

# GEE manual

JVS GP JFM

## Índice

<b>Introducción</b>	<b>4</b>
¿Qué es Google Earth Engine? . . . . .	4
¿Cómo Funciona Google Earth Engine (GEE)? . . . . .	4
background . . . . .	4
Ventajas de la API de Java . . . . .	5
Lenguaje Javascript . . . . .	5
<b>Primeros pasos</b>	<b>7</b>
Registro para el uso de GEE Javascript API . . . . .	7
Elementos básicos de la interfaz gráfica . . . . .	7
<b>Tipos de objetos</b>	<b>9</b>
Server vs Client side . . . . .	9
Tipos de objetos del lado del cliente . . . . .	9
Strings (cadenas de texto) . . . . .	9
Number (números) . . . . .	9
Lists (listas) . . . . .	9
Diccionarios (Dictionaries) . . . . .	10
Funciones (Functions) . . . . .	10
Tipo de objetos del lado del servidor . . . . .	10
ee.String . . . . .	11
ee.List . . . . .	11
ee.Dictionary . . . . .	11
ee.Date . . . . .	12
ee.Image . . . . .	12
ee.ImageCollection . . . . .	12
ee.Feature . . . . .	12
ee.FeatureCollection . . . . .	13
ee.Algorithms . . . . .	13

ee.Array . . . . .	13
ee.Clusterer . . . . .	13
ee.Filter . . . . .	13
ee.Geometry . . . . .	13
ee.Join . . . . .	13
ee.Terrain . . . . .	14
<b>Interfaz de usuario: Display de datos, gráficos y exportación</b>	<b>15</b>
Map . . . . .	15
Map.addLayer . . . . .	15
Map.centerObject . . . . .	15
Map.add . . . . .	15
Chart . . . . .	15
Chart.array . . . . .	16
Chart.feature . . . . .	16
Chart.image . . . . .	16
Export . . . . .	16
Export.image . . . . .	16
Export.table . . . . .	17
Export.map . . . . .	17
Export.video . . . . .	17
Puntos para recordar . . . . .	17
<b>Features</b>	<b>19</b>
Trazado de polígonos, líneas, puntos . . . . .	19
<b>Feature Collections</b>	<b>20</b>
<b>Imagen</b>	<b>21</b>
Características . . . . .	21
Información y metadatos . . . . .	21
Operaciones básicas . . . . .	21
Operaciones matemáticas . . . . .	21
Operaciones relacionales, condicionales y booleanas . . . . .	22

<b>Colecciones de imágenes</b>	<b>23</b>
Características . . . . .	23
Información y metadatos . . . . .	23
Filtros sobre Colecciones de imágenes . . . . .	23
Ejercicio 1: Filtro de colección de imágenes y visualización . . . . .	24
Mapeo . . . . .	26
Reducción . . . . .	26
Compuestos y mosaicos . . . . .	27
Ejercicio 2: Enmascaramiento de nubes, cálculo de índices y reducción . . . . .	27
Exportación de resultados . . . . .	30
Ejercicio 3: Cálculo de diferencia de dos imágenes y exportación de resultado. . . . .	31
<b>Importación de información a GEE (Assets)</b>	<b>33</b>
<b>Clasificación supervisada</b>	<b>35</b>
Classifier . . . . .	35
Ejercicio 4: Clasificación . . . . .	35

# Introducción

## ¿Qué es Google Earth Engine?

Google Earth Engine (GEE) es una plataforma, desarrollada por Google, que permite realizar procesamientos geoespaciales a gran escala y utilizando bases de datos inmensas.

## ¿Cómo Funciona Google Earth Engine (GEE)?

### background

Desde hace varios años hay misiones espaciales tomando datos de la tierra a través de distintos satélites. Y toda esa información se ha ido almacenando en acervos de imágenes. Aunque esa información ha sido muy útil en el desarrollo científico, siempre había habido el problema de la “Big-Data” (bases de datos enormes). Es decir, dichos acervos de imágenes son de gran dimensión (millones de imágenes disponibles), lo cual resultaba imposible de procesar y analizar toda la información disponible, evitando así poder aprovechar el potencial total de esta información.

GEE nace entonces de la necesidad de lograr análisis de esa “Big-Data”, ofreciendo una herramienta que pueda lidar tecnológicamente con la parte más complicada y engorrosa del manejo de información, permitiendo entonces que los usuarios se concentren en la generación de resultados, y nuevos desarrollos científicos.

para lograr este objetivo, se creó una infraestructura con 3 elementos claves que son quienes permiten el funcionamiento de GEE: catálogo de información, capacidad computacional, APIs

**Catálogo de información** GEE, recopiló información geoespacial de diferentes fuentes alrededor del mundo, creó copias de esos datos en su propio datacenter logrando así almacenar más de 20 petabytes (20'000.000 de gigabytes) de información, en un solo lugar.

GEE ofrece un amplio repositorio de información geográfica global, cuya información ya se encuentra cargada en su catálogo, facilitando un sencillo acceso a diferentes Datasets. entre la información que se puede consultar directamente en GEE se encuentran: acervos para todo el mundo de imágenes Landsat (1-8), MODIS, Sentinel (1-3, 5), SRTM, AVHRR, GOESS, capas como el WorldClim, límites políticos, densidad poblacional, cuencas, WWF watersheds (cuencas), entre otras.



Para consultar toda la información que está disponible en GEE se puede consultar la siguiente liga: <https://developers.google.com/earth-engine/datasets>

**Capacidad computacional** Paralelamente GEE puso a disposición, junto al datacenter, unos servidores (los cuales en 2010 tenían una capacidad más o menos equivalente a 10.000 computadores personales), para poder realizar rápida y eficazmente cálculos y computaciones sobre esos datos

**API** GEE es una API (Application Program Interface), lo que básicamente significa, es una interfaz que permite la comunicación entre nosotros (usuarios) y los servidores de Google, proporcionando el acceso y uso de la capacidad computacional de Google para nuestros análisis. GEE tiene el objetivo de facilitar la creación de programas. Por ello, cuenta con una serie de funciones, métodos y algoritmos preprogramados que se pueden llamar con una simple línea.

GEE ofrece dos APIs una en JavaScript la cual se accede online, es la más conocida, más actualizada, más amigable con los usuarios, con más documentación disponible, y sobre la cual estaremos hablando; y una API

en Python, la cual se puede trabajar desde la consola de Pyhton y permite entonces -hasta cierto punto- usar complementariamente librerias de Python, permitiendo asi procesamientos mas complejos, o funcionalidades que la API de Java no permite. la APi de Pyhton tiene las desventajas que tiene menos documentacion, y tiene algunas diferencias (que pueden resultar poco intuitivas) a la hora de programar.

#####Grafico#####

## Ventajas de la API de Java

Entre las principales ventajas de utilizar esta API radica en:

1. Los datos pueden ser consultados directamente en la nube, por lo que no requieren ser descargados para trabajar con ellos (lo que ahorra tiempo y espacio de almacenamiento para el usuario). En cambio, si se quiere usar alguna libreria de pPython será necesaria la descarga de la informacion
2. El procesamiento se hace en la nube, a traves de internet, utilizando el poder de cómputo asignado para la API, lo cual reduce el gasto de memoria RAM (solo necesita una conexiaon estable y el consumo de ram depende del comsuno del navegador). en la API de Python hay mayor consumo de memoria RAM (para ejecutar la consola de python y sus librerias) y requiere igualmente una conexion estable a internet.
3. Su interfaz es mucho mas amigable con los ususarios, ofreciendo una plataforma mas interactiva para programar, y mas sencilla para enseñar
4. Se pueden realizar facilmente consultas de las colecciones de imagenes, y sus metadatos, antes de ecidir importarlos. la API de Pyhton no permite realizar ese tipo de consultas facilmente, de modo que tener previamente absoluta claridad sobre la colección de imagenes que se va a utilizar.
5. Otra gran ventaja de la plataforma radica en que los códigos se guardan en la sesión de cada usuario. Esto permite mantener un control de los códigos y automáticamente se genera un registro histórico (parecido al git), lo cual facilita el seguimiento y comparación de cambios entre las versiones (y permite revertir a versiones anteriores). Además, se pueden generar repositorios compartidos para generar proyectos colaborativos.

## Lenguaje Javascript

La API permite trabajar mediante la programación en lenguaje Javascript, en una interfaz gráfica que resulta más amigable con el usuario. Además, para utilizarla únicamente requiere de una conexión a internet y un navegador. Toda la información del usuario es guardada en la cuenta asociada de Google en dicha plataforma. Existen otras opciones para trabajar con GEE, como la API de Python, sin embargo, los autores consideramos que la API de Javascript resulta la más sencilla de utilizar y se encuentra mejor documentada.

La sintaxis de JavaScript tiene algunas peculiaridades que deben cumplirse para que se pueda correr el código sin problemas. Entre la sintaxis básica se pueden considerar los siguientes puntos:

1. Es un lenguaje sensible a mayúsculas y minúsculas. De tal manera que `a` puede ser un objeto y `A`, otro.
2. Siempre se debe cerrar cualquier comando con un `;`. (aunque de no hacerlo el codigo funcionara igualmente)
3. Comúnmente en Javascript se utiliza el *lower Camel Case* para unir palabras, en lugar de guiones o guión bajo. Por ejemplo: `intervalMean` o `updateMask`.
4. Todas las variables, funciones, objetos deben ser definidos mediante la función `var`.
5. Para definir variables se utiliza el operador `=`.
6. Los operadores matemáticos son: `+` `-` `*` `/`.
7. Para concatenar dos cadenas de caracteres se utiliza el símbolo `+`.
8. Para realizar comentarios se puede utilizar `//` para comentarios de una línea o `/* ... */` para comentarios de varias líneas

9. Se puede utilizar el operador . para aplicar una función al objeto que lo precede. Por ejemplo,  
`imagen.updateMask(mascara);`
10. No es sensible a las tabulaciones, aunque son recomendadas para un código ordenado.
11. Son igualmente validas las comillas sencillas '' como las comillas dobles "", pero no deben mezclarse ambos tipos de comillas en una misma linea

## Primeros pasos

### Registro para el uso de GEE Javascript API

#####graficos

Lo primero que hay que hacer para poder utilizar la API Javascript de Google Earth Engine es ingresar a su sitio para registrarse como usuario. Para ello, hay que acceder a la siguiente liga:

<https://earthengine.google.com/>

Después le damos click en la esquina superior derecha donde dice **Sign Up**. Se ingresan todos los datos que pide el formulario. Recordar que para utilizar GEE requerimos de una cuenta de Google para poder utilizar la API.



Se recomienda dar de alta una cuenta de Google que tenga suficiente espacio disponible en Google Drive, ya que será la forma más fácil de exportar los resultados generados en GEE.

Una vez realizado el registro, hay que esperar unos minutos hasta que llegue una confirmación por parte de GEE a nuestro correo informando que ya se puede hacer uso de la API.

Una vez que tengamos dicha confirmación, podemos acceder a la API accediendo a la siguiente liga:

<https://code.earthengine.google.com/>

y se accede con la cuenta de Google con la que nos registramos.

### Elementos básicos de la interfaz gráfica

Una vez abierto la Javascript API, se observa la siguiente pantalla

Pantalla de repositorios

Pantalla de rutinas

Pantalla de: Inspector Consola Trabajos

Pantalla de Mapa

Los elementos de la API se enlistan a continuación:

## **Consola**

La consola consiste en la pantalla de comunicación con el servidor. En ella se muestran los errores que se obtienen al correr un script o se pueden mostrar la información indicada por la función `print`.

## **Pantalla de Repositorios**

Es el espacio donde se guardan los scripts del usuario. En ella se pueden crear repositorios, folders y scripts para organizar éstos últimos. Dentro de este espacio existen varias categorías: propietario (owner), editor (writer), lector (reader), ejemplos y archivo. Estos se pueden utilizar para determinar distintos niveles de acceso para distintos usuarios. Dentro de cada repositorio se puede asignar el control de acceso para otros usuarios.



El usuario puede compartir repositorios con otros usuarios de GEE como lector o editor. De esta manera, todos los archivos que se encuentren dentro de un repositorio serán compartidos con los usuarios indicados. Esta opción está disponible al darle click en el símbolo del engranaje a la derecha de cada repositorio (aparece una vez que se coloca el puntero sobre el nombre del repositorio).

## **Docs**

Es un área donde se puede consultar todas las funciones y algoritmos que se encuentran cargadas en GEE. Además, para cada función le indica lo que hace, la entrada y salida que requiere y los argumentos de la función.

## **Assets**

En este apartado el usuario puede subir su propia información para ser utilizada dentro de GEE. Se pueden subir archivos en formato raster, vector o separado por comas, únicamente.

## **Search**

Es una barra de búsqueda en la que se pueden buscar fuentes de datos o sitios.

## **Map**

El mapa donde se pueden dibujar puntos, polígonos, líneas o rectángulos y muestra la información que se haya indicado mediante la función `Map.addLayer`.

## **Layer manager**

Permite prender y apagar las capas que se estén mostrando en el área del mapa. Además, permite modificar las características necesarias para su display.

## **Inspector**

Permite consultar los valores de las capas que se muestran en el mapa, al dar click sobre el punto de interés.

## **Tasks**

Muestra los trabajos que se hayan exportado mediante la función `Export.image` o `Export.table` y permite correr el trabajo de exportación al sitio donde el usuario le haya indicado (drive, assets). También muestra el tiempo demorado en el trabajo, así como cuando el trabajo se ha finalizado.

# Tipos de objetos

## Server vs Client side

Existen dos lados de la programación de la API de GEE: el del servidor y el del cliente. De tal manera un objeto puede ser convertido entre los dos. Por ejemplo, mientras que del lado del cliente una cadena de caracteres puede ser definida simplemente como: "cadena", para convertirla en objeto del lado del servidor deben utilizarse las funciones del servidor, es decir: ee.String("cadena"). Muchos de los algoritmos pre cargados en GEE únicamente corren sobre objetos del lado del servidor, por lo cual, es recomendable utilizar dicha sintaxis. Adicionalmente algunas operaciones se pueden hacer utilizando ambos tipos de sintaxis. Por ejemplo, una suma se puede realizar del lado del cliente mediante obj1 + obj2, mientras que del lado del servidor se utilizaría obj1.add(obj2). En la mayoría de los casos se va a utilizar la programación del lado del servidor, ya que es la que permite hacer todo el procesamiento en GEE. Por ejemplo, para el caso de los condiciones se sugiere utilizar en lugar de if y else, ee.Algorithms.If. Sin embargo, cabe aclarar que algunas funciones sólo corren del lado del cliente. Por ejemplo, las funciones para exportar la información a algún archivo (ya sea un raster, un vector o una tabla) sólo corren del lado del cliente.

## Tipos de objetos del lado del cliente

### Strings (cadenas de texto)

Se refiere a objetos de cadenas de símbolos de tipo carácter alfanumérico. Cualquier secuencia de caracteres puede ser un string. Estos se definen como cualquier cadena de caracteres que se encuentren entre un par de comillas bien sean dobles “” o sencillas ”. Por ejemplo:

```
var cadena = "Esto es una cadena de caracteres";
var telefono= '1234567890';
var direccion = 'calle cuarta casa 16';
```

Resulta importante notar que los números en un String no serán interpretados como valores numéricos sino como texto.

### Number (números)

Se refiere a objetos numéricos que indican un valor. Cualquier secuencia de números puede ser un Number. Para números decimales se utiliza el punto decimal y no la coma decimal. Por ejemplo:

```
var numero = 1;
var numero2 = 2.5;
```

Resulta importante notar que al declarar un objeto como Number no se hace uso de comillas.

### Lists (listas)

Se refieren a objetos que contienen varias entradas, las cuales pueden ser numéricas (Number) o cadenas de texto (String). Las listas se definen mediante el uso de corchetes [] y cada entrada es separada mediante una coma (,). Permiten encadenar una serie de valores. Por ejemplo:

```
var lista = [1, 2, 3, 4, 5, 6, 7, 8];
var listaA = ["primero", "segundo", "tercero"];
var listaB = ["primero", "segundo", "tercero", 4];
```

Todas las listas automáticamente asignan, en orden, un número a cada elemento dentro de ellas, siempre empezando desde 0. Entonces se puede consultar un solo elemento dentro de una lista, aportando el número de su posición dentro de ella.

```
print(lista[0]);
print(listaA[1]);
print(listaB[2]);
```

## Diccionarios (Dictionaries)

Los diccionarios son objetos que contienen claves y valores asociados a estas claves. Los diccionarios se definen mediante el uso de {} donde se define cada clave seguida de dos puntos y el valor o cadena de caracteres asociada a esa clave. Para ingresar varias entradas, éstas deben ir separadas por una coma.

```
var dict = {
  clave1: 1,
  clave2: "A"
};
```

Para consultar los valores dentro de un diccionario se puede hacer llamando directamente el nombre de la clave deseada.

```
print(dict["clave1"]);
print(dict["clave2"]);
```

## Funciones (Functions)

Se refieren a objetos que contienen algún proceso que se realizará a alguna variable. Siempre comienzan con la función function seguida por el objeto al que se le aplicará la función y entre corchetes se coloca el procedimiento que va a realizar la función. Por último, deben regresar un objeto mediante la función return. Por ejemplo:

```
var maskIm = function(image){
  var qaImage = image.select('pixel_qa');
  var clearData = qaImage.eq(322);
  return image.updateMask(clearData);
};
```

## Tipo de objetos del lado del servidor

Estas funciones permiten definir objetos como objetos en el lado del servidor o convertir objetos en el lado del cliente al servidor. También, en algunos casos los objetos que se obtienen a partir de ciertas funciones retornan un objeto de tipo objeto, por lo cual, se recomienda “castear” el objeto al tipo de objeto deseado antes de proseguir con su uso.

Por otro lado, toda la información que se encuentra disponible en GEE corresponderá a objetos del servidor. Por ende, la mayoría de los métodos disponibles en GEE trabajarán únicamente con objetos del lado del servidor. Para revisar todos los métodos disponibles en GEE clasificados por tipo de objeto del servidor se puede consultar en la sección de **Client libraries**: <https://developers.google.com/earth-engine/apidocs>

Los objetos del lado del servidor se pueden conceptualizar como contenedores que le indican al servidor qué tipo de objeto es el que se está enviando. Además, cuando se trabaja del lado del servidor, los objetos necesariamente son enviados al servidor para ser evaluados.



Para recordar cuáles objetos son del servidor, resulta útil recordar que todos ellos cuentan con el prefijo **ee**, seguido del nombre del tipo de objeto con mayúsculas.

### ee.String

Permite enviar un objeto como cadena de caracteres al servidor. Además, al definir un objeto de este tipo, se pueden utilizar cualquier función indicada bajo la librería de **ee.String** al revisar las librerías disponibles en la API (ver liga anterior).

```
print(cadena);
var cadenaServ = ee.String(cadena);
print("Esto está evaluado en el servidor", cadenaServ);
```

### ee.List

Permite enviar un objeto como lista al servidor. Además, al definir un objeto de este tipo, se pueden utilizar cualquier función indicada bajo la librería de **ee.List** al revisar las librerías disponibles en la API (ver liga anterior).

```
print("Esto está del lado del cliente", lista);
var listaServ = ee.List(lista);
print("Esto está evaluado en el servidor", listaServ);
```

Para acceder a un elemento de tipo ee.List se utiliza la función `get`.

```
print(listaServe.get(0));
print(listaServe.get(1));
```

### ee.Dictionary

Permite enviar un objeto como diccionario al servidor. Además, al definir un objeto de este tipo, se pueden utilizar cualquier función indicada bajo la librería de **ee.Dictionary** al revisar las librerías disponibles en la API (ver liga anterior).

```
print("Esto está del lado del cliente", dicc);
var diccServ = ee.Dictionary(dicc);
print("Esto está evaluado en el servidor", diccServ);
```

Para acceder a un elemento de tipo ee.Dictionary se utiliza la notación de `.` seguido del nombre de la clave. Adicionalmente, se puede consultar utilizando también la función `get` y el nombre de la clave, aunque se recomienda más el uso de la primera.

```
print(diccServe.clave1);
print(diccServe.clave2);

print(diccServe.get("clave1"));
print(diccServe.get("clave2"));
```

Por último, si se desea obtener las claves disponibles en un diccionario se utiliza la función `keys`.

```
print(diccServe.keys());
```

### **ee.Date**

Este es la forma en la que GEE permite trabajar con fechas. Permite enviar un objeto como fecha al servidor. Además, al definir un objeto de este tipo, se pueden utilizar cualquier función indicada bajo la librería de `ee.Date` al revisar las librerías disponibles en la API (ver liga anterior).

```
var fechaString = '2015-01-01';

print("Fecha String",fechaString);

var fecha = ee.Date(fechaString);
var fecha2 = ee.Date.fromYMD(2015,01,01);

print("Fecha como ee.Date",fecha);
print("Fecha como ee.Date también",fecha2);
```

### **ee.Image**

Permite enviar un objeto como imagen al servidor. Además, al definir un objeto de este tipo, se pueden utilizar cualquier función indicada bajo la librería de `ee.Image` al revisar las librerías disponibles en la API (ver liga anterior). Este va a ser el tipo de objetos para trabajar con cualquier elemento de tipo raster en GEE. Más adelante se explican con mayor detalle.

### **ee.ImageCollection**

Permite enviar un objeto como una colección de imágenes al servidor. Además, al definir un objeto de este tipo, se pueden utilizar cualquier función indicada bajo la librería de `ee.ImageCollection` al revisar las librerías disponibles en la API (ver liga anterior). Las colecciones de imágenes están formadas por imágenes. Más adelante se explican con mayor detalle.

### **ee.Feature**

Permite enviar un objeto como atributo al servidor. Además, al definir un objeto de este tipo, se pueden utilizar cualquier función indicada bajo la librería de `ee.Feature` al revisar las librerías disponibles en la API (ver liga anterior). Este va a ser el tipo de objetos para trabajar con cualquier objeto de tipo vector o tabla. Más adelante se explican con mayor detalle.

## **ee.FeatureCollection**

Permite enviar un objeto como colección de atributos al servidor. Además, al definir un objeto de este tipo, se pueden utilizar cualquier función indicada bajo la librería de **ee.FeatureCollection** al revisar las librerías disponibles en la API (ver liga anterior). Las colecciones de atributos están formadas por atributos. Más adelante se explican con mayor detalle.

## **ee.Algorithms**

Este tipo de objetos contienen algoritmos precargados en GEE. Estos algoritmos tienen una gran variedad de aplicaciones, desde operaciones sencillas como una evaluación lógica **ee.Algorithms.If**, hasta algoritmos de segmentación temporal de una serie de imágenes **ee.Algorithms.TemporalSegmentation.Ccdc**.

## **ee.Array**

Este tipo de objetos corresponden a arreglos multidimensionales. Los arreglos se pueden interpretar como matrices de más de dos dimensiones (p.ej., filas y columnas). Su uso más común se da en análisis de series de tiempo con imágenes. Este tipo de objetos cuentan con una serie de funciones, las cuales se pueden consultar bajo la librería de **ee.Array** (ver liga anterior).

## **ee.Clusterer**

Este tipo de objetos corresponden a algoritmos de agrupamiento de datos que se encuentran pre cargados en GEE. Por ejemplo, se encuentra el algoritmo de *k-means*, disponible mediante la función **ee.Clusterer.wekaKMeans** o *Cobweb*, disponible mediante la función **ee.Clusterer.wekaCobweb**.

## **ee.Filter**

Este tipo de objetos que normalmente se utilizan para filtrar colecciones ya sean de atributos o de imágenes. Este tipo de objetos permiten definir filtros de distintos tipos, ya sean espaciales, temporales o en función de características de las imágenes o atributis. Por otro lado, también contienen funciones para combinar filtros.

## **ee.Geometry**

Este tipo de objetos corresponden a distintos tipos de geometrías, desde líneas, polígono y puntos. En GEE se encuentran varias funciones que se pueden aplicar sobre este tipo de objetos, p.ej., **ee.Geometry.MultiPolygon.Simplify** para simplificar polígonos múltiples o calcular el área **ee.Geometry.Polygon.area**.

## **ee.Join**

Este conjunto de funciones permite realizar uniones entre colecciones de atributos, utilizando los campos de éstos como las claves para realizar las uniones. Por ejemplo, se pueden unir dos colecciones de atributos mediante **ee.Join.merge** o unir los campos de una primera colección con los de una segunda mediante **ee.Join.inner**

## ee.Terrain

Este conjunto de funciones permite calcular algunas operaciones topográficas básicas como calcular el aspecto a partir de un modelo digital de elevación (DEM). Por ejemplo, en GEE se encuentran las funciones `ee.Terrain.aspect` para calcular el aspecto o `ee.Terrain.slope` para calcular la pendiente,



Como resultado de algunas operaciones realizadas en GEE, por ejemplo, `first` o `get`, el objeto resultante no tiene un tipo definido, por lo cual, resulta necesario meterlo en un contenedor que indique el tipo de objeto. Este tipo de errores suelen mostrar un mensaje `"...is not a function"`

# Interfaz de usuario: Display de datos, gráficos y exportación

## Map

Permite agregar objetos del lado del servidor a la pantalla de mapas, así como controlar algunos parámetros de dicha pantalla.

### Map.addLayer

La función `Map.addLayer` permite mostrar elementos en la parte de mapa de la API. Por lo tanto, permite mostrar objetos de tipo `ee.Feature` o `ee.Image`. Adicionalmente se pueden indicar otros parámetros como la forma de visualización de los objetos y el nombre de la capa. Como argumentos de la función `Map.addLayer` se indica el objeto a añadir a la pantalla de mapas, seguido de un diccionario que señala las bandas a añadir y los valores máximos y mínimos para el despliegue de la capa y por último, se puede indicar el nombre de la capa.

Al utilizar `Map.addLayer` sobre una imagen se pueden indicar los siguientes argumentos: `bands`, `min` o `max`. Estos se deben pasar dentro de un {} como un diccionario. Esta función también permite asignar colores mediante el argumento de `palette`. Dependiendo de las bandas que se elija mostrar se puede mostrar únicamente una banda en tonos de gris o un compuesto RGB.

```
Map.addLayer(image, {bands: ['B1', 'B2', 'B3'], min: 0, max: 2000}, 'RGB');
```

Para el caso de los atributos (`features`) se puede realizar una operación similar como

```
Map.addLayer(feature, {palette: ['FF0000', 'FFFF00', '008000'], min: 0, max: 10},  
'featuresColored');
```



El código de los colores pasados al argumento de `palette` corresponden a códigos hexadecimales de color.

### Map.centerObject

### Map.add



En algunas ocasiones, al intentar cargar objetos, que sean producto de procesamientos demandantes, en la pantalla de mapas, la consola puede mostrar un error indicando que se superó el tiempo de espera o de el límite de memoria disponible. En estos casos, se sugiere exportar el resultado, en lugar de tratar de cargarlo en la pantalla de mapas.

## Chart

La función principal para realizar gráficos es `ui.Chart`. Esta función tiene varios algoritmos para realizar diferentes tipos de gráficos, sin embargo, existen diferentes opciones de gráficos para distintos tipos de

objetos. Los objetos básicos que permite graficar GEE son: atributos (`'Feature'`), colecciones de atributos (`FeatureCollection`), imágenes (`Image`), colección de imágenes (`ImageCollection`), arreglos (`Array`) y listas (`List`).

### `Chart.array`

### `Chart.feature`

### `Chart.image`

Por ejemplo, para graficar una serie de tiempo a partir de una colección de imágenes se utiliza la función `ui.Chart.image.seriesByRegion`. Este tipo de gráficos permiten graficar la serie de tiempo de algunas áreas determinadas utilizando una colección de imágenes.

```
chart1 = ui.Chart.image.seriesByRegion({
  imageCollection:leccionImagenes,
  regions:areas.filter(ee.Filter.eq('Tipo','Deforestacion')),
  reducer:ee.Reducer.mean(),
  band:'NDVI',
  scale:30,
  seriesProperty:'Tipo'
})
.setOptions({
  title:'Deforestacion',
  colors:[ '#EE3A19']
});
```



El nombre de las funciones para generar gráficos en GEE dan una idea del tipo de gráfica que se puede generar con dicha función y el tipo de insumos que requiere cada tipo de gráfico.

## Export

Una vez que se haya obtenido el resultado de interés en GEE se puede exportar para ser manejado en otro programa. Sin embargo, sólo existen cuatro formatos válidos para exportar desde GEE: raster, vector, mapa y video.

### `Export.image`

Resulta obvio que el primero se va a utilizar para exportar imágenes en formato raster, sin embargo, no siempre resulta obvio ya que algunas veces se pueden convertir imágenes a formato de arreglo (array), tras lo cual hay que volver a convertir la información a formato de imagen (Image) para poder exportarla. La aplicación permite exportar únicamente en formato GTiff o TRFrecord (tensores).

Un ejemplo de cómo exportar una imagen se presenta a continuación.

```

Export.image.toDrive({
  image: imgDiff,
  description: 'DiferenciaNDVI_2016-2017',
  scale: 30,
  fileFormat: 'GeoTIFF',
  folder: 'DiferenciaNDVIL8'
});

```



En algunas ocasiones, al intentar exportar una imagen muy grande (mayor a 10 000 000 de pixeles), la consola puede mostrar un error indicando que el objeto a exportar tiene un número muy alto de pixeles. En este caso, se debe aumentar el número de pixeles máximo permitido para la exportación. Esto se logra indicando el argumento `maxPixels` dentro del diccionario que se pasa a `Export.image.toDrive`. Por ejemplo, `maxPixels: 1e10`, lo cual permite exportar una imagen con hasta  $1 \times 10^{10}$  pixeles.

## Export.table

Por su parte, el formato vector permite exportar información vector y tablas sin información geográfica. De nuevo, para poder exportar en el formato vector, el objeto exportado debe ser en formato atributo (`feature`). Para el caso de tablas, sin información geográfica asociada, se debe definir el objeto como un objeto atributo sin geometría (es decir, `null`) y en la tabla de atributos anexar la información deseada.

## Export.map

## Export.video

## Puntos para recordar

1. Existen algunas características adicionales que conviene tener en cuenta para trabajar en GEE. En GEE muchas operaciones utilizan el término de `scale`, sin embargo, ésta no se refiere a la escala de trabajo en un sentido tradicional (p.ej., 1:50 000), sino que se refiere al tamaño de pixel expresado en m.
2. La proyección de trabajo para todas las capas es de EPSG:3857 WGS84 pseudomercator. Por ello, todas las capas se trabajan en esa proyección. Si se requiere cambiar de proyección se puede utilizar la función `reproject`.
3. Cuando se corre un script en la consola con el botón de Run, éste no corre directamente en los servidores de Google, sino que éstos se transcriben a código GeoJSON, se envían a los servidores de Google y se espera una respuesta.
4. No se puede escribir o cambiar los objetos que se encuentran directamente hospedados en GEE. Por ejemplo, si se quiere mapear una función que sobreescriba alguna característica de las colecciones de imágenes en GEE, se debe hacer una copia de éstas para que se pueda sobreescribir la característica de interés.
5. Las funciones, al igual que las funciones para exportar, sólo funcionan si regresan un objeto de tipo `Feature`, `FeatureCollection`, `Image` o `ImageCollections`, por lo cual, a veces se deben realizar ciertas conversiones para evitar un error.
6. El límite de memoria destinado a objetos que se desean ver en consola, mediante un `print` o `Map.addLayer`, tienen un tamaño mucho menor que el que se permite al únicamente correr un objeto con `export`. Por lo tanto, resulta una mejor práctica exportar los resultados, en lugar de mostrarlos en consola.

7. Si un proceso muestra un error de “computation timed out” se puede elegir alterar los valores de **scale** o de **bestEffort** o **Tiles** para lograr que este corra.

## Features

Trazado de polígonos, líneas, puntos

## **Feature Collections**

# Imagen

## Características

Las imágenes que se encuentran disponibles en colecciones de imágenes de GEE corresponden a información en formato raster que cuenta con una resolución espacial, espectral, radiométrica particular. Los metadatos de cada imagen se encuentran como propiedades de ésta. A diferencia de las imágenes a las que posiblemente el usuario esté acostumbrado a manejar, GEE permite el manejo de imágenes donde cada banda puede tener una resolución de pixel o tipo de datos (Integer, Float, 16 bits, etc) distinta. Esto no afecta el uso de estas imágenes pero puede ser importante a considerar para exportar imágenes cuyas bandas tienen distinta resolución espacial (p.ej., Sentinel-2).

## Información y metadatos

Los metadatos de la imagen se encuentran como propiedades. Para consultarlas se puede utilizar el siguiente comando: `ee.Image.propertyNames()`; Así se pueden consultar todas las propiedades de una imagen. Sin embargo, algunos comandos que facilitan la consulta de algunos elementos son: `.bandNames()`, `.projection()`, `.projection().nominalScale()`, para conocer el nombre de las bandas de la imágenes, su proyección y el tamaño de pixel en m que presenta, respectivamente.

Una vez, revisado los nombres de las bandas de una imagen, se pueden seleccionar ciertas bandas mediante:

```
ee.Image.select('B1')
ee.Image.select(['B1', 'B2', 'B3'])
```

## Operaciones básicas

Se pueden hacer mosaicos mediante el uso de la función `mosaic` sobre una colección de imágenes. Por ejemplo:

```
ee.ImageCollection.mosaic();
```

También se pueden realizar cortes de una imagen para quedarse con un área específica utilizando la función:

```
ee.Image.clip();
```



Se sugiere evitar el uso de la función `clip` a menos que sea estrictamente necesario, ya que aumenta el tiempo de procesamiento.

## Operaciones matemáticas

Dentro de GEE se puede realizar cualquier operación matemática entre bandas o entre imágenes. Los operadores para realizar esta operaciones son: `ee.Image.subtract()`, `.add()`, `.divide()`, `.multiply()`. Además, GEE cuenta con una función de `.normalizedDifference()` para calcular un índice normalizado, a partir de las bandas que se indiquen como: `ee.Image.normalizedDifference(['B4', 'B3'])`. Por ejemplo, para calcular el NDVI primero se indica la banda del infrarrojo y luego la del rojo. Si se quiere realizar un índice un poco más complicado se puede calcular mediante la función de `.expression()`. En esta función se indica primero la fórmula del cálculo que se va a realizar y posteriormente se indica en un diccionario la equivalencia de los términos utilizados en la fórmula y los nombres de las bandas de la imagen:

```

var NDVI = ee.Image.expression(' (NIR - R) / (NIR + R)', {
  'NIR' = image.select('B4'),
  'R' = image.select('B3')
});

```

Dentro de las fórmulas utilizadas dentro de expression se puede utilizar la notación estándar matemática (+, -, /, \*), modulo (%), exponente (\*\*), relaciones(>, <, >=, <=, !=, ==), lógicos (&&, ||, !, ^), o if then else (?:).

## Operaciones relacionales, condicionales y booleanas

Se pueden utilizar operadores relacionales, condicionales, booleanos sobre las imágenes para crear clases o máscaras. Se pueden utilizar las siguientes notaciones para hacer comparaciones de los valores una imagen con cierto valor .lt(), .lte(), .gt(), .gte(), .eq(), .neq(). Además se pueden utilizar operadores relacionales para unir dos condiciones mediante .and() u .or(). Por ejemplo:

```

ee.Image.lte(0.2).and(ee.Image.gte(0));

```

## Colecciones de imágenes

La mayoría de los objetos que se encuentran disponibles en los servidores de GEE se encuentran en forma de colecciones: `FeatureCollection` o `ImageCollection`. Estas se pueden definir como colecciones de objetos Feature o Image. Normalmente, el primer paso para trabajar con estas colecciones radica en delimitar el área y periodo de estudio, para lo cual se utilizan las funciones de filter. Por ejemplo, para delimitar un área se puede utilizar `ImageCollection.filterBounds(area)`, mientras que para delimitar por fecha `ImageCollection.filterDate(periodStart,periodEnd)`. Además, se pueden filtrar, de acuerdo con otros criterios contenidos en los metadatos de las imágenes. Por ejemplo, se puede definir un valor umbral de porcentaje de nubosidad de las imágenes con las que se desean trabajar o en ciertos parámetros de la geometría del registro.

Adicionalmente, si se desea calcular una nueva banda para todas las imágenes de la colección, se debe aplicar una función que agregue la nueva banda a cada imagen y aplicar dicha función a cada imagen de la colección mediante la función map.

### Características

Las colecciones de imágenes son objetos de GEE que contienen un conjunto de imágenes. En GEE este tipo de objetos va a ser el que va a contener los acervos de imágenes disponibles. Para el manejo de varias imágenes se recomienda utilizar esta estructura, en lugar de listas u otro tipo de objetos.

### Información y metadatos

Las colecciones de imágenes contienen los metadatos e información de todas las imágenes que contienen. De tal manera, se puede utilizar esta información para filtrar y utilizar únicamente las imágenes que cumplen con ciertos criterios.

### Filtros sobre Colecciones de imágenes

Los filtros permiten filtrar las colecciones de imágenes a partir de los metadatos de las imágenes que contiene. De tal manera, se puede filtrar por: superposición con algún polígono de interés, fecha de registro, características de las escenas (path, row, porcentaje de nubosidad), entre otras.

Para filtrar una colección de imágenes se van a utilizar los objetos `ee.Filter`. Existen diversos tipos de filtros, algunos ya se encuentran precargados, pero otros se pueden definir manualmente. Por ejemplo, para filtrar por fecha se pueden utilizar:

```
ee.ImageCollection.filterDate('2015-01-01', '2016-01-01');
```

Por otro lado para filtrar espacialmente con la extensión de un polígono se puede realizar mediante:

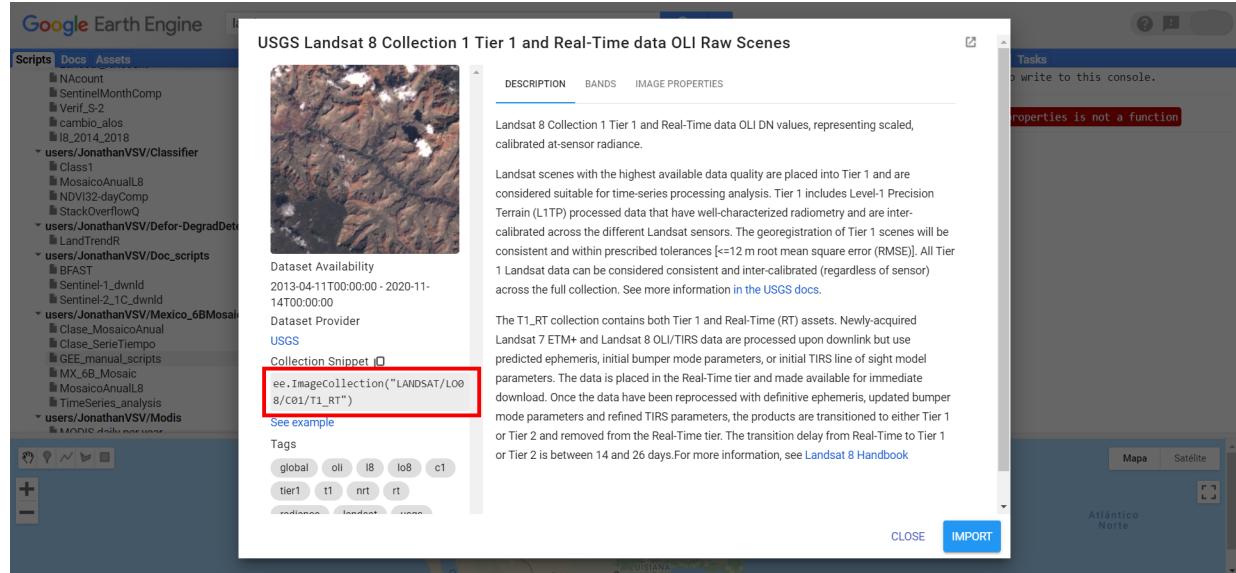
```
ee.ImageCollection.filterBounds(polygon1);
```

Si se desea filtrar por alguna característica de los metadatos se puede hacer mediante:

```
ee.ImageCollection.filter(ee.Filter.lte('CLOUD_COVER', 20));
```

## Ejercicio 1: Filtro de colección de imágenes y visualización

Para filtrar la colección de reflectancia de la superficie de imágenes de Landsat 8 de mayor calidad (tier 1) por fecha, porcentaje de cobertura de nubes de las imágenes, así como por su “path” y “row” se puede realizar en un solo paso aplicando múltiples filtros. En este caso, primero se indica la ruta de la colección que se desea utilizar. Para encontrarla esta ruta se puede buscar la colección de interés en la barra de **Search**, después se le da click y del lado izquierdo aparecerá la ruta de la colección. Por otro lado, en esta misma ventana, en la pestaña de ‘BANDS’ se puede consultar las bandas y los nombres de las bandas que contiene cada imagen. Por su parte, en la pestaña de ‘IMAGE PROPERTIES’ se pueden consultar los metadatos que contienen las imágenes de esta colección.



Una vez que se conoce la ruta de la colección de interés, se llama dicha colección. Después se especifican los filtros. En este caso, primero se filtra por fecha, en formato YYYY-MM-DD (año, mes, día), seguido de filtros de los metadatos de las imágenes: ‘CLOUD\_COVER\_LAND’, ‘WRS\_PATH’, ‘WRS\_ROW’. En este caso se debe escribir exactamente igual el campo de los metadatos (CLOUD\_COVER\_LAND) y en la definición del filtro se debe explicitar si se desea utilizar los valores que sean iguales (eq), distintos (neq), mayores (gt), menores (lt), mayores o igual (gte) o menores o iguales(lte).

```
var L8imgCol = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR")
  .filterDate('2015-01-01', '2016-01-01')
  .filter(ee.Filter.lte('CLOUD_COVER_LAND', 50))
  .filter(ee.Filter.eq('WRS_PATH', 28))
  .filter(ee.Filter.eq('WRS_ROW', 46));
```

Para explorar algunas características de la colección de imágenes filtradas podemos escribir el siguiente comando para ver sus características en la consola.

```
print(L8imgCol);
```

Una vez corrido el comando anterior, en la consola se puede consultar cuántas imágenes cumplieron con los filtros indicados. En este ejemplo, la colección filtrada incluye 17 imágenes. Despues se pueden consultar las características de las imágenes incluidas en la colección. Para ello, en la consola se le da un click a la flecha ubicada a la izquierda de la colección en la consola a **Features**, después a cualquier imagen (**Image**) y por último a **Properties**. Ahí se pueden ver los metadatos de cada imagen y por lo tanto, los campos de información que se pueden utilizar para filtrar una colección de imágenes, basado en los metadatos de las imágenes. Esta información es la misma que se puede obtener en la barra de **Search**.

The screenshot shows the Google Earth Engine interface. On the left, there's a sidebar with 'Owner (12)', 'Writer (1)', 'Reader (3)', 'Archive (1)', and 'Examples'. Below it is a 'Scripts' tab with a 'NEW' button, followed by 'Docs', 'Ass', and 'GEE\_manual\_scripts'. The code editor contains the following script:

```

1 // 
2 var L8imgCol = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR")
3 .filterDate('2015-01-01','2016-01-01')
4 .filter(ee.Filter.lt('CLOUD_COVER_LAND',50))
5 .filter(ee.Filter.eq('WRS_PATH',28))
6 .filter(ee.Filter.eq('WRS_ROW',46));
7 
8 // print(L8imgCol);
9
10

```

The right panel has tabs for 'Inspector', 'Console', and 'Tasks'. The 'Inspector' tab shows a detailed JSON object for an image element, including properties like 'CLOUD\_COVER', 'CLOUD\_COVER\_LAND', 'EARTH\_SUN\_DISTANCE', 'ESPA\_VERSION', 'GEOMETRIC\_RMSE\_MODEL', 'GEOMETRIC\_RMSE\_MODEL\_X', 'GEOMETRIC\_RMSE\_MODEL\_Y', 'IMAGE\_QUALITY\_OLI', 'IMAGE\_QUALITY\_TERRAIN', 'LANDSAT\_ID', 'LEVEL1\_PRODUCTION\_DATE', 'PIXEL\_QA\_VERSION', 'SATellite', and 'SATELLITE'. The main map view shows a portion of Mexico with satellite imagery overlaid on a road network.

A continuación podemos ordenar la colección en orden ascendente de acuerdo al metadato de 'CLOUD\_COVER\_LAND'. Posteriormente, se selecciona la primera imagen, es decir, la de menor cobertura de nubes. Nótese que en este paso al definir la variable L8imgFirst, ya únicamente seleccionamos una sola imagen, por lo tanto, éste va a ser un objeto ee.Image.

```

L8imgCol = L8imgCol.sort('CLOUD_COVER_LAND');
var L8imgFirst = L8imgCol.first();
print(L8imgFirst);

```

Por último, se pueden agregar esta imagen a la ventana de mapas para visualizar la información. En el primer caso se agrega la información sin pasar más argumentos. En el segundo caso, se indican las bandas y el orden que se desea cargar, así como los valores mínimos y máximos del histograma, y un nombre para la capa.

```

Map.addLayer(L8imgFirst);
Map.addLayer(L8imgFirst,{bands: ['B4','B3','B2'], min: 95, max: 1288}, 'RGB menos nubes L8');

```

This screenshot shows the Google Earth Engine interface again. The code editor has the same script as before, plus an additional line 12: 'var L8imgFirst = L8imgCol.first();'. The 'Inspector' tab shows the 'Image' object for the first image in the collection. The main map view shows a 3D perspective view of a satellite image (L8imgFirst) overlaid on a map of Mexico, specifically focusing on the states of Jalisco, Guanajuato, and Michoacán.

## Mapeo

Para realizar alguna operación sobre todas las imágenes de una ImageCollection se utilizar la función `map`. Esto va a aplicar la función que se defina en el interior de ella a cada imagen dentro de la colección. Por ejemplo, si quisieramos agregar una banda con un índice de vegetación a todas las imágenes se podría hacer de la siguiente manera:

```
ee.ImageCollection.map(function(image){  
    return image.normalizedDifference('NIR','R');  
});
```



Notar que la función `map` es distinta de la función `Map` que permite agregar objetos a la pantalla de mapas.

Este tipo de operación es la sugerida para realizar operaciones de tipo ciclo, en lugar de otras funciones de ciclo (e.g., `for`).



La función `map` sólo funciona con funciones del lado del servidor, así que se recomienda evitar el uso de funciones del lado del servidor como `print`, `Map`, `Export`, `Chart`.

## Reducción

Los reductores permiten crear un mosaico (una sola imagen), a partir de una colección de imágenes. Estos son muy útiles para hacer mosaicos multitemporales u obtener información de la colección de imágenes (por ejemplo, cuántas observaciones válidas tiene cada pixel en un año).

Para reducir una colección de imágenes se utilizará la función `reduce` e indicando el método a utilizar mediante un objeto tipo `ee.Reducer`. De tal manera, se puede indicar si el resultado corresponderá a la media `ee.Reducer.mean()`, mediana `ee.Reducer.median()`, moda `ee.Reducer.mode()`, mínimo `ee.Reducer.min()`, máximo `ee.Reducer.max()` o inclusive la media de un intervalo determinado de observaciones `ee.Reducer.intervalMean()`. Esto se puede realizar mediante el siguiente procedimiento:

```
ee.ImageCollection.reduce(ee.Reducer.mean());
```



Al aplicar una reducción a una colección de imágenes, el nombre de las bandas será el mismo que el de las bandas de cada imagen con un sufijo que indica la función utilizada para hacer la reducción. Por ejemplo, al utilizar `ee.Reducer.mean()` sobre una colección de imágenes cuyas bandas se llaman `B1` y `B2`, las bandas de la imagen producto de la reducción se llamarán `B1_mean` y `B2_mean`.

## Compuestos y mosaicos

Si se desean hacer mosaicos a partir de varias imágenes se puede realizar en GEE mediante dos funciones: `ee.ImageCollection.mosaic()` y `ee.ImageCollection.qualityMosaic()`. La principal diferencia entre estas dos funciones radica en que la primera simplemente pega las imágenes de acuerdo al orden que tienen en la colección (es decir, la última hasta arriba). Por el contrario, la función de `qualityMosaic()` permite priorizar el pixel que quedará en el mosaico final a partir del valor más alto de alguna banda. Esto puede ser útil para realizar por ejemplo un mosaico del pixel con mayor valor de NDVI.

### Ejercicio 2: Enmascaramiento de nubes, cálculo de índices y reducción

Utilizando la colección que ya se filtró en el ejercicio anterior se hará un mapeo y una reducción. En este ejercicio se obtendrá el ndvi promedio entre 2015 y 2016.

Para calcular el ndvi promedio lo primero que hay que hacer es aplicar la máscara de nubes que viene en la misma colección de imágenes de Landsat 8. Al aplicar esta máscara se eliminarán los pixeles cuya información provenga de nubes. Después se definirá una función para calcular el ndvi para cada imagen contenida en la colección de imágenes. Para ello se utiliza la función `normalizedDifference`, indicando las bandas de NIR y R. También se utilizará la función `rename` para cambiar el nombre dado por default a la nueva banda calculada. Posteriormente, se agrega esta banda a la imagen con `add.Bands`. Despues, se utiliza la función `map` para aplicarla a cada una de las imágenes.

Primero se define la función para enmascarar las nubes. En este caso, se indican los bits que corresponden a nubes y a sombras, es decir, los bits 5 y 3 respectivamente. Después se selecciona la banda que contiene la información de la máscara de nubes, la cual en landsat 8 se llama “pixel\_qa”. Posteriormente se selecciona esta banda y se evalua que en los dos bits antes mencionados el valor sea igual a cero, es decir, que sea un pixel clasificado como despejado. En este procedimiento se indica que la consulta de los valores se debe hacer por bits (`bitwiseAnd`) y que las dos evaluaciones deben de ser cero (`and`). Posteriormente, se define una máscara en función de esos valores y se actualiza la máscara (`updateMask`).

```
function maskL8sr(image) {
  var cloudShadowBitMask = (1 << 3);
  var cloudsBitMask = (1 << 5);
  var qa = image.select('pixel_qa');
  var mask = qa.bitwiseAnd(cloudShadowBitMask).eq(0)
    .and(qa.bitwiseAnd(cloudsBitMask).eq(0));
  return image.updateMask(mask);
}
```



Aunque en GEE existe la función `mask` para enmascarar áreas de una imagen, se sugiere siempre utilizar `updateMask`. La función `mask` simplemente substituye la máscara original de la imagen, mientras que `updateMask` combina la máscara anterior con la nueva. Por ello, el usar `updateMask` evita desenmascarar áreas que podrían no ser de interés para el usuario.

Posteriormente se define la función para calcular el NDVI de todas las imágenes.

```
var ndviCalc = function(image){
  var ndvi = image.normalizedDifference(['B5','B4']);
  ndvi = ndvi.rename('ndvi');
```

```

    return image.addBands(ndvi);
};

```



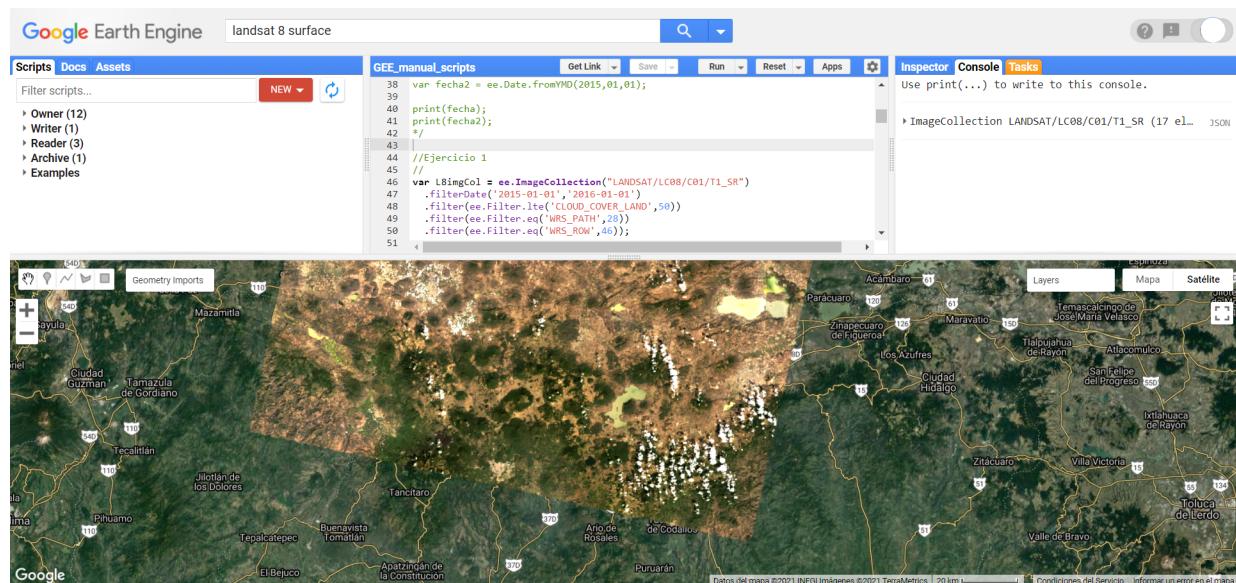
Si no se cambia el nombre de la banda producida por `normalizedDifference`, por default, ésta va a recibir el nombre de `nd`.

Para ver una muestra de cómo se ve la primer imagen con y sin la máscara de nubes, primero se agregará la imagen con nubes a la pantalla de mapa.

```

Map.addLayer(L8imgCol.first(),
  {bands: ['B4', 'B3', 'B2'], min: 0, max: 1300},
  'Imagen con nubes');

```



A continuación se aplican las dos funciones utilizando `map` y sobreescribir el objeto `L8imgCol`.

```

L8imgCol = L8imgCol
  .map(maskL8sr)
  .map(ndviCalc);
print(L8imgCol);

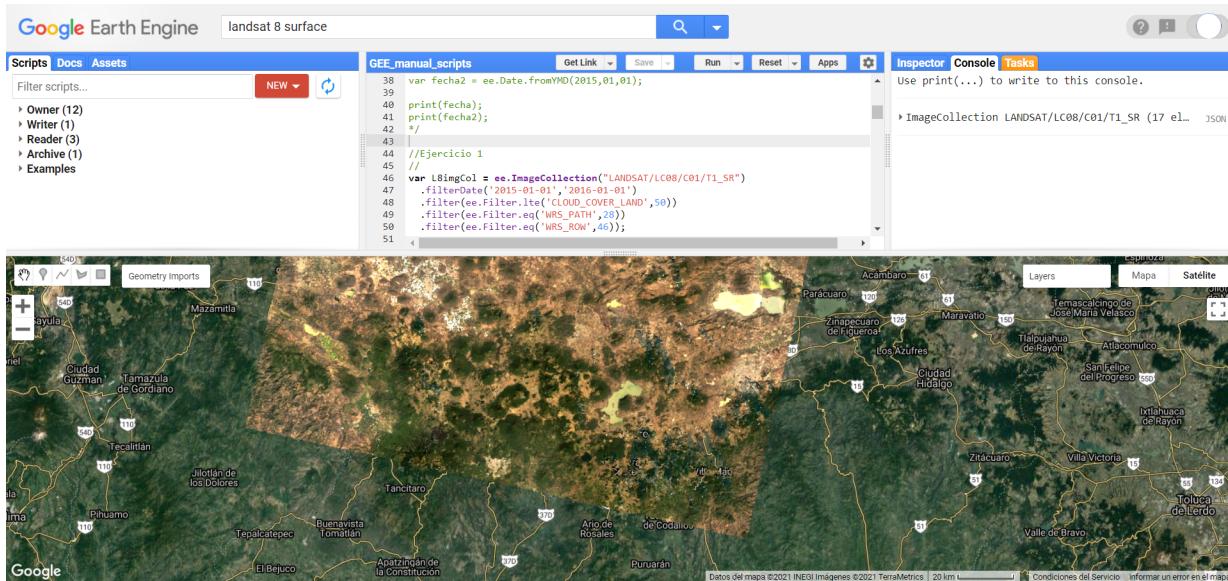
```

Después se muestra la primera imagen sin nubes.

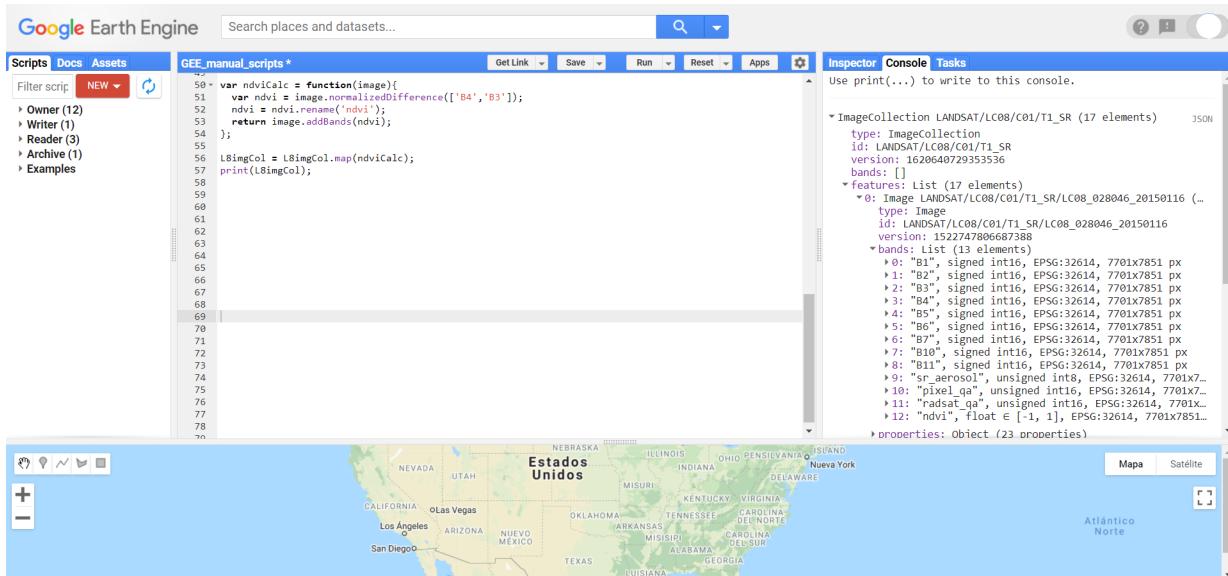
```

Map.addLayer(L8imgCol.first(),
  {bands: ['B4', 'B3', 'B2'], min: 0, max: 1300},
  'Imagen sin nubes');

```



En la consola se puede observar que todas las imágenes ahora contienen una nueva banda llamada ndvi.



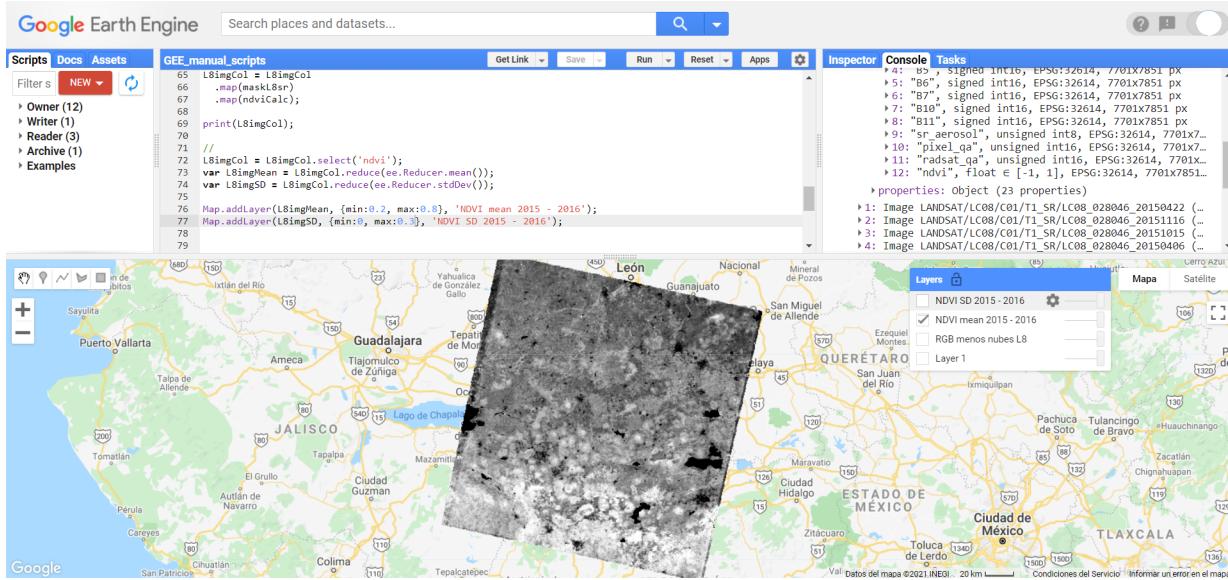
Posteriormente, se puede reducir la colección de imágenes para formar una sola imagen que represente el valor promedio de ndvi en el periodo de la colección (2015 - 2016). Para ello se utiliza la función `reduce`. También se calculará la desviación estándar de los valores de NDVI. Estas dos imágenes se guardarán en dos nuevos objetos.

```
var L8imgMean = L8imgCol.reduce(ee.Reducer.mean());
var L8imgSD = L8imgCol.reduce(ee.Reducer.stdDev());
```

Por último, agregamos los dos resultados a la pantalla de mapas.

```
Map.addLayer(L8imgMean, {min:-0.14, max:0.15}, 'NDVI mean 2015 - 2016');
Map.addLayer(L8imgSD, {min:0, max:0.13}, 'NDVI SD 2015 - 2016');
```

El resultado se ve así:



Se pueden prender y apagar las capas de la pantalla de mapas utilizando el menú que se despliega al posicionar el puntero sobre el letrero de layers y prendiendo y apagando las capas con las palomitas a la izquierda de cada capa.

Estás funciones van a ser de mucha utilidad para juntar resultados de varias imágenes en una sola colección o unir dos colecciones (p.ej., Landsat 7 y 8).

El último paso será cortar la imagen a una extensión de interés. En este caso primero se define un área de interés mediante coordenadas:

```

var geometry = ee.Geometry.Polygon(
  [[[-101.82737418916153, 19.836437094032178],
    [-101.82737418916153, 19.368119068204525],
    [-101.15171500947403, 19.368119068204525],
    [-101.15171500947403, 19.836437094032178]]], null, false);

```

En este caso se puede definir un polígono, indicando que es un objeto del lado del servidor `ee.Geometry` de tipo polígono, es decir, `ee.Geometry.Polygon`. Otra forma de trazar el polígono es utilizando las funciones de trazado ubicadas en la esquina superior izquierda de la pantalla de mapas. Una última forma de cargar un archivo de un objeto en GEE es importarlo desde un archivo externo. Esto se verá más adelante en la sección **XXX**.

Para cortar las dos imágenes que se crearon y mostrarlas en la pantalla de mapas.

```

L8imgMean = L8imgMean.clip(geometry);
L8imgSD = L8imgSD.clip(geometry);

Map.addLayer(L8imgMean, {min:0.2, max:0.8}, 'NDVI mean 2015 - 2016 clip');
Map.addLayer(L8imgSD, {min:0, max:0.3}, 'NDVI SD 2015 - 2016 clip');

```

## Exportación de resultados

Una vez que se obtuvieron los resultados deseados. Estos se pueden exportar fuera de GEE con las funciones Export. En el caso de las imágenes, la función para exportar es `Export.image`. Dentro de GEE hay 3 opciones para exportar los resultados de una imagen: `Export.image.toAsset`, `Export.image.toDrive` o

`Export.image.toCloudStorage`. El primero permite exportar la imagen como una imagen a la sección de **Assets**, es decir, la sección donde el usuario puede subir su información a GEE. Esta opción es útil para cuando el resultado se va a utilizar en otro procedimiento de GEE. La segunda opción permite exportarlo al Google Drive de la cuenta con la que se tiene acceso a GEE. Esta opción es útil para trabajar con las imágenes en algún entorno local como algún SIG. La última opción permite exportar la imagen al Google Cloud Storage para utilizarla en algún otro proceso a realizar en Google Cloud. La opción que creemos es más común para cualquier usuario será la de `Export.image.toDrive`.

### Ejercicio 3: Cálculo de diferencia de dos imágenes y exportación de resultado.

Se reutilizarán las funciones previamente definidas para generar un mosaico igual al ya generado, pero ahora con fechas de 2016-01-01 al 2017-01-01. Primero se filtra la colección de imágenes y se mapean las funciones para enmascarara las nubes y agregar el NDVI a las imágenes.

```
var L8imgColT2 = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR")
  .filterDate('2016-01-01', '2017-01-01')
  .filter(ee.Filter.lte('CLOUD_COVER_LAND', 50))
  .filter(ee.Filter.eq('WRS_PATH', 28))
  .filter(ee.Filter.eq('WRS_ROW', 46));

L8imgColT2 = L8imgColT2
  .map(maskL8sr)
  .map(ndviCalc);
```



Para trabajar con una colección de imágenes se sugiere que primero se apliquen los filtros espaciales y temporales (`filter`), así como la selección de las bandas a utilizar (`select`), antes de realizar operaciones sobre todas las imágenes de la colección (`map`).

Después se selecciona únicamente las bandas de NDVI y se reduce la colección para generar una imagen del valor promedio de NDVI. Posteriormente, se recorta la imagen a la extensión del polígono del área de interés. Finalmente, a esta imagen se le restarán los valores de la imagen del 2015-01-01 al 2016-01-01. De esta manera, los valores más negativos representarán valores más bajos de NDVI en el año 2 respecto al año 1.

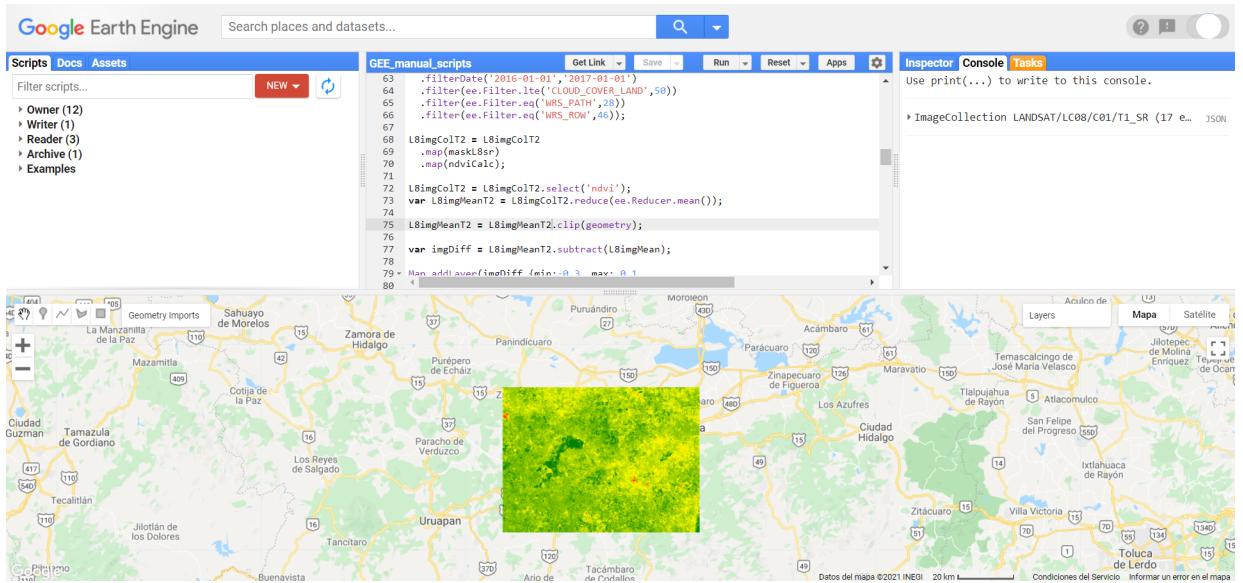
```
L8imgColT2 = L8imgColT2.select('ndvi');
var L8imgMeanT2 = L8imgColT2.reduce(ee.Reducer.mean());

L8imgMeanT2 = L8imgMeanT2.clip(geometry);

var imgDiff = L8imgMeanT2.subtract(L8imgMean);
```

Posteriormente, se agrega el resultado a la pantalla de mapas.

```
Map.addLayer(imgDiff, {min:-0.3, max: 0.1,
  palette:[ 'FF0000', 'FFFF00', '008000' ] },
  'Diferencia NDVI 2015 vs 2016');
```



Por último, para exportar el resultado se ingresan varios parámetros a la función que incluyen: la imagen a exportar, el nombre para guardar el archivo, la escala de la imagen (tamaño de pixel en metros), formato a utilizar y si se desea guardar dentro de un folder del Google Drive. Estos argumentos, al igual que otras funciones en GEE se pasa dentro de un diccionario.

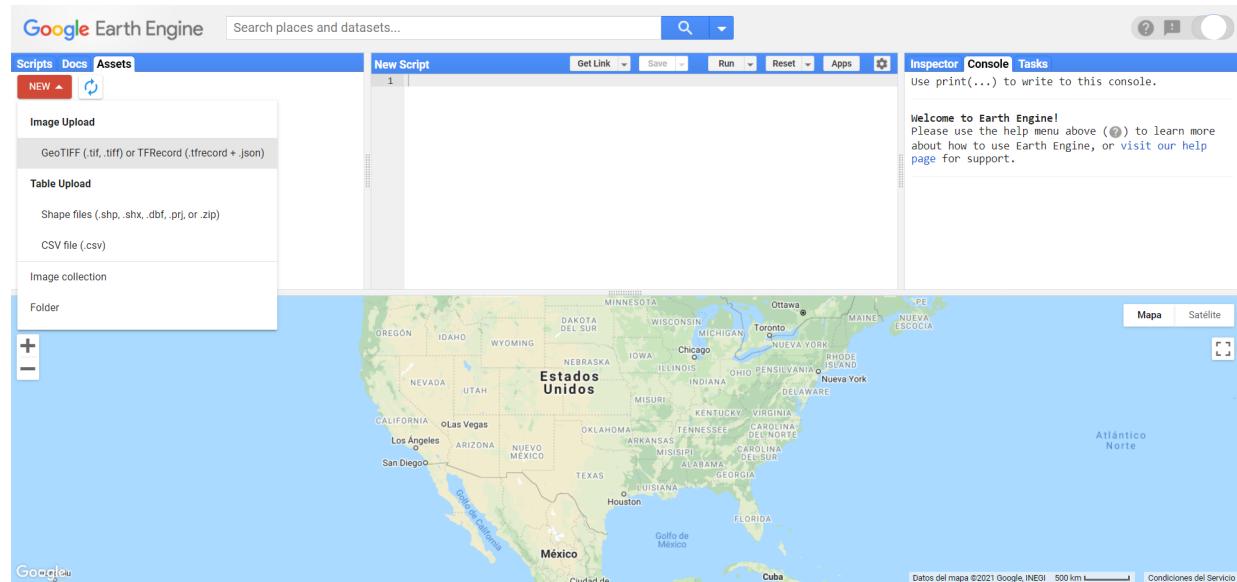
```

Export.image.toDrive({
  image: imgDiff,
  description: 'DiferenciaNDVI_2016-2017',
  scale: 30,
  fileFormat: 'GeoTIFF',
  folder: 'DiferenciaNDVIL8'
});

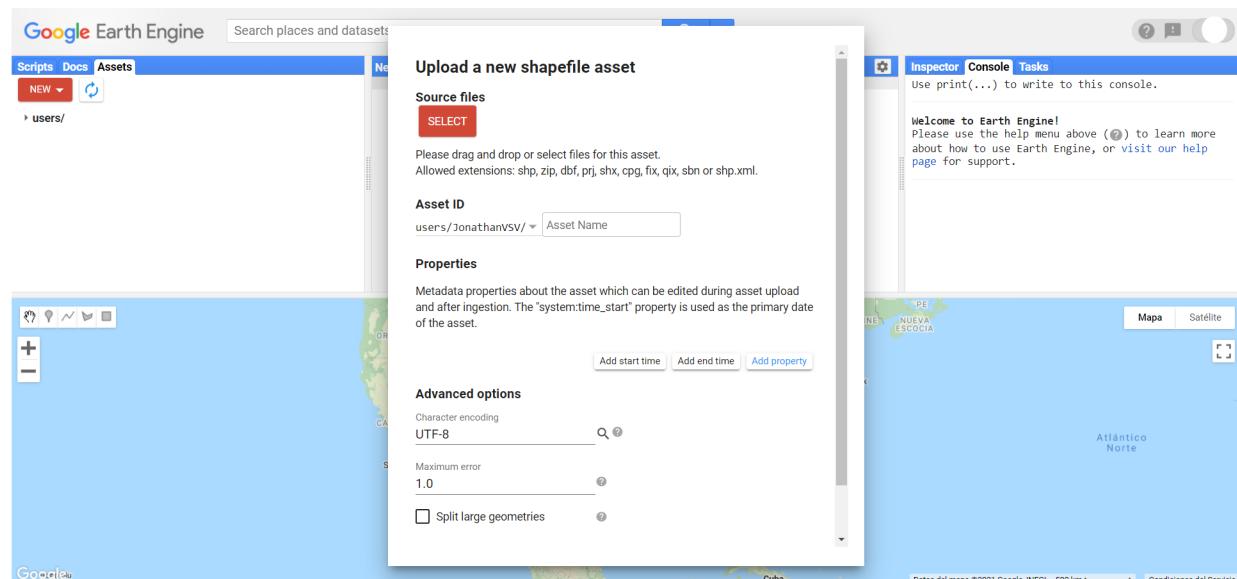
```

## Importación de información a GEE (Assets)

Se pueden importar varios tipos de archivos a GEE para utilizarlos en la API: imágenes, archivos vector (shapefiles), archivos separados por comas (csv), colecciones de imágenes exportadas por el mismo GEE y folders. Para importar cualquier de estos tipos de archivos simplemente se tiene que ingresar a la pestaña de **Assets** en el panel izquierdo y después dar click en NEW e indicar el archivo que se quiere importar a GEE.

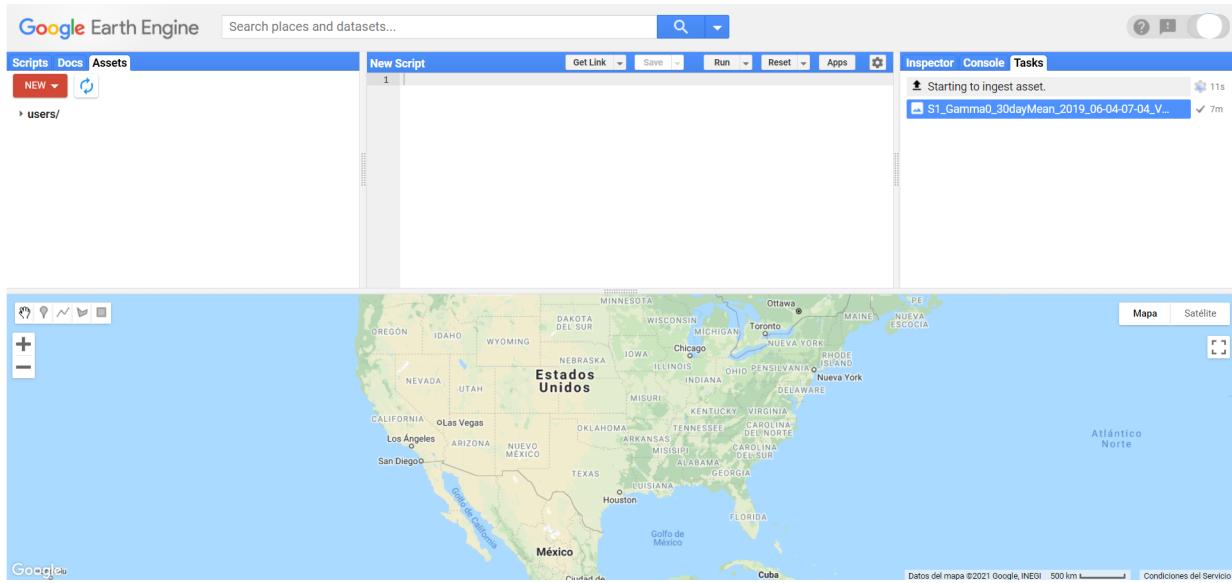


Posteriormente se da click en SELECT y se seleccionan los arhivos que se quieren subir. Además, directamente en esta ventana se pueden agregar algunos campos a la información. Por último, se da click en UPLOAD (ubicado en la esquina inferior derecha de esta ventana) para subir los archivos a la cuenta de GEE (ocupando espacio en Google Drive). Recordar que para archivos shapefile sólo se aceptan archivos con las extensiones: shp, zip, dbf, prj, shx, cpg, fix, qix, sbn o shp.xml. Si se agrega algún otro archivo que tenga una extensión distinta a las anteriores va a arrojar un error.



Una vez que se haya dado click en UPLOAD, el progreso en la subida del archivo a GEE se puede consultar en la pestaña de TASKS. Un vez terminado, el archivo podrá ser accesado dentro de la pestaña de **Assets**.

A veces no aparece el archivo recién importado, por lo cual se sugiere refrescar los elementos de esta sección presionando el botón que se encuentra a la derecha de NEW.



Al igual que en la pestaña donde se organizan los repositorios y los códigos, en la pestaña de **Assets** se puede organizar la información en folders.

## Clasificación supervisada

Dentro de GEE se pueden hacer clasificaciones supervisadas con algunos algoritmos populares como CART (Classification and Regression Trees), Random Forests, Naive Bayes, Gradient Tree Boost, Support Vector Machine, MaxEnt (Maximum Entropy) y Decision Trees. Como todos los demás objetos dentro de GEE tienen su propio tipo de objetos contenedor.

### Classifier

Los objetos `ee.Classifier` son los objetos que van a permitir trabajar con clasificadores dentro de GEE. Algunos ejemplos de estos son: `ee.Classifier.smileCART`, `ee.Classifier.smileRandomForest`, `ee.Classifier.amnhMaxent`, entre otros. De nuevo, todos los algoritmos disponibles en GEE se pueden consultar en la página de ayuda, en la pestaña de referencia (Reference) en el apartado de librerías del cliente (Client libraries).

Debido a que estos clasificadores trabajan en un esquema supervisado, requieren de la provisión de muestras de entrenamiento. Esto se puede hacer directamente en la API, aunque quizás no sea lo más cómodo para el usuario. En este paso se recomienda capturar los datos de entrenamiento en algún SIG, p.ej., QGIS, y luego importar el archivo vector a GEE (ver sección importar datos a GEE).

### Ejercicio 4: Clasificación

En este ejercicio se hará una clasificación supervisada utilizando el algoritmo Random Forests. Para ello se utilizarán 7 polígonos como sitios de entrenamiento, pertenecientes a 4 clases: Bosque, No bosque, Agua y Urbano. Estos polígonos se definirán directamente en la API a través de sus coordenadas como objetos `ee.Geometry` de tipo rectángulo. Posteriormente, cada polígono se transforma en un objeto de tipo atributo (`Feature`) y se le asigna la propiedad de “clase” como número a través de un diccionario. Por último, se hace una colección de atributos.

```
var bosque1 = ee.Geometry.Rectangle(-101.53892, 19.74148,
    -101.51906, 19.72362);
var bosque2 = ee.Geometry.Rectangle(-101.64826, 19.49264,
    -101.62839, 19.47545);
var noBosque1 = ee.Geometry.Rectangle(-101.35448, 19.65992,
    -101.34509, 19.65990);
var noBosque2 = ee.Geometry.Rectangle(-101.56907, 19.58312,
    -101.54939, 19.56435);
var urbano = ee.Geometry.Rectangle(-101.20999, 19.71300,
    -101.19688, 19.70338);
var agua1 = ee.Geometry.Rectangle(-101.60826, 19.66885,
    -101.58569, 19.64316);
var agua2 = ee.Geometry.Rectangle(-101.74686, 19.43937,
    -101.733453, 19.427012);

var poligonos = ee.FeatureCollection([
    ee.Feature(noBosque1, {'clase': 0}),
    ee.Feature(noBosque2, {'clase': 0}),
    ee.Feature(bosque1, {'clase': 1}),
    ee.Feature(bosque2, {'clase': 1}),
    ee.Feature(agua1, {'clase': 2}),
    ee.Feature(agua2, {'clase': 2}),
    ee.Feature(urbano, {'clase': 3}),
]);

```

```
Map.addLayer(poligonos);
```

A continuación se vuelve a filtrar la colección de Landsat 8 por fecha, porcentaje de nubosidad sobre la superficie terrestre, y un filtro espacial. En este caso, se usó la función `filterBounds` que permite filtrar espacialmente la colección de acuerdo a la extensión del polígono de interés. Por último, se aplica la función para enmascarar nubes a todas las imágenes de la colección usando `map`.

```
var L8imgCol = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR")
  .filterDate('2015-01-01','2016-01-01')
  .filter(ee.Filter.lte('CLOUD_COVER_LAND',50))
  .filterBounds(poligonos)
  .map(maskL8sr);
```

El siguiente paso consta de reducir la colección para generar una sola imagen con el valor promedio para todas las bandas. Después, se corta la imagen a la extensión del área de estudio. Ya que no se van a utilizar todas las bandas de la imagen para la clasificación, se indica el nombre de las bandas que se van a utilizar dentro de una lista. Por último, se van a renombrar las bandas para recordar más fácilmente el espectro de cada banda.

```
var L8imgMean = L8imgCol.reduce(ee.Reducer.mean())
  .select(['B1_mean','B2_mean','B3_mean','B4_mean','B5_mean','B6_mean','B7_mean'])
  .rename(['CB','B','G','R','NIR','SWIR1','SWIR2'])
  .clip(geometry);

print(L8imgMean);
```

Una vez que ya se tiene la imagen sobre la que se va a hacer la clasificación, se muestran los valores de los pixeles que se encuentren dentro de cada polígono de los datos de entrenamiento utilizando la función `sampleRegions`. A esta función hay que indicar la colección de atributos que se va a utilizar, así como la propiedad de los atributos que indica la clase y la escala a la que se van a muestrear los pixeles en m. El resultado de este procedimiento corresponde a una colección de atributos, en la cual cada atributo corresponde a un pixel dentro de un polígono de entrenamiento. De tal manera, para cada atributo se indica la clase a la que pertenece, así como el valor en cada una de las bandas en la imagen muestreada.

```
var training = L8imgMean
  .sampleRegions({
    collection: poligonos,
    properties: ['clase'],
    scale: 30
  });
```

El siguiente paso es entrenar al clasificador utilizando la colección de atributos anterior. En este caso, se va a utilizar el algoritmo Random Forests. Para llamar este algoritmo se utiliza `ee.Classifier.smileRandomForest()`. En este caso, se utilizarán 30 árboles de clasificación. Posteriormente, se entrena este algoritmo mediante la función `train`. Los argumentos para la función `train` se pasan como un diccionario el cual indica la colección de atributos a utilizar para el entrenamiento, el nombre de la clase objetivo y el nombre de las propiedades de la imagen, es decir, el nombre de las bandas.

```
var trainedClassifier = ee.Classifier.smileRandomForest(30).train({
  features: training,
  classProperty: 'clase',
  inputProperties: ['CB','B','G','R','NIR','SWIR1','SWIR2']
});
```

Una vez entrenado el clasificador, se puede clasificar la imagen completa utilizando la función `classify`. Como argumentos de esta función se indica el clasificador entrenado que se desea utilizar. El resultado de la función `classify` va a ser un objeto de tipo `ee.FeatureCollection` que contiene la clase predicha en el campo `classification`. Por último, se agrega a la pantalla de mapas para ver los resultados.

```
var classifiedImg = L8imgMean.classify(trainedClassifier);

Map.addLayer(classifiedImg, {min:0, max:3,
  palette:[ '#fcff21', '#20da25', '#05a9da', '#dadada' ]},
  'Clasificación RF L8');
```

Una vez que se tiene la clasificación del área de interés se van a cargar los datos de verificación para calcular la matriz de error y la precisión total de la clasificación. Primero se carga la información de los polígonos de verificación.

```
var bosque3 = ee.Geometry.Rectangle(-101.62016,19.61895,
-101.60543,19.60492);
var noBosque3 = ee.Geometry.Rectangle(-101.313746,19.716531,
-101.306380,19.709505);
var urbano2 = ee.Geometry.Rectangle(-101.609880,19.517665,
-101.602523,19.510752);
var agua3 = ee.Geometry.Rectangle(-101.272720,19.612100,
-101.272668,19.611594);

var poligonosVerif = ee.FeatureCollection([
  ee.Feature(noBosque3, {'clase': 0}),
  ee.Feature(bosque3, {'clase': 1}),
  ee.Feature(agua3, {'clase': 2}),
  ee.Feature(urbano2, {'clase': 3}),
]);
```

El siguiente paso es extraer la información de la imagen clasificada de acuerdo con la extensión de los datos de verificación. Este paso es exactamente igual al que se hizo con los datos de entrenamiento, pero ahora utilizando los datos de verificación y la imagen clasificada en lugar de la imagen de reflectancia. De igual manera, el resultado de este paso es una colección de atributos.

```
var validating = classifiedImg
.sampleRegions({
  collection: poligonosVerif,
  properties: ['clase'],
  scale: 30,
});
```

El siguiente paso es calcular la matriz de error mediante la función `errorMatrix` indicando en primer lugar el campo de la clase “verdadera” (`'clase'`) y en segundo lugar el campo de la predicción (`'classification'`). Después se van a mostrar estos dos objetos en la consola mediante `print` para inspeccionar su contenido.

```
var validErrMat = validating.errorMatrix('clase', 'classification');
var validAcc = validErrMat.accuracy();

print('Validation error matrix: ', validErrMat);
print('Validation overall accuracy: ', validAcc);
```

Para exportar estos resultados primero hay que pasar los datos a tipo atributo. Para ello, se van a crear atributos sin geometría (que es lo que indica el `null` en el primer campo del `ee.Feature`), seguido de un diccionario que contiene los datos que se desean exportar. En el caso de la matriz de error, primero se debe convertir a un arreglo mediante la función `.array()`.

```
var expConfMatrix = ee.Feature(null, {matrix: validErrMat.array()});
var expAccuracy = ee.Feature(null, ee.Dictionary({Accuracy: validAcc}));
```

Posteriormente se define una función para exportar (`ExportTable`) y se utiliza para exportar la matriz de error y el valor de precisión total. Esta función toma dos argumentos, el objeto a exportar y el nombre que se le va a dar al archivo a exportar. Debido a que la función `Export.table.toDrive` sólo permite exportar objetos de tipo `ee.FeatureCollection`, se debe meter a los objetos de tipo `ee.Feature` en un contenedor `ee.FeatureCollection`. Por último, se debe indicar que el formato para guardar el archivo es uno separado por comas (CSV) y se puede indicar el folder dentro del que se quiere guardar la tabla exportada.

```
var exportTable = function(feature, name){
  Export.table.toDrive({
    collection: feature,
    description: name,
    folder: 'Clasificacion_RF',
    fileFormat: 'CSV'
  });
}

exportTable(ee.FeatureCollection(expConfMatrix), 'confMatrix');
exportTable(ee.FeatureCollection(expAccuracy), 'validAcc');
```