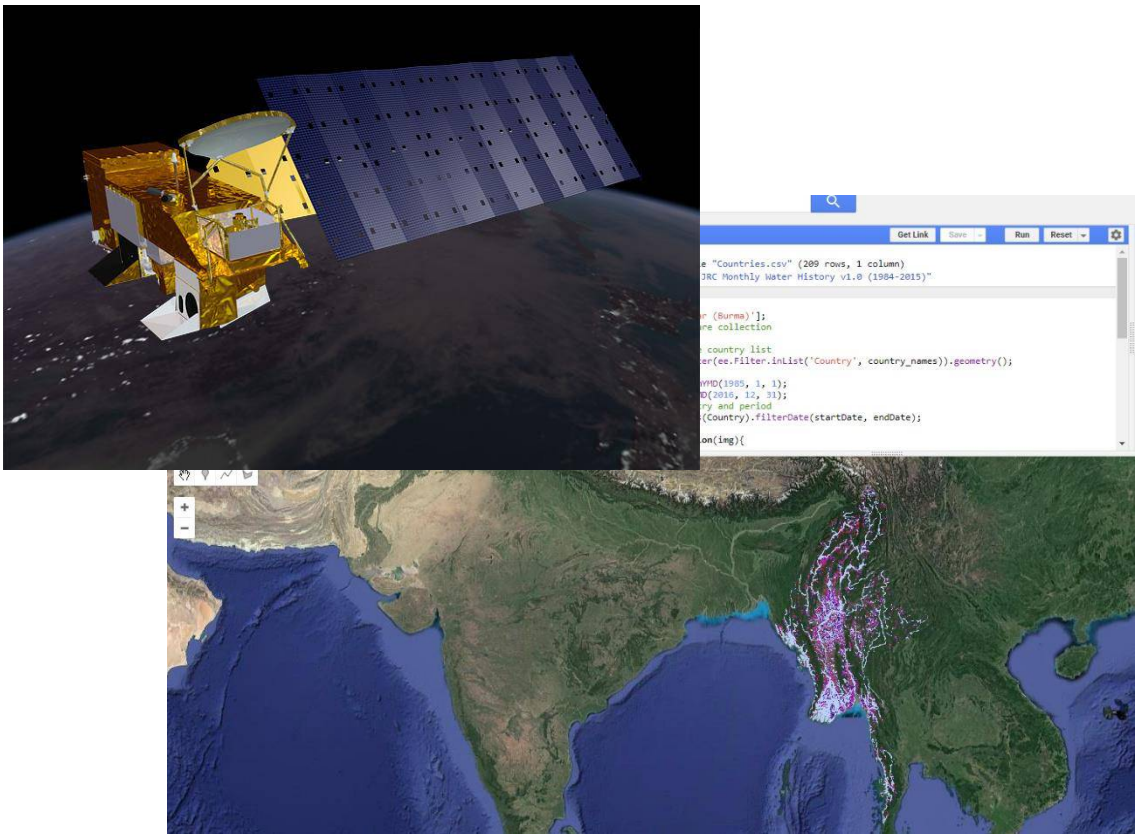
Satellite-based remote sensing to support water professionals in Myanmar: Advanced hands-on training 2

Tutorial

Yangon Technical University, Myanmar
4 – 15 November 2019



Trainers
Carolien Wegman (c.wegman@hkv.nl)
Corjan Nolet (c.nolet@futurewater.nl)

FutureWater

HKV CONSULTANTS

Acknowledgements

This training is organized in the framework of the Orange Knowledge Tailor-Made Training (TMT) Programme, funded by NUFFIC the Dutch organization for internationalization in education. Next to the practical experience of the trainers, the content of this tutorial is based on various Google Earth Engine scripts and examples found around the web. In particular, the excellent https://mygeoblog.com of Dr. Ate Poortinga been a valuable source of information and inspiration and is strongly recommended to anyone wishing to pursue working with GEE and exploring further applications. Other useful online resources, amongst other, include:

https://developers.google.com

- User guide: https://developers.google.com/earth-engine/
- Tutorials: https://developers.google.com/earth-engine/tutorials
- Educational and training resources: https://developers.google.com/earth-engine/edu
- Developers discussion group: https://groups.google.com/forum/#!forum/google-earth-engine-developers.
- Google Git: https://earthengine.googlesource.com/?format=HTML

https://gis.stackexchange.com

- Questions tagged [google-earth-engine]: https://gis.stackexchange.com/questions/tagged/google-earth-engine

https://stackoverflow.com

- Questions tagged [google-earth-engine]: https://stackoverflow.com/questions/tagged/google-earth-engine

https://github.com

- Earth engine github repositories: https://github.com/search?q=google+earth+engine

Table of contents

1 Training content and objectives

The objective of this training is to build capacity of the participants on the use of satellite-derived data for water management in Myanmar. The training builds upon the skills obtained during the first course organized in March 2018 and the second course in December 2018 at Yangon Technological University. This training focuses on three themes: river morphology, Integrated Water Resources Management and landcover classification. These topics were identified as highly relevant in the Myanmar context by the participants during the first training. Participants will gain further hands-on experience with state-of-the-art tools, in particular Google Earth Engine, and are encouraged to assess how these tools can be beneficially utilized in their own organization.

The training is aimed at participants from a variety of backgrounds, including policy makers, researchers and the private sector. It is foreseen that an improved awareness of the opportunities offered by remote sensing data and an enhanced skillset of processing and analyzing these data will support improved water resources management in Myanmar, particularly in ungauged regions.

This tutorial describes the training exercises step-by-step. A recap of the March and December 2018 training program is covered in Chapter 2. Chapter 3 focuses on Integrated Water Resources Management applications of Google Earth Engine and Chapter 4 contains exercises on working with river morphology. Finally, Chapter 5 focuses on Landcover classification

Overview of development of skills in this course:

- Learn JavaScript and Earth Engine syntax.
- Learn how to access data from the public data catalog, add them to your map window and set the visualization parameters, process and analyze these, and export them to your hard drive.
- Learn how to write your own functions and map these over image collections.
- Gain in-depth knowledge on river morphology, Integrated Water Resources Management and landcover classification.

2 Google Earth Engine: A recap

2.1 Google Earth Engine

Google Earth Engine is a cloud-computing platform for processing satellite imagery and other geospatial and observation data. It provides access to a large database of satellite imagery and the computational power needed to analyze those images. Google Earth Engine allows observation of dynamic changes in agriculture, water resources, and climate (among others) using geospatial data with varying levels of spatial detail and overpass frequencies. The Google Earth Engine provides a data catalog along with computers for analysis; this allows various user groups, such as researchers and water professionals, to collaborate using data, algorithms, and visualizations.

The use of Google Earth Engine has increased rapidly over the past few years, in a wide range of sectors (research, government and private sector) as well as in many different fields of application (water management, agriculture, nature conservation, etc.). The popularity of the tool is related to its great benefits over conventional technologies, including:

- Geoprocessing is performed in the cloud, so it is no longer needed to download gigabytes of data to your PC. This saves space on your hard drive, infrastructure costs, and overcomes limitations related to internet speed;
- Special remote sensing image processing software is no longer required, which saves software licensing costs;
- Processing speed is high and saves valuable working time;
- It provides access to huge datasets not only at the national scale but also at the global scale, allowing for transboundary analyses (essential for water management!);
- It provides access to decades of historical data which allows for time-series analyses;
- It provides access to pre-processed multi-sensor satellite datasets, allowing easy integration of various data sources without time-consuming correction / pre-processing
- It allows you to share your work and knowledge with other experts, as well as the general public.

The use of Google Earth Engine requires a Google account. Signing up to Google Earth Engine can be done here: https://earthengine.google.com/signup/. If you are already signed up, you do not have to do anything.

2.2 Getting started with the GEE interface

The Code Editor of GEE is the place where you can develop your own scripts for analyzing the wealth of available data. The Code Editor is found here: https://code.earthengine.google.com/.

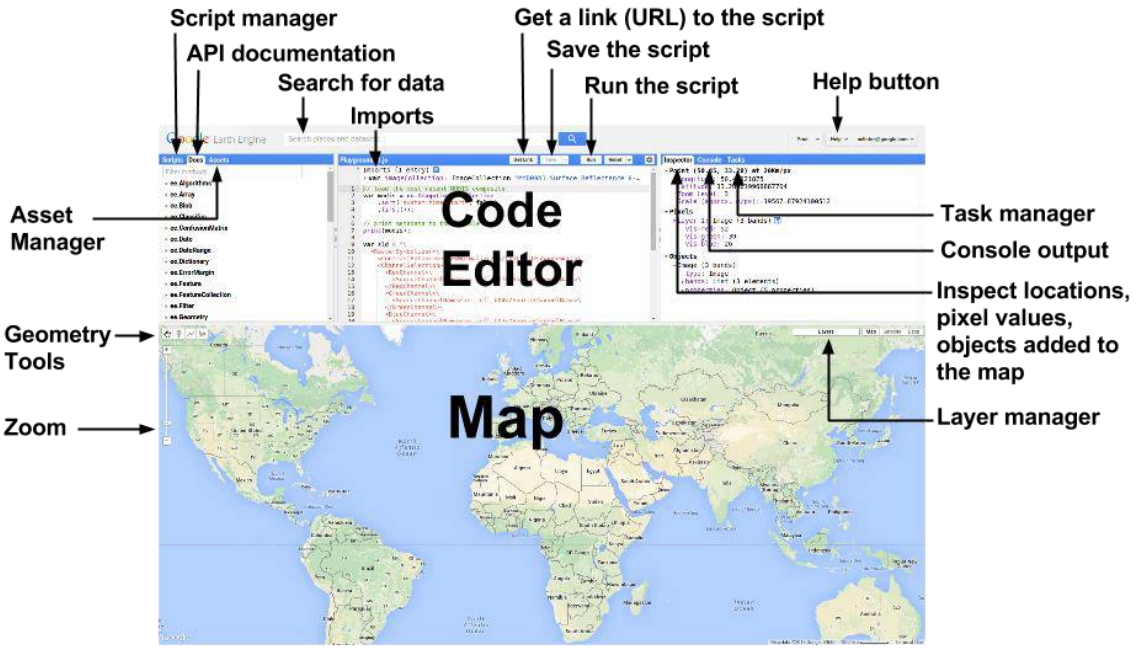
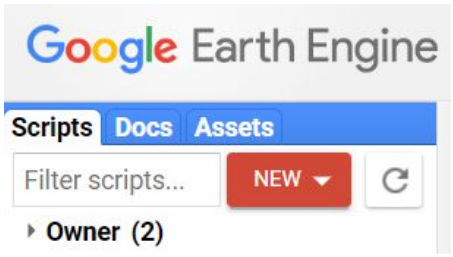


Figure 1. Interface of the GEE Code Editor

The layout of the Code Editor and locations of the basic functions are shown in Figure 1. This tutorial describes several basic actions in a step-wise, concise manner. In case you would like any further information, e.g. on the functions not covered in the tutorial, please refer to https://developers.google.com/earth-engine/playground or ask one of the trainers.

Step 1. Click on "Scripts" and create a new script by clicking New → File and name it Landsat8_Myanmar.



In a similar way, you can create new folders and repositories. In order to make a copy of a script, create a new repository and click and drag the script to the new repository.

The Scripts window has several subsections, including *Examples*. Here, an elaborate list of example scripts is provided, which are all ready to run. Whenever you want to develop a new GEE script, it may be useful to check some examples to see what is already available.

Under *Docs*, you can find all methods (functions) available for data processing. You will likely often use e.g. some of the functions under *ee.Image*, which contains all operations that can be performed on an image. Another example is the *ee.Reducer* category, which contains several useful functions to extract statistics from an image. Clicking on a method provides you with a description of its functionality and its appropriate syntax.

Finally, under Assets, you can manage and upload geospatial datasets and tables. These assets are initially private but can be shared with others.

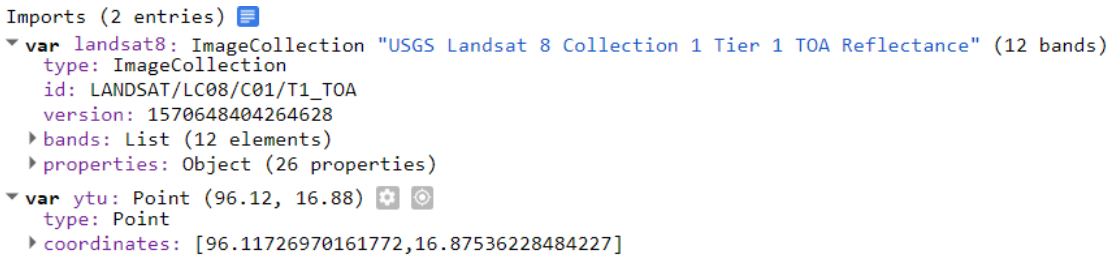
2.3 Exploring and visualizing Landsat 8 data

The objective of this exercise is to explore data collected by the Landsat 8 satellite, an American Earth observation satellite launched on February 11, 2013. The Landsat 8 satellite payload consists of two science instruments—the Operational Land Imager (OLI) and the Thermal Infrared Sensor (TIRS). These two sensors provide seasonal coverage of the global landmass at a spatial resolution of 30 meters (visible, NIR, SWIR); 100 meters (thermal); and 15 meters (panchromatic). Landsat 8 captures more than 700 scenes a day, circles the Earth every 99 minutes and has a 16-day revisit time (https://landsat.gsfc.nasa.gov/landsat-data-continuity-mission/).

The search bar on the top of the screen in the Code Editor can be used to explore the available data on GEE. Clicking on a dataset will show its metadata, i.e. information on the data type, unit, distributing agency and other relevant properties. When you have found the dataset you are looking for, you can click Import to use it in a script.

Step 2. Search for landsat 8 in the search bar and import the USGS Landsat 8 Collection 1 Tier 1 TOA Reflectance imageCollection into your Landsat8_Myanmar script. The standard name of the imported variable is ImageCollection, rename this to **landsat8**.

Step 3. The next step is to define a region of interest. Select the point drawing tool (teardrop icon) from the geometry tools, go to the campus of Yangon Technical University and draw a single point as the region of interest. Then 'Exit' from the drawing tools. Note that a new variable is created in the imports, containing the single point, imported as a Geometry. Rename this import from geometry to **ytu**. Your imports section will look similar to the image below:



Step 3. Now we are ready to select a single Landsat 8 image from the imageCollection. To do this we first filter all the images in the collection on a date range and geographical footprint. Then, to ensure we obtain a cloud-free image, we retrieve the image with the lowest percentage of cloud cover. Type the following code below in the editor:

```
// Filter the Landsat8 image collection using filterBounds() and filterDate() method.
// Sort the collection by cloud cover metadata

var ls8Image = ee.Image(landsat8
  .filterBounds(ytu)
  .filterDate('2018-01-01', '2018-12-31')
  .sort('CLOUD_COVER')
  .first());
```

Step 4. To inspect the obtained image, we have a look at its metadata:

```
// Get information about the bands as a list.
var bandNames = ls8Image.bandNames();
print('Band names: ', bandNames); // ee.List

// Get the timestamp and convert it to a date.
var ls8Date = ee.Date(ls8Image.get('system:time_start'));
print('Timestamp: ', ls8Date); // ee.Date

// Get the percentage cloud cover
var cloudiness = ls8Image.get('CLOUD_COVER');
print('Cloud cover: ', cloudiness); // ee.Number
```

Question

1. At which date was the obtained image acquired? And what is the cloud cover?

.....

Step 4. Before we can display the image, we must first define the visualization parameters as a property of the image. We will define two sets, one to display the image in true color and one for false-color. (see also: https://developers.google.com/earth-engine/image_visualization).

```
// Define visualization parameters
var trueColor_ls8 = {
  bands: ['B4', 'B3', 'B2'],
  min: 0,
  max: 0.5,
};

var falseColor_ls8 = {
  bands: ['B5', 'B4', 'B3'],
  min: 0,
  max: 0.5
};
```

Step 5. We can now add the obtained Landsat 8 image to the Map using the Map.addLayer() function. After running the two lines of code below, you will get the same image as displayed in Figure 2. You can toggle between the two image layers using the Layers manager in the right-top corner of the Map panel. Here you can also open and manipulate the visualization parameters of the displayed layers. You can adjust the sliders and see how it affects the image display. If you like a particular setting you can import those parameters as a new variable, which will appear in the Imports section.

```
// Display the image via Map.addLayer(eeObject, visParams, name)
Map.addLayer(ls8Image, trueColor_ls8, 'Ls8 2018 true color');
Map.addLayer(ls8Image, falseColor_ls8, 'Ls8 2018 false color');
```

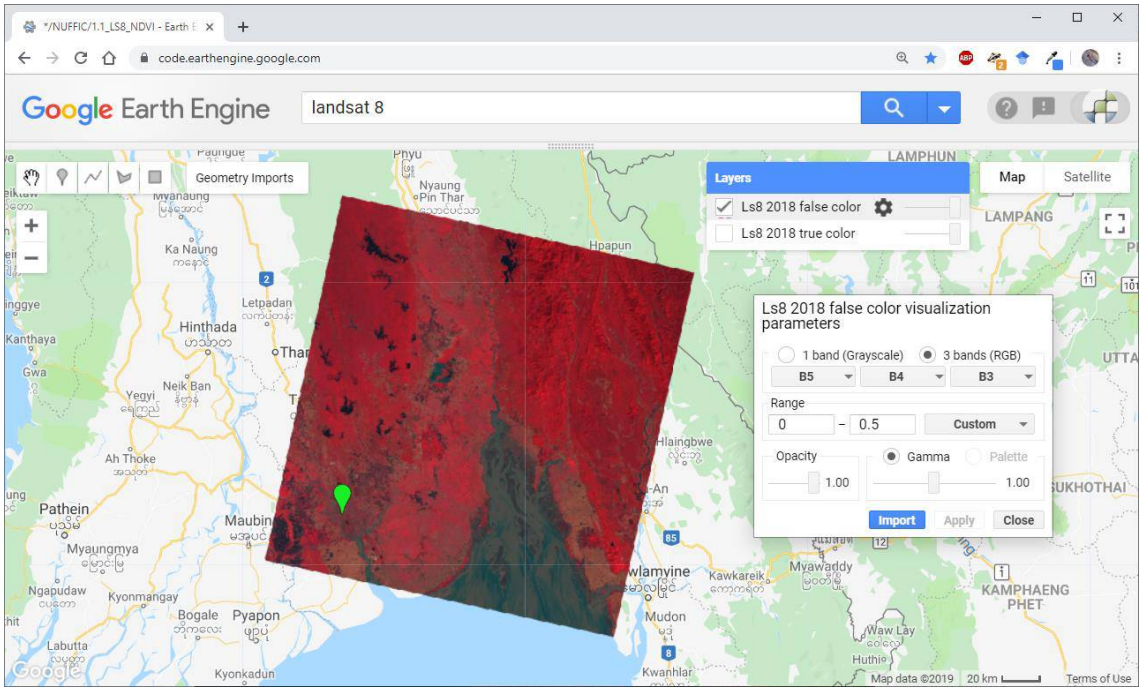


Figure 2. Landsat 8 false-color image

Don't forget to save your script! The full code corresponding with this exercise can be found here:https://code.earthengine.google.com/f61a9effe53009e998d6a1d469abd859

2.4 Create a chart of monthly NDVI values

The objective of this exercise is to create a time series of monthly NDVI values of a region of interest using Landsat 8 satellite imagery and to display it in a chart. The Normalized Difference Vegetation Index (NDVI) is the ratio (0 -1) between a plant's reflectance of Red and Near-Infrared light:

$$NDVI = \frac{(NIR - Red)}{(NIR + Red)}$$

NDVI is a very useful index for detecting vegetation and gives an indication about the health of a plant. As Figure 3 shows, a healthy plant will absorb most red light for photosynthesis but reflect most infrared light, giving high NDVI values. Unhealthy vegetation will absorb less red light but more infrared light, which results in a lower NDVI value.

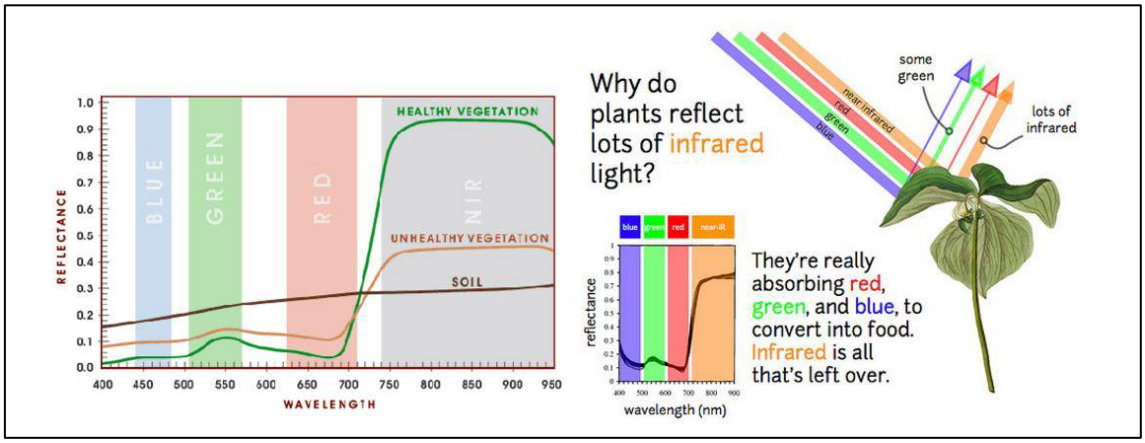


Figure 3. Normalized Difference Vegetation Index (NDVI)

Step 1. To start, create a new script and call it NDVI_Myanmar. We will re-use the imports created in the previous exercise. Go back to the script Landsat8_Myanmar, hoover over the blue icon in the Imports section and click it to Show generated code. Copy and paste the code into your new script and convert the code back to Imports records.

Below is the JavaScript code representing the current imports. To transfer them to another script, paste this code into the editor and click "Convert" in the suggestion tooltip.

```
var landsat8 = ee.ImageCollection("LANDSAT/LC08/C01/T1_TOA"),
    ytu = /* color: #19f025 */ee.Geometry.Point([96.11726970161772, 16.87536228484227]
```

Step 2. We will again filter the Landsat 8 ImageCollection, but now over a multi-year year date range (2013 – 2018) and geographical footprint. And we are now interested in all the images within these criteria and will therefore not sort and select an image based on percentage cloud cover. Type the following code in de Editor:

```
// Center the map on region of interest
Map.centerObject(ytu, 10);

/* Filter the Landsat8 image collection
using filterBounds() and filterDate() method */
var ls8 = landsat8
  .filterBounds(ytu)
  .filterDate('2013-01-01', '2018-12-31');

// print to Console
print(ls8);
```

Question

2. How many images do we obtain after applying the two filter functions?

……

Step 3. We are now ready to calculate the normalized difference vegetation index (NDVI) using the red (Band ??) and near-infrared (Band ??) band of the Landsat imagery. To do this we will first declare a function to compute the normalized difference between these two bands and to

add the result as a new band to each input image. Fill in the blue boxes with the names of the bands we need to use.

```
// Function to add a NDVI band
function addNDVI(image) {
  var NDVI = image.normalizedDifference(['', '']).rename("NDVI");
  return image.addBands(NDVI);
}
```

Step 4. We will use the map() method to apply the function to every image in our collection. Print the newly created ImageCollection to verify that each image now contains an extra band named NDVI.

```
// Map the function over the collection
var ls8NDVI = ls8.map(addNDVI);
print(ls8NDVI);
```

Step 5. To visualize the results of our NDVI calculation we will apply a reducer to aggregate the ls8NDVI ImageCollection to an individual composite image (see also: https://developers.google.com/earth-engine/reducers_image_collection.html). As shown in Figure 4, we can reduce an ImageCollection over space, time and/or bands. In our example we will apply a reducer to aggregate over time, to calculate the median and max NDVI values for the year 2018. For basic statistics there are shortcut methods, so we do not have to explicitly call the Reducer method.

```
/* Statistical aggregation (min, max, mean, median, mode, count)
is needed to convert the ImageCollection into a single image */

var maxNDVI = ls8NDVI.max();
print('maxNDVI',maxNDVI);

var medianNDVI = ls8NDVI.median();
print('medianNDVI',medianNDVI);
```

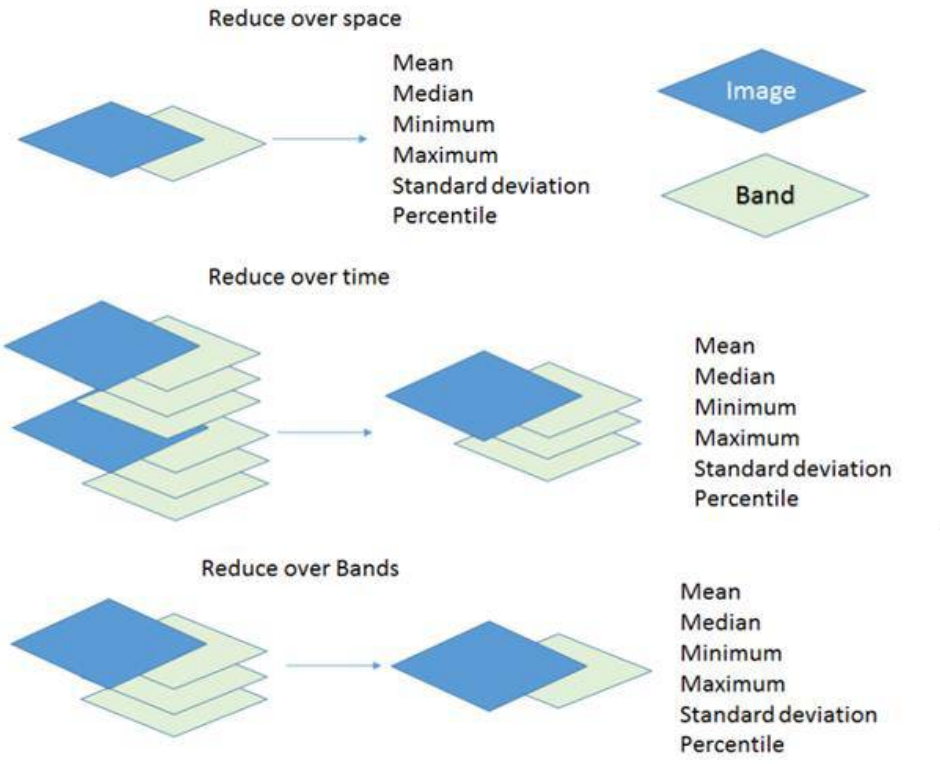


Figure 4. Reduce for statistical aggregation over space, time and/or bands

Step 6. We will now define the visualization parameters and add the NDVI band of both images to the Map. Because NDVI is a single band, we must define a color palette to set a color scheme:

```
// Define visualization parameters
var vegVis = {
  bands : ["NDVI"], // added NDVI band
  palette : ['darkblue', 'blue', 'red', 'orange', 'yellow', 'green', 'darkgreen'],
  min : -0.4,
  max : 0.9
};

// Add images to Map
Map.addLayer(maxNDVI, vegVis, 'maxNDVI');
Map.addLayer(medianNDVI, vegVis, 'medianNDVI');
```

The colors are defined using the web standard CSS color value scheme (https://en.wikipedia.org/wiki/Web_colors). Colors can be specified by name (as used in step 6) or strings indicating the combination of red, green and blue. The lowest value in any of the three positions is 00 (representing the decimal number 0), while the highest is FF (representing the decimal number 255). The string '000000' represents the color black, 'FFFFFF' is white, 'FF0000' is red, '00FF00' is green, and '0000FF' is blue.

Step 7. Feel free to adjust the min and max values, the colors, and the number of colors, and observe what happens.

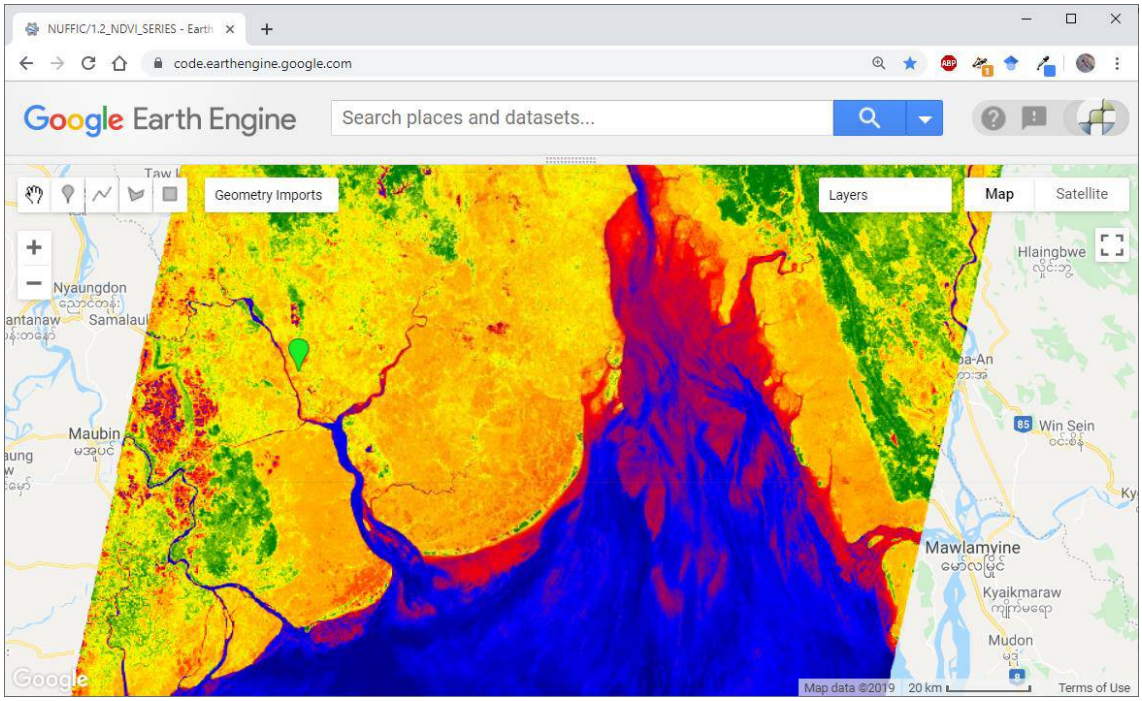


Figure 5. Reducing Landsat 8 ImageCollection to a median NDVI image for the year 2018

Step 8. We are now going to select an area within the Landsat 8 ImageCollection footprint and draw a polygon (as done in Figure 6). For this area we will calculate what the average NDVI value per month was for the chosen period 2013 – 2018. To do this we will first declare a function to calculate the average NDVI within a given geometry. Type the code below in the editor:

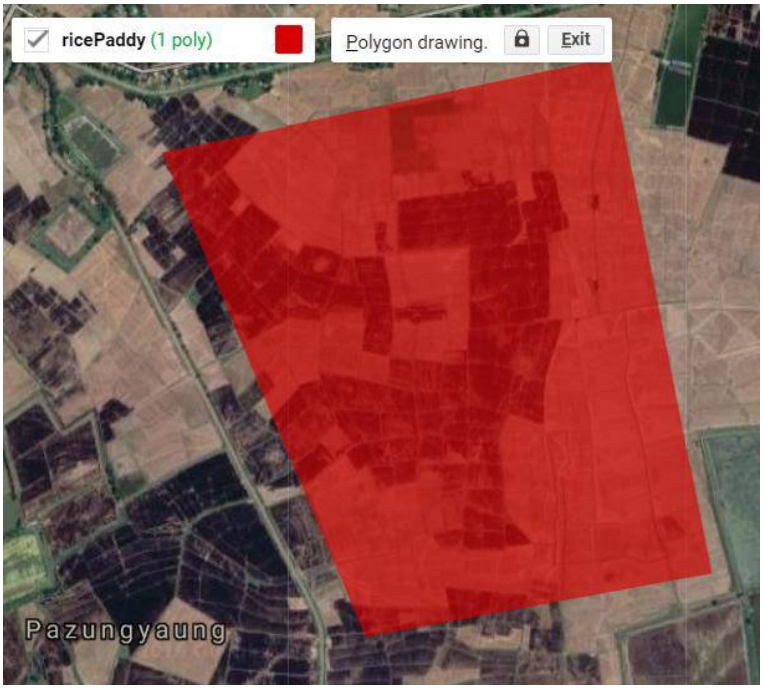


Figure 6. Polygon drawn over a rice field using Geometry Tools

```
//-------------------------------------------------
// Use Geometry tools to draw an area of interest.
// In this example a polygon named ricePaddy is drawn
// -------------------------------------------------

// Function to calculate average NDVI within the given geometry
var meanNDVI = function(image) {
  var stats = image.reduceRegion({
    reducer: ee.Reducer.mean(),
    geometry: ricePaddy,
    scale: 30
  });
  return stats.get('NDVI');
};
```

Step 9. We will call this function within another function where first all the images are grouped together according to which month of the year the image was acquired and then the mean monthly NDVI values are calculated:

```
// Function to group all images by a given month on acquisition date and
// to compute a mean monthly NDVI composite by calling the meanNDVI function

var monthNDVI = function(month) {
  var monthImages = ls8NDVI.filter(
    // Use calendarRange function to filter by month
    ee.Filter.calendarRange({start: month, field: 'month'}));
  // For simplicity, pick one image and compute mean NDVI
  var meanFirst = meanNDVI(ee.Image(monthImages.first()));
  // You can also take all images and take the mean value of the stack
  var meanAll = meanNDVI(ee.Image(monthImages.mean()));
  // Return NDVI value
  return ee.Number(meanFirst); // or choose ee.Number(meanFirst)
};
```

Step 10. We will map the function over a declared list to obtain the NDVI values. (i.e. the list is ‘filled’ with the NDVI values). Print the variable NDVIByMonth to see the result:

```
// Declare list of month numbers
var months = ee.List.sequence(1, 12);

// Map the function over the list
var NDVIByMonth = months.map(monthNDVI);
print(NDVIByMonth);
```

Step 11. The final step is to display the NDVI values in a graph. To set up the graph we will first declare a new list that holds the names of the months and make a variable that can hold graphical parameters related to axis labels, title, legend and color etc:

```
// Set chart options
var chartOptions = {
  title: 'Monthly NDVI',
  hAxis: {title: 'Months'},
  vAxis: {title: 'NDVI'},
  legend: { position: "none" },
};
```

Step 12. We can now create the chart. With the code snippet below a barplot ('ColumnChart') is created. Feel free to explore the setChartType options to create a different type of plot.

```
// Create chart
var chart = ui.Chart.array.values({
  array: NDVIByMonth,
  axis: 0,
  xLabels: monthName})
  .setChartType('ColumnChart')
  .setOptions(chartOptions);
print(chart);
```

In this example we get the following NDVI pattern for each month of the year. Of course, depending on your chosen area, your graph might look very different from this one.

Question

3. Considering the NDVI values below were obtained from a rice field, can you explain the pattern? Can you tell in which month the rice was most likely planted, and which month it was harvested?

……

……

……

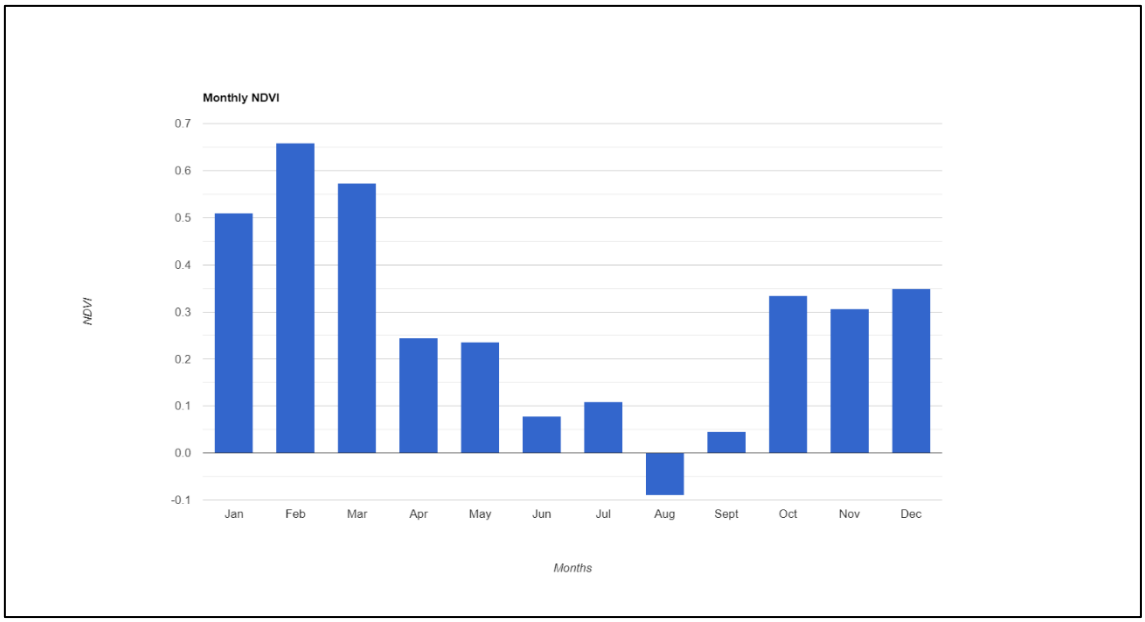


Figure 7. Average monthly NDVI values (2013 – 2018) for a rice paddy field

Don't forget to save your script! The full code corresponding with this exercise can be found here:https://code.earthengine.google.com/c41ab6506abda8430324532f096350ba

3 GEE applications for Integrated Water Resources Management

Although rainfall stations continue to be an irreplaceable source of data, satellite-derived products provide valuable additional information on spatial patterns and remote sites where no stations are present. Thanks to GEE these data products are now publicly available and can improve water management policies, practice and research worldwide.

3.1 Detecting rainfall anomalies using CHIRPS (part I)

The Climate Hazards Group InfraRed Precipitation with Station data (CHIRPS) is a 30+ year quasi-global rainfall dataset. CHIRPS was created in collaboration with scientists at the U.S. Geological Survey (USGS) Earth Resources Observation and Science (EROS) Center in order to deliver reliable, up to date, and more complete datasets for several early warning objectives. Spanning 50°S-50°N (and all longitudes), starting in 1981 till present, CHIRPS incorporates 0.05° resolution (about 5 x 5 km) satellite imagery with in-situ station data to create gridded rainfall time series for trend analysis and seasonal monitoring.

What can CHIRPS be used for?

- Assessing precipitation at national, regional and local scale
- Evaluating the seasonality of precipitation
- Identifying regions where precipitation have increased or decreased

What can CHIRPS not be used for?

- Predicting precipitation in the coming season/year
- Investigating disasters caused by factors other than the distribution of precipitation.

Step 1. We are going to use CHIRPS to analyze and display rainfall variations in various time frames and spatial areas. The CHIRPS precipitation data can be modified to output a time-series of precipitation for a user-defined region. This user-defined region can be defined through use of the geometry drawing tools or selected from a FeatureCollection (i.e. shapefile). We are going to work with a shapefile that contains the most important basins of Myanmar. Open a new code editor and enter the following lines into your editor and convert the variable into your Imports section:

```
1.  // Import Myanmar basins FeatureCollection
2.  var basins = ee.FeatureCollection("users/futurewaternl/myanmar_basins");
```

Step 2. To display the basins FeatureCollection, enter the following lines of code:

```
// Center the map and display the image.
Map.setCenter(96.1, 19.8, 6); // Coordinates Naypyidaw

// Define the visualization parameters.
var visPar = {
  fillColor: 'b5ffb4',
  color: '00909F',
  width: 3.0,
};
Map.addLayer(basins,visPar,"basins");
```

Step 3. We are going to make time-series of monthly precipitation sums using CHIRPS over a user-defined time range. The following code snippet will compute the monthly sum of precipitation for the entire CHIRPS ImageCollection over a period (in this case) from 1998 until present:

```
/* Function to calculate the precipitation sum per month
/ over a sequence of n number of months */

var monthSum = ee.List.sequence(0, 22*12).map(function(n) {
  var start = ee.Date('1998-01').advance(n, 'month'); // Starting date
  var end = start.advance(1, 'month'); // Step by each iteration
  return ee.ImageCollection("UCSB-CHG/CHIRPS/DAILY")
        .filterDate(start, end)
        .sum()
        .set('system:time_start', start.millis());
});

print( "CHIRPS monthly sum", monthSum);
```

Question.

1. When you print the variable monthSum to the console, the code above will return the monthly CHIRPS sum as a list of images. How many images have you obtained? Till which month were you thus able to get a CHIRPS image?

.....

.....

Step 4. To make a chart it is often easier to convert (i.e. cast) the list of single images into a single ImageCollection. This can be done with this line of code:

```
// Cast CHIRPS list to ImageCollection
var monthCollection = ee.ImageCollection(monthSum);
print("CHIRPS monthly sum", monthCollection);
```

Step 5. Before we can make the chart, we must first select a single basin for which we want to display precipitation sums. In this example we will use the basin Sittaung but feel free to use another basin. To select the basin, we need to filter the basins FeatureCollection. Enter the following code into the editor and replace the blue box with the correct text string:

```
// Select a basin
var Sittaung = basins.filter(ee.Filter.eq("[redacted]","Sittaung"));
```

Step 6. Now we are ready to make the chart. As you did in the previous chapter, to set up the graph we first have to make a variable that can hold graphical parameters:

```
// Set chart options
var chartOptions = {
  title: 'Monthly precipitation sum',
  hAxis: {title: 'Time'},
  vAxis: {title: 'Precipitation (mm)'},
};
```

Step 7. The code snippet below will make the time series chart. The arguments to the ui.Chart.image.series() method include an ImageCollection (*monthCollection*), a region (*Sittaung*), a Reducer (*mean*) and a scale (*meter*). CHIRPS data has a 0.05° resolution which (roughly) translates to a scale of 5000 m. For more details on charts and chart options, see also: https://developers.google.com/earth-engine/charts

```
// Create chart
var timeSeries = ui.Chart.image.series({
    imageCollection: monthCollection,
    region: Sittaung,
    reducer: ee.Reducer.mean(),
    scale: 5000,
    xProperty: 'system:time_start',
  }).setChartType('ColumnChart')
    .setOptions(chartOptions);

print('timeseries of selected area:', timeSeries);
```

Step 8. After running the full script, you will get a graph that will look very similar to the one below. With the  icon you can open the chart in a new tab for closer inspection and options are available to download the image or the underlying data.

Question

2. Looking at the precipitation sums, which year(s) probably experienced the most rain? And which the least? Can you think of a way to quickly verify this visually? And after doing that, were it the years that you initially expected?

   Hint 1: Adapt the function monthSum into a function called yearSum
   Hint 2: https://code.earthengine.google.com/14a012b2aa985d3a43f9c26b846d4173)

……

……

……

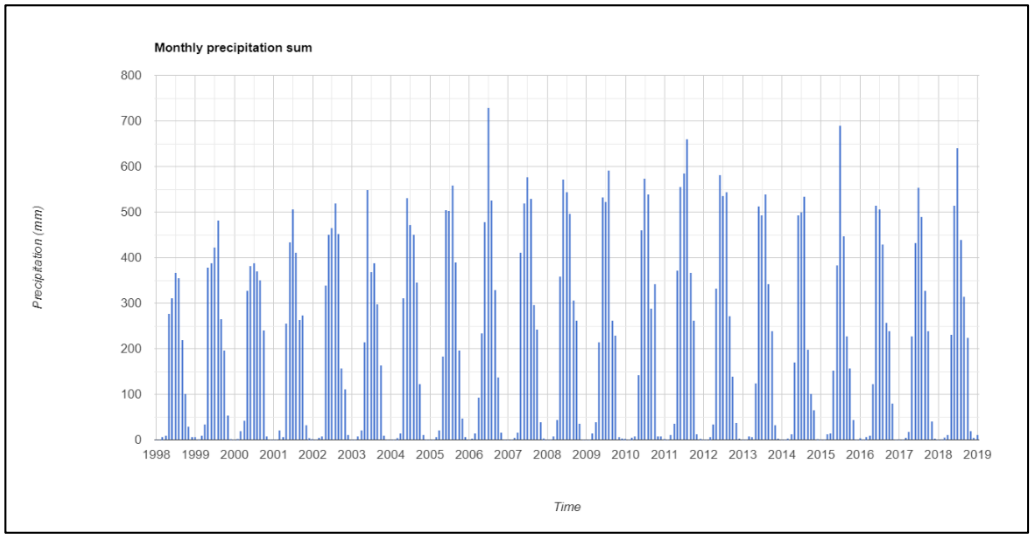


Figure 8. Monthly precipitation sums for Sittaung basin

Step 9. Besides displaying the data in a chart, it can also be very informative to display the data spatially on the Map. Using the output of the function yearSum we will first select the wettest year in our timeseries using the code below. Replace the ?? with the correct number.

```
/* Select the wettest year (2011) from the list
and cast to an ee.Image() */
var chirps2011 = ee.Image(yearSum.get(??));
print("wettest year", chirps2011);
```

And then display the precipitation sum for the Sittaung basin using pre-defined visualization parameters and the clip() function:

```
// Define the visualization parameters.
var prVis = {
  min: 1.0,
  max: 5000,
  palette: ['001137','0aab1e','e7eb05','ff4a2d','e90000'],
};

// Add Map to the canvas
Map.addLayer(chirps2011.clip(Sittaung), prVis, 'yearSum 2011');
```

Step 10. For being able to easily interpret the obtained map, it is useful to also include a legend of what the colors mean. A standard code for adding a gradient legend to GEE maps is provided here: https://code.earthengine.google.com/a7526b6646951d32e889d83aab0c3c29. When you copy this code to the bottom of your script, your map should now look like the picture below.

Question

3. When you inspect the distribution of precipitation sums, can you describe what the causes are of the spatial variation in rainfall?

……

……

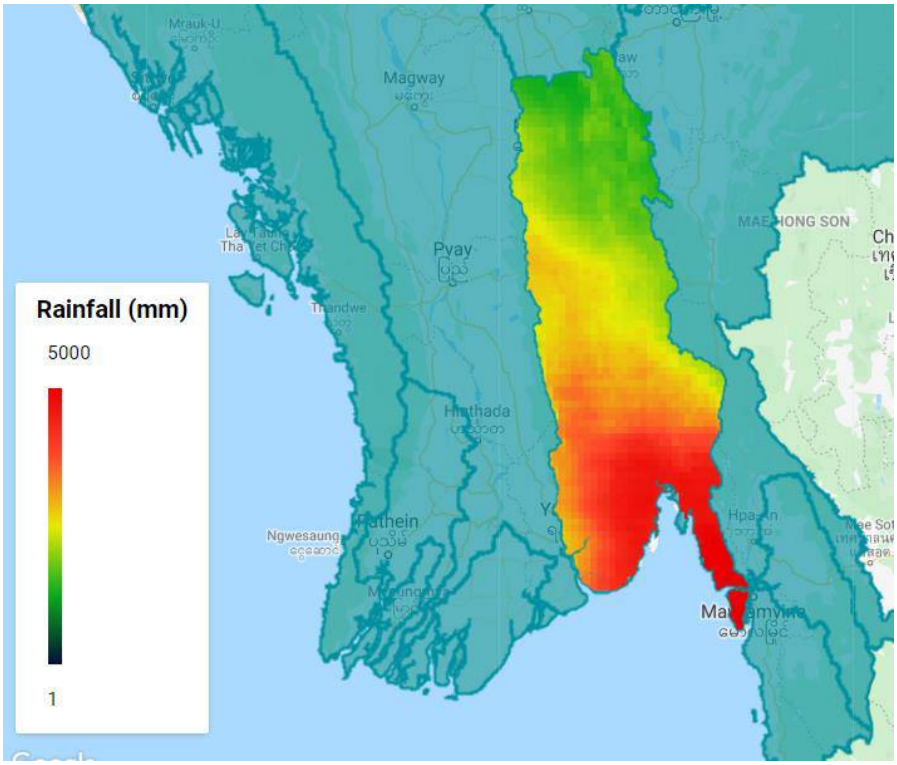


Figure 9. Precipitation sum for Sittaung basin in 2011

Don't forget to save your script! The full code corresponding with this exercise can be found here:https://code.earthengine.google.com/e5da54314da46c685018ff6a2d630089

3.2 Detecting rainfall anomalies using CHIRPS (part II)

In the previous exercise we calculated and mapped average monthly and annual precipitation sums to investigate how the distribution of precipitation has varied over the years. In this exercise we are going to take a closer look at so-called precipitation anomalies. In the world of climate, the term *anomaly* means the difference between the value of a quantity and its climatological mean value. A monthly anomaly is the difference between the original monthly value of a quantity in a given month and the monthly climatological value for that month of the year. This can be written in a formula as:

$$r'_{ij} = r_{ij} - \frac{1}{N_i}\sum_{j=1}^{N} r_{ij}$$

Here, for months i and years j, r'_{ij} is the monthly anomaly, r_{ij} is the original monthly value, and the remainder of the equation is the calculation of the monthly climatology (which is subtracted from the original monthly values). Monthly anomalies of precipitation indicate the difference (positive or negative) between a monthly precipitation value and its "normal" value for that month of the year, in terms of the original units of the quantity (e.g., mm/month, or monthly mean mm/day).

Calculating anomalies is one way to remove the annual cycle from a time series. This can be a useful thing to do in some types of analyses, such as the calculation of correlations between two-time series. If the annual cycle is left in the time series, a high correlation between two variables may sometimes be the result of this annual cycle, which may mask other variations of greater interest to the analysis.

Step 1. To get started we will re-use some code from the previous exercise. Open a new code editor tab and type the following code to import the basins FeatureCollection and display the Sittaung basin:

```
// Import basins FeatureCollection
var basins = ee.FeatureCollection("users/futurewaternl/myanmar_basins");

// Center the map and display the image.
Map.setCenter(96.1, 19.8, 6); // Coordinates Naypyidaw

// Select a basin
var Sittaung = basins.filter(ee.Filter.eq("Name","Sittaung"));

// Define the visualization parameters.
var visPar = {
  fillColor: 'b5ffb4',
  color: '00909F',
  width: 3.0,
};

// Add Map to canvas
Map.addLayer(Sittaung,visPar,"Sittaung basin");
```

Step 2. Before we can continue, we also have to import the Chirps ImageCollection. To speed up computing time we will now work with the CHIRPS pentad ImageCollection instead of the daily that we used in the previous exercise. This time we will also declare it as variable. Search for CHIRPS pentad in the search bar, import the collection and rename it as *pentadChirps*:

```
// Import CHIRPS ImageCollection
var pentadChirps = ee.ImageCollection("UCSB-CHG/CHIRPS/PENTAD");
```

Question

1. How does Chirps define a pentad? As a result, in which unit are the precipitation values in this ImageCollection?

……

……

Step 3. We will now declare a function that, when called, will calculate the 'normal' precipitation value for every month of the year (for a user-defined region). Using a multi-year CHIRPS timeseries as input, this function will compute the average precipitation for each month (e.g. January) using all months of January in the time-series. At its core, this function gives an almost identical output as the function that we used in exercise 2.3 to compute mean monthly NDVI values.

```
/* Function to compute the 'normal' precipitation
value for every month of the year */

var meanMonth = ee.ImageCollection(ee.List.sequence(1, 12)
  .map(function(m) {
    return pentadChirps.filter(ee.Filter.calendarRange(m, m, 'month'))
        .mean()
        .set('month', m);
}));
```

Step 4. Now that we have the function meanMonth to compute the multi-year monthly mean, the next step is to make a list of dates over which we will map the function. We will make a list of dates (where each date is the first day of a month) starting at the 1st of January 1998 until the most recent CHIRPS image available in GEE. Enter the following two lines into the editor to retrieve the last image:

```
// Find date last CHIRPS image
var lastChirps = pentadChirps.limit(1, 'system:time_start', false).first();
var endDate = ee.Date(lastChirps.get('system:time_start'));
```

Question

2. From what exact date is the most recent CHIRPS image available in GEE?

……

Step 5. Now enter the next lines of code to complete making the list of dates. Print the list to the console and inspect the elements in the list.

```
// Define start date
var startDate = ee.Date('1998-01-01');

// Create list with numbers
var numberMonth = endDate.difference(startDate,'month').round().subtract(1);
var monthSeq = ee.List.sequence(0,numberMonth,1);

// Function to convert list of numbers to list of dates
var dateMake = function(n) {
  return startDate.advance(n,'month');
};

// Map the function over the numbered list
var dateSeq = monthSeq.map(dateMake);
print(dateSeq);
```

Step 5. The code below is the main algorithm to compute the rainfall anomalies. Using the previously declared list of dates we will group the CHIRPS images by month and then apply a reducer to compute the average rainfall within each month. The multi-year monthly mean (i.e. the 'normal' precipitation for that month) is then subtracted from the mean of a single month. The output of the function is an ImageCollection where each image contains the monthly rainfall anomaly (mm / month) for every month in the user-defined period. (We can verify this when we print the variable anomolyMonth to the console):

```
/* Code to compute monthly precipitation anomalies:
1) Group pentad CHIRPS by month and reduce within groups by mean();
2) Subtract the multi-year monthly mean from the spatial mean of single month
3) Result is an ImageCollection with one image for each month */

var anomalyMonth = ee.ImageCollection.fromImages(
      dateSeq.map(function(date) {
        var start = date;
        var end = ee.Date(date).advance(1, 'month');
        var mean = pentadChirps.filterDate(start, end)
                    .mean()
                    .set('date', date);
        var month = ee.Date(date).getRelative('month', 'year').add(1);

        return mean.subtract(meanMonth.filter(ee.Filter.eq('month', month))
                    .first())
                    .set('date', date);
}));
print('Monthly anomalies ImageCollection',anomalyMonth);
```

Step 6. Now that we have obtained our rainfall anomalies, we are going to display the results in a chart. For this we can re-use the code from the previous exercise in 3.1 (step 6-7), but we have to adjust which property is to be used as the label for each image on the x-axis. Can you find out what we should type in the place of the blue box?

```
// Set chart options
var chartOptions = {
 title: 'Monthly precipitation anomaly',
 hAxis: {title: 'Time'},
 vAxis: {title: 'Precipitation (mm)'},
};

// Create chart
var anomalyChart = ui.Chart.image.series({
  imageCollection: anomalyMonth,
  region: Sittaung,
  reducer: ee.Reducer.mean(),
  scale: 5000,
  xProperty: '■'
}).setOptions(chartOptions);

print('precipitation anomalies chart', anomalyChart);
```

Questions

3. After running the script, you will get a chart very similar to the chart in Figure 10. When you open de graph in a new tab (via pop-up icon) you can inspect the values by hovering over the line with your mouse pointer.
   a) In general, what does it mean when a month has positive values? and how about the negative values?
   b) Inspect the positive rainfall spikes. Can you relate these rainfall anomalies to flooding events that have occurred in recent years?

……

……

……

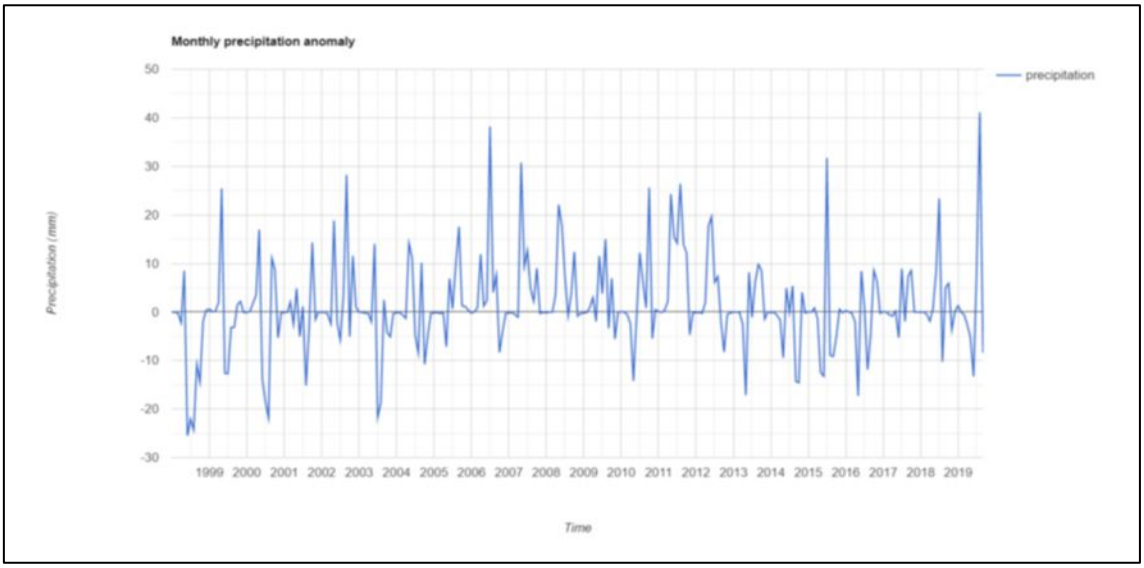


Figure 10. Precipitation anomaly chart for Sittaung basin

Step 7. To understand where the rainfall anomalies occurred, we will select a month with a high (positive) anomaly and display the image containing that data on the Map. To do this, we first must convert the ImageCollection to a List. Then, with the .get() method we can select a single month from the list. As you can see in the chart, there was a very big rainfall spike very recently. Which number do we have to fill in place of the ▢ to select that month? Also try to set the range of rainfall amounts (by adjusting min: ▢ and max: ▢ to get a visually pleasing result

```
// Select a single monthly anamoly image
var listImages = anomalyMonth.toList(anomalyMonth.size()); // to list
var singleImage = ee.Image(listImages.get(▢)); // select list element

// Define the visualization parameters.
var prVis = {
  min: ▢,
  max: ▢,
  palette: ['001137', '0aab1e', 'e7eb05', 'ff4a2d', 'e90000'],
};

// Add image to canvas
Map.addLayer(lastImage.clip(Sittaung), prVis, 'precipitation anomaly');
```

Question

4. Which part of the Sittaung basin was responsible for the big rainfall anomaly? Does that coincide with recent flooding events?

……

……

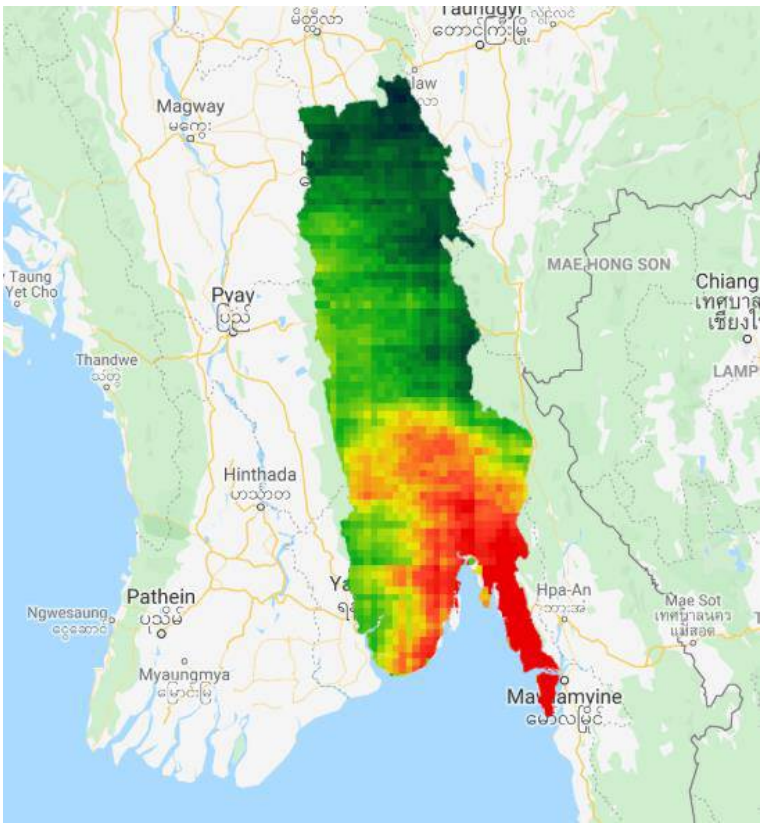


Figure 11. Precipitation anomaly map of Sittaung basin

Don't forget to save your script! The full code corresponding with this exercise can be found here: https://code.earthengine.google.com/bf44b26d21c4c284b1d199945d189ef2

3.3 Comparing satellite rainfall products (bonus)

CHIRPS is a popular satellite rainfall product because of its (relatively) high spatial resolution. It is, however, by no means the only precipitation data available on the GEE. Below is a table with the five satellite rainfall products available and its characteristics:

Table 1 Satellite rainfall products in GEE

Product	Method	Cell size	Temporal resolution	Spatial extent	Time span
CFSV2	GFS, GFDL MOM4, NOAH	0.5^{o}	6-hourly	Global	?? - ??
CHIPRS	Infrared and microwave (TRMM), CFSv2, CHPClim, gauge data	0.05^{o}	Daily	50°N-50°S	?? - ??
GLDAS2-1	Infrared SSMI/SSMIS) and passive microwave (GOES, POES), gauge data	0.25^{o}	3-hourly	90°N-60°S	?? - ??
PERSIANN	Infrared (GridSat-B1), gauge data (GPCP)	0.25^{o}	Daily	60°N-60°S	?? - ??
TRMM	Microwave (TMI, SSMI, AMSU	0.25^{o}	3-hourly	50°N-	?? - ??

	and AMSR), Infrared (GMS), gauge data			50°S	

Step 1. Go to https://developers.google.com/earth-engine/datasets/catalog and have a closer look at the five satellite rainfall products by adding the following tags to the hyperlink above:

1. CFSV2 : /NOAA_CFSV2_FOR6H
2. CHIPRS : /UCSB-CHG_CHIRPS_PENTAD
3. GLDAS2-1 : /NASA_GLDAS_V021_NOAH_G025_T3H
4. PERSIANN : /NOAA_PERSIANN-CDR
5. TRMM : /TRMM_3B42

Although each rainfall product is different in terms of method and resolution, what they all have in common is that under dataset availability each dataset is listed as available from a starting time until Present. We are going to check what this term Present means for each dataset. To do this type the following into the code editor:

```
/* Declare function to get date range
of precipitation ImageCollections */

var getDates = function(imCollect) {
  var first = imCollect.limit(1, 'system:time_start').first();
  var last = imCollect.limit(1, 'system:time_start', false).first();
  var dateFirst = ee.Date(first.get('system:time_start'));
  var dateLast = ee.Date(last.get('system:time_start'));
  var firstLast = [dateFirst, dateLast];
  return(print(imCollect.get("system:id"),firstLast));
};

// Call the function
var chirpsDate = getDates(chirps.select('▢'));
var trmmDates = getDates(trmm.select('▢'));
var cfsv2Dates = getDates(cfsv2.select('▢'));
var gldas21Dates = getDates(gldas21.select('▢'));
var parsiannDates = getDates(parsiann.select('▢'));
```

Question

1. List the dataset availability for each rainfall product in Table 1. Which dataset contains the most recent precipitation data, and which the oldest? Taking all the characteristics of each rainfall product into account, which would you find most useful?

……

……

……

Step 3. We are now going to use an advanced script to compare the precipitation data of each of the five satellite rainfall products. Instead of building the script ourselves step-by-step we will load it directly into our browser. After running the script, we will obtain the average annual precipitation sums for Myanmar for a 10-year period from 2009 until 20018:
https://code.earthengine.google.com/73081931ec67ce4b7a411d8e5d782f3b

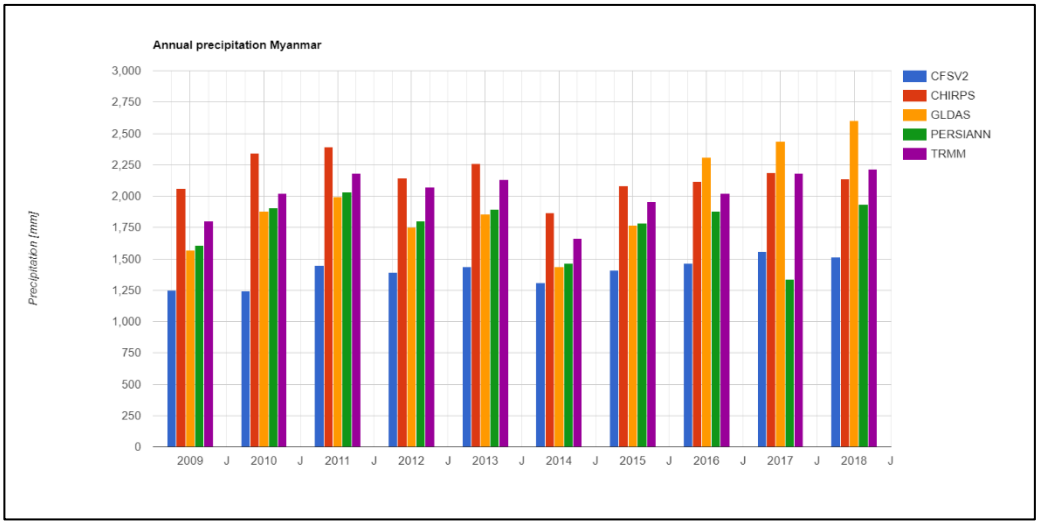


Figure 12. Annual rainfall for Myanmar

Question

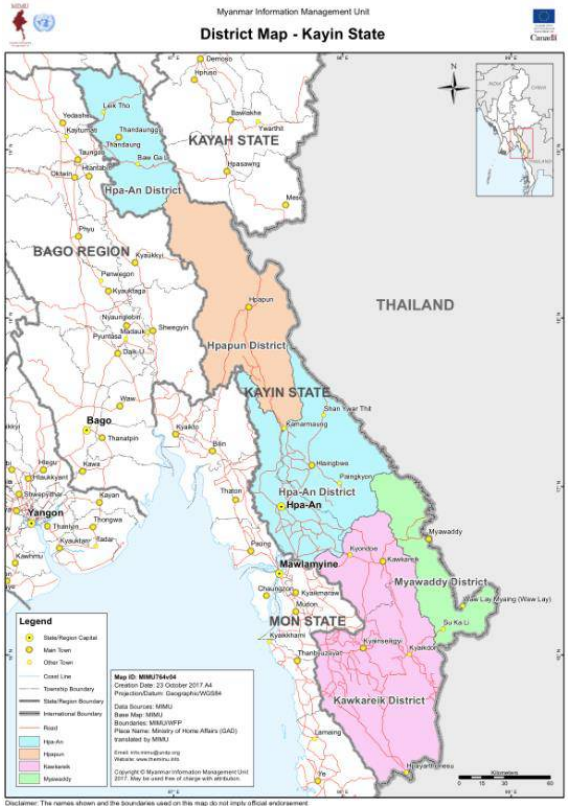
2. From you own expert knowledge, which of the five datasets would you consider the most accurate? Do you know of an (external) source of information to check the average annual precipitation sums for Myanmar?

……

……

……

3.4 Flood risk analysis



Floods in Myanmar happen every year. They are mostly beneficial. By flooding the area becomes more fertile and the area is heightened by the deposition of sediment. However, when flooding takes place in a highly inhabited area such as a city or town it can be dangerous.

Therefore, to increase awareness of danger levels and prevent new houses being built in the areas with highest risk, it is important to know where most people live and which areas are most prone to flooding. In this chapter we will work on both these cases.

We will focus on the areas Kayin and Mon State. They have suffered severe flooding in the 2018 en 2019 monsoon seasons.

3.5 Population

To get an idea of the population density in Myanmar we can use open source data available in Google Earth Engine. We use the 'GHSL: Global Human Settlement Layers, Population Grid 1975-1990-2000-2015 (P2016)' for this. This source automatically generated the population data on population for the whole world on a 250x250 m grid cellsize. They do not use land survey or information from the countries on population count. Therefore, it is not very precise but gives you a first insight in the areas with a growing population in Myanmar.

Step 1. Look-up and read the description of the 'GHSL: Global Human Settlement Layers, Population Grid 1975-1990-2000-2015 (P2016)'
Tip: use the searchbar at the top of the code editor

Question
1. What is the Image Collection ID of this dataset?
……

Question
2. This dataset only has one band. What is the name and description of this band?
……

Using the Image Collection ID of the dataset you can import the dataset into your code.

Step 2. Import the 'GHSL: Global Human Settlement Layers, Population Grid 1975-1990-2000-2015 (P2016)' into a new script ('Population'). And select the oldest available data as a filter ('1975').

```
var dataset = ee.ImageCollection('JRC/GHSL/P2016/POP_GPW_GLOBE_V1')
                  .filter(ee.Filter.date('1975-01-01', '1975-12-31'));
```

Step 3. Select the band you want to show on the map (this dataset only has one band, nonetheless you still need to select it). Call the variable 'populationCount'.

```
var populationCount = dataset.select('population_count');
```

We have imported the GHSL-dataset as an image collection. Even though this image 'collection' only has one image in it, because it is stored as an image collection it can't be displayed directly on the map. And we have not given the command yet to display it on the map.
Tip: to check the number of images in a data collection use 'print(dataset)' to show the number of features in the console.

To reduce the image collection to just one image you can use a 'reducer'.
"Reducers take an input dataset and produce a single output. When a single input reducer is applied to a multi-band image, Google Earth Engine automatically replicates the reducer and applies it separately to each band."
More information: https://developers.google.com/earth-engine/reducers_intro

Step 4. Use the 'mean' reducer on the dataset to bring it back to only one image which you can show on the map.

```
var populationCount1975 = populationCount.reduce(ee.Reducer.mean());
```

We want to visualize this dataset in a map. For that, we need to give them a color palette. The colors are defined using the web standard CSS color value scheme (https://en.wikipedia.org/wiki/Web_colors). Colors can be specified by name or strings indicating the combination of red, green and blue. The lowest value in any of the three positions is 00 (representing the decimal number 0), while the highest is FF (representing the decimal number 255). The string '000000' represents the color black, 'FFFFFF' is white, 'FF0000' is red, '00FF00' is green, and '0000FF' is blue.

But what colors should we choose? Google Earth Engine offers a suggestion. You can browse all available datasets in the Data Catalog: https://developers.google.com/earth-engine/datasets/catalog/ . If you scroll to the bottom, you can always find a few lines of code to visualize the data. In the case below, a search for 'JRC/GHSL/P2016/POP_GPW_GLOBE_V1' gave the following result:

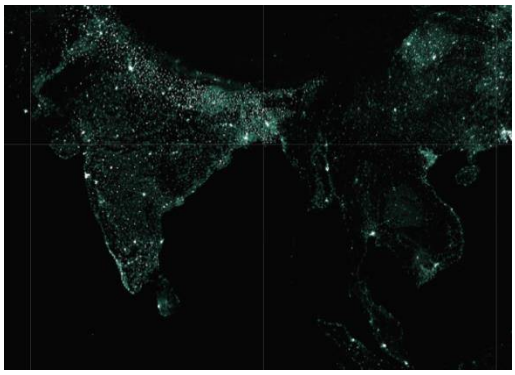
```
var populationCountVis = {
  min: 0.0,
  max: 200.0,
  palette: ['060606', '337663', '337663', 'ffffff'],
};
```

Question
3. What do you think of the proposed minimum number for population count?
……

Step 5. Insert the proposal of the developers.google.com website and use 'Map.addLayer' to add the layer with these characteristics to the map.

```
Map.addLayer(populationCount1975, populationCountVis, 'Population Count');
```

Question
4. Why is the image so dark?
……



As you can see Myanmar is almost completely black (and hard to find) in this image. This is because the number of people living in Myanmar is not very large compared to some other countries and people live quite spread around the country.

Question
5. What could you do to get more information from this image for Myanmar?
……

At the moment it is even hard to find Myanmar on the worldmap because in the seas no people live either. Therefore, it is hard to distinguish countries. This is why we will clip the image on Myanmar (show only Myanmar).

Step 6. Use the code below to define the country of Myanmar on the map.

```
var countries = ee.FeatureCollection("ft:1tdSwUL7MVpOauSgRzqVTOwdfy17KDbw-1d9omPw")
// Define country name
var country_names = ['Myanmar (Burma)'];

// Find the country in the countries list
var Country = countries.filter(ee.Filter.inList('Country', country_names)).geometry();
```

Step 7. To only show Myanmar population density change the code of Step 5 to:

```
Map.addLayer(populationCount1975.clip(Country), populationCountVis, 'Population Count');
```

Step 8. Find the colours you would use to enhance the information on population in Myanmar and change the ‘palette’ in that way. You could also increase the number of colours you use.

Example:
```
palette: ['FFFFFF','00ff04','075e09','0000FF','FDFF92','FF00E7']
```

To give some extra information with your map you can add a legend. Below an example of a legend is given. In red all the lines that you can or need to change are given. The ‘palette’

needs to be the exact same as your ‘palette’ in the map of course. The names in the legend should correspond with the number of colours in the palette and the difference between the min and the max in your map.

In the legend it is important that in ‘// Add color and names’ the ‘6’ (number) in red corresponds to the number of colours in the palette.

Step 9. Add a legend to your map and play around with it.

```
// set position of panel
var legend = ui.Panel({
  style: {
    position: 'bottom-left',
    padding: '8px 15px'
  }
});

// Create legend title
var legendTitle = ui.Label({
  value: 'Number of people per cell (250m) 1975',
  style: {
    fontWeight: 'bold',
    fontSize: '18px',
    margin: '0 0 4px 0',
    padding: '0'
    }
});

// Add the title to the panel
legend.add(legendTitle);

// Creates and styles 1 row of the legend.
var makeRow = function(color, name) {

      // Create the label that is actually the colored box.
      var colorBox = ui.Label({
        style: {
          backgroundColor: '#' + color,
          // Use padding to give the box height and width.
          padding: '8px',
          margin: '0 0 4px 0'
        }
      });

      // Create the label filled with the description text.
      var description = ui.Label({
        value: name,
        style: {margin: '0 0 4px 6px'}
      });

      // return the panel
      return ui.Panel({
        widgets: [colorBox, description],
        layout: ui.Panel.Layout.Flow('horizontal')
      });
};

//  Palette with the colors
var palette =['FFFFFF','00ff04','075e09','0000FF','FDFF92','FF00E7'];

// name of the legend
var names = ['0','20','40','60','80','100']
```

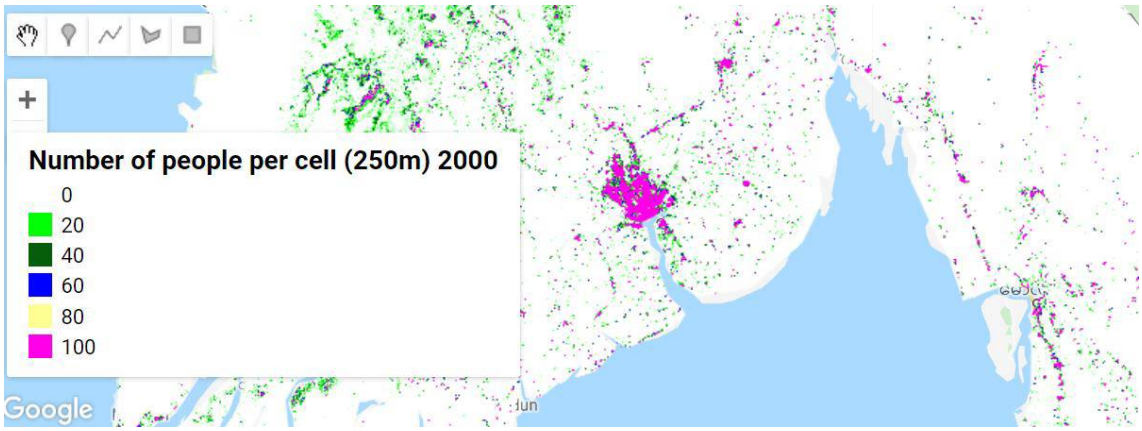
```
// Add color and and names
for (var i = 0; i < 6; i++) {
  legend.add(makeRow(palette[i], names[i]));
  }

// add legend to map (alternatively you can also print the legend to the console)
Map.add(legend);
```

Step 10. In the previous steps we have made a map of the population of Myanmar in 1975. Save your work and do the same for the population in 2015.

Question

6. What stands out when you compare the population images of 1975 and 2015? Does this correspond with your knowledge of population growth in Myanmar?

……..



Don't forget to save your script! The full code corresponding with this exercise can be found here: https://code.earthengine.google.com/0cbe55cefafc605cdc13c7fa6a95a5c6

3.6 Flood frequency

'A flood is an overflow of water that submerges land that is usually dry.'
Like we already mentioned at the start of this chapter: this is not always a bad thing. For farmers it can be very convenient that their land floods. This increases the fertility of the soil. Furthermore, traditional housing in Myanmar is on poles and therefore it takes a very large flooding before people actually have to evacuate their houses.

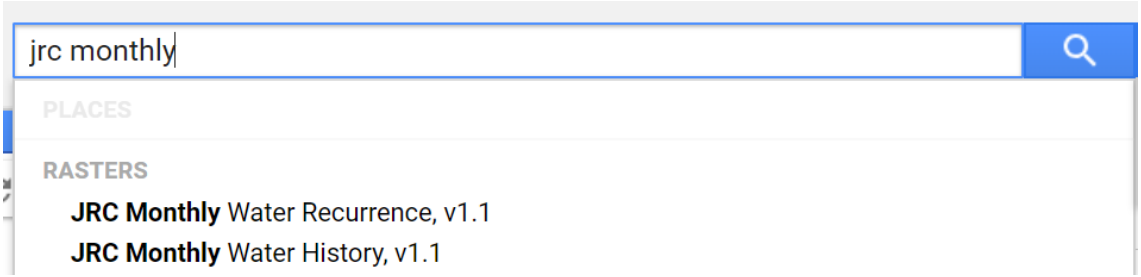
Flood maps can be used to assess affected regions after a flood. However, over a longer timespan they can also be used in spatial planning. If a region is prone to flooding and was flooded regularly in history (because it is a floodplain for example) it might not be the best idea to build in this area unless you take measures.

To make your own historic flood map we will work with the JRC Monthly Water History dataset, v1.1[1]. This dataset contains maps which show you where there was water from 1984 to 2015

[1] Jean-Francois Pekel, Andrew Cottam, Noel Gorelick, Alan S. Belward, High-resolution mapping of global surface water and its long-term changes. Nature 540, 418-422 (2016). (doi:10.1038/nature20584)

and also provides statistics on the occurrence interval of water at a certain location. The resolution is 30x30 meters, the information is clustered in monthly data.

Step 1. In the searchbar type ‘JRC Montly’, several datasets will be shown. Select the ‘JRC Montly Water History, v1.1’. Read the introduction to this dataset and click the ‘Import’ button.



The JRC-dataset will appear at the top of your screen.

Step 2. The dataset is called ‘imageCollection’ change this to ‘jrc’.

```
var jrc: ImageCollection "JRC Monthly Water History, v1.1"
```

Step 3. Define a start and an end date by creating a startDate and endDate variable

```
// Define study period
var startDate = ee.Date.fromYMD(2015, 1, 1);
var endDate = ee.Date.fromYMD(2018, 12, 31);
```

Step 4. Filter the JRC-data by country (see paragraph Population) and by start- and enddate

```
// Filter JRC data by country and period
var myjrc = jrc.filterBounds(Country).filterDate(startDate, endDate);
```

Checking of your data is very important in every study, and also when working with Google Earth Engine.

Step 5. Check the completeness of your JRC-data you are using by using the print-function.

```
print(myjrc)
```

In the console on the right you should see in the features list all the months of the area between the start- and enddate that you selected.

We want to build a flood frequency map. To do so we need to know for every gridcell how often it had water in the study period. We can do this by defining how often water is detected in a certain gridcell divided by how many observations there were. If you multiply this number by a 100, you get the percentage of the time that water was present in the gridcell.

Step 6. To figure out how to do this by yourself will be a bit too hard we thought. Therefore we give you the first bit, it is described below.

```
// Detect observations
var myjrc = myjrc.map(function(img){
  // observation is img > 0
  var obs = img.gt(0);
  return img.addBands(obs.rename('obs').set('system:time_start', img.get('system:time_start')));
});

// Detect all observations with water
var myjrc = myjrc.map(function(img){
  // if water
  var water = img.select('water').eq(2);
  return img.addBands(water.rename('onlywater').set('system:time_start', img.get('system:time_start')));
});

// Calculate the total amount of observations
var totalObs = ee.Image(ee.ImageCollection(myjrc.select("obs")).sum().toFloat());
```

Now you have defined the total number of observations and also defined a selection criteria for only water.

Step 7. Can you think of how to define 'only water' in a gridcell yourself? Afterwards you would have to divide this only water by the totalObs and multiply it by 100 to get the percentage.

```
// Calculate all observations with water
var totalWater = ee.Image(ee.ImageCollection(myjrc.select("onlywater")).sum().toFloat());

// Calculate the percentage of observations with water
var floodfreq = totalWater.divide(totalObs).multiply(100);
```

We now want to mask the areas that have never had water, we do not see those. They have a very low risk of flooding.

Step 8. Mask the areas that never have had water in your study period.

```
// Mask areas that are not water
var myMask = floodfreq.eq(0).not();
floodfreq = floodfreq.updateMask(myMask);
```

Step 9. Add a palette and clip the image by country as you have done in paragraph Population.

Step 10. Add a legend to go with this flood frequency map in the same way as you have done in paragraph Population.

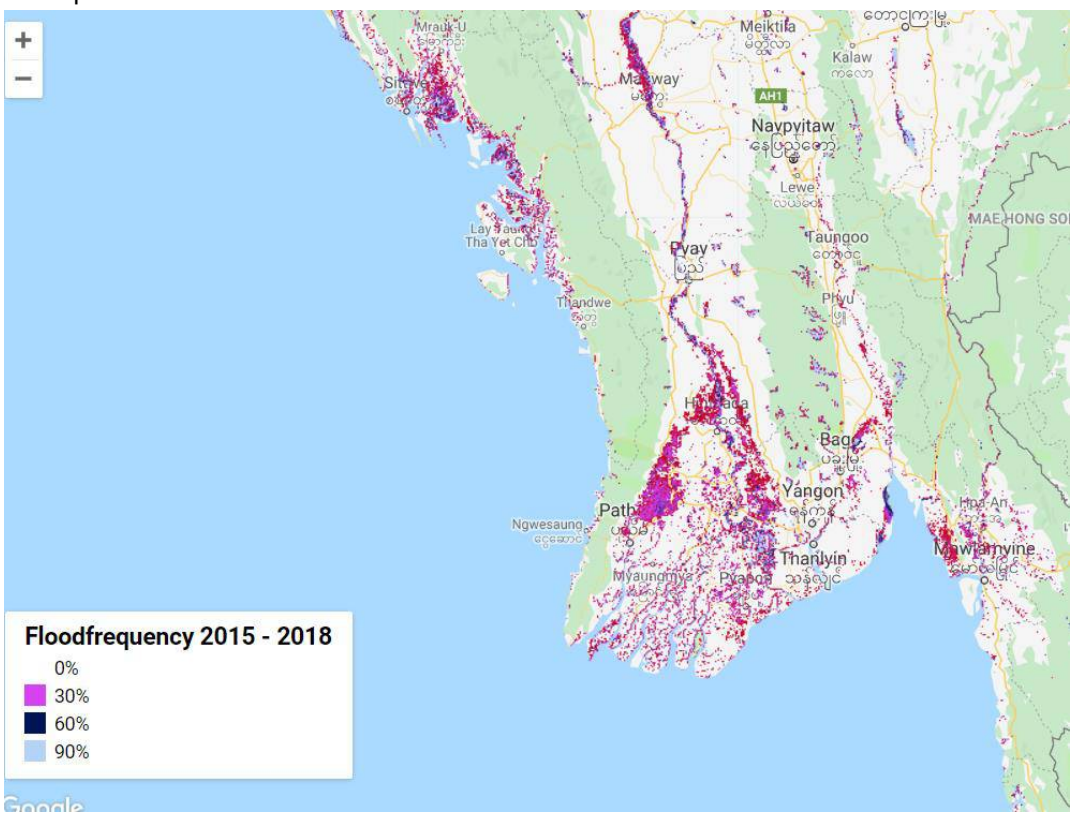
Now you have created a flood frequency map that indicates which areas in Myanmar are most prone to flooding. As you can see the area around Yangon has a large flooding frequency as well as the region around Mawlamyine.

Question

1. Why are these areas most prone to flooding?

……..

Example



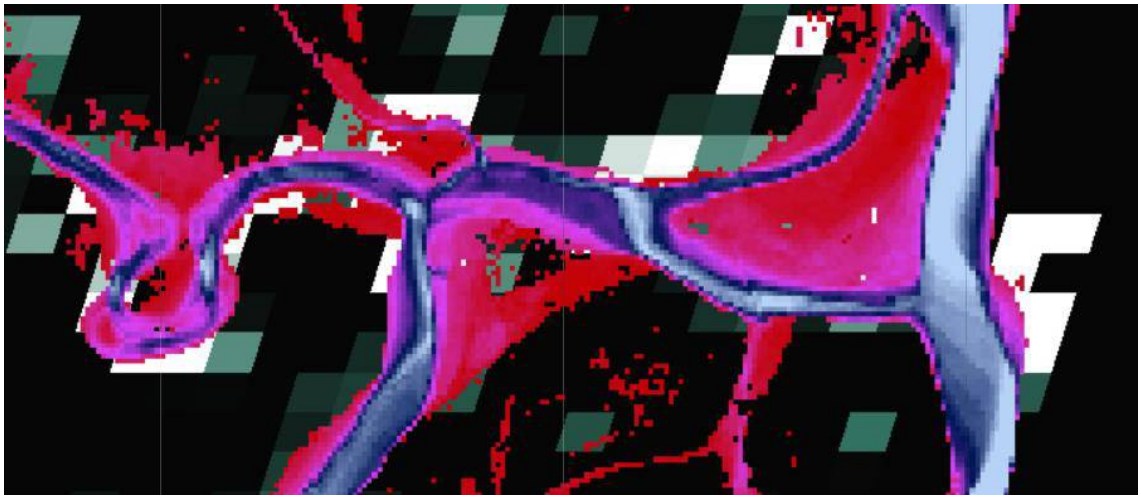
Don’t forget to save your script! The full code corresponding with this exercise can be found here: https://code.earthengine.google.com/6d87bca44f454179e50bf5dc00eba83b

3.7 Flood risk

In this paragraph we will combine the information of paragraph Population and Flood frequency. We will find out which areas in Myanmar have the highest flood risk. Flood risk is a combination of flood frequency combined with number of residents in an area.

Step 1. Combine the scripts of paragraph Population and Flood frequency.

If you zoom in you get something like the figure below.



Question
1. What stands out when you look at the combined image of Population and Flood frequency?
……

Question
2. Wat is the gridcellsize of the population map, and what is the gridcellsize of the flood frequency map?
……

Step 2. We can lengthen the flood frequency period to cover flood frequency from 2000 onwards. Change the startdate of the flood frequency to the 1st of January 2000.

```
// Define study period
var startDate = ee.Date.fromYMD(2000, 1, 1);
var endDate = ee.Date.fromYMD(2018, 12, 31);
```

Now we will multiply the two datasets. We will do this by using the function 'multiply' in Google Earth Engine.
See this page for more information on Mathematical Operations in Google Earth Engine:
https://developers.google.com/earth-engine/image_math

Step 3. Use the 'multiply' function to combine the two datasets.

```
// Risk assessment, more people and more regular floodings is a larger problem
var Risk = floodfreq.multiply(populationCount2015)
```

We have to think of a colourcoding and the minimum and maximum that we want to take into account in the flood risk map. The flood frequency has a value of 0.1 to 100. The population can go up to 100 people per gridcell in Myanmar.

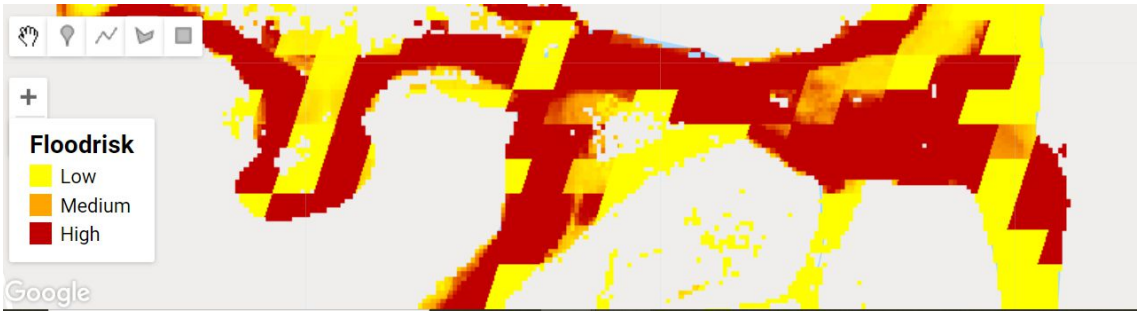
Question
3. What is the maximum value that can be reached in the flood risk map?
……

Step 4. Think of your own colour coding and add the flood risk layer to the map.

Example (not necessarily a good one)

```
var viz2 = {min:0, max:1000, palette:['ffff00', 'ffa500', 'c00000']};
Map.addLayer(Risk.clip(Country),viz2,"Risk")
```

Step 5. Add also a legend to go with this flood risk map.

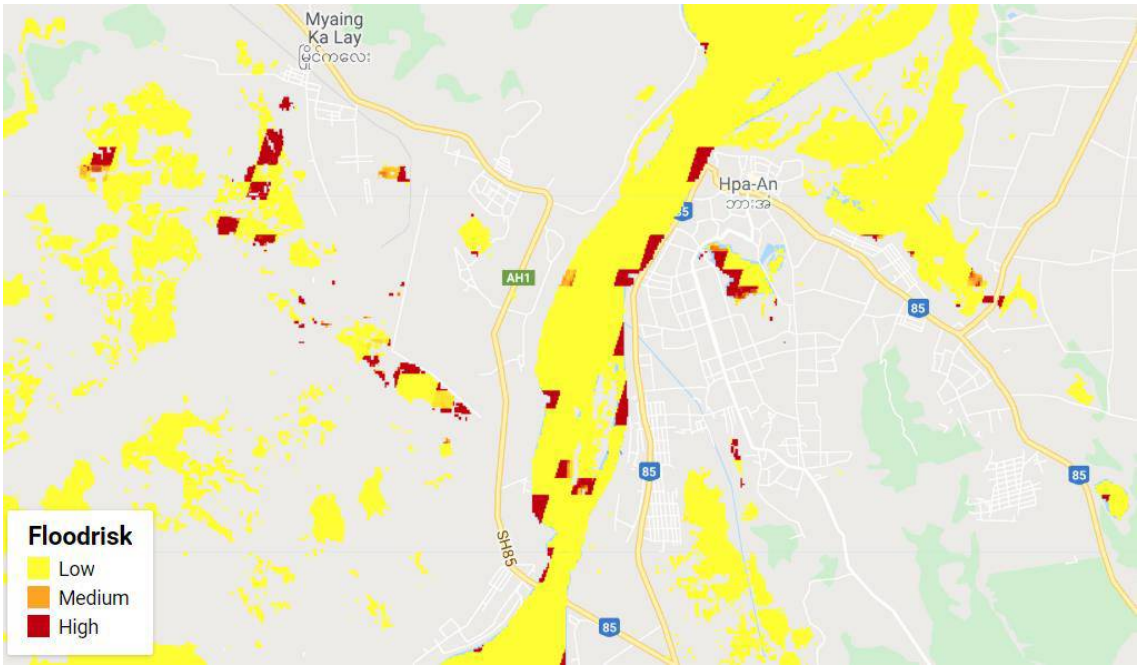


Step 6. Zoom in on the city of Hpa-An. This might take a while because of the large amount of data.

Question

4. What areas do you recognize on the risk map? Why is this?

……..



Question

5. Why do we see that some areas that seem to be in the river have a high flood risk?

……

Question

6. There is one specific area in Hpa-An city centre with high flood frequency. What area is this?

……

Don’t forget to save your script! The full code corresponding with this exercise can be found here: https://code.earthengine.google.com/42ec3541531e047a3bedaade024fbc5e

3.8 Rescue shelters

In this paragraph we will work in the same way as you would have to define your individual assignment in the second training week. Therefore, we have chosen the subject for you: ‘Where to build a rescue shelter in Bilin?’

A research question that could be answered with Google Earth Engine is ‘What are the best places to build a rescue shelter in Bilin considering flood risk and height of the area?’

Question
1. What do you know about the suitable height for a rescue shelter?
……

The flood risk maps were already made in in the previous paragraph. Therefore we will focus now on the height of the area. For this we will use the SRTM Digital Elevation Data 30 m.

Step 1. Look up this image of SRTM, import it and call it ‘dem’

```
var dem: Image "SRTM Digital Elevation Data 30m" (1 band)
```

Of course people in Bilin need to be able to reach the rescue shelter. Therefore, the search area for a suitable location for a rescue shelter needs to be close to Bilin.

Step 2. Draw a polygon around Bilin in which you want to look for a suitable location for a rescue shelter.

Step 3. Map the dem of only this polygon. Fill the blue box with the name of the polygon you defined and fill the min and max with suitable numbers (play around with it).

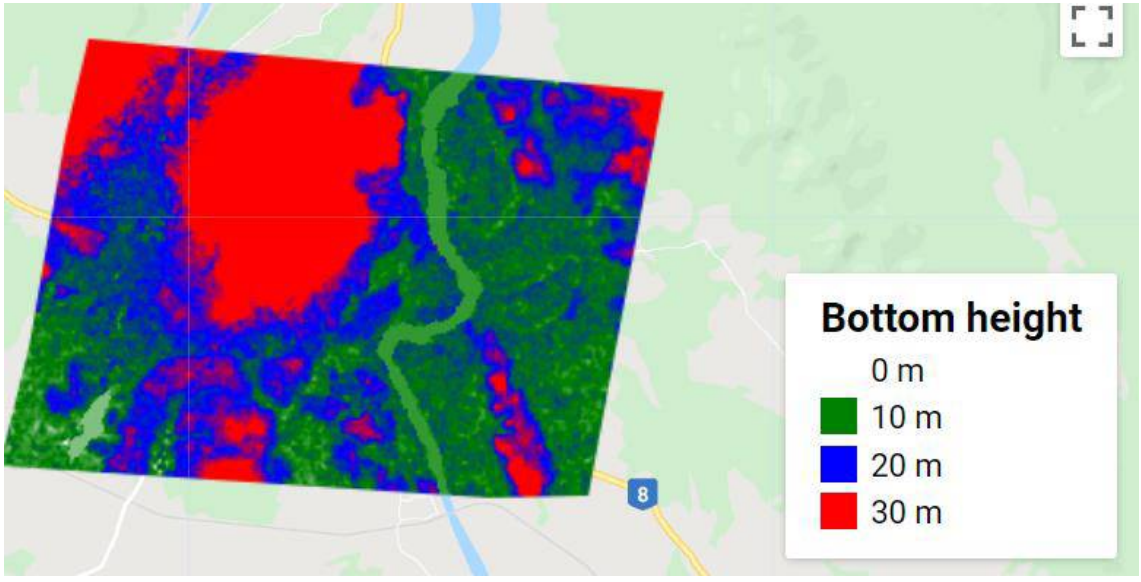
```
var demClip = dem.clip(          )
Map.addLayer(demClip,{'min':  ,'max':  },"dem Myanmar")
```

You could improve the image by defining colour in the image.

Step 4. Therefore, make your own customized layout. You would have to play around with the min and max to get as much distinction as you can in the image.

```
//Make a customized layout
Map.addLayer(demClip,{min:  ,max:  , palette: ['FFFFFF','008000','0000FF','FF0000']},
'dem Myanmar colour')
```

Step 5. Add a legend to the map (use the code of the previous paragraph ‘Population’ for this)



Step 6. Export the DEM-map you made using the below code. Fill the correct values and colours in the blue boxes:

```
// export map to jpeg file and print download url
var map = demClip.visualize({bands:"elevation",
min:[redacted],
max:[redacted],
palette:['[redacted]','[redacted]','[redacted]','[redacted]'],
forceRgbOutput:true});

var url = map.getThumbURL({dimensions:'800x800',
format:'jpg'});
print(url)
```

Don't forget to save your script! The full code corresponding with this exercise can be found here: https://code.earthengine.google.com/a30cfe9192b14d2922304182115de4ac

Step 7. Save your dem-script of Bilin as 'DEM' and go to your flood risk script. Add the dem-map and the polygon to your flood risk script. Copy the code of the dem-script to the bottom of your flood risk script and run it.

Question
2. Circle the best possible areas for a rescue shelter considering only height and flood risk.

Question
3. What other essential prerequisites of rescue shelters do we not take into account in this exercise?
……

Don't forget to save your script! The full code corresponding with this exercise can be found here: https://code.earthengine.google.com/884e1b5e387226fb5399b67df89dfc80

4 GEE applications for River Morphology

Rivers in Myanmar are highly dynamic and are among the least disturbed rivers in the world. The dynamics can cause problems when they are not understood.

4.1 Seasonal changes

One of the characteristics of Myanmar is it strong differences between the wet and the dry season. These seasonal changes have a large impact on the width of the river. During the monsoon season a lot more water flows through the rivers than in the dry season. This causes a wider river during the monsoon season, and changes in river morphology after the monsoon. The website we will work with in this exercise is based on Google Earth Engine and shows the erosion (loss of land) and deposition (land gain) after the monsoon.

Go to this website and read the provided description[1]: https://servir.adpc.net/tools/dancing-rivers

First click 'Launch Tool'. If you click the word 'Map' in the upper right corner of this website you can recognize the cities of Myanmar.

Question
1. Select a year on the left of which you know the monsoon rains were heavy. What areas show most changes and why (red and yellow)?
……

Question
2. Do you understand specific river characteristics concerning erosion and sedimentation that you see in the 'Dancing Rivers' maps?
……

Using this tool you can get a first insight in the morphologically active and stable channels. Be aware, larger morphological changes and channel shifting takes place over multiple years.

4.2 River definition

To get insight in the larger morphological changes over the years we will also depict erosion and deposition but over multiple years instead of over seasons.

To do so we will first look at the quality of the data by filtering out the clouds of the satellite images we will use.

Cloud filtering
When we use satellite images, often clouds disturb these images. Optical remote sensing technology is unfortunately not able to penetrate clouds. This exercise uses the Landsat 8 Collection.

[1] Developer(s)
ADPC/SERVIR-Mekong, Stockholm Environment Institute (SEI),Spatial Informatics Group (SIG)

Question

1. What is the ImageCollection ID of the 'USGS Landsat 8 Collection 1 Tier 1 TOA Reflectance'?

……

Step 1. Define the variable for the 'USGS Landsat 8 Collection 1 Tier 1 TOA Reflectance' dataset and give it a useful name.

```
var [redacted] = ee.ImageCollection('[redacted]');
```

For the purpose of this exercise we do not use information on the whole world. Instead we want to focus on the area around Hpa-An.

Question

2. What is the advantage of selecting only a small section of the Landsat 8 collection?

……

Step 2. Because we will focus on the area around Hpa-An go to 'Geometry Imports' underneath the code editor. Click '+ new layer' and add a point just north of Hpa-An.

You will see this point appear as a variable in the Imports-section at the top of the code editor.

Step 3. Give this point the name 'Hpa_An' by clicking ones on the word 'geometry' in the imports section.

Step 4. Now we will filter on the tiles of the Landsat images which cover our geometry point.

```
var spatialFiltered = [redacted].filterBounds(Hpa_An);
print('spatialFiltered', spatialFiltered);

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

Question

3. What needs to be at the location of the blue box?

……..

Question

4. Look at the printed information on 'spatialFiltered' and 'temporalFiltered' in the Console. By filtering the image collection on only 2015 by how many features is the Landsat 8 image collection reduced? Check the dates in the 'temporalFiltered' are they indeed all of 2015?

……

For this exercise it does not matter from which month the image is used, as long as it is in 2015. The 'USGS Landsat 8 Collection 1 Tier 1 TOA Reflectance' has an image property called 'Cloud_Cover' which defines the percentage of clouds in the image (see the description of this dataset).

Question

5. In which months will you have the most cloudy images in general?

……

Step 5. Sort the images in 'temporalFiltered' from least to most cloudy and select the least cloudy image.

```
// This will sort from least to most cloudy.
var sorted = temporalFiltered.sort('[redacted]');

// Get the first (least cloudy) image.
var scene = ee.Image(sorted.first());
```

These steps can also be shortened, see the steps below for 2014. In the blue box the name of the dataset Landsat 8 should be filled in.

```
// Get the least cloudy image in 2014.
var image2014 = ee.Image(
  [redacted].filterBounds(Hpa_An)
    .filterDate('2014-01-01', '2014-12-31')
    .sort('CLOUD_COVER')
    .first()
);
```

NDWI

The NDWI is the Normalized Difference Water Index. It is used for water mapping.

"The NDWI index is most appropriate for water body mapping. Waterbodies have a strong absorbability and low radiation in the range from visible to infrared wavelengths. The index uses the green and Near Infra-red bands of remote sensing images based on this phenomenon. The NDWI can enhance the water information effectively in most cases. It is sensitive to built-up land and often results in over-estimated water bodies."

Values description: Values of water bodies are larger than 0.5." (Sentinel Hub, https://www.sentinel-hub.com/eoproducts/ndwi-normalized-difference-water-index)

The NDWI is expressed as follows (McFeeters 1996):

$$NDWI = \frac{Green - NIR}{Green + NIR} \quad (1)$$

where *Green* is a green band such as TM band 2, and *NIR* is a near infrared band such as TM band 4.

This index is designed to (1) maximize reflectance of water by using green wavelengths; (2) minimize the low reflectance of NIR by water features; and (3) take advantage of the high reflectance of NIR by vegetation and soil features. As a result, water features have positive values and thus are enhanced, while vegetation and soil usually have zero or negative values and therefore are suppressed (McFeeters 1996). [1]

Question

6. Which bands correspond with the Green and Near InfraRed in the Landsat 8 image collection?

……

[1] Xu, H. (2006) Modification of normalised difference water index (NDWI) to enhanceopen water features in remotely sensed imagery. International Journal of Remote Sensing Vol. 27, No. 14, 3025–3033

Step 6. Make the NDWI for the 2014 image and add this layer to the map.

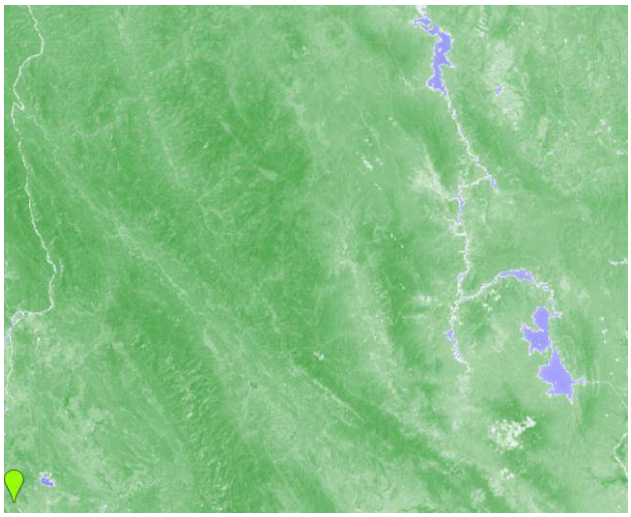
```
// Compute the Normalized Difference Water Index (NDWI) for 2014.
var nir2014 = image2014.select('B5');
var green2014 = image2014.select('B3');
var NDWI2014 = nir2014.subtract(green2014).divide(nir2014.add(green2014)).rename('NDWI2014');

// Display the result for 2014.
Map.centerObject(image2014, 9);
var ndwiParams2014 = {min: -1, max: 1, palette: ['blue', 'white', 'green']};
Map.addLayer(NDWI2014, ndwiParams2014, 'NDWI image2014');
```

Step 7. Repeat the steps above for the 2018 image.

Question

7. Describe a few differences between the 2014 and 2018 image.

……



Don't forget to save your script! The full code corresponding with this exercise can be found here: *https://code.earthengine.google.com/ce6379b7e145ef761094c35987ade583*

Centerline of river

One of the advantages of GEE is that it is Open Source software. This enables users to use scripts and FeatureCollections from others as well. One of those scripts we can use is the centerline of the river data. This is worldwide data on centerlines of rivers.

One of the disadvantages of using pre-setup FeatureCollection is that it is harder to know what analysis was done to make this FeatureCollection. It is therefore very important to read all the information to understand these FeatureCollections and be critical about it.

https://www.researchgate.net/figure/The-Global-River-Widths-from-Landsat-GRWL-Database-contains-more-than-58-million_fig1_326049115

Step 8. Start a new script and add the "users/eeProject/grwl" FeatureCollection

```
var grwl = ee.FeatureCollection("users/eeProject/grwl") // Global database of river centerline, RivWidthCloud Paper
```

You can imagine that mapping the centerlines of rivers of the whole world might take a while. Therefore we want only to display the centerlines of rivers in Myanmar.

Step 9. To do so, make sure you import the 'Myanmar' outline again and of the centerlines only display the centerlines of Myanmar on a map.

```
var countries = ee.FeatureCollection("ft:1tdSwUL7MVpOauSgRzqVTOwdfy17KDbw-1d9omPw")

// Define country name
var country_names = ['Myanmar (Burma)'];

// Find the country in the countries list
var Country = countries.filter(ee.Filter.inList('Country', country_names)).geometry();

var centerlineFiltered = grwl.filterBounds(Country)
Map.addLayer(centerlineFiltered,{}, 'centerline')
```

Don't forget to save your script! The full code corresponding with this exercise can be found here: https://code.earthengine.google.com/1e507669a68c0a6841555ab52e1adbd6

4.3 Erosion/sedimentation

In this paragraph we will show the erosion and sedimentation patterns of rivers in Myanmar over multiple years. This gives insight in the dynamics of rivers. For this exercise the 'USGS Landsat 5 TM Collection 1 Tier 1 TOA Reflectance' is used.

Step 1. Look-up this collection and read through its bands and image properties.

Question
1. What is the data availability and the revisit interval?
……

For this we will use 4 images of 1990, 1995, 2000 and 2005 from the Image Collection.

Step 2. Use the below code to select these specific images from the imagecollection.

```
// Manual imagecollection selection
var imagecollection = ee.ImageCollection(['LANDSAT/LT05/C01/T1_TOA/LT05_133048_19900111',
'LANDSAT/LT05/C01/T1_TOA/LT05_133048_19950109', 'LANDSAT/LT05/C01/T1_TOA/LT05_133048_20000123',
'LANDSAT/LT05/C01/T1_TOA/LT05_133048_20050104']);
```

Question
2. Of which month are all the images?
……..

Now we will start with something a bit more difficult. We will use 'functions' in our next part. To understand this concept it is necessary to read the below text from https://developers.google.com/earth-engine/tutorial_js_03
Also, the first element in a list/string/row is the 0th argument.
Thus this print-request returns '1', the first element in the list:

```
var eeList = ee.List([1, 2, 3, 4, 5]);
var value = eeList.get(0);
print(value); // 1
```

The last element in a list/string/row is the -1th argument.
Thus this print-request returns ‘5’, the last element in the list

```
var eeList = ee.List([1, 2, 3, 4, 5]);
var value = eeList.get(-1);
print(value); // 5
```

If you have done some programming before with different programming languages, you might know ‘for-loops’. These are not used often in Google Earth Engine but we have another solution[1]:

For Loops

The use of for-loops is discouraged in Earth Engine. The same results can be achieved using a `map()` operation where you specify a function that can be independently applied to each element. This allows the system to distribute the processing to different machines.

The example below illustrates how you would take a list of numbers and create another list with the squares of each number using `map()`:

```
// This generates a list of numbers from 1 to 10.
var myList = ee.List.sequence(1, 10);

// The map() operation takes a function that works on each element independently
// and returns a value. You define a function that can be applied to the input.
var computeSquares = function(number) {
  // We define the operation using the EE API.
  return ee.Number(number).pow(2);
};

// Apply your function to each item in the list by using the map() function.
var squares = myList.map(computeSquares);
print(squares);  // [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

You will see some other things in this function that you might not recognize. If you are interested read and test these functions to understand them better:
ee.: https://developers.google.com/earth-engine/tutorial_js_02
CannyEdgeDetector: https://developers.google.com/earth-engine/image_edges
Focal_max: https://developers.google.com/earth-engine/image_morph

‘ee.Array.get()’

get(position) ×

Extracts the value at the given position from the input array.

Arguments:

- **this:array (Array):**
  The array to extract from.
- **position (List):**
  The coordinates of the element to get.

Returns: Number

‘ee.Dictionary.get()’

[1] Developers.google.com

get(key) ×

Extracts a named value from a dictionary.

Arguments:

- this:dictionary (Dictionary)
- key (String)

Returns: Object

Step 3. Type the code below. This is one big function (Determinewatermask) with several other functions inside (WaterMaskFunc, bss, extract channel). The function 'Determinewatermask' will be called later in the script.

```
var Determinewatermask = function(index, geometry, centerline, MaxDist, FILL_SIZE) {

  var WaterMaskFunc = function(image){
    var addNDWI = ee.Image(image).addBands(ee.Image(image).normalizedDifference(['B2','B4']).rename('NDWI'))
    var edge = ee.Algorithms.CannyEdgeDetector(addNDWI.select('NDWI'), 0.4)
    // value 0.4 is minimal difference for the edge
    var scale = 200 // Determine the size of the buffer. In this case 200 square meters,
    //the larger the scale the larger the bufferzone around focal_max.
    var edgeBuffer = edge.focal_max(scale, 'square' , 'meters' ); // create a buffer around the focal_max.
    var EdgeCut = ee.Image(addNDWI).mask(edgeBuffer) //mask the bufferzone around the focal_max
    //edges between land and water
```

Step 4. Make a variable 'threshold' which you set to 0.

```
var threshold = 0
```

Step 5. Define the WaterMask by using the NDWI and the above defined threshold to define the difference between 'water' and 'not water' (land).

```
  // Now determine the water mask
var WaterMask = addNDWI.select('NDWI').gt(threshold)
```

With the WaterMask we have defined for each gridcell if it is water or land. However, to find the sedimentation and erosion patterns of the rivers, we do not want other waterbodies (lakes, fields that are underwater) to disturb this image. Therefore, we will erase all areas from the analysis on sedimentation and erosion that are too far away from the river centerline.

Step 6. We will now create the function 'ExtractChannel' in which this area around the centerline of the river is defined.

```
  // filter noise with the help of the centerline
  var ExtractChannel = function(ctrline, maxDistance) {
  // extract the channel water bodies from the water mask, based on connectivity to the reference centerline.
  // to only show the rivers and not random other lakes etc.
    var cost = WaterMask.not().cumulativeCost({
        source: ee.Image().toByte().paint(ee.Image(centerline), 1).and(WaterMask),
        // only use the centerline that overlaps with the water mask
        maxDistance: maxDistance,
        geodeticDistance: false
      }).eq(0);
    return WaterMask.updateMask(cost).unmask(0).updateMask(WaterMask.gte(0)).rename(['channelMask']);
    };

  var channel = ExtractChannel(centerline, MaxDist)

  return channel

  }

return WaterMaskFunc
}
```

As you can see from all the '}' we have now closed of all the functions that we have created above.

Question

3. What are the names of the functions we have created in this paragraph so far?

……

In the first part of this paragraph we have only created functions. Now we will start using them. To do so we will use the centerlines of rivers that you have used in the paragraph before.

Step 6. Add the FeatureCollection of the river centerlines.

```
var grwl = ee.FeatureCollection("users/eeProject/grwl") // Global database of river centerline
// RivWidthCloud Paper
```

Step 7. Select the first image of the 'imagecollection' (the Landsat images). This is the image of 1990. Then filter the river centerlines that fit in this first image.

```
var bounds = imagecollection.first().geometry(); // First image of Landsatimages (1990)
var centerlineFiltered = grwl.filterBounds(bounds) // Only use the river centerlines that fit in the 'bounds'
```

Step 8. Draw a polygon around the part of a Myanmarese river you want to find the erosion and sedimentation bands around.

Do this by clicking the '  ' (draw a polygon) shape. It will then appear in your 'imports' screen at the top of the code editor. Rename this polygon from 'geometry' to 'polygon'

```
Imports (1 entry)
  var polygon: Polygon, 5 vertices
```

Step 7. Use the function 'Determinewatermask' created in Step 3 to define the watermask. Use the 'NDWI', you created in the previous paragraph, polygon and the filtered centerline for this.
MaxDist = 5000
Fill_Size = 333

```
var WaterMasks = imagecollection.map(Determinewatermask('NDWI',polygon, centerlineFiltered, [redacted], [redacted]))
```

Question

4. What are the parameters you need to define when using the function 'Determinewatermask'?

……..

Step 8. Add two-layers to the map. Of the 4 Landsat images use the first two. Of the bands, use the Blue, Green and Red band. With this information fill the blue blocks below.

```
var visParams = {bands: ['[redacted]', '[redacted]', '[redacted]'], max: 0.3};
Map.addLayer(ee.Image((imagecollection.toList(imagecollection.size())).get([redacted])), visParams, 't0')
Map.addLayer(ee.Image((imagecollection.toList(imagecollection.size())).get([redacted])), visParams, 't1')
```

Question

3. Of which year are imagecollection 't0' and imagecollection 't1'?

……..

Step 9. The variable WaterMasks is an imagecollection at the moment (see step 7). An imagecollection is a set of different images. To be able to select each of these images separately we will make a 'List' out of them using '.toList'.

```
var WaterMasks = WaterMasks.toList(WaterMasks.size())
```

Question

4. Of how many images does the ImageCollection WaterMasks exist? Tip: to check use 'print(WaterMasks)'.

……

Step 10. Now we want to calculate the difference between the first image (0, 1990) and the second image (1, 1995). We will subtract image 0 from image 1. With the option 'slice' we can return part of a List, in this case WaterMasks.
If you want to know more on Arrays and Array Images check:
https://developers.google.com/earth-engine/arrays_array_images
If you want to know more about the 'zip' used below check the 'Docs' on the left side of your screen:

zip(other)

Pairs the elements of two lists to create a list of two-element lists. When the input lists are of different sizes, the final list has the same size as the shortest one.

Arguments:

- this:list (List)
- other (List)

Returns: List

And play around with parts of the script to check what happens when you change certain parts of the script.

```
// Function calculates the difference between t+1 and t
var DifferenceMasks = WaterMasks.slice(0,-1).zip(WaterMasks.slice(1)).map(function(f){
    return ee.Image(ee.List(f).get(1)).subtract(ee.Image(ee.List(f).get(0)))
  });

Map.addLayer(ee.ImageCollection(DifferenceMasks).first())
print(DifferenceMasks)
```

Step 11: Define the eroded, deposited and masked images. Tips: Use the DifferenceMasks list for the eroded and deposited images. Use the WaterMasks list to Mask the waterbodies outside the range around the centerline.

```
var ID1 = 1 // Select difference image 2000-1995 and 2005-2000
var ID2 = 2

var eroded = ee.Image(ee.List(DifferenceMasks).get(ID1)).eq(1)
.updateMask(ee.Image(ee.List(DifferenceMasks).get(ID1)).eq(1))
var deposited = ee.Image(ee.List(DifferenceMasks).get(ID1)).eq(-1)
.updateMask( ee.Image(ee.List(DifferenceMasks).get(ID1)).eq(-1))
var Mask = ee.Image(ee.List(WaterMasks).get(ID2))
// In eroded and deposited, 0 is land, 1 is water - that is why if you subtract them from eachother the outcome is -1 of 1
//if the grid cell was first water and than land and vice versa.
```

Step 12. In this step the eroded, deposited and masked area will be combined to create one MapLayer. To do so 'visualize' the variable 'Mask' as ['black', 'blue'], the eroded layer as ['red'] and the deposited layer as ['yellow'].

```
var mosaic = ee.ImageCollection([Mask.visualize({palette: ['black', 'blue']}),
eroded.visualize({palette: ['red']}), deposited.visualize({palette: ['yellow']})]).mosaic()

Map.addLayer(mosaic, {}, 'mosaic')
```

Save your script

Step 13. Change the script in such way that you see the erosion and sedimentation between 1990 and 2010 (add the image of 2010).

Save the script under a different name.

In this paragraph we have created a script to define the sedimentation and erosion all around the globe. You would just have to change the location of the polygon to do the same exercise for another part of the world.

Step 14. Change the polygon to another part of the world/Myanmar you are interested in and run the script.

This exercise is highly dependent on the satellite images you chose and the way you define water and land. The threshold you have chosen in step 4 for example is essential. Furthermore, the timing of the satellite images is essential. We have chosen all images in January, a dry month. However, we have seen in data that water levels could still be up to 2 meters different in January.

Question

5. Why is it essential that water levels/discharges in the rivers are not too different over the different images we have used in this analysis?

……

Question

6. What would influence the sensitivity of the image to higher water levels?

……

Save your script and change the location of your polygon to the lower Ayeyarwady (region Nyaungdon/Yangon).

Step 15. Import the 'dike_sections' layer in your assets by clicking 'Import' after opening this link: https://code.earthengine.google.com/?asset=users/carolien/dike_sections

```
Imports (3 entries)
  var polygon: Polygon, 5 vertices
  var Landsat: ImageCollection "USGS Landsat 5 TM Collection 1 Tier 1 TOA Reflectance" (8 bands)
  var table: Table users/carolien/dike_sections
```

Step 16. Load the 'dike_sections' in your script by clicking on the  -button to the right of your import 'dike_sections' and copy this link into your script. Call the variable 'dikes'

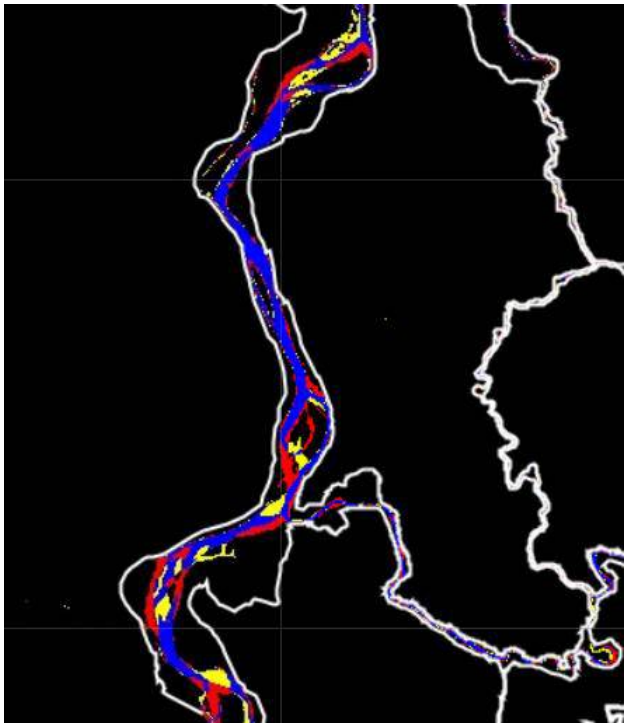
Step 17. Add the dikes layer to your map in white. Figure out what to put in the 'blue boxes' below.

```
//Add Dikes
var dikes = ee.FeatureCollection(        , 'dikes')
Map.addLayer(dikes, {color: '        '}, 'dikes');
```

Question

7. Does the image of sedimentation/erosion combined with the dikes look as you expected?

……



Step 18. Add a legend to the map in which you define the meaning of the colours in the combined sedimentation/erosion and dikes map.

Don't forget to save your script! The full code corresponding with this exercise can be found here: https://code.earthengine.google.com/6034541c8d77e55d0c121a63f4ab0033

4.4 Direction of morphological change

In the previous paragraph we have looked at sedimentation and erosion patterns over years. However, the evolution of the morphology over time can't be seen from this. Luckily, Joanne Craven has done a study for IHE Delft in which she shows the yearly morphological change for the Brahmaputra river. But, because she has used Google Earth Engine it is applicable around the world, and also for Myanmar.

First read the information about this website on:
http://joannecraven.co.uk/projects/brahmaputra-morphology/

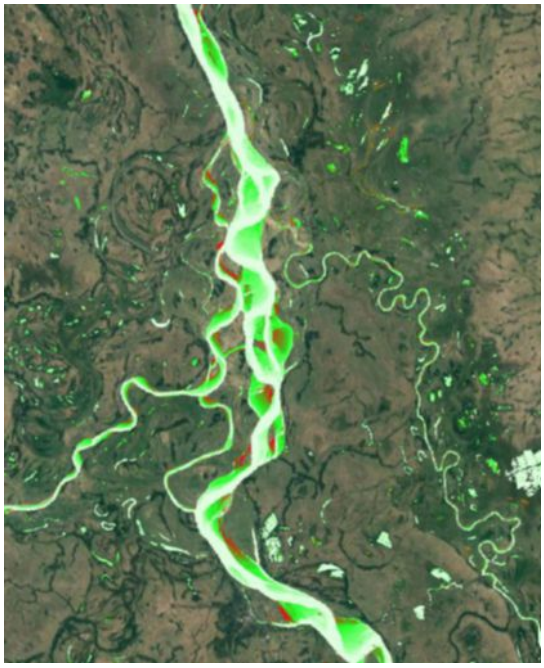
Then have a look at the following website and scroll to Myanmar:
http://brahmaputra-morphology.appspot.com/

Question
1. Which surfacewater dataset is used?
……

Step 1. Look for a part of a river in Myanmar which has had multiple shifts since 2000.

Question
2. Do you think that the river you choose has had a different path before 2000? And how can you see that in the satellite images?
……

Step 2. Now find a part of a river that has only moved in one direction since 2000. Do you expect it to continue to move in that direction in the future, and why?
……



4.5 HAND **bonus**

HAND is the Height above nearest drainage. Do some online search to create an HAND-image of Myanmar using Google Earth Engine scripting, this will require some searching on:

- What information is necessary for a HAND?
- How to script the HAND?

It is important to practice how to online search on these topics for Google Earth Engine because this is often the way to go to learn more on Google Earth Engine and work on the topics you are interested in.

Step 1. Create your own HAND-map of Myanmar

Question

1. What is a HAND-image mostly used for?

……..

5 GEE applications for Landcover Classification

For different purposes, it is important to have a reliable and up to date land cover map. This is needed for running hydrological models, but also for performing “stand-alone” analyses of land cover changes. Several land cover products are available on the national and global scales, but as you will discover these are often outdated or not detailed enough. It is possible to build your own land cover map for your area of interest based on satellite information, but until recently this involved many complex and time-consuming steps including the download of large amounts of data, and pre-processing or reprojection of the images using dedicated image analysis software. Nowadays, the GEE offers powerful and easy-to-use solutions to produce your own land cover map using several classification algorithms.

5.1 GlobCover – Global Land Cover Map

The global GlobCover land cover product is a scientific and technical demonstration of the first automated mapping of land cover on a global scale. It was derived from data acquired by the MEdium Resolution Imaging Spectrometer (MERIS) instrument on board the (now inactive) Envisat satellite of the European Space Agency (ESA). MERIS is a passive optical push broom wide-field instrument that measured reflected radiation the Earth's surface, the ocean, and from clouds at 300 m spatial resolution and in 15 spectral bands ranging between 390 - 1040 nm wavelengths in the visible and near-infrared spectrum. The global land cover map is derived by an automatic and regionally-tuned classification of a time series of global MERIS mosaics for the year 2009 (see Figure 13). The global land cover map counts 22 land cover classes defined with the United Nations (UN) Land Cover Classification System (LCCS).

The data can be downloaded via http://due.esrin.esa.int/page_globcover.php. For doing analyses for specific areas of interest, the data can be accessed and explored through the GEE Code Editor. The goal of this exercise is to use GEE to display the GlobCover land cover map for Myanmar, to add a legend to the map, and evaluate its quality.

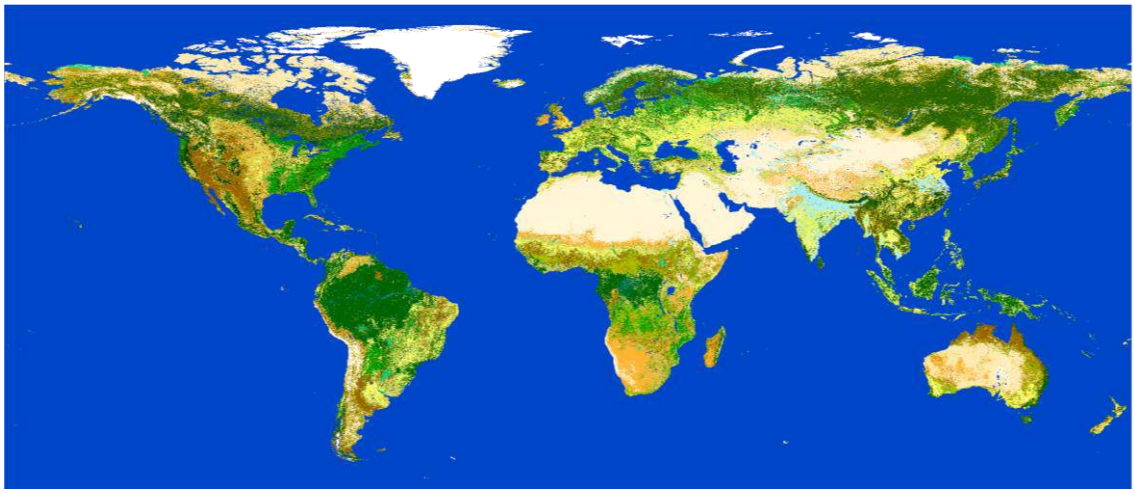


Figure 13. GlobCover 2009 global land cover map

Step 1. First, we need to combine the global dataset with the country borders of Myanmar. Import the GlobCover 2009 Global Land Cover Map and the FeatureCollection LSIB (Large Scale International Boundary Polygons) which contains polygons of all the countries in the world. To import these datasets, you can search for the datasets in the search bar or type the script below into the code editor.

```
var globcover = ee.Image("ESA/GLOBCOVER_L4_200901_200912_V2_3"),
    borders = ee.FeatureCollection("USDOS/LSIB_SIMPLE/2017");
```

Questions

1. Myanmar needs to be selected from the LSIB FeatureCollection. For that we need the two-letter FIPS 10-4 country code for Myanmar. What is this code?

.....

Step 2. Filter the LSIB FeatureCollection to select Myanmar using the country code and the function *ee.Filter.eq()*. Replace the '' with country code.

```
// Filter Myanmar
var Myanmar = borders.filter(ee.Filter.eq("country_co",""));
```

Step 3. The GlobCover 2009 map contains two bands, a binary quality band and the actual land cover map. We are only interested in the latter and will define it as a separate variable. Now we are ready to display the land cover map for Myanmar:

```
// select the land cover band
var cover = globcover.select("landcover");
print(cover);

// display the land cover map for Myanmar
Map.addLayer(cover.clip(Myanmar));
```

Question

2. The global land cover map counts 22 land cover classes, yet when the variable cover is printed to the console it becomes clear that 23 classes are listed. What could be the extra class included in the land cover map?

.....

.....

Step 4. To understand what we are looking at it is helpful to add a legend to the map. To do that we first need to define three lists that contain the land cover classes (names and codes) and the colors assigned to each class:

```
// extract land cover names, codes and palette
var landcoverNames = ee.List(cover.get('landcover_class_names'));
var landcoverCode = ee.List(cover.get('landcover_class_values'));
var landcoverPalette = ee.List(cover.get('landcover_class_palette'));
```

Step 5. Now we can start building the legend. The first step is defining a panel that will hold the legend and give it a position on the map and a title:

```
// Set position of legend panel
var legend = ui.Panel({
  style: {
    position: 'bottom-left',
    padding: '8px 15px'
  }
});

// Create legend title
var legendTitle = ui.Label({
  value: 'Land Cover',
  style: {
    fontWeight: 'bold',
    fontSize: '18px',
    margin: '0 0 4px 0',
    padding: '0'
    }
});

// Add the title to the legend panel
legend.add(legendTitle);
```

Step 6. Then the next step is to declare a function that will create and style 1 row of the legend. The function arguments are color and name, and when the function is called it will create a colored box and a description of the land cover class associated with that color:

```
// Declare function to create and style 1 row of the legend.
var makeRow = function(color, name) {

      // Create the label that is actually the colored box.
        var colorBox = ui.Label({
        style: {
          backgroundColor: '#' + color,
          // Use padding to give the box height and width.
          padding: '8px',
          margin: '0 0 4px 0'
        }
      });

      // Create the label filled with the description text.
      var description = ui.Label({
        value: name,
        style: {margin: '0 0 4px 6px'}
      });

      // return the panel
      return ui.Panel({
        widgets: [colorBox, description],
        layout: ui.Panel.Layout.Flow('horizontal')
      });
};
```

Step 7. To add land cover categories to the legend we can now call the makeRow() function in a for-loop that runs over the three previously declared lists containing the land cover names, codes and colors. An important aspect included in the script is the **getInfo()** function, which is needed to 'pull' the contents of the three lists from the server to the client-side (which is your PC and web-browser). Generally speaking, it advised to avoid client-side for-loops, but in this case it is the only way to add the legend the map. For more detailed explanation about server vs. client functionalities of GEE see https://developers.google.com/earth-engine/client_server.

```
/* Loop over the list containing land cover classes and colors
 and call the user-defined makeRow() function */
for (var i = 0; i < 23; i++) {
   var myCode = ee.Number(landcoverCode.get(i).getInfo());
   var myName = ee.String(landcoverNames.get(i).getInfo());
   var myNameCode = myName.cat(" [").cat(myCode.toInt()).cat("]");
   legend.add(makeRow(landcoverPalette.get(i).getInfo(),myNameCode.getInfo()));
}
```

Step 8. Finally, to add the legend to the map call the following function:

```
// add legend to the map
Map.add(legend);
```

If all steps in the script ran successfully you should get a similar map as shown in Figure 14. Take some time to scroll through your map and zoom in on some areas you know well. Also use the inspector tool to click on several locations to retrieve the land cover codes.

Questions

3. What, in your opinion, is the quality of the GlobCover map when you compare the land cover classifications to the areas that you inspected? Which land cover classes perform well and which don't?

……

……

……

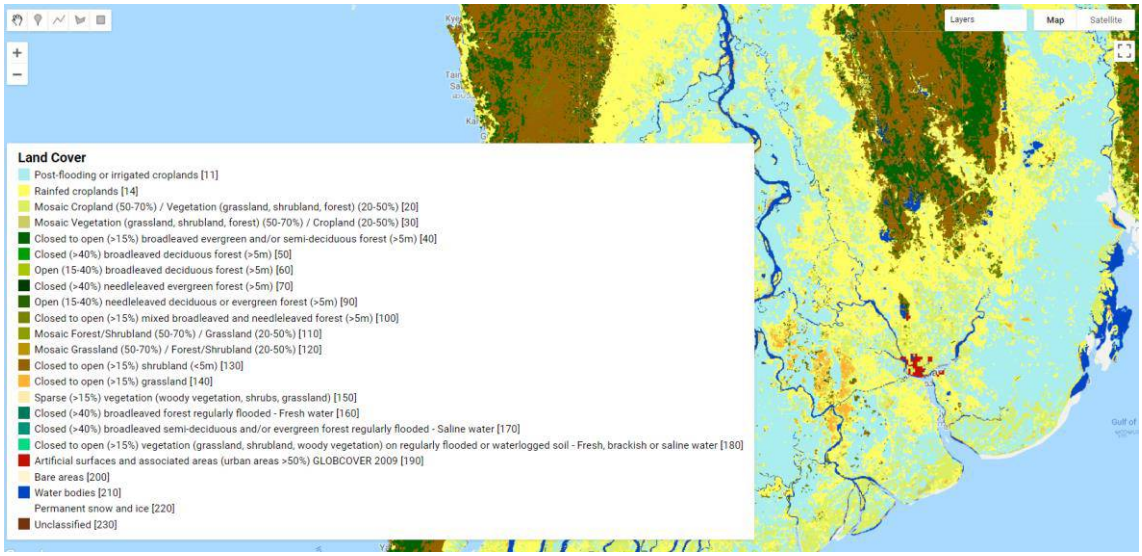


Figure 14. GlobCover 2009 map and legend for Myanmar

The full code corresponding with this exercise can be found here:
https://code.earthengine.google.com/05b400241bce3f94815e115729e55efb

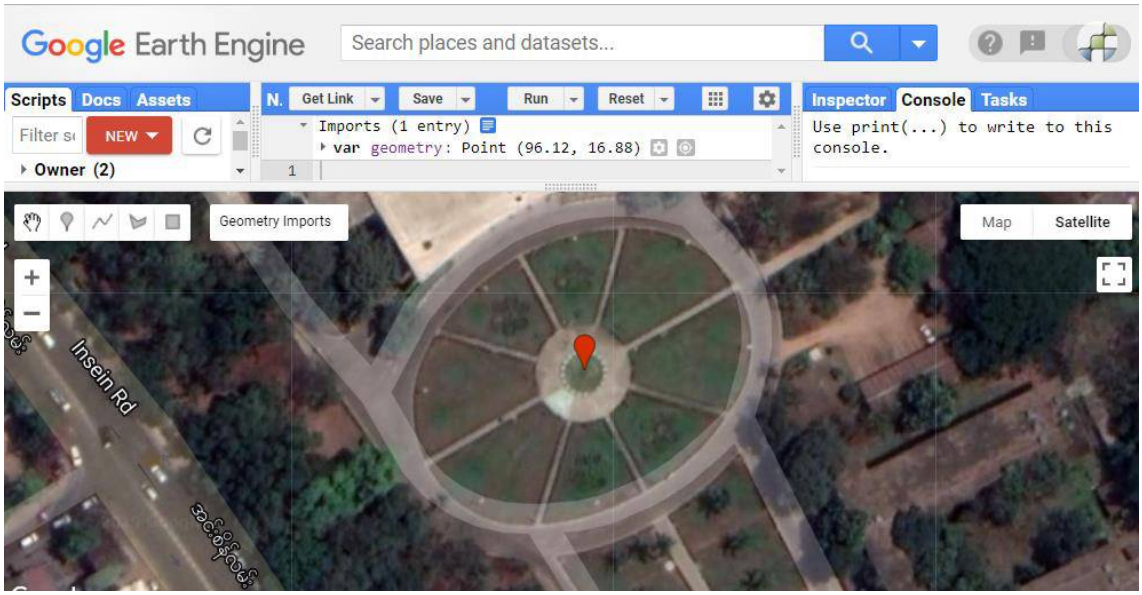
5.2 Supervised land cover classification

In this exercise we will use supervised classification applications in GEE to build a simple land cover map for a region of interest in Myanmar. The ***Classifier*** package handles supervised classification by traditional machine learning algorithms running in the Earth Engine. These classifiers include CART, Random Forest, NaiveBayes and SVM. The general workflow for classification is:

1. Collect training data. Assemble features which have a property that stores the known class label and properties storing numeric values for the predictors
2. Instantiate a classifier algorithm and set its parameters if necessary
3. Train the classifier using the training data.
4. Classify an image (or feature collection)
5. Estimate classification error with independent validation data

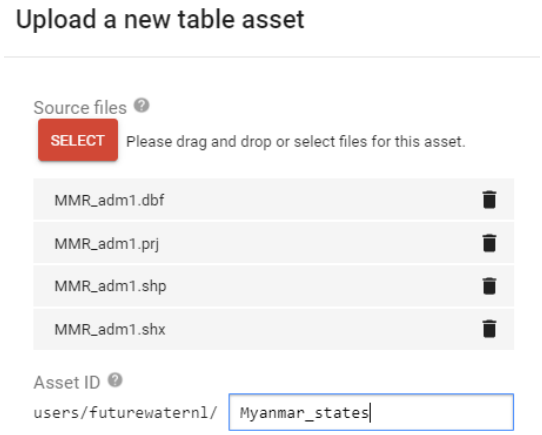
Training and/or validation data can come from a variety of sources. In this exercise we will collect training data interactively in GEE using the geometry drawing tools and a series of monthly Normalized Difference Vegetation Index (NDVI) images calculated from recent Landsat-8 satellite images

Step 1. Create a new script "landcover_NDVI". The first step is to define a region of interest. Select the point drawing tool (teardrop icon) from the geometry tools, go to the campus of Yangon Technical University and draw a single point as the region of interest. Then 'Exit' from the drawing tools. Note that a new variable is created in the imports, containing the single point, imported as a Geometry. Rename this import from ***geometry*** to ***roi***.



Step 2. To define the boundaries for the image classification exercise, the next step is to import a shapefile (which is a FeatureCollection) containing the states of Myanmar.

I. Go to https://www.diva-gis.org/gdata and download the Administrative Areas for Myanmar (*MMR_adm.zip*) to your PC
II. Unzip the folder and upload the following 4 files listed below into your Google Earth Engine user environment via Assets → NEW → Table upload → Shape files



III. Wait for the asset to be ingested (this may take a few minutes), then import the table into the editor window and display the map by copying the code below in the editor. Rename the Imports variable *table* into *states* and also give the displayed shapefile layer the name *states*:

```
// Zoom to region of interest
Map.setCenter(96.12, 16.88, 8); // Coordinates YTU

// Display states
Map.addLayer(states,{},"states");
print(states);
```

Note: In case the downloading and uploading takes too much time, you can also link to the shapefile uploaded through the FutureWater GEE account:

```
// import shapefile via FutureWater GEE account
var states = ee.FeatureCollection("users/futurewaternl/Myanmar_states");
```

IV. Now run the script: you will see the map zoom to the area of interest and the state boundaries being added.

We will perform the image classification on Yangon State. To select this state, we must apply a filter function to the FeatureCollection containing all states. Display its properties in the Console by selecting the Inspector tool and clicking on the Yangon State in the shapefile layer.

Question

1. Type the code below in the editor and replace the "■" with the correct character strings. What text should be typed instead of "■"

   ……

```
// Filter to Yangon state
var yangonState = states.filter(ee.Filter.eq("■","Yangon"));
Map.addLayer(yangonState, {},"Yangon state");
```

Step 3. Now that we have defined Yangon State as our exercise area it is time to load the Landsat-8 ImageCollection. Search for Landsat-8 in the search bar and import the following

ImageCollection: USGS Landsat 8 Tier 1 TOA Reflectance. After importing, rename the Imports variable ImageCollection to landsat8.

Step 4. Now, we will define the time period and the images for which we want to perform the analysis. For now, we will focus on the period 2008 – 2018.

```
// Define time range
var startyear = 2008;
var endyear = 2018;

// Set date in ee date format
var startdate = ee.Date.fromYMD(startyear,1,1);
var enddate = ee.Date.fromYMD(endyear,12,31);

// Filter image collection
var l8Collection = landsat8.filterDate(startdate,enddate).filterBounds(roi);

print(l8Collection);
```

Question

2. The final line of code above prints the list of images in the collection to the console. Have a look at this list. How many images are in the list? What do you notice with regards to their dates, compared to the defined time range? Can you explain this?

   ……..

   ……..

Step 5. The variable *l8Collection* now consists of all available images for Yangon State in the defined period. We can visualize a true-color composite of all these images by typing the following code and running the script:

```
// Build median true-color composite using ee.Reducer
var medianImage = l8Collection.reduce(ee.Reducer.median());
print(medianImage);

Map.addLayer(medianImage.clip(yangonState),
{bands: ['B4_median', 'B3_median', 'B2_median'], min: 0, max: 0.3},
'median true-color composite');
```

Question

3. Why do we need to select bands B4, B3 and B2? Try to change the other of the bands in the above code and re-run the script; what happens, and can you explain this?

   ……..

   ……..

Note: For basic statistics (e.g. min, max, mean, median) there are shortcut methods. They function in exactly the same way as calling reduce(), except the resultant band names will not have the name of the reducer appended (see also: https://developers.google.com/earth-engine/reducers_image_collection.html). The code snippet above can thus be shortened into:

```
// Shortcut method median() true-color composite
var medianImage = l8Collection.median();
print(medianImage);

Map.addLayer(medianImage.clip(yangonState),
{bands: ['B4', 'B3', 'B2'], min: 0, max: 0.3},
'median true-color composite');
```

A potential problem for building a land cover map using Landsat 8 satellite images, is the presence of clouds. GEE has standard functions available to filter out clouds in a Landsat image. We will now define and apply a cloud masking function, based on the *simpleCloudScore* algorithm which calculates cloud likelihood per pixel based on reflectance values.

Step 6. Type the below code to use the CloudScore algorithm with a threshold value of 40%. This function removes pixels with a cloud likelihood of over 40% from each of the individual image, before calculating the median value from all images. Re-run the script and note the difference in the resulting image composite.

```
// set cloud threshold
var cloud_thresh = 40;

// // Cloud mask function
var cloudfunction = function(image){
  //use add the cloud likelihood band to the image
  var CloudScore = ee.Algorithms.Landsat.simpleCloudScore(image);
  //isolate the cloud likelihood band
  var quality = CloudScore.select('cloud');
  //get pixels above the threshold
  var cloud01 = quality.gt(cloud_thresh);
  //create a mask from high likelihood pixels
  var cloudmask = image.mask().and(cloud01.not());
  //mask those pixels from the image
  return image.updateMask(cloudmask);
};

// mask all clouds in the image collection
var l8NocloudCollection = l8Collection.map(cloudfunction);
print(l8NocloudCollection);
```

We have now successfully created a cloud-free ImageCollection on which we can base the NDVI calculations. Computing the NDVI requires usage of the near-infrared (NIR) and red (R) bands of the images included in the *l8NocloudCollection* image collection.

Step 7. We declare and call a function to perform a normalized difference calculation and add a NDVI band to every image in the *l8NocloudCollection:*

```
// Add NDVI band to cloudfree landsat 8 imageCollection
function l8addNDVI(image) {
  var ndvi = image.normalizedDifference(['B5', 'B4']).rename('NDVI');
  return image.addBands(ndvi);
}
var l8NocloudCollection = l8NocloudCollection.map(l8addNDVI);
print(l8NocloudCollection);
```

Step 8. We can now reduce the *l8NocloudCollection* to a single image for example with the *mean()* function (i.e. shortcut reducer method), which calculates the mean NDVI value for each

pixel over all available images in the collection. The following code performs this calculation and displays the result:

```
// Define NDVI visualization parameters
var ndviVis = {
  palette : ['darkblue', 'blue', 'red', 'orange', 'yellow', 'green', 'darkgreen'],
  min : -0.4,
  max : 0.9
};

// Add NDVI map to canvas
Map.addLayer(l8NocloudCollection.select("NDVI").mean().clip(yangonState),
ndviVis,"mean NDVI composite");
```

Question

4. Compare the average NDVI map with the Google Satellite basemap and investigate local values for different land cover types. What are NDVI typical values for forest areas, agricultural land, bare soils and water bodies?

   ……

   ……

As we have seen under Step 4, our *ImageCollection* consists of 160 images captured between 2013 – 2018. Sometimes there are gaps between images due to cloud cover, and sometimes they are with 16-day intervals. To do the land cover classification based on NDVI values, it is necessary to have one consistent dataset, with the same dates and time intervals for each pixel. We therefore need to convert *l8NocloudCollection* to an ImageCollection that contains 12 images, each one with the average 2013-2018 NDVI value for a specific month.

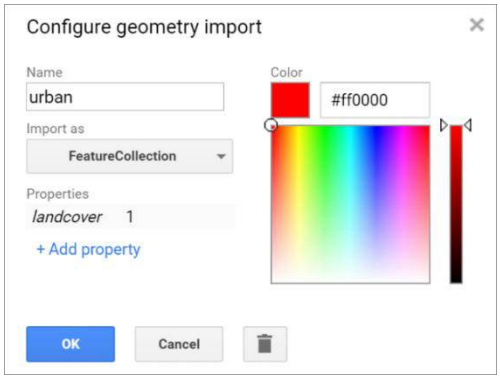
Step 9. Type the code below to create an image with 12 bands based on the mean NDVI values for each month. A *TrainingImage* is then created with band names corresponding with each month (“jan”, “feb”, etc.). Note that the *Date.fromYMD* function requires the definition of a specific year to work. In the example below the year 2000 is entered, although actual values will correspond with long-term averages between 2013 and 2018.

Note: To save yourself some time, the code snippet below can also be found here:
https://code.earthengine.google.com/8aee8565aa7cec4a6b504789f3ad5aff

```
// ---------------------------------------------------------------------
// Algorithm to create training image with 12 bands, one for each month
// Each band holds a monthly average 2013 - 2018 NDVI value
// ---------------------------------------------------------------------

// List to hold monthly NDVI
var months = ee.List.sequence(1,12);

// Function to calculate monthly NDVI from cloudfree imageCollection
function calculateNDVI(month){
  var monthNDVI = l8NocloudCollection.select("NDVI")
  .filter(ee.Filter.calendarRange(month, month, 'month')).mean();
  return monthNDVI.set('year', 2000).set('month', month)
  .set('date', ee.Date.fromYMD(2000,month,1))
  .set('system:time_start',ee.Date.fromYMD(2000,month,1));
}

// Map the function to create new imageCollection with 12 images
var monthlyNDVI =  ee.ImageCollection.fromImages(months.map(calculateNDVI).flatten());

// List the 12 images into new list
var myNDVIlist = monthlyNDVI.select("NDVI").toList(12);

// Training image with NDVI values of January as band 1
var trainingImage = ee.Image(myNDVIlist.get(0)).rename("jan");

// List of month names
var monthNames = ["jan","feb","mar","apr","may","jun","jul",
                  "aug","sep","oct","nov","dec"];

// Add bands (NDVI feb - dec) to image
for (var i = 1; i < myNDVIlist.length().getInfo(); i++) {
  var myMap = ee.Image(myNDVIlist.get(i)).rename(monthNames[i]);
  trainingImage = trainingImage.addBands(myMap);
}
```

Step 10. Now that we have our image with monthly NDVI data to which to work with, it is time to collect training data for the supervised classification. You can use the satellite view of the map panel and the cloud-free median true-colour composite image for reference.

1. Hover on the Geometry Imports box next to the geometry drawing tools and click + new layer. In the import entries in the code editor a new variable is created. Each new layer is going to represent one class within the training data.



2. Let the first new layer represent urban. Click on the ⚙ icon and configure the import to be a FeatureCollection and via + Add property button name the import landcover and set its value to 1 (Subsequent classes will be set to 2, 3, 4, etc.). Click OK to exit.

3. Now locate points in the cloud free image that are urban or built up areas (buildings, roads, parking lots, etc.). Collect around 50 sample points. When finished collecting points, click 'Exit' to close the point drawing for the active layer.

4. Make 3 additional layers and repeat the training data collection steps for:
   - ***forest*** with properties ***landcover*** set to class value **2**
   - ***river*** with properties ***landcover*** set to class value **3**
   - ***Lake*** with properties **landcover** set to class value **4**
   - ***agriculture*** with properties ***landcover*** set to class value **5**

5. After finishing, the Imports section of your script should look something like this:

```
Imports (7 entries)
  var landsat8: ImageCollection "USGS Landsat 8 Collection
  var roi: Point (96.12, 16.88)
  var states: Table users/futurewaternl/Myanmar_states
  var urban: FeatureCollection (64 elements)
  var water: FeatureCollection (37 elements)
  var forest: FeatureCollection (64 elements)
  var agriculture: FeatureCollection (66 elements)
```

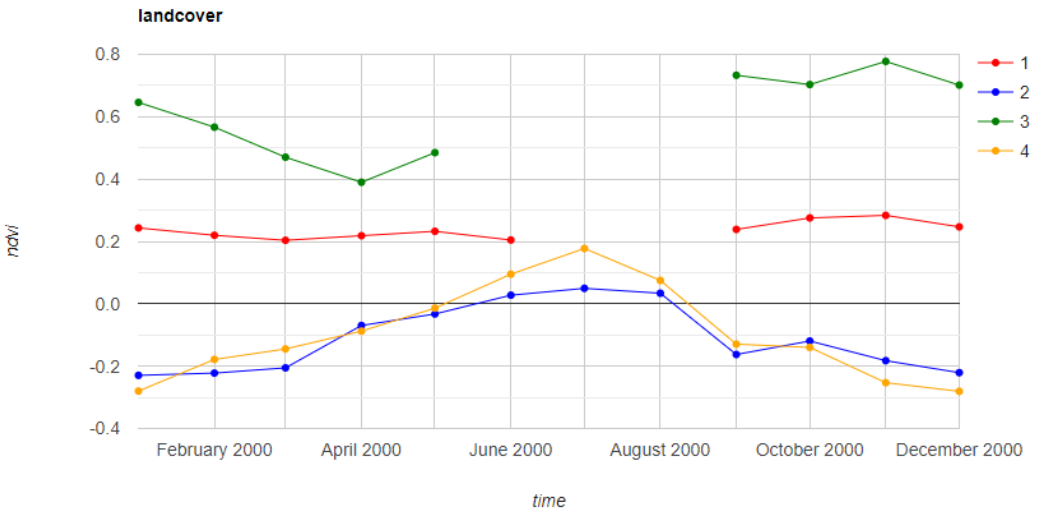
Step 11. The next step is to merge all collected training points into a single FeatureCollection called *trainingFeatures*, where the property landcover has a value (1, 2, 3, 4) for each class:

```
// Merge all training samples into new FeatureCollection
var trainingFeatures = urban // 1
.merge(water)                // 2
.merge(forest)               // 3
.merge(agriculture)          // 4

// Inspect the merged FeatureCollection
print(trainingFeatures, "training features");
```

Step 12. We can obtain insight in NDVI dynamics for each of these classes by creating a time-series graph that shows the average monthly NDVI for each class between 2013 - 2018

```
// Predefine the chart options
var chartOptions = {
  title: 'Landcover',
  hAxis: {title: 'Time'},
  vAxis: {title: 'NDVI'},
  lineWidth: 1,
  pointSize: 4,
  series: {
    0: {color: 'FF0000'}, // urban
    1: {color: '0000FF'}, // water
    2: {color: '008000'}, // forest
    3: {color: 'FFA500'}, // agriculture
}};

// create chart
var chart = ui.Chart.image.seriesByRegion(
monthlyNDVI,trainingFeatures,ee.Reducer.mean(),'NDVI',30,'system:time_start','landcover')
.setOptions(chartOptions);
print(chart);
```

You should get a graph that looks something like this (depending on the specific points you selected):



Question

5. Evaluate the monthly NDVI dynamics for each of the four land use classes in the graph. Can you explain the patterns you observe for each of the classes? And what could be the reason for the gaps in the lines for the classes urban and forest?

……

……

……

……

Step 13. The next step is to create the training data by overlaying the collected training points on the cloud free Landsat 8 image. The code below will extract NDVI values from the 12 bands and adds them as new properties to each training point in the merged feature collection. The FeatureCollection called *classifierTraining* now has the reflectance value from each selected band stored for every training point along with its class label. You can check this by printing it to the console and inspecting its properties.

```
// Create training data
var classifierTraining = trainingImage.select(monthNames).sampleRegions({
    collection: trainingFeatures,
    properties: ['landcover'],
    scale: 30
});
print("training",classifierTraining);
```

Question

6. The scale for the sampleRegions function is set to 30. Can you explain why this is a logical value for extracting reflectance values for the Landsat 8 image?

……

……

Step 14. The final step is to train the train the classifier using the training data and classify the image by executing the classifier. Run the code below and display the result, which will look like Figure 15. Change the colours if you like.

```
// Train the classifier
var trainClassifier = ee.Classifier.randomForest().train({
    features: classifierTraining,
    classProperty: 'landcover',
    inputProperties: monthNames
});

// Execute the classification
var landcoverClassified = trainingImage.select(monthNames).classify(trainClassifier);

// add the result to the Map
Map.addLayer(landcoverClassified.clip(yangonState), {min: 1, max: 4,
palette: ['FF0000','0000FF','008000','FFA500']},'land cover classification');
```

Question

7. What do you think about the result?

……

……

……

……

If you are not satisfied with the result, you can try the following options to improve the map:

- Experiment with other classifiers. Under Step 13 we selected the randomForest algorithm, but GEE offers multiple options (see ee.Classifier description under Docs)
- Add more classes in the selection of training features, to get a more detailed classification;
- Define additional (or different) locations for training the algorithm for specific classes that you are not satisfied with.

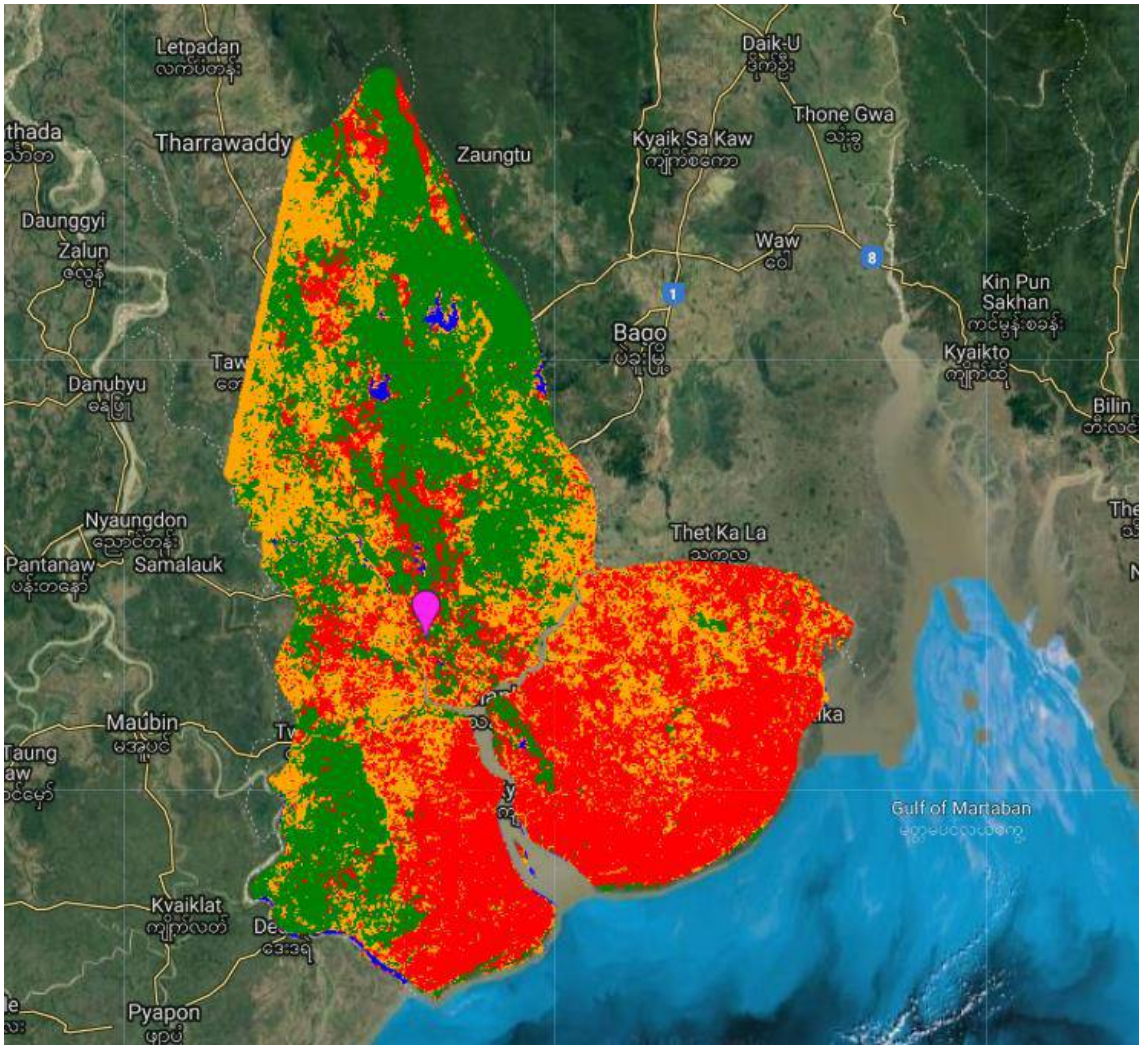
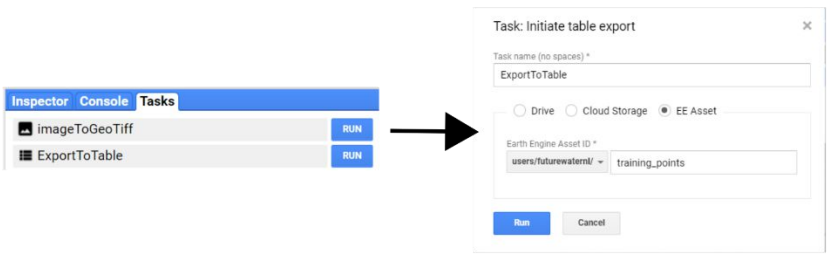


Figure 15. Example land cover map of Yangon state created with GEE

Step 15. To use the training points in another script, you can export this FeatureCollection as an Asset to your Google Earth Engine user environment:

```
// Export training points FeatureCollection as Asset.
Export.table.toAsset({
  collection: trainingFeatures,
  description:'ExportToTable',
  assetId: 'training_points',
});
```

Under *Tasks* the task ExportToTable appears which you can then run to start the export:



Likewise, you can also export the classified landcover map to your Google Drive as a GeoTiff:

```
// Export classified map as GeoTIFF
Export.image.toDrive({
  image: landcoverClassified.clip(yangonState),
  description: 'imageToGeoTiff',
  scale: 30,
  region: Yangon,
  fileFormat: 'GeoTIFF',
  formatOptions: {
    cloudOptimized: true
  }
});
```

The full code corresponding with this exercise can be found here:
https://code.earthengine.google.com/ab74726ad9740cd8a578562d9020989d

Extra: The SERVIR Mekong project (https://servir.adpc.net/about/about-servir-mekong) has set up an state-of-the art land cover monitoring system using GEE supervised classification algorithms. All their GEE applications can be found here: https://servir.adpc.net/tools, the land cover portal for Myanmar can be found here: https://rlcms-servir.adpc.net/en/myanmar-national-portal/

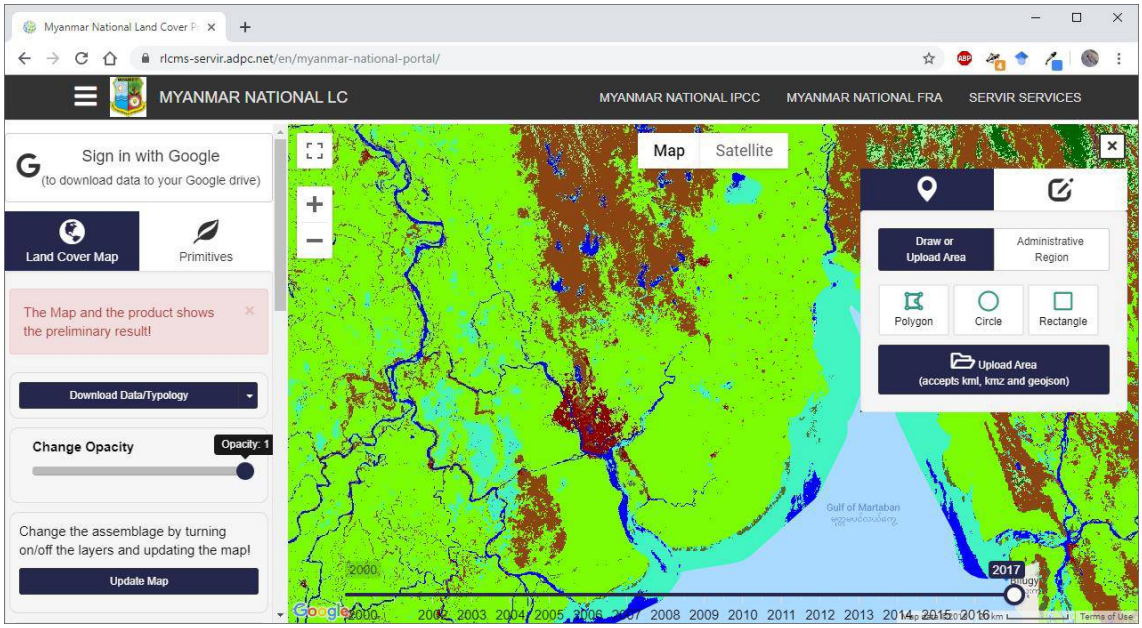


Figure 16. Myanmar Land Cover portal of SERVIR Mekong