[CP-05-023] Google Earth Engine

Abstract

Google Earth Engine (GEE) is a cloud-based platform for planetary scale geospatial data analysis and communication. By placing more than 17 petabytes of earth science data and the tools needed to access, filter, perform, and export analyses in the same easy to use application, users are able to explore and scale up analyses in both space and time without any of the hassles traditionally encountered with big data analysis. Constant development and refinement have propelled GEE into one of the most advanced and accessible cloud-based geospatial analysis platforms available, and the near real time data ingestion and interface flexibility means users can go from observation to presentation in a single window.

Keywords: big data, cloud-based geospatial analysis, cyberGIS, cyberinfrastructure, geospatial big data, remote sensing

Author & citation

Coll, J. M. and Li, X. (2020). Google Earth Engine. The Geographic Information Science & Technology Body of Knowledge (1st Quarter 2020 Edition), John P. Wilson (ed.).
DOI: 10.22224/gistbok/2020.1.9.

Explanation

1. Google Earth Engine: A cloud-based, planetary-scale geospatial analysis platform

Google Earth Engine (GEE) is a cloud-based data and analysis platform which combines more than 17 petabytes of geospatial data, analytic APIs, and a web-based Integrated Development Environment (IDE) in one package and runs on Google's computational infrastructure, enabling interactive earth data analyses on scales not previously feasible. This platform was first introduced to the public in 2013 as a means of performing an analysis of global forest cover change (Hansen et al., 2013). In this flagship application, the authors sought to quantify the spatial distribution and global state of forest loss and gain from 2000 to 2012 by blending more than 654,000 Landsat scenes across the 12 years of the analysis — the resultant analysis of 700 Terapixels of data took more than 1 million hours of computation and exported results in a comparatively trivial 4 days across the Google compute infrastructure. Such analyses once used to sit behind such extreme barriers of entry that no one would have been able to produce, much less reproduce, these results.

This entry will cover how to access GEE, some of the most common tools and methods used to operate over data, and how to export analyses. As of the time of writing, GEE has been in beta testing for almost 5 years and is free for use in development, research, or educational purposes, but things may (and will) change. This entry will be kept as up to date as possible but monitor GEE documentation for the most up to date resources.

2. Overview of GEE

2.1 Accessing GEE

To use GEE, request a user account. Once approved, you have access to the datasets and the Google computation infrastructure. This access takes the form of a REST API. There are currently two means of doing so, one Python based and the other JavaScript based. The less popular of the two methods, the Python library, allows users to interact with Earth Engine using the Python programming language. The Google Earth Engine API guide has a full walkthrough of how to install the needed libraries. The more popular means of accessing GEE is through the JavaScript library which is accessed through a web-based IDE, more commonly called the Code Editor. Although not specifically required, it is recommended that you use Google Chrome, and as this is the most popular means of accessing GEE.

The rest of this entry will be written using the JavaScript IDE. Don't worry if you are more familiar with Python, the transition to JavaScript is relatively painless and primers are available. One of the first steps for those new to GEE is to explore the features of the JavaScript API. In the upper right-hand corner of the code editor, click on **help > Feature Tour** to take a quick guided tour of the platform.

2.2 GEE Operations

Broadly speaking there are three major types of operations one can perform in GEE: Methods, Algorithms, and Functions. The general structure of these is exemplified in Figure 1. Methods require an object to act on and take inputs (Figure 1, line 24) where we query the SRTM dataset for all values greater than 2000. Algorithms are objects themselves and take a specific input to return an object (Figure 1, line 27) where we use the Terrain algorithm to create a hillshade image from SRTM. Functions take an input and do something within the API (Figure 1, line 30) where we add the Hillshade layer to the map.

```
23  //   NewObject  =      OldObject       .Method(Inputs);
24  var StudyArea2k = ee.Image('srtm90_v4').gte(2000);
25
26  // NewObject  =    Algorithm         (Inputs)               .Method(Inputs)
27  var Hillshade = ee.Algorithms.Terrain(ee.Image('srtm90_v4')).select('hillshade');
28
29  // Function(Inputs)
30  Map.addLayer(Hillshade);
```

Figure 1. An example of the different types of operations in Google Earth Engine. Source: authors.

2.3 Data Structures / Objects

Data objects in GEE are slightly different from more traditional desktop GIS software, but the concepts are very similar. These differences are most pronounced with the data structures (Figure 2) . What GEE calls Images can be thought of as Rasters, and just as Rasters may have more than one band, so too may Images. What GEE refers to as an ImageCollection are simply a collection of Images. What GEE refers to as Features are analogous to features in a shapefile or geodatabase, so FeatureCollections are analogous to the larger shapefile or feature classes in a geodatabase. These data structures also borrow concepts from Object Oriented Programming (OOP) in that Feature can be thought of as the parent of Image and likewise ImageCollections are children of FeatureCollections. Therefore, many of the methods and operations available for Features and FeatureCollections are inherited by Images and ImageCollections.

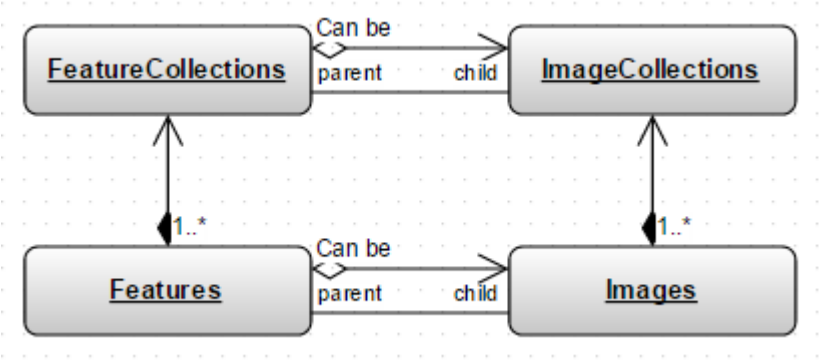

Figure 2. Data structures in Google Earth Engine. Source: authors.

2.4 Asset Management

GEE hosts more than 17 petabytes of publicly available data which can be redistributed. However, users also have the option to upload their own data in popular GeoTIFF or shapefile formats for ingestion into the system as user assets. Assets are limited to 10 GB in size, and are counted as part of your shared Google storage quota (spread across Gmail, Google Drive, etc.). Once in the platform, they are treated the same as any other dataset in the platform and can be shared with other users.

2.5 Data Preparation

GEE handles many of the concerns typically associated with data preparation, such as storage, cataloging, and projecting the data appropriately. GEE has ingested many of the most popular remote sensing data from various satellite platforms such as LANDSAT, MODIS, and Sentinel, to name just a few. The full collection can be browsed through a catalog. When ingested, the data is stored in its native projection and format to preserve data integrity, and the metadata necessary to effectively use them is also included in each image as properties. Users may inspect, project, and resample the data as necessary, but because the GEE team takes care of these details when the data is ingested, the “time to

science" is rapidly accelerated and users are free to spend that time on the analysis and presentation instead of the relatively canned portion of data preprocessing. By using the Search Box at the top of the IDE, you can find relevant datasets and explore how they can be used in GEE.

2.6 Mappers and Reducers

When working with data in GEE, there are two types of operations we can perform over the data: mapping and reducing. Mapping applies a function to each image in a collection, so a stack of n elements results in a transformed stack of n elements. This is useful when we need to do something to every image (for instance, calculating NDVI over a timeseries of images). Reducers on the other hand take one or more input features and return a reducer number of outputs. These often take the form of conventional map algebra expressions. These expressions can be as conceptually simple as pulling the maximum value of a stack of pixels in an ImageCollection to as complex as non-parametric trends of slope or regional reducers.

2.7 Joins

Joining data is a hallmark of geospatial analysis, and GEE provides several ways in which different datasets may be joined together based on a specified condition. These conditions, or filters as they are called in GEE, can be spatial, tabular or temporal in nature. GEE applies the conditional filter, and items in the input collections that match the conditions are saved in the output collection depending what output was specified.

2.8 Image Classification

GEE has several image classification methods built into the API, including unsupervised and supervised classifiers. As of the time of this writing, these include: Cart, Naïve Bayes and Continuous Naïve Bayes, Decision Tree, Gmo Linear Regression and Max Ent, Ikpamir, Minimum Distance, Pegasos linear, and Polynomial, and Gaussian, Perceptron, Random Forest, Spectral Region, Scm, and Winnow. Additionally, if provided with training data, a confusion matrix can be calculated from the classification to present an accuracy assessment.

2.9 Exporting analyses

Although it is often more efficient to perform a desired analysis completely within GEE, exporting results for further specialized analysis or publication can take place in several forms depending on the desired purposes, such as CSV files of time series, images, web map tiles, or videos. In addition, a suite of standard JavaScript User Interface (UI) items are available within the IDE which enables users to rapidly design and prototype user interfaces to include items such as combo and check boxes, sliders, and legend elements. These powerful features allow users to publish advanced and fully fleshed out web applications so that end users without GEE account can interact with or export analysis without looking at a single line of code.

3. Example Analysis

To tie these concepts together, this section will walk through a demonstration of many of the above aspects of GEE as they are used in a single application: quantifying the global trend of snow cover frequency.

3.1 Background and Outline

Snow cover frequency, or the number of days that a spot is covered by snow divided by the number of days it has a valid observation, is an important metric of the cryosphere, and is critical to a number of biogeophysical processes. Using the daily MODIS snow cover dataset, we will calculate the snow cover frequency on a yearly basis for water years 2003 through 2018. A water year here is defined as October 1st to September 30th and is labeled as the year of the end date. For example, water year 2004 is from Oct. 1, 2003 to Sept. 30, 2004. We will then find and map the linear trend of snow cover frequency at every pixel across the globe. To add a little more nuance on this analysis, we will take into account the sensor zenith angle of each MODIS observation. At the extremes of the MODIS instruments sensor zenith, the pixel length has essentially doubled, and the width has increased more than 10 times that of the nidar pixel. These observations can skew aerial coverages, so we want to limit our daily snow cover dataset to just those observations that have a sensor zenith angle less than 25o, limiting pixels to 110% of the nominal area of a nadir pixel. To do this we need two daily datasets, the daily snow cover product MOD10A1 and the sensor properties contained in the MOD09GA dataset, both are available in GEE.

To accomplish this analysis, we will first join the two datasets (MOD10A1 and MOD09GA) to create a single, combined dataset. We then mask out those pixels with a high sensor zenith angle. After that, we need to reclassify the snow cover to a simple snow/no snow/missing dataset, which will make future calculations easier. After reclassifying the data, we want to count at each pixel the number of days that have a snow observation and the number of days that have a valid observation. We will then need to add a band representing the water year (time) and calculate the trend at each pixel before finally exporting our results. We will wrap much of that data up into functions, as outlined below, in the larger ovals. The steps of this process are exemplified in Figure 3.

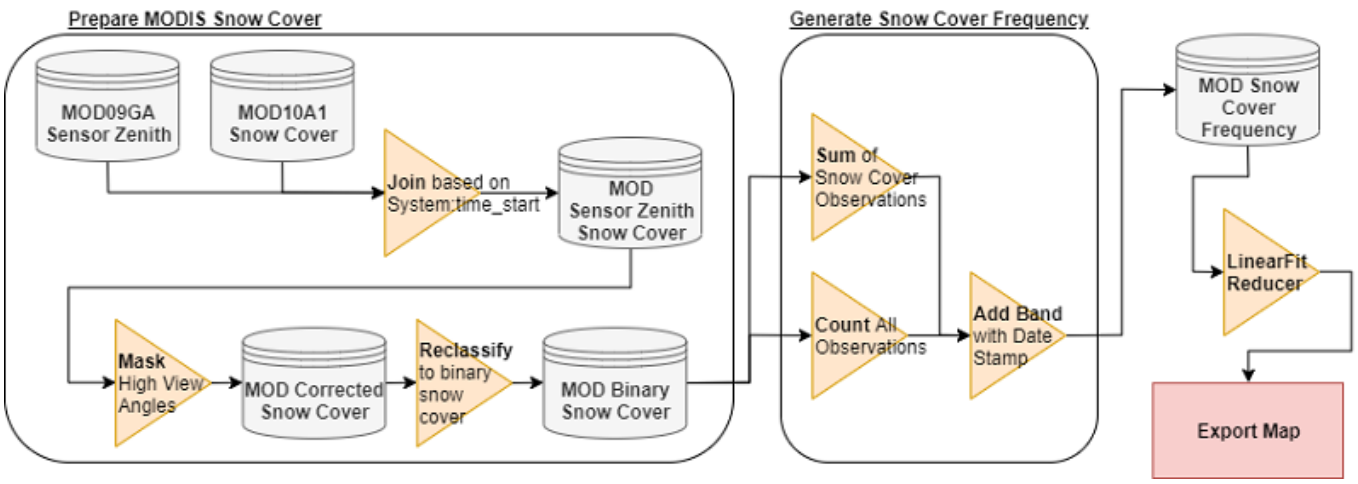

Figure 3. Overview of the workflow that will be followed in this example of snow cover analysis. Source: authors.

3.2 Creating ImageCollections

We will start by accessing the MOD10A1 dataset and examine its structure (Figure 4).

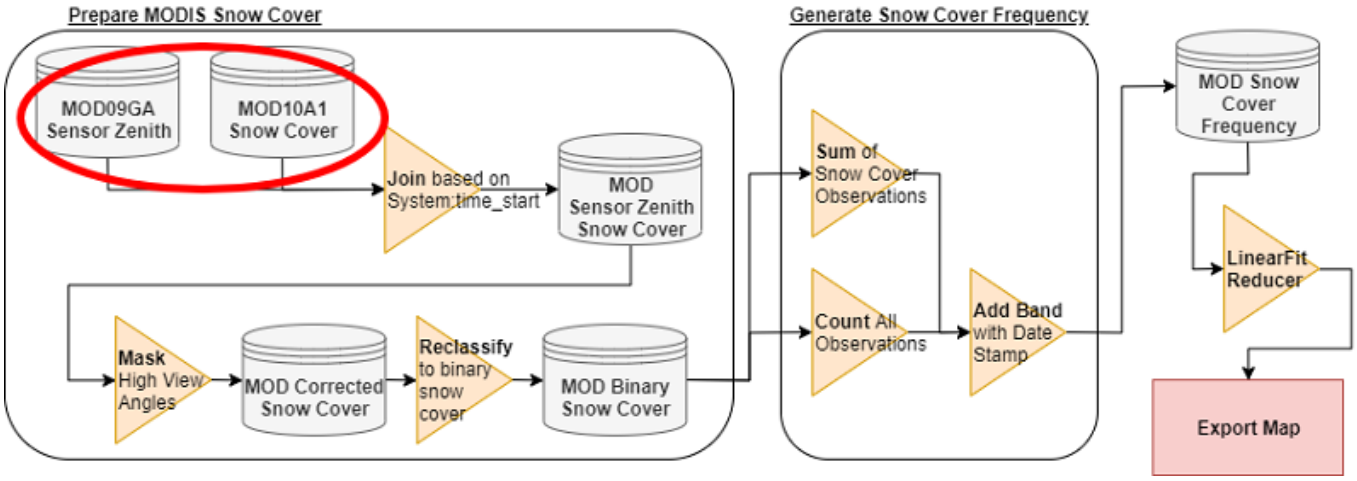

Figure 4. Beginning the GEE workflow by exploring the means of accessing data and their structure. Source: author.

Searching for datasets takes place at the top of the Code Editor in the search bar, and we can use associated GEE data IDs to access selections. For this example, we start by searching for MOD10A1, and importing it, as shown in Figure 5. Using just over 40 characters, we have access to the entire MOD10A1 dataset in less than 20 seconds. These static variables can also be included as Imports, which sit above the code editor as Imports.

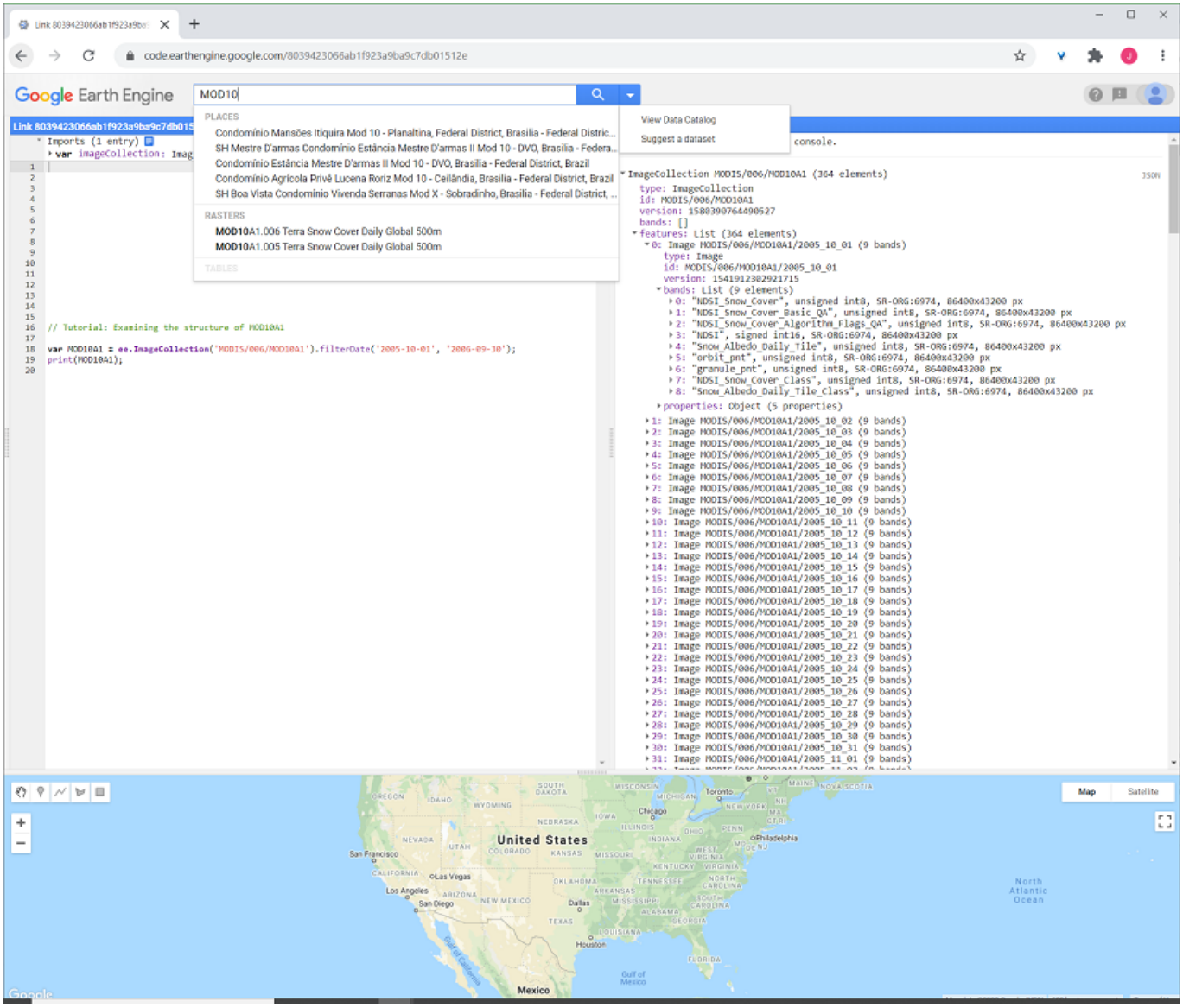

Figure 5. The code Editor IDE and how to import an ImageCollection into the API for analysis. The code to create this can be found at the Google Earth Engine Code Editor site, available at this link for users once they have a GEE account. Source: authors.

MOD10A1 is now imported into the code Editor as an ImageCollection. If you recall, the MOD10A1.006 data product is a daily global snow cover dataset and each Image in the ImageCollection has 9 bands. The one we are interested is the “NDSI_Snow_Cover” band which is a fractional snow coverage band with valid values from 0-100. Although the quality band has a high sensor zenith flag, the full sensor zenith angle data is only found in the MOD09GA.006 dataset. Therefore, our next goal is to join the two datasets together.

3.3 Joining the Collections

After we import the MOD09GA dataset as another ImageCollection, our goal is to get the sensor zenith angle band from the MOD09GA ImageCollection to join with our snow cover data (Figure 6).

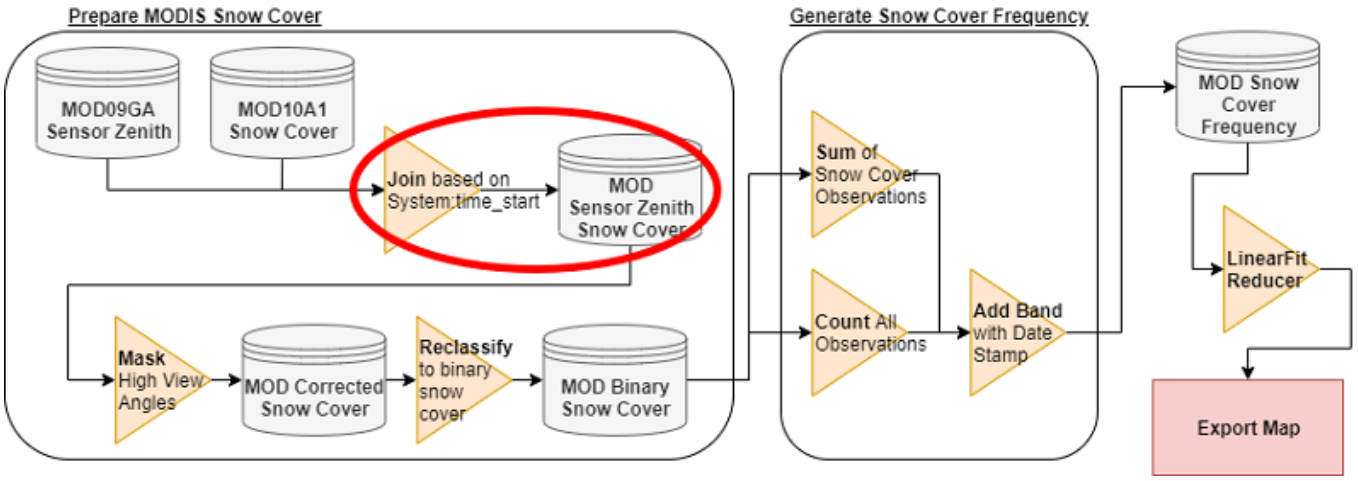

Figure 6. Next step in the workflow is to join two data sets. Source: author.

To select a particular band to work with, we can use the .select() method (Figure 7).

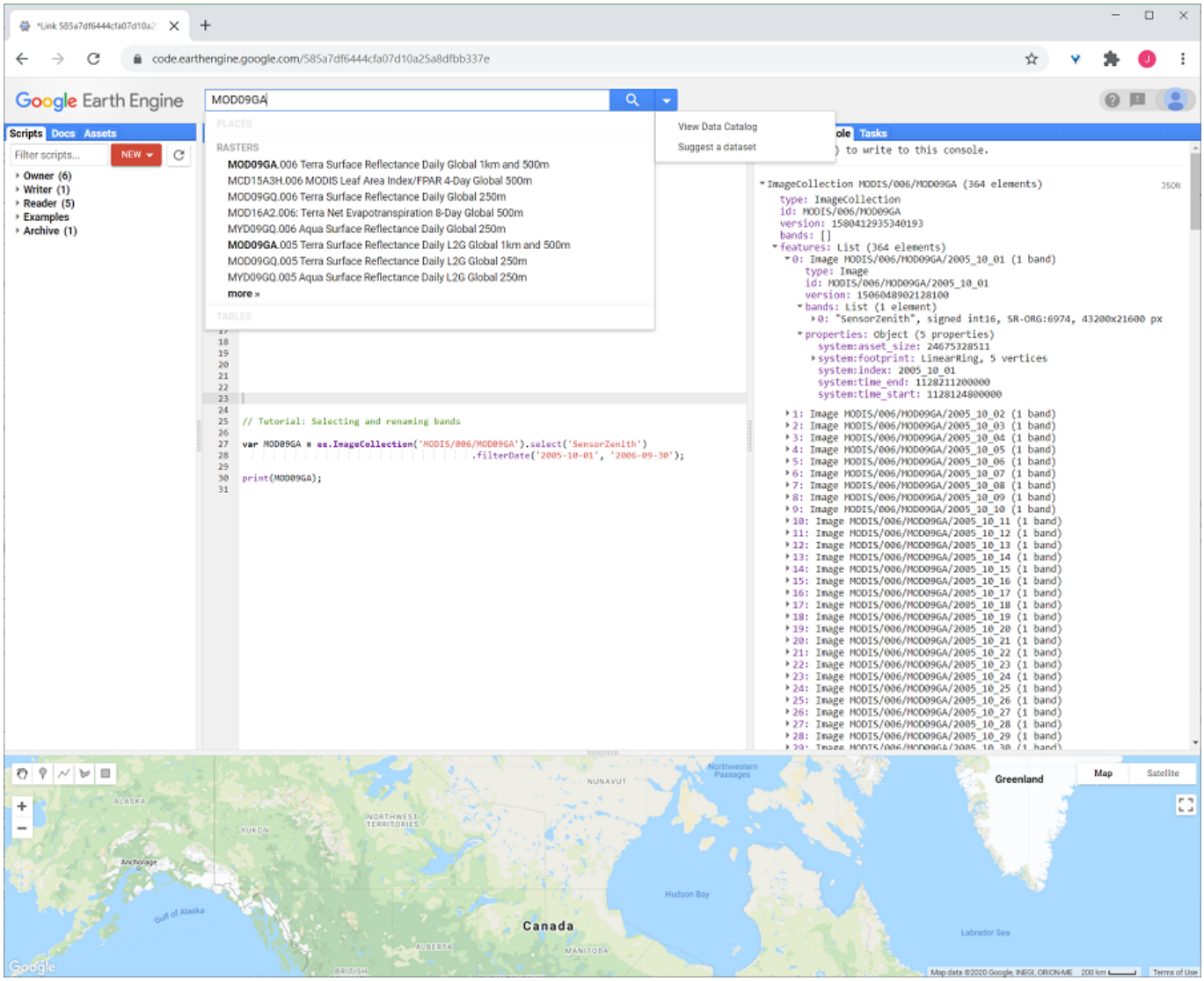

Figure Figure 7. Figure 7. Example of using the .select() and .filterDate() methods in GEE to

pull images we need and examine their structure. The code to create this can be found at the Google Earth Engine Code Editor site, available at this link for users once they have a GEE account. Source: authors.

Joining two collections requires a common property or key item between them, just as it does in standard GIS software. In this case, basing the join on their acquisition times is a good choice, using the "system:Time_Start" property as the filter. If you look at the various types of joins in the documentation, you will see that there are a few ways to do this but we will use the join.inner(). The code to join the two datasets together is shown in Figure 8.

```
// Tutorial: Joining two collections

// First, we define the two datasets, selecting the bands we are interested in
var MOD09GA = ee.ImageCollection('MODIS/006/MOD09GA')
                .select('SensorZenith')
                .filterDate('2005-10-01', '2005-10-31');
var MOD10A1 = ee.ImageCollection('MODIS/006/MOD10A1')
                .select('NDSI_Snow_Cover')
                .filterDate('2005-10-01', '2005-10-31');

// Define the join and filter
var Join = ee.Join.inner();
var FilterOnStartTime = ee.Filter.equals({'leftField': 'system:time_start',
                                          'rightField': 'system:time_start'
                                         });

// Join the two collections, passing entries through the filter
var JoinedMODIS = Join.apply(MOD09GA, MOD10A1, FilterOnStartTime);

print(JoinedMODIS);
```

Figure 8. The code used to join the two MODIS datasets together for our example. This code can be found at the Google Earth Engine Code Editor site, available at this link for users once they have a GEE account. Source: authors.

We've just joined two MODIS collections together, and it took less than 20 lines of code to do so! However, if you print the resulting collection you will get something that looks like Figure 9.

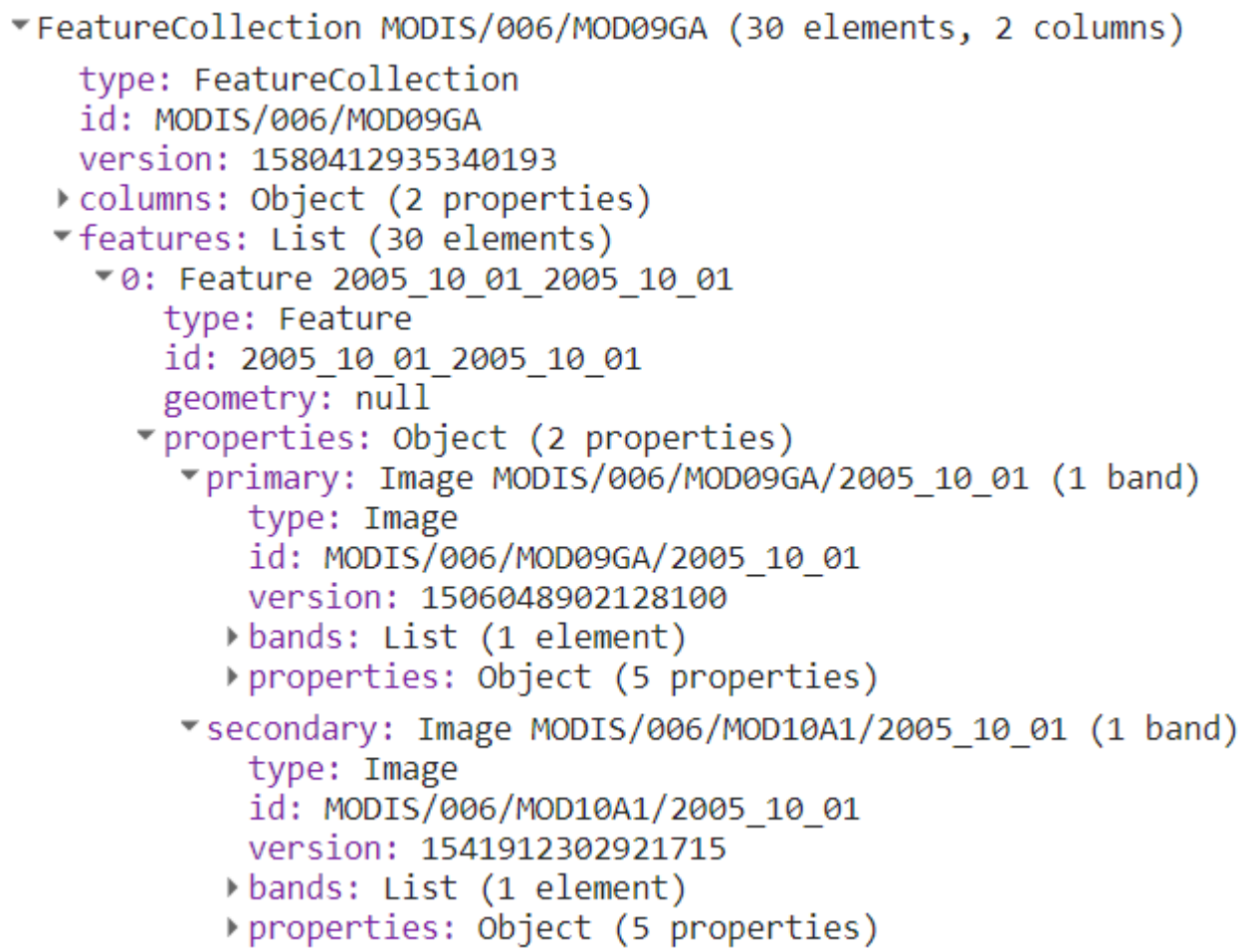

Figure 9. The results of the join performed above. Source: author.

The output collection from the join operation is a FeatureCollection, where the matching images are the "primary" and "secondary" properties of the features. In order to convert this to a format that we can work with, we need to run a function across the FeatureCollection, creating a new image that has the bands from the primary and secondary images. You can visualize the conversion as a series of operations (Figure 10).

Figure 10. A visual representation of the MergeBands function we need to write to transform the results of our join into a more usable form. Source: authors.

The code for the MergeBands function and for applying the function to each matching pair of images in the FeatureCollection are shown in Figure 11. When running the code, be sure to note the difference between the input and output collections.

```
// Tutorial: Joining two collections

// A function to merge the bands together after a join
// the bands are referred to as the 'primary' and 'secondary' properties
var MergeBands = function(aRow) {
  var anImage = ee.Image.cat(aRow.get('primary'), aRow.get('secondary'));
  return anImage;
};

// First, we define the two datasets, selecting the bands we are interested in
var MOD09GA = ee.ImageCollection('MODIS/006/MOD09GA')
                .select('SensorZenith')
                .filterDate('2005-10-01', '2005-10-31');
var MOD10A1 = ee.ImageCollection('MODIS/006/MOD10A1')
                .select('NDSI_Snow_Cover')
                .filterDate('2005-10-01', '2005-10-31');

// Define the join and filter
var Join = ee.Join.inner();
var FilterOnStartTime = ee.Filter.equals({'leftField': 'system:time_start',
                                          'rightField': 'system:time_start'
                                         });

// Join the two collections, passing entries through the filter
var JoinedMODIS = Join.apply(MOD09GA, MOD10A1, FilterOnStartTime);

print(JoinedMODIS, "Joined MODIS");

var MergedMODIS = JoinedMODIS.map(MergeBands);

print(MergedMODIS, "Joined MODIS with Merged Bands");
```

Figure 11. The code for the MergeBands function, and how to use the .map() function to each pair of matched images. This code can be found at the Google Earth Engine Code Editor site, available at this link for users once they have a GEE account. Source: authors.

3.4 Masking Out Large Zenith Angle Pixels

With snow cover and zenith angle now joined into images as bands, we will remove the pixels from each image that have a sensor zenith angle larger than 2500 (the angle is stored as a multiple of *100) (Figure 12).

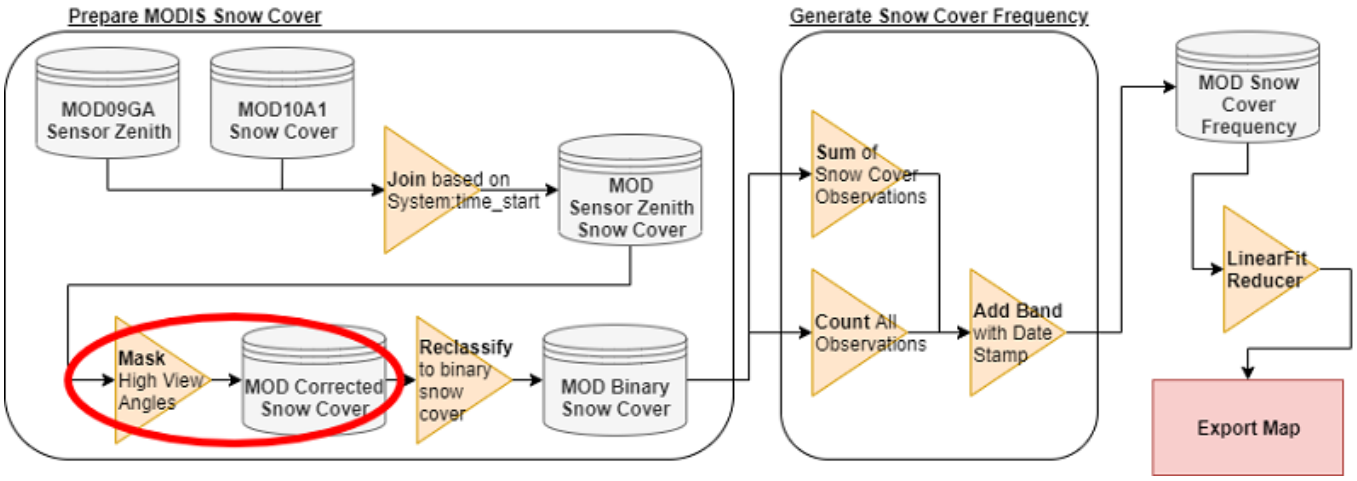

Figure 12. The next step in the GEE workflow is masking out data that meet specified criteria. Source: authors.

To accomplish this, two of the most useful functions in GEE, .map() and .mask(), will be used. As seen in the MergeBands function, we used the .map() method. This method executes the same function across each element in a collection, and returns a collection of the same depth. Masking is another helpful method we'll use. The input of mask is applied to the image and anything which isn't specified within is converted to a null value. Figure 13 shows a simplistic example where we use the .mask() method to keep only the areas with an elevation larger than 2000 m from the SRTM90 elevation dataset in the image.

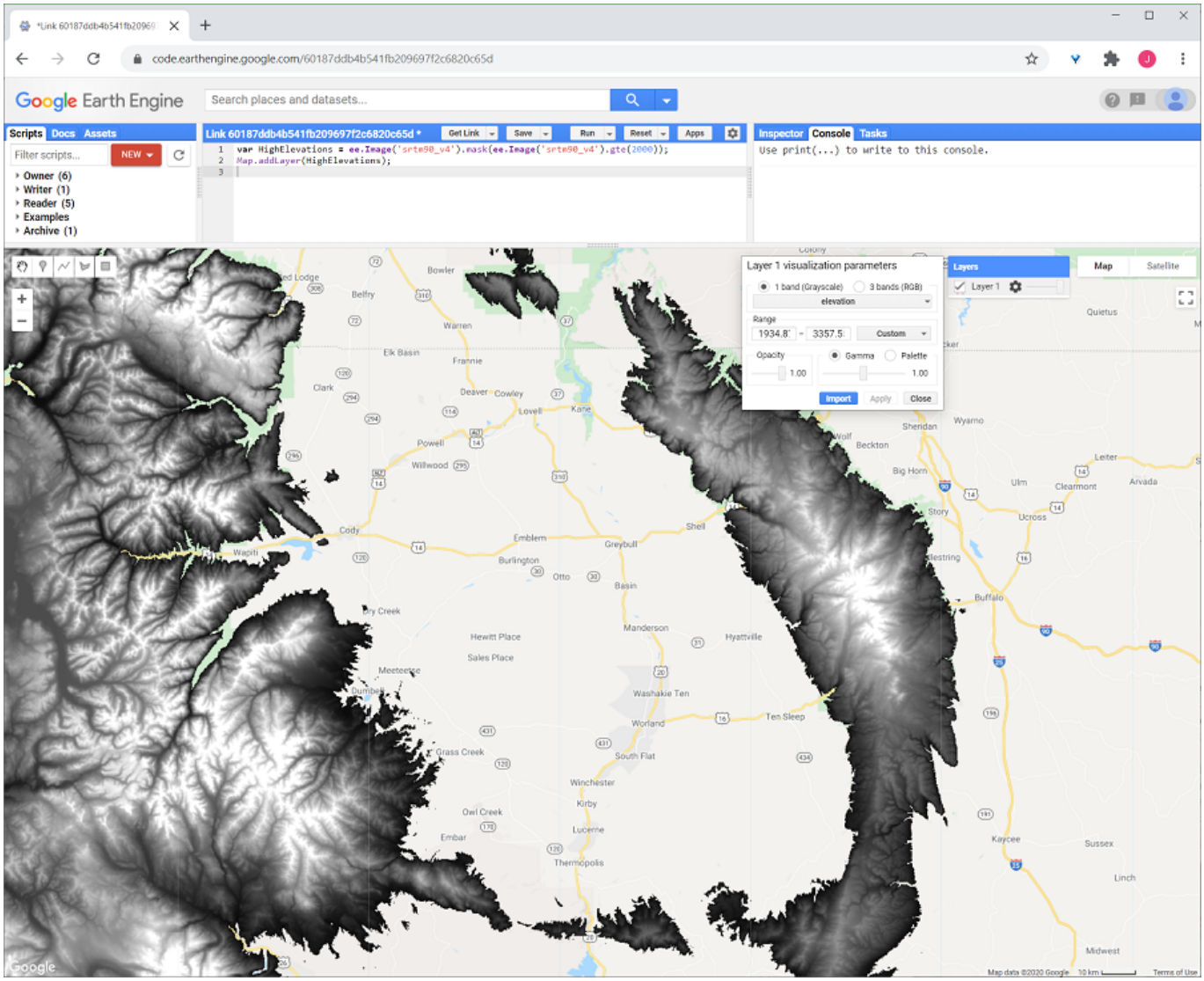

Figure 13. The results of the .mask() method. The code to create this can be found at the Google Earth Engine Code Editor site, available at this link for users once they have a GEE account. Source: authors.

Using the .mask() and .map() methods, we'll modify our code to mask out any pixels with a sensor zenith angle greater than 250. Because we want to do this for every image in the ImageCollection we will use the .map() method with the MaskSensorPixels() function as shown in the highlighted portion of Figure 14.

```
// Tutorial: Masking collections

// A function to assign high sensor zenith angle cells to null
var MaskSensorPixels = function(anImage) {
  var LessThan25 = anImage.select('SensorZenith').lte(2500); // Angle is stored as angle * 100 in MOD09
  return anImage.mask(LessThan25);
};

// A function to merge the bands together after a join
// the bands are referred to as the 'primary' and 'secondary' properties
var MergeBands = function(aRow) {
  var anImage = ee.Image.cat(aRow.get('primary'), aRow.get('secondary'));
  return anImage;
};

// First, we define the two datasets, selecting the bands we are interested in
var MOD09GA = ee.ImageCollection('MODIS/006/MOD09GA')
                .select('SensorZenith')
                .filterDate('2005-10-01', '2005-10-31');
var MOD10A1 = ee.ImageCollection('MODIS/006/MOD10A1')
                .select('NDSI_Snow_Cover')
                .filterDate('2005-10-01', '2005-10-31');

// Define the join and filter
var Join = ee.Join.inner();
var FilterOnStartTime = ee.Filter.equals({'leftField': 'system:time_start',
                                          'rightField': 'system:time_start'
                                         });

// Join the two collections, passing entries through the filter
var JoinedMODIS = Join.apply(MOD09GA, MOD10A1, FilterOnStartTime);

print(JoinedMODIS);

var MergedMODIS = JoinedMODIS.map(MergeBands);
var MakedMODIS = ee.ImageCollection(MergedMODIS).map(MaskSensorPixels);
```

Figure 14. The code for masking out large zenith sensor angle pixels. The code can be found at the Google Earth Engine Code Editor site, available at this link for users once they have a GEE account. Source: authors.

3.5 Reclassifying

The next step is to reclassify the pixels on each image to snow, non-snow or missing (Figure 15).

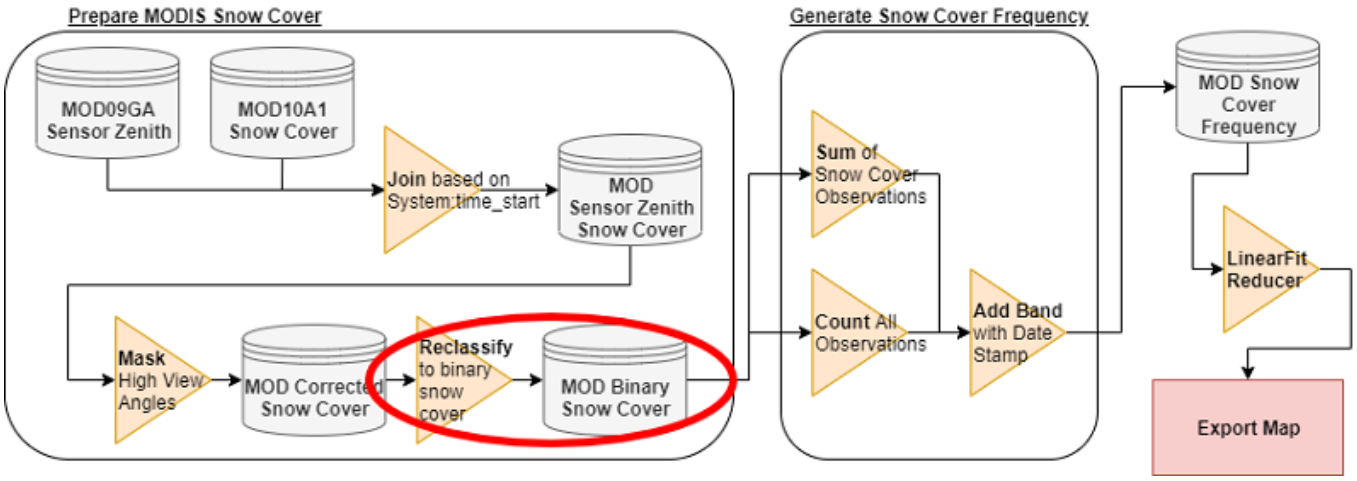

Figure 15. The next step in the GEE workflow is to reclassify the data. Source: authors.

The “NDSI_Snow_Cover” band in MOD10A1.006 is a fractional band with valid values from 0-100, indicating the fractional snow coverage at a pixel. To perform a snow cover frequency analysis, we need to reclassify the band into the no-snow/snow/missing categories, using values that represent “snow” as 1, “No snow” as 0, and null for 'missing' values. To accomplish this GEE has a remap() method, which takes a list of values and maps the values to another list of values you specify. Because we need to reclassify every image in the collection, we will use the handy .map() method again. We are now at the end of our first data processing chain, "Prepare MODIS Snow cover", meaning we can wrap up all the previous steps into a single function, PrepareModisSnowCover(), that takes two dates, a StartDate and a StopDate. The code to do this is shown in Figure 16. As an aside, these last few steps could be wrapped up in the same .map() call, but were separated here for demonstration purposes.

```
//
// Tutorial: Global snow/ice cover change analysis using LinearFit
//

// Creating constants for
var inList = ee.List.sequence(0,100,1).cat([225,237,239]);                  // Create a list with values 0-100,225,237,239
var outList = ee.List.repeat(0,10).cat(ee.List.repeat(1,91)).cat([0,0,0]);  // Create a list with values of 0 for no snow and 1 for snow

// A function to reclassify MODIS snow cover products (MOD10A1/A2) values
// snow/ice--1; no snow/ice--0; all others--null
var Reclassify = function(anImage) {
  var ClassifiedImage = anImage.remap(inList,    // Original pixel values from MODIS Snow Products
                                      outList,   // Reclassified values: 1--snow/ice; 0--no snow/ice
                                      null,                 // All other MODIS snow product pixel values (0, 1, 11, 50, 254, 255)
                                      'NDSI_Snow_Cover');    // The band we wish to remap
  return ClassifiedImage;
};

// A function to merge the bands together after a join
// the bands are referred to as the 'primary' and 'secondary' properties
var MergeBands = function(aRow) {
  var anImage = ee.Image.cat(aRow.get('primary'), aRow.get('secondary'));
  return anImage;
};

// A function to assign high sensor zenith angle cells to null
var CorrectSensorPixels = function(anImage) {
  var LT25 = anImage.select('SensorZenith').lte(2500); // Angle is stored as angle * 100 in MOD09
  return anImage.mask(LT25);
};

// Prepare MODIS snow cover data for calculating snow cover frequency
// Join MOD09 and MOD10, mask out cells with gte 25 sensor-zenith-angle, and reclassify to: 1, 0, null
var PrepareModisSnowCover = function(StartDate, EndDate) {
  // Create MOD09GA and MOD10A1 image collections for the time period
  var MOD09GA = ee.ImageCollection('MODIS/006/MOD09GA')
                  .select('SensorZenith')
                  .filterDate(StartDate, EndDate);
  var MOD10A1 = ee.ImageCollection('MODIS/006/MOD10A1')
                  .select('NDSI_Snow_Cover')
                  .filterDate(StartDate, EndDate);

  // Define the join type and filter
  var innerJoin = ee.Join.inner();
  var joinFilter = ee.Filter.equals({
                                     'leftField': 'system:time_start',
                                     'rightField': 'system:time_start'
                                     });

  // Join the two collections, passing entries through the filter
  var joinedMods = ee.ImageCollection(innerJoin.apply(MOD09GA, MOD10A1, joinFilter));

  // Map our functions over the Image Collections
  var FinalModisDataset = joinedMods.map(MergeBands)
                                    .map(CorrectSensorPixels)
                                    .map(Reclassify);

  return FinalModisDataset;
};

print(PrepareModisSnowCover('2005-10-01', '2005-10-31'));
```

Figure 16. The code used to generate a year of sensor-zenith-angle filtered, reclassified snow cover data. The code can be found at the Google Earth Engine Code Editor site, available at this link for users once they have a GEE account. Source: authors.

3.6 Calculating Snow Cover Frequency

Now that we have a function to create a processed MODIS snow cover dataset for a given time interval, we construct a function to generate a time series of snow cover frequency (Figure 17).

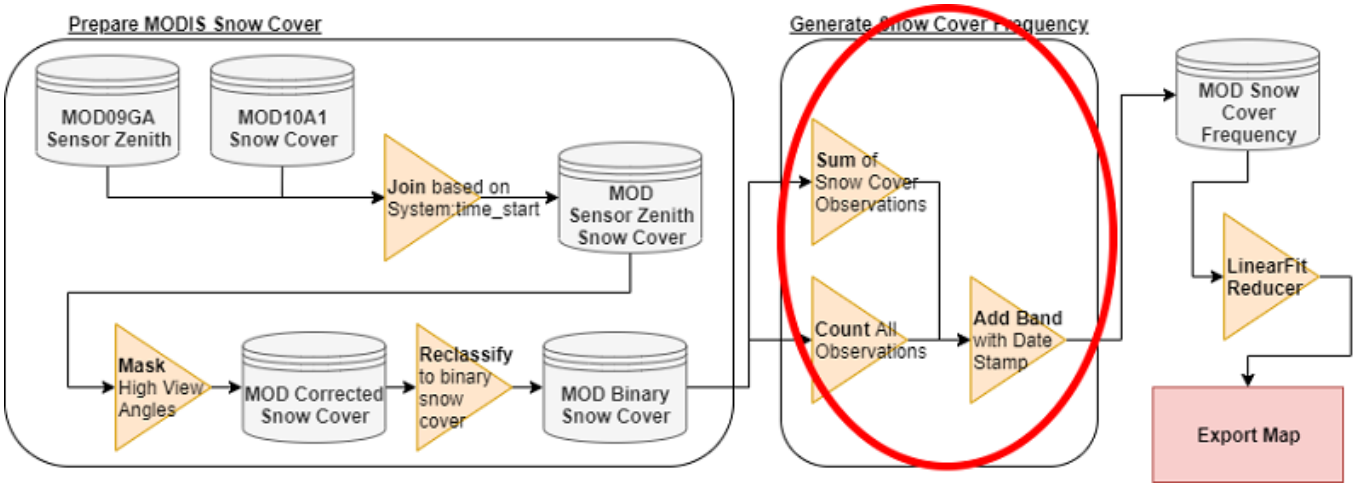

Figure 17. The next step in this workflow is to create a function that generates the time series of images, which stores the snow cover frequency and a date for the image. Source: authors.

The PrepareMODISSnowCover function returns a single band, called remapped, that represents a sensor adjusted and reclassified snow cover for each day of the collection. In order to calculate snow cover frequency we need to know both the number of snow days and the number of days we have valid observations. The .count() method in GEE counts the number of times a pixel has a valid value within an image collection, and will ignore (not count) those images that have a null value, so it counts both snow and no-snow values in a pixel stack. We can also count the number of snow days at a pixel by using the .sum() method, which adds the values of a stack of pixels. We will wrap this up in a function that takes a start and end date, and the number of intervals to advance. This code to accomplish this is shown in Figure 18 below.

```
// A function that generates a list of SCF images in yearly intervals
var GenerateSnowCoverFrequency = function(StartDate, EndDate, TotalSteps) {
  var SCF_List = ee.List([]);
  for(var i = 1; i <= TotalSteps; i++) {
    var MODISSnowCover = PrepareModisSnowCover(StartDate, EndDate);
    var NumOfSnowDays =  MODISSnowCover.sum();                                    // Calculate the number of days with snow cover with .sum()
    var NumOfValidObsDays = MODISSnowCover.count();                               // Count the number of days with a valid observation with .count()
    var SnowCoverFrequency = NumOfSnowDays.toFloat().divide(NumOfValidObsDays);   // perform the SCF calculation
    var SCF_Image = ee.Image(SnowCoverFrequency.select(['remapped'], ['Snow Cover Frequency'])).toDouble();
    SCF_List = SCF_List.add(SCF_Image);
    StartDate = ee.Date(StartDate).advance(1, 'year');
    EndDate = ee.Date(EndDate).advance(1, 'year');
  }
  return ee.ImageCollection(SCF_List);
};

var SCFYearCollection = GenerateSnowCoverFrequency(ee.Date('2000-10-01'), ee.Date('2001-10-01'), 18);
```

Figure 18. The code which calculates snow cover frequency within a given time interval. The code can be found at the Google Earth Engine Code Editor site, available at this link for users once they have a GEE account. Source: authors.

3.7 Trend Analysis

We have now created the necessary functions to create a time series of snow cover frequency, but to calculate a linear trend we also need a band which represents the time stamp (Figure 19).

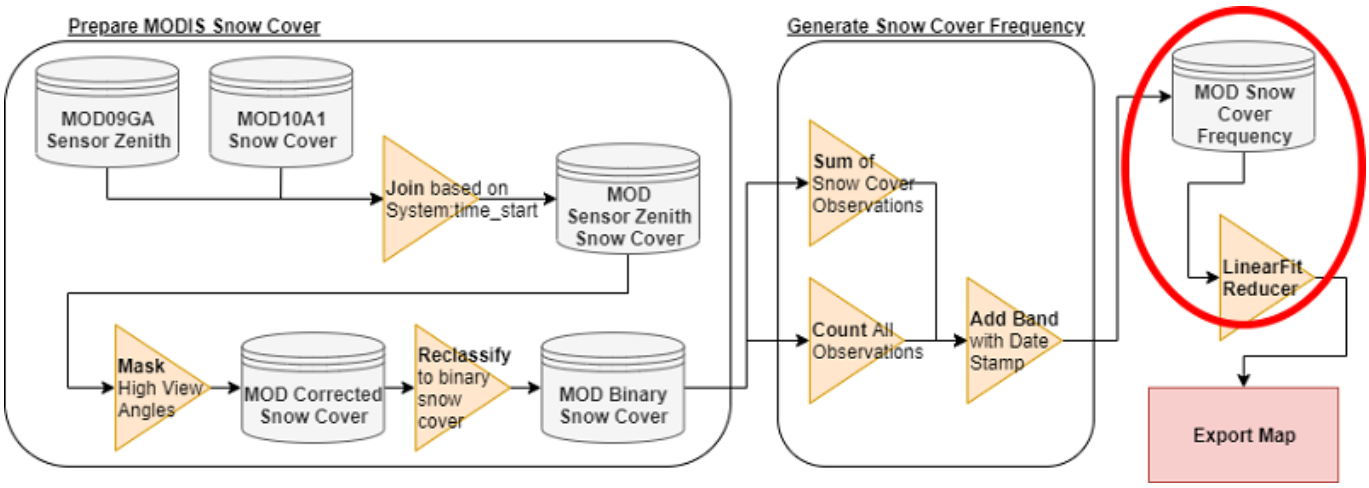

Figure 19. The final analytical step in this workflow example involves fitting a linear trend to our data. Source: authors.

With a small modification to the GenerateSnowCoverFrequency, we can add the EndDate as a numerical band, which gives us everything we need to calculate the linear trend. If we look at the documentation, ee.Reducer.linearFit() returns an image with two bands, one called 'scale' and one called 'offset'. This is Google's terms for 'slope' and 'intercept', so the value we are after is the scale. The code to do so is shown in Figure 20.

```
// A function that generates a list of SCF images in yearly intervals
var GenerateSnowCoverFrequency = function(StartDate, EndDate, TotalSteps) {
  var SCF_List = ee.List([]);
  for(var i = 1; i <= TotalSteps; i++) {
    var MODISSnowCover = PrepareModisSnowCover(StartDate, EndDate);
    var NumOfSnowDays =  MODISSnowCover.sum();
    var NumOfValidObsDays = MODISSnowCover.count();
    var SnowCoverFrequency = NumOfSnowDays.toFloat().divide(NumOfValidObsDays);
    var SCF_Image = ee.Image(ee.Number.parse(EndDate.format('YYYY')))
                      .addBands(SnowCoverFrequency.select(['remapped'], ['Snow Cover Frequency'])).toDouble();
    SCF_List = SCF_List.add(SCF_Image);
    StartDate = ee.Date(StartDate).advance(1, 'year');
    EndDate = ee.Date(EndDate).advance(1, 'year');
  }
  return ee.ImageCollection(SCF_List);
};

var SCFYearCollection = GenerateSnowCoverFrequency(ee.Date('2000-10-01'), ee.Date('2001-10-01'), 18);
var LinearFit = SCFYearCollection.reduce(ee.Reducer.linearFit());
```

```
// Calculate the number of days with snow cover with .sum()
// Count the number of days with a valid observation with .count()
    // perform the SCF calculation
    // Attach a date band (timestamp) used for trend analysis
```

Figure 20. The code to create snow cover frequency and linear trend. The code can be found at the Google Earth Engine Code Editor site, available at this link for users once they have a GEE account. Source: authors.

3.8 Exporting the Results

The final step is to visualize and export the analysis so that others can view it (Figure 21).

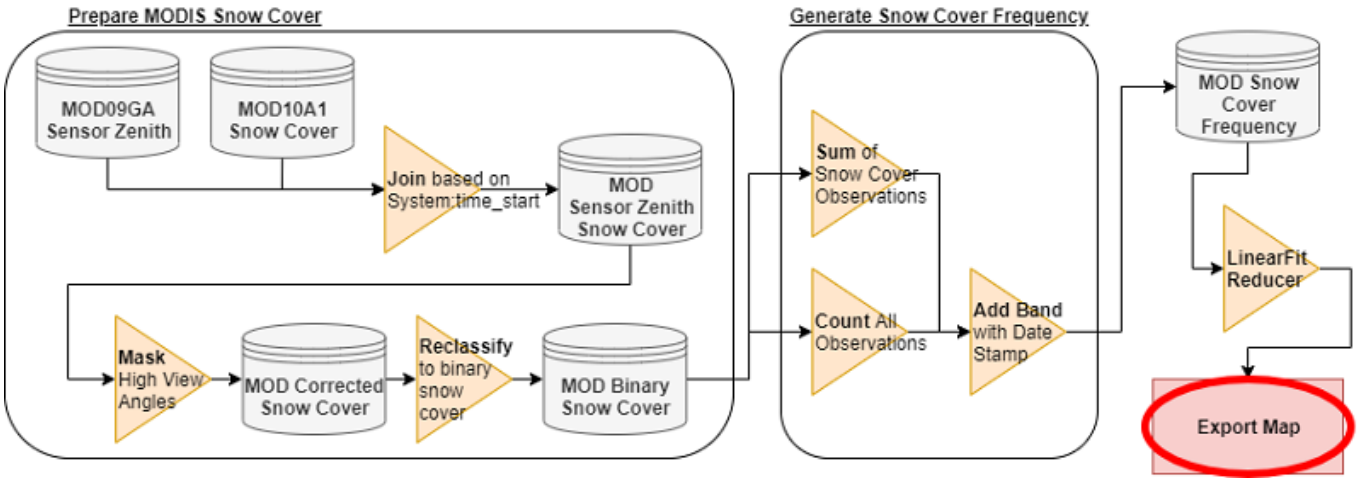

Figure 21. The final step in our workflow is to display and export a final image of our snow cover trend. Source: authors.

The first thing we need is to do is create color palettes. We then filter the slope based on a minimum value of snow cover frequency using the .mask() function with a .min() reducer over the entire collection. This ensures that we are not displaying the trend analyses for areas that have very low snow coverages. We then use the Map.addLayer() method to add a layer to the map, setting the min and max values and the palette we want. Then we will export our analysis as a GeoTIFF using Export.Image.toDrive(). For a little more flair, we'll also export the timeseries of snow cover frequency as a video using Export.video.toDrive(), using another .map() call to force images into rgb space for video. The code to accomplish this is shown in Figure 22.

```
// design a color ramp
var BlueToBrown = ['964B00', 'A15F1C', 'AD7338', 'B98755', 'C49B71', 'D0AF8D', 'DCC3AA', 'E7D7C6', 'F3EBE2', 'FFFFFF', 'E7F1FA', 'CFE4F6',
var WhiteToBlack = ["#000000", "#050505", "#0A0A0A", "#0F0F0F", "#141414", "#1A1A1A", "#1F1F1F", "#242424", "#292929", "#2E2E2E", "#343434

// Mask out low snow areas
var LinearFitSlopeImage = LinearFit.select(['scale']).mask((SCFYearCollection.select(['Snow Cover Frequency']).min()).gte(0.04));

// Add the image to the Map
var imageToExport = LinearFitSlopeImage.visualize({'min':-0.03, 'max':0.03, 'palette':BlueToBrown, 'forceRgbOutput':true});
Map.addLayer(imageToExport,{},'Linear Fit Slope', true);

// Export and image of SCF
Export.image.toDrive({image: imageToExport, folder:'GEE', description: "LinearSnowCoverTrend", region: geometry, maxPixels:67000000000});

// Export a video of SCF timeseries
var VidCollection = SCFYearCollection.select(['Snow Cover Frequency']).map(function(anImage) {
  return anImage.visualize({'min':0, 'max':1, 'palette':WhiteToBlack, 'forceRgbOutput':true});
});
Export.video.toDrive({collection: VidCollection,
                      folder:'GEE',
                      region: geometry,
                      description: 'SnowCoverFreqency',
                      framesPerSecond :0.5,
                      maxPixels:67000000000});
```

Figure 22. The code to accomplish the display and export of our analysis. The code can be found at the Google Earth Engine Code Editor site, available at this link for users once they have a GEE account. Source: authors.

3.9 Building a Web-based Application

Finally, let's capitalize on the advantage of the new UI and app features in GEE to create an interactive web map so that anyone may view our analysis. The UI Examples provide several great frameworks to construct an app, so we will borrow some of them to create an application using our new snow cover trend data. Once published, anyone can access the analysis online (Figure 23).

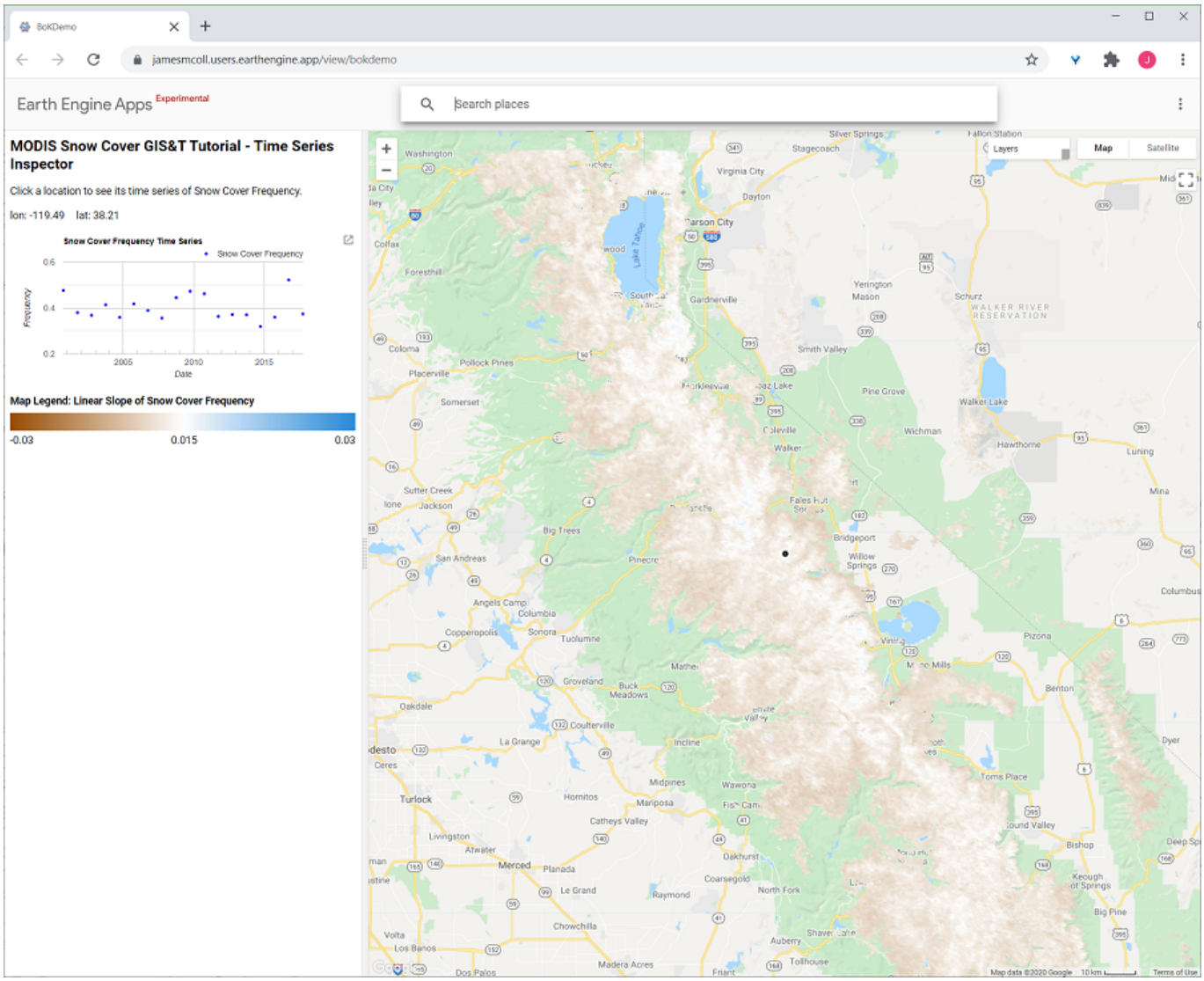

Figure 23. A screenshot of the final published online interactive analysis of GEE-derived snow cover frequency. Source: authors.

Taking a step back to look at what we've accomplished in this tutorial really accentuates what can be accomplished using GEE:

- We pulled in 18 years for two daily MODIS datasets
- Joined them in space and time
- Filtered and reclassified the resulting dataset
- Calculated snow cover frequency
- Performed the linear fit with the snow cover frequency dataset
- Exported a GeoTIFF and video of the resulting slope map
- and created a web application so that anyone can interact with the analysis

And the entire process probably took less time to perform than it would have needed to download a single day of MODIS data. This is just a small sample of the processing power and capabilities that Google Earth Engine has to offer, and an example of how this platform has revolutionized global scale geospatial analyses.

4. Conclusion

Taken as a whole, GEE represents the most advanced, cloud-based geoprocessing platform to date. Although several other platforms encompass some of these aspects, no suite of tools currently available can replicate the access to geospatial data, relative simplicity of use, and sheer power of analysis that GEE offers. The platform is constantly under development and new features and algorithms are continuously updated. The GEE team and community is also incredibly approachable and helpful with GEE booths now commonly seen at many geospatial meetings, where they demo use cases and announce new features.

References

Hansen, M.C., Potapov, P.V., Moore, R., Hancher, M., Turubanova, S.A., Tyukavina, A., Thau, D., Stehman, S.V., Goetz, S.J., Loveland, T.R., Kommareddy A., Egorov, A., Chini, L., Justice1, C.O., Townshend, J.R.G. (2013). High-Resolution Global Maps of 21st-Century Forest Cover Change. Science, 342(6160), 850-853.