Hands-On Introduction to Earth Observation Applications

Activity Guide

Nuno Carvalho

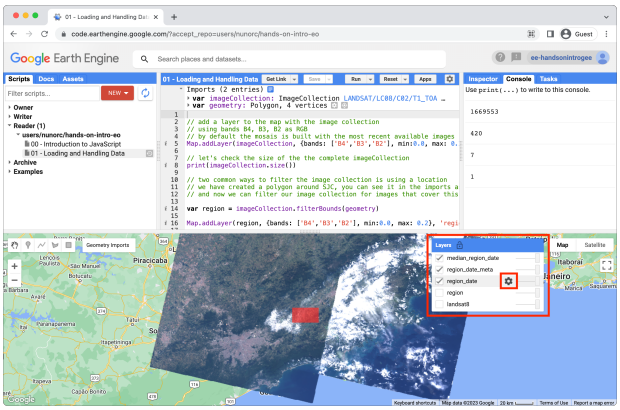
<nuno.carvalho@live.isunet.edu>

v0.1 | Last updated: July 25, 2023

Abstract

This document is the companion for the hands-on activity in the context of the ”Hands-On Introduction to Earth Observation Applications” workshop. Google Earth Engine is used for this activity, a cloud-based web application for geospatial analysis that enables users to visualize and analyze satellite images of our planet. The first section of the guide illustrates setting up and how to get started, no software or data downloads are required, every activity is done using a web browser.

This document is organized as follows: Section 1 illustrates how to get started with Google Earth Engine (GEE); Section 2 gives an overview of the Earth Engine Code Editor (EECE); Section 3 gives a brief introduction to JavaSript (JS), the programming language used by the engine; Section 4 illustrates some of the available datasets and how to load data; Section 5 discusses the first example application using GEE to quantify green vegetation; and, Section 6 illustrates the second example application for land classification. In the end of each section some hands-on activities are suggested to experiment with the corresponding discussed topics.



Contents

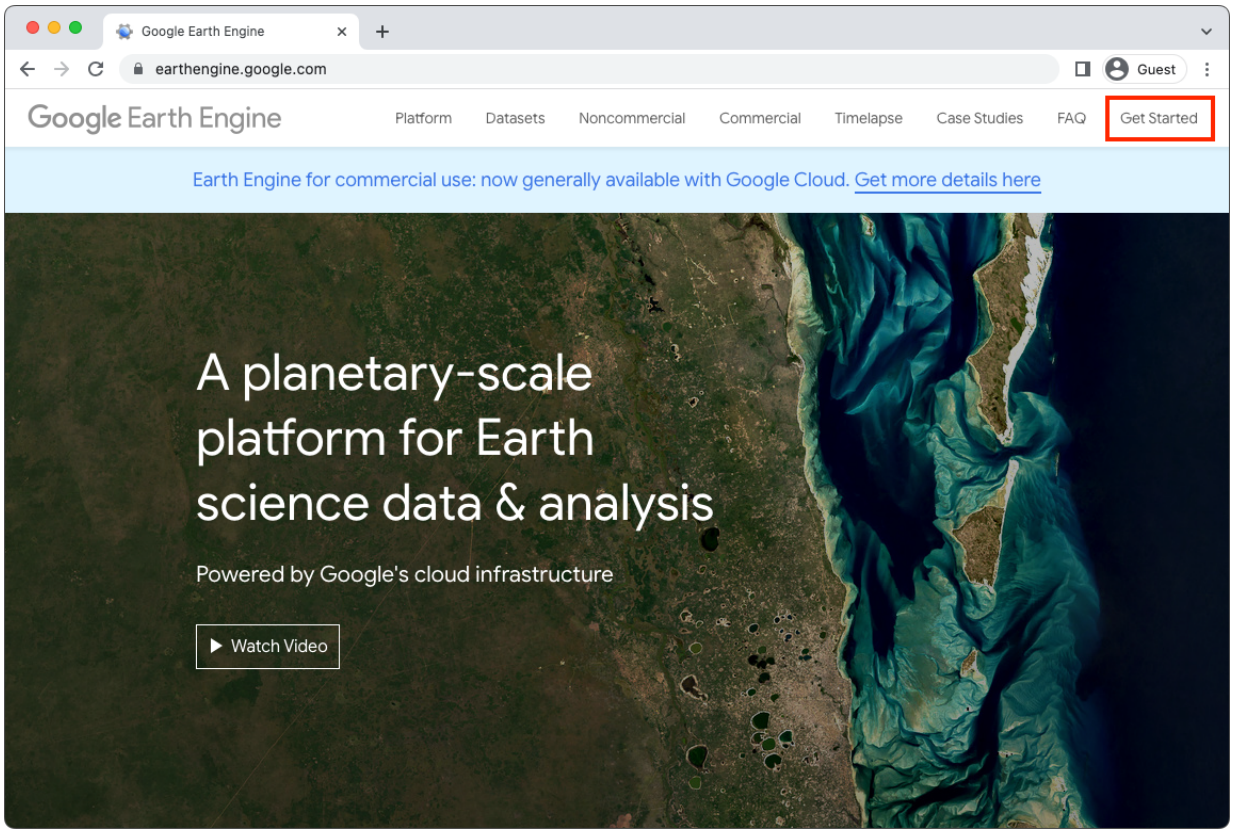


Figure 1: GEE landing page.

1 Getting Started

The first step is to visit the GEE homepage, illustrated in Figure 1, available from:

https://earthengine.google.com

and follow the "Get Started" link. A Google Account is required, which you are immediately prompted for after following the link. You can login with your own Google account if you have one or create a new account. After logging in with your own account or a newly created account, we can select with account to use with GEE, illustrated in Figure 2. Once we select the Google account to use, we need to setup a new project, the process starts after a Google account is selected, by following the corresponding link, illustrated in Figure 3. We select to use the GEE as an unpaid service, and set the project type to "Trainer & trainees" for the specific case of this workshop, as illustrated in Figure 4 and follow the "NEXT" button. The next step is to register a new "Google Cloud Project", we can just create a new project, with "No organization" selected, and give it a identifier and a name, an example is illustrated in Figure 5. If you have previous projects available you can also select to use one of them. Depending on if you created a new Google account, or already using a previous account you may getting the error illustrated in Figure 6, in this case just follow the "Cloud Terms of Services" link, accept the terms and follow the "Agree and Continue" button as illustrated in Figure 7. If you have any concerns make sure to read through the terms of service. The last step is to confirm the select project, by following the "Confirm and Continue" button as illustrated in Figure 8. Finally you are redirected to the EECE page, where you have the option to take an automatic tour of the editor.

Activity: follow the illustrated procedure to set up a Google account to use Google Earth Engine tool, you should be able to access the Earth Engine Code Editor.

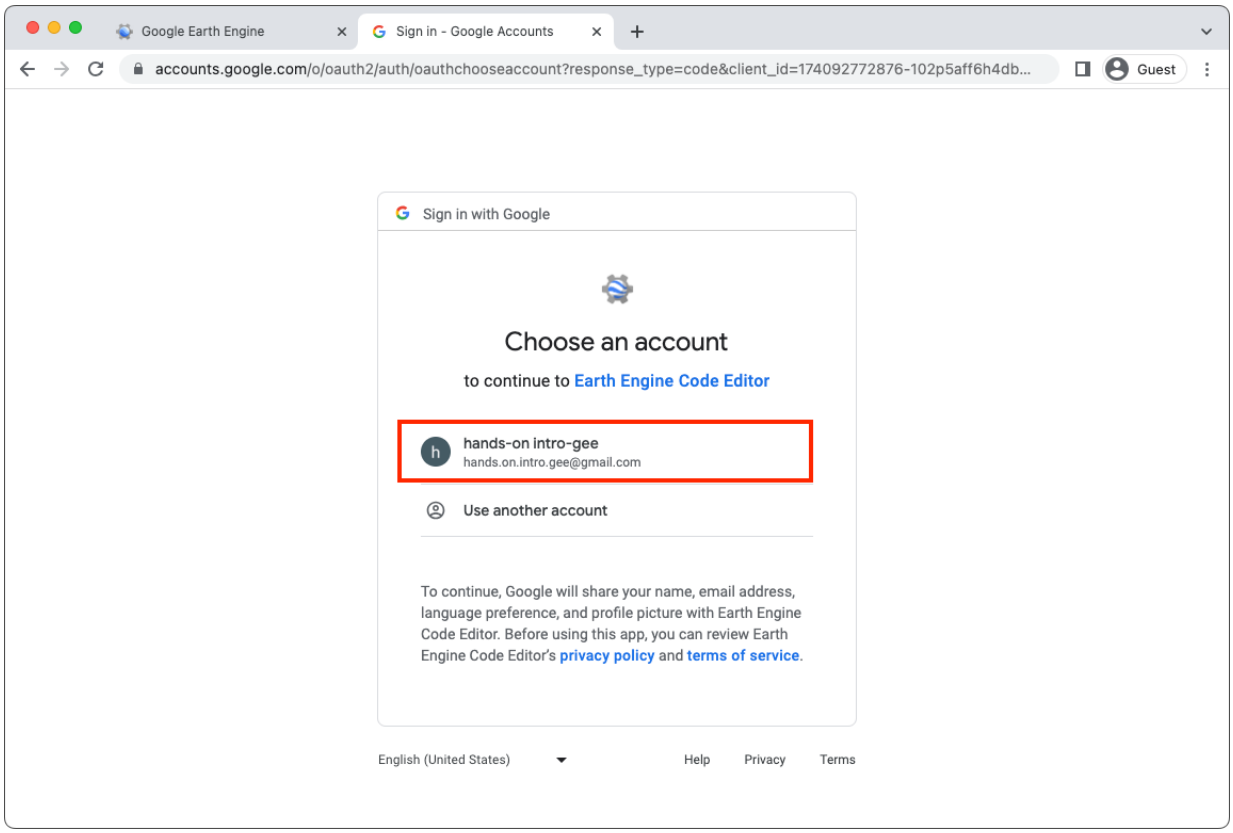


Figure 2: Select Google account.

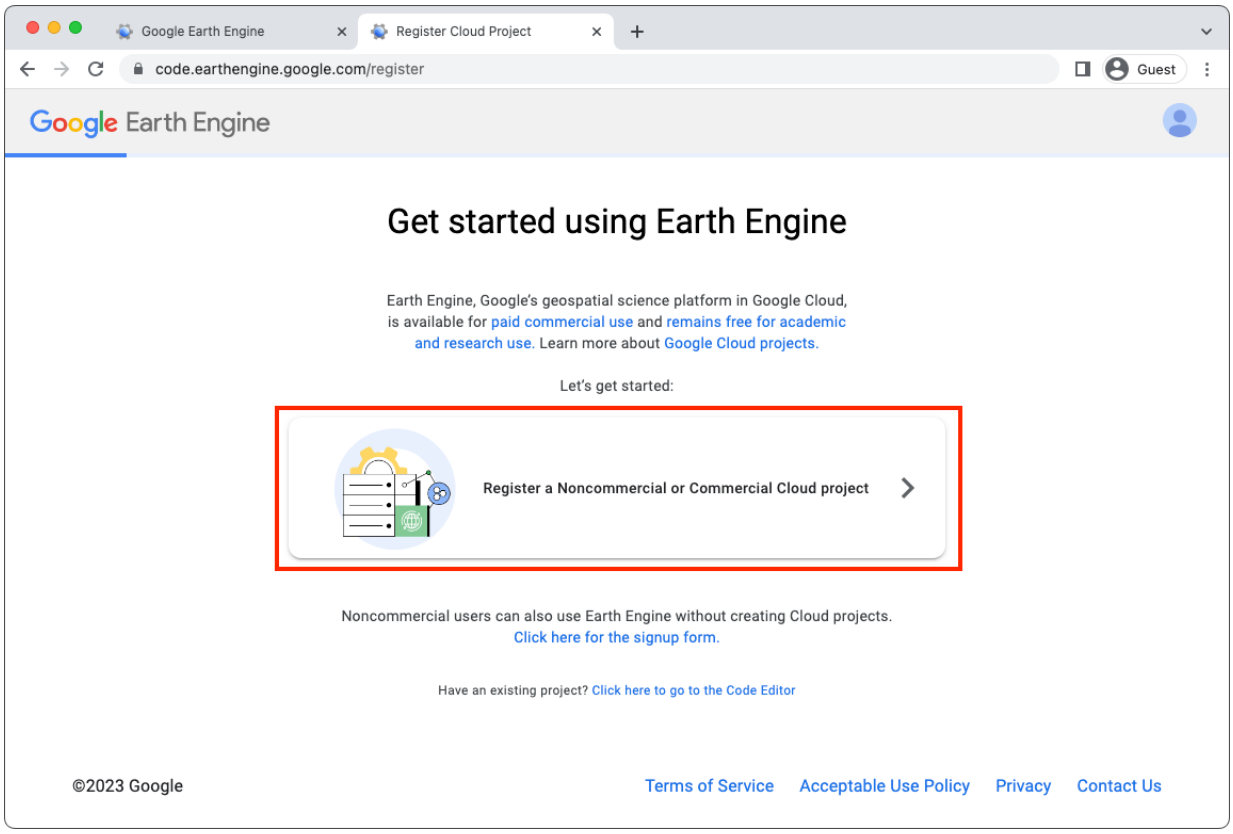


Figure 3: Register a new project.

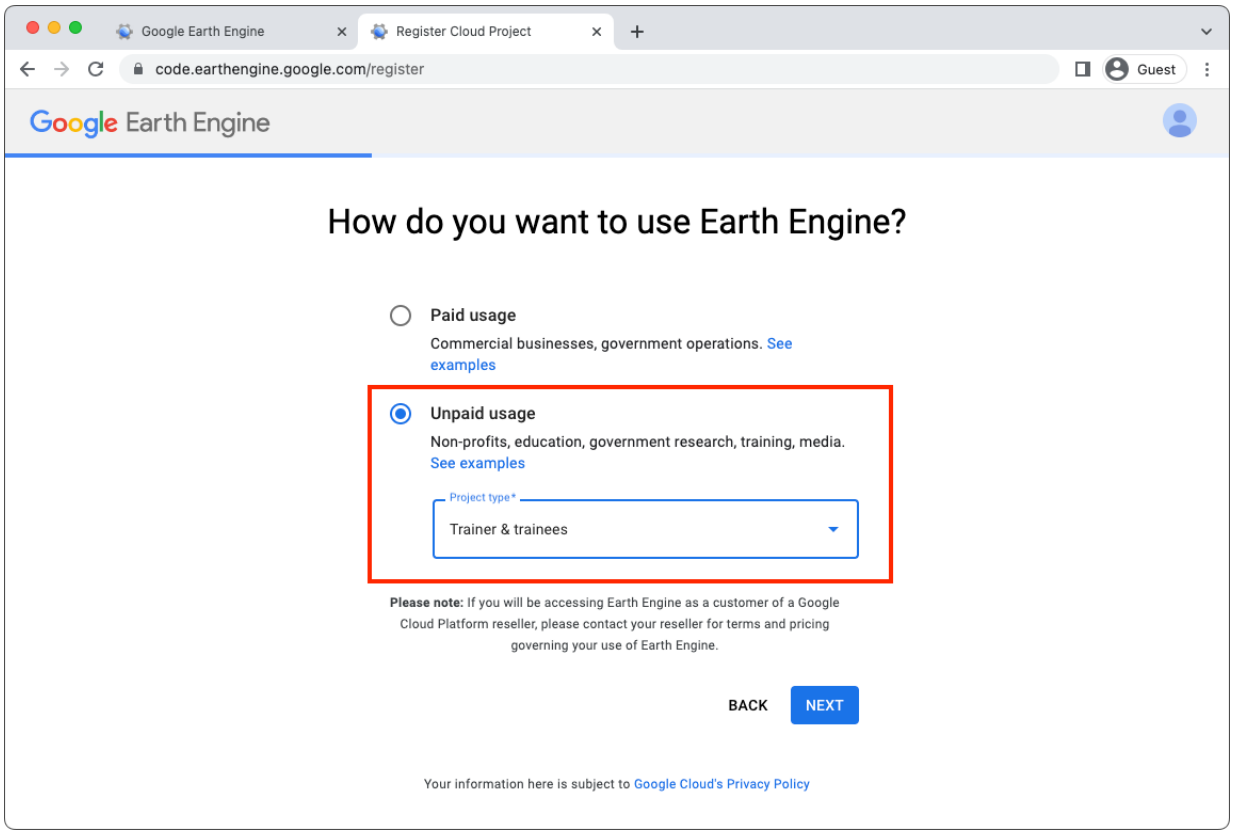


Figure 4: Select how to use GEE

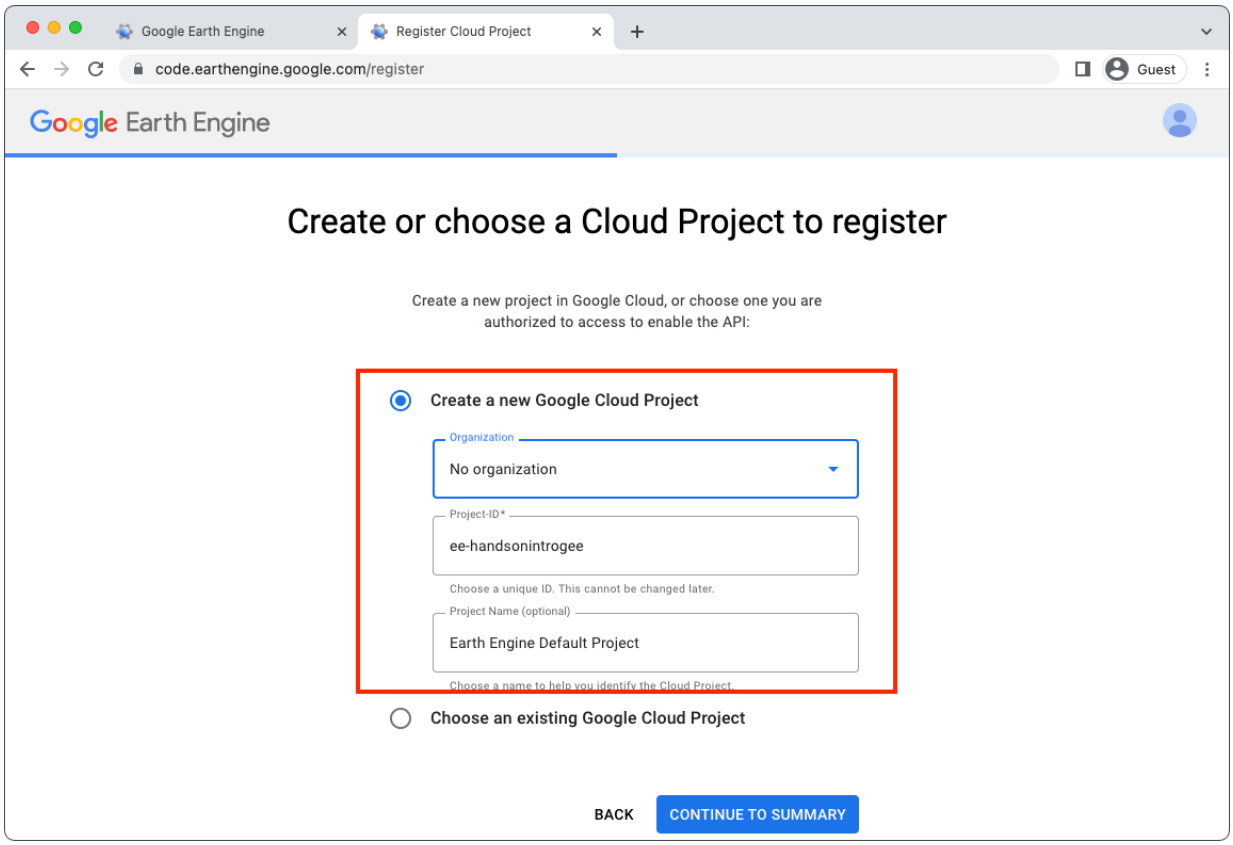


Figure 5: Create or select a project to use the cloud service.

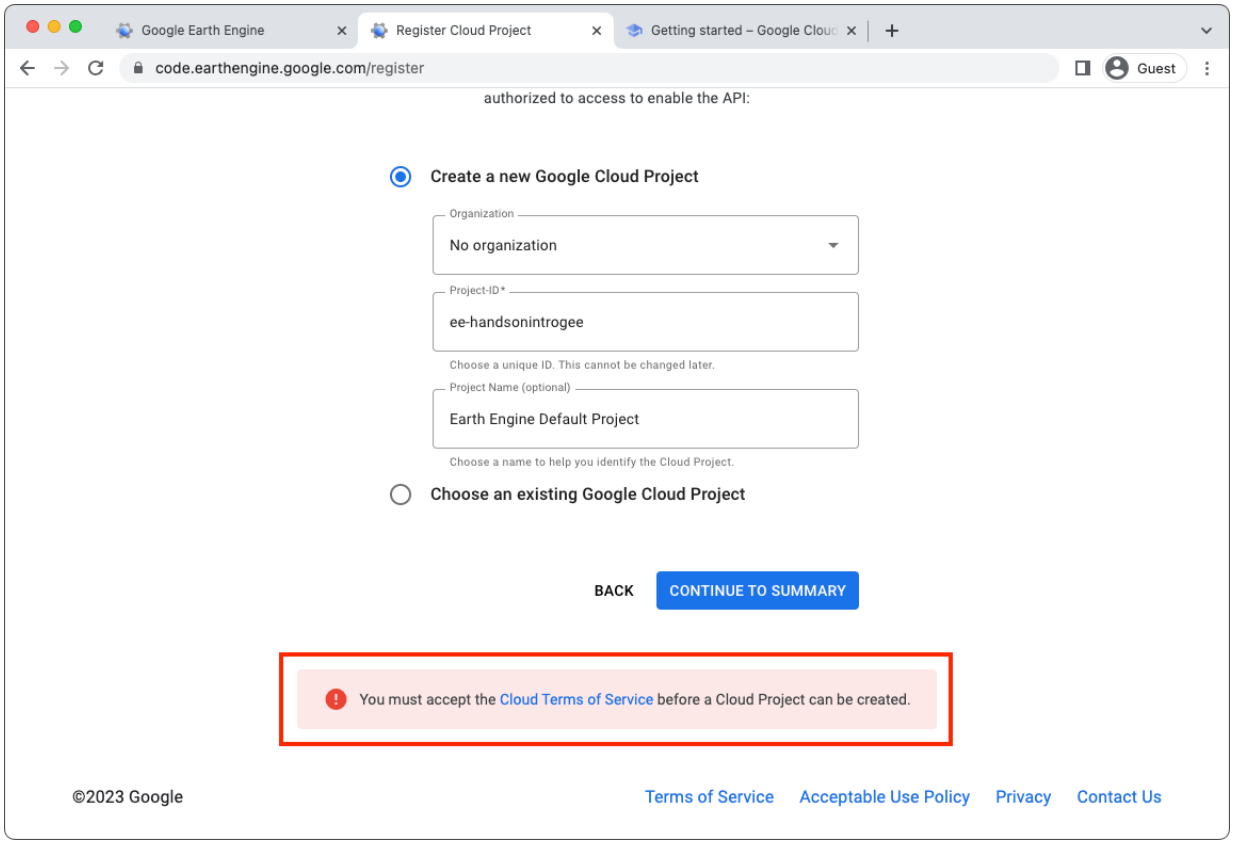


Figure 6: Possible alert to accept the terms of service.

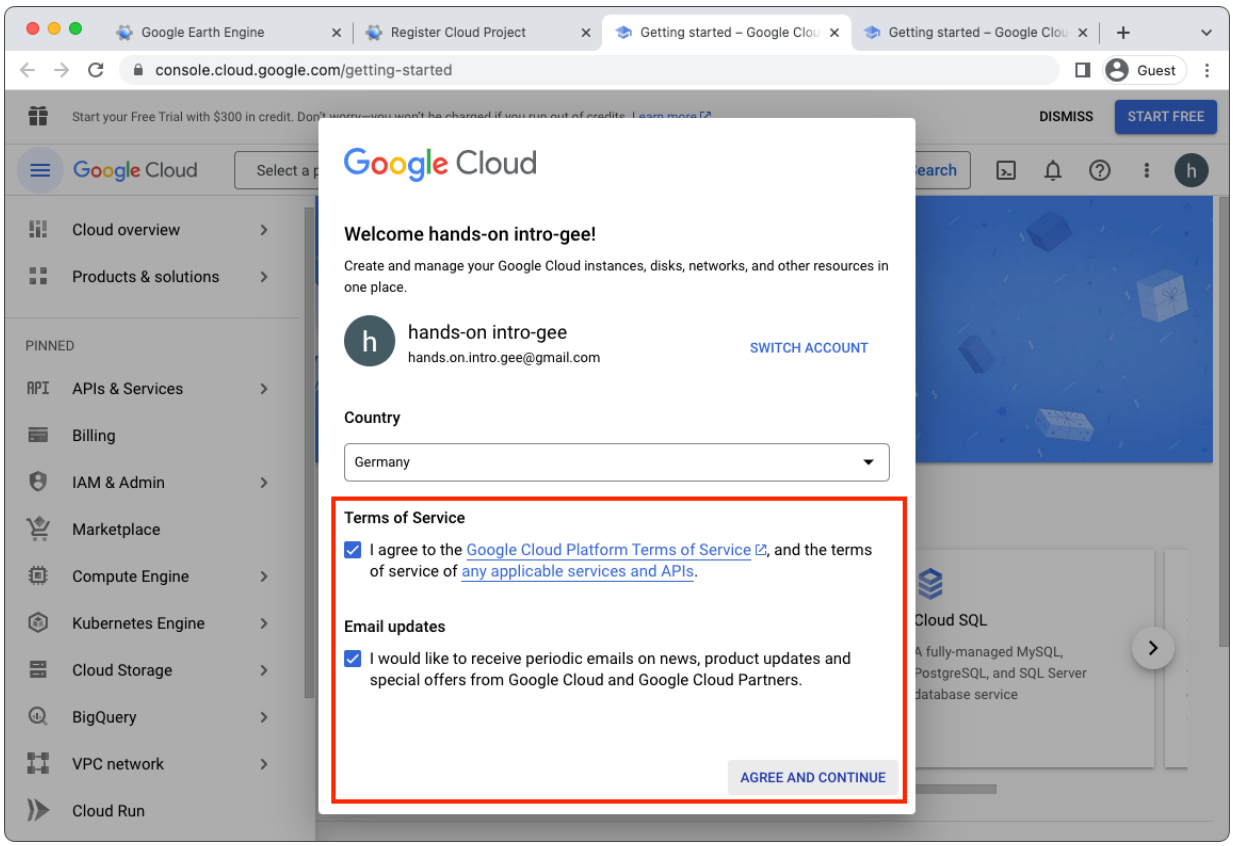


Figure 7: Terms of service acceptance screen.

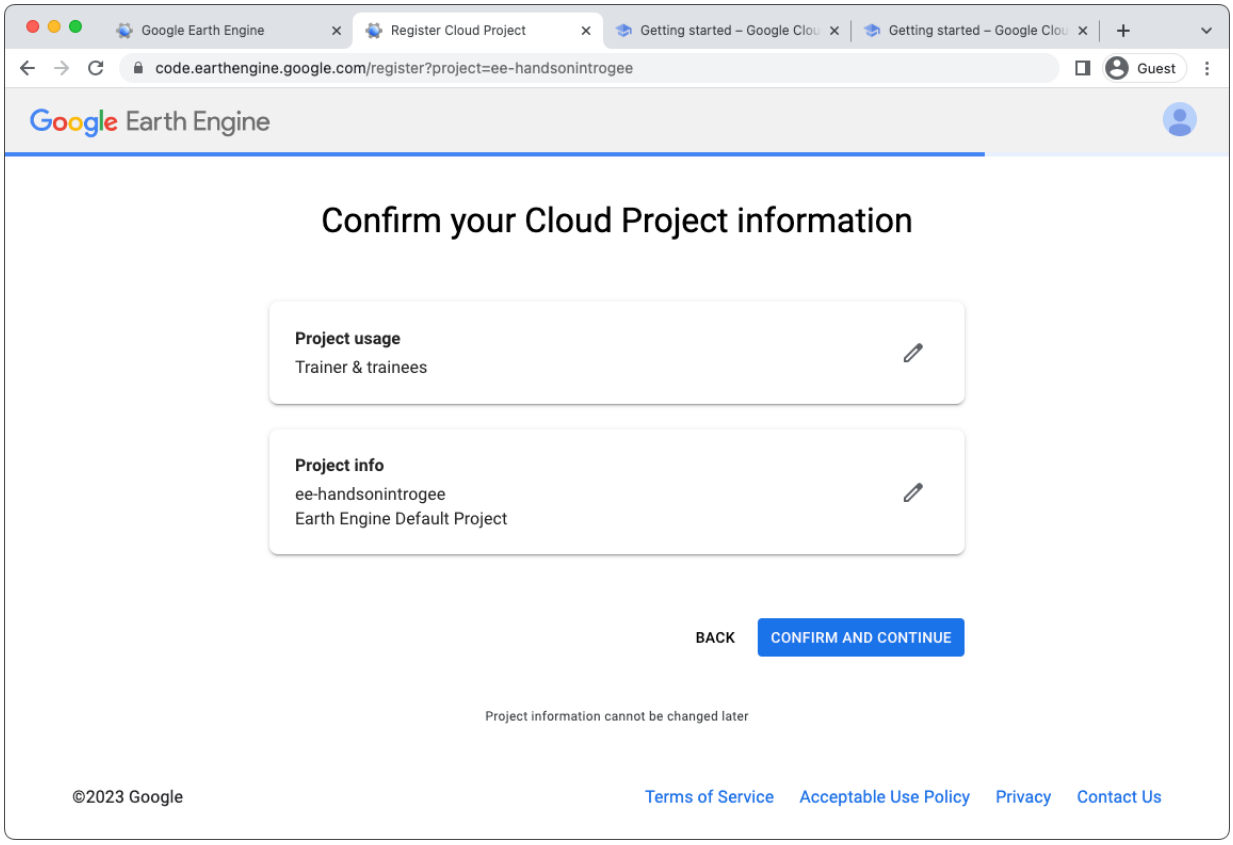


Figure 8: Confirm project information.

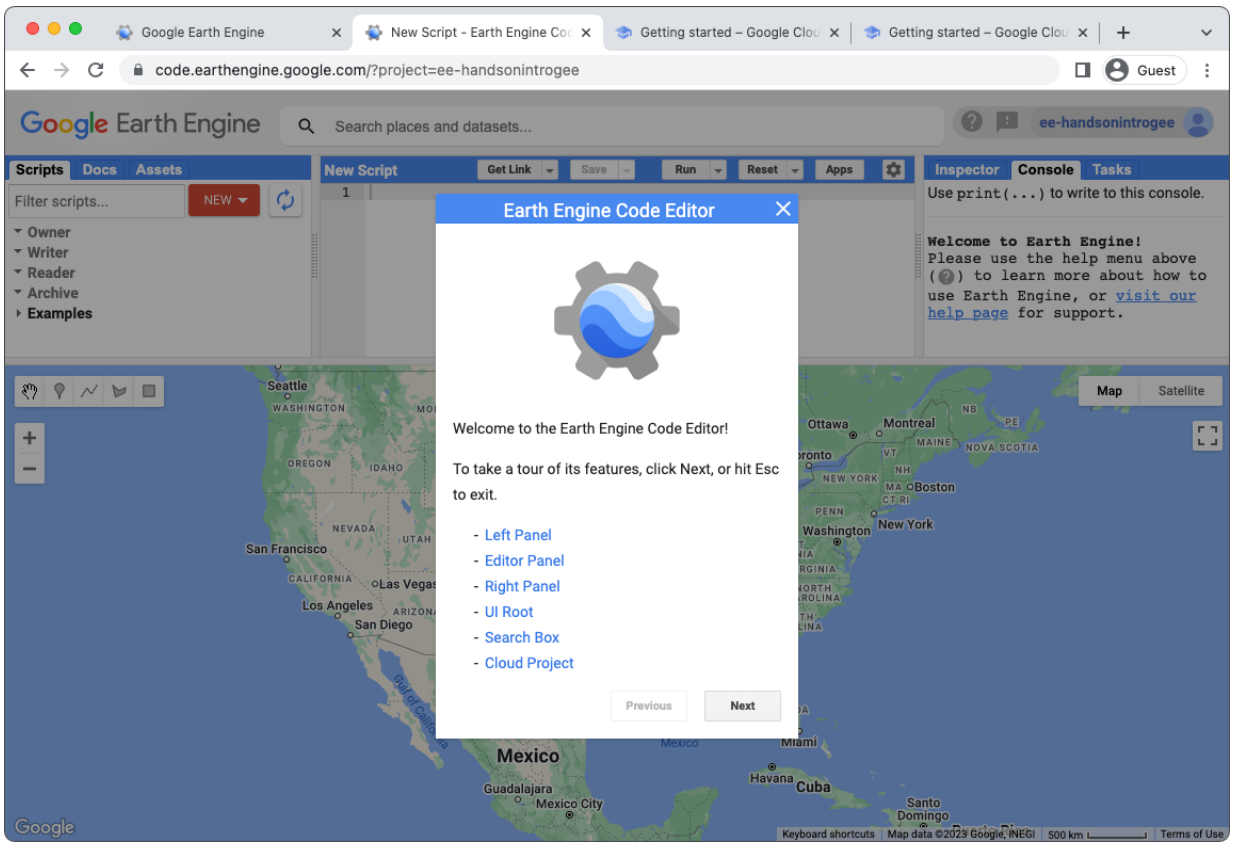


Figure 9: EECE landing page and tour.

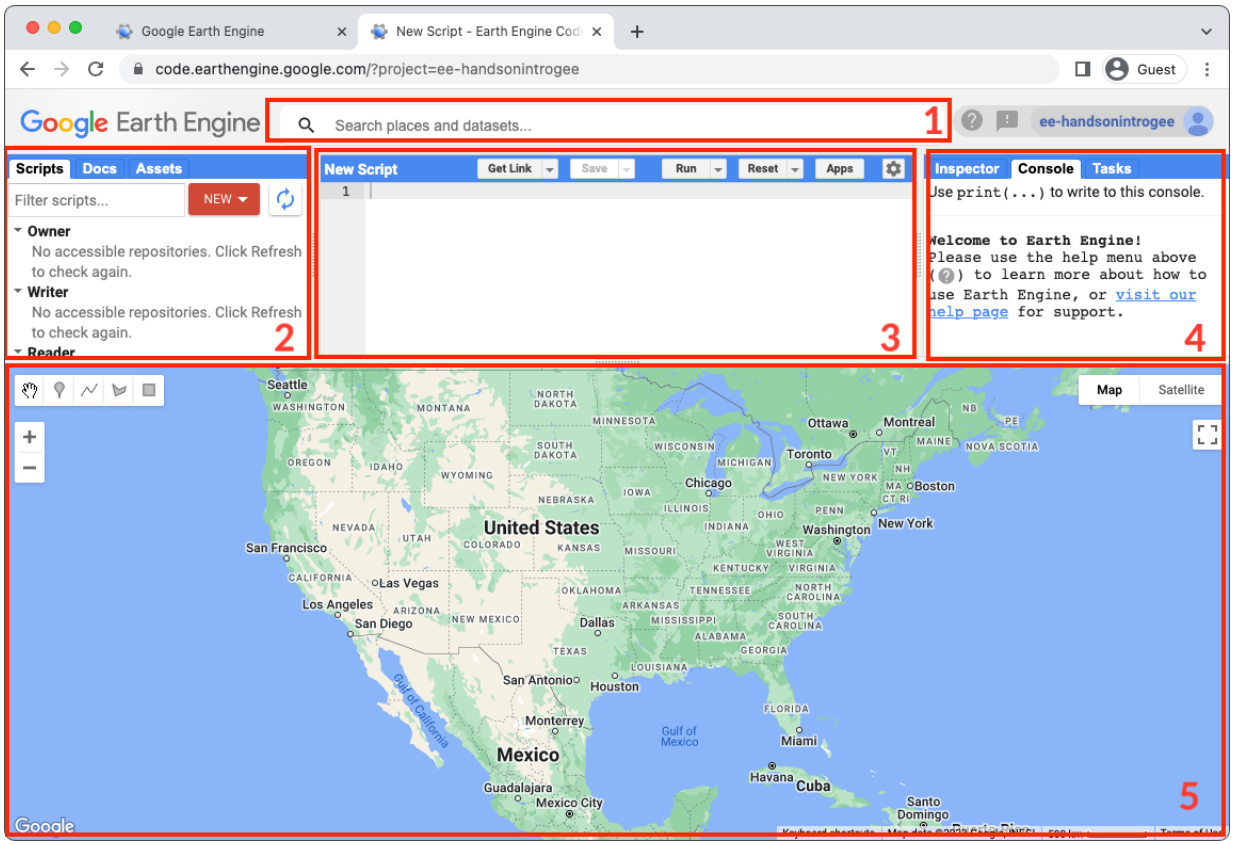


Figure 10: EECE overview.

2 Earth Engine Code Editor Overview

The Earth Engine Code Editor (EECE) is the main interface via the web browser, it allows access to all the data in the archive and different kinds of operations, and is illustrated in Figure 10. The red box outlines illustrate the major sections of the interface:

Section 1. includes the search bar that allows searching for places to show in the map and datasets to use;

Section 2. allows access to the script manager where we can store select scripts to load, access all the documentation and a list of assets that we can import for example;

Section 3. is the main script editor window for editing scripts, including buttons for some common tasks like running or sharing the code;

Section 4. illustrates the inspector that can be used to show information about any point on the map, and also gives access to the JS console and the tasks manager;

Section 5. shows the map window, including some common visualization options and actions, it is also used to show the data layers available to display.

> **Activity:** spend some minutes going through the introduced sections, windows and corresponding tabs in order to get familiar with the interface.

> **Activity:** once you feel familiar with the interface, follow the URL below, that will add to your scripts tab a copy of the repository that includes the scripts used in the remaining of this guide, the result is illustrated in Figure 11.
>
> https://code.earthengine.google.com/?accept_repo=users/nunorc/hands-on-intro-eo

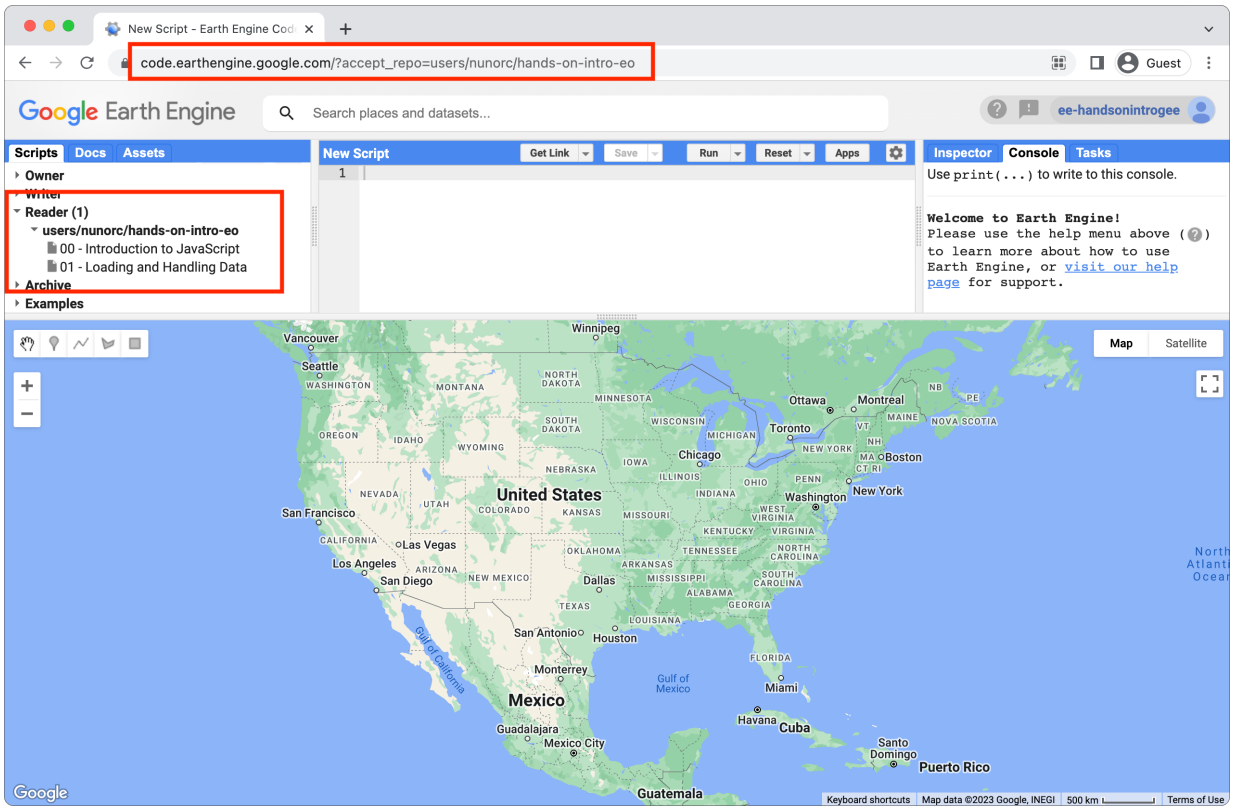


Figure 11: Add activity guide repository.

3 Introduction to JavaScript

JavaScript is the programming language that is used by the EECE to write scripts. JS is a versatile programming language primarily used for developing interactive web applications. It enables dynamic functionality and enhances user experience on the client-side of web development. This section quickly introduces some of the features of the language and its' syntax used by the example scripts in later sections.

Comments in are non-executable lines of code used to add explanations, notes, or disable code temporarily. They are ignored by the JavaScript interpreter and serve as helpful documentation for developers. In the following example, the text following the double forward slashes `//` is treated as a comment, i.e. It is ignored by the JS interpreter:

```
// this is a single line comment
```

In the next example, the comment is enclosed between `/*` and `*/`, allowing it to span multiple lines:

```
/*
 this is a multi-line
 comment.
*/
```

Variables are containers that hold values, allowing data to be stored, manipulated, and accessed throughout the program. The keyword `var` is used to create new variables, and the attribution operator = can be used to initialize its' value, for example to create a new variable named `number` and set it's initial value to `10`:

```
var number = 10
```

Lists, also known as arrays, are ordered collections of values that can hold multiple elements, including numbers, strings, objects, or even other arrays. They provide a versatile way to store and access multiple related data items within a single variable. In the following example, the variable `fruits` represents and array that holds three different fruits, each of one stored as an element of the array:

```
var fruits = ['apple', 'orange', 'banana']
```

Individual elements can be accessed using index positions, starting from 0. For example, `fruits[0]` retrieves the first element of the array, which is `apple`:

```
fruits[0]  // apple
fruits[1]  // orange
fruits[2]  // banana
```

Dictionaries, also known as objects, are key-value pairs that allow you to store and retrieve data using descriptive keys instead of numeric indices. In the following example, the variable `person` represents an object that holds information about a person. The keys (e.g., `"name"`, `"age"`, `"occupation"`) are used to label and access the corresponding values (e.g., `"John"`, 30, `"Engineer"`) within the object. In this case, `person["name"]` retrieves the value associated with the key `"name"`, which is `"John"`

```
var person = { name: "John", age: 30, occupation: "Engineer" }

person["name"]  // accessing the value associated with the key "name" ("John")
```

Functions are blocks of reusable code that perform specific tasks when invoked and are defined using the `function` keyword. They can accept inputs, called parameters, and return outputs using a `return` statement. For example, the following `addNumbers` function takes two parameters (`a` and `b`) and returns their sum using the `return` statement.

```
function addNumbers(a, b) {
  return a + b;
}
```

The function can be invoked by passing arguments the numbers 5 and 3, and the returned value is stored in the `result` variable:

```
var result = addNumbers(3, 5)

print(result)  // 8
```

Functions provide a way to write modular and reusable code, while improve readability.

> **Activity:** run and go through the "00 - Introduction to JavaScript" script available from the repository to get familiar with some JS, by executing the code and inspecting the output in the console.

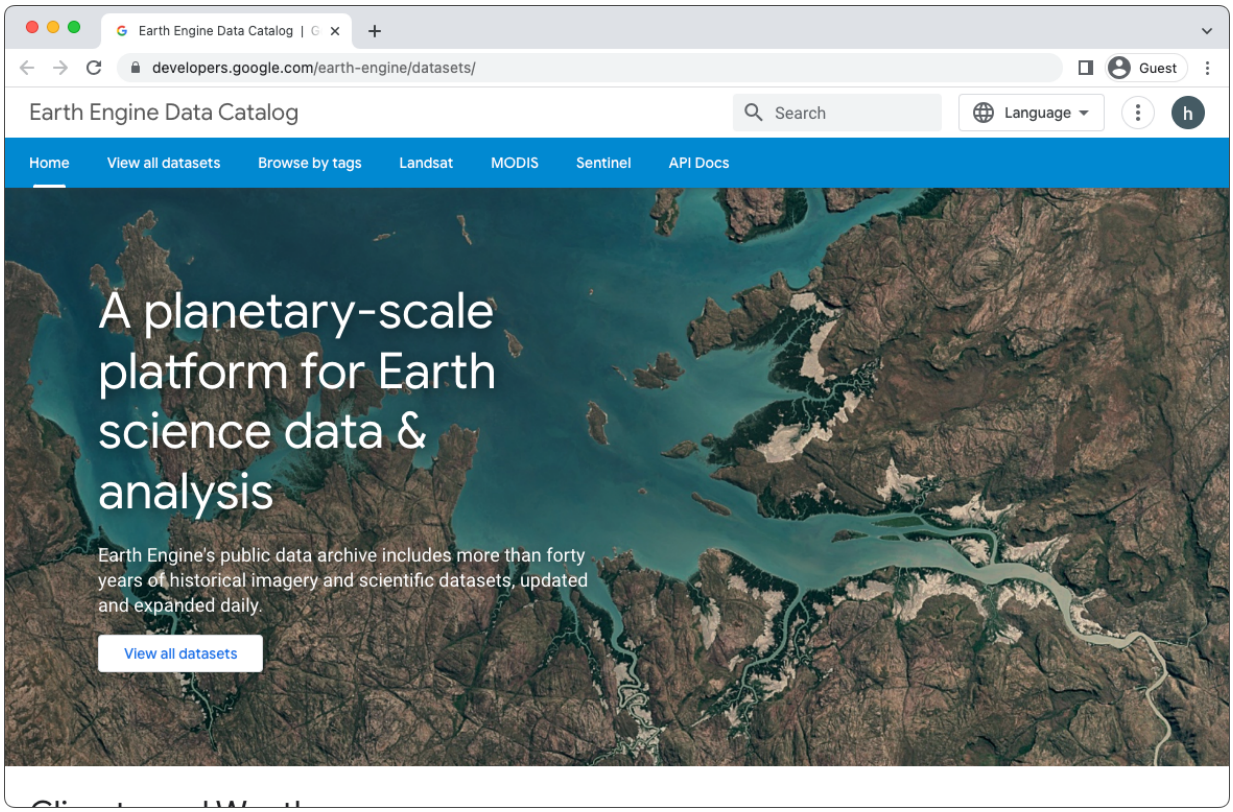


Figure 12: EECE landing page and tour.

4 Datasets Overview

The Earth Engine Data Catalog, illustrated in Figure 12 is available from:

https://developers.google.com/earth-engine/datasets/

or by following the "Datasets" link from from the top navigation bar in the GEE homepage. The platform provides a wide range of datasets from various sources, making it a powerful tool for conducting geospatial analysis and environmental monitoring. These datasets include satellite imagery, climate and weather data, land cover and use, elevation data and more. All of the datasets can be immediately loaded from the code editor and manipulated in some different ways.

> **Activity:** spend some time quickly browsing the available datasets to get familiar with the available data and its' details.

To import a dataset to our current script there are different options. From the EECE we can use the search bar to search for an dataset by name, and follow the link to import the dataset into our script, as illustrated in Figure 13. We can also follow the link of the dataset full name to get more information for the corresponding image collection, this is illustrated in Figure 14. From the more information page we can also copy the dataset unique identifier to load it in a script or follow the import button to have the dataset automatically imported into our script. From the more information page we can also quickly have access to more information about the bands used by the related instruments.

There are different ways to manipulate the imported datasets, a common task is also to filter images, by region of interest or between specific dates. It is also possible filter the data by looking at some meta data value, the specific details of the meta information available for each image collection.

> **Activity:** run and go through the "01 - Loading and Handling Data" example script from the repository that illustrates how to load some image collections including some common filter operations. Remember to check the different image collections by selecting which layers to overlay on the map, and setting different values each layer specific options from the layers widget as illustrated in Figure 15.

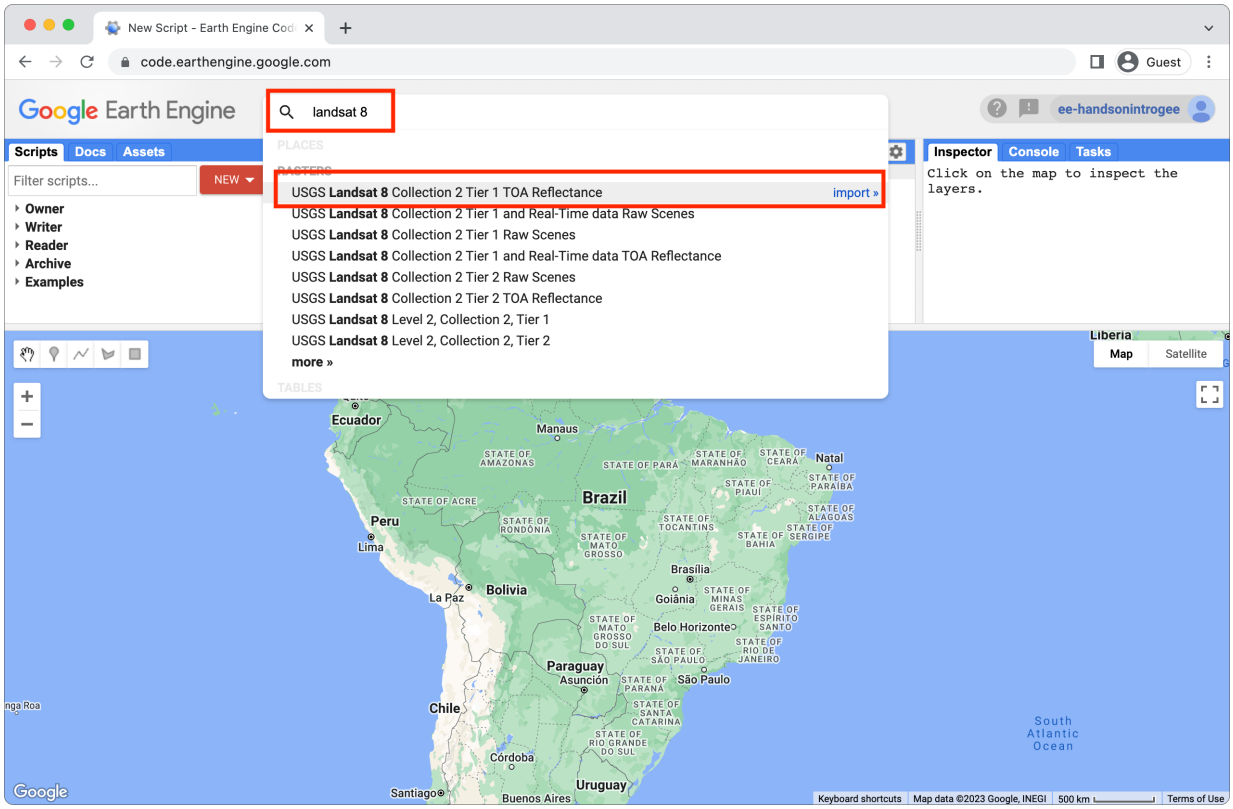


Figure 13: Searching for dataset in the EECE.

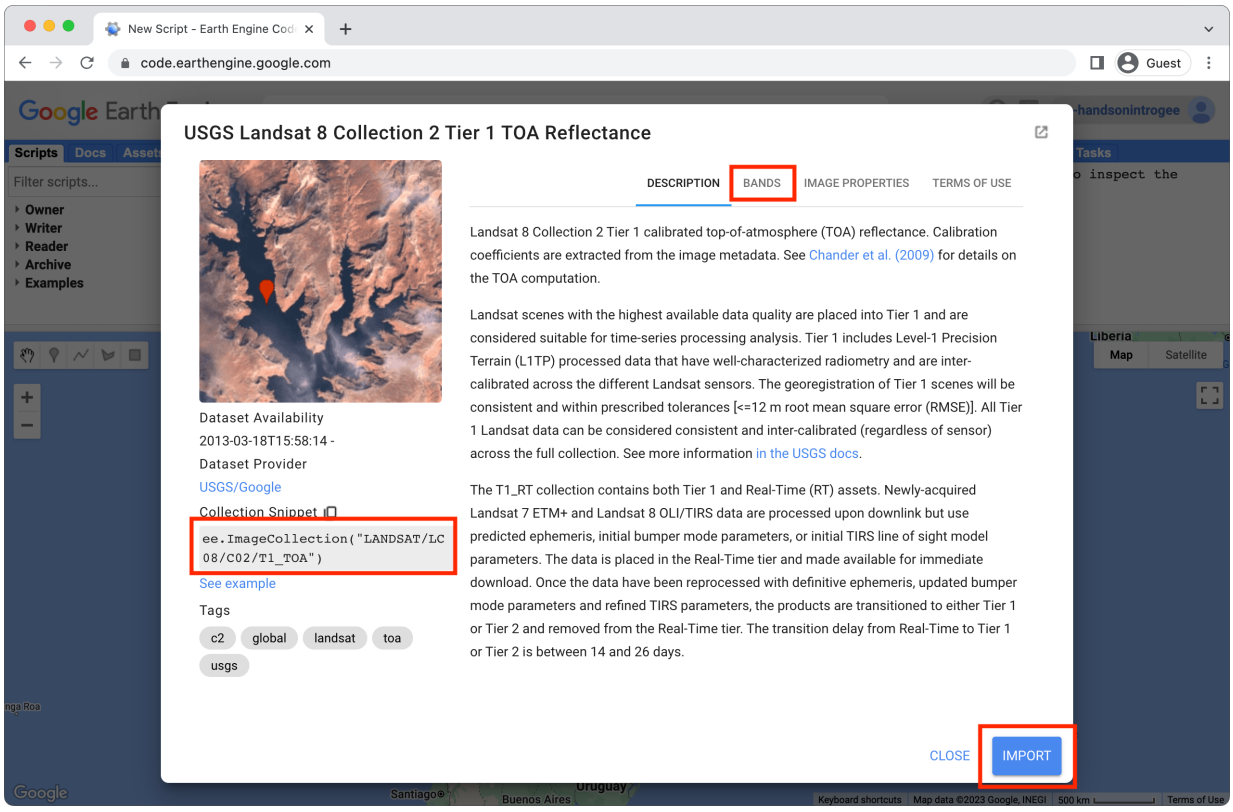


Figure 14: More information for a dataset from the EECE.

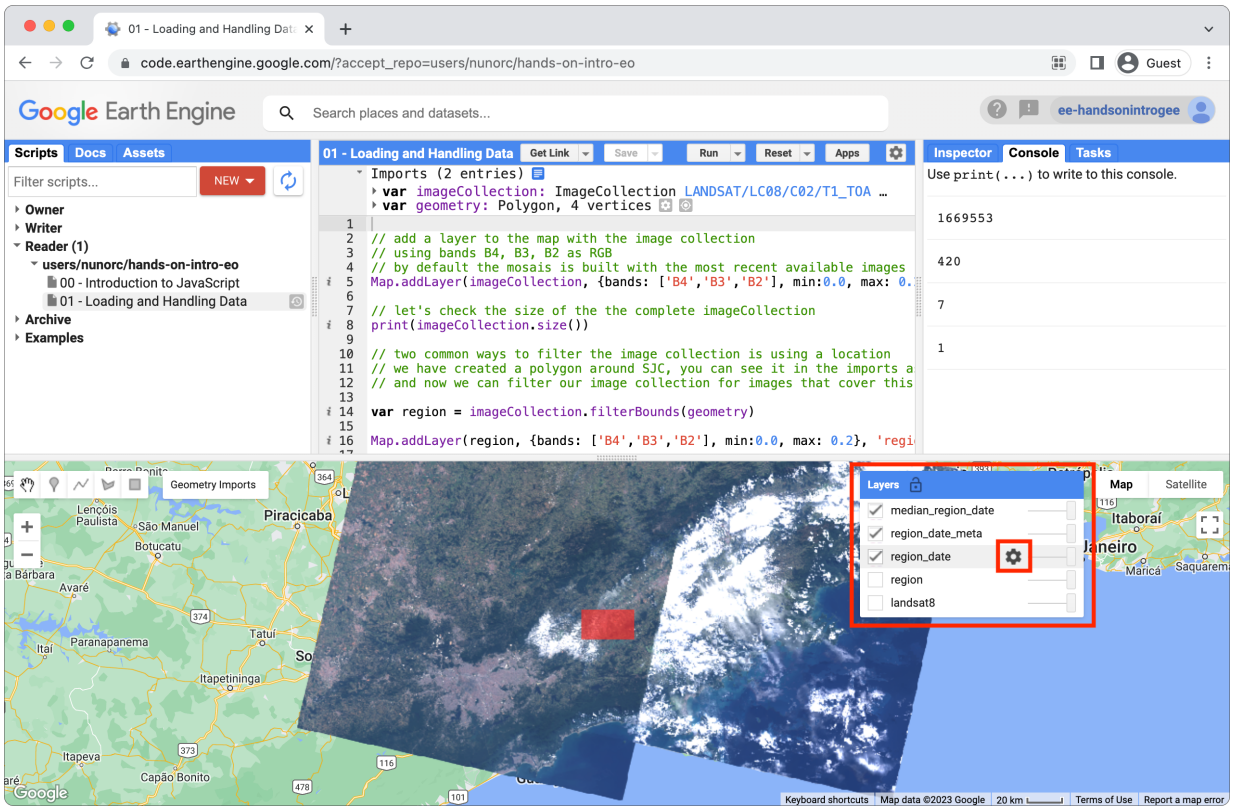


Figure 15: Layers widget and option to change layer overlay settings.

5 Application 1: Green Vegetation Quantification

The Normalized Difference Vegetation Index (NVDI) is a widely used vegetation quantification metric in the field of Earth Observation (EO). It allows to quantify and monitor the health and density of vegetation on the Earth's surface using remote sensing data, such as satellite imagery. The index is calculated from red band of the visible area of the spectrum (Red) and near-infrared (Nir) bands of the electromagnetic spectrum using the formula:

$$\text{NVDI} = \frac{\text{Red} - \text{Nir}}{\text{Red} + \text{Nir}}$$

The goal of this first application is to compute the NVDI index for an area of interest.

Activity: run the "03 - Application 1: NVDI" script to see how quickly we can calculate the NVDI for a specific area of interest, the result is illustrated in Figure 16. Also try to change the NVDI layer options from the layers widget to see the differences on the visualization. The script also exports the visualization of the data to Google Drive as an image, make sure to run the export task from the Tasks tab and check the image in your Google drive.

Activity: try to calculate the NVDI index for a different area of interest, by creating a new geometry in the map, and updating the code to use the new geometry.

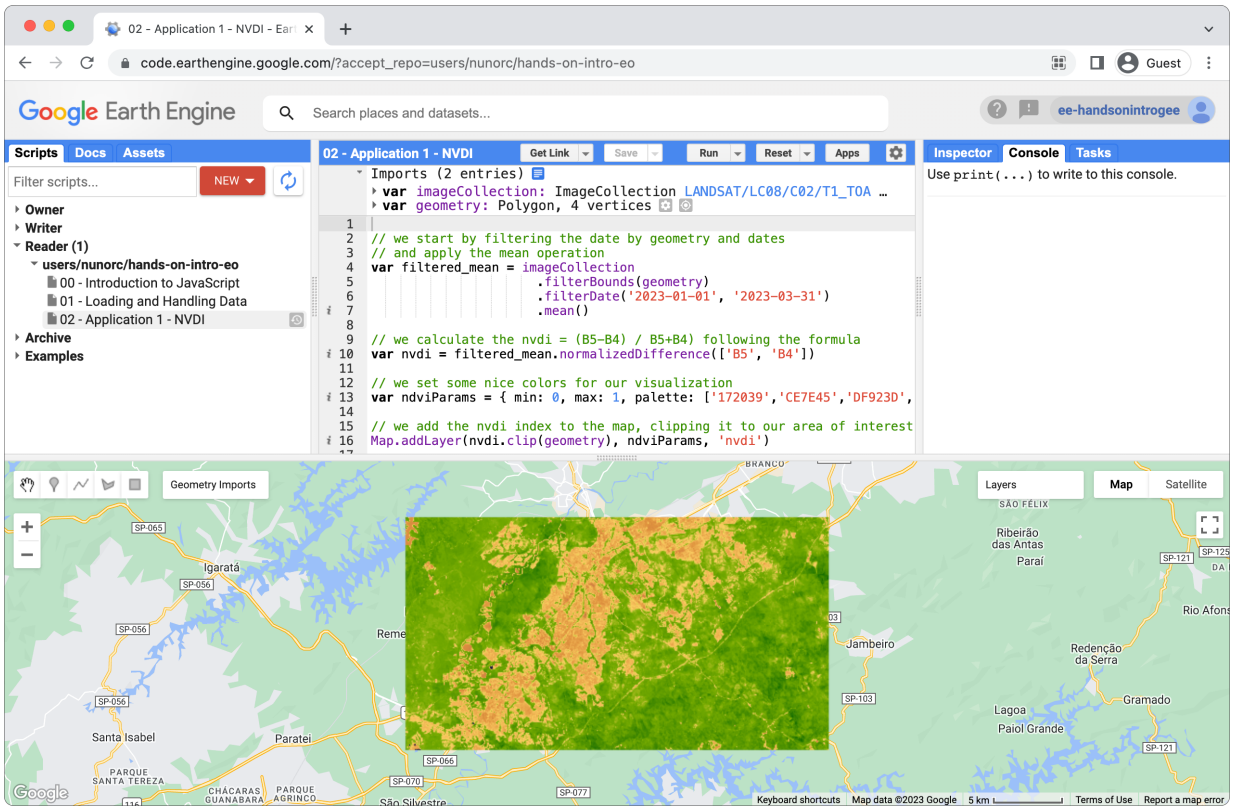


Figure 16: NVDI vegetation quantification for an area of interest.

6 Application 2: Urban Land Cover

For this example we use a supervised ML classification model, to try to do some land classification. The goal is to map the urban areas in our area of interest. We start by giving some information data to train the model, in particular we define some points for each class that we want to classify, in this example we are only using two classes: “urban” and “non_urban”. So, we need to define some points that describe both our classes, and then use the features from this points of interest to train a classification model. To manage the points of interest for each class we use the Geometry Imports widget, the “land_class” is a property from the feature collection that you can check from the properties button.

Activity: run the “03 - Application 2: Urban Cover” script to see how quickly we can calculate train a model to classify urban land cover in a map, the result is illustrated in Figure 17. Also try to change the classified land cover layer options from the layers widget to see the differences on the visualization.

Activity: try adding more points for training data to the current classes, and inspect the values for the bands at specific points. You can also try to add a new class to classify water for example, you need to create a new geometry feature collection to add more points with a given class, and then update the code to include this new feature.

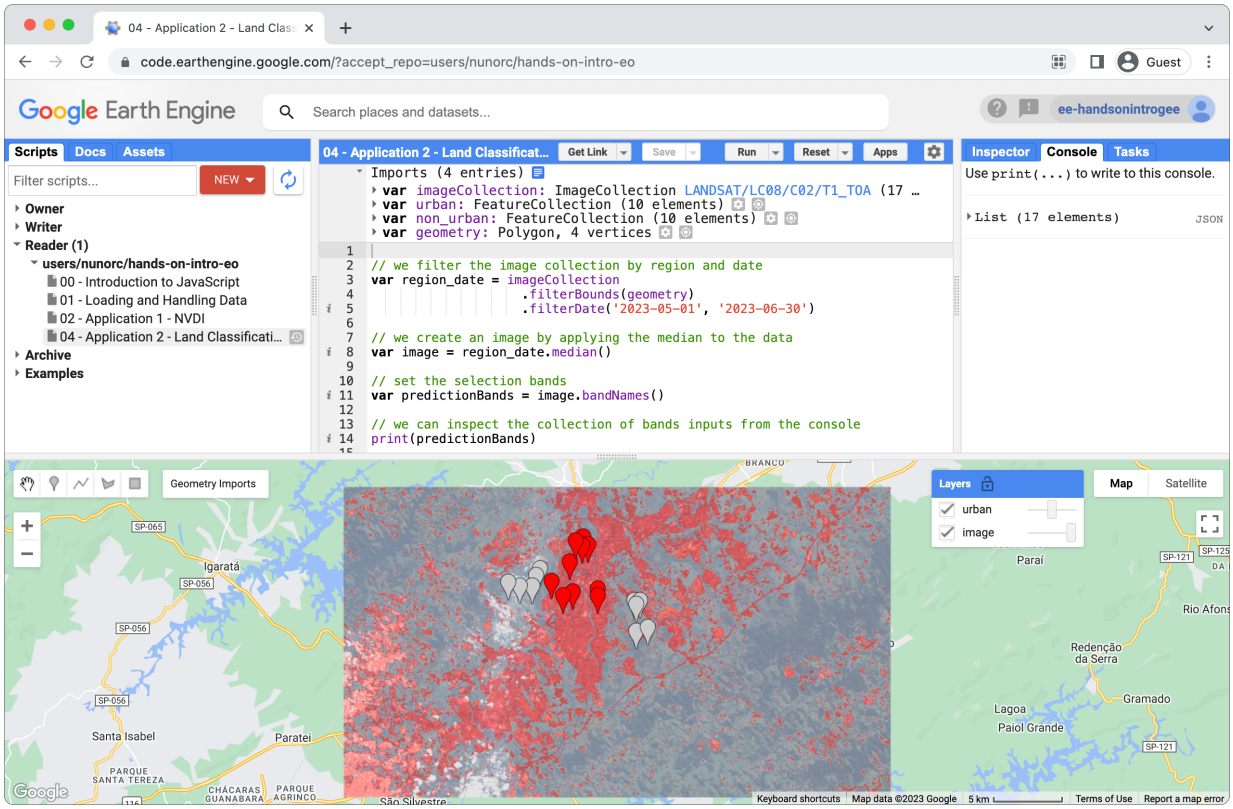


Figure 17: Urban land cover.