



POLITECNICO
MILANO 1863

Analyzing the Amazon Deforestation with Machine Learning and the Google Earth Engine Part II

Teachers: Maria Antonia Brovelli and Vasil Yordanov
maria.brovelli@polimi.it vasil.yordanov@polimi.it

Assitants: Ahmed Abdalgade Ahmed Eisaa, Alessandro Austoni, Juan Francisco Amieva, Julia Anna Leonardi, Ahmed Omer Ahmed Mukhtar, Nikolina Zallemei, Martina Giovanna Esposito and Ahmed Mohamed Eltahir Yassin

Politecnico di Milano – DICA | GEOLab
29 March 2022

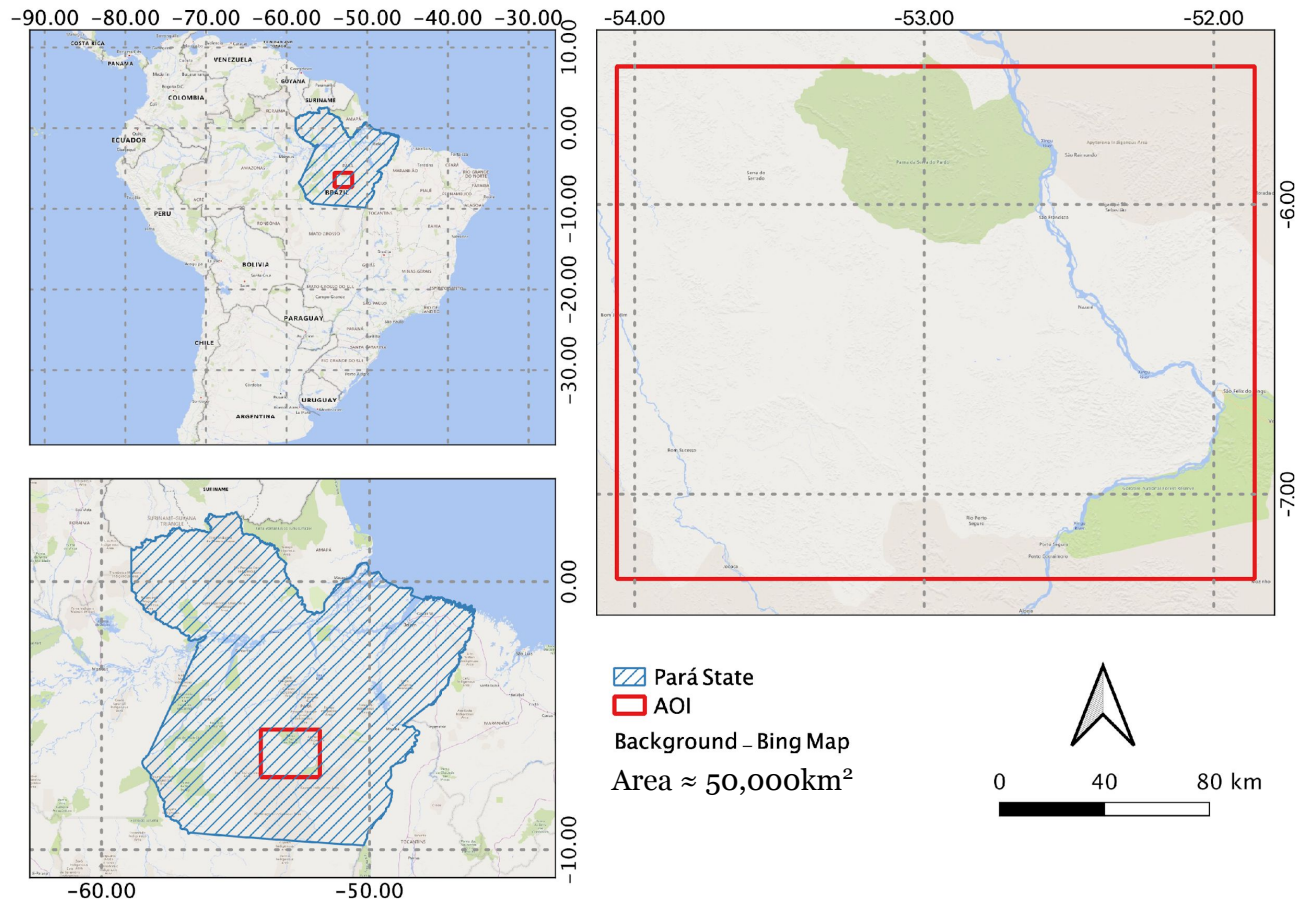


01

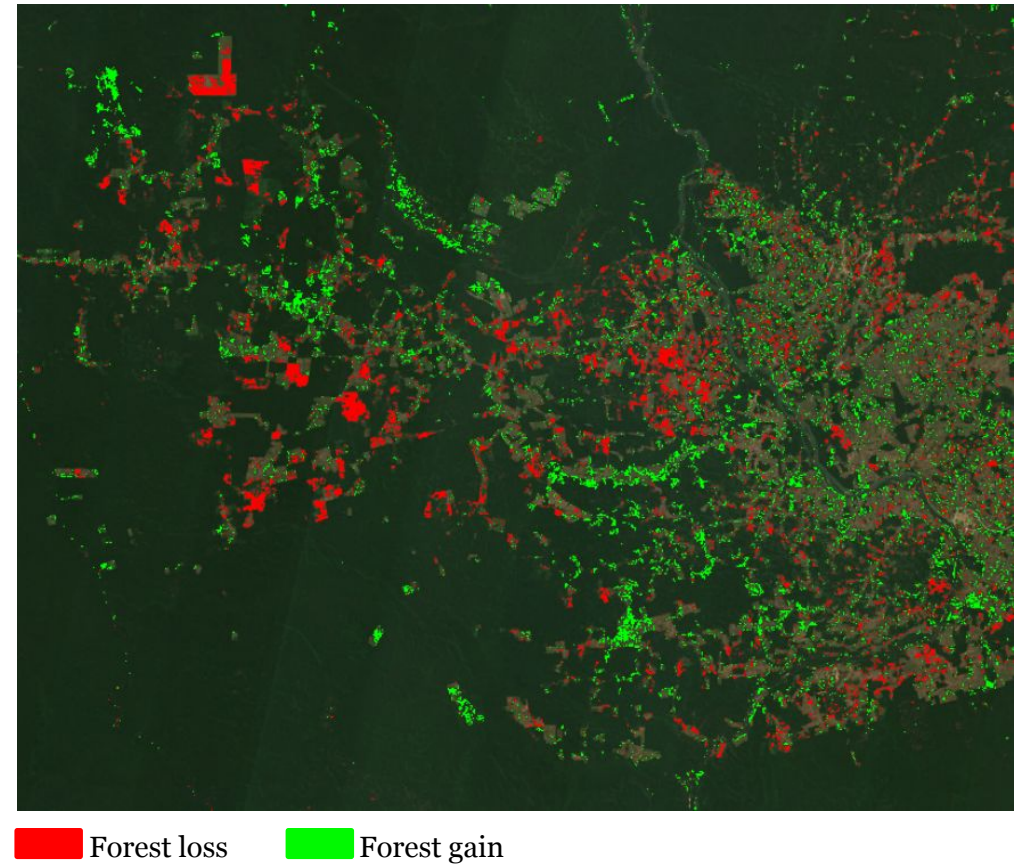
Let's recap what we did in Part I

Case study and aim - recap

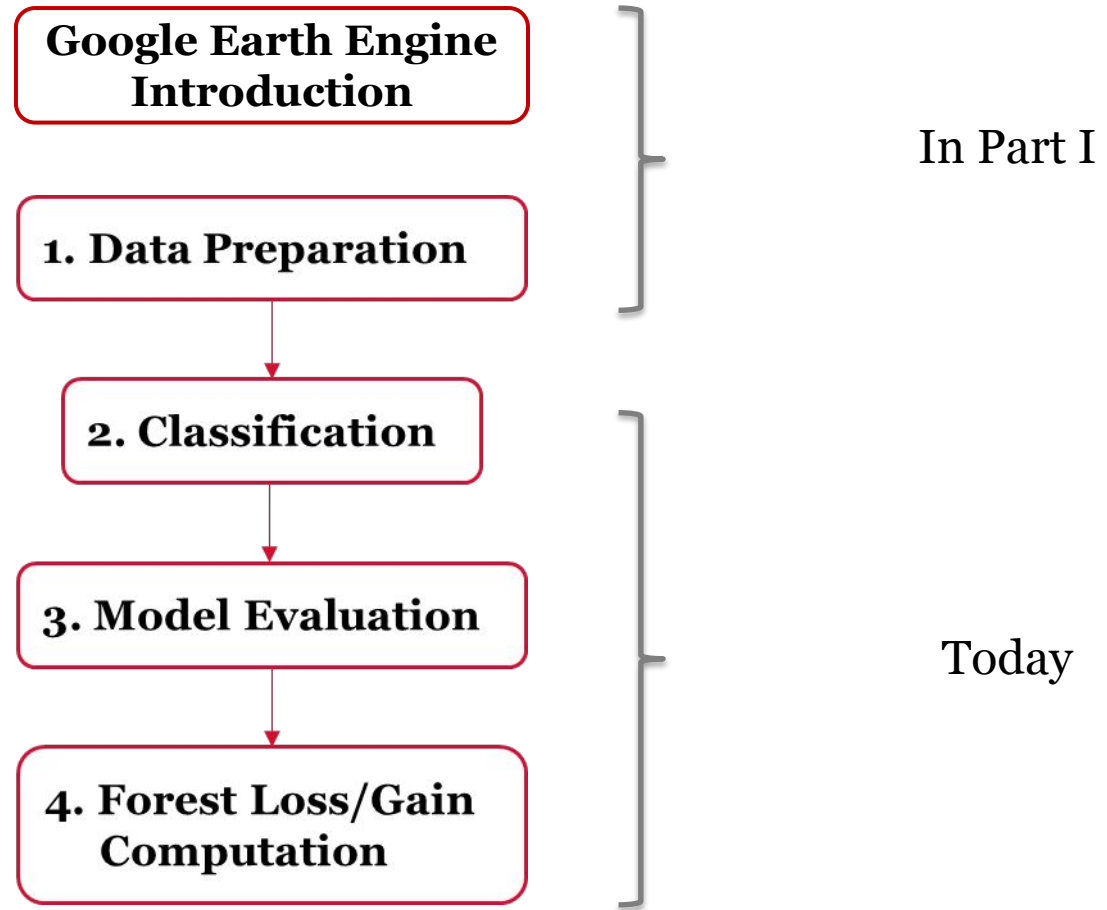
Area of interest – Parà state, Brazil



Aim – map forest loss and gain in the period 2015 - 2019



Workflow - Recap



1. Data Preparation

2. Classification

3. Model Evaluation

4. Forest Loss/Gain Computation

Define Area of Interest (AOI)

Create cloud mask

Define two periods of the analysis

Define data to be used

Apply filters to the data

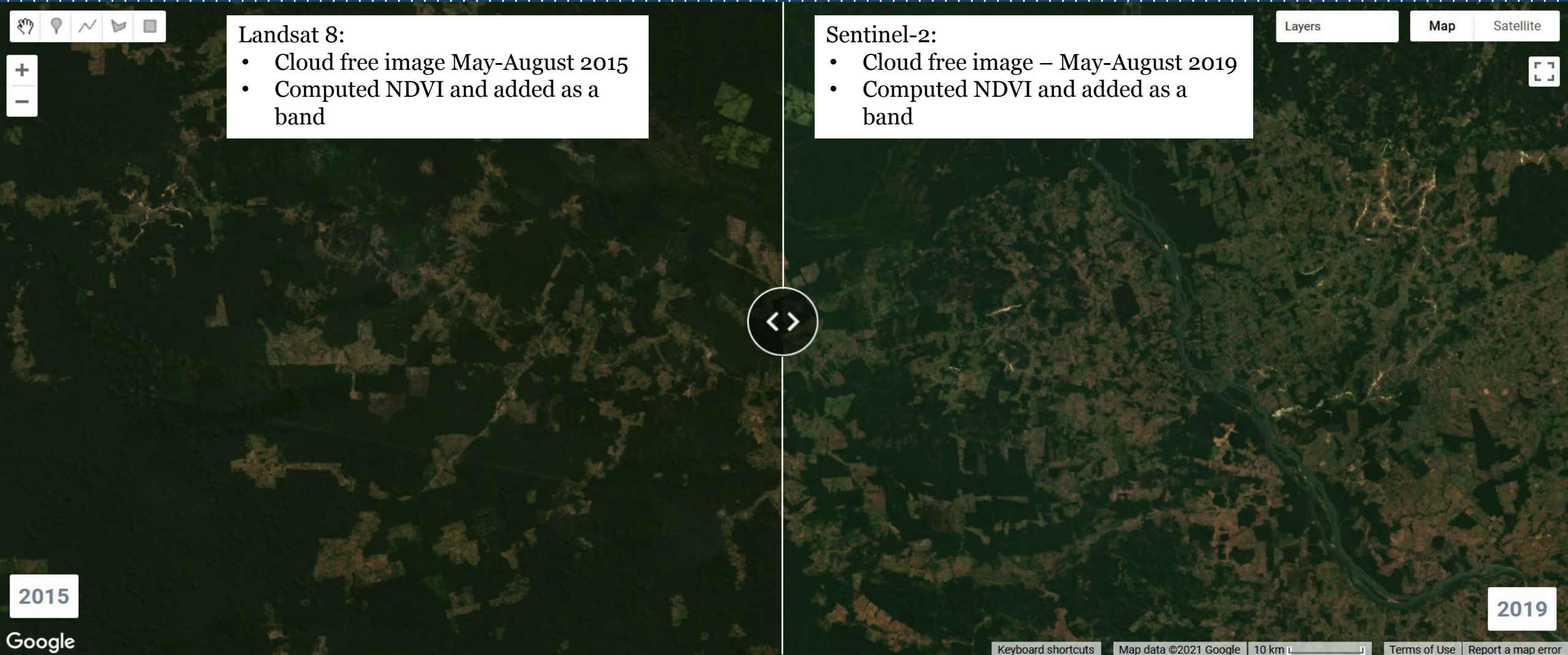
Where to find the code written last time ?

Here we are, [link to the code from Part I](#)

If you have saved the script from the previous session, you can directly continue working in it.

If you are just joining us, copy and paste the script from Part I in your code editor and continue from there.

Results of Part I





What we will do today

1. Data Preparation

2. Classification

3. Model Evaluation

4. Forest Loss/Gain
Computation

Define areas that will be
used for model training and
testing

Train and apply classifier for
both periods

1. Data Preparation

2. Classification

3. Model Evaluation

4. Forest Loss/Gain
Computation

Compute error matrix



Compute evaluation metrics
(overall accuracy, Cohen's Kappa)

1. Data Preparation

2. Classification

3. Model Evaluation

4. Forest Loss/Gain
Computation

Compute the difference
between the two classified
images

Mask the forest loss/gain
areas

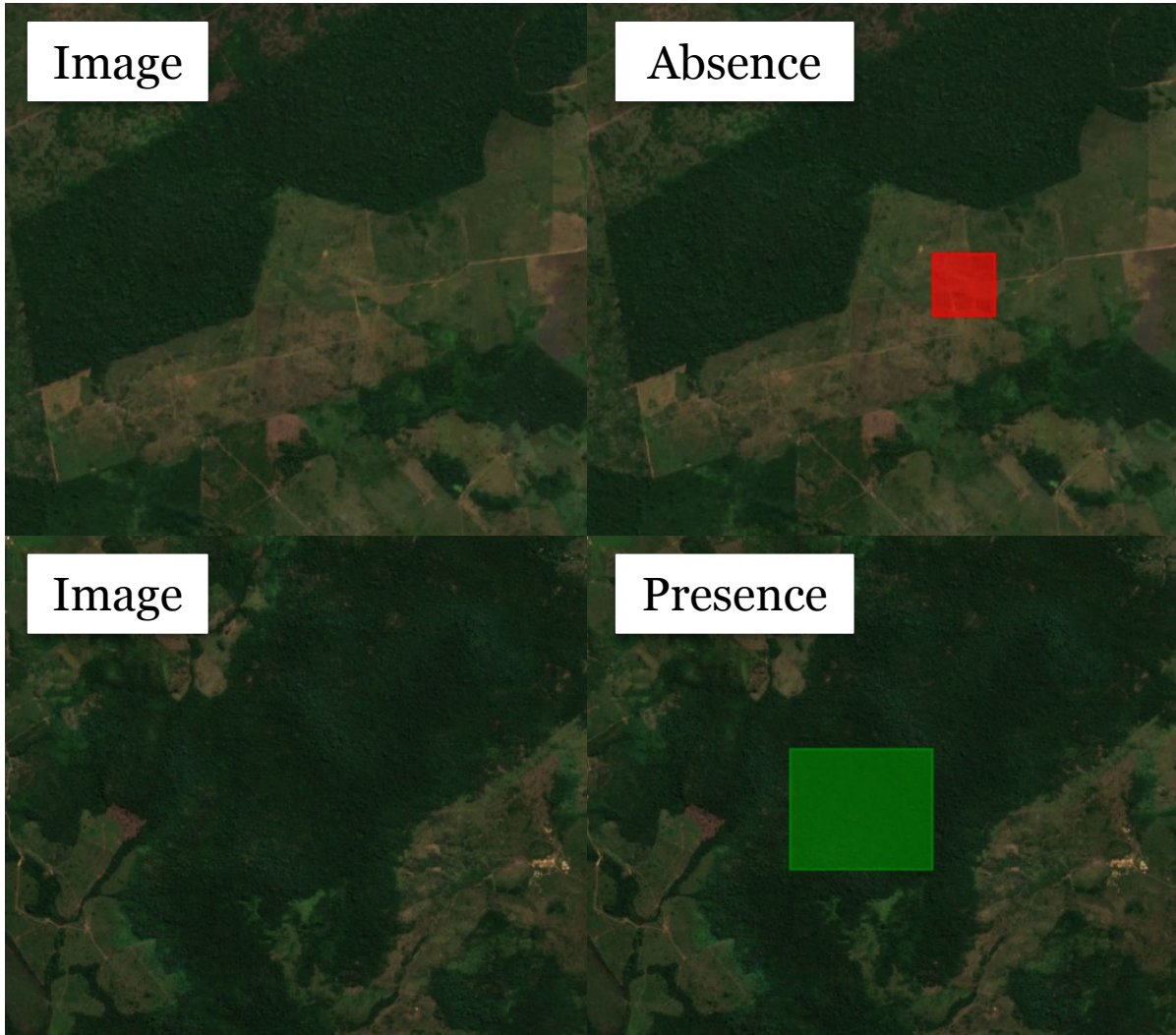
Compute the total and
absolute areas



02

Preparation of training data

Preparing training and testing datasets



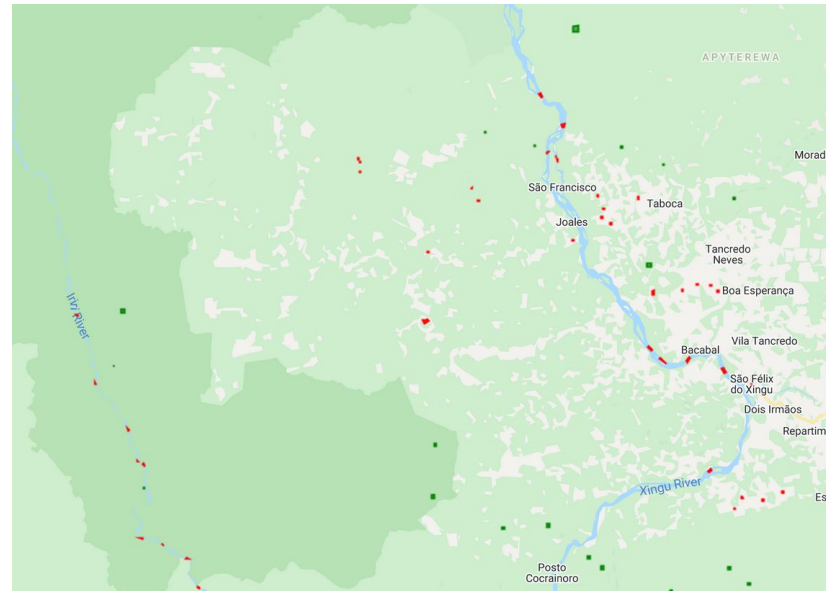
Feature collections with relevant and consistent property and values

The screenshot shows the 'Configure geometry import' dialog in QGIS for two feature collections. The top dialog is for 'no_forest' with a red color swatch (#d63000) and a 'landcover' property set to 0. The bottom dialog is for 'forest' with a green color swatch (#07d600) and a 'landcover' property set to 1. Both dialogs show a 'FeatureCollection' import type and a '+ Property' button.

Collection Name	Color	Property	Value
no_forest	#d63000	landcover	0
forest	#07d600	landcover	1

Preparing training and testing datasets

```
// Load training and testing datasets
var no_forest =
ee.FeatureCollection('users/vyordanov/workshops/deforestation/nonForestPolygons')
var forest =
ee.FeatureCollection('users/vyordanov/workshops/deforestation/forestPolygons')
Map.addLayer(no_forest.draw('red'), {}, 'No Forest Polygons')
Map.addLayer(forest.draw('green'), {}, 'Forest Polygons')
```



Landsat 2015 image classification

```
// Define the spectral bands that will be used during the classification,  
// Red (SR_B4), Green (SR_B3), Blue (SR_B2), Near-Infrared (SR_B5) and NDVI:  
var bandsLandsat = ['SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'NDVI'];  
  
// Sample the created regions from the Landsat image:  
var sampledNonForestLandsat = landsat2015.select(bandsLandsat).sampleRegions({  
  collection: no_forest, // The regions to sample over  
  properties: ['landcover'], //The list of properties to copy from each input feature  
  scale: 30 // resolution of the image (in this case Landsat)  
});  
// Split the sampled points into train set containing 80% of the sampled points  
// and test set containing 20% of the sampled points:  
var threshold=0.8;  
var trainNonForestLandsat =  
sampledNonForestLandsat.randomColumn('random').filter(ee.Filter.lte('random',threshold));  
var testNonForestLandsat =  
sampledNonForestLandsat.randomColumn('random').filter(ee.Filter.gt('random', threshold));
```

Landsat 2015 image classification

```
// Define the spectral bands that will be used during the classification,  
// Red (SR_B4), Green (SR_B3), Blue (SR_B2), Near-Infrared (SR_B5) and NDVI:  
var bandsLandsat = ['SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'NDVI'];  
  
// Sample the created regions from the Landsat image:  
var sampledNonForestLandsat = landsat2015.select(bandsLandsat).sampleRegions({  
  collection: no_forest, // The regions to sample over  
  properties: ['landcover'], //The list of properties to copy from each input feature  
  scale: 30 // resolution of the image (in this case Landsat)  
});  
// Split the sampled points into train set containing 80% of the sampled points  
// and test set containing 20% of the sampled points:  
var threshold=0.8;  
var trainNonForestLandsat =  
sampledNonForestLandsat.randomColumn('random').filter(ee.Filter.lte('random',threshold));  
var testNonForestLandsat =  
sampledNonForestLandsat.randomColumn('random').filter(ee.Filter.gt('random', threshold));
```

Landsat 2015 image classification

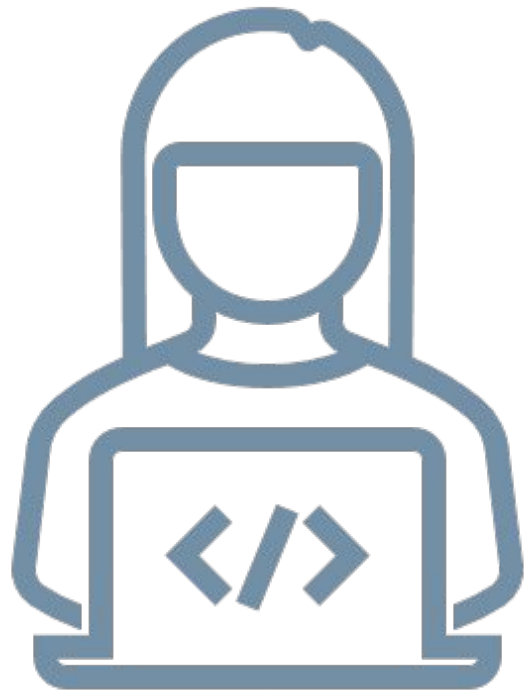
```
// Sample the created regions from the Landsat image:
var sampledForestLandsat = landsat2015.select(bandsLandsat).sampleRegions({
  collection: forest, // The regions to sample over
  properties: ['landcover'], //The list of properties to copy from each input feature
  scale: 30 // resolution of the image (in this case Landsat)
});
// Split the sampled points into train set containing 80% of the sampled points
// and test set containing 20% of the sampled points:
var threshold=0.8;
var trainForestLandsat =
sampledForestLandsat.randomColumn('random').filter(ee.Filter.lte('random',threshold));
var testForestLandsat =
sampledForestLandsat.randomColumn('random').filter(ee.Filter.gt('random', threshold));

// Merge both feature collections in one:
var trainLandsat = trainForestLandsat.merge(trainNonForestLandsat);
var testLandsat = testForestLandsat.merge(testNonForestLandsat);
```


Landsat 2015 image classification

```
// Sample the created regions from the Landsat image:
var sampledForestLandsat = landsat2015.select(bandsLandsat).sampleRegions({
  collection: forest // The regions to sample over
  properties: ['landcover'], //The list of properties to copy from each input feature
  scale: 30 // resolution of the image (in this case Landsat)
});
// Split the sampled points into train set containing 80% of the sampled points
// and test set containing 20% of the sampled points:
var threshold=0.8;
var trainForestLandsat =
sampledForestLandsat.randomColumn('random').filter(ee.Filter.lte('random',threshold));
var testForestLandsat =
sampledForestLandsat.randomColumn('random').filter(ee.Filter.gt('random', threshold));

// Merge both feature collections in one:
var trainLandsat = trainForestLandsat.merge(trainNonForestLandsat);
var testLandsat = testNonForestLandsat.merge(testNonForestLandsat);
```



Google Earth Engine



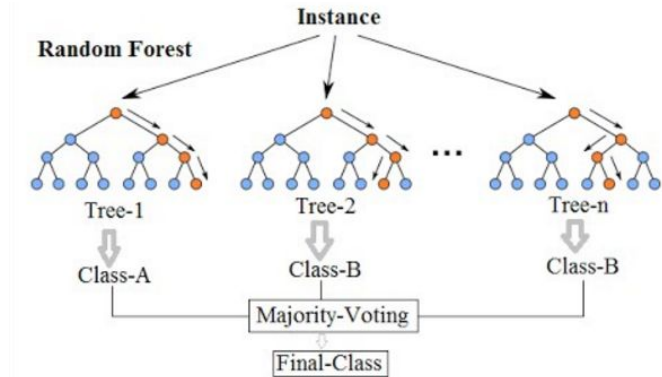
03

Classification

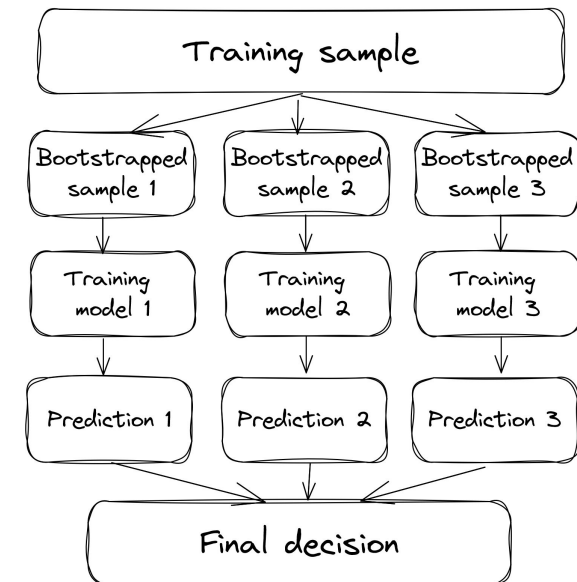
Random Forest

Supervised machine learning algorithm used widely for *classification* and *regression* problems.

- It is based on decision trees
- Relies on bagging technique – is the process in which multiple models of the same learning algorithm are trained with bootstrapped samples of the original dataset (reduces variance).
- Where the final vote is based on *majority* (classification) or *average* (regression)
- Good performance “out-of-the-box” with few parameters to tune
- Can work both with continuous and categorical data
- It can be computationally slow during training process because each tree has to create an output



Source: Wikimedia Commons



Landsat 2015 image classification

```
// Train the RandomForest classifier with the train dataset,  
// assigning the desired number of trees, specifying the class property  
// and the input properties:  
var classifier2015 = ee.Classifier.smileRandomForest(10).train({  
  features: trainLandsat,  
  classProperty: 'landcover',  
  inputProperties: bandsLandsat  
});  
  
// Get a confusion matrix and compute the accuracy of the model performance:  
print('Landsat RF error matrix: ', classifier2015.confusionMatrix());  
print('Landsat RF accuracy: ', classifier2015.confusionMatrix().accuracy());  
print("Landsat RF Cohen's Kappa:", classifier2015.confusionMatrix().kappa());
```

```
Landsat RF error matrix:      JSON  
▼[[[12583,14],[14,13145]]]   JSON  
  ▶0: [12583,14]  
  ▶1: [14,13145]
```

```
Landsat RF accuracy:      JSON  
0.9989128746699798
```

```
Landsat RF Cohen's Kappa:  JSON  
0.9978247136441142
```

Landsat 2015 image classification

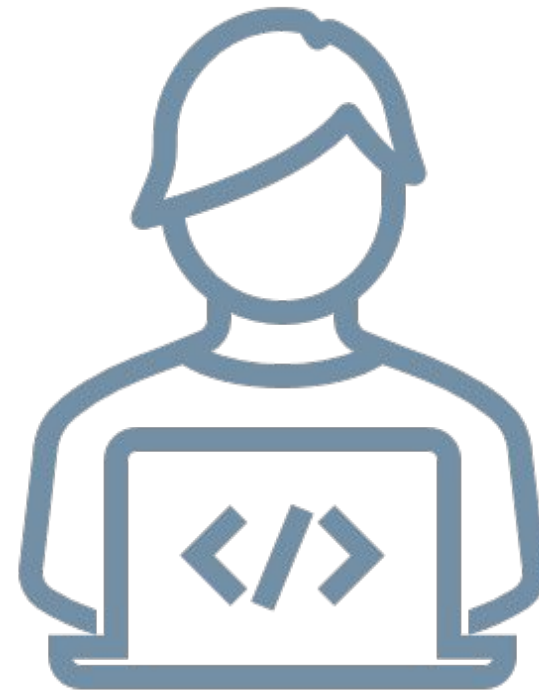
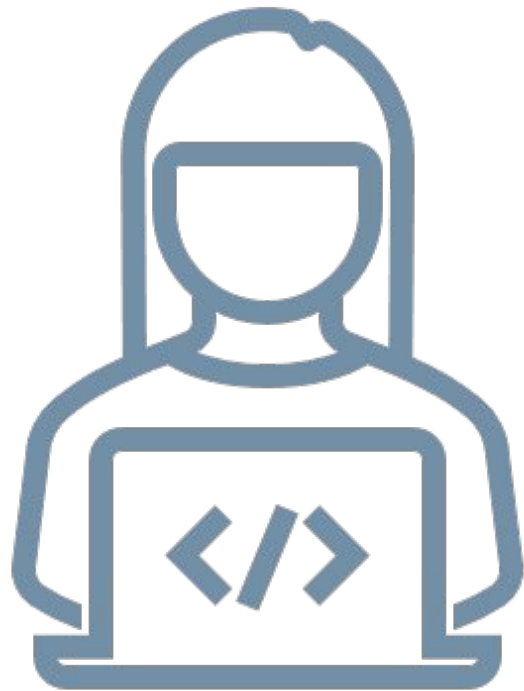
```
// Apply the classifier to the Landsat image:
var classified2015 = landsat2015.select(bandsLandsat).classify(classifier2015);

// Classify the test sample, get a confusion matrix and compute the accuracy of the
// model test performance:
var testingLandsat = testLandsat.classify(classifier2015);
var testAccuracyLandsat = testingLandsat.errorMatrix('landcover', 'classification');
print('Landsat Validation error matrix: ', testAccuracyLandsat);
print('Landsat Validation overall accuracy: ', testAccuracyLandsat.accuracy());
print("Landsat Validation Cohen's Kappa:", testAccuracyLandsat.kappa());
```

```
Landsat Validation error matrix: JSON
▼ [[3127,25],[26,3174]] JSON
  ▶ 0: [3127,25]
  ▶ 1: [26,3174]
```

```
Landsat Validation overall accuracy: JSON
0.991971032745592
```

```
Landsat Validation Cohen's Kappa: JSON
0.9839411866881662
```



Google Earth Engine

Testing and validation

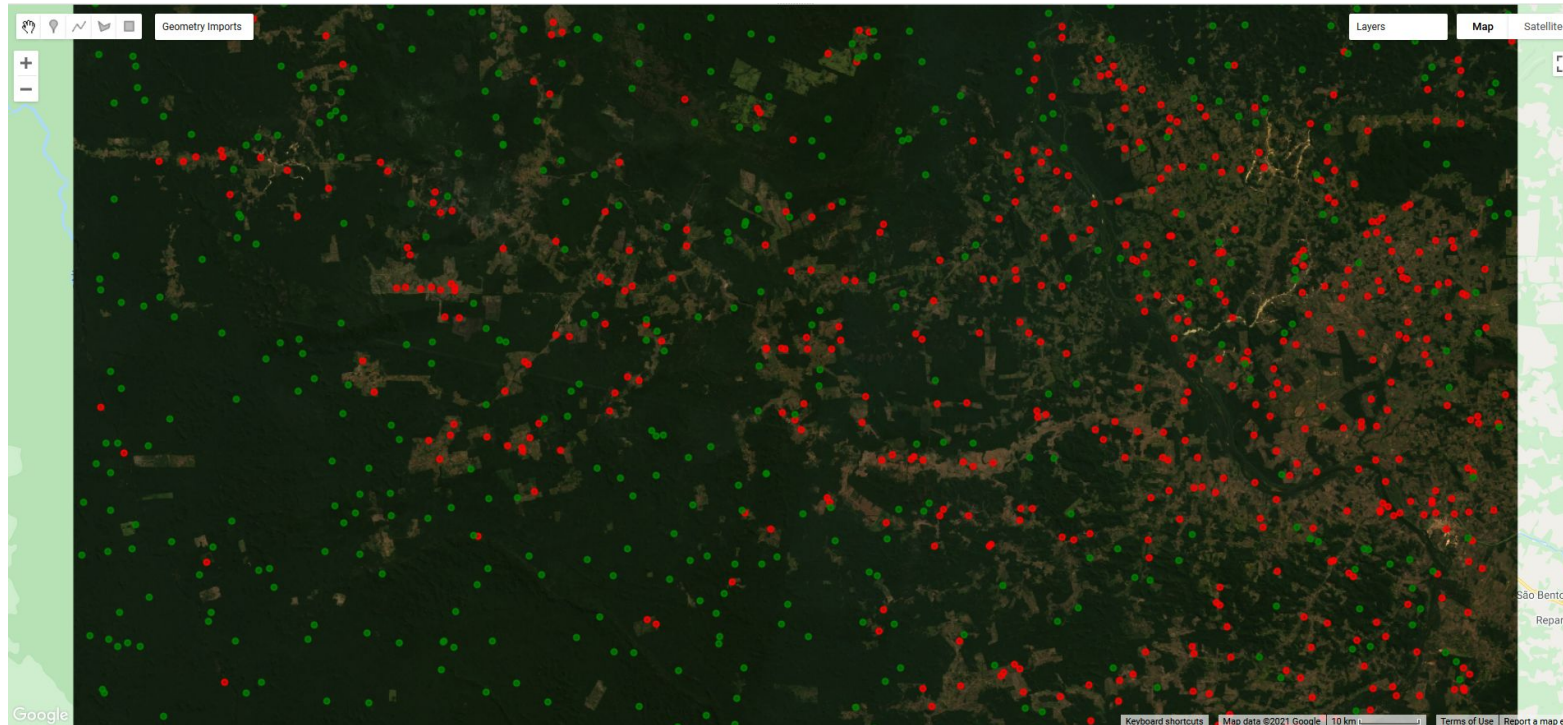
- **Model fit** – we test how the model is trained using the training set
- **Model test** – we test the predictive performance of the model using the testing set
- **Model validation** – we test the predictive performance of the model using external validation set (if present)
- **Confusion matrix** - summarizing table of correct and incorrect predictions
- **Overall accuracy** - is proportion of correctly classified pixels over the total number of pixels $OA = \frac{\text{all correctly classified pixels}}{\text{all pixels}}$
- **Cohen's Kappa coefficient** - is a metric often used to assess the agreement between two sets

$$K = \frac{OA - P_e}{1 - P_e}, \text{ where } P_e = \frac{(TP+FP)*(TP+FN) + (TN+FP)*(TN+FN)}{(TP+TN+FP+FN)^2}$$

Confusion matrix

	Actual positive	Actual negative
Predicted positive	True Positives (TP)	False Positives (FP)
Predicted negative	False Negatives (FN)	True Negatives (TN)

Landsat 2015 image classification – External validation



External validation:
We are using a manually classified external dataset obtained through photointerpretation. It is additional validation, that can be done only when there is such external dataset.

```
// Load, classify the External dataset, get a confusion matrix and compute the accuracy of the model test performance:  
var refPoints_2015 = ee.FeatureCollection('users/vyordanov/Amazon/Amazon2015_refPoints')  
Map.addLayer(refPoints_2015.filter(ee.Filter.eq('land_cover',0)).draw('red'), {}, 'External Validation Points NF 2015', 0)  
Map.addLayer(refPoints_2015.filter(ee.Filter.eq('land_cover',1)).draw('green'), {}, 'External Validation Points F 2015', 0)
```

Landsat 2015 image classification – External validation

```
refPoints_2015 = refPoints_2015.map(function(feat){
  return ee.Feature(feat.geometry(), {
    landcover: feat.get('land_cover'),
  })
})

var sampleRefPointsLandsat = classified2015.select('classification').sampleRegions({
  collection: refPoints_2015,
  properties: ['landcover'],
  scale: 30
});
print(sampleRefPointsLandsat)
var refAccuracyLandsat = sampleRefPointsLandsat.errorMatrix('landcover', 'classification');
print('Landsat External Validation error matrix: ', refAccuracyLandsat);
print('Landsat External Validation overall accuracy: ', refAccuracyLandsat.accuracy());
print("Landsat External Validation Cohen's Kappa:", refAccuracyLandsat.kappa());
```

```
Landsat External Validation error matrix:
▼ [[496,21],[52,521]]
  ▶0: [496,21]
  ▶1: [52,521]
```

```
Landsat External Validation overall accuracy:
0.9330275229357798
```

```
Landsat External Validation Cohen's Kappa:
0.8660929153946868
```

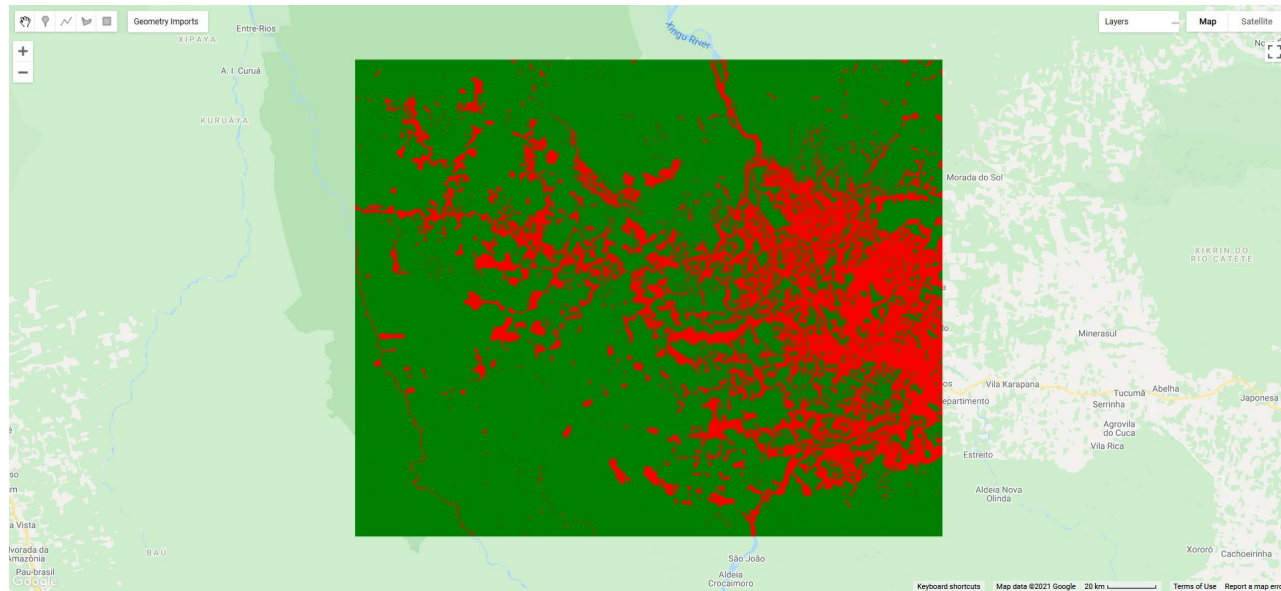
Landsat 2015 image classification – adding the map

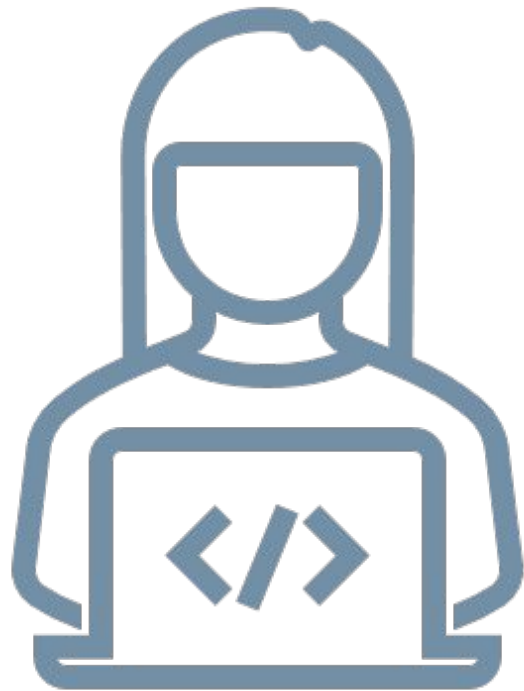
```
// Define a color palette for the classification map
```

```
var palette = [  
  'red', // non-forest  
  'green', // forest  
];
```

```
// Add the classified maps
```

```
Map.addLayer(classified2015, {min: 0, max: 1, palette: palette}, 'Forest Classification 2015',0);
```





Google Earth Engine



04

Classifying Sentinel-2 2019 image

Sentinel 2019 image classification – same procedure

PART 1

```
/* Classify Sentinel-2*/
// Define the spectral bands that will be used during the classification,
// Red (B4), Green (B3), Blue (B2) and Near-Infrared (B8):
var bandSentinel = ['B2', 'B3', 'B4', 'B8','NDVI'];
// Sample the created regions from the Landsat image:
var sampledNonForestSentinel= sentinel2019.select(bandSentinel).sampleRegions({
  collection: no_forest,
  properties: ['landcover'],
  scale: 10
});
// Split the sampled points into train set containing 80% of the sampled points
// and test set containing 20% of the sampled points:
var threshold=0.8;
var trainNonForestSentinel = sampledNonForestSentinel.randomColumn('random').filter(ee.Filter.lte('random',threshold));
var testNonForestSentinel = sampledNonForestSentinel.randomColumn('random').filter(ee.Filter.gt('random', threshold)); //
// Sample the created regions from the Landsat image:
var sampledForestSentinel= sentinel2019.select(bandSentinel).sampleRegions({
  collection: forest,
  properties: ['landcover'],
  scale: 10
});
// Split the sampled points into train set containing 80% of the sampled points
// and test set containing 20% of the sampled points:
var threshold=0.8;
var trainForestSentinel = sampledForestSentinel.randomColumn('random').filter(ee.Filter.lte('random',threshold));
var testForestSentinel = sampledForestSentinel.randomColumn('random').filter(ee.Filter.gt('random', threshold));

// Merge both feature collections in one:
var trainSentinel = trainForestSentinel.merge(trainNonForestSentinel);
var testSentinel = testForestSentinel.merge(testNonForestSentinel);

// Train the RandomForest classifier with the train dataset,
// assigning the desired number of trees, specifying the class property
// and the input properties:
var classifierSentinel = ee.Classifier.smileRandomForest(10).train({
  features: trainSentinel,
  classProperty: 'landcover',
  inputProperties: bandSentinel
});
```

PART 2

```
// Get a confusion matrix and compute the accuracy of the model performance:
print('Sentinel RF error matrix: ', classifierSentinel.confusionMatrix());
print('Sentinel RF accuracy: ', classifierSentinel.confusionMatrix().accuracy());
print("Sentinel RF Cohen's Kappa:", classifierSentinel.confusionMatrix().kappa());

// Apply the classifier to the Landsat image:
var classified2019 = sentinel2019.select(bandSentinel).classify(classifierSentinel);

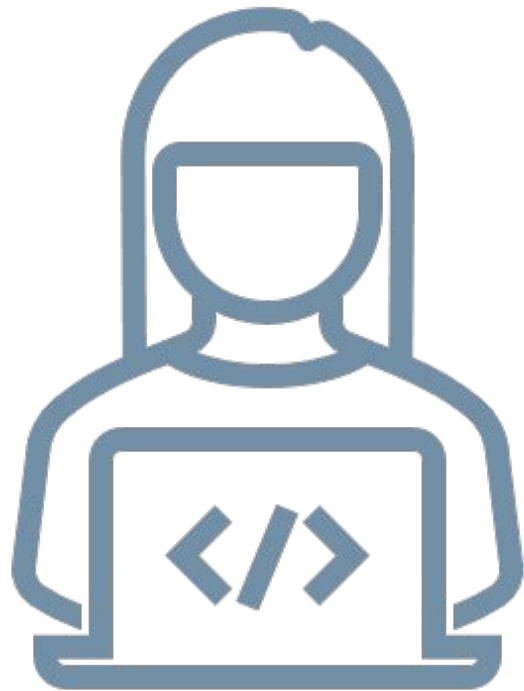
// Classify the test sample, get a confusion matrix and compute the accuracy of the model test performance:
var testingSentinel = testSentinel.classify(classifierSentinel);
var testAccuracySentinel = testingSentinel.errorMatrix('landcover', 'classification');
print('Sentinel Validation error matrix: ', testAccuracySentinel);
print('Sentinel Validation overall accuracy: ', testAccuracySentinel.accuracy());
print("Sentinel Validation Cohen's Kappa:", testAccuracySentinel.kappa());

// Classify the External dataset, get a confusion matrix and compute the accuracy of the model test
//performance:
var refPoints_2019 = ee.FeatureCollection('users/vyordanov/Amazon/Amazon2019_refPoints')

refPoints_2019 = refPoints_2019.map(function(feats){
  return ee.Feature(feats.geometry(), {
    landcover: feats.get('land_cover'),
  })
});

var sampleRefPointsSentinel = classified2019.select('classification').sampleRegions({
  collection: refPoints_2019,
  properties: ['landcover'],
  scale: 10
});
print(sampleRefPointsSentinel)
var refAccuracySentinel = sampleRefPointsSentinel.errorMatrix('landcover', 'classification');
print('Sentinel External Validation error matrix: ', refAccuracySentinel);
print('Sentinel External Validation overall accuracy: ', refAccuracySentinel.accuracy());
print("Sentinel External Validation Cohen's Kappa:", refAccuracySentinel.kappa());

// Add the classified maps
Map.addLayer(classified2019, {min: 0, max: 1, palette: palette}, 'Forest Classification 2019');
```



Google Earth Engine



05

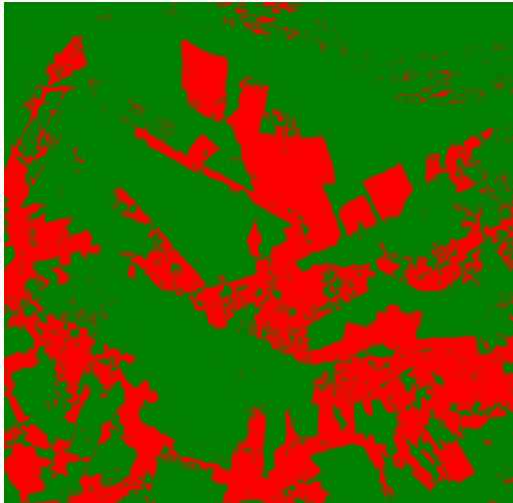
Filtering the classification results

Let's explore the classification maps

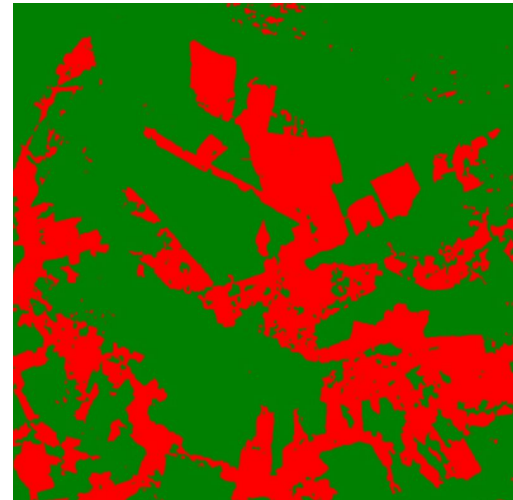
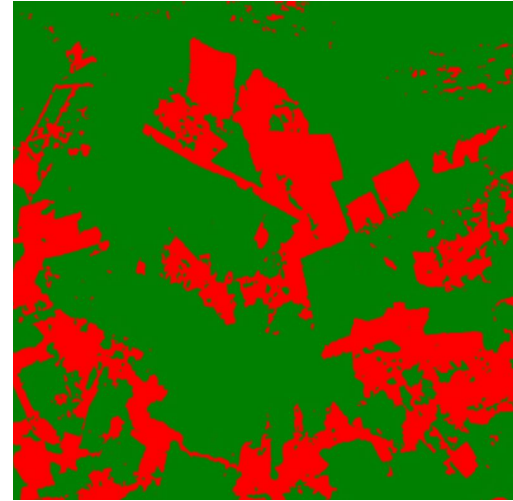
Landsat 2015



Sentinel 2019



Focal mode



Apply focal mode reducer

```
// Clean the maps from misclassified pixels
```

```
var classified2015clean = classified2015.focalMode(2)  
Map.addLayer(classified2015clean, {min: 0, max: 1, palette: palette}, 'Forest Classification 2015 Cleaned');
```

```
var classified2019clean = classified2019.focalMode(2)  
Map.addLayer(classified2019clean, {min: 0, max: 1, palette: palette}, 'Forest Classification 2019 Cleaned');
```



06

Resampling and matching spatial resolution

Resample to match spatial resolution

```
//Resample the images and force GEE to compute them at 10m/pix
```

```
var classified2015_10m = classified2015clean.resample('bicubic').reproject({  
  'crs': 'EPSG:32721',  
  'scale': 10})
```

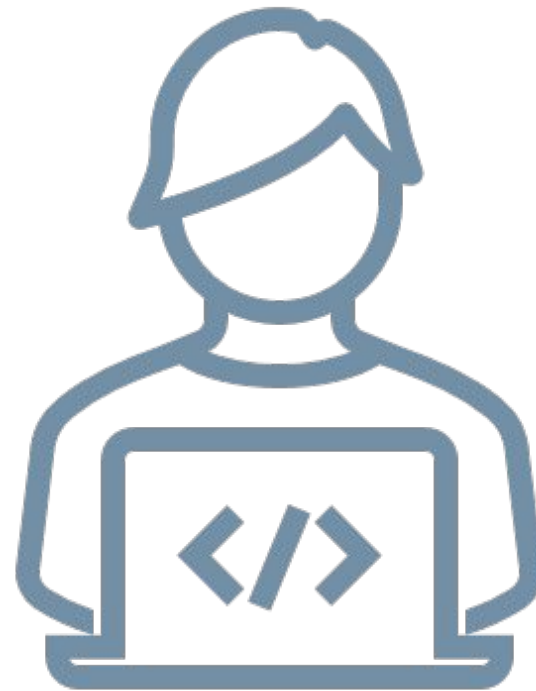
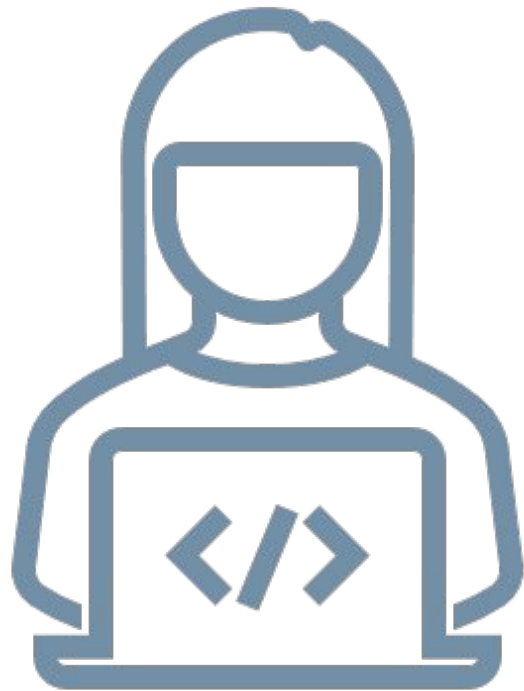
```
print('Forest Classification 2015 10m scale:', classified2015_10m.projection().nominalScale());
```

```
var classified2019_10m = classified2019clean.reproject({  
  'crs': 'EPSG:32721',  
  'scale': 10})
```

```
print('Forest Classification 2019 10m scale:', classified2019_10m.projection().nominalScale());
```

```
Forest Classification 2015 10m scale:          JSON  
10
```

```
Forest Classification 2019 10m scale:          JSON  
10
```



Google Earth Engine



07

Intermediary save

Intermediary save of the classification maps*

```
// Export them as an asset
Export.image.toAsset({
  image: classified2015_10m,
  description: 'classified2015_10m',
  assetId: 'PATH_TO_YOUR_ASSET_FOLDER/classified2015_10m',
  scale: 10,
  crs: 'EPSG:32721',
  region: AOI,
  maxPixels: 1e13
})

Export.image.toAsset({
  image: classified2019_10m,
  description: 'classified2019_10m',
  assetId: 'PATH_TO_YOUR_ASSET_FOLDER/classified2019_10m',
  scale: 10,
  crs: 'EPSG:32721',
  region: AOI,
  maxPixels: 1e13
})
// Load them back in GEE as new layers
var classified2015_10m = ee.Image('users/vyordanov/workshops/deforestation/classified2015_10m')
var classified2019_10m = ee.Image('users/vyordanov/workshops/deforestation/classified2019_10m')
```

* Step suggested only if you are working over big AOIs



08

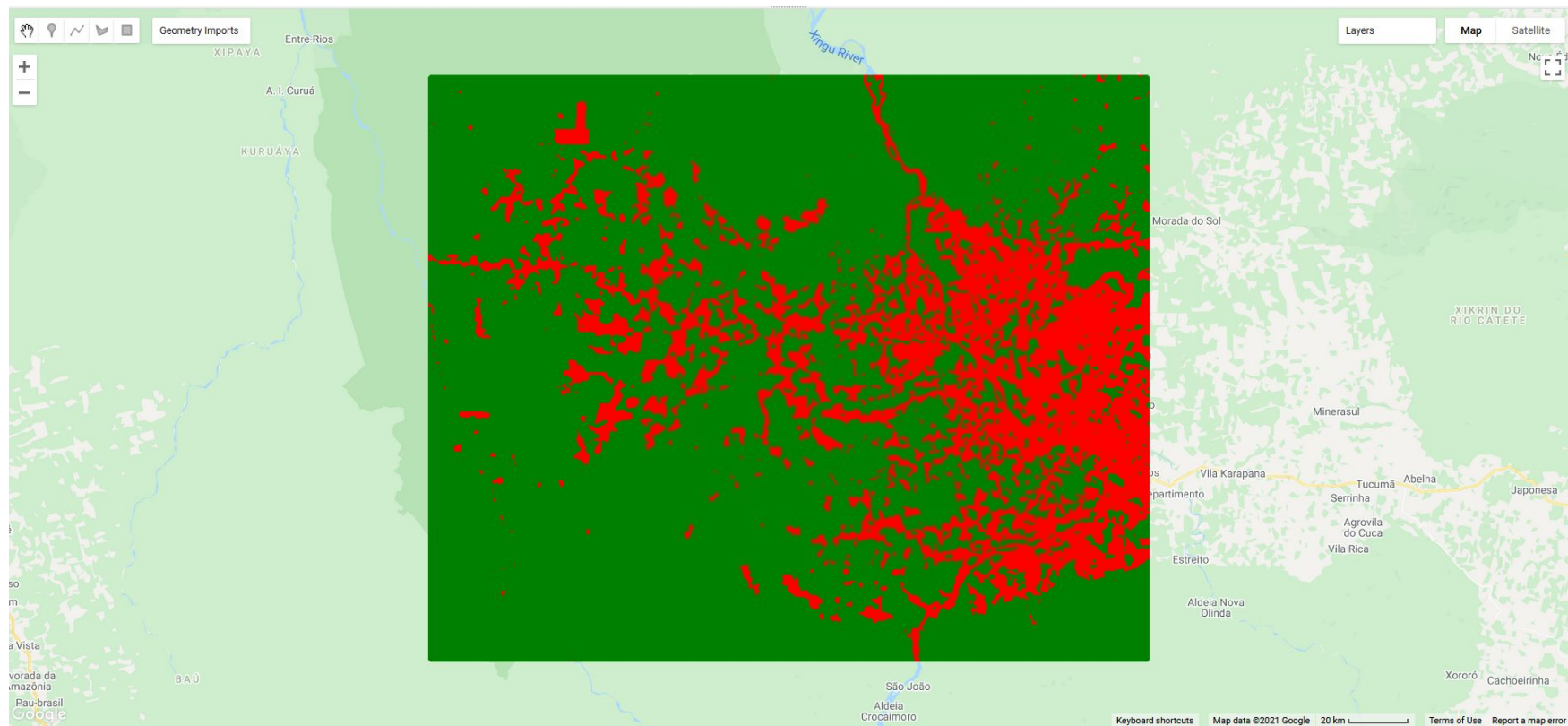
Let's add the maps to the canvas

Add the classification maps to the canvas

```
// Add the classification maps to the canvas
```

```
Map.addLayer(classified2015_10m,{min: 0, max: 1, palette: palette},'Forest Classification 2015 10m',0)
```

```
Map.addLayer(classified2019_10m,{min: 0, max: 1, palette: palette},'Forest Classification 2019 10m',0)
```



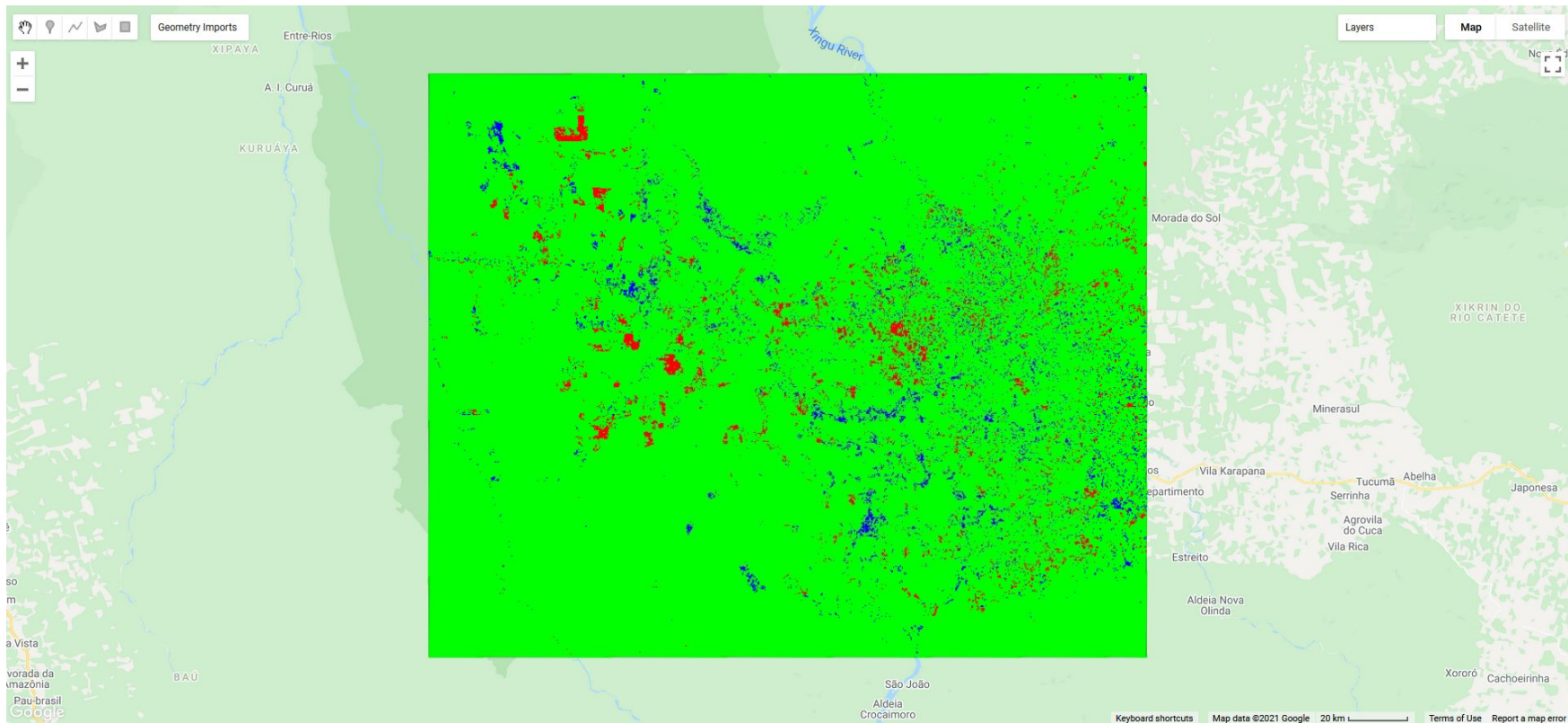


08

Computing map difference

Compute the difference between the two classification maps

```
// Compute the difference between the two classification maps and add it as a new layer:  
var diff=classified2015_10m.subtract(classified2019_10m);  
Map.addLayer(diff,{min:-1,mean:0,max:1, palette:['0000FF','00FF00','FF0000']},'2015-2019 Difference');
```



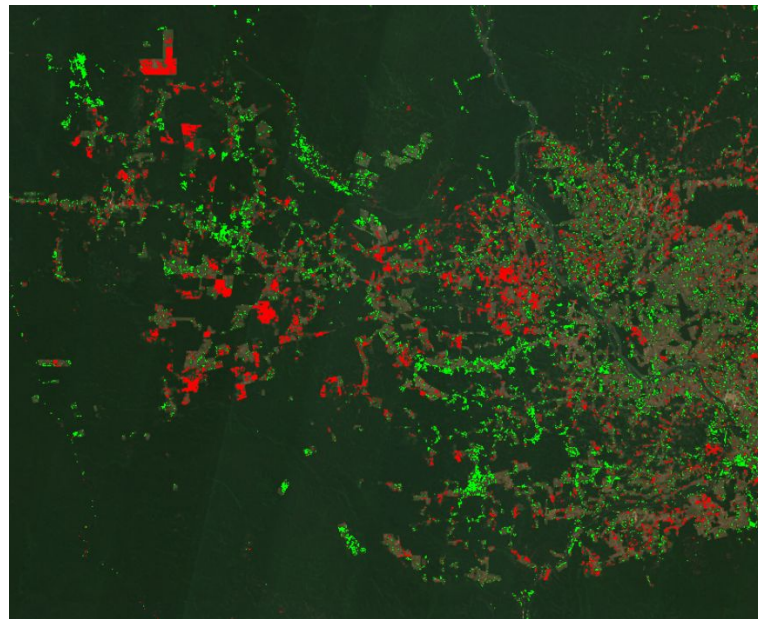


09

Computing forest loss and gain in the period 2015-2019

Extract only the forest loss and gain maps

```
/* 1 Forest - 0 NonForest = 1 Loss // 0 NonForest - 1 Forest = -1 Gain */  
// Add as a layer just the areas that are representing the forest loss:  
var forest_loss=diff.updateMask(diff.eq(1))  
Map.addLayer(forest_loss,{palette:'FF0000'}, 'Forest Loss 2015-2019')  
// Add as a layer just the areas that are representing the forest gain:  
var forest_gain=diff.updateMask(diff.eq(-1))  
Map.addLayer(forest_gain,{palette:'00FF00'}, 'Forest Gain 2015-2019');
```



■ Forest loss ■ Forest gain

Compute the forest loss and gain areas

```
// Compute and print the area of our AOI in km²:
var aoiArea=AOI.geometry().area().divide(1000000);
print('AOI Area: ',aoiArea,"[km²]")

// Compute and print the area of the forest loss and forest gain:
var areaLoss =
forest_loss.multiply(ee.Image.pixelArea().divide(1000000));
var statsLoss = areaLoss.reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: AOI,
  scale: 10,
  maxPixels: 1e13,
  tileSize:16
}).getNumber('classification');
print(
  'Forest Loss 2015-2019:',
  statsLoss,
  "[km²]"
);
```

```
AOI Area:
48551.96934273438
[km²]

Forest Loss 2015-2019:
1601.1842782076726
[km²]

Forest Gain 2015-2019:
1346.5306363939446
[km²]

Relative Loss:
3.2978770992886286
%

Relative Gain:
2.77338005980482
%
```

```
var areaGain = forest_gain.multiply(ee.Image.pixelArea().divide(-1000000));
var statsGain = areaGain.reduceRegion({
  reducer: ee.Reducer.sum(),
  geometry: AOI,
  scale: 10,
  maxPixels: 1e13,
  tileSize:16
}).getNumber('classification');
print(
  'Forest Gain 2015-2019:',
  statsGain,
  "[km²]"
);

// Compute and print the relative area of the forest loss and forest gain
// in relation to the total area of the AOI:
var relLoss=statsLoss.divide(aoiArea);
print("Relative Loss: ",relLoss.multiply(100),"%")

var relGain=statsGain.divide(aoiArea);
print("Relative Gain: ",relGain.multiply(100),"%")
```



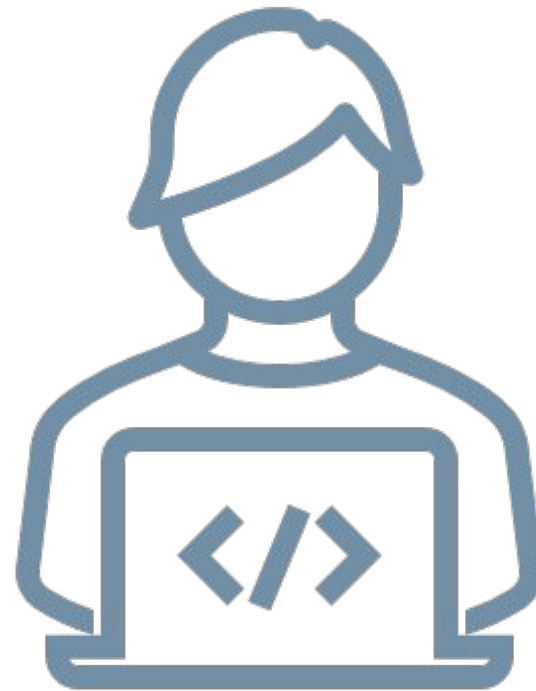
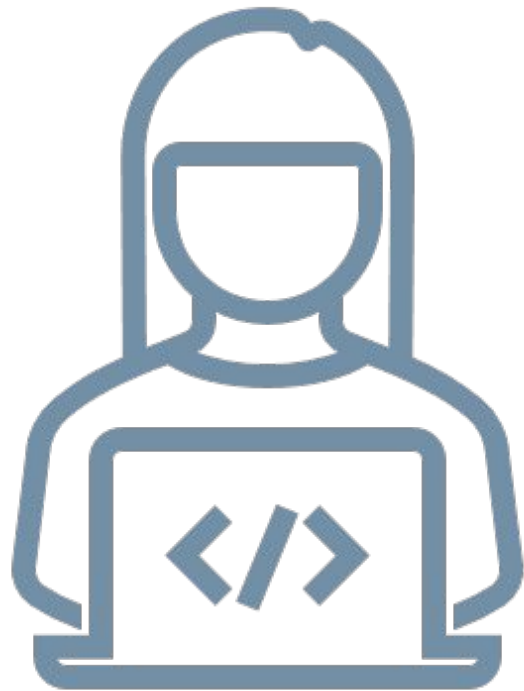
10

Exporting results

Export raster maps

```
// Export the Forest Loss maps:  
Export.image.toDrive({  
  image: forest_loss,  
  description: 'ForestLoss',  
  region: AOI,  
  scale: 10,  
  crs: 'EPSG:32721',  
  folder: 'GEE'  
});
```

```
// Export the Forest Gain maps:  
Export.image.toDrive({  
  image: forest_gain,  
  description: 'ForestGain',  
  region: AOI,  
  scale: 10,  
  crs: 'EPSG:32721',  
  folder: 'GEE'  
});
```



Google Earth Engine

Here you can find the [complete script](#) of both sessions

Thank you for the attention!

maria.brovelli@polimi.it

vasil.yordanov@polimi.it