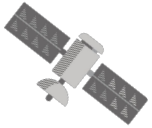
Time-Series Processing using Google Earth Engine



Ujaval Gandhi

Spatial Thoughts

October 2022 | https://bit.ly/g4g22-time-series | #GeoForGood22

Agenda

Slides: https://bit.ly/g4g22-time-series

01. **Moving-Window Smoothing**
    Compute moving-average to remove outliers

02. **Time-Series Gap-Filling**
    Fill masked pixels from temporal neighbors

03. **Time-Series Interpolation**
    Create a regularly spaced time-series

04. **Advanced Time-Series Filtering**
    Apply Savitzky–Golay filter for time-series smoothing

All the methods covered in this talk are dataset agnostic. They can be applied on any gridded dataset having any number of bands.

Earth Observation data is noisy.

The reflected light reaching the satellite is altered by the composition of atmosphere, presence of clouds and directional effects. This introduces noise to the data. [1]

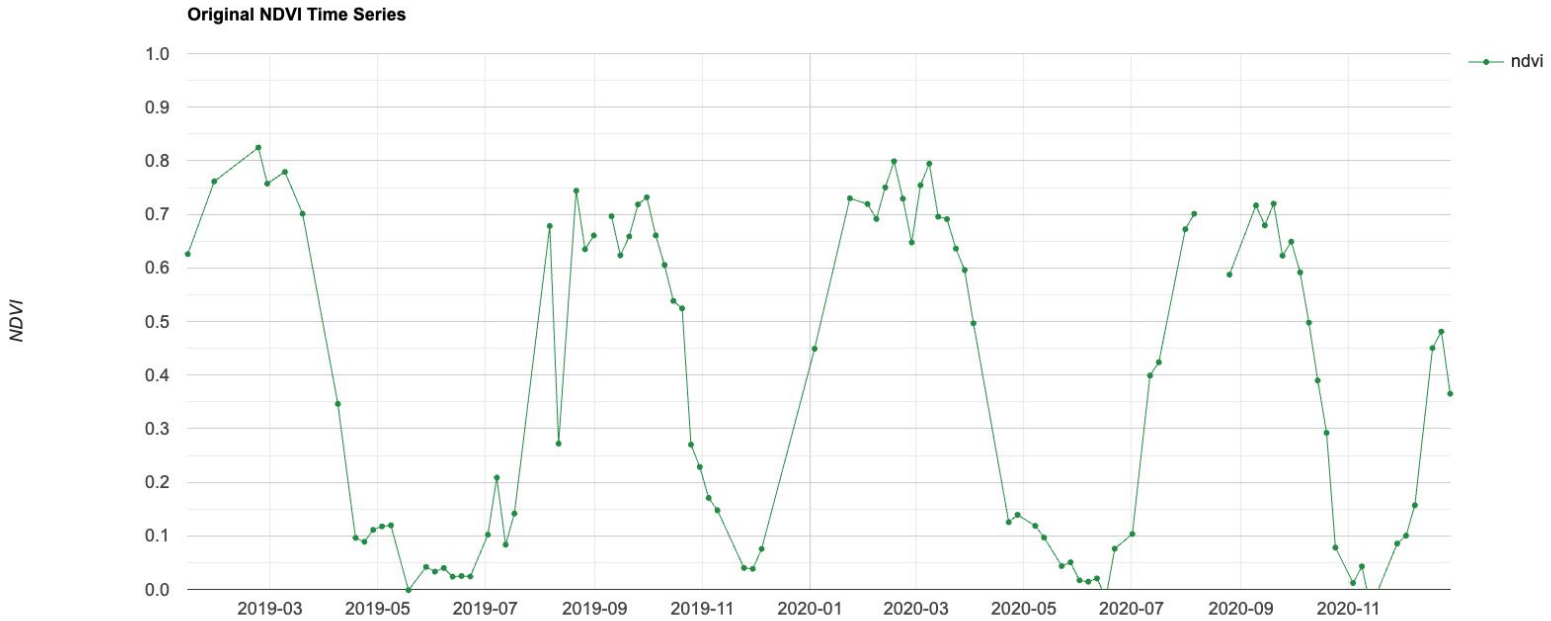
Smoothing methods can reduce noise and improve the accuracy of measured signal.[2]


Let’s construct a Time-Series

```
var s2 = ee.ImageCollection('COPERNICUS/S2_HARMONIZED');
var geometry = ee.Geometry.Point([74.80368345518073, 30.391793042969]);

var startDate = ee.Date.fromYMD(2019, 1, 1);
var endDate = ee.Date.fromYMD(2021, 1, 1);


var originalCollection = s2
  .filter(ee.Filter.date(startDate, endDate))
  .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 30))
  .filter(ee.Filter.bounds(geometry))
  .map(maskS2clouds)
  .map(addNDVI);
```

https://code.earthengine.google.com/29938f5fcf023a281d71f98323cb0281

```
ImageCollection COPERNICUS/S2_HARMONIZED (97 elements)                    JSON
  type: ImageCollection
  id: COPERNICUS/S2_HARMONIZED
  version: 1664756154299240
  bands: []
  features: List (97 elements)
    0: Image (14 bands)
    1: Image (14 bands)
    2: Image (14 bands)
    3: Image (14 bands)
    4: Image (14 bands)
    5: Image (14 bands)
    6: Image (14 bands)
    7: Image (14 bands)
      type: Image
      bands: List (14 elements)
        0: "B1", float ∈ [0, 6.553500175476074], EPSG:32643, 1830x1830 px
        1: "B2", float ∈ [0, 6.553500175476074], EPSG:32643, 10980x10980 px
        2: "B3", float ∈ [0, 6.553500175476074], EPSG:32643, 10980x10980 px
        3: "B4", float ∈ [0, 6.553500175476074], EPSG:32643, 10980x10980 px
        4: "B5", float ∈ [0, 6.553500175476074], EPSG:32643, 5490x5490 px
        5: "B6", float ∈ [0, 6.553500175476074], EPSG:32643, 5490x5490 px
        6: "B7", float ∈ [0, 6.553500175476074], EPSG:32643, 5490x5490 px
        7: "B8", float ∈ [0, 6.553500175476074], EPSG:32643, 10980x10980 px
        8: "B8A", float ∈ [0, 6.553500175476074], EPSG:32643, 5490x5490 px
        9: "B9", float ∈ [0, 6.553500175476074], EPSG:32643, 1830x1830 px
        10: "B10", float ∈ [0, 6.553500175476074], EPSG:32643, 1830x1830 px
        11: "B11", float ∈ [0, 6.553500175476074], EPSG:32643, 5490x5490 px
        12: "B12", float ∈ [0, 6.553500175476074], EPSG:32643, 5490x5490 px
        13: "ndvi", float ∈ [-1, 1], EPSG:32643, 10980x10980 px
      properties: Object (2 properties)
    8: Image (14 bands)
    9: Image (14 bands)
    10: Image (14 bands)
    11: Image (14 bands)
```


Original NDVI Time Series
NDVI
ndvi
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11

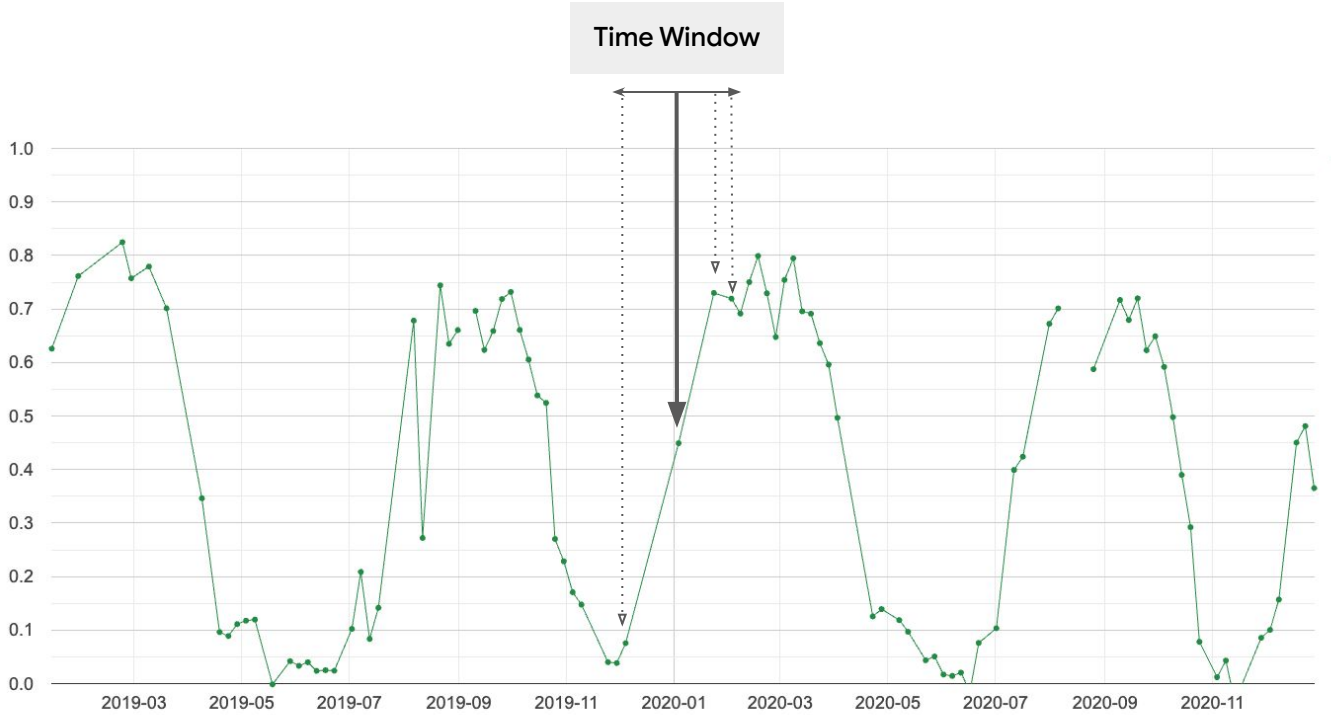

Original NDVI Time-Series

Moving Window Smoothing

Moving-Window Smoothing

To remove the fine-grained variation between each time-step, we can apply a moving-window smoothing.

1. Select a time-window
2. Find all images within the time-window for each image
3. Calculate the mean of all images for each time-step


Time Window
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11

Let’s learn about Joins

Joins in Earth Engine

Joins allow you to combine different collections based on one or more conditions.

- **ImageCollection to ImageCollection**
  - Useful in Data fusion (find matching images from different datasets)
- **FeatureCollection to ImageCollection**
  - Useful in Data extraction (extract images at multiple locations)
- **FeatureCollection to FeatureCollection**
  - Useful in Geoprocessing (spatial joins, count points in polygons etc.)

Using Joins

Select the type of join

- **Get matching items** ee.Join.saveFirst(), ee.Join.saveBest(), ee.Join.saveAll()
- **Retain matching/non-matching items** ee.Join.simple(), ee.Join.inverted()
- **Enumerate matching items** ee.Join.inner()

Select one or more binary filters

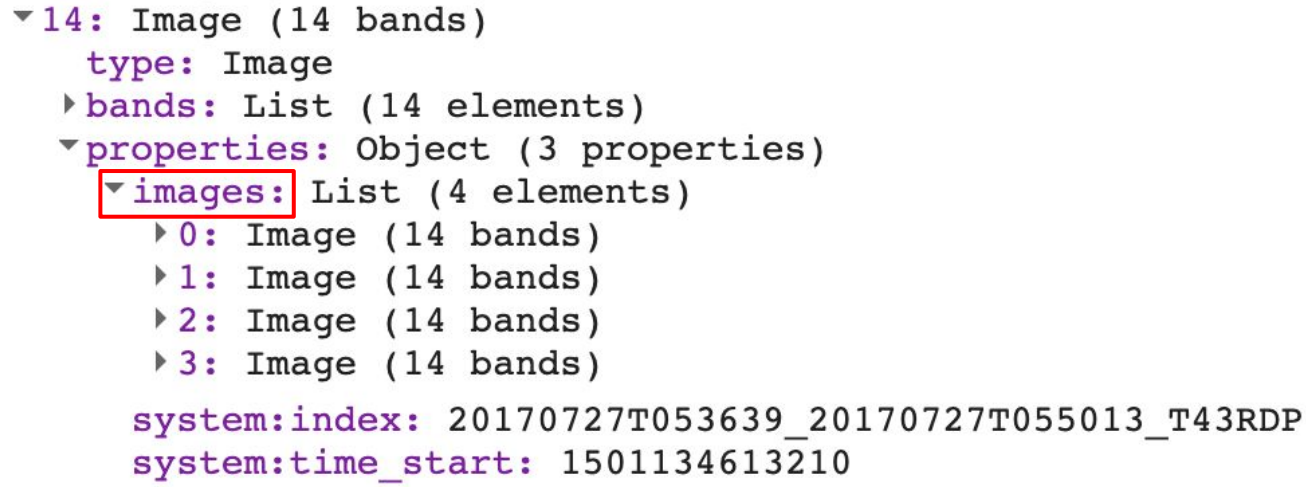
- **Filter for items within a time period** ee.Filter.maxDifference()
- **Filter for items having the same property** ee.Filter.equals()
- **Filter for items by comparing properties** ee.Filter.greaterThan(), ...
- **Filter for items by geometry** ee.Filter.withinDistance(), ee.Filter.intersects(), ...

Apply the join

- .apply(primaryCollection, secondaryCollection, filter)

```
// Specify the time-window
var days = 15;

// Convert to milliseconds
var millis = ee.Number(days).multiply(1000*60*60*24)
```

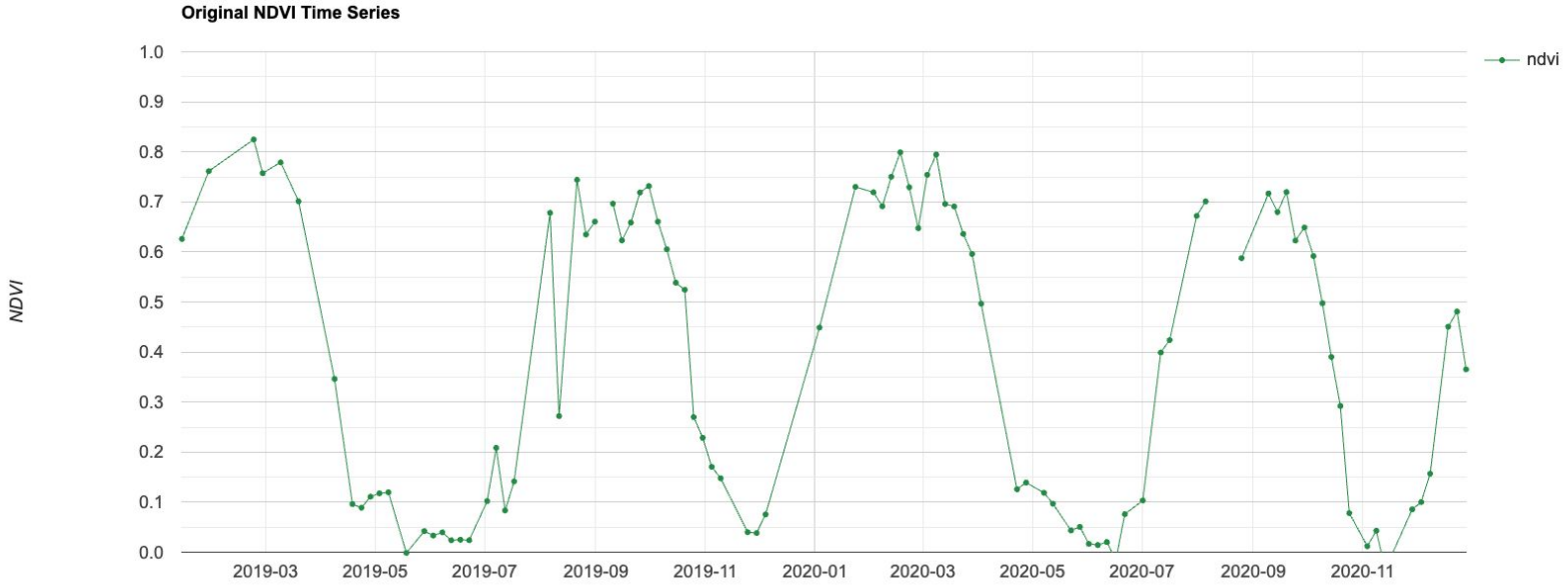
```
var join = ee.Join.saveAll({
  matchesKey: 'images'
});



var diffFilter = ee.Filter.maxDifference({
  difference: millis,
  leftField: 'system:time_start',
  rightField: 'system:time_start'
});



var joinedCollection = join.apply({
  primary: originalCollection,
  secondary: originalCollection,
  condition: diffFilter
});
```

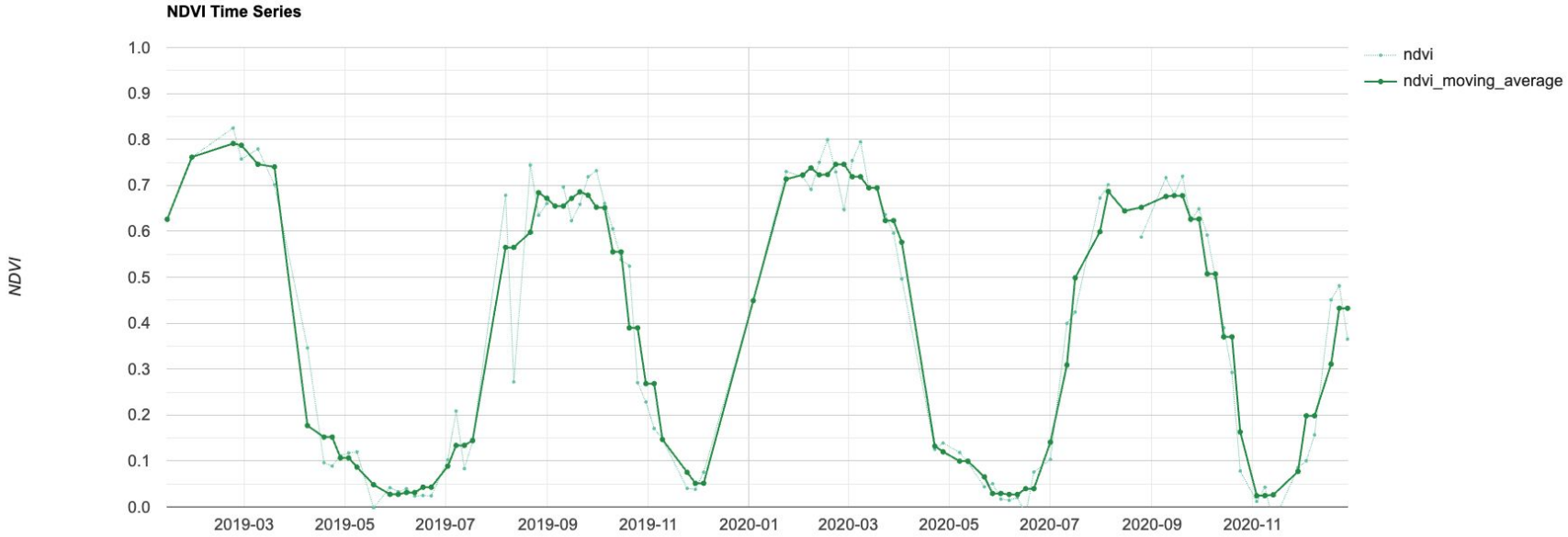

14: Image (14 bands)
type: Image
bands: List (14 elements)
properties: Object (3 properties)
images: List (4 elements)
0: Image (14 bands)
1: Image (14 bands)
2: Image (14 bands)
3: Image (14 bands)
system:index: 20170727T053639_20170727T055013_T43RDP
system:time_start: 1501134613210

```
var extractAndComputeMean = function(image) {
  var matchingImages = ee.ImageCollection.fromImages(image.get('images'));
  var meanImage = matchingImages.reduce(ee.Reducer.mean())
  return ee.Image(image).addBands(meanImage)
}


var smoothedCollection = joinedCollection.map(extractAndComputeMean);
```

https://code.earthengine.google.com/9477d1c8738d9a879e9f0d86c5e11cb0

```
▼ 10: Image (28 bands)
    type: Image
  ▼ bands: List (28 elements)
    ▸ 0: "B1", float ∈ [0, 6.553500175476074], EPSG:32643, 1830…
    ▸ 1: "B2", float ∈ [0, 6.553500175476074], EPSG:32643, 1098…
    ▸ 2: "B3", float ∈ [0, 6.553500175476074], EPSG:32643, 1098…
    ▸ 3: "B4", float ∈ [0, 6.553500175476074], EPSG:32643, 1098…
    ▸ 4: "B5", float ∈ [0, 6.553500175476074], EPSG:32643, 5490…
    ▸ 5: "B6", float ∈ [0, 6.553500175476074], EPSG:32643, 5490…
    ▸ 6: "B7", float ∈ [0, 6.553500175476074], EPSG:32643, 5490…
    ▸ 7: "B8", float ∈ [0, 6.553500175476074], EPSG:32643, 1098…
    ▸ 8: "B8A", float ∈ [0, 6.553500175476074], EPSG:32643, 549…
    ▸ 9: "B9", float ∈ [0, 6.553500175476074], EPSG:32643, 1830…
    ▸ 10: "B10", float ∈ [0, 6.553500175476074], EPSG:32643, 18…
    ▸ 11: "B11", float ∈ [0, 6.553500175476074], EPSG:32643, 54…
    ▸ 12: "B12", float ∈ [0, 6.553500175476074], EPSG:32643, 54…
    ▸ 13: "ndvi", float ∈ [-1, 1], EPSG:32643, 10980x10980 px
    ▸ 14: "B1_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 15: "B2_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 16: "B3_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 17: "B4_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 18: "B5_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 19: "B6_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 20: "B7_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 21: "B8_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 22: "B8A_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 23: "B9_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 24: "B10_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 25: "B11_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 26: "B12_mean", float ∈ [0, 6.553500175476074], EPSG:4326
    ▸ 27: "ndvi_mean", float ∈ [-1, 1], EPSG:4326
```


Original NDVI Time Series
NDVI
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
ndvi


NDVI Time Series
NDVI
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
ndvi
ndvi_moving_average


Original
Moving-Average Smoothed

Slides https://bit.ly/g4g22-time-series

Live Demo

https://code.earthengine.google.com/9477d1c8738d9a879e9f0d86c5e11cb0

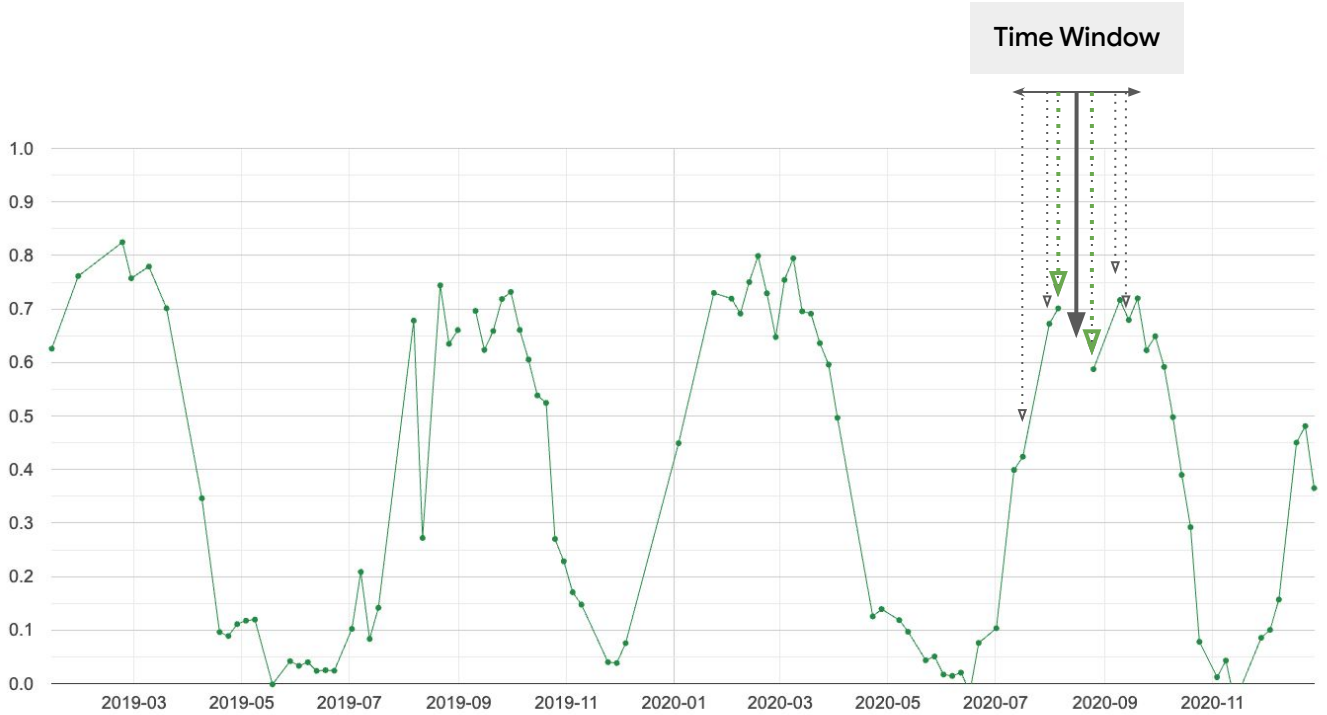
Time-Series Gap-Filling

Time-Series Gap-filling

Gap-filling is an interpolation technique to replace missing observations with the best-estimated value from temporal neighbors.

Replace the cloudy/no-data pixels with linearly interpolated value from before and after images.

1. Select the time-window
2. Find all images the time-window before and after the current image.
3. Sort the after images so the earliest image is on the top and create a mosaic.
4. Sort the before images so the latest image is on the top and create a mosaic.
5. Apply time-weighted linear interpolation on the before and after images.
6. Replace the masked pixels in the current image with interpolated pixels.

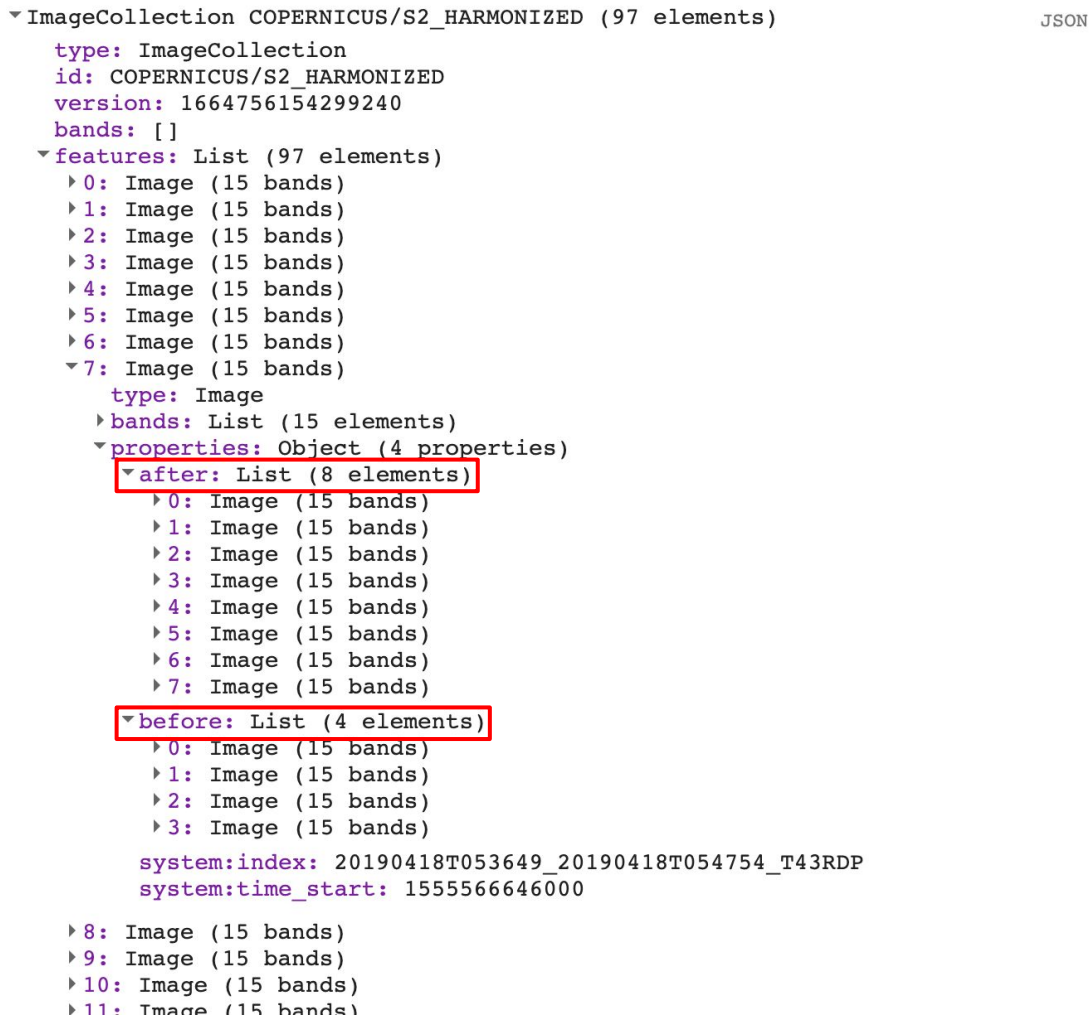

Time Window
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11

```
// Add a band containing timestamp to each image
// This will be used to do pixel-wise interpolation later


var originalCollection = originalCollection.map(function(image) {
  var timeImage = image.metadata('system:time_start').rename('timestamp')
  // The time image doesn't have a mask.
  // We set the mask from the first band of the image
  var timeImageMasked = timeImage.updateMask(image.mask().select(0))
  return image.addBands(timeImageMasked)
})
```

```
var days = 45;
var millis = ee.Number(days).multiply(1000*60*60*24)



var maxDiffFilter = ee.Filter.maxDifference({
  difference: millis,
  leftField: 'system:time_start',
  rightField: 'system:time_start'
})
```

```
var lessEqFilter = ee.Filter.lessThanOrEquals({
  leftField: 'system:time_start',
  rightField: 'system:time_start'
})

var filter1 = ee.Filter.and(maxDiffFilter, lessEqFilter)


var join1 = ee.Join.saveAll({
  matchesKey: 'after',
  ordering: 'system:time_start',
  ascending: false})


 var join1Result = join1.apply({
   primary: originalCollection,
   secondary: originalCollection,
   condition: filter1
 })
```

```
var greaterEqFilter = ee.Filter.greaterThanOrEquals({
  leftField: 'system:time_start',
  rightField: 'system:time_start'
})

var filter2 = ee.Filter.and(maxDiffFilter, greaterEqFilter)


var join2 = ee.Join.saveAll({
  matchesKey: 'before',
  ordering: 'system:time_start',
  ascending: true})


var join2Result = join2.apply({
  primary: join1Result,
  secondary: join1Result,
  condition: filter2
})


var joinedCol = join2Result;
```


ImageCollection COPERNICUS/S2_HARMONIZED (97 elements)
JSON
type: ImageCollection
id: COPERNICUS/S2_HARMONIZED
version: 1664756154299240
bands: []
features: List (97 elements)
0: Image (15 bands)
1: Image (15 bands)
2: Image (15 bands)
3: Image (15 bands)
4: Image (15 bands)
5: Image (15 bands)
6: Image (15 bands)
7: Image (15 bands)
type: Image
bands: List (15 elements)
properties: Object (4 properties)
after: List (8 elements)
0: Image (15 bands)
1: Image (15 bands)
2: Image (15 bands)
3: Image (15 bands)
4: Image (15 bands)
5: Image (15 bands)
6: Image (15 bands)
7: Image (15 bands)
before: List (4 elements)
0: Image (15 bands)
1: Image (15 bands)
2: Image (15 bands)
3: Image (15 bands)
system:index: 20190418T053649_20190418T054754_T43RDP
system:time_start: 1555566646000
8: Image (15 bands)
9: Image (15 bands)
10: Image (15 bands)
11: Image (15 bands)

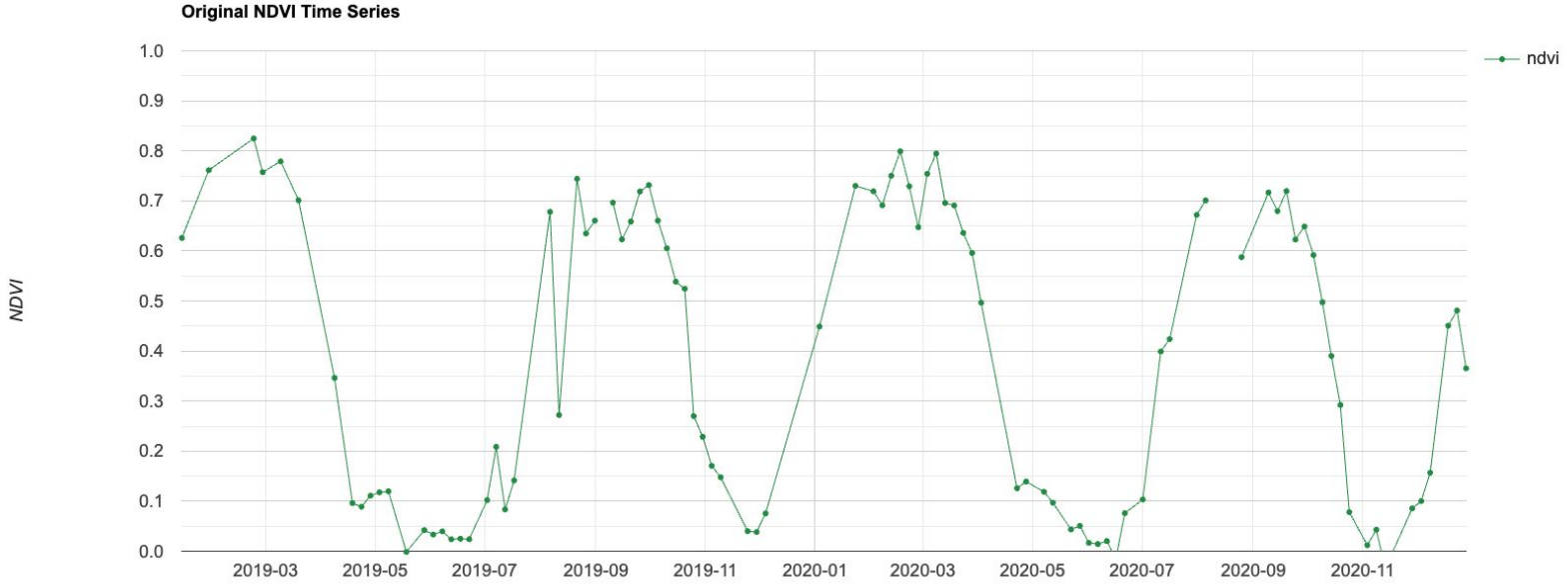
```
var interpolateImages = function(image) {
  var image = ee.Image(image)

  var beforeImages = ee.List(image.get('before'))
  var beforeMosaic = ee.ImageCollection.fromImages(beforeImages).mosaic()
  var afterImages = ee.List(image.get('after'))
  var afterMosaic = ee.ImageCollection.fromImages(afterImages).mosaic()

…
```

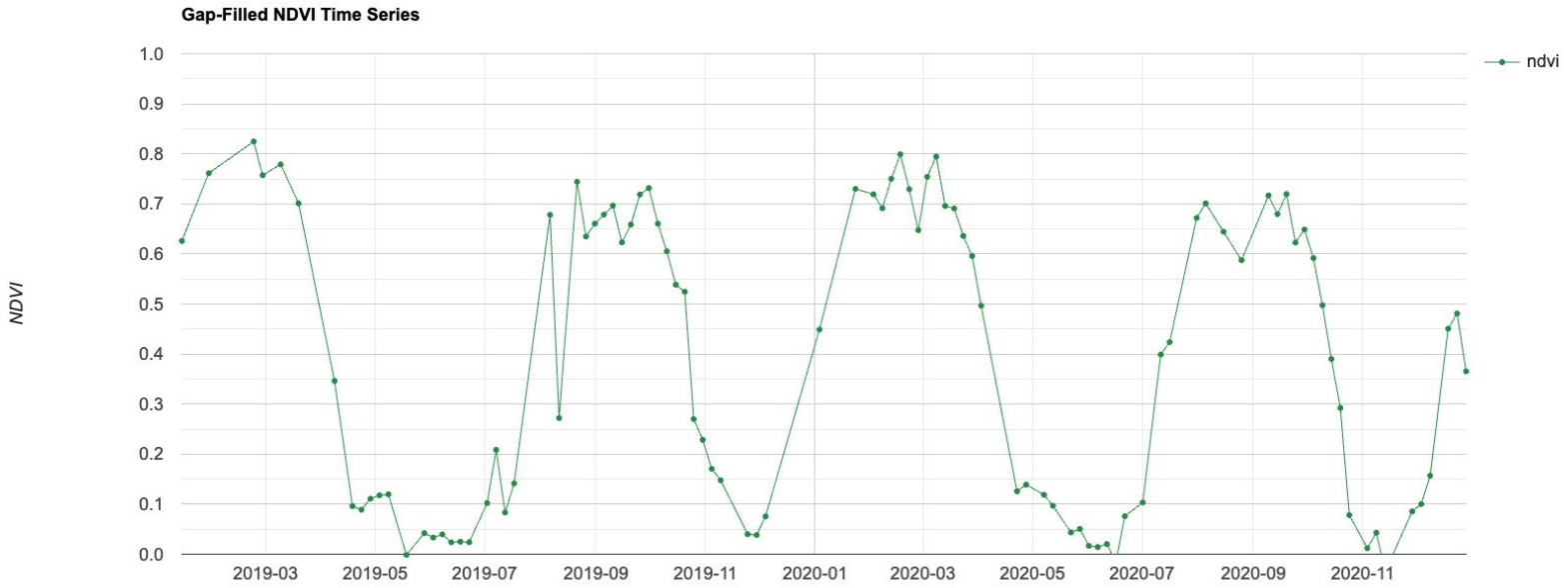
```
var interpolateImages = function(image) {
  …

  var t1 = beforeMosaic.select('timestamp').rename('t1')
  var t2 = afterMosaic.select('timestamp').rename('t2')

  var t = image.metadata('system:time_start').rename('t')

  var timeImage = ee.Image.cat([t1, t2, t])

  var timeRatio = timeImage.expression('(t - t1) / (t2 - t1)', {
    't': timeImage.select('t'),
    't1': timeImage.select('t1'),
    't2': timeImage.select('t2'),
  })

  …
}
```

```
var interpolateImages = function(image) {
  …

  var interpolated = beforeMosaic
    .add((afterMosaic.subtract(beforeMosaic).multiply(timeRatio)))

  // Replace the masked pixels in the current image
  var result = image.unmask(interpolated)

  return result.copyProperties(image, ['system:time_start'])
}
```

```
// map() the function to gap-fill all images in the collection

var gapFilledCol = ee.ImageCollection(joinedCol.map(interpolateImages))
```

https://code.earthengine.google.com/cf34b9579588dbb5b3c2cb376fa30e9b


Original NDVI Time Series
NDVI
ndvi
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11


Gap-Filled NDVI Time Series
NDVI
ndvi
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11


Original
Gap-filled

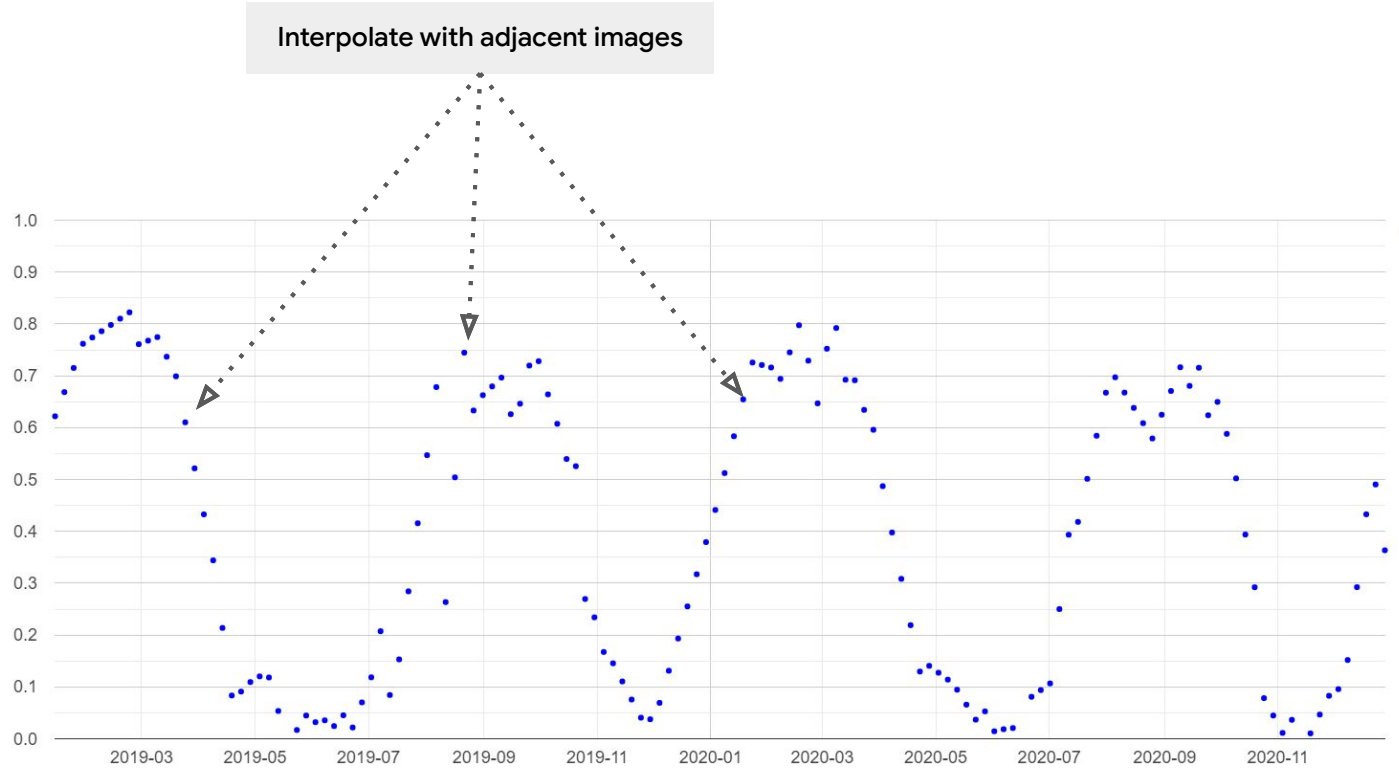
Time-Series Interpolation

Creating a Regular Time-Series

Many applications require having observations at regular time intervals.

We can take an unevenly spaced time-series and create a regularly spaced time-series using interpolation.

1. Select a time interval.
2. Generate empty images from start date to end date at the selected time intervals.
3. Gap-fill empty images using linear interpolation from adjacent images.

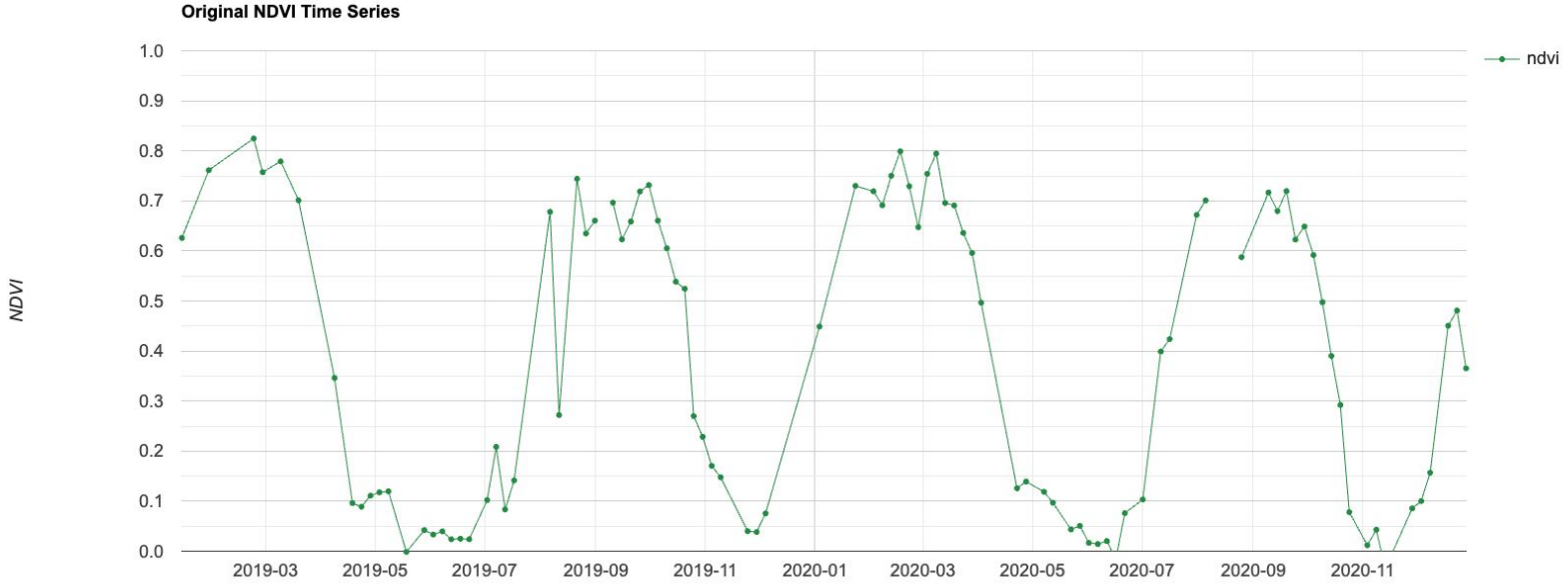

Interpolate with adjacent images
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11

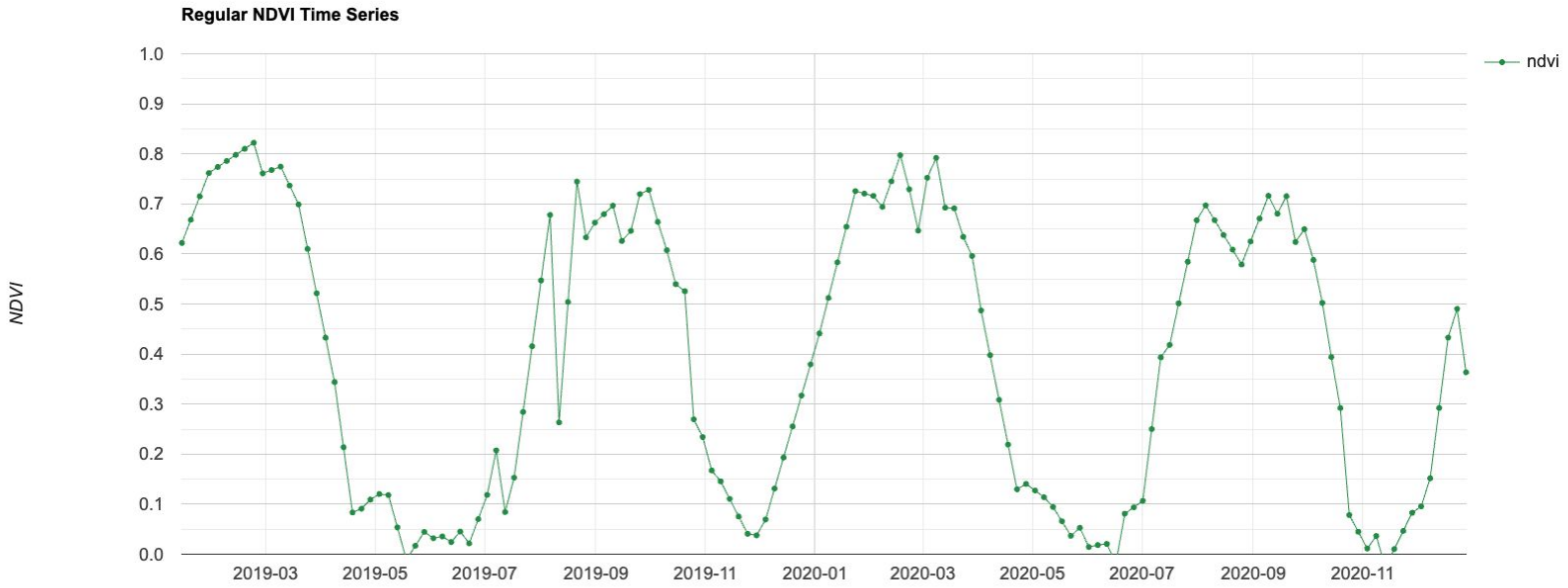
```
// Generate an empty multi-band image matching the bands
// in the original collection

var bandNames = ee.Image(originalCollection.first()).bandNames();
var numBands = bandNames.size();
var initBands = ee.List.repeat(ee.Image(), numBands);
var initImage = ee.ImageCollection(initBands).toBands().rename(bandNames);
```

```
// Select the interval. We will have 1 image every n days
var n = 5;

var totalDays = timeEnd.difference(timeStart, 'day');
var daysToInterpolate = ee.List.sequence(0, totalDays, n)

var initImages = daysToInterpolate.map(function(day) {
  var image = initImage.set({
    'system:index': ee.Number(day).format('%d'),
    'system:time_start': timeStart.advance(day, 'day').millis(),
    // Set a property so we can identify interpolated images
    'type': 'interpolated'
  })
  return image
})

var initCol = ee.ImageCollection.fromImages(initImages)


// Merge original and empty collections
var originalCollection = originalCollection.merge(initCol)
```

```
// map() the function to interpolate all images in the collection
var interpolatedCol = ee.ImageCollection(joinedCol.map(interpolateImages))


// Once the interpolation are done, remove original images
// We keep only the generated interpolated images
var regularCol = interpolatedCol.filter(ee.Filter.eq('type', 'interpolated'))
```

https://code.earthengine.google.com/fe4e9c27f262efc5bdbc7610fa03b146


Original NDVI Time Series
NDVI
ndvi
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11


Regular NDVI Time Series
NDVI
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
ndvi


Regular Time-Series

Advanced Time-Series Filtering

Geo for Good Summit 2022

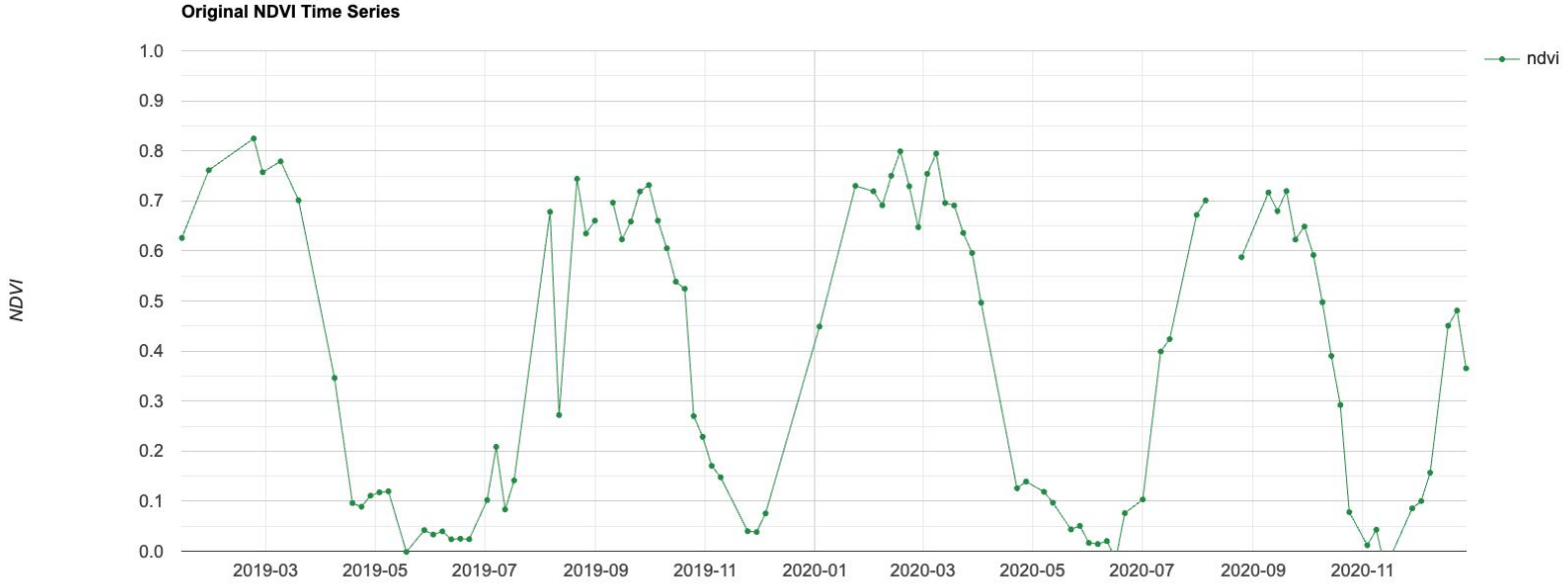
Savitzky-Golay Filter

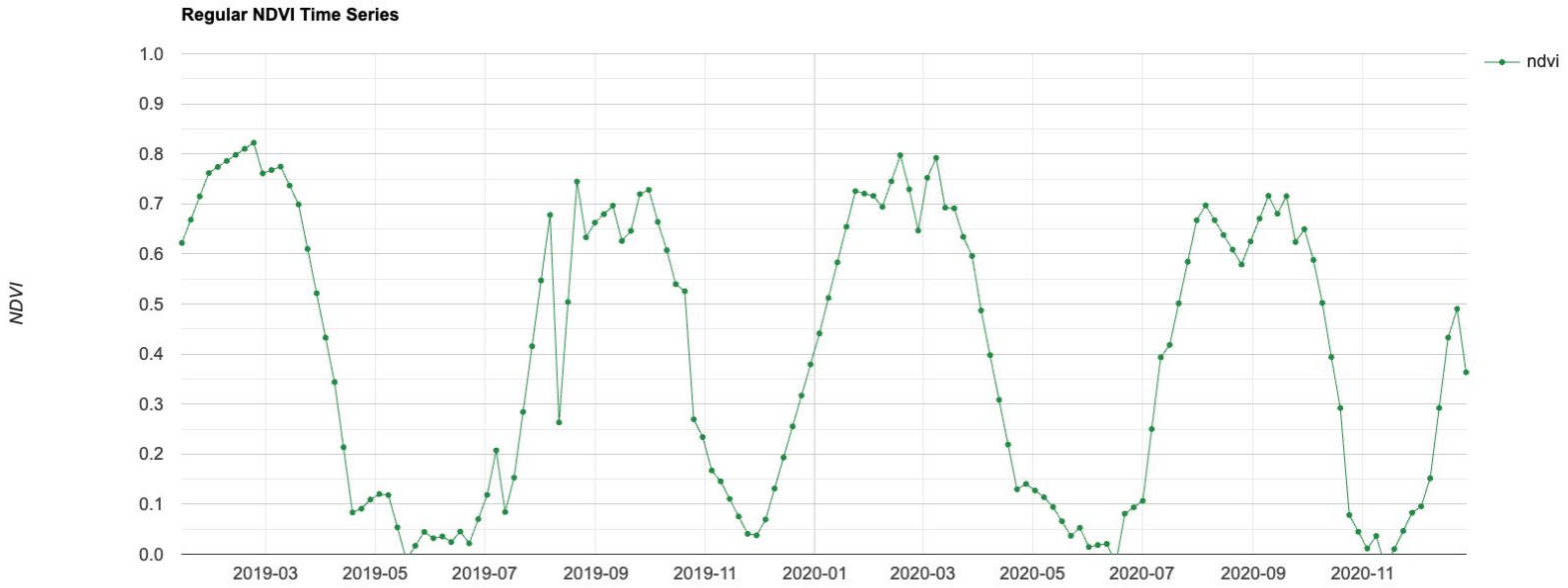
The Savitzky-Golay filter works by fitting a successive array of adjacent data points with a polynomial through linear least squares.

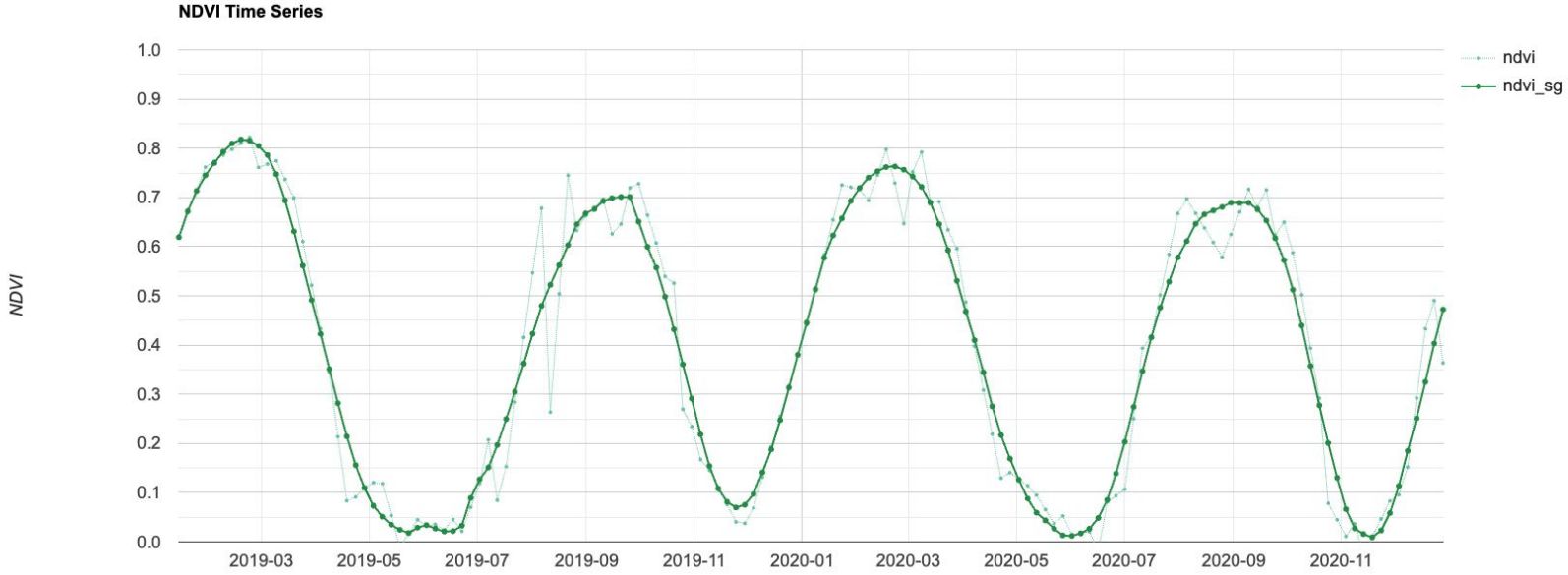
We will use the SG-Filter implementation from Open Earth Engine Library (OEEL) and apply it to filter our time-series.

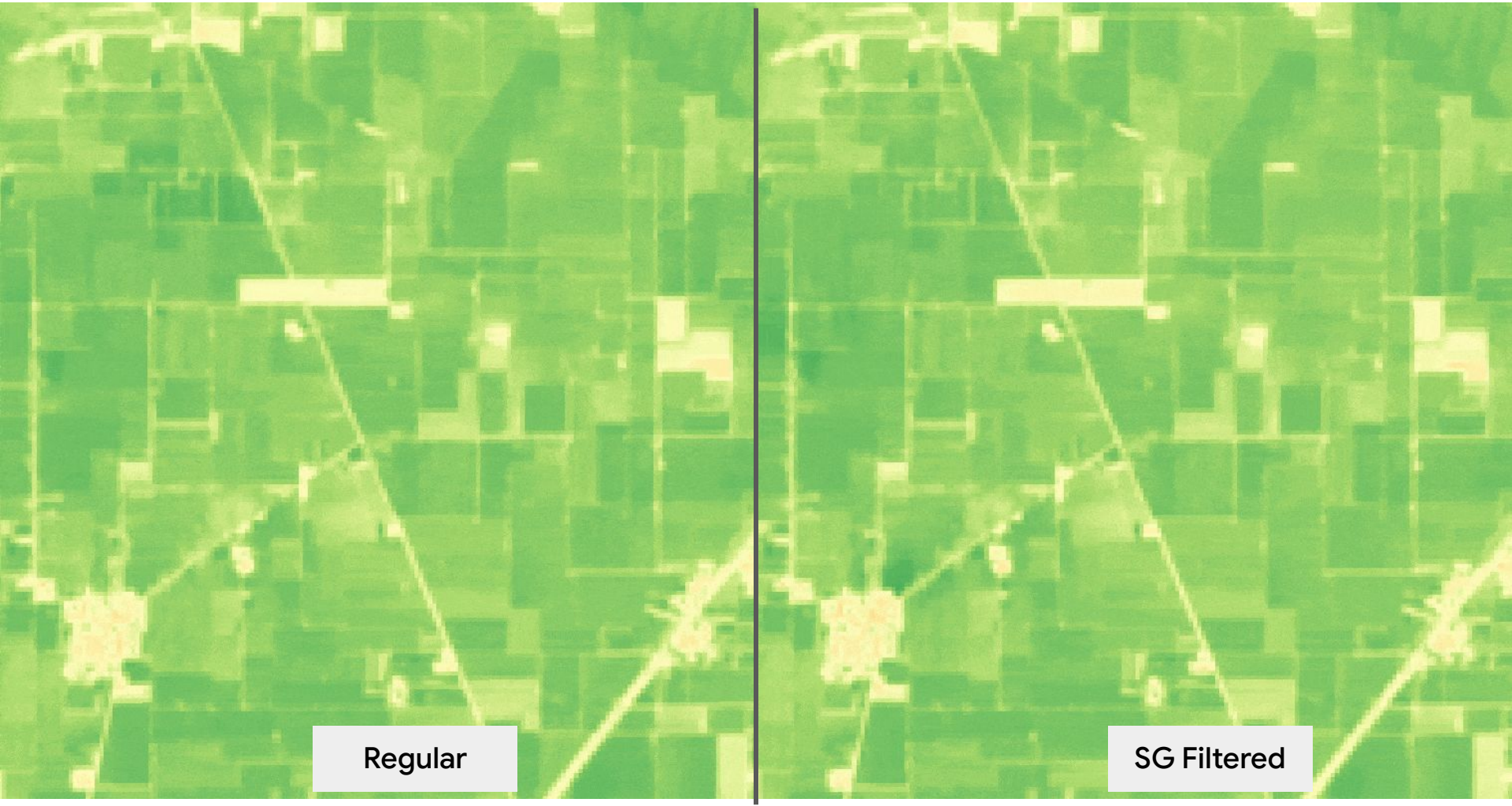
```
var oeel=require('users/OEEL/lib:loadAll');

var order = 3;

// Use the default distanceFunction
var distanceFunction = function(infromedImage, estimationImage) {
  return ee.Image.constant(
      ee.Number(infromedImage.get('system:time_start'))
      .subtract(
        ee.Number(estimationImage.get('system:time_start')))
        );
  }


var sgFilteredCol = oeel.ImageCollection.SavatskyGolayFilter(
  regularCol,
  maxDiffFilter,
  distanceFunction,
  order)
```

https://code.earthengine.google.com/ffd0b9b3c372d5e21181a6144acb0ae3


Original NDVI Time Series
NDVI
ndvi
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11


Regular NDVI Time Series
NDVI
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
ndvi


NDVI Time Series
NDVI
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
2019-03
2019-05
2019-07
2019-09
2019-11
2020-01
2020-03
2020-05
2020-07
2020-09
2020-11
ndvi
ndvi_sg


Regular
SG Filtered


SG Filtered Time-Series

Thank you!

01 Moving-Window Smoothing
Compute moving-average to remove outliers

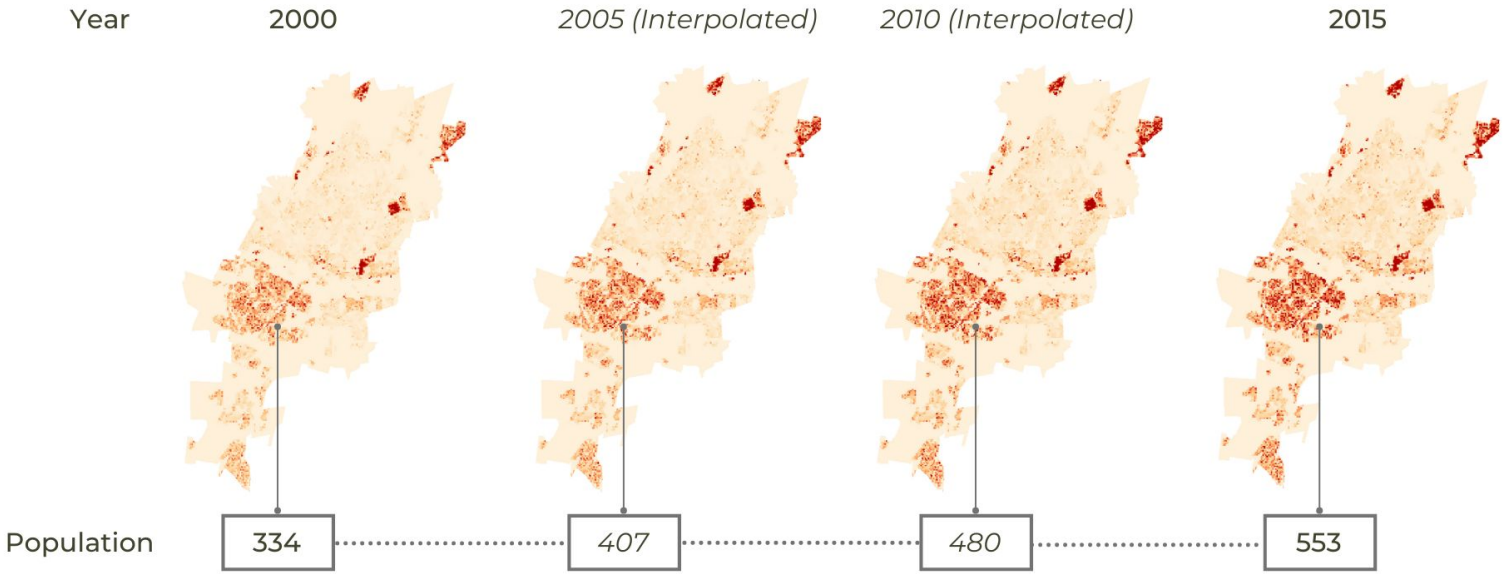
02 Time-Series Gap-Filling
Fill masked pixels from temporal neighbors

03 Time-Series Interpolation
Create a regularly spaced time-series

04 Advanced Time-Series Filtering
Apply Savitzky–Golay filter for time-series smoothing

Supplement

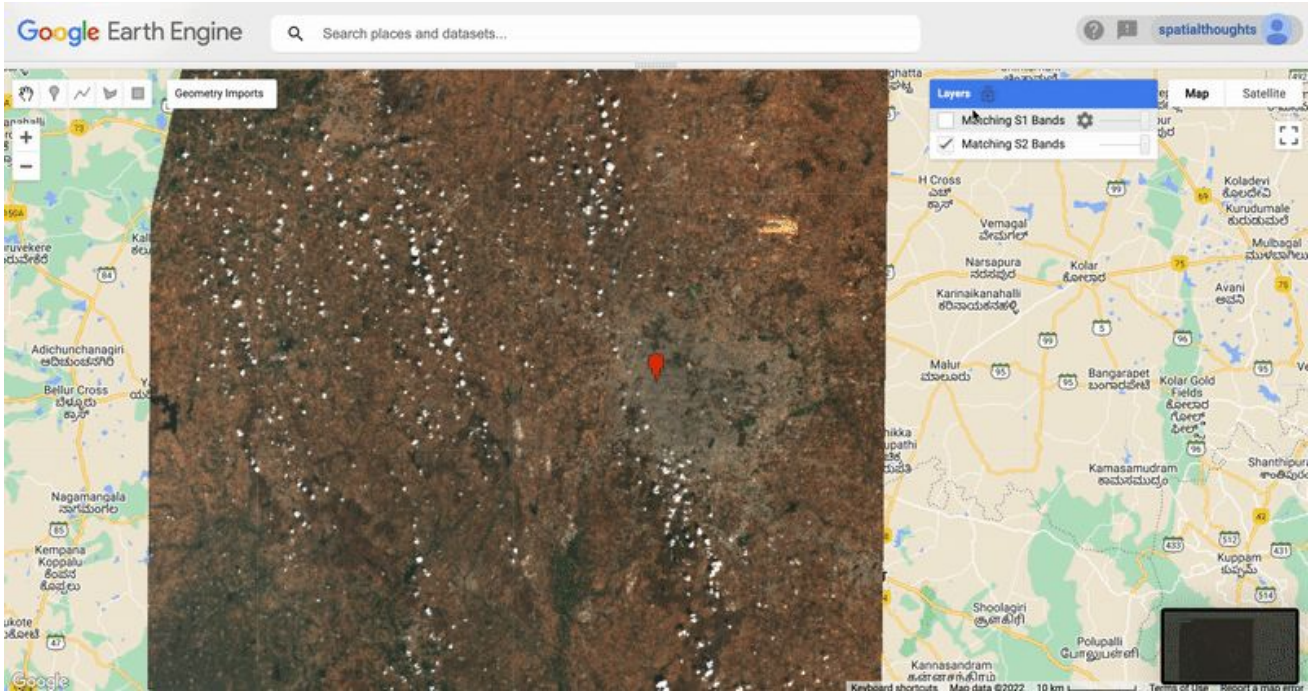
Interpolate Population Time-Series



https://code.earthengine.google.com/1ffd9f11b3ec68ed59fda1d177673c58

#GeoForGood22

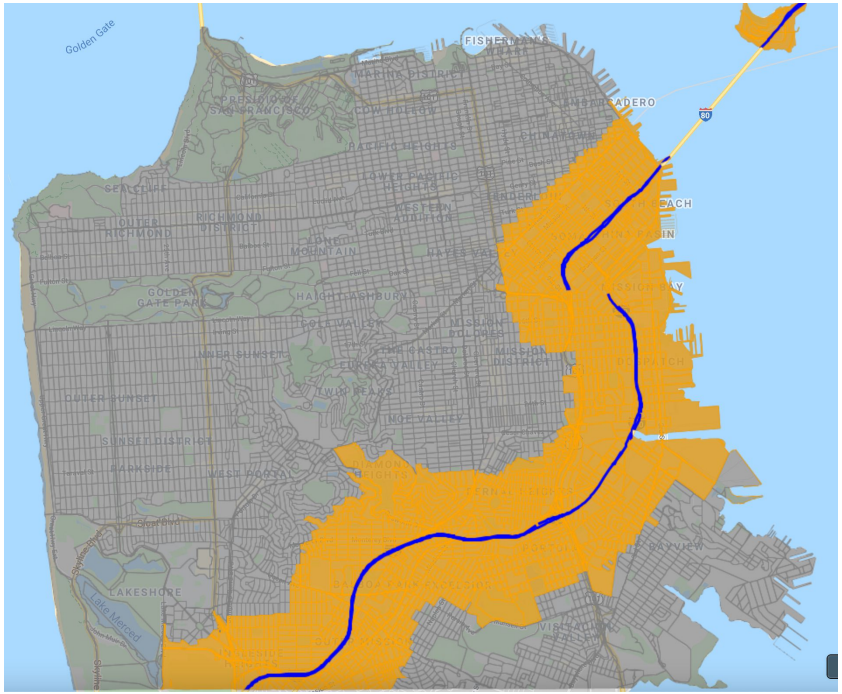
Join ImageCollection to ImageCollection (Data Fusion)



https://code.earthengine.google.com/f23f7cbca4981a5cf0f13aef2cf84954

#GeoForGood22

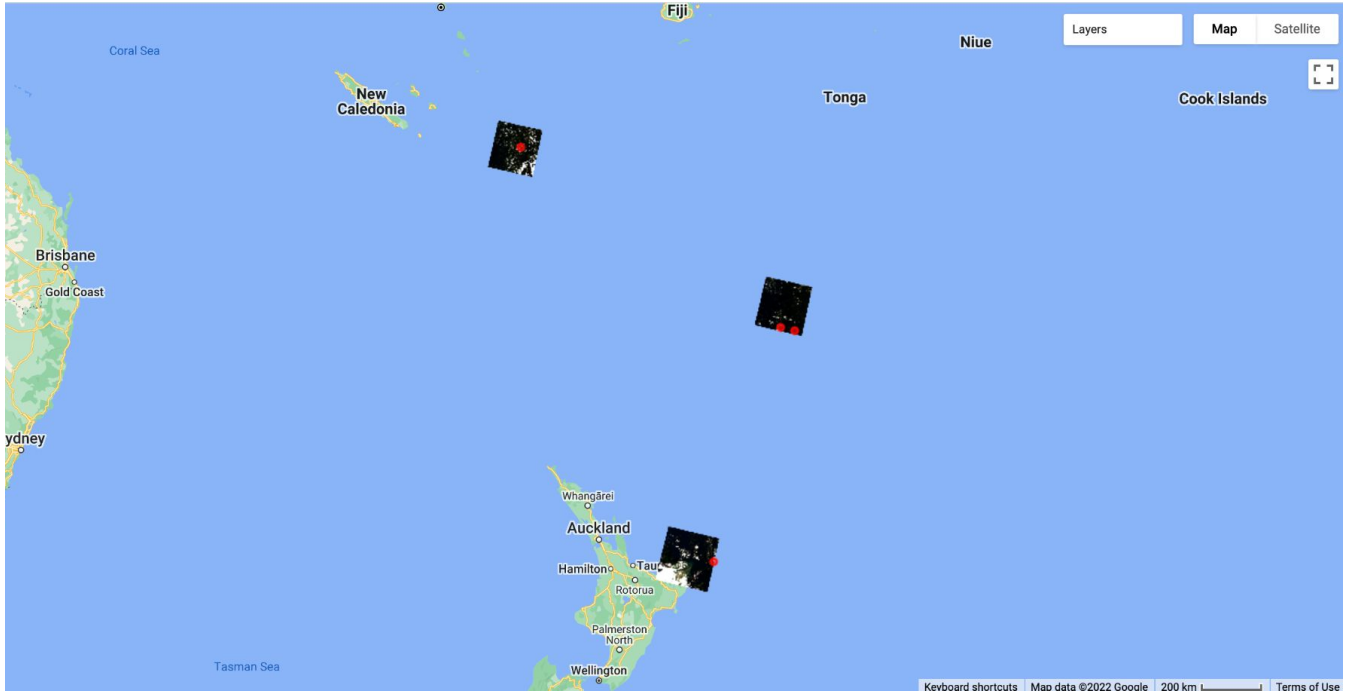
Join FeatureCollection to FeatureCollection (Spatial Join)



https://code.earthengine.google.com//77fa42ddaaf19b884e86446fcc4b89c1

#GeoForGood22

Join FeatureCollection to ImageCollection (Data Extraction)



https://code.earthengine.google.com/93dccf63a6d94327f1ac95884710cdb5

#GeoForGood22

Useful Resources

- Time Series Modeling - Earth Engine Community Tutorial
- Time Series Analysis - Earth Engine Video Tutorial
- Temporal Gap Filling with Linear Interpolation in GEE - Spatial Thoughts Tutorial
- Exporting ImageCollections
  - Python
  - Javascript
- Vector Data Visualization & Analysis - Earth Engine India Summit Presentation
- Interpolating Landcover Time-Series Data - Forward/Backward Correction Code