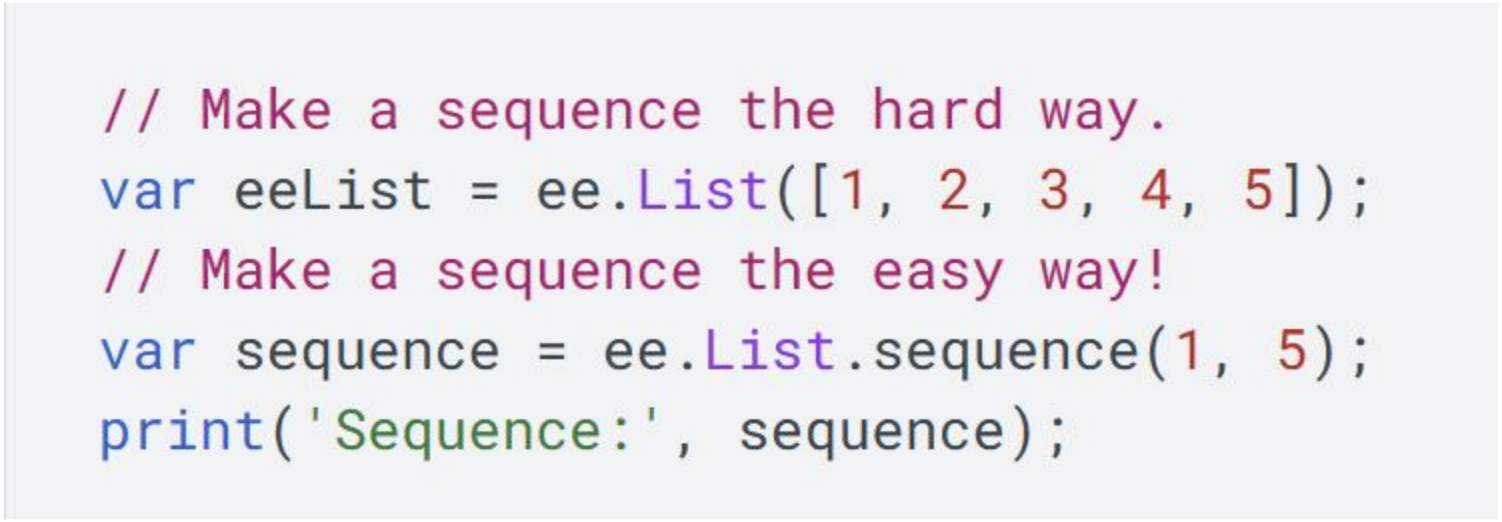
PS1 PRESENTATION

TITLE: CHARACTERIZING THE SPATIAL-TEMPORAL PATTERNS OF FLOODS IN THE ASSAM VALLEY: INSIGHTS FROM RAINFALL ANALYSIS

CHAPTER 1:
OUR LEARNINGS FROM MODULE 2

EARTH ENGINE OBJECTS

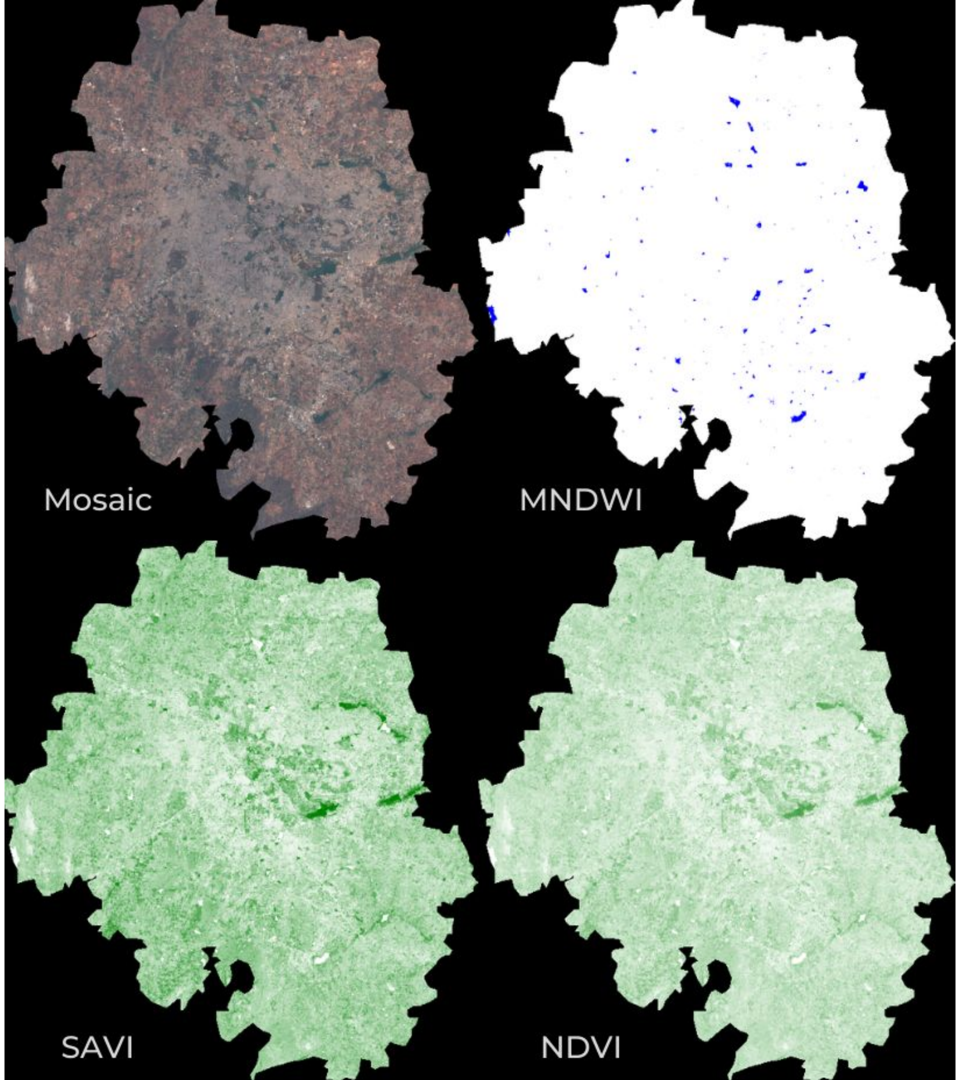
- In general, it is better to use Earth Engine API methods in your code so that your computations can use the Google Earth Engine servers.



```javascript
// Make a sequence the hard way.
var eeList = ee.List([1, 2, 3, 4, 5]);
// Make a sequence the easy way!
var sequence = ee.List.sequence(1, 5);
print('Sequence:', sequence);
```

CALCULATING INDICES

- Spectral Indices are central to many aspects of remote sensing. Whether you are studying vegetation or tracking fires - you will need to compute a pixel-wise ratio of 2 or more bands.
- The most commonly used formula for calculating an index is the Normalized Difference between 2 bands.
- Earth Engine provides a helper function normalizedDifference() to help calculate normalized indices
- For more complex formulae, you can also use the expression() function to describe the calculation.



MNDWI -> Water
SAVI -> Soil
NDVI -> Vegetation

COMPUTATIONS ON IMAGE COLLECTIONS

- So far we have learnt how to run computation on single images. If you want to apply some computation - such as calculating an index - to many images, you need to use **map()**.
- You first define a function that takes 1 image and returns the result of the computation on that image. Then you can map() that function over the ImageCollection which results in a new ImageCollection with the results of the computation.
- Similar to a *for-loop*

CLOUD MASKING

- Masking pixels in an image makes those pixels transparent and excludes them from analysis and visualization.
- The Code Editor contains pre-defined functions for masking clouds for popular datasets
- To mask an image, we can use the **updateMask()** function

REDUCERS

- *Reduce* operation allows you to compute statistics on a large amount of inputs.
- It makes use of the parallel computation resources that Earth Engine gives us.
- The Earth Engine API comes with a large number of built-in reducer functions (such as ee.Reducer.sum(), ee.Reducer.histogram(), ee.Reducer.linearFit() etc.) that can perform a variety of statistical operations on input data.
- Earth Engine supports running reducers on all data structures that can hold multiple values, such as Images (reducers run on different bands), ImageCollection, FeatureCollection, List, Dictionary etc.

TIME-SERIES CHARTS

- We can now put together all the skills learnt so far to make charts that show temporal changes in a region.
- Earth Engine API comes with support for charting functions based on the Google Chart API.

CHAPTER 2
OUR WORK SO FAR

AIM OF THE PROJECT

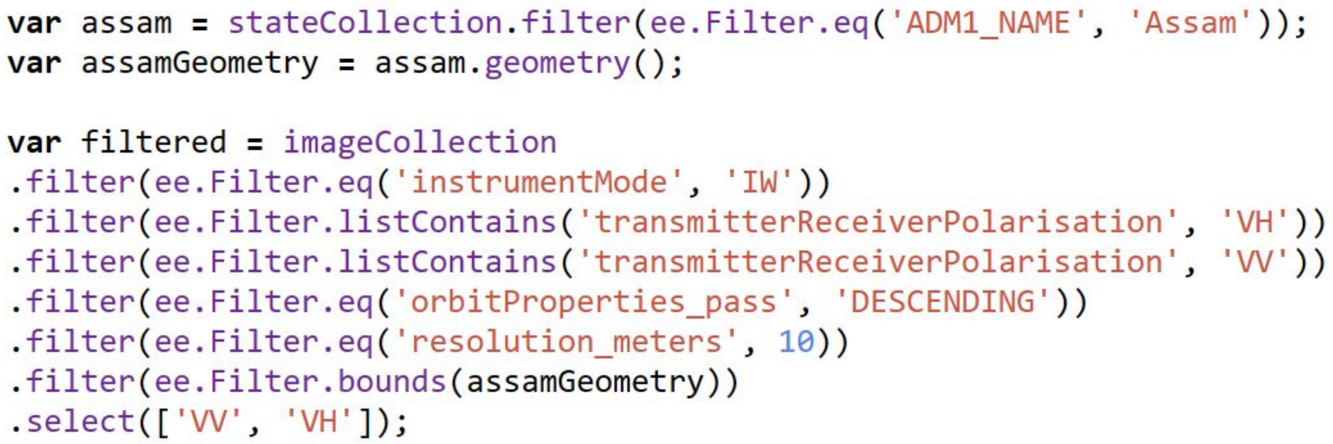
- Utilize Sentinel-1 Synthetic Aperture Radar (SAR) data to characterize the spatio-temporal patterns of floods in the Assam Valley
- Acquire and preprocess the data, employ thresholding and image segmentation algorithms to identify and delineate flooded areas and compare with rainfall data

DATA PREPROCESSING

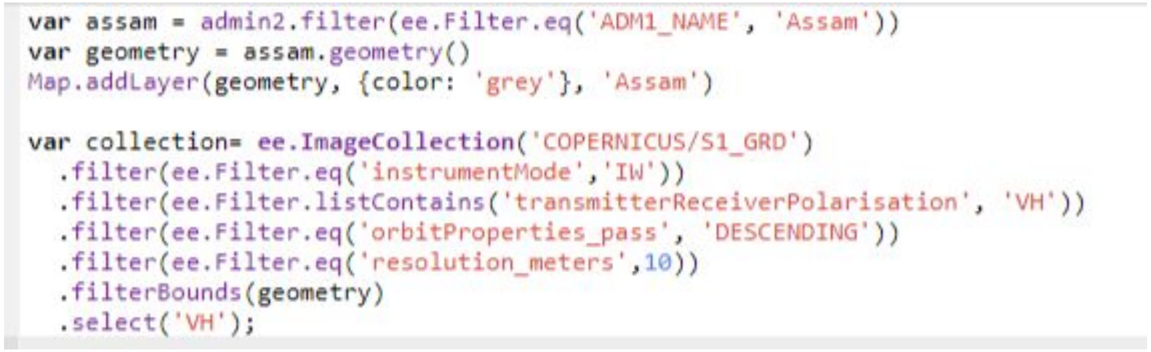
- There are 4 types of data - optical, microwave, thermal and hyperspectral
- Microwave data is best suited for our project
- To be more specific, we will be using the Sentinel 1 SAR GRD dataset
- Water is easy to detect using SAR data. We will generate an RGB composite from VV, VH, and VV/VH
- Sentinel 1 can receive specific polarizations simultaneously and send signals in horizontal or vertical polarization → HH, HV, VV, VH. It has three different operational modes, which are:
    - Interferometric Wide swath: majorly used
    - Extra wide swath: low resolution but wide
    - Strip map: high resolution but small regions

METHODOLOGY

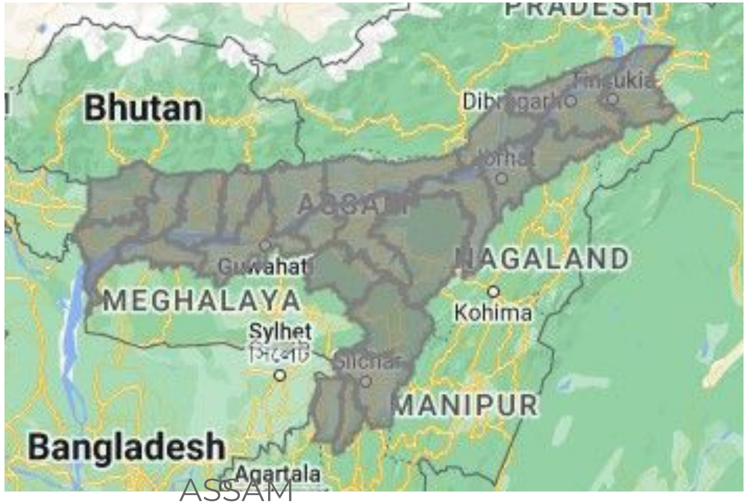
- Select images from the dataset using time filters
- Apply the other necessary filters
- Calculate difference and apply a threshold
- Calculate flooded area
- Monitor the impact of rainfall on water spread of Brahmaputra



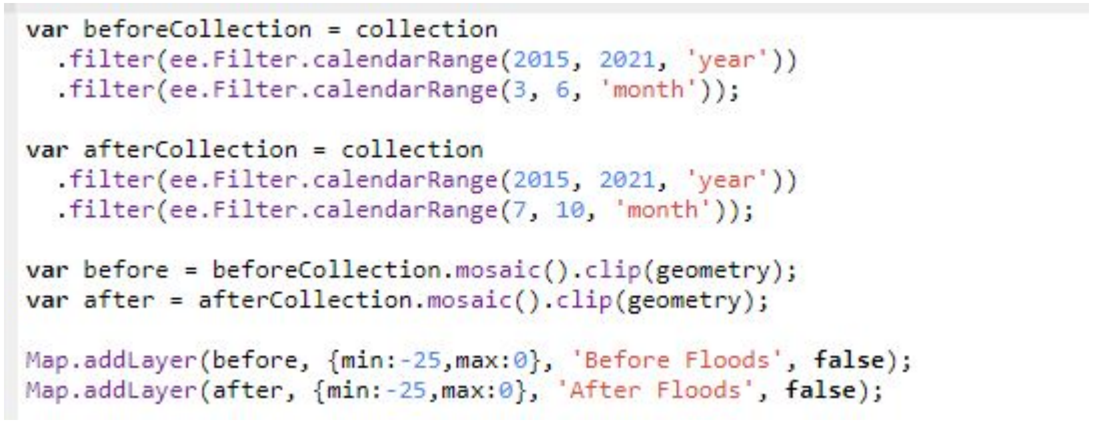
```
var assam = stateCollection.filter(ee.Filter.eq('ADM1_NAME', 'Assam'));
var assamGeometry = assam.geometry();

var filtered = imageCollection
.filter(ee.Filter.eq('instrumentMode', 'IW'))
.filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH'))
.filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VV'))
.filter(ee.Filter.eq('orbitProperties_pass', 'DESCENDING'))
.filter(ee.Filter.eq('resolution_meters', 10))
.filter(ee.Filter.bounds(assamGeometry))
.select(['VV', 'VH']);
```



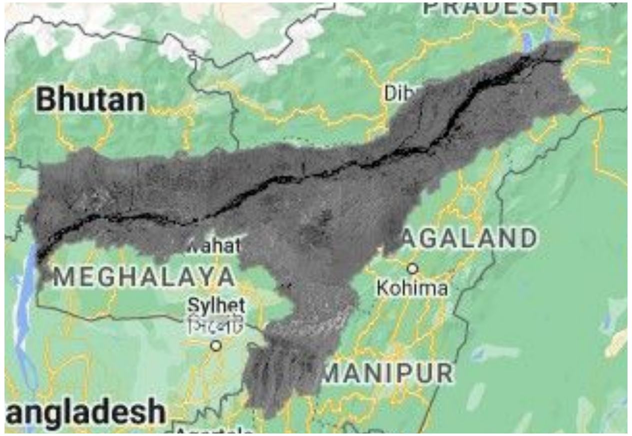
```
var assam = admin2.filter(ee.Filter.eq('ADM1_NAME', 'Assam'))
var geometry = assam.geometry()
Map.addLayer(geometry, {color: 'grey'}, 'Assam')

var collection= ee.ImageCollection('COPERNICUS/S1_GRD')
  .filter(ee.Filter.eq('instrumentMode','IW'))
  .filter(ee.Filter.listContains('transmitterReceiverPolarisation', 'VH'))
  .filter(ee.Filter.eq('orbitProperties_pass', 'DESCENDING'))
  .filter(ee.Filter.eq('resolution_meters',10))
  .filterBounds(geometry)
  .select('VH');
```



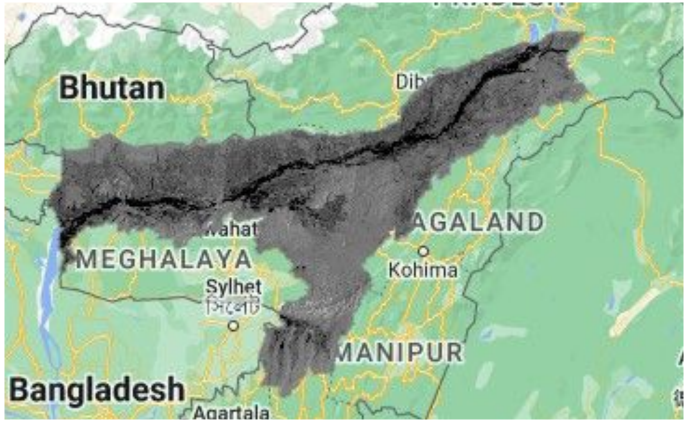
ASSAM



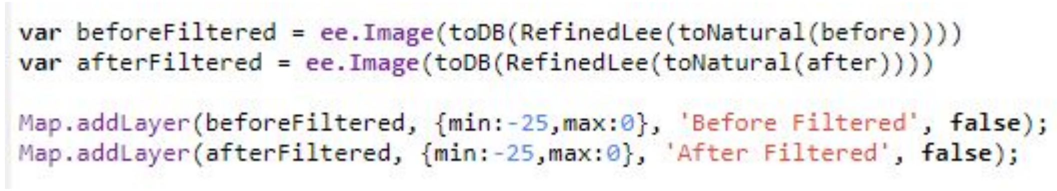
```
var beforeCollection = collection
  .filter(ee.Filter.calendarRange(2015, 2021, 'year'))
  .filter(ee.Filter.calendarRange(3, 6, 'month'));

var afterCollection = collection
  .filter(ee.Filter.calendarRange(2015, 2021, 'year'))
  .filter(ee.Filter.calendarRange(7, 10, 'month'));

var before = beforeCollection.mosaic().clip(geometry);
var after = afterCollection.mosaic().clip(geometry);

Map.addLayer(before, {min:-25,max:0}, 'Before Floods', false);
Map.addLayer(after, {min:-25,max:0}, 'After Floods', false);
```



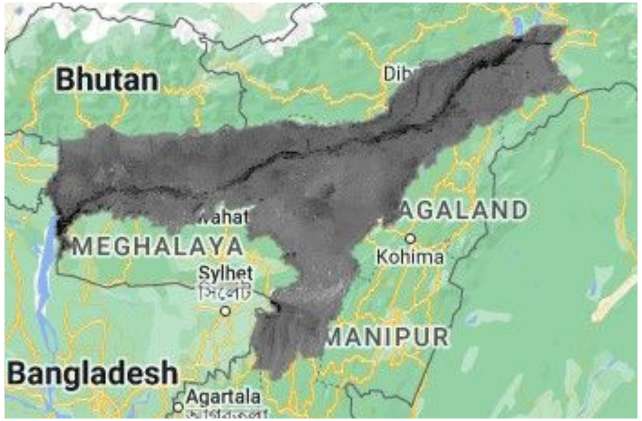
BEFORE FLOODS



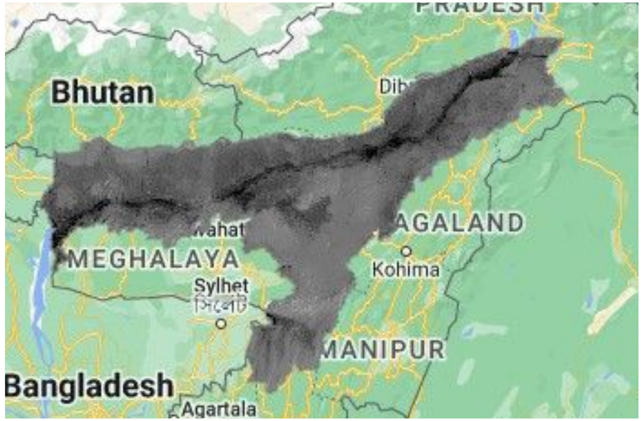
AFTER FLOODS



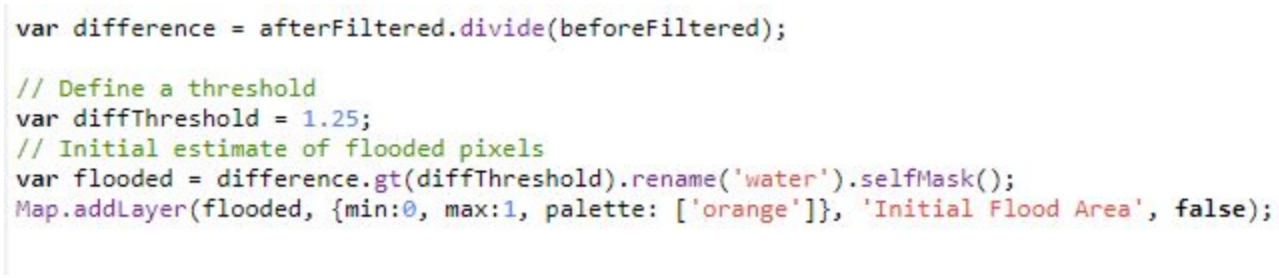
```
var beforeFiltered = ee.Image(toDB(RefinedLee(toNatural(before))))
var afterFiltered = ee.Image(toDB(RefinedLee(toNatural(after))))

Map.addLayer(beforeFiltered, {min:-25,max:0}, 'Before Filtered', false);
Map.addLayer(afterFiltered, {min:-25,max:0}, 'After Filtered', false);
```



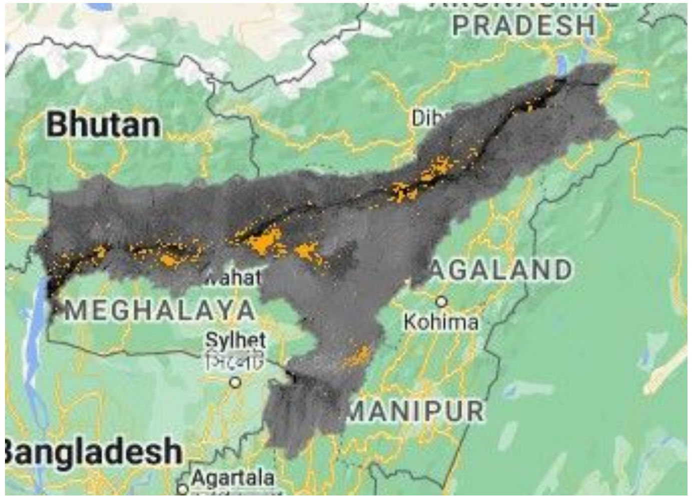
BEFORE FILTERING



AFTER FILTERING

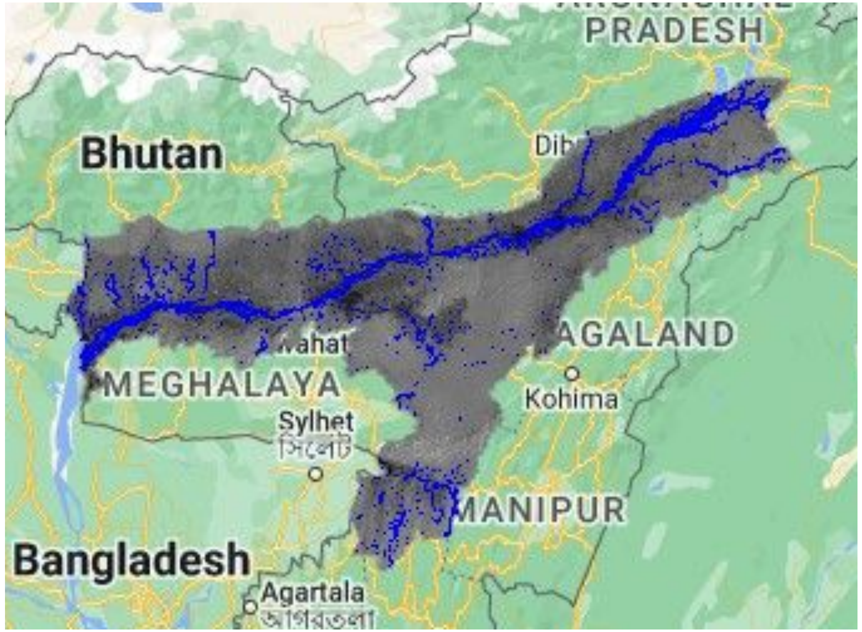


```
var difference = afterFiltered.divide(beforeFiltered);

// Define a threshold
var diffThreshold = 1.25;
// Initial estimate of flooded pixels
var flooded = difference.gt(diffThreshold).rename('water').selfMask();
Map.addLayer(flooded, {min:0, max:1, palette: ['orange']}, 'Initial Flood Area', false);
```

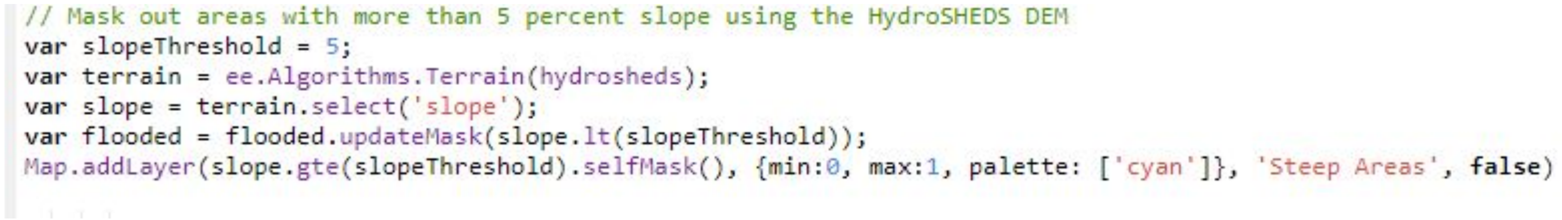


FLOODED AREA

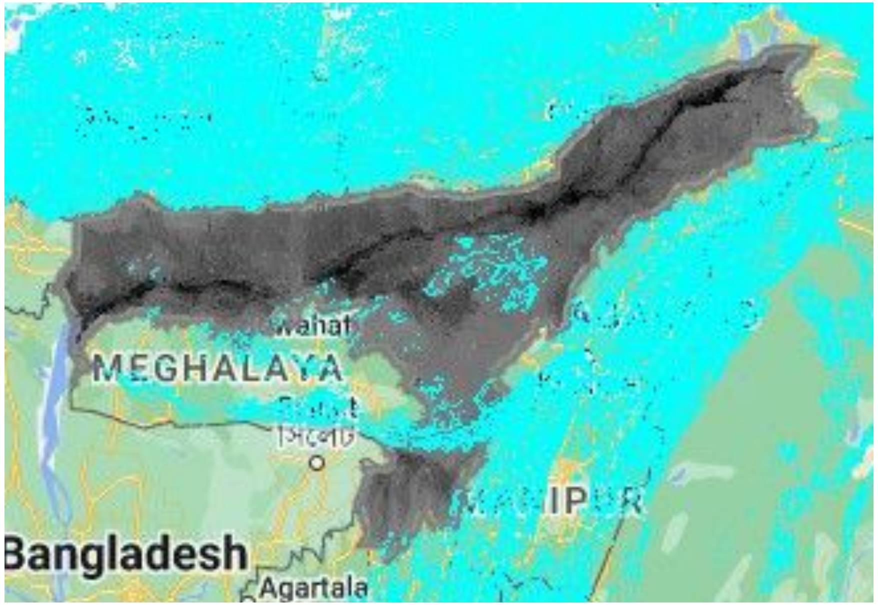
```
// Mask out area with permanent/semi-permanent water
var permanentWater = gsw.select('seasonality').gte(5).clip(geometry)
var flooded = flooded.where(permanentWater, 0).selfMask()
Map.addLayer(permanentWater.selfMask(), {min:0, max:1, palette: ['blue']}, 'Permanent Water')
```



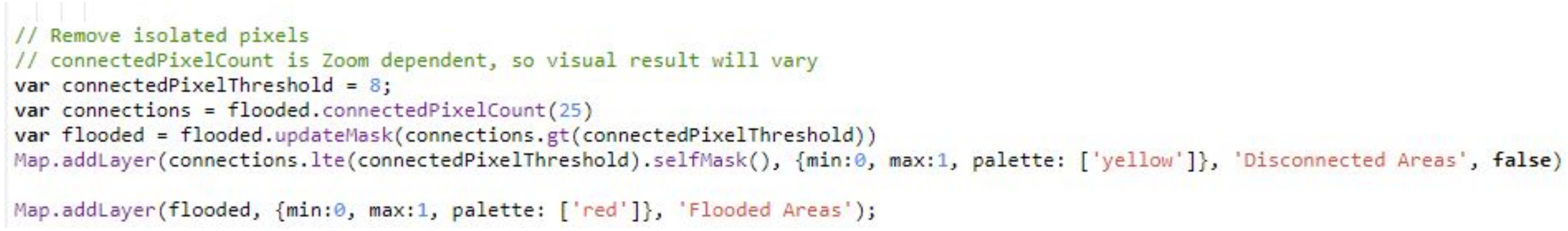
PERMANENT WATER



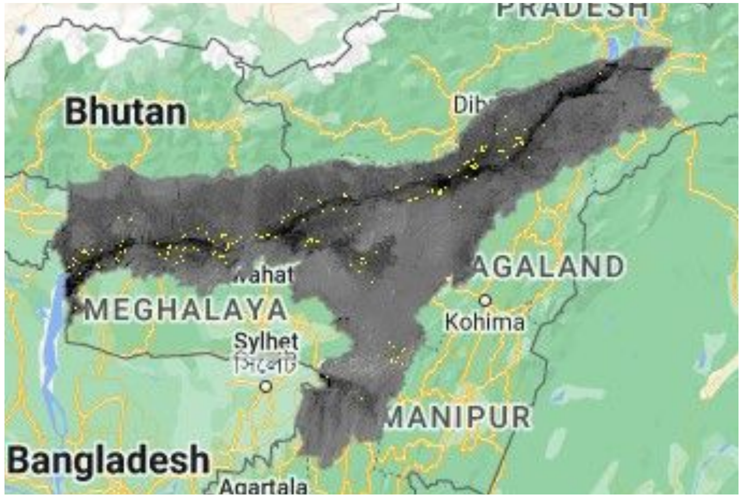
```
// Mask out areas with more than 5 percent slope using the HydroSHEDS DEM
var slopeThreshold = 5;
var terrain = ee.Algorithms.Terrain(hydrosheds);
var slope = terrain.select('slope');
var flooded = flooded.updateMask(slope.lt(slopeThreshold));
Map.addLayer(slope.gte(slopeThreshold).selfMask(), {min:0, max:1, palette: ['cyan']}, 'Steep Areas', false)
```



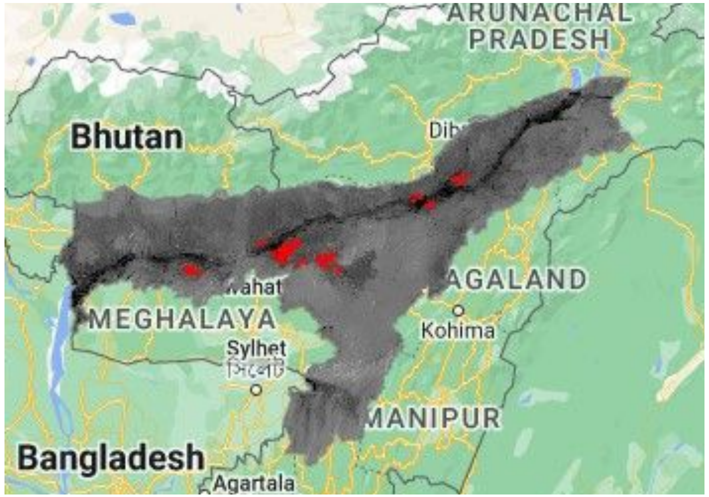
STEEP AREAS



```
// Remove isolated pixels
// connectedPixelCount is Zoom dependent, so visual result will vary
var connectedPixelThreshold = 8;
var connections = flooded.connectedPixelCount(25)
var flooded = flooded.updateMask(connections.gt(connectedPixelThreshold))
Map.addLayer(connections.lte(connectedPixelThreshold).selfMask(), {min:0, max:1, palette: ['yellow']}, 'Disconnected Areas', false)

Map.addLayer(flooded, {min:0, max:1, palette: ['red']}, 'Flooded Areas');
```



DISCONNECTED AREAS



FLOODED AREAS

OUR PLAN GOING AHEAD

- Convert the data to vector form
- Design ML models to predict future flood spread based on current rainfall