classes that appear within the study area in the ESA WorldCover dataset

Value	Description
10	Trees
30	Grassland
40	Cropland
50	Built-up
60	Barren / sparse vegetation

Code Checkpoint A38a. The book's repository contains a script that shows what your code should look like at this point.

Question 3. Spend a few minutes turning the different land cover layers on and off and using the Inspector to identify the classes for different pixels. How do the MODIS and WorldCover datasets differ? In what ways are they similar?

Question 4. Return to the points that you flagged in Section 1. Do your estimations of land cover correspond to the classifications from the two different data sources?

Question 5. Qualitatively compare change over time according to the MODIS data. What do you observe from these data? Approximately how much of the AOI has changed between 2001 and 2020, according to these data? Where are the regions of changing classification located? What regions seem to be fairly stable?

Section 2. Compile the Time Series of Vegetation Cover

The Normalized Difference Vegetation Index (NDVI) is an index of vegetative productivity derived from the relationship between the red (approx. 650 nm) spectra and the near-infrared (750–2500 nm) spectra. NDVI is a consistently good proxy for productivity in this region (de Beurs et al. 2015). It is also highly correlated with precipitation (John et al. 2016). This relationship has the potential to introduce short-term responses of increased vegetation productivity that could be falsely identified as land cover change (Fig. A3.8.8 a and b). In order to account for this effect, we need to remove the main effect of precipitation on NDVI for a given year so that we can assess changes in greenness outside of that variability. To do this, we will derive individual regression models for each pixel (as detailed in Chap. F4.6) for the relationship between total annual water-year precipitation and maximum greenness in each year. We will then predict greenness for each pixel in each year based on observed precipitation, and use

the residual values of greenness from the predicted model as an input to LandTrendr. We can think of the residuals as the remaining annual primary productivity, after we have removed the effect of precipitation (Fig. A3.8.8 c).

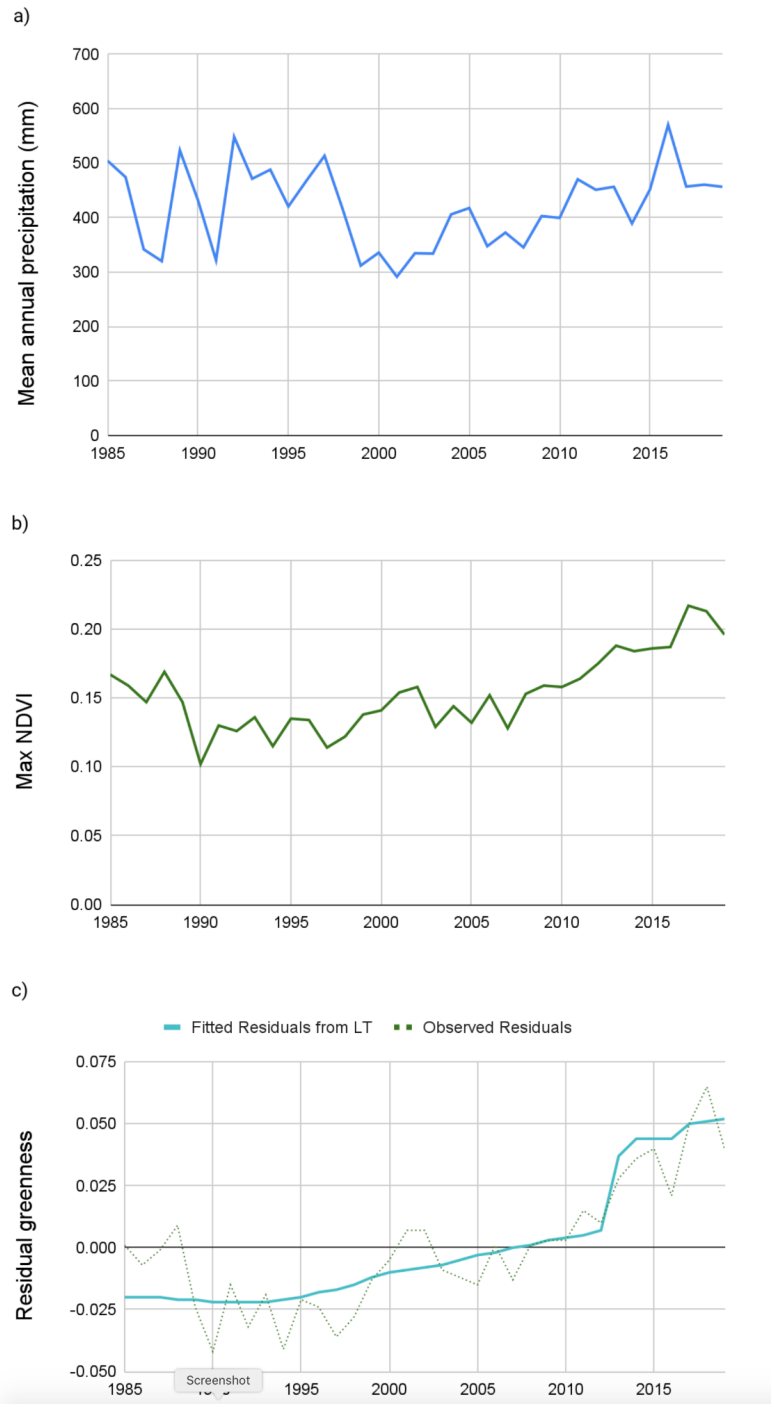


Fig. A3.8.8 Comparison of the variation over time in a) mean annual precipitation and b) maximum NDVI for the AOI; and a time series of residual greenness from a regression of precipitation and NDVI.

Before we go any further, let's add two pre-generated image collection assets for greenness and precipitation (`greennessColl` and `precipColl`) and explore them. Print each of them to the **Console** to inspect the contents (Fig. A3.8.9).

```
var greennessColl = ee.ImageCollection(
    'projects/gee-book/assets/A3-8/GreennessCollection_aoi');
var precipColl = ee.ImageCollection(
    'projects/gee-book/assets/A3-8/PrecipCollection');
print(greennessColl, 'Greenness Image Collection');
print(precipColl, 'Precip Image Collection');
```

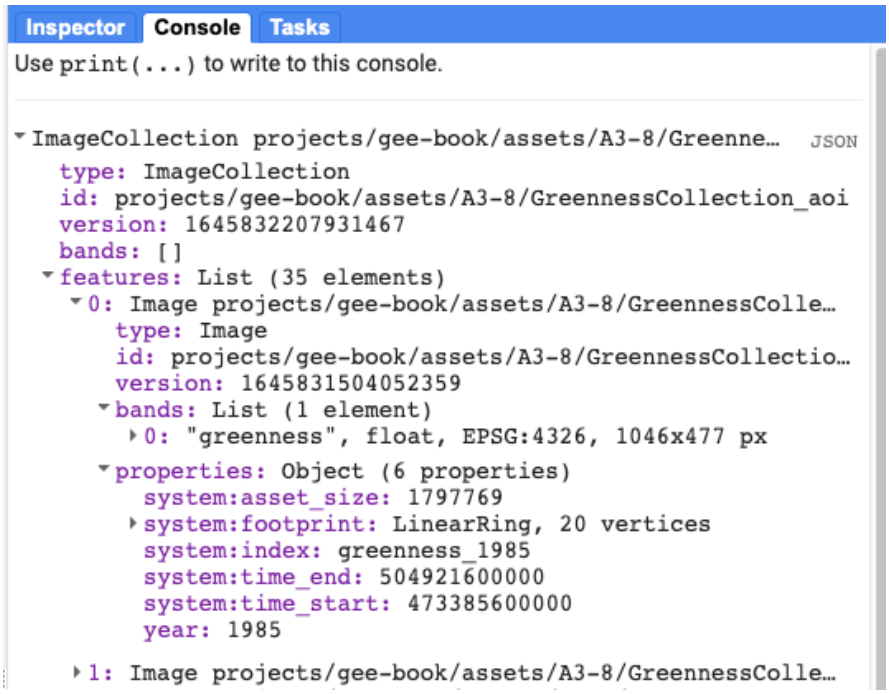


Fig. A3.8.9 Properties of the pre-generated Image Collection of maximum annual greenness values, generated from NDVI

We saw in the plots above (Figs. A3.8.1 and A3.8.2) how NDVI can vary significantly over time in this region. We can also select out a few years to visualize how NDVI ("greenness") varies spatially as well.

```
var greennessParams = {
    bands: ['greenness'],
    max: 0.5,
    min: 0.06,
    opacity: 1,
    palette: ['e70808', 'ffffff', '1de22c']
};

var greenness1985 = greennessColl.filterDate('1985-01-01',
    '1986-01-01').select('greenness');
var greenness1999 = greennessColl.filterDate('1999-01-01',
    '2000-01-01').select('greenness');

print(greenness1999);
var greenness2019 = greennessColl.filterDate('2019-01-01',
    '2020-01-01').select('greenness');

Map.addLayer(greenness1985, greennessParams, 'Greenness 1985', false);
Map.addLayer(greenness1999, greennessParams, 'Greenness 1999', false);
Map.addLayer(greenness2019, greennessParams, 'Greenness 2019', false);
```

Turn the layers for the different years on and off to compare the range and spatial distribution of NDVI (Fig. A3.8.10).

Question 6. What do you observe about the similarities and differences in the distribution of NDVI across the selected years?

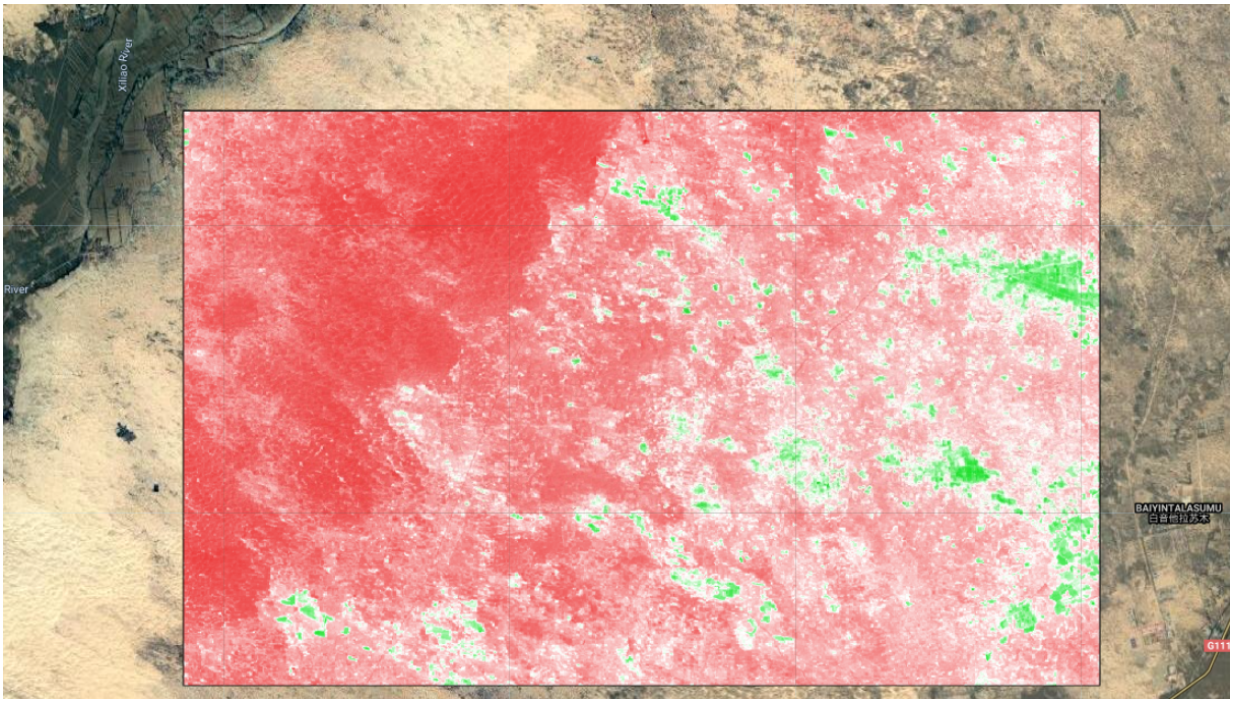


Fig. A3.8.10 Observed greenness (NDVI) in the study region in 2019. The values range from a minimum of 0.07 to a maximum of 0.5.

Combining the greenness and precipitation collections and calculating the model to generate residuals is a relatively long process. To speed things along, we will employ a function that has been defined in a script module called `residFunctions`.

```
// Load a function that will combine the Precipitation and Greenness
collections, run a regression, then predict NDVI and calculate the
residuals.

// Load the module
var residFunctions = require(
    'projects/gee-edu/book:Part A - Applications/A3 - Terrestrial
Applications/A3.8 Detecting Land Cover Change in
Rangelands/modules/calcResid'
);

// Call the function we want that is in that module
```

```
// It requires three input parameters:
// the greenness collection, the precipitation collection and the aoi
var residualColl = (residFunctions.createResidColl(greennessColl,
    precipColl, aoi));

// Now inspect what you have generated:
print('Module output of residuals', residualColl);
```

Print the resulting `ImageCollection` and inspect the bands and properties in the **Console** (Fig. A3.8.11).

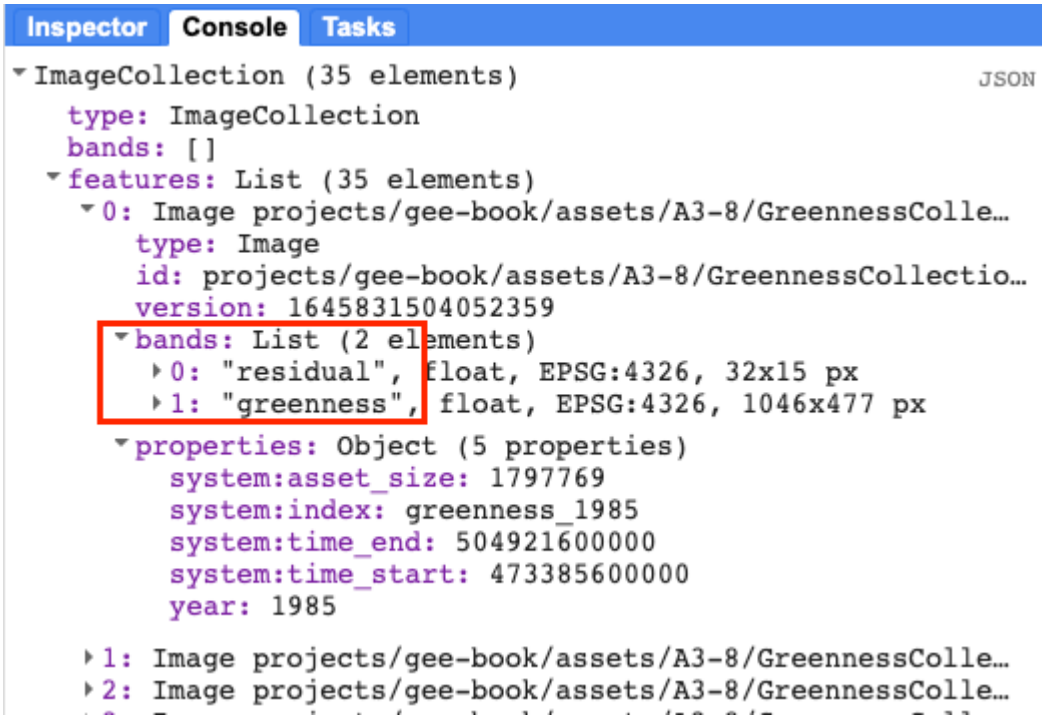


Fig. A3.8.11 Each image in the residualColl image collection contains two bands, residual and greenness. The residual band for will be passed to LandTrendr.

Next, you will filter the `residualColl` collection to map the same years we explored for the observed NDVI (greenness). The code chunk below will pull the image for 1985 (the first year). Use `filterDate` to select the other years you want to view. Add each to the map as new layers.

```
var resids = residualColl.first();
var res1 = resids.select(['residual']);
print(res1.getInfo(), 'residual image');
Map.addLayer(res1, {
    min: -0.2,
    max: 0.2,
    palette: ['red', 'white', 'green']
}, 'residuals 1985', false);
```

Map and compare the residuals versus greenness for a few different years (Fig. A3.8.12).

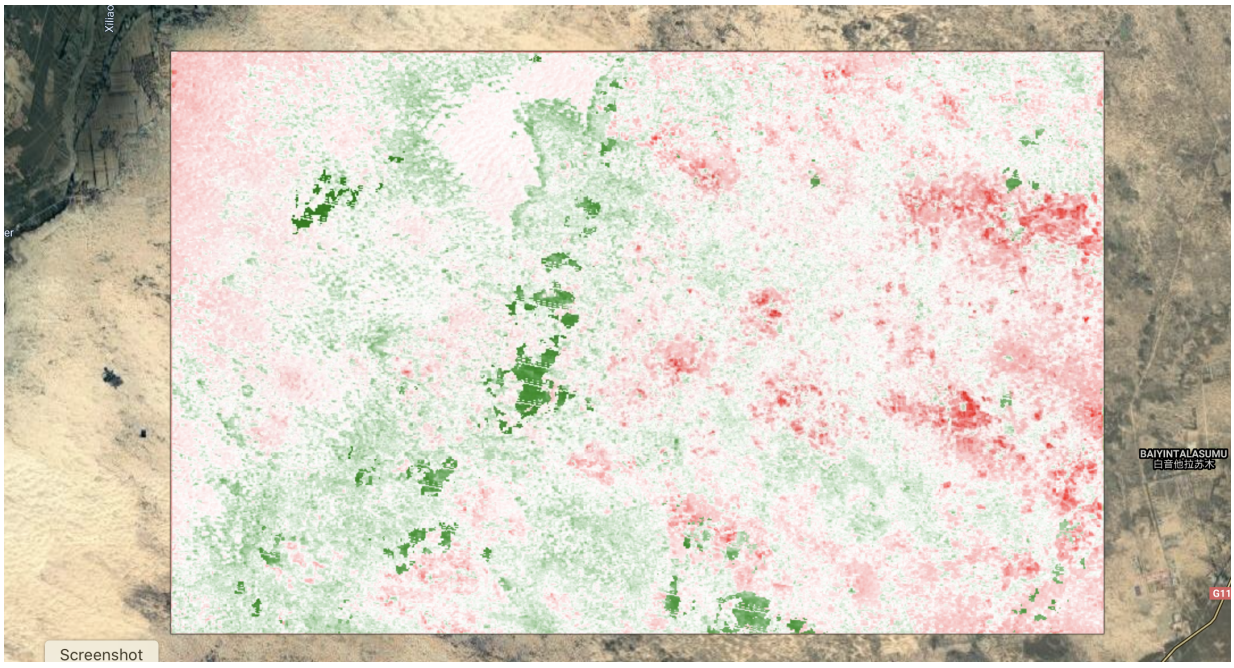


Fig. A3.8.12 Residual greenness in 1985, after removing the effect of precipitation. Models were fit on a per-pixel basis.

Code Checkpoint A38b. The book's repository contains a script that shows what your code should look like at this point.

Question 7. Compare the layers for residual greenness to the observed greenness values from Question 6. Why are we passing residuals to LandTrendr rather than the beginning NDVI values?

Section 3. Time Series Segmentation

Now you are ready to apply the LandTrendr time series segmentation algorithm to your Image Collection of residual greeness for the years 1985-2019. This will generate information for each pixel in the study area about how it has changed over time. In order to execute the code, you need to first define pertinent parameters in a dictionary, which you will provide to the LandTrendr algorithm along with your data. Chap. F4.5 gives an explanation of the LandTrendr algorithm and its parameters. That chapter showed an interface for LandTrendr; below, we utilize JavaScript functions to execute LandTrendr functions directly in the code editor.

```
//---- DEFINE RUN PARAMETERS---//
// LandTrendr run parameters
var runParams = {
    maxSegments: 6,
    spikeThreshold: 0.9, //
    vertexCountOvershoot: 3,
    preventOneYearRecovery: true,
    recoveryThreshold: 0.25, //
    pvalThreshold: 0.05, //
    bestModelProportion: 0.75,
    minObservationsNeeded: 10 //
};
```

Follow the next steps to combine the dictionary of parameter settings with the image collection and apply LandTrendr with the API functionality `ee.Algorithms.TemporalSegmentation.LandTrendr`. Then print and explore the output.

```
// Append the image collection to the LandTrendr run parameter
dictionary
var srCollection = residualColl;
```

```
runParams.timeSeries = srCollection;

// Run LandTrendr
var lt = ee.Algorithms.TemporalSegmentation.LandTrendr(runParams);
// Explore the output from running LT
var ltlt = lt.select('LandTrendr');
print(ltlt);
```

The implementation of the LandTrendr Algorithm in GEE generates a multidimensional array containing subarrays for the observation year, observed residual, fitted residual, and Boolean vertex layer that tells the user if a change in pixel trajectory occured in a given observation year. In order to analyze the outputs year over year, we first must slice out the subarrays of the output multidimensional array, and transform them into Image collections. We will not go into detail about slicing arrays in this lesson. For your purposes here, you can just follow along with the provided code. If you wish to learn more about array indexing and slicing, you can refer back to Chap. F4.6).

```
//---- SLICING OUT DATA -----------------//

// Select the LandTrendr band.
var ltlt = lt.select('LandTrendr');
// Observation Year.
var years = ltlt.arraySlice(0, 0, 1);
// Slice out observed Residual value.
var observed = ltlt.arraySlice(0, 1, 2);
// Slice out fitted Residual values (predicted residual from final LT
model).
var fitted = ltlt.arraySlice(0, 2, 3);
// Slice out the 'Is Vertex' row - yes(1)/no(0).
var vertexMask = ltlt.arraySlice(0, 3, 4);
// Use the 'Is Vertex' row as a mask for all rows.
var vertices = ltlt.arrayMask(vertexMask);
```

Next, we will extract fitted residual values for each pixel in each year from the array slice and convert them to an image with one band per year. First we need to define a few parameters that we will need to call in future steps.

```
// Define a few params we'll need next:
var startYear_Num = 1985;
var endYear_Num = 2019;
var numYears = endYear_Num - startYear_Num;
var startMonth = '-01-01';
var endMonth = '-12-31';
```

And now we can use the following code block to create a multi-band Image of the residual greeness predicted by LandTrendr for each pixel, with one band per year.

```
// Extract fitted residual value per year, per pixel and aggregate
into an Image with one band per year
var years = [];
for (var i = startYear_Num; i <= endYear_Num; ++i) years.push(i
    .toString());
var fittedStack = fitted.arrayFlatten([
    ['fittedResidual'], years
]).toFloat();
print(fittedStack, 'fitted stack');
```

Add the fitted residuals for 1985 as a layer and compare them to the observed values you mapped previously. Notice how the range of values is more constrained than the observed values shown in Fig. A3.8.13. This is because the values are predicted from the best-fit model for the series, and thus do not contain the outliers and extreme values we might capture in the observed data.

```
Map.addLayer(fittedStack, {
    bands: ['fittedResidual_1985'],
    min: -0.2,
    max: 0.2,
    palette: ['red', 'white', 'green']
}, 'Fitted Residuals 1985');
```

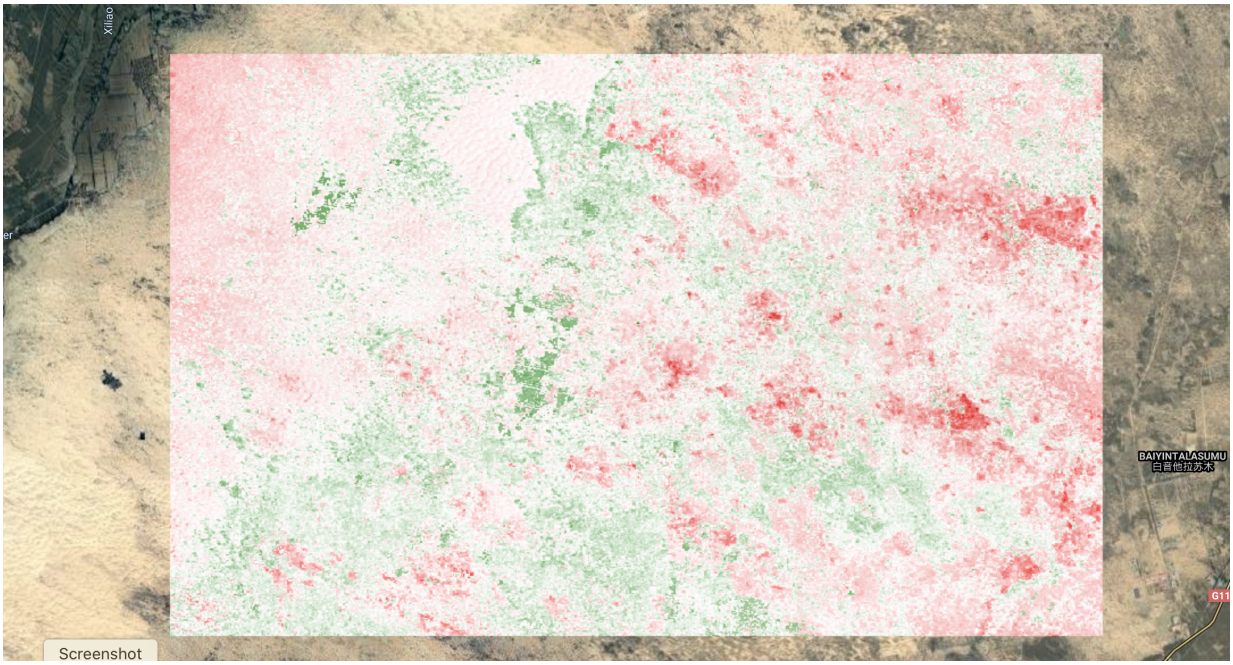


Fig. A3.8.13 Fitted values for residual greenness in 1985, as predicted by the model fit via LandTrendr

One of the very useful outputs from LandTrendr is information on whether the algorithm assigned a vertex to a given pixel in a given year. We can generate a raster with a band for each year, which indicates whether or not a pixel had a vertex identified in that year with a Boolean no/yes (0/1). We can use that information to assess how much of the AOI changed in a given year by assessing the prevalence of pixels with a value of 1 (indicating that a vertex was identified). To do this, we need to slice the Boolean information out of the LandTrendr output array, and then assign a year to each band.

```
// Extract Boolean 'Is Vertex?' value per year, per pixel and
aggregate into image w/ Boolean band per year
var years = [];
for (var i = startYear_Num; i <= endYear_Num; ++i) years.push(i
    .toString());

var vertexStack = vertexMask.arrayFlatten([
    ['bools'], years
]).toFloat();
```

```
print(vertexStack.getInfo(), 'vertex Stack');
```

If you print the resulting image, you should see something like Fig. A3.8.13 in your **Console**. You'll notice that again we have a band for each year, but this time each band is a binary raster, where a value of one (1) indicates that the pixel had a vertex in that year.

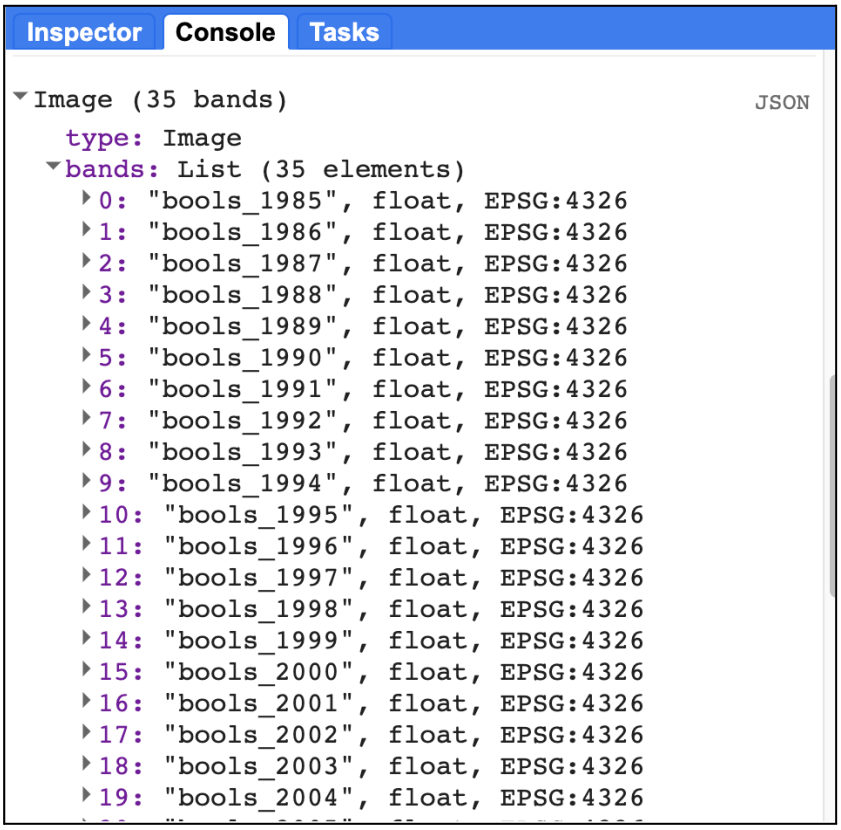


Fig. A3.8.13 This is how the Boolean value binary rasters should appear in your **Console** once they are sliced out of the LandTrendr. Notice that the observation year has been appended to the name of each raster.

In this next step, we will inspect a plot of the mean value of the Boolean (vertex) layers for each year in order to identify which years had the most change. We will estimate the proportion of the AOI that has a vertex identified in each year by mapping a Reducer over a collection to calculate the mean pixel values for each year. A mean of zero would indicate that no vertices were identified in that year, and a mean of one would indicate

that all the pixels changed. Of course, neither of these values are likely; most years will have a relatively small number of pixels that change, and thus the value will be low, but not zero.

Charting functions in Earth Engine requires an `ImageCollection`. In the interest of time, we will load this for you as a new asset. If you would like to see an example script for transforming a multi-band image to an `ImageCollection`, you can find it in script **A38s1 - Supplemental** in the book's repository.

```
// Load an Asset that has the Booleans converted to Collection
var booleanColl = ee.ImageCollection(
    'projects/gee-book/assets/A3-8/BooleanCollection');
```

Now that we have our Boolean bands in an `ImageCollection` with one image per year, we can run a Reducer over it and create a chart of our results.

```
var chartBooleanMean = ui.Chart.image
    .series({
        imageCollection: booleanColl.select('bools'),
        region: aoi,
        reducer: ee.Reducer.mean(),
        scale: 60,
        xProperty: 'system:time_start'
    })
    .setChartType('ScatterChart')
    .setOptions({
        title: 'Naiman Boolean Mean Per Year',
        vAxis: {
            title: 'Boolean Mean Per Year'
        },
        lineWidth: 1
    });

print(chartBooleanMean);
```

You should end up with a plot in the **Console** that looks something like Fig. A3.8.14.

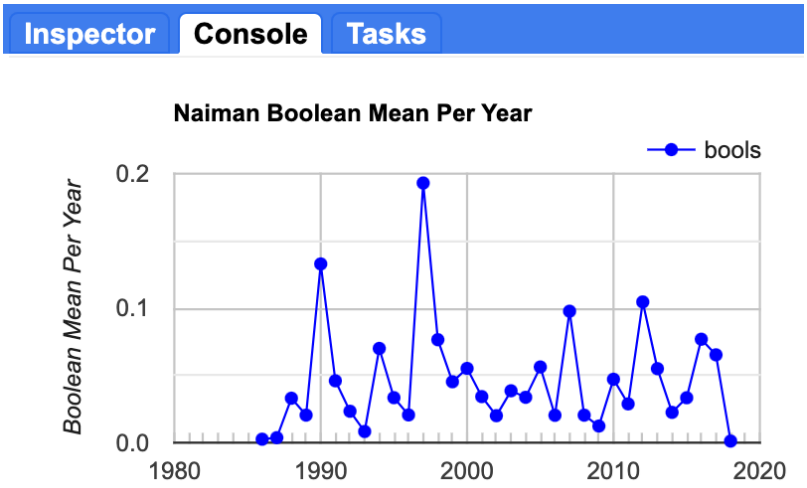


Fig. A3.8.14 Average pixel value across the study area in each year. A value of one would indicate that all pixels in the image had a vertex identified in that year. The maximum proportion of landscape that changed in a given year was approximately twenty percent, in 1997.

Question 9. Which years had the greatest number of pixels with a vertex (indicating a change in the trajectory of the timeseries)?

Now that you have identified those years, let's inspect the spatial patterns and locations of those pixels for the major change years. For this, we can use our `vertexStack` image. Set the visualization parameters using the code provided below, then edit the specific year in the `bands` setting to match the year you want to visualize. Figure A3.8.14 displays the pixels that changed in 1997.

```
// Plot individual years to see the spatial patterns in the vertices.
var boolParams = {
    // change this for the year you want to view
    bands: 'bools_1997',
    min: 0,
    // no vertex
    max: 1,
    // vertex identified by LT for that year
    palette: ['white', 'red']
};
```

```
Map.addLayer(vertexStack, boolParams, 'vertex 1997', false);
// this visualizes all pixels with a vertex in that year.
```

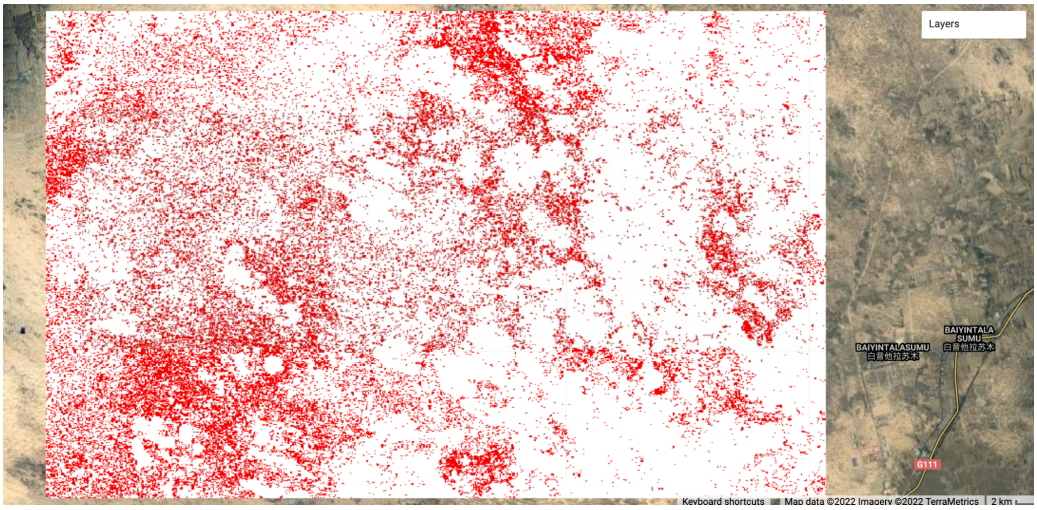


Fig. A3.8.14 Location of pixels with a vertex identified in 1997

Run the script several times, changing the year in the bands parameter to match the top change years you identified in the previous step. Take a screenshot of each one and store it so that you can compare them.

Question 10. Do you notice any differences in the spatial patterns of pixels across the top four change years, either in where they are located or their relative patch sizes or aggregation?

Code Checkpoint A38c. The book's repository contains a script that shows what your code should look like at this point.

Section 4. Classify Pixels Based on Similarities In Time Series Trajectories

In the `vertexStack` object that we created, we have information about every time there was a change in trajectory of the time series of residuals for every pixel in the AOI. In this next step, we are going to look for patterns or similarities in those trajectories to characterize different trajectory archetypes, representing unique pixel histories, as the basis for a classification.

In the visualizations above, we focused on the timing and spatial patterns for the top four years of change. If we extract the time series of residual greenness for every pixel in Fig. A3.8.14 above, we get something like the plots shown in Fig. A3.8.15. If we extract the time series of all pixels that changed in 1990, we get something like Fig. A3.8.15a. If we compare these plots back to our screenshots of the locations of the vertex pixels in those years, we can see that there are distinct patterns both in the spatial location and the shape of the time series over time between pixels that had a vertex in 1990 and those that changed in 1997 (Fig. A3.8.15b), or even 2012 (Fig. A3.8.15c). Notice the differences in the trajectory of greenness prior to the vertex across the three plots, and that the scale of the y-axis varies across the plots as well.

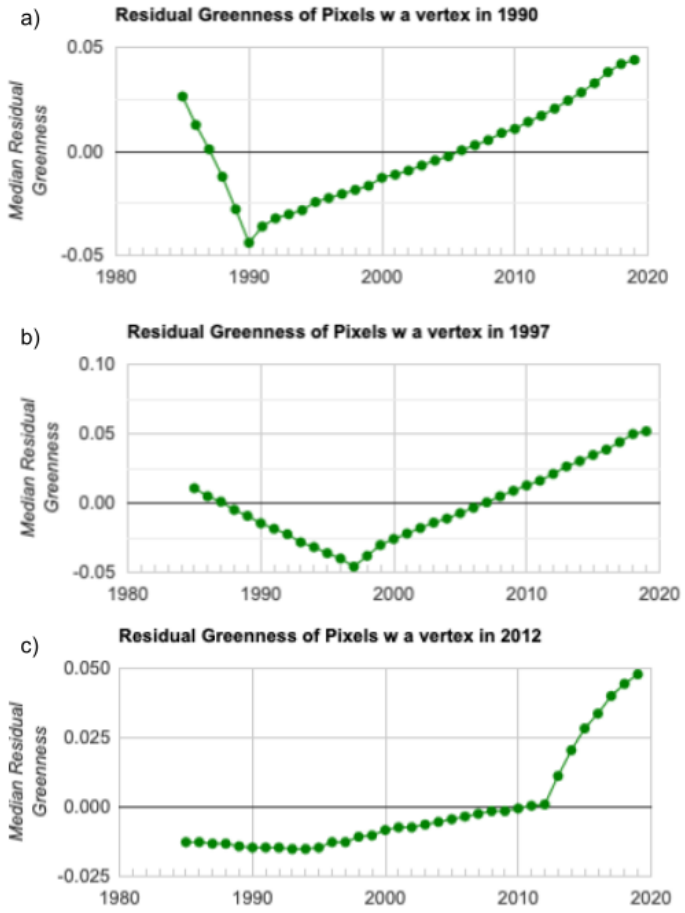


Fig. A3.8.15 Median residual greenness values for all pixels in the AOI with a vertex identified by LandTrendr in a) 1990; b) 1997; and c) 2012

The plots shown in Fig. A3.8.15 show the median value for each year across all pixels that had a vertex in that year. But it is also possible that a single pixel has more than one vertex in its history (i.e., that it changed trajectory more than once) (Fig. A3.8.2), so there may be some overlap between pixels with a vertex in 1990 and those with a vertex in 1997, or in other years. We are going to use all of the information on the changes across all years to determine the different types of change that have occurred.

We are going to employ an unsupervised classification approach to clustering these data, as we do not have ground truth or pre-classified training data that we are trying to replicate. Rather, we are interested in finding out what kinds of inherent patterns might exist across the pixels in our AOI, based on similarities in the shape of their trajectories. Different trajectories in the time series of greenness might be due to things like starting conditions, as well as different kinds of management or environmental stress.

We will start by creating some naive training data from our multiband image of Boolean layers.

```
// Create training data.
var training = vertexStack.sample({
    region: aoi,
    scale: 60,
    numPixels: 5000
});
```

Now, set the maximum number of allowable clusters to 10, train the clusterer and apply it to the vertex data.

```
var maxclus = 10;

// Instantiate the clusterer and train it.
var trained_clusterer = ee.Clusterer.wekaKMeans(maxclus).train(
    training);

// Cluster the input using the trained clusterer
var cluster_result = vertexStack.cluster(trained_clusterer);
```

The default indexing in Java starts at zero, so the first class assigned by the clusterer is labeled with the value 0. This can pose problems if you want to mask out the results to view only one cluster at a time, so we will quickly remap the 0 value to be a 10. Then, we'll add the results as a new layer to quickly visualize the classified raster (Fig. A3.8.16).

```
// Remap result_totalChange so that class 0 is class 10
cluster_result = cluster_result.remap(
        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
        [10, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    .toFloat()
    .rename('cluster');
Map.addLayer(cluster_result.randomVisualizer(), {}, maxclus
.toString() + '_clusters');
```

Turn on the satellite basemap and move the opacity slider for the classified raster layer `10_clusters`. (The colors assigned to classes in your map may differ.) Some classes seem to align somewhat with observable features in the landscape, but many of them do not. This indicates that there is some similarity in the history of these pixels that is not immediately obvious from their current cover.

Code Checkpoint A38d. The book's repository contains a script that shows what your code should look like at this point.

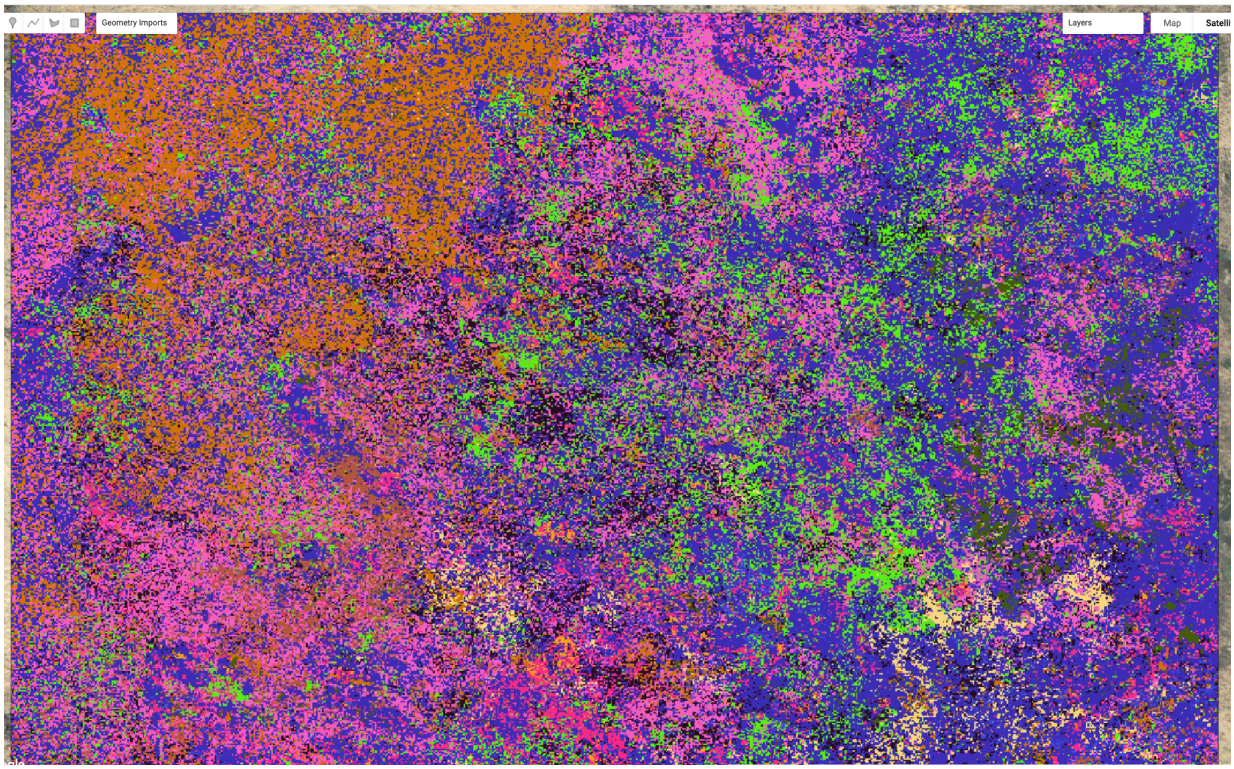


Fig. A3.8.16 Random-color visualization of unsupervised clusters

Section 5. Explore the Characteristics of the New Classes

We have now generated a classified raster based on similarities in the trajectory of the greenness in each pixel. In order to understand what these classes actually mean and what particular trajectories these classes represent, we will create summaries of the median greenness of the pixels in each class and compare them.

We will use the `ImageCollection` of observed greenness that we used in Sect. 2, and also a collection of the fitted residuals generated by LandTrendr in Sect. 3. Our first step will be to add a band with the cluster number to those collections.

```
// GOAL: Find Median Greenness for each cluster per year in the image
// define a function to add the cluster number band to each Image in
the collection
```

```
var addClusters = function(img) {
    return img.addBands(cluster_result);
};

// Add the cluster band
var ObvGreen_wClusters = greennessColl.map(addClusters);
```

Next, we need to select and mask out the class we are interested in exploring. We'll just start with the first class.

```
//---Select and mask pixels by cluster number
var cluster_num = 1; // change this to the class of interest

// Mask all pixels but the selected cluster number
// Define a function so we can map it over the entire collection
var maskSelCluster = function(img) {
    var selCluster = img.select('cluster').eq(cluster_num);
    return img.mask(selCluster);
};
// map the function over the entire collection
var selClusterColl = ObvGreen_wClusters.map(maskSelCluster);

// Use the following to visualize the location of the focal class:
Map.addLayer(selClusterColl.select('cluster').first(), {
    palette: 'green'
}, 'Cluster ' + cluster_num.toString());
```

Next, you will utilize the `ui.Chart.image` functionality, combined with a Reducer, to plot the median value of observed greenness for each year for the focal class.

```
var chartClusterMedian = ui.Chart.image.seriesByRegion({
        imageCollection: selClusterColl,
        regions: aoi,
        reducer: ee.Reducer.median(),
        band: 'greenness',
        scale: 90,
```

```
        xProperty: 'system:time_start',
        seriesProperty: 'label'
    })
    .setChartType('ScatterChart')
    .setOptions({
        title: 'Median Observed Greenness of Cluster ' +
            cluster_num.toString(),
        vAxis: {
            title: 'Median Observed Greenness'
        },
        lineWidth: 1,
        pointSize: 4,
        series: {
            0: {
                color: 'green'
            },
        }
    });

print(chartClusterMedian);
```

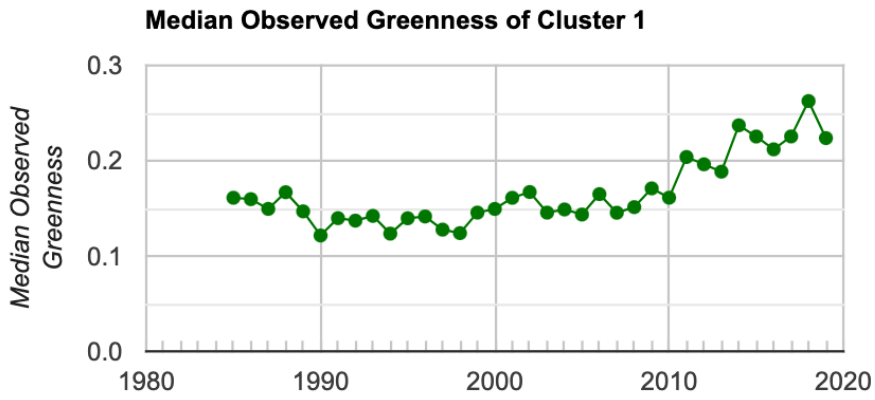


Fig. A3.8.17 Median value of the observed greenness value for all pixels in the AOI that are identified in Cluster 1

Now we will do the same, but for the fitted residual greenness values predicted from LandTrendr.

```
var fittedresidColl = ee.ImageCollection(
    'projects/gee-book/assets/A3-8/FR_Collection');
// add the cluster number band to each (function defined above, just
use again here)
var fittedresid_wClusters = fittedresidColl.map(addClusters);

//Mask all pixels but the selected cluster number
// again, function defined above, just call it here
var selFRClusterColl = fittedresid_wClusters.map(maskSelCluster);

Map.addLayer(selFRClusterColl.select('cluster').first(), {
    palette: ['white', 'blue']
}, 'Cluster ' + cluster_num.toString());

//Chart Median Fitted Residual Values by cluster

var chartClusterMedian = ui.Chart.image.seriesByRegion({
        imageCollection: selFRClusterColl,
        regions: aoi,
        reducer: ee.Reducer.median(),
        band: 'FR',
        scale: 90,
        xProperty: 'system:time_start',
        seriesProperty: 'label'
    }).setChartType('ScatterChart')
    .setOptions({
        title: 'Median Fitted Residual Greenness of Cluster ' +
            cluster_num.toString(),
        vAxis: {
            title: 'Median Residual Greenness'
        },
        lineWidth: 1,
        pointSize: 4,
        series: {
            0: {
```

```
                color: 'red'
            },
        }
    });

print(chartClusterMedian);
```

From the **Console**, expand each of the two plots to a new tab, then download the data as a .csv file and rename it something like "observed_green_Class1.csv." **Repeat the steps above for each of the classes.**

Fig. A3.8.18 provides an example of the outputs for a few of the classes that you have generated. When viewed side by side, it is easier to see how pixels in some parts of the AOI have experienced different trajectories of vegetation cover over time, and that the exact time and nature of the shifts in their trajectories are also different. This may be due to differences in underlying vegetation, land use, and management activity.

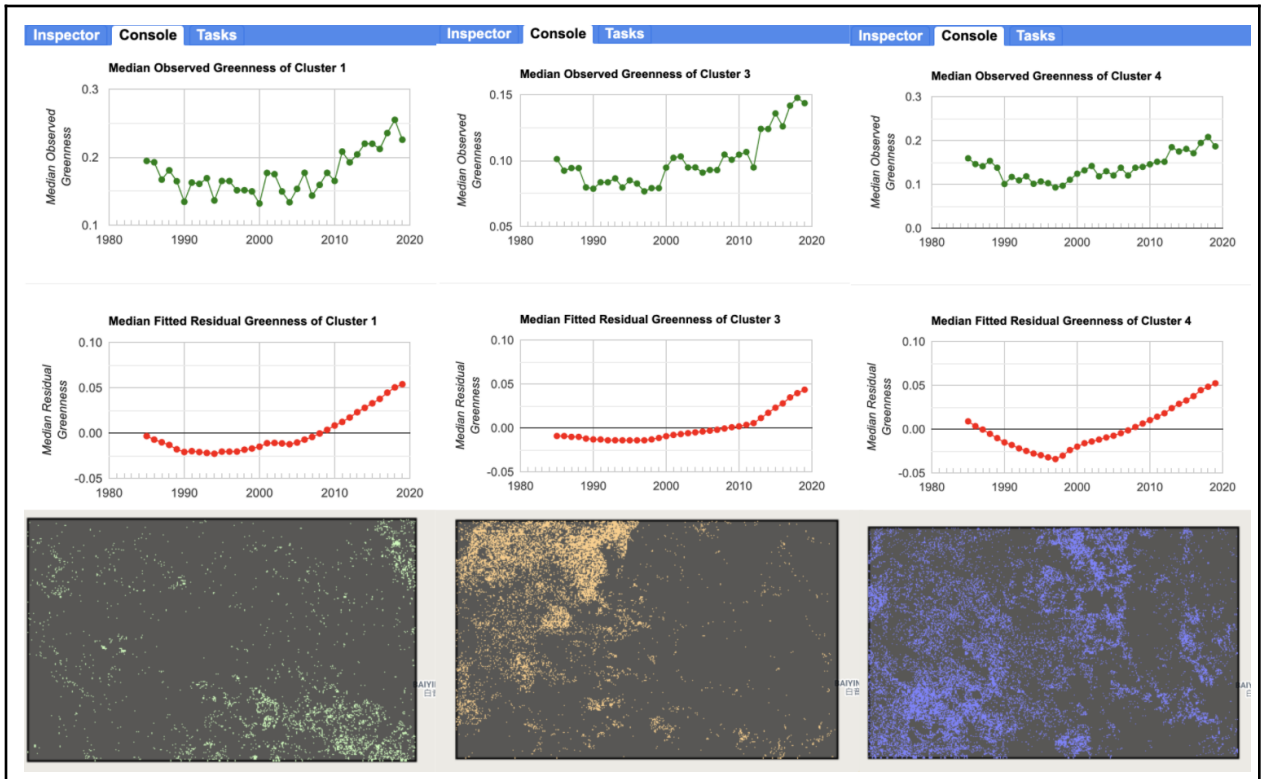


Fig. A3.8.18 Example outputs for three of the clusters identified in the data. The top row is the median greenness value for all pixels in the cluster. The middle row is the residual greenness value predicted by the fitted LandTrendr model. The bottom row is the location of all pixels in the AOI within that class; cluster 1 is more common in the southeast, Cluster 3 is aggregated in the northwestern region and Cluster 4 has patches throughout the AOI.

In a final step, we will compare the classes you generated by clustering the time series data to how the same locations are classified over time by the MODIS land cover data. We will do this by picking a few points within the AOI and plotting the land cover type number for each year of the MODIS data (see Table **A3.8.1** to link the value to the descriptions of the class types).

Use the Inspector tool to select a pixel in a region that interests you. Copy the code chunk below to your script. In the Inspector window, expand the information for the Point by clicking on the triangle next to Point and copy the longitude and latitude values and replace the values you copied into the script.

```
// Generate a point geometry.
var expt = ee.Geometry.Point(
    [120.52062120781073, 43.10938146169287]);
// Convert to a Feature.
var point = ee.Feature(expt, {});
```

The second line in the code chunk above converts the point to a Feature. Now you can run a Reducer over that point to extract values and plot them to a Chart (**Fig. A3.8.19).**

```
// Create a time series chart of MODIS Classification:
var chart_LC = ui.Chart.image.seriesByRegion(
        MODIS_LC, point, ee.Reducer.mean(), 'LC_Type1', 30,
        'system:time_start', 'label')
    .setChartType('ScatterChart')
    .setOptions({
        title: 'LC of Selected Pixels',
        vAxis: {
            title: 'MODIS landcover'
        },
        lineWidth: 1,
        pointSize: 4
    });

print(chart_LC);
```

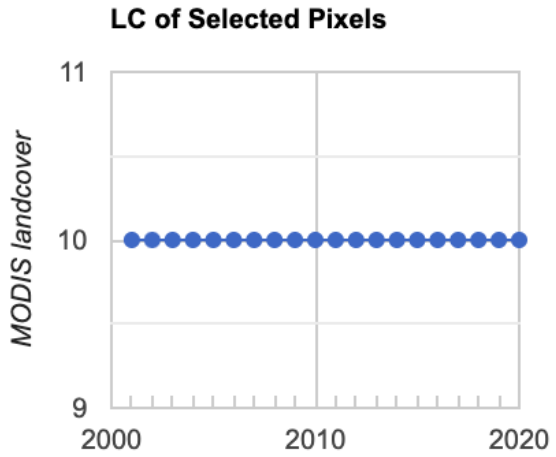


Fig. A3.8.19 Example output of the MODIS MCD12Q1 landcover class values over time for a single point in the AOI. Class 10 is "Grasslands: dominated by herbaceous annuals (<2m)".

Repeat the steps above for a few points of interest to you and save screenshots of the plots.

Code Checkpoint A38e. The book's repository contains a script that shows what your code should look like at this point.

Synthesis

Assignment 1. Combine all of your downloaded data files to create a single plot (in Excel or Google Sheets, for example) of the trajectory of median observed greenness for each of the classes. Compare each of these to the spatial patterns of the different classes (as in Fig. A3.8.15 and in your Map view), and to the underlying satellite image view. Pick three of the classes and describe the general trend in the timeseries (how has greenness changed over time?) and the spatial distribution of the pixels in that class.

We implemented this by specifying that the clusterer should look for 10 classes in the data. In a true implementation, we would want to explore the outputs across a range of cluster numbers. We may have forced the algorithm to split the data into too many (or too few) groups. Based on your inspection of the timing of the vertices and the spatial distribution of the final classes, are there any that you think could be grouped together in a final classification? What would you estimate to be the final number of classes in these data?

How much variation over time has there been for the different points according to the MODIS data? How does this compare to the variation in greenness represented in the final class?

Conclusion

In this module, you explored a new approach to classifying land cover that is based on the temporal trajectory of individual pixels. Earth Engine is a valuable tool for this analysis, because you are able to access the historical archive of imagery and climate data, as well as the computational tools needed to process, analyze, and classify these data. You learned how to create new derived datasets to use both as inputs to the analysis and as a final classified product. By comparing the temporal trajectories of the new classes against traditional land cover data, you learned how to distinguish the pros and cons of existing datasets for meeting land cover mapping objectives. Now that you understand the basics of the challenges of detecting land cover change in rangelands and have explored a new approach to classifying different trajectories, you can apply this approach to your own areas of interest to better understand the history of response.

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

Alexandratos N, Bruinsma J (2012) World agriculture towards 2030/2050: The 2012 Revision. ESA Work Pap 12:146. https://doi.org/10.22004/ag.econ.288998

Arino O, Bicheron P, Achard F, et al (2008) GlobCover: The most detailed portrait of Earth. Eur Sp Agency Bull 2008:24–31

Asner GP, Elmore AJ, Olander LP, et al (2004) Grazing systems, ecosystem responses, and global change. Annu Rev Environ Resour 29:261–299. https://doi.org/10.1146/annurev.energy.29.062403.102142

Bontemps S, Defourny P, Van Bogaert E, et al (2011) GLOBCOVER 2009: Products description and validation report. ESA Bull 136:53

Briske DD (2017) Rangeland Systems: Processes, Management and Challenges. Springer Nature

Campbell DJ, Lusch DP, Smucker TA, Wangui EE (2005) Multiple methods in the study of driving forces of land use and land cover change: A case study of SE Kajiado District, Kenya. Hum Ecol 33:763–794. https://doi.org/10.1007/s10745-005-8210-y

Cao S, Sun G, Zhang Z, et al (2011) Greening China naturally. Ambio 40:828–831. https://doi.org/10.1007/s13280-011-0150-8

Chen J, Chen J, Liao A, et al (2015) Global land cover mapping at 30 m resolution: A POK-based operational approach. ISPRS J Photogramm Remote Sens 103:7–27. https://doi.org/10.1016/j.isprsjprs.2014.09.002

Derner JD, Hunt L, Ritten J, et al (2017) Livestock Production Systems. In: Rangeland Systems. Springer, Cham, pp 347–372

Fan P, Chen J, John R (2016) Urbanization and environmental change during the economic transition on the Mongolian Plateau: Hohhot and Ulaanbaatar. Environ Res 144:96–112. https://doi.org/10.1016/j.envres.2015.09.020

Friedl MA, McIver DK, Hodges JCF, et al (2002) Global land cover mapping from MODIS: Algorithms and early results. Remote Sens Environ 83:287–302. https://doi.org/10.1016/S0034-4257(02)00078-0

Friedl MA, Sulla-Menashe D, Tan B, et al (2010) MODIS Collection 5 global land cover: Algorithm refinements and characterization of new datasets. Remote Sens Environ 114:168–182. https://doi.org/10.1016/j.rse.2009.08.016

Fritz S, See L (2005) Comparison of land cover maps using fuzzy agreement. Int J Geogr Inf Sci 19:787–807. https://doi.org/10.1080/13658810500072020

Ganguly S, Friedl MA, Tan B, et al (2010) Land surface phenology from MODIS: Characterization of the Collection 5 global land cover dynamics product. Remote Sens Environ 114:1805–1816. https://doi.org/10.1016/j.rse.2010.04.005

García-Mora TJ, Mas JF, Hinkley EA (2012) Land cover mapping applications with MODIS: A literature review. Int J Digit Earth 5:63–87. https://doi.org/10.1080/17538947.2011.565080

Healey SP, Cohen WB, Yang Z, et al (2018) Mapping forest change using stacked generalization: An ensemble approach. Remote Sens Environ 204:717–728. https://doi.org/10.1016/j.rse.2017.09.029

Homer C, Dewitz J, Yang L, et al (2015) Completion of the 2011 national land cover database for the conterminous United States – Representing a decade of land cover change information. Photogramm Eng Remote Sensing 81:345–354.

Huang J, Yu H, Guan X, et al (2016) Accelerated dryland expansion under climate change. Nat Clim Chang 6:166–171. https://doi.org/10.1038/nclimate2837

John R, Chen J, Lu N, Wilske B (2009) Land cover/land use change in semi-arid Inner Mongolia: 1992–2004. Environ Res Lett 4:45010. https://doi.org/10.1088/1748-9326/4/4/045010

Kennedy RE, Andréfouët S, Cohen WB, et al (2014) Bringing an ecological view of change to Landsat-based remote sensing. Front Ecol Environ 12:339–346. https://doi.org/10.1890/130066

Kennedy RE, Yang Z, Cohen WB (2010) Detecting trends in forest disturbance and recovery using yearly Landsat time series: 1. LandTrendr - Temporal segmentation algorithms. Remote Sens Environ 114:2897–2910. https://doi.org/10.1016/j.rse.2010.07.008

Lambin EF, Meyfroidt P (2011) Global land use change, economic globalization, and the looming land scarcity. Proc Natl Acad Sci USA 108:3465–3472. https://doi.org/10.1073/pnas.1100480108

Lang W, Chen T, Li X (2016) A new style of urbanization in China: Transformation of urban rural communities. Habitat Int 55:1–9. https://doi.org/10.1016/j.habitatint.2015.10.009

Li WJ, Ali SH, Zhang Q (2007) Property rights and grassland degradation: A study of the Xilingol Pasture, Inner Mongolia, China. J Environ Manage 85:461–470. https://doi.org/10.1016/j.jenvman.2006.10.010

Liu M, Dries L, Heijman W, et al (2015) Tragedy of the commons or tragedy of privatisation? The impact of land tenure reform on grassland condition in Inner Mongolia, China. In: International Conference of Agricultural Economists. pp 9–14

Melillo JM, Richmond TT, Yohe G (2014) Climate change impacts in the United States. US Global Change Research Program Washington, DC

Millenium Ecosystem Assessment (2005) Ecosystems and Human Well-Being: Desertification Synthesis. Island Press Washington, DC

Prestele R, Alexander P, Rounsevell MDA, et al (2016) Hotspots of uncertainty in land-use and land-cover change projections: A global-scale model comparison. Glob Chang Biol 22:3967–3983. https://doi.org/10.1111/gcb.13337

Reid RS, Kruska RL, Muthui N, et al (2000) Land-use and land-cover dynamics in response to changes in climatic, biological and socio-political forces: The case of Southwestern Ethiopia. Landsc Ecol 15:339–355. https://doi.org/10.1023/A:1008177712995

Schneider A, Mertes CM, Tatem AJ, et al (2015) A new urban landscape in East-Southeast Asia, 2000-2010. Environ Res Lett 10:34002. https://doi.org/10.1088/1748-9326/10/3/034002

Sleeter BM, Sohl TL, Loveland TR, et al (2013) Land-cover change in the conterminous United States from 1973 to 2000. Glob Environ Chang 23:733–748. https://doi.org/10.1016/j.gloenvcha.2013.03.006

Zanaga D, Van De Kerchove R, De Keersmaecker W, et al (2021) ESA WorldCover 10 m 2020 v100. Meteosat Second Gener Evapotranspiration 1–27. https://doi.org/10.5281/zenodo.5571936

Chapter A3.9: Conservation Applications - Assessing the spatial relationship between burned area and precipitation

Authors
Harriet Branson and Chelsea Smith

Overview

The purpose of this chapter is to introduce the need for fire and rainfall trend monitoring to inform conservation management practices.

Practical conservation requires an understanding of key environmental factors such as fire and rainfall, which impact the amount of forage and habitat available for a variety of species. This chapter will guide you through how to create fire and rainfall time series and visualize this data. At the end of this chapter, you will be able to present this information in an accessible way on a graph to help inform conservation management practices such as early burning and supplementary feeding.

Learning Outcomes

- Understanding why fire and rainfall trends are useful in conservation management.
- Creating a time series of burned areas.
- Writing a function to map areal mean rainfall calculations over an `ImageCollection`.
- Generating an interactive graph displaying fire and rainfall trends over a 10-year period.

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).
- Summarize an image with `reduceRegion` (Chap. F3.1).
- Write a function and `map` it over an `ImageCollection` (Chap. F4.0).
- Work with CHIRPS rainfall data (Chap. F4.2).
- Mask cloud, cloud shadow, snow/ice, and other undesired pixels (Chap. F4.3).

Introduction to Theory

Globally, biodiversity is under threat from changing climates, habitat loss, and fragmentation. Conservation work to protect, manage and restore ecosystems is vitally important to maintain global biodiversity, which supports climate regulation and a host of other ecosystem services we depend upon (Reddy 2021).

Remote sensing offers a valuable tool to collect, analyze, and display data over an entire ecosystem. Environmental variables such as fire and rainfall have direct impacts on habitat and forage availability. Monitoring of these variables can help inform conservation, such as fire management to reduce the severity of fires and amount of habitat lost (Ribeiro et al. 2021). Monitoring rainfall patterns can help conservationists understand which climate conditions species prefer in their habitats and if the conditions are at risk of changing (Pinto-Ledezma and Cavender-Bares 2021).

In this chapter we will see how using the Earth Engine platform allows conservationists to scope the fire and rainfall conditions for any site using global datasets. The data we visualize is being directly used in real-world conservation by informing fire management, as well as indirectly through the identification of climate conditions to which species are adapted.

Practicum

Section 1. Assess Area of Interest

The first step is to upload and explore the area of interest. Northern Mozambique's Niassa Reserve is home to 40% of the country's entire elephant population, and is a haven for two of Africa's threatened carnivores, lion and wild dog. Fire is a key ecological process in forested savanna ecosystems that are prevalent in the Niassa Reserve, and the knowledge of the fire regime is an important factor in forest fire management. Our area of interest (AOI) has very stark wet and dry seasons; take a look at the satellite imagery basemap that shows the landscape in the wet season.

```
// ** Upload the area of interest ** //
var AOI = ee.Geometry.Polygon([
    [
        [37.72, -11.22],
        [38.49, -11.22],
        [38.49, -12.29],
        [37.72, -12.29]
    ]
]);
Map.centerObject(AOI, 9);
Map.addLayer(AOI, {
    color: 'white'
}, 'Area of interest');
```

Section 2. Load the MODIS Burned Area Dataset

Next, we will use the MCD64A1 dataset, which is a global layer representing burned area at 500 m resolution from 2001 to the present. The layer is also accompanied by a band named `'BurnDate'`, which enables you to disaggregate into daily fire data.

First, we will filter the MODIS `ImageCollection` based on the timespan requirements. We are looking at fire patterns over the past 10 years. The MCD64A1 dataset comes with three main bands, `'BurnDate'`, `'Uncertainty'`, and `'QA'` (quality assurance). In this case, we select only the `'BurnDate'` band, which associates each pixel of burned area with a day-of-year value.

```
// ** MODIS Monthly Burn Area ** //

// Load in the MODIS Monthly Burned Area dataset.
var dataset = ee.ImageCollection('MODIS/006/MCD64A1')
    // Filter based on the timespan requirements.
    .filter(ee.Filter.date('2010-01-01', '2021-12-31'));

// Select the BurnDate band from the images in the collection.
var MODIS_BurnDate = dataset.select('BurnDate');
```

Next we can create the function that will calculate the area of pixels associated with each day. The function is structured so that it will map over each image in the `MODIS_BurnDate` `ImageCollection`. It begins by using the command `pixelArea`, which calculates the area of each pixel in square meters. To limit the calculation to specifically burned areas, we need to use `updateMask` with our input `img` as the variable. Because we want our final area calculation to be in square kilometers instead of square meters, we divide the value by 1,000,000 (1e6).

The `reduceRegion` command gathers the sum of burned area within our input area of interest at the correct scale of the input image (500 m for our `MODIS_BurnDate` collection). After this, the `getNumber` command recalls the area calculation per image per day of year and appends it as a new band `area` on the `ImageCollection`.

Since the MODIS `ImageCollection` has a `'system:time_start'` embedded within each image, we can select the `area` band and continue without the `'BurnDate'`.

```
// A function that will calculate the area of pixels in each image by
date.
var addArea = function(img) {
    var area = ee.Image.pixelArea()
        .updateMask(
            img
        ) // Limit area calculation to areas that have burned data.
        .divide(1e6) // Divide by 1,000,000 for square kilometers.
        .clip(AOI) // Clip to the input geometry.
        .reduceRegion({
            reducer: ee.Reducer.sum(),
            geometry: AOI,
            scale: 500,
            bestEffort: true
        }).getNumber(
            'area'
        ); // Retrieve area from the reduce region calculation.
    // Add a new band to each image in the collection named area.
    return img.addBands(ee.Image(area).rename('area'));
};

// Apply function on image collection.
```

```
var burnDateArea = MODIS_BurnDate.map(addArea);

// Select only the area band as we are using system time for date.
var burnedArea = burnDateArea.select('area');
```

To show the total amount of burned area over the past 10 years, we can plot this on a time-series graph. By defining the `xProperty` with `'system:time_start'` for the date variable on this graph, we can plot the total area burned with time.

```
// Create a chart that shows the total burned area over time.
var burnedAreaChart =
    ui.Chart.image
    .series({
        imageCollection: burnedArea, // Our image collection.
        region: AOI,
        reducer: ee.Reducer.mean(),
        scale: 500,
        xProperty: 'system:time_start' // time
    })
    .setSeriesNames(['Area']) // Label for legend.
    .setOptions({
        title: 'Total monthly area burned in AOI',
        hAxis: {
            title: 'Date', // The x axis label.
            format: 'YYYY', // Years only for date format.
            gridlines: {
                count: 12
            },
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        vAxis: {
            title: 'Total burned area (km²)', // The y-axis label
            maxValue: 2250, // The bounds for y-axis
            minValue: 0,
```

```
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        lineWidth: 1.5,
        colors: ['d74b46'], // The line color
    });
print(burnedAreaChart);
```

Code Checkpoint A39a. The book's repository contains a script that shows what your code should look like at this point.

Question 1. We have calculated the area burned in square kilometers; which line of code would you have to change, and how, to calculate the area burned in hectares?

Question 2. By hovering your mouse over the graph, which month has the greatest area burnt, and does this vary year on year?

Question 3. The most appropriate fire dataset and analysis to use for your study will vary depending on location and scale of analysis. If you wanted to understand the impact of fires on koala habitat in southern Australia, what would be the most appropriate analysis? Hint: Consider the analysis run in Chap. A3.1 and the work by Bonney et al. (2020).

Section 3. Areal Mean Rainfall Time Series

To inform habitat management and further understand fire patterns within northern Mozambique, we have to also consider patterns in rainfall. To do this, we will use the Climate Hazards Group InfraRed Precipitation with Station (CHIRPS) Precipitation dataset. This dataset measures precipitation levels every five days from 1981 to present day at 500 m resolution to make a long-term quasi-global dataset.

First, we will define our temporal range (matching our burned area data) from 2010 to 2021, and set the advancing dates from these years. After this, we create a sequence of years and months that will be used to filter the dataset chronologically.

After setting these parameters, filter the CHIRPS dataset using the start and end date, and sort chronologically in descending order using the same `'system:time_start'`

property used in Sect. 1. Filter the bounds to the AOI, and select the precipitation band from the `ImageCollection`.

```
// Load in the CHIRPS rainfall pentad dataset.
var chirps = ee.ImageCollection('UCSB-CHG/CHIRPS/PENTAD');

// Define the temporal range
var startyear = 2010;
var endyear = 2021;

// Set the advancing dates from the temporal range.
var startdate = ee.Date.fromYMD(startyear, 1, 1);
var enddate = ee.Date.fromYMD(endyear, 12, 31);

// Create a list of years
var years = ee.List.sequence(startyear, endyear);
// Create a list of months
var months = ee.List.sequence(1, 12);

// Filter the dataset based on the temporal range.
var Pchirps = chirps.filterDate(startdate, enddate)
    .sort('system:time_start',
        false) // Sort chronologically in descending order.
    .filterBounds(AOI) // Filter to AOI
    .select('precipitation'); // Select precipitation band
```

Once the dataset has been filtered, we can calculate the monthly precipitation using a function. The function maps the input, `y`, over the list of years generated above. The function then returns the total precipitation for the month, alongside a date variable and the `'system:time_start'` property. The command `millis` is used to keep the system number that refers to the date collected.

Print the `ImageCollection` for checking.

```
// Calculate the precipitation per month.
var MonthlyRainfall = ee.ImageCollection.fromImages(
    years.map(function(
```

```
        y
    ) { // Using the list of years based on temporal range.
        return months.map(function(m) {
            var w = Pchirps.filter(ee.Filter
                    .calendarRange(y, y, 'year'))
                .filter(ee.Filter.calendarRange(m, m,
                    'month'))
                .sum(); // Calculating the sum for the month
            return w.set('year', y)
                .set('month', m)
                .set('system:time_start', ee.Date
                    .fromYMD(y, m, 1).millis()
                ) // Use millis to keep the system time number.
                .set('date', ee.Date.fromYMD(y, m,
                    1));
        });
    }).flatten());
// Print the image collection.
print('Monthly Precipitation Image Collection', MonthlyRainfall);
```

Once the monthly precipitation levels have been calculated, we can use a reducer to calculate the mean total rainfall across our AOI, also known as the areal mean rainfall (AMR), and plot these on a time series graph. This process is very similar to the burned area chart in Sect. 2.

```
// ** Chart: CHIRPS Precipitation ** //

// Create a chart displaying monthly rainfall over a temporal range.
var monthlyRainfallChart =
    ui.Chart.image
    .series({
        imageCollection: MonthlyRainfall.select(
            'precipitation'), // Select precipitation band
        region: AOI,
        reducer: ee.Reducer
            .mean(), // Use mean reducer to calculate AMR
        scale: 500,
```

```
        xProperty: 'system:time_start' // Use system time start for
x-axis
    })
    .setSeriesNames(['Precipitation']) // /The label legend
    .setOptions({
        title: 'Total monthly precipitation in AOI', // Add title
        hAxis: {
            title: 'Date',
            format: 'YYYY', // Year only date format
            gridlines: {
                count: 12
            },
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        vAxis: {
            title: 'Precipitation (mm)', // The y-axis label
            maxValue: 450, // The bounds for y-axis
            minValue: 0,
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        lineWidth: 1.5,
        colors: ['4f5ebd'],
    });
print(monthlyRainfallChart);
```

Print the monthly rainfall chart. Using the chart, we can see which months receive rainfall, and can categorize these as the wet season. With data from the chart, we will categorize any month that receives over 5 mm of rainfall as a wet season month. We can then calculate total seasonal rainfall across our AOI by aggregating wet season months together.

```
// 2010/2011 wet season total
var year = 2010; // Adjust year
var startDate = ee.Date.fromYMD(year, 11, 1); // Adjust months/days
var endDate = ee.Date.fromYMD(year + 1, 5, 31); // Adjust months/days
var filtered = chirps
    .filter(ee.Filter.date(startDate, endDate));
var Rains10_11Total = filtered.reduce(ee.Reducer.sum()).clip(AOI);

// 2011/2012 wet season total
var year = 2011; // Adjust year
var startDate = ee.Date.fromYMD(year, 11, 1); // Adjust months/days
var endDate = ee.Date.fromYMD(year + 1, 5, 31); // Adjust months/days
var filtered = chirps
    .filter(ee.Filter.date(startDate, endDate));
var Rains11_12Total = filtered.reduce(ee.Reducer.sum()).clip(AOI);
```

Question 4. Classify the remaining wet seasons using the areal mean rainfall chart, and calculate the total seasonal rainfall over the AOI using the code you have just learned, to calculate wet seasons in 2010–2011 and 2011–2012.

Question 5. We have created a 10-year time series. Is this a long enough time period to start to consider changes in rainfall patterns?

Code Checkpoint A39b. The book's repository contains a script that shows what your code should look like at this point.

Section 4. Visualizing Fire and Rainfall Time Series

Now that we have visualized patterns in both burned area and precipitation separately, it is important to assess these patterns together. To do this, we need to combine the two image collections in order to plot them on the same graph.

Once they are combined, we can begin creating a multi-variable time-series chart. By keeping the `'system:time_start'` property on both sets of image collections, we are able to easily plot each variable temporally. Following this, we set both series' names and set `interpolateNulls` to true to provide continuous precipitation data that can be plotted alongside the near daily burned area data.

To enable two y-axes, we need to set some series parameters. By defining our `targetAxisIndex` as 0 and 1, we can then set our `vAxes` to match, using two sets of parameters with different labels and different bounds for ease of plotting. This is useful since the burned area and precipitation datasets have different minimum and maximum values, so it would be difficult to analyze on the same axis.

You can print the chart to the **Console** for export if necessary, but for interactivity we will add it to the map later.

```
// ** Combine: CHIRPS Average Rainfall & MODIS Monthly Burn ** //

// Combine the two image collections for joint analysis
var bpMerged = burnedArea.merge(MonthlyRainfall);
print('Merged image collection', bpMerged);

// ** Chart: CHIRPS Average Rainfall & MODIS Monthly Burn ** //
// Plot the two time series on a graph
var bpChart =
    ui.Chart.image.series({
        imageCollection: bpMerged, // The merged image collection
        region: AOI,
        reducer: ee.Reducer.mean(),
        scale: 500,
        xProperty: 'system:time_start' // Use system time start for
synchronous plotting
    })
    .setSeriesNames(['Burned Area', 'Precipitation']) // Label series
    .setChartType('LineChart') // Define chart type
    .setOptions({
        title: 'Relationship between burned area and rainfall in
Chuilexi',
        interpolateNulls: true, // Interpolate nulls to provide
continuous data
        series: { // Use two sets of series with a target axis to
create the two y-axes needed for plotting
            0: { // 0 and 1 reference the vAxes settings below
                targetAxisIndex: 0,
```

```
                type: 'line',
                lineWidth: 1.5,
                color: 'd74b46'
            },
            1: {
                targetAxisIndex: 1,
                type: 'line',
                lineWidth: 1.5,
                color: '4f5ebd'
            },
        },
        hAxis: {
            title: 'Date',
            format: 'YYYY',
            gridlines: {
                count: 12
            },
            titleTextStyle: {
                italic: false,
                bold: true
            }
        },
        vAxes: {
            0: {
                title: 'Burned area (km²)', // Label left-hand y-axis
                baseline: 0,
                viewWindow: {
                    min: 0
                },
                titleTextStyle: {
                    italic: false,
                    bold: true
                }
            },
            1: {
                title: 'Precipitation (mm)', // Label right-hand
y-axis
```

```
                baseline: 0,
                viewWindow: {
                    min: 0
                },
                titleTextStyle: {
                    italic: false,
                    bold: true
                }
            },
        },
        curveType: 'function' // For smoothing
    });
bpChart.style().set({
    position: 'bottom-right',
    width: '492px',
    height: '300px'
});
```

Once we have created our final chart that displays burned area and precipitation, we can build some legends on the map for the spatial data. Using two different functions, we can create a horizontal legend with a set gradient palette and custom markers that correspond to the precipitation data.

The burned area legend is simpler in that we are creating a square of red to indicate that any pixel marked red on the image was burned at that particular time point.

```
// ** Legend: Rainfall ** //
var rain_palette = ['ffffcc', '#a1dab4', '#41b6c4', '#2c7fb8',
    '#253494'
];

function ColorBar(rain_palette) {
    return ui.Thumbnail({
        image: ee.Image.pixelLonLat().select(0),
        params: {
            bbox: [0, 0, 1, 0.1],
            dimensions: '300x15',
```

```
            format: 'png',
            min: 0,
            max: 1,
            palette: rain_palette,
        },
        style: {
            stretch: 'horizontal',
            margin: '0px 22px'
        },
    });
}

function makeRainLegend(lowLine, midLine, highLine, lowText, midText,
    highText, palette) {
    var labelheader = ui.Label(
        'Total precipitation in wet season (mm)', {
            margin: '5px 17px',
            textAlign: 'center',
            stretch: 'horizontal',
            fontWeight: 'bold'
        });
    var labelLines = ui.Panel(
        [
            ui.Label(lowLine, {
                margin: '-4px 21px'
            }),
            ui.Label(midLine, {
                margin: '-4px 0px',
                textAlign: 'center',
                stretch: 'horizontal'
            }),
            ui.Label(highLine, {
                margin: '-4px 21px'
            })
        ],
        ui.Panel.Layout.flow('horizontal'));
    var labelPanel = ui.Panel(
```

```
        [
            ui.Label(lowText, {
                margin: '0px 14.5px'
            }),
            ui.Label(midText, {
                margin: '0px 0px',
                textAlign: 'center',
                stretch: 'horizontal'
            }),
            ui.Label(highText, {
                margin: '0px 1px'
            })
        ],
        ui.Panel.Layout.flow('horizontal'));
    return ui.Panel({
        widgets: [labelheader, ColorBar(rain_palette),
            labelLines, labelPanel
        ],
        style: {
            position: 'bottom-left'
        }
    });
}
Map.add(makeRainLegend('|', '|', '|', '0', '250', '500', ['#ffffcc',
    '#a1dab4', '#41b6c4', '#2c7fb8', '#253494'
]));

// ** Legend: Burned area ** //
var burnLegend = ui.Panel({
    style: {
        position: 'top-left',
        padding: '8px 15px'
    }
});

var makeRow = function(color, name) {
    var colorBox = ui.Label({
```

```
        style: {
            backgroundColor: '#' + color,
            padding: '10px',
            margin: '0 10px 0 0'
        }
    });
    var description = ui.Label({
        value: name,
        style: {
            margin: 'o o 6px 6px'
        }
    });
    return ui.Panel({
        widgets: [colorBox, description],
        layout: ui.Panel.Layout.Flow('horizontal')
    });
};

var burnPalette = ['FF0000'];
var names = ['Burned area'];
for (var i = 0; i < 1; i++) {
    burnLegend.add(makeRow(burnPalette[i], names[i]));
}
Map.add(burnLegend);
```

Now that we have our legends complete, we can add our double variable time-series chart to the map, and center the map on our area of interest.

```
Map.centerObject(AOI, 9); // Centre the map on the AOI
Map.add(
    bpChart
); // Add the merged burned area & precipitation chart to the map
```

While visualizing the data on a static chart is useful for further analysis and identifying patterns, it is also useful to see the spatial data corresponding to a particular date or fire. To do this, we can add an interactive element to the chart. If you click a particular point

on the chart, it will reveal the corresponding image for burned area and precipitation on the map.

To do this, we need to create a function that uses the burned area and precipitation chart (`bpChart`) and executes `onClick`, displaying the relevant data based on the input values. By utilizing the `'system:time_start'` property, we can query the date, which will then be used to identify the first image (`first`) that appears within the list of images, within the area of interest. We can then format the legend box within the map so that it shows the date and time of the layer selected.

Following this, we can set the symbology of each layer (burned area in red, and the precipitation color gradient indicated in the legend settings), and display the relevant layer with the date text string on the map.

```
// ** Chart: Adding an interactive query ** //

// Add a function where if you click on a point in the map it displays
the burned area and rainfall for that date
bpChart.onClick(function(xValue, yValue, seriesName) {
    if (!xValue) return;
    // Show layer for date selected on the chart
    var equalDate = ee.Filter.equals('system:time_start',
        xValue);
    // Search for the layer in the image collection that links to the
selected date
    var classificationB = ee.Image(MODIS_BurnDate.filter(
        equalDate).first()).clip(AOI).select('BurnDate');
    var classificationR = ee.Image(MonthlyRainfall.filter(
        equalDate).first()).clip(AOI).select(
        'precipitation');
    var burnImage = ee.Image(MODIS_BurnDate.filter(equalDate)
        .first());
    var date_string = new Date(xValue).toLocaleString(
        'en-EN', {
            dateStyle: 'full'
        });
    var rainImage = ee.Image(MonthlyRainfall.filter(equalDate)
```

```
        .first());
    var date_stringR = new Date(xValue).toLocaleString(
        'en-EN', {
            dateStyle: 'full'
        });
    // Reset the map layers each time a new date is clicked
    Map.layers().reset([classificationB]);
    Map.layers().reset([classificationR]);
    var visParamsBurnLayer = { // Visualisation for burned area
        min: 0,
        max: 365,
        palette: ['red']
    };
    var visParamsRainLayer = { // Visualisation for rain
        min: 0,
        max: 450,
        palette: ['#ffffcc', '#a1dab4', '#41b6c4',
            '#2c7fb8', '#253494'
        ]
    };
    // Add the layers to the map
    Map.addLayer(classificationR, visParamsRainLayer,
        'Total monthly rainfall on [' + date_string + ']');
    Map.addLayer(classificationB, visParamsBurnLayer,
        'Burned area on [' + date_string + ']');
});
```

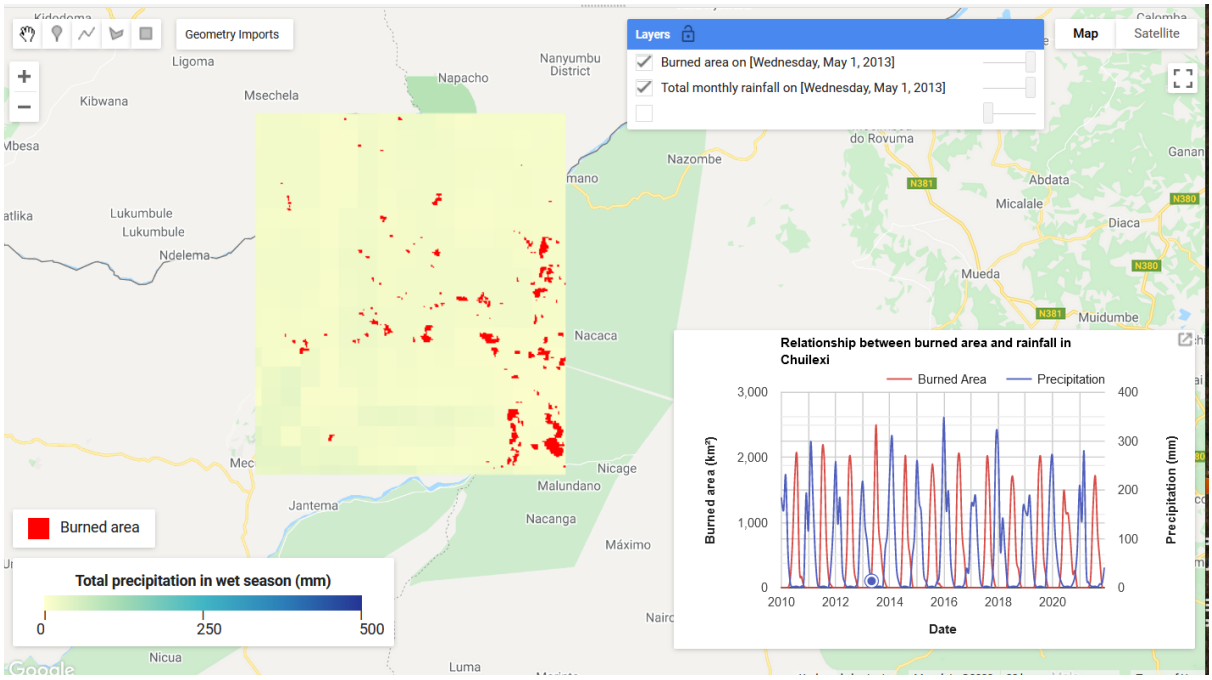


Fig. A3.9.1 The final result of the code once you have clicked on the chart to select a month to display, here showing the burned area and total precipitation at the beginning of the dry season in May 2021

Code Checkpoint A39c. The book's repository contains a script that shows what your code should look like at this point.

Question 6. Considering the importance of environmental variables in conservation work, what other datasets and common earth observation analysis could provide data to inform conservation management?

Synthesis

In this chapter, you have learned how to create a time series of burned area, using the global MODIS burned area product. You also calculated areal mean rainfall using the CHIRPS dataset, and graphed total seasonal rainfall using your time-series chart. Additionally, you can now merge two image collections to show two variables on one interactive chart in the Earth Engine map.

With the code you have learned in this chapter, you can now create area burned and areal mean rainfall time series for your own region of interest anywhere in the world. Recreate the analysis in a different environment, and consider extending the time series over a longer period; can you detect any trends?

Monitoring vegetation is also important in conservation and can be significantly impacted by fire and rainfall trends. Try to modify the code you have learned to calculate areal mean NDVI from the NOAA CDR AVHRR daily NDVI dataset. Can you add this as another variable in the interactive chart?

Conclusion

In this chapter we understand and map the dynamic relationship between fire and rainfall, and how this can influence conservation action and land management needs. We began by mapping burned areas using MODIS Burned Area Monthly Global 500 m to understand how much of, and where, the landscape is affected by burning. Following this, we understood how to access rainfall data and calculate areal mean rainfall, plotting this on a graph to understand changes and patterns over time. By combining the burned area and rainfall data, we can see how rainfall (or lack thereof) can exacerbate burning, and begin to spot patterns in the landscape. This analysis, which would often be undertaken in the field or by hand using satellite imagery, is made accessible via Earth Engine—not only because we can perform this on a workstation that only requires an internet connection rather than computing power, but also because Earth Engine generates quick, consistent results that can inform conservation management practices.

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

Bonney MT, He Y, Myint SW (2020) Contextualizing the 2019–20 kangaroo island bushfires: Quantifying landscape-level influences on past severity and recovery with Landsat and Google Earth Engine. Remote Sens 12:1–32. https://doi.org/10.3390/rs12233942

Holden ZA, Swanson A, Luce CH, et al (2018) Decreasing fire season precipitation increased recent Western US forest wildfire activity. Proc Natl Acad Sci USA 115:E8349–E8357. https://doi.org/10.1073/pnas.1802316115

Nhongo E, Fontana D, Guasselli L (2020) Spatio-temporal patterns of wildfires in the Niassa Reserve –Mozambique, using remote sensing data. bioRxiv 1–7. https://doi.org/10.1101/2020.01.16.908780

Pinto-Ledezma JN, Cavender-Bares J (2021) Predicting species distributions and community composition using satellite remote sensing predictors. Sci Rep 11:1–12. https://doi.org/10.1038/s41598-021-96047-7

Reddy CS (2021) Remote sensing of biodiversity: What to measure and monitor from space to species? Biodivers Conserv 30:2617–2631. https://doi.org/10.1007/s10531-021-02216-5

Ribeiro NS, Armstrong AH, Fischer R, et al (2021) Prediction of forest parameters and carbon accounting under different fire regimes in Miombo woodlands, Niassa Special Reserve, Northern Mozambique. For Policy Econ 133:102625. https://doi.org/10.1016/j.forpol.2021.102625

Chapter A3.10: Conservation II - Assessing Agricultural Intensification Near Protected Areas

Authors
Pradeep Koulgi and MD Madhusudan

Overview

Protected Areas (PAs) in many densely populated tropical regions are often small in area, and are enormously influenced by the broader production landscapes in which they are found. Changes in the agricultural matrix surrounding a PA can have a profound impact on the PA's wildlife and on neighboring resident human communities. In this chapter, we will examine greening trend changes in the exteriors of 186 PAs in Western India from 2000 to 2021 using MODIS Terra vegetation indices, a Sen's slope linear trend estimator, and other summary techniques available in Earth Engine. We will use these techniques to investigate how these greening trends are distributed in relation to the precipitation regimes of a given PA site.

Learning Outcomes

- Computing a metric of a monotonic trend (Sen's slope) in dry-season pixel greenness for each pixel.
- Inferring the nature and intensity of change in agricultural practice based on the trend metric.
- Exploring the relationship between changes in vegetation greenness and ecosystem type as determined by average annual precipitation.

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).
- Calculate and interpret vegetation indices (Chap. F2.0)
- Use reducers to implement linear regression between image bands (Chap. F3.0).
- Write a function and map it over an `ImageCollection` (Chap. F4.0).
- Map an annual reducer across multiple years (Chap. F4.0, Chap. F4.1).

- Write a function and map it over a collection (Chap. F4.0).
- 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

In many regions of the densely populated tropics, Protected Areas (PAs) are often small, and their interfaces with production landscapes are sharp. As a result, changes in the agriculture surrounding a PA can have a profound impact on the PA's wildlife, as well as on their interactions with local human communities. For example, across India's Gangetic Plains, increased crop irrigation via groundwater exploitation has enabled its traditional rainfed agriculture with long fallow periods and annual crops (cereals, legumes, and oilseeds) to be to be replaced by year-round agriculture (Chen et al 2019, Maina et al 2022). This intensification can have subtle yet significant changes in the landscape in terms of spatial regimes of primary productivity, and to wildlife in terms of both dietary and habitat resources. These changes, in turn, can influence wildlife interactions with people (Kumar et al. 2018).

In this chapter, we describe greening trends in vegetation in the exteriors of PAs in Western India, and ask how these greening trends are distributed in relation to the precipitation regimes of a given PA site. Based on our experience and our reading of literature pertaining to wildlife in India, we hypothesize that as crop irrigation via groundwater extraction increases, farming in the moisture-limited semi-arid tracts of India becomes more independent of seasonal differences, and their constituent habitats become structurally more complex. These changes could help create novel anthropogenic habitats that can be accessed by adaptable wildlife species living in PAs, such as leopards and elephants (Athreya and Ghoshal 2010, Odden et al. 2014). While such changes to habitat structure and resource availability could improve functional connectivity between PAs (e.g., Rodrigues et al. 2022), intensification can also lead to a greater overlap between humans and wildlife, possibly leading to greater conflict over crops and livestock (Kumar et al. 2018). The potential trade-offs involved between conservation and conflict imply that it is crucial to assess and map land use changes around PAs over time, and eventually to relate this to changes in wildlife distribution and behavior, for which this chapter provides a framework.

Practicum

Section 1. Initializing Parameters

The Normalized Difference Vegetation Index (NDVI) estimates vegetation greenness; the MODIS Terra vegetation indices products, MOD13Q1.006, provides historical 16-day composites of it at a global span and 250 m per pixel resolution. This dataset is used here to estimate a monotonic trend in annual changes of dry-season vegetation greenness around a set of PAs in India. Various parameters used in the script are initialized first.

Section 1.1 Annual dry-season Maximum NDVI Calculation

The MODIS vegetation indices dataset MOD13Q1.006 is initialized, along with the NDVI band name and relevant scale values. The dataset starts from the year 2000 and extends to the present. For this analysis, we use data from 2000 to 2021 as the sequence of full years the data are available for. The annual dry season in much of India occurs over three to five months, starting roughly in January. Here, a period of 90 days (spanning January through March) starting on the first day of each year is taken to be the dry season. Convenient names are chosen for bands holding NDVI and time values for regression analysis.

```
var modis_veg = ee.ImageCollection('MODIS/006/MOD13Q1');
var ndviBandName = 'NDVI';
var ndviValuesScaling = 0.0001;
var modisVegScale = 250; // meters
var maxNDVIBandname = 'max_dryseason_ndvi';
var yearTimestampBandname = 'year';
var years = ee.List.sequence(2000, 2021, 1);
var drySeasonStart_doy = 1;
var drySeasonEnd_doy = 90;
```

Section 1.2 Boundaries of PAs of Interest

In the code below, a set of PAs in the western part of India is identified for study. This collection of PAs in the western belt of India spans a wide range of rainfall regimes, from very moist areas in the southern Western Ghats to very arid regions in the Thar Desert of Rajasthan in the north, with diverse rainfall regimes in between. This belt is home to a diverse array of forests and savanna ecosystems, ranging from moist ecosystems to

semi-arid and arid ecosystems. The landscapes in this belt also have a complex, diverse intervening matrix of natural and intensively human-utilized lands making up wildlife corridors. The size of the buffer area around each PA is also defined.

```
var paBoundaries = ee.FeatureCollection(
    'projects/gee-book/assets/A3-10/IndiaMainlandPAs');
var boundaryBufferWidth = 5000; // meters
var bufferingMaxError = 30; // meters
// Choose PAs in only the western states
var western_states = [
    'Rajasthan', 'Gujarat', 'Madhya Pradesh',
    'Maharashtra', 'Goa', 'Karnataka', 'Kerala'
];
var western_pas = paBoundaries
    .filter(ee.Filter.inList('STATE', western_states));
```

Section 1.3 Regression Analysis

The Sen's slope linear trend estimator (Sen, 1968) is a non-parametric estimator useful for monotonic trend estimation applications using remotely sensed indices. Its suitability for applications with remotely sensed indices comes from a key difference when compared to linear regression using the least squares estimator: while linear regression assumes that the regression residuals are normally distributed, Sen's slope estimator makes no such assumptions about the statistical structure of the data it is being applied on. The reducer for this Sen's slope regression is built into the Earth Engine API, and in the code below its x and y variables are initialized.

```
var regressionReducer = ee.Reducer.sensSlope();
var regressionX = yearTimestampBandname;
var regressionY = maxNDVIBandname;
```

Section 1.4 Surface Water Layer to Mask Water Pixels from Assessment

NDVI values of pixels spanning water bodies tend to be highly noisy and can mislead trend analyses. Masking out all pixels that were ever water during the period of interest can mitigate this problem. The European Union's Joint Research Centre mapped monthly surface water from 1984 to 2021, including the maximum water extent detected during the period.

```
// Selects pixels where water has ever been detected between 1984 and
2021
var surfaceWaterExtent = ee.Image('JRC/GSW1_3/GlobalSurfaceWater')
    .select('max_extent');
```

Section 1.5 Average Annual Precipitation Layer

To relate the estimated trends in vegetation greenness to rainfall regime, we use WorldClim's estimated long-term average annual precipitation data.

```
var rainfall = ee.Image('WORLDCLIM/V1/BIO').select('bio12');
```

Section 1.6 Visualization and Saving Parameters

The estimated metric of change in vegetation greenness can be visualized as a raster on the map. Additionally, the relationship between changes in vegetation greenness and precipitation around each PA can be charted using a scatter plot. The visualization parameters for these are defined.

```
var regressionResultVisParams = {
    min: -3,
    max: 3,
    palette: ['ff8202', 'ffffff', '356e02']
};
var regressionSummaryChartingOptions = {
    title: 'Yearly change in dry-season vegetation greenness ' +
        'in PA buffers in relation to average annual rainfall',
    hAxis: {
        title: 'Annual Precipitation'
    },
    vAxis: {
        title: 'Median % yearly change in vegetation greenness ' +
            'in 5 km buffer'
    },
    series: {
```

```
        0: {
            visibleInLegend: false
        }
    },
};
```

Code Checkpoint A310a. The book's repository contains a script that shows what your code should look like at this point.

Section 2. Raster Processing for Change Analysis

The 16-day composite NDVI rasters are filtered, processed, and reduced according to parameters appropriately defined and initialized above to generate a raster of percentage annual change in greenness as a metric for vegetation change. A summary of this for each PA buffer is also calculated. Finally, these results are visualized.

Section 2.1 Annual Dry-season Maxima of NDVI

The source MODIS vegetation indices dataset is first filtered to the dry-season days for all years and the NDVI band selected. From this `ImageCollection` containing only dry-season observations for each year, maximum dry-season NDVI is calculated. This represents the vegetation at its greenest condition within the dry season each year. The maximum NDVI is the least likely to be influenced by episodic changes, such as fires that occur during this dry period. An image band containing the year value is also created and combined with the maximum NDVI, in preparation for time versus NDVI regression analysis in the next step. This is accomplished through defining a function `annualDrySeasonMaximumNDVIAndTime` to do this for each year, and then mapping that function over all the years in our period of interest.

```
function annualDrySeasonMaximumNDVIAndTime(y) {
    // Convert year y to a date object
    var yDate = ee.Date.fromYMD(y, 1, 1);
    // Calculate max NDVI for year y
    var yMaxNdvi = drySeasonNdviColl
        // Filter to year y
        .filter(ee.Filter.date(yDate, yDate.advance(1, 'year')))
        // Compute max value
        .max()
```

```
        // Apply appropriate scale, as per the dataset's
        // technical description for NDVI band.
        .multiply(ndviValuesScaling)
        // rename the band to be more comprehensible
        .rename(maxNDVIBandname);
    // Create an image with constant value y, to be used in
regression. Name it something comprehensible.
    // Name it something comprehensible.
    var yTime = ee.Image.constant(y).int().rename(
        yearTimestampBandname);
    // Combine the two images into a single 2-band image, and return
    return ee.Image.cat([yMaxNdvi, yTime]).set('year', y);
}

// Create a collection of annual dry season maxima
// for the years of interest.  Select the NDVI band and
// filter to the collection of dry season observations.
var drySeasonNdviColl = modis_veg.select([ndviBandName])
    .filter(ee.Filter.calendarRange(drySeasonStart_doy,
        drySeasonEnd_doy, 'day_of_year'));
// For each year of interest, calculate the NDVI maxima and create a
corresponding time band
var dryseason_coll = ee.ImageCollection.fromImages(
    years.map(annualDrySeasonMaximumNDVIAndTime)
);
```

Section 2.2 Annual Regression to Estimate Average Yearly Change in Greenness

The dry-season maximum NDVI collection with time band is reduced with a Sen's slope regression reducer to estimate the linear rate of change of dry-season maximum greenness for the years from 2000 to 2021. Be sure to use the `select` operation to select the x and y variables for regression, in that order, as expected by the `ee.Reducer.sensSlope`. The resulting image will have a slope and an offset band, which are the slope and y-intercept of the trend estimation, respectively.

The values in the slope band are the pixel-wise annual rate of change in maximum dry-season NDVI values. The magnitude of these values signifies the intensity of change

over the years, and their sign indicates whether the change manifested as greening (positive sign) or browning (negative sign) of the vegetation.

```
var ss = dryseason_coll.select([regressionX, regressionY]).reduce(
    regressionReducer);

// Mask surface water from vegetation change image
var ss = ss.updateMask(surfaceWaterExtent.eq(0));
```

Section 2.3 Summarize Estimates of Change in Buffer Regions of PAs of Interest

In order to investigate the vegetation greening and browning changes within the buffer areas of our PAs of interest, buffer regions have to be defined. This can be done by computing the geometry 'difference" between the buffered version of each PA and the PA itself. A function `extractBufferRegion` is defined to perform this for each PA feature, and that function is mapped over the feature collection with boundaries of all the PAs of interest.

```
function extractBufferRegion(pa) {
    //reduce vertices in PA boundary
    pa = pa.simplify({
        maxError: 100
    });
    // Extend boundary into its buffer
    var withBuffer = pa.buffer(boundaryBufferWidth,
        bufferingMaxError);
    // Compute the buffer-only region by "subtracting" boundary with
buffer from boundary
    // Subtracting the whole set of boundaries eliminates inclusion of
forests from adjacent PAs into buffers.
    var bufferOnly = withBuffer.difference(paBoundaries.geometry());

    return bufferOnly;
}

// Create buffer regions of PAs
var pa_buff = western_pas.map(extractBufferRegion);
```

The greenness condition of vegetation in the dry season is inherently variable in large diverse landscapes like around the PAs of choice here, and it depends strongly on the type of vegetation itself, such as deciduous or evergreen tree cover, grasslands, shrublands, and croplands. In order to take this inherent variability into account, the raw rate of change value can be converted into a percent rate of change by normalizing the change value against a baseline value. Here, the maximum year 2000 dry-season NDVI is used as the baseline in each pixel. The slope values are normalized using this baseline and converted to percent values. This can be visualized on the map.

In order to understand the overall picture at a PA buffer-region level, the median of percent rate of vegetation change is calculated for every PA buffer. And to relate this to the amount of rainfall in a buffer region, the corresponding median of average annual rainfall over the region is also calculated. These can be visualized in a chart.

```
// Normalize the metric of NDVI change to a baseline (dry-season max
NDVI in the very first year)
var baselineNdvi = dryseason_coll.select([maxNDVIBandname]).filter(ee
    .Filter.eq('year', years.getNumber(0))).first();
var stats = ss.select('slope').divide(baselineNdvi).multiply(100)
    .rename('vegchange');

// Combine it with average annual rainfall data
stats = stats.addBands(rainfall.rename('rainfall'));

// Calculate mean of change metric over buffer regions of each PA of
interest
var paBufferwiseMedianChange = stats.reduceRegions({
    collection: pa_buff,
    reducer: ee.Reducer.median(),
    scale: 1000,
    tileScale: 16
});
```

Code Checkpoint A310b. The book's repository contains a script that shows what your code should look like at this point.

Section 3. Visualizing Results

The degree of vegetation change, with a chosen palette, is visualized as a map (Fig. A3.10.1), and the PA buffer-wise median value is charted as a scatter plot (Fig. A3.10.2). In the map, green color suggests significant vegetation greening, brown color suggests significant vegetation browning, and white color suggests no detectable change, as chosen in the palette. The vegetation in large areas in the surroundings of Nauradehi Wildlife Sanctuary, as highlighted by Fig. A3.10.1, appear to have experienced greening or no detectable change. PA buffer-wise summaries showing moderate to high positive values indicate that, across the buffers of the PAs, there appears to have been a greater extent of greening, rather than browning, of vegetation cover.

The degree of vegetation change is also arranged by amount of rainfall in a chart, to visualize how buffer regions of PAs in moist regions fare relative to those in arid regions.

```
var medianChangeChart = ui.Chart.feature.byFeature({
    features: paBufferwiseMedianChange,
    xProperty: 'rainfall',
    yProperties: ['vegchange']
}).setOptions(regressionSummaryChartingOptions).setChartType(
    'ScatterChart');
print(medianChangeChart);

Map.centerObject(western_pas, 9);
Map.setCenter(79.2205, 23.3991, 9);
Map.setOptions('SATELLITE');
Map.addLayer(stats.select('vegchange').clipToCollection(pa_buff),
    regressionResultVisParams, 'yearly % change');
Map.addLayer(western_pas, {}, 'Western PAs');
```

Code Checkpoint A310c. The book's repository contains a script that shows what your code should look like at this point.

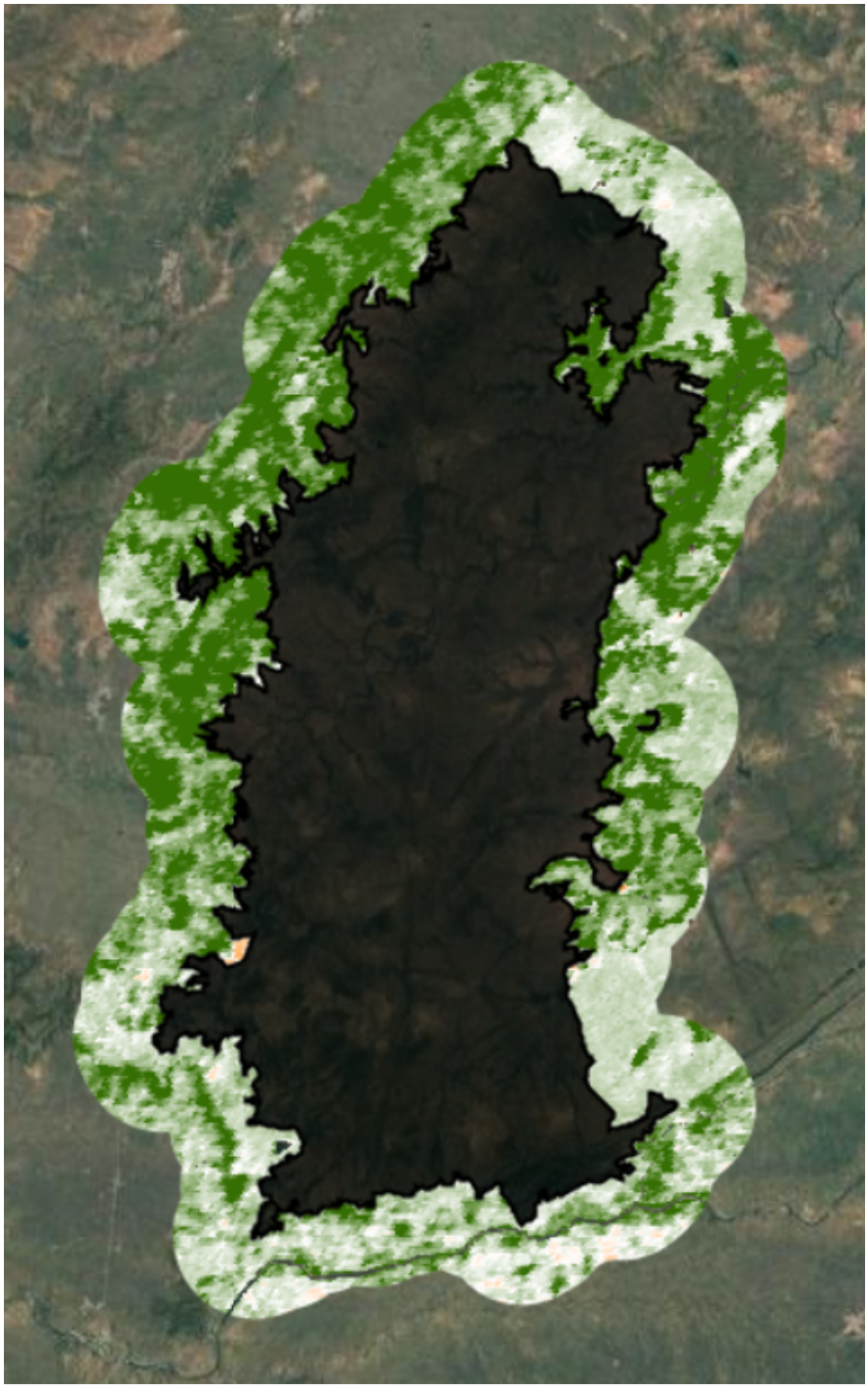


Fig. A3.10.1 Map of percent vegetation greenness change for 2000–2021 in a 5-km buffer area around Nauradehi Wildlife Sanctuary (polygon in black shade), India. Note the relative absence of vegetation browning.

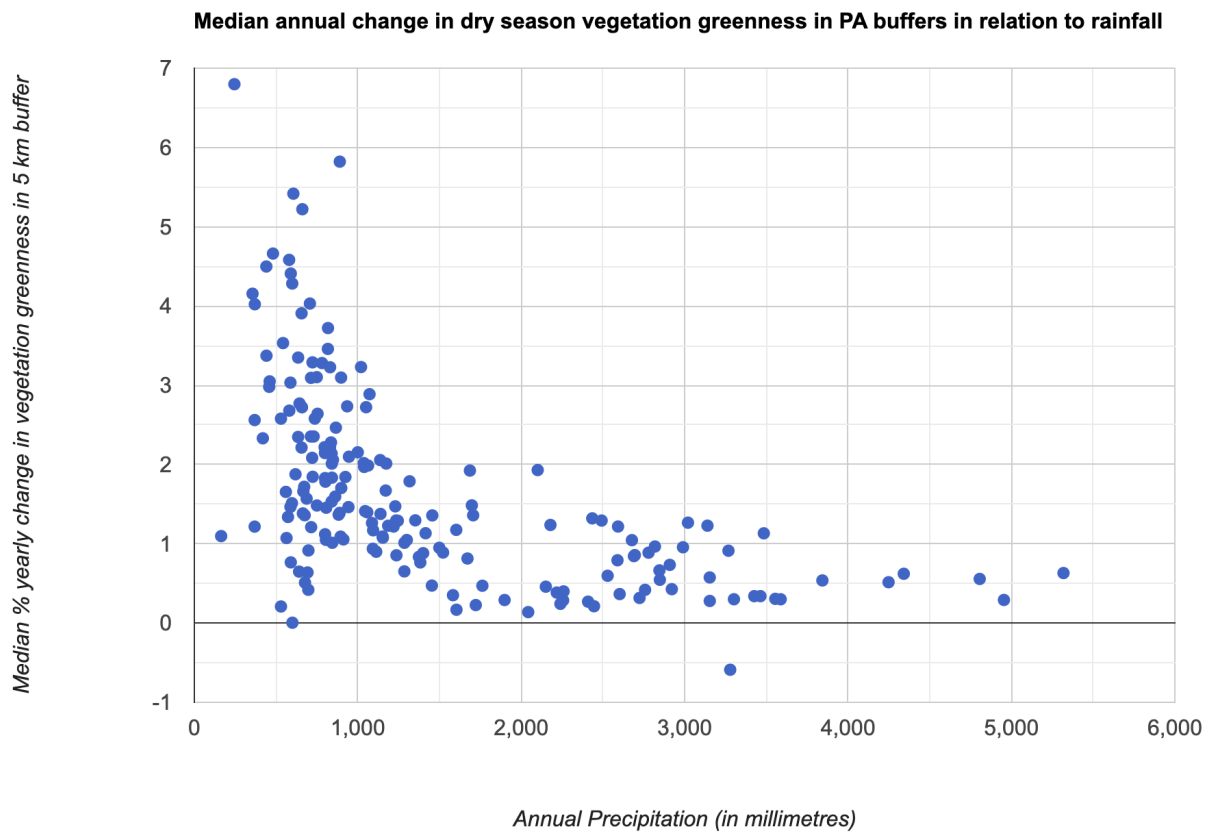


Fig. A3.10.2 Median yearly percent change in vegetation greenness observed in 5-km exterior buffers of PAs in India (N=186), arranged along a precipitation gradient. Greening is denoted by a positive percent change in Sen's slope of dry-season maximum NDVI between 2000 and 2021. The greatest greening extent (>2%) is observed in PAs that fall in the semi-arid zone (annual precipitation ≤ 1000 mm).

Question 1. Create a map of percent vegetation greenness change for a different PA than the one shown in Fig. A3.10.1. How does it compare with the map made for Nauradehi Wildlife Sanctuary?

Question 2. We used MODIS Terra vegetation indices for our greening trend analysis in the previous example. Can you alter the analysis to calculate an NDVI time series from Landsat imagery for one PA? How do you think the results would change with Landsat's 30 m spatial resolution instead of 250 m from MODIS?

Question 3. Can you repeat this analysis, dividing the time period of 21 years into two approximate halves, and try to determine whether the slope of vegetation greening observed has been similar across the two periods, or whether greening has accelerated or decelerated between the two time periods?

Synthesis

Assignment 1. In this chapter, you learned how to estimate greening trends surrounding PAs in Western India, a densely populated tropical region. In your opinion, how generalizable are these observed patterns across space and time?

Assignment 2. Perform this same PA analysis in another region in India. Then, perform the same analysis in a tropical region in another part of the world. How do the results compare with ours from Western India?

Conclusion

In this chapter, we estimated greening trends in Western India PAs using MODIS Terra vegetation indices, Sen's slope, and reducer functions. We demonstrated that the immediate exterior buffers of most PAs (184 out of 186) show a positive—i.e., greening—trend in their vegetation cover over the two-decade time frame between 2000 and 2021. Note, however, that in this example we do not estimate the significance level of the greening trend values modeled in our analyses. While there is considerable variation in the greening extent seen across these PA buffers, we do see clearly that PAs that fall within the semi-arid zone (annual precipitation $\leq$ 1000 mm) seem to show the greatest extent of greening (Fig. A3.10.2). Such a change in the land cover characteristics of the matrix surrounding PAs in the semi-arid zone suggests that they are more likely to be sites where the distribution of wildlife, especially of more adaptable species, could show expansion into the agricultural landscape beyond PA boundaries, as well as greater rates of conflict with humans over crops and livestock. Corroborating this, of course, requires ground survey data on wildlife, which were unavailable in this example.

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

Chen C, Park T, Wang X, et al (2019) China and India lead in greening of the world through land-use management. Nat Sustain 2:122–129. https://doi.org/10.1038/s41893-019-0220-7

Kumar MA, Vijayakrishnan S, Singh M (2018) Whose habitat is it anyway? Role of natural and anthropogenic habitats in conservation of charismatic species. Trop Conserv Sci 11:1940082918788451. https://doi.org/10.1177/1940082918788451

Lenin J (2010) Sugarcane leopards. In: Current Conservation, Special: Wildlife-Human Conflict. https://www.currentconservation.org/sugarcane-leopards/

Maina FZ, Kumar SV, Albergel C, Mahanama SP (2022) Warming, increase in precipitation, and irrigation enhance greening in High Mountain Asia. Commun Earth Environ 3:1–8. https://doi.org/10.1038/s43247-022-00374-0

Odden M, Athreya V, Rattan S, Linnell JDC (2014) Adaptable neighbours: Movement patterns of GPS-collared leopards in human dominated landscapes in India. PLoS One 9:e112044. https://doi.org/10.1371/journal.pone.0112044

Rodrigues RG, Srivathsa A, Vasudev D (2022) Dog in the matrix: Envisioning countrywide connectivity conservation for an endangered carnivore. J Appl Ecol 59:223–237. https://doi.org/10.1111/1365-2664.14048

Sen PK (1968) Estimates of the regression coefficient based on Kendall's tau. J Am Stat Assoc 63:1379–1389. https://doi.org/10.1080/01621459.1968.10480934

Outline

Below is an outline of the entire section, including every section header.