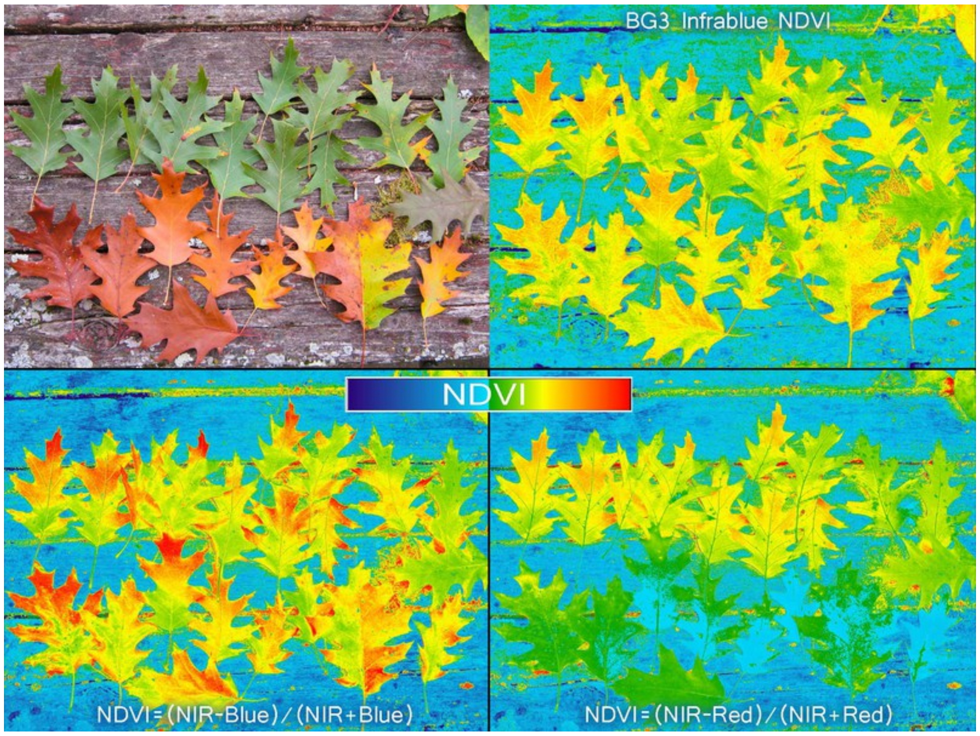
EXERCISE 2
Calculate spectral indices



CC-BY-SA 2017 @cfastie and other Public Lab contributors

Introduction
You have already accessed Landsat imagery to produce a seasonal composite, with six distinct spectral bands. But we can leverage even more information from the spectral bands by calculating ratios and other indices that are linked to the biotic and geologic phenomena that we are interested in. In this exercise, you will continue to develop your Earth Engine skills by writing scripts and using functions, and you will gain hands-on experience calculating spectral indices to use as input layers to your analysis.

Objectives
- Use raster math and EE functions to calculate band ratios
- Load in a module to access specific functions in Google Earth Engine (GEE)
- Understand the function and utility of well-commented code
- Understand how adjusting visualization parameters adjusts how images are displayed in GEE

Required Data:

- **VT_boundary.shp** – shapefile representing example area of interest

Prerequisites

- **Completion of Exercise 1 (you can review completed code in the course repository)**
- **Google Chrome installed on your machine**
- **An approved Google Earth Engine account**
- **Follow the links below to gain read access to the GEE code repositories we will refer to in the script.**
  - Click here to gain access to the GTAC module repository
  - Click here to gain access to the GTAC training repository

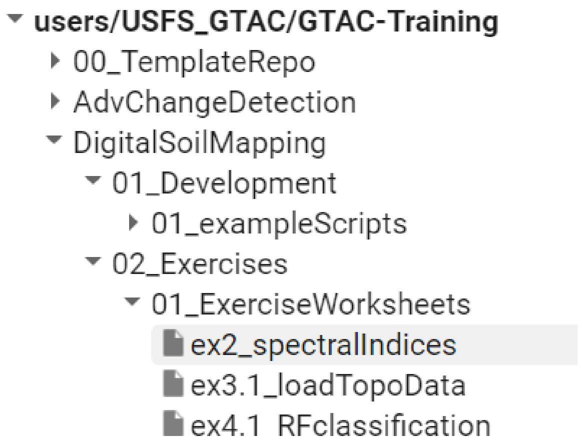
Table of Contents

Part 1: Prepare Landsat image composite

For our first step, we will prepare a Landsat image composite. This is accomplishing the same thing that we did in the previous exercise. This time, the script that we used in the previous exercise has been combined into a function in a library that we will load. So, we can do everything that we did in the previous exercise—without re-writing all the code.

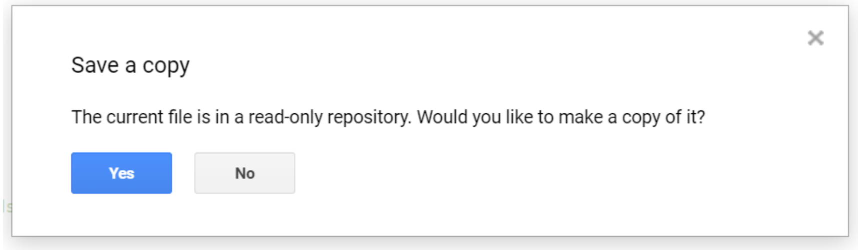
A. Load the exercise script and input data

1. In the course repository, navigate to the script ex2_spectralIndices (located in DigitalSoilMapping, 02_Exercises, 01_ExerciseWorkSheets). For this exercise, we have provided an outline of a script into which you will write and copy the appropriate code. For your reference, completed scripts are provided for you as well (located in DigitalSoilMapping, 02_Exercises, 02_ExerciseCompleteScripts).



```
▾ users/USFS_GTAC/GTAC-Training
    ▸ 00_TemplateRepo
    ▸ AdvChangeDetection
    ▾ DigitalSoilMapping
        ▾ 01_Development
            ▸ 01_exampleScripts
        ▾ 02_Exercises
            ▾ 01_ExerciseWorksheets
                ex2_spectralIndices
                ex3.1_loadTopoData
                ex4.1 RFclassification
```

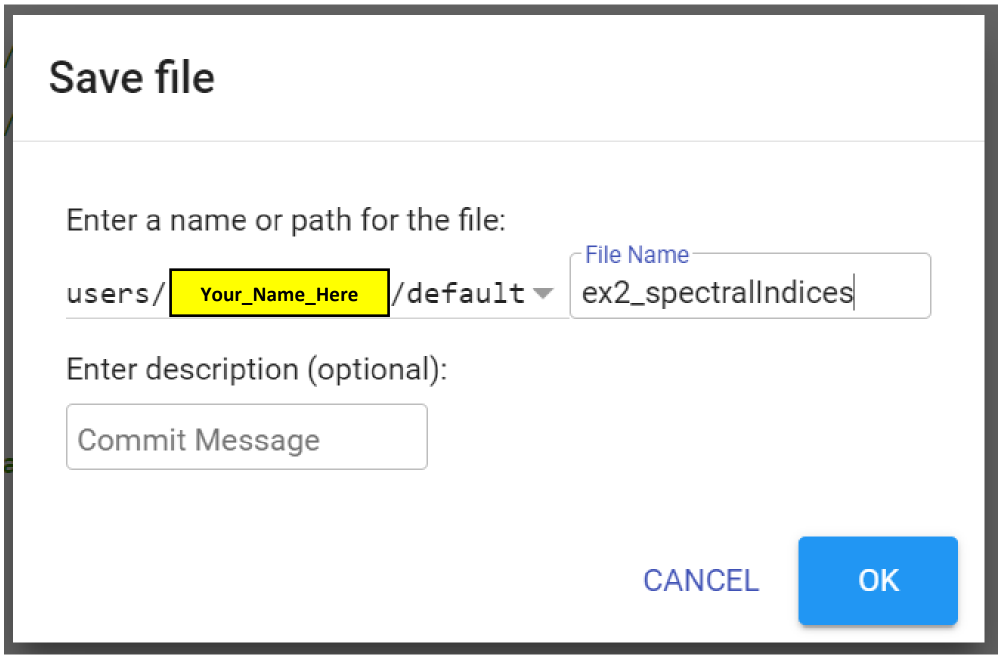
2. Open the script and inspect it. The first thing you will notice is the heading, describing the title, authorship, date modified, and summary of the script. If you scroll down, you will see headings describing the different tasks that the script plans to accomplish. But, there is no code written.
3. Save the code with a unique name to your own code repository for the course.
    i. If the Save button is greyed out, you may have to make a small edit to the code first – you can add a space after text somewhere within the header (or make any other edits to the header that you wish). Once you make a small edit, the Save button with appear.
    ii. Click the Save drop down and select **Save as…** - you will get a pop-up that asks if you would like to make a copy. Select **Yes**.



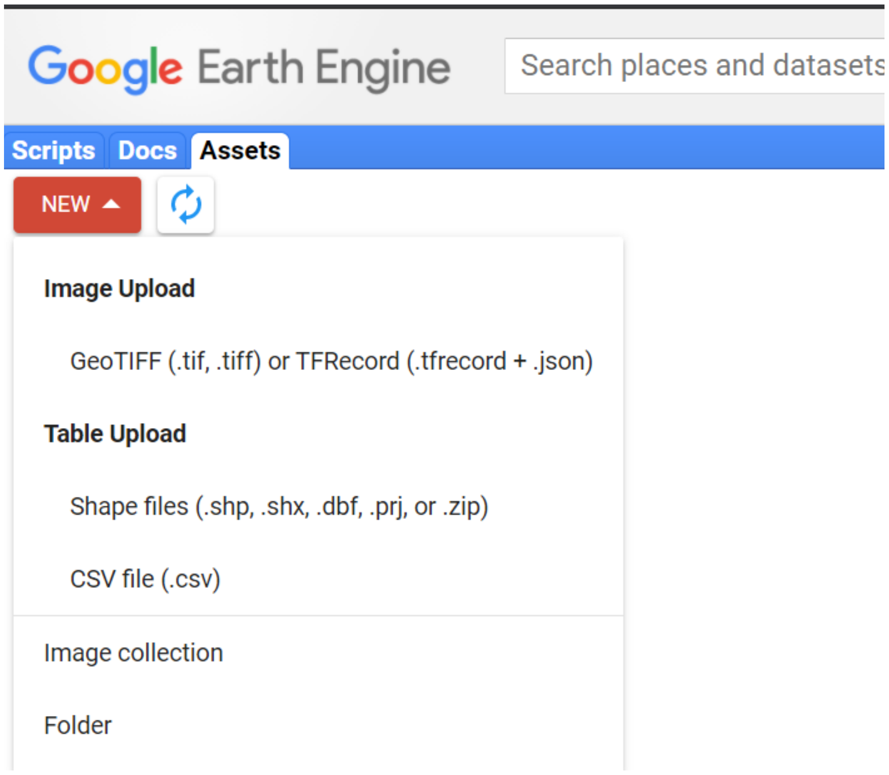
Save a copy

The current file is in a read-only repository. Would you like to make a copy of it?

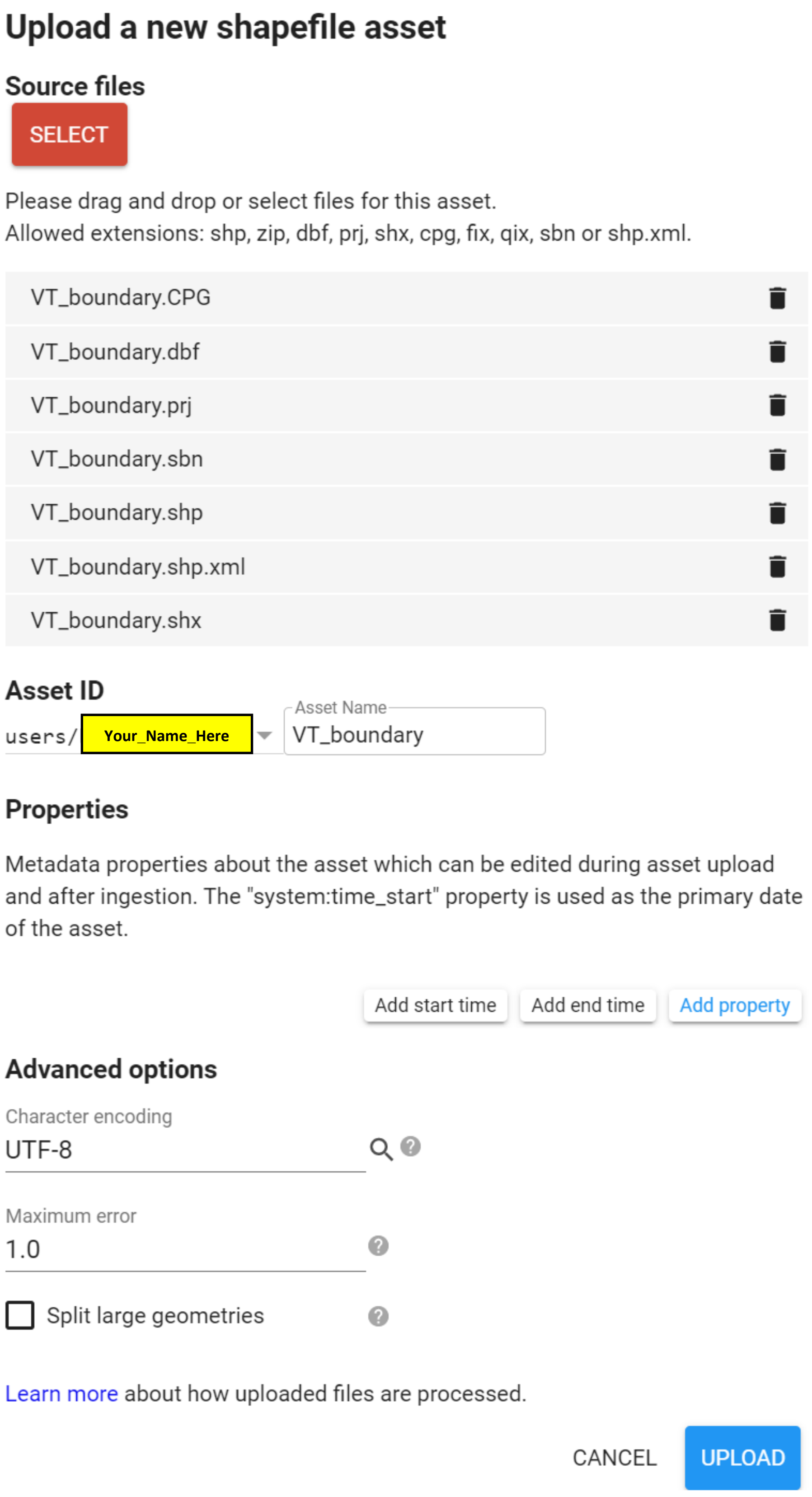
    iii. Choose a repository in which to save the script, and name it something intuitive, like **ex2_spectralIndices**.


Save file
Enter a name or path for the file:
users/ Your_Name_Here /default
File Name
ex2_spectralIndices
Enter description (optional):
Commit Message
CANCEL
OK

4. Ensure that you have uploaded the **VT_boundary.shp** file to your assets folder.
   i. If you need to upload the file as an Asset, select the **Assets** tab.
   ii. Select the red **NEW** button and then select **Shape files (.shp, .shx, .dbf, .prj, or .zip)** from the **Table Upload** section, shown below.


Google Earth Engine
Search places and datasets
Scripts
Docs
Assets
NEW
Image Upload
GeoTIFF (.tif, .tiff) or TFRecord (.tfrecord + .json)
Table Upload
Shape files (.shp, .shx, .dbf, .prj, or .zip)
CSV file (.csv)
Image collection
Folder

   iii. Click on the red **SELECT** button. Navigate to where you have stored your course data, open the **Essex_VT** folder, and then open the **VT_Boundary** folder. Hold the **CTRL** key and select all but the file with the SBX extension. Select **Open**.



Upload a new shapefile asset

Source files

Please drag and drop or select files for this asset.
Allowed extensions: shp, zip, dbf, prj, shx, cpg, fix, qix, sbn or shp.xml.

VT_boundary.CPG
VT_boundary.dbf
VT_boundary.prj
VT_boundary.sbn
VT_boundary.shp
VT_boundary.shp.xml
VT_boundary.shx

Asset ID

users/ Your_Name_Here

Asset Name: VT_boundary

Properties

Metadata properties about the asset which can be edited during asset upload and after ingestion. The "system:time_start" property is used as the primary date of the asset.

Advanced options

Character encoding: UTF-8

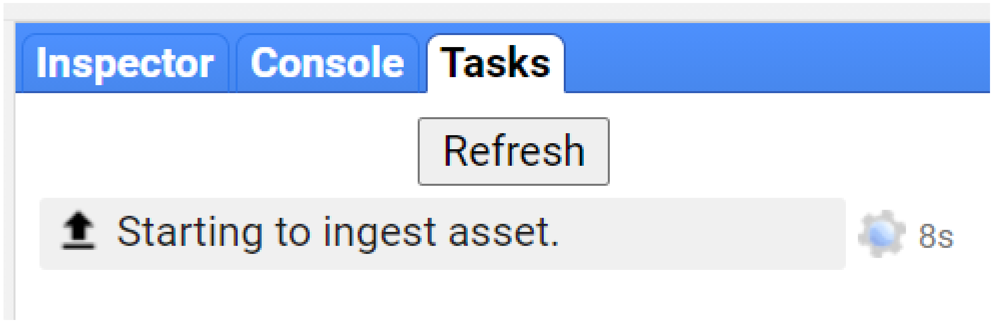
Maximum error: 1.0

☐ Split large geometries

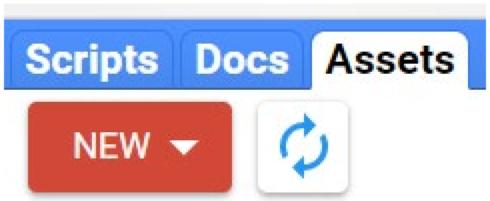
Learn more about how uploaded files are processed.

iv. It is recommended to leave it as the default. Note that if you change the name, you may need to edit the code to reflect this difference.

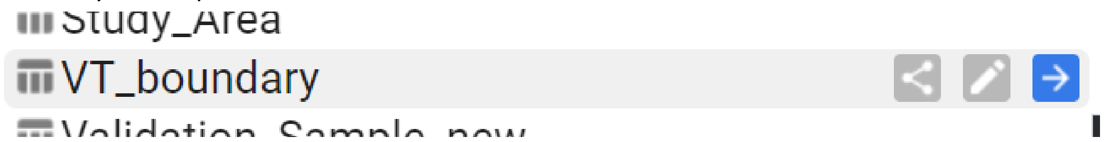
v. Click the Upload button. The upload will appear in your Tasks tab and may take a few minutes to complete. You can click Refresh to update the status.



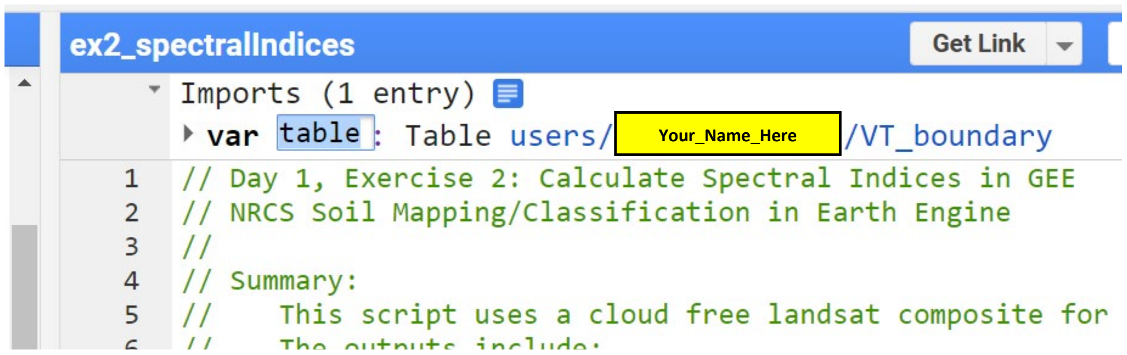
vi. When finished, you may need to click the Refresh button, just to the right of the red NEW button, in your Assets tab for the new asset to appear.



vii. Once your asset is loaded, click the blue arrow that points to the right to import it into your script.



viii. The asset import will appear at the top of your script – click where it says **table** to edit the name of the variable, change it to **VT_boundary**, and hit the **Enter** key to complete the edit.



```
Imports (1 entry)
var table: Table users/Your_Name_Here/VT_boundary
// Day 1, Exercise 2: Calculate Spectral Indices in GEE
// NRCS Soil Mapping/Classification in Earth Engine
//
// Summary:
//     This script uses a cloud free landsat composite for
```

B. Prepare the composite

1. We have combined the commands we wrote in the previous exercise into a single function that loads a composite for a given area. This function takes a suite of input parameters that we can adjust for our areas and applications of interest.

2. Load the library by copying the following lines of code into line 16 of the script outline, below the comment that reads “Load in library with function to load Landsat composite.”

```
// Load in library with function to load Landsat composite

var loadComposite = require('users/USFS_GTAC/GTAC-Training:DigitalSoilMapping/03_Library/DSM_Lib');
```

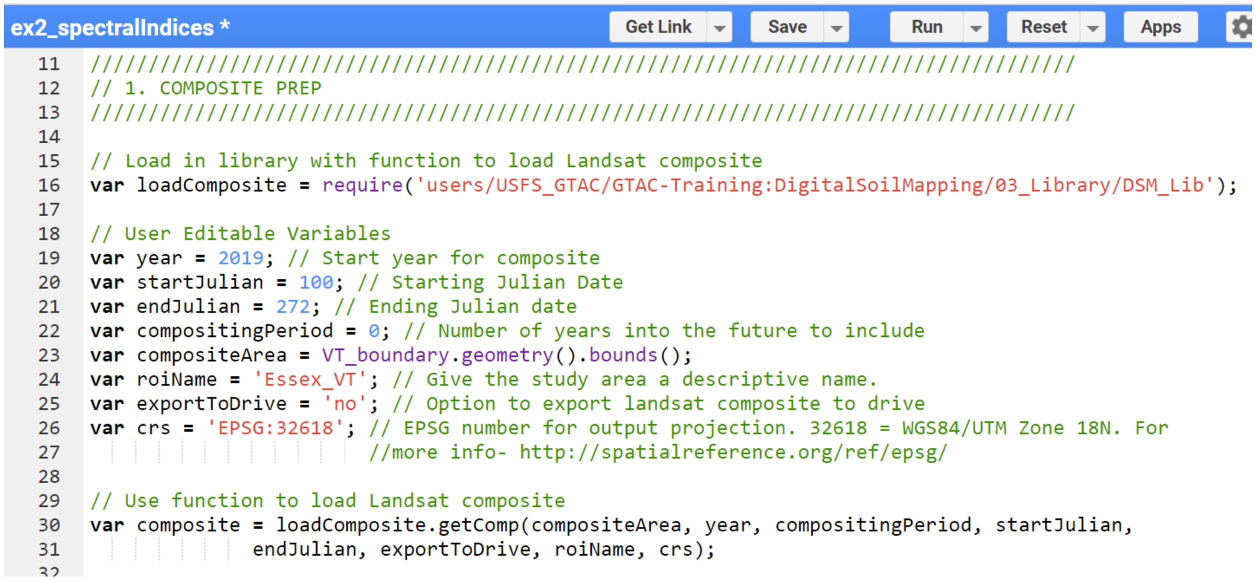
3. Add a descriptive comment above this line of code to describe what the code is doing. Two backslashes at the beginning of the line designate the code as a comment, so that you can describe what you’re doing without running the text as code.
4. Add in the user editable variables. These variables are the parameters that are used in the compositing function. Again, these will be the same as in the first exercises. Copy these lines of code below the comment that reads “User Editable Variables” (which should be the next comment line in the code). Note that we have already included comments describing what each variable represents.

```
var year = 2019; // Start year for composite

var startJulian = 100; // Starting Julian Date

var endJulian = 272; // Ending Julian date

var compositingPeriod = 0; // Number of years into the future to include

var compositeArea = VT_boundary.geometry().bounds();

var roiName = 'Essex_VT'; // Give the study area a descriptive name.

var exportToDrive = 'no'; // Option to export landsat composite to drive

var crs = 'EPSG:32618'; // EPSG number for output projection. 32618 = WGS84/UTM Zone 18N. For more info- http://spatialreference.org/ref/epsg/
```

5. Use the function and the parameters to load the Landsat composite. Here, we create the object “composite” by referencing the library that we loaded, calling the function “getComp”, and then providing the input parameters. Copy this line of code into your script below the line that reads “Use function to load Landsat composite.”

```
var composite = loadComposite.getComp(compositeArea, year, compositingPeriod,
startJulian, endJulian, exportToDrive, roiName, crs);
```

6. The function that we called includes a command to print the results of the composite to the console, so this is all we need to do.
7. Go back and add any additional comments that will help you remember what the code is doing, or clarify your understanding. You can also add line breaks to separate functions and organize input parameters. Your code should look something like this:


ex2_spectralIndices * Get Link Save Run Reset Apps

```
//////////////////////////////////////////////////////////////////////////////////////
// 1. COMPOSITE PREP
//////////////////////////////////////////////////////////////////////////////////////

// Load in library with function to load Landsat composite
var loadComposite = require('users/USFS_GTAC/GTAC-Training:DigitalSoilMapping/03_Library/DSM_Lib');

// User Editable Variables
var year = 2019; // Start year for composite
var startJulian = 100; // Starting Julian Date
var endJulian = 272; // Ending Julian date
var compositingPeriod = 0; // Number of years into the future to include
var compositeArea = VT_boundary.geometry().bounds();
var roiName = 'Essex_VT'; // Give the study area a descriptive name.
var exportToDrive = 'no'; // Option to export landsat composite to drive
var crs = 'EPSG:32618'; // EPSG number for output projection. 32618 = WGS84/UTM Zone 18N. For
                        //more info- http://spatialreference.org/ref/epsg/

// Use function to load Landsat composite
var composite = loadComposite.getComp(compositeArea, year, compositingPeriod, startJulian,
              endJulian, exportToDrive, roiName, crs);
```

8. Click **Save** to save the code.
9. Click **Run**. Only click Run once and be patient. GEE is slow about loading libraries.
10. An object called “Landsat Composite” will appear in the console. Click the down arrows next to “Image” and then next to “bands” to inspect this object. Observe that we have created a Landsat image composite with 6 bands, using only about 20 lines of code!

Part 2: Calculate NDVI and NDVI percentiles

A. Calculate NDVI

1. We can calculate NDVI by using the **.normalizedDifference()** function. NDVI is a ratio between the red and near-infrared bands. We could write out the function by hand using mathematical functions, but the **.normalizedDifference()** function already contains the math necessary—all we have to do is choose which bands we want to use to compute the ratio. As you learned in the lecture, NDVI is only one of many indices that are calculated as a normalized difference between bands.

We’ll continue to use and introduce new functions throughout this exercise and the subsequent exercises. We’ll explain a bit about what they do here, but you can always access the documentation for each function in GEE itself. Navigate to the Docs pane on the left-hand side of the Code Editor and search for a function, or browse all the different functions available. The documentation will tell you what a function accomplishes, the syntax for running a function, and describe all the parameters that the function takes as inputs.

2. Copy this code below the comment that reads “Calculate NDVI from composite.”

```
var ndvi = composite.normalizedDifference(['nir', 'red'])

                    .rename("ndvi");
```

3. Note that we are using two separate functions, colored purple in the script. The function **.normalizedDifference()** takes the near-infrared and red bands from the composites as inputs,

and calculates NDVI from those two bands. The **.rename()** function changes the name of the NDVI band inside the image. We can use multiple functions in a row on the same input by writing them in order. We could write all of these on the same line, and get the same result! Here, we've used line breaks and tabs to line up the functions on the same vertical axis to make it easier to read what functions we're using.

4. We calculated **ndvi** as a single band. We need to add it to the composite, to create a stack of input image layers that we will ultimately use as predictor layers for our soil maps. To do this, we use the **.addBands()** function. Copy this code below the comment that reads "Add NDVI band to composite."

```
var composite_ndvi = composite.addBands(ndvi);
```

B. Inspect and add NDVI to the map

1. You can check to see that the **ndvi** band has been added to the composite by using the print command. Copy this line of code below "inspect new image collection to see what layers are available."

```
print(composite_ndvi, "composite NDVI");
```

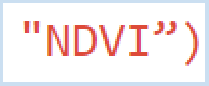
2. Now that we've calculated NDVI, we can add it to the map to inspect the layer visually. We can specify visualization parameters in order to view the layer in the color scheme that we're interested in. To save us from writing this every time we add the layer to the map, we can create an object that contains visualization parameters. Copy this line of code below "Set visualization parameters for ndvi bands."
   i. Because NDVI is a normalized difference ratio, all values of NDVI will be between -1 and 1.
   ii. This sets the minimum value we visualize to -1, and the maximum value to 1. We also specify that we want blue to represent the lowest end of the values we're visualizing, white to represent 0, and green to represent the top end of the values we're visualizing.

```
var ndviParams = {min: -1, max: 1, palette: ['blue', 'white', 'green']};
```

3. Next, we can add the layer to the map to visually inspect it. Copy this line of code to the script below "Add the NDVI layer to the map."
   i. This line of code uses **.select()** to choose only the NDVI band from the composite, specifies our already-created visualization parameters, and specifies a name for what the layer will be called in the Layer display.

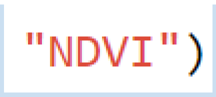
```
Map.addLayer(composite_ndvi.select("ndvi"), ndviParams, "NDVI”);
```

***Note:** if you get an 'unclosed string' error, just be sure to double check that your open and close quotes match – sometimes copy and pasting maintains formatting that doesn't play nicely with Earth Engine. In these cases, just delete and re-enter the open and close quotation marks to properly define your string.*



```
"NDVI”)
```

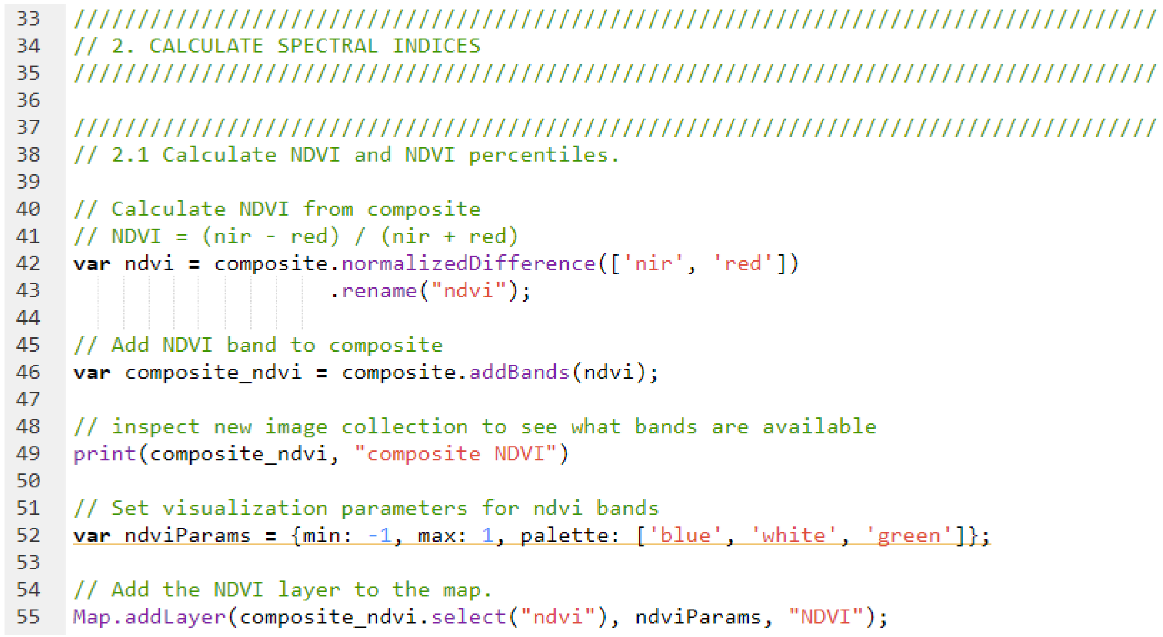
Notice how the second quotation mark is more curved and the close parenthesis is red (indicating that it is part of the string?



```
"NDVI")
```

Once the second quotation mark is deleted and re-entered, it matches the first one, and the close parenthesis turns black, meaning that it is no longer part of the string.

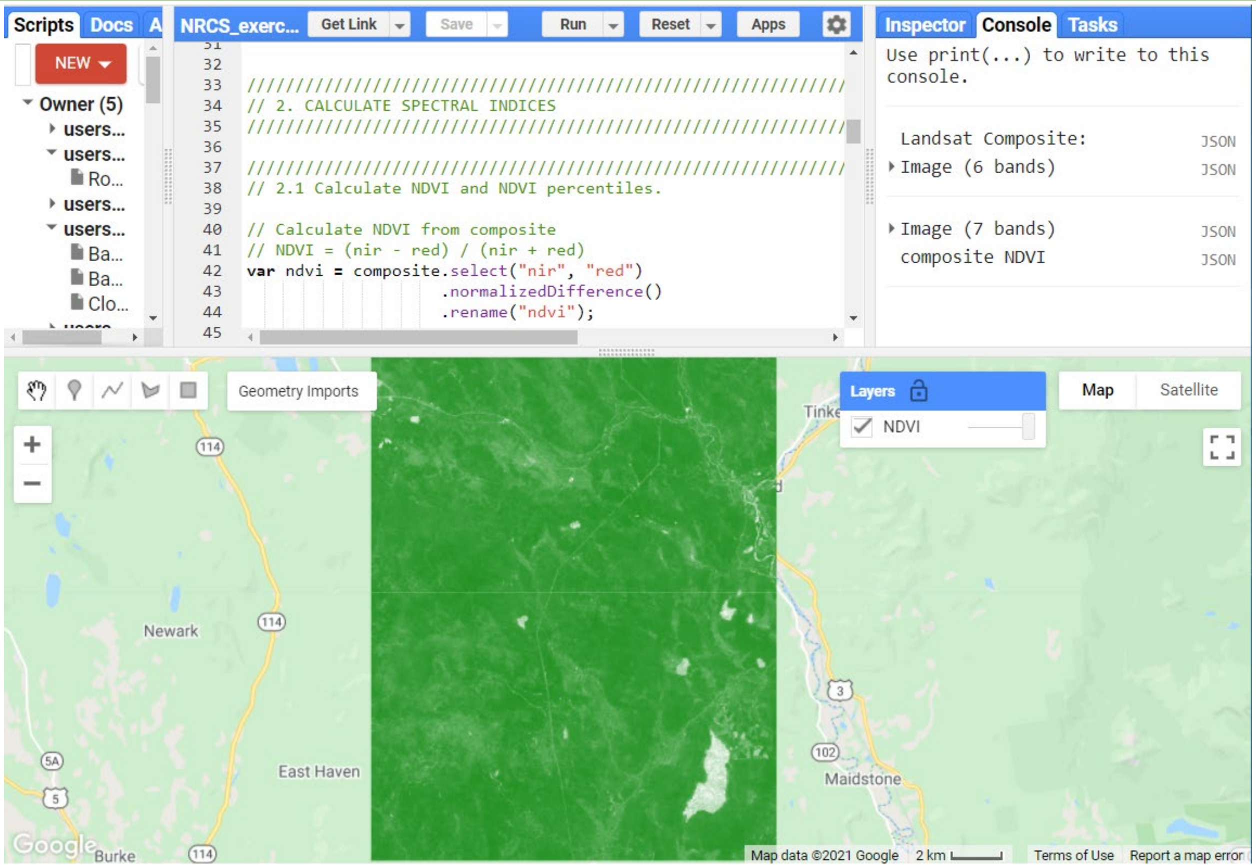
4. Your final code for this section should look something like this.



```
/////////////////////////////////////////////////////////////////////////////////////
// 2. CALCULATE SPECTRAL INDICES
/////////////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////////////
// 2.1 Calculate NDVI and NDVI percentiles.

// Calculate NDVI from composite
// NDVI = (nir - red) / (nir + red)
var ndvi = composite.normalizedDifference(['nir', 'red'])
                    .rename("ndvi");

// Add NDVI band to composite
var composite_ndvi = composite.addBands(ndvi);

// inspect new image collection to see what bands are available
print(composite_ndvi, "composite NDVI")

// Set visualization parameters for ndvi bands
var ndviParams = {min: -1, max: 1, palette: ['blue', 'white', 'green']};

// Add the NDVI layer to the map.
Map.addLayer(composite_ndvi.select("ndvi"), ndviParams, "NDVI");
```

5. **Save** the code and click **Run**.

   i. You should see an NDVI layer appear on the map, an NDVI Layer appear in the Layers tab on the Map pane, and you should see the Landsat Composite and composite NDVI in the Console.



```
//////////////////////////////////////////////////////////////////
// 2. CALCULATE SPECTRAL INDICES
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
// 2.1 Calculate NDVI and NDVI percentiles.

// Calculate NDVI from composite
// NDVI = (nir - red) / (nir + red)
var ndvi = composite.select("nir", "red")
                    .normalizedDifference()
                    .rename("ndvi");
```

```
Use print(...) to write to this console.

Landsat Composite:                    JSON
▸Image (6 bands)                      JSON

▸Image (7 bands)                      JSON
composite NDVI                        JSON
```

Part 3: Calculate mineral / geologic indices

A. Perform calculations

1. We can use the same **.normalizedDifference()** function to calculate spectral indices that are related to minerals and local geology, rather than vegetation.
   i. Note that for each index, we have provided the calculation that is being performed.
2. The code is very similar, but these spectral indices are calculated from different combinations of input bands. We still are chaining multiple functions and renaming our output band. Copy these lines of code below the appropriate comments.

```
var carbonateIndex = composite.normalizedDifference(['red','green'])
                              .rename('carbonateIndex');
```

```
var rockOutcropIndex = composite.normalizedDifference(['swir1','green'])
                              .rename('rockOutcropIndex');
```

3. Like how we calculated NDVI above, we have calculated these indices as stand-alone bands that we will add to a composite stack shortly.

4. Next, we'll calculate two indices that are not normalized differences. Instead, we use mathematical functions to divide one index by another. Instead of the **.normalizedDifference()** function, we use the **.divide()** function. Copy these lines of code below the appropriate comments.

```
var clayIndex = composite.select('swir1')
                         .divide(composite.select('swir2'))
                         .rename('clayIndex');
var ferrousIndex = composite.select('swir1')
                           .divide(composite.select('nir'))
                           .rename('ferrousIndex');
```

B. Add to composite and visualize

1. Add the bands to the composite, creating a new composite stack with the mineral indices. We are still using the **.addBands()** function, but we can use the brackets [] within the function to add a list of bands instead of a single band. Copy this line of code below the appropriate comment.

```
var composite_minerals = composite.addBands([carbonateIndex, rockOutcropIndex,
clayIndex, ferrousIndex]);
```

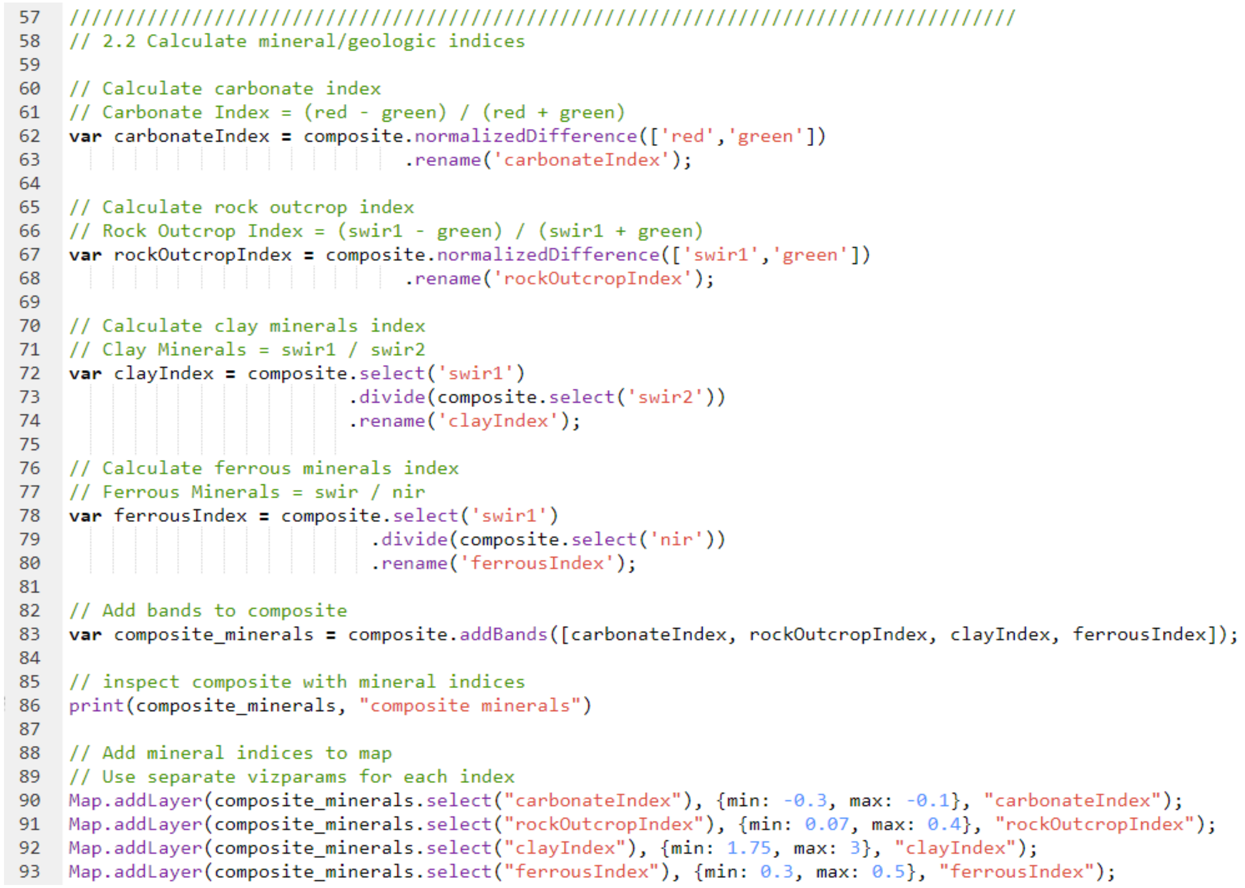
2. Print the new composite to inspect and ensure that the appropriate layers were added.

```
print(composite_minerals, "composite minerals");
```

3. Now, we'll add them to the map. Here, because the indices have varied ranges, we're specifying distinct minimum and maximum values for each one. Since we aren't supplying any colors to the palette, these will be visualized in greyscale.

```
Map.addLayer(composite_minerals.select("carbonateIndex"), {min: -0.3, max: -0.1},
"carbonateIndex");
Map.addLayer(composite_minerals.select("rockOutcropIndex"), {min: 0.07, max:
0.4}, "rockOutcropIndex");
Map.addLayer(composite_minerals.select("clayIndex"), {min: 1.75, max: 3},
"clayIndex");
Map.addLayer(composite_minerals.select("ferrousIndex"), {min: 0.3, max: 0.5},
"ferrousIndex");
```

4. Click Save. Your code should look like the image below.



```
////////////////////////////////////////////////////////////////////////////////////////
// 2.2 Calculate mineral/geologic indices

// Calculate carbonate index
// Carbonate Index = (red - green) / (red + green)
var carbonateIndex = composite.normalizedDifference(['red','green'])
                              .rename('carbonateIndex');

// Calculate rock outcrop index
// Rock Outcrop Index = (swir1 - green) / (swir1 + green)
var rockOutcropIndex = composite.normalizedDifference(['swir1','green'])
                              .rename('rockOutcropIndex');

// Calculate clay minerals index
// Clay Minerals = swir1 / swir2
var clayIndex = composite.select('swir1')
                         .divide(composite.select('swir2'))
                         .rename('clayIndex');

// Calculate ferrous minerals index
// Ferrous Minerals = swir / nir
var ferrousIndex = composite.select('swir1')
                           .divide(composite.select('nir'))
                           .rename('ferrousIndex');

// Add bands to composite
var composite_minerals = composite.addBands([carbonateIndex, rockOutcropIndex, clayIndex, ferrousIndex]);

// inspect composite with mineral indices
print(composite_minerals, "composite minerals")

// Add mineral indices to map
// Use separate vizparams for each index
Map.addLayer(composite_minerals.select("carbonateIndex"), {min: -0.3, max: -0.1}, "carbonateIndex");
Map.addLayer(composite_minerals.select("rockOutcropIndex"), {min: 0.07, max: 0.4}, "rockOutcropIndex");
Map.addLayer(composite_minerals.select("clayIndex"), {min: 1.75, max: 3}, "clayIndex");
Map.addLayer(composite_minerals.select("ferrousIndex"), {min: 0.3, max: 0.5}, "ferrousIndex");
```

5. Click **Run**. Now, in addition to the NDVI layer, the four mineral and geologic indices will populate on the map and in the Layers pane. Toggle the geologic indices on and off to explore the different layers. You can also use the swipe bar for faster comparisons.

Part 4: Calculate tasseled-cap image transformations

A. Perform calculations

1. Lastly, we'll use another pre-existing function to calculate the Tasseled Cap image transformations. We'll load a module created by GTAC that has a large suite of different functions for acquiring and processing satellite imagery! Here, though, we'll only use the functions for calculating Tasseled Cap.
   i. Because the Tasseled Cap image transformations are long linear transformations with very specific coefficients, using the GTAC library not only saves us work, but also can prevent us from making mistakes by entering the coefficients incorrectly.
2. We'll use the **.require()** command again to load the GTAC **getImagesLib**.
   i. Make sure you've added the USFS_GTAC/modules repository to your Earth Engine account. When you have Read access to this repository, you can access any of the functions within it.

3. Copy this line of code below the appropriate comment.

```
var getImages = require('users/USFS_GTAC/modules:getImagesLib.js');
```

4. The **.getTasseledCap()** function takes as an input any composite or image with the appropriate set of input bands. The output of the **.getTasseledCap()** function is the composite, with six distinct Tasseled Cap indices also added as bands to the image.
   i. You can also inspect the source code for this function (and many more!) by viewing the script in the USFS_GTAC/modules/getImagesLib.js script. Navigate to this script in your Scripts pane, click to open, and scroll or use CTRL+F to search for your function of interest.
5. Copy this line of code below the appropriate comment.

```
var TCall = getImages.getTasseledCap(composite);
```

6. Print the results of the function to the console to inspect the output. Copy this line of code below the appropriate comment.

```
print(TCall, "TCall");
```

7. Click run and toggle open the TCall object and inspect the bands present. Note that in addition to “brightness,” “greenness,” “wetness,” there are also bands titled “fourth,” “fifth,” and “sixth.”
   i. Here, we’re just interested in getting the Tasseled Cap bands alone so that we can combine them with all the other predictor layers later. Though, if we were only interested in the input spectral bands and the Tasseled Cap layers, we could select all of those layers in this step, rather than selecting only the Tasseled Cap layers and later adding them to the composite.
8. Select the Tasseled Cap transformation bands from the TCall object. Copy this line of code to the appropriate location in the script.

```
var TCbands = TCall.select("brightness", "greenness", "wetness");
```

B. Combine and visualize

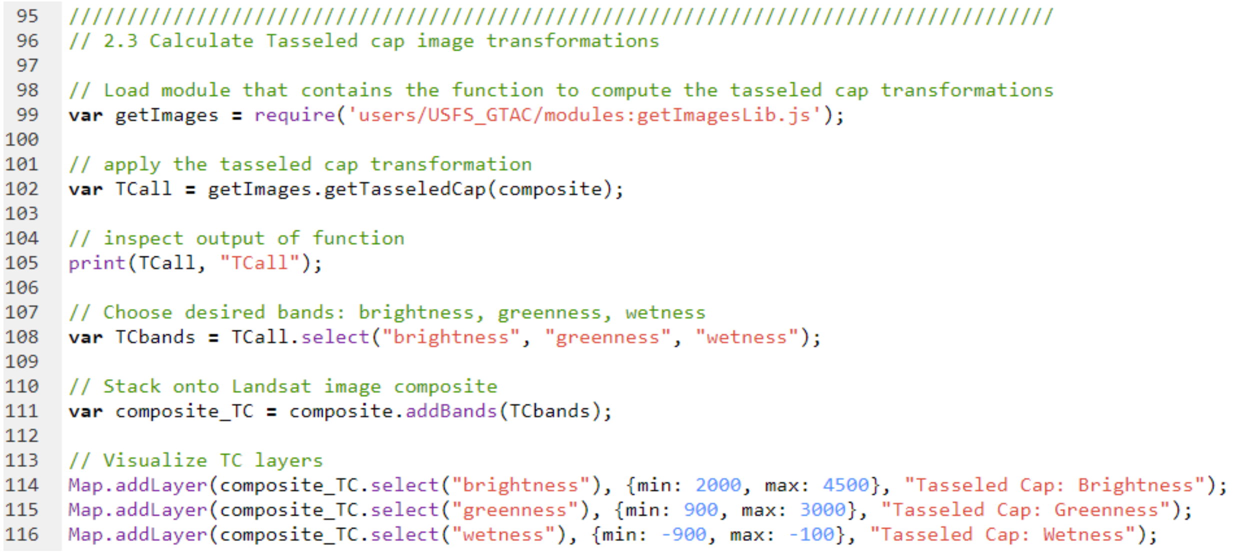
1. Add the bands to the composite, creating a new composite stack with the Tasseled Cap transformations. We are still using the **.addBands()** function, but are adding one image stack to another. Copy this line of code below the appropriate comment.

```
var composite_TC = composite.addBands(TCbands);
```

2. Now, we’ll add them to the map. Here, because the indices have varied ranges, we’re again specifying distinct minimum and maximum values for each one. Like the mineral indices, since we aren’t supplying any colors to the palette, these will be visualized in greyscale.

```
Map.addLayer(composite_TC.select("brightness"), {min: 2000, max: 4500}, "Tasseled Cap: Brightness");

Map.addLayer(composite_TC.select("greenness"), {min: 900, max: 3000}, "Tasseled Cap: Greenness");
Map.addLayer(composite_TC.select("wetness"), {min: -900, max: -100}, "Tasseled
Cap: Wetness");
```

3. Click **Save**. Your code should look like the image below.



```
/////////////////////////////////////////////////////////////////////////////////////
// 2.3 Calculate Tasseled cap image transformations

// Load module that contains the function to compute the tasseled cap transformations
var getImages = require('users/USFS_GTAC/modules:getImagesLib.js');

// apply the tasseled cap transformation
var TCall = getImages.getTasseledCap(composite);

// inspect output of function
print(TCall, "TCall");

// Choose desired bands: brightness, greenness, wetness
var TCbands = TCall.select("brightness", "greenness", "wetness");

// Stack onto Landsat image composite
var composite_TC = composite.addBands(TCbands);

// Visualize TC layers
Map.addLayer(composite_TC.select("brightness"), {min: 2000, max: 4500}, "Tasseled Cap: Brightness");
Map.addLayer(composite_TC.select("greenness"), {min: 900, max: 3000}, "Tasseled Cap: Greenness");
Map.addLayer(composite_TC.select("wetness"), {min: -900, max: -100}, "Tasseled Cap: Wetness");
```

4. Click **Run**. Now, you should see all the predictor layers that we calculated added to the map. Again, toggle the different layers on and off to inspect, compare, and verify that there is data in all of them.

Part 5: Combine layers

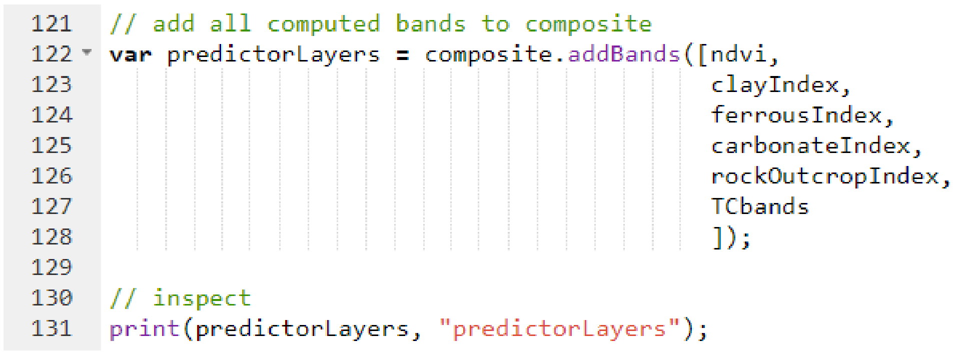
1. We've calculated three different types of spectral indices and band combinations that we can use as inputs to the classifier. The last step is to combine them into one image stack so that we can use them as predictor layers together.
2. We're using the **.addBands()** command again. Copy the lines below to the appropriate location.

```
var predictorLayers = composite.addBands([ndvi,
                                          clayIndex,
                                          ferrousIndex,
                                          carbonateIndex,
                                          rockOutcropIndex,
                                          TCbands
                                          ]);
```

3. Print to the console to inspect the final output.

```
print(predictorLayers, "predictorLayers");
```

4. Your finished code for this section should look like this.



```
// add all computed bands to composite
var predictorLayers = composite.addBands([ndvi,
                                          clayIndex,
                                          ferrousIndex,
                                          carbonateIndex,
                                          rockOutcropIndex,
                                          TCbands
                                          ]);

// inspect
print(predictorLayers, "predictorLayers");
```

5. **Save** your script and click **run**. See that we have added all the predictor layers individually to the map, then combined them with the original composite in one collected ImageCollection.
6. Go back through the script and add your own comments to anything that is unfamiliar, or that you would like to clarify for yourself.

Congratulations! You have successfully completed this exercise. You have used a variety of techniques to calculate spectral indices in Google Earth Engine.