CÓMO USAR GOOGLE EARTH ENGINE

Y NO FALLAR EN EL INTENTO

Jonathan Vidal Solórzano Villegas
y Gabriel Alejandro Perilla Suárez

CÓMO USAR GOOGLE EARTH ENGINE

Y NO FALLAR EN EL INTENTO

Centro de Investigaciones en Geografía Ambiental
Universidad Nacional Autónoma de México
Instituto de Investigación de Recursos Biológicos Alexander von Humboldt

CÓMO USAR GOOGLE EARTH ENGINE

Y NO FALLAR EN EL INTENTO

Jonathan Vidal Solórzano Villegas
y Gabriel Alejandro Perilla Suárez

UNAM

Universidad Nacional Autónoma de México

INSTITUTO HUMBOLDT COLOMBIA

Instituto de Investigación de Recursos Biológicos Alexander von Humboldt

Catalogación en la publicación UNAM. Dirección General de Bibliotecas y Servicios Digitales de Información
Nombres: Solórzano Villegas, Jonathan Vidal, autor. | Perilla Suárez, Gabriel Alejandro, autor.
Título: Cómo usar Google Earth Engine y no fallar en el intento / Jonathan Vidal Solórzano Villegas y Gabriel Alejandro Perilla Suárez. **Descripción:** Primera edición. | Morelia, Michoacán de Ocampo: Universidad Nacional Autónoma de México, Centro de Investigaciones en Geografía Ambiental; Bogotá, Colombia : Instituto de Investigación de Recursos Biológicos Alexander von Humboldt, 2022.
Identificadores: LIBRUNAM 2167765 (libro electrónico) | ISBN 978-607-30-6696-9 (UNAM) | ISBN 978-958-5183-55-1 (IAvH) CO. **Temas:** Computación en nube. | Sistemas de información geográfica. | Almacenamiento de datos. | Google Earth Engine (Programa para computadora). **Clasificación:** LCC QA76.585 (libro electrónico) | DDC 004.6782—dc23

Cómo usar Google Earth Engine y no fallar en el intento / Jonathan Vidal Solórzano Villegas, Gabriel Alejandro Perilla Suárez – 1 edición. - Bogotá, D.C. **Instituto de Investigación de Recursos Biológicos Alexander von Humboldt, 2022.**
200 páginas.
Incluye referencias bibliográficas, imágenes, tablas, gráficas
ISBN digital: 978-958-5183-55-1 (IAvH) CO | ISBN 978-607-30-6696-9 (UNAM)
1. Programa informático **2.** Estructuras de datos **3.** Datos geoespaciales **4.** Análisis de datos medioambientales **5.** Tecnología geoespacial **6.** Guía introductoria **I.** Solórzano Villegas, Jonathan Vidal **II.** Perilla Suárez, Gabriel Alejandro **III.** Instituto de Investigación de Recursos Biológicos Alexander von Humboldt.
Número de contribución: 624 **Registro en el catálogo Humboldt:** 15062
CEP – Biblioteca Francisco Javier Matís - Instituto Alexander von Humboldt

Este libro ha sido arbitrado por pares académicos y fue posible gracias al Proyecto PE117519-Programa de Apoyo a Proyectos para la Innovación y Mejoramiento a la Enseñanza (PAPIME), DGAPA, UNAM.

Este trabajo tiene algunos derechos reservados según lo especifica la licencia internacional *Creative Commons Attribution Non Commercial Share Alike (BY-NC-SA) 4.0,* la cual puede consultar en https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.es

Diseño y diagramación:
Jonathan Vidal Solórzano Villegas
Laura Perilla Suárez
https://www.behance.net/hermosafuria

Imagen en portada y contraportada:
Laura Perilla 2022, contiene imágenes modificadas de Copernicus Sentinel data 2018, (*20181227T151659_20181227T151707_T19PBN* y *20181227T151659_20181227T151707_T19PCN*, Sentinel-2), procesadas por la ESA.

Imágenes: todas las imágenes fueron creadas por los autores, los screenshots pueden contener pequeñas anotaciones con fines aclaratorios para los usuarios, estas imágenes cumplieron con las pautas y uso de marca de Google, de modo que aplican los términos y condiciones de marca registradas.

Todos los nombres de marcas y productos mencionados en esta obra están sujetos a la protección de marcas comerciales, marcas o patentes y las marcas comerciales o marcas comerciales registradas por sus respectivos titulares. El uso de nombres de marca, nombres de productos, nombres comunes, nombres comerciales, descripciones, etc. incluso sin una marca particular en este trabajo no puede de ninguna manera interpretarse en el sentido de que dichos nombres pueden considerarse sin restricciones con respecto a la legislación sobre marcas comerciales y protección de marcas y, por lo tanto, podrían ser utilizados por cualquiera.

Revisión académica: Andrea Pamela Flores, Victoria Nazarena Guzmán, Sandra Lucía Hernández Zetina, Xanat Antonio Némiga

Primera edición, octubre 2022. Morelia - México, Bogotá - Colombia.

D. R. © 2022:

- Universidad Nacional Autónoma de México, Ciudad Universitaria sin número, Coyoacán, C.P. 04510, Ciudad de México, México. www.unam.mx

- Centro de Investigaciones en Geografía Ambiental (CIGA, UNAM), Antigua carretera a Pátzcuaro 8701, Exhacienda de San José de la Huerta, C.P. 58190, Morelia, Michoacán de Ocampo, México. publicaciones@ciga.unam.mx

- Instituto de Investigación de Recursos Biológicos Alexander von Humboldt, Calle 72 no 12-65, piso 7, Bogotá, Colombia. comunicaciones@humboldt.org.co
http://www.humboldt.org.co/es/

ISBN obra digital Colombia: 978-958-5183-55-1
ISBN obra digital México: 978-607-30-6696-9

Esta edición y sus características son propiedad de la Universidad Nacional Autónoma de México y del Instituto de Investigación de Recursos Biológicos Alexander von Humboldt. Prohibida la reproducción total o parcial por cualquier medio sin la autorización escrita del titular de los derechos patrimoniales.

AGRADECIMIENTOS

La presente publicación recibió apoyo financiero del Programa de Apoyo a Proyectos para la Innovación y Mejoramiento a la Enseñanza, Dirección General Asuntos del Personal Académico, Universidad Nacioanl Autónoma de México (PAPIME PE117519, https://lae.ciga.unam.mx/proyectos/geomatica/).

Este libro se elaboró con el apoyo de la Universidad Nacional Autónoma de México, México, puntualmente:

Centro de Investigaciones en Geografía Ambiental (CIGA),
La editorial del CIGA

Y el Instituto de Investigación de Recursos Biológicos Alexander von Humboldt, Colombia, puntualmente:

La oficina Jurídica
La oficina de Comunicaciones
La línea 13, Análisis y Modelamiento, Programa de Evaluación y Monitoreo de la Biodiversidad, de la Subdirección de Investigaciones.

Agradecemos a Jean-François Mas por concebir la idea inicial de este manual e invitar a los autores a participar en dicho proyecto.

De igual manera, damos las gracias a Andrea Pamela Flores, Victoria Nazarena Guzmán, Sandra Lucía Hernández Zetina, Xanat Antonio Némiga y a dos revisores anónimos por la minuciosa revisión del manuscrito y sus valiosos comentarios que ayudaron a mejorar la calidad de este documento.

Agradecemos también a los desarrolladores de la API de libre acceso y uso gratuito (para propositos educativos y de investigación sin fines de lucro) que se presenta en este libro (Google Earth Engine) y a los desarrolladores de programas de código abierto que se usaron, en parte, para la elaboración de esta obra (R y LaTeX), así como algunos de sus paquetes (Rmarkdown, knitr, bookdown).

La imagen de la portada y contraportada fue creada por Laura Perilla, a partir de modificaciones de las imágenes "20181227T151659_20181227T151707_T19PBN" y "20181227T151659_20181227T151707_T19PCN", Copernicus Sentinel 2 data (2018), procesadas por la European Space Agency (ESA).

DECLARACIÓN DE AFILIACIÓN

Google, Google Earth Engine, Google Drive, Google Cloud o cualquier otra marca comercial de Google que sea mencionada o referenciada a lo largo del presente documento son marcas comerciales propiedad de Google LLC y no nuestra. Por lo tanto, el presente libro no es un producto oficial de Google ni se encuentra afiliado, o respaldado de ninguna manera por Google. A lo largo del texto mencionaremos diferentes productos de Google LLC, pero es importante recalcar que dichos productos y nombres son propiedad de Google, de modo que a continuación declaramos los nombres correctos de cada una de dichas marcas comerciales:

Google Cloud™ enterprise services.
Google Drive™ online storage service.
Google Earth Engine™ analytics platform.
Google Earth™ mapping service.
Google Maps™ mapping service.
Google Groups™ discussion forums
TensorFlow™ open-source software library.

ADVERTENCIA DE ENLACES

Este libro contiene múltiples enlaces (hipervínculos) a páginas web de terceros, de modo que en un futuro, dichos enlaces podrían caducar, cambiar de dirección o simplemente no contar con una actualización. De la misma manera, en su labor de mejorar y actualizar sus herramientas, Google puede cambiar algunas funciones, o los IDs de sus productos en sus catálogos (o realizar cualquier otro cambio que afecte la plataforma de Google Earth Engine). Estas circunstancias podrían impedir que algunos de los ejemplos aquí mostrados puedan ser ejecutados al pie de la letra. Por lo tanto, se recomienda a cualquier lector estar al tanto de posibles futuros cambios en los enlaces incluidos, en las actualizaciones de Google Earth Engine o de su catálogo.

Existe un (grupo virtual) Google Groups dedicado a las discusiones y las dudas de todos los usuarios de Google Earth Engine, si tiene alguna duda de cómo usar la herramienta siempre puede acceder, preguntar, contestar y consultar en ese grupo en el siguiente enlace:
https://groups.google.com/g/google-earth-engine-developers

ÍNDICE

AGRADECIMIENTOS VI
DECLARACIÓN DE AFILIACIÓN VII
ADVERTENCIA DE ENLACES VII
ÍNDICE 1
1 INTRODUCCIÓN 4
1.1 Propósito del libro 4
1.2 Organización del libro 5
1.3 Descripción general de Google Earth Engine 5
2 PRIMEROS PASOS 11
2.1 Registro para el uso de GEE JavaScript API 11
2.2 Elementos básicos de la interfaz gráfica 14
2.3 Programación en GEE 24
3 INTERFAZ DE USUARIO 27
3.1 Impresión en consola 27
3.2 Interacción con la pantalla de mapa 28
3.3 Creación de gráficos 30
3.4 Exportación de objetos fuera de GEE 32
4 IMPORTACIÓN DE INFORMACIÓN A GEE 37
5 TIPOS DE OBJETOS 44
5.1 Objetos del cliente y del servidor 44
5.2 Tipos de objetos del lado del cliente 45
5.3 Tipo de objetos del lado del servidor 52
6 EE.GEOMETRY 64
6.1 Información y metadatos 64
6.2 Creación de geometrías 64
6.3 Métodos comunes 70

7 EE.FEATURE 72
7.1 Información y metadatos 72
7.2 Visualización de vectores 72
7.3 Creación de vectores 73
7.4 Métodos comunes 73
8 EE.FEATURECOLLECTION 81
8.1 Información y metadatos 82
8.2 Creación de colecciones de vectores 83
8.3 Visualización de colecciones de vectores 87
8.4 Métodos comunes 89
9 EE.IMAGE 111
9.1 Información y metadatos 112
9.2 Visualización de una imagen 114
9.3 Métodos comunes 115
10 EE.IMAGECOLLECTION 134
10.1 Información y metadatos 134
10.2 Creación de colecciones de imágenes 135
10.3 Visualización de colecciones de imágenes 137
10.4 Métodos comunes 138
11 USO CONJUNTO DE VECTORES E IMÁGENES 163
11.1 Métodos comunes 163
11.2 Interpolación de un vector a una imagen 165
12 CLASIFICACIÓN SUPERVISADA 171
12.1 Clasificadores 171
12.2 Realización de la fase de entrenamiento 172
12.3 Obtención de la clasificación 173
12.4 Evaluación de la clasificación 173
13 REFLEXIONES FINALES 185
14 REFERENCIAS 187

Introducción

1

INTRODUCCIÓN

1.1 Propósito del libro

Este manual pretende ser una guía introductoria a Google Earth Engine (GEE) que ayude a los nuevos usuarios a entender su estructura, funcionamiento, capacidades, y facilitar el uso de esta poderosa herramienta gratuita. Existen algunos artículos previos que han presentado las características básicas de GEE, sin embargo, la mayoría están escritos en inglés (Gorelick *et al*., 2017; Amani *et al*., 2020), aunque en español existen algunas publicaciones también (ver Perilla y Mas, [2020]). La documentación propia de GEE ya existe en inglés, pero con este manual escrito en español se busca ampliar el público objetivo y aumentar el alcance del programa hacia países hispanoparlantes.

Es importante aclarar que esta guía es solamente introductoria, por lo tanto, hay muchas funciones que se pueden realizar en GEE que quedan fuera de este documento (por ejemplo, los análisis de series de tiempo o la creación de aplicaciones). Además, debido a su carácter introductorio, este manual está dirigido a usuarios con y sin conocimientos previos de programación. GEE está en constante actualización, por lo que este manual se enfoca en el corazón del funcionamiento de la API de GEE, que es lo que menos cambia en el tiempo. Sin embargo, aconsejamos buscar información adicional sobre las actualizaciones de GEE, así como revisar su documentación oficial.

Para aprender a usar GEE, como sucede con cualquier programa informático, se debe llevar a cabo un proceso de prueba y error, mediante el desarrollo de códigos propios. Los autores esperamos que esta herramienta ayude a los nuevos usuarios a comenzar a utilizar GEE y a alcanzar sus objetivos particulares de investigación.

Idealmente, para sacar el máximo provecho de este manual, los lectores deben contar con una formación mínima en programación básica, para comprender la aplicación de términos relacionados con el uso de variables y estructuras de datos como listas y diccionarios, entre otras, que se usan en los ejemplos. De no tenerla, sugerimos que paralelamente a la lectura del manual se revisen conceptos básicos de programación en JavaScript.

1.2 Organización del libro

El manual está organizado en doce capítulos que cubren diferentes aspectos de esta API (*Application Programming Interface*, o interfaz de programación de aplicaciones). El primer capítulo describe algunos aspectos generales de GEE. El siguiente capítulo detalla los pasos a seguir para poder usar GEE, así como los elementos básicos de dicha herramienta. Posteriormente, se presentan tres capítulos que muestran algunas funciones básicas para ayudar al usuario a interactuar con la API, cómo importar información a GEE y las diferencias entre la programación del lado del servidor y del usuario. A continuación, en cinco capítulos se describen los tipos de objetos más frecuentemente utilizados en GEE: geometrías, vectores, colecciones de vectores, imágenes y colecciones de imágenes, así como sus métodos. Después, se describen algunos métodos más avanzados para utilizar en conjunto información vectorial y ráster. El último capítulo muestra el procedimiento para realizar una clasificación supervisada en la API.

El manual incluye ejemplos que ayudarán a entender la sintaxis y experimentar con distintos tipos de datos, provenientes de diversas fuentes. De manera particular, en los capítulos sobre imágenes y colecciones de imágenes se incluyen ejemplos más prácticos con aplicaciones reales que tratan de integrar varios de los métodos revisados. Por último, este manual cuenta con una serie de ejercicios complementarios y explicativos, los cuales pueden ser implementados directamente en línea. El enlace para acceder a este material se encontrará más adelante.

1.3 Descripción general de Google Earth Engine

¿Qué es Google Earth Engine?

Google Earth Engine (GEE) es una plataforma desarrollada por Google que permite realizar procesamientos geoespaciales a gran escala, utilizando bases de datos con millones de estos. Uno de los principales intereses de esta plataforma es reducir el tiempo invertido en el preprocesamiento y facilitar los análisis realizados con información geoespacial.

Desde hace varios años existen distintas misiones espaciales que registran datos de la Tierra a través de satélites, los cuales se almacenan en acervos de imágenes. Aunque

esta información ha sido muy útil para el desarrollo científico, siempre ha existido el problema de la *big data* (bases de datos enormes). Es decir, que dichos acervos de imágenes cuentan con millones de imágenes disponibles que resultan imposibles de procesar y analizar en su totalidad utilizando una computadora personal. Esta situación evitaba la posibilidad de aprovechar el potencial total de esta información.

GEE nace de la necesidad de aprovechar esa *big data* al ofrecer una herramienta que pueda lidiar tecnológicamente con el manejo de enormes volúmenes 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 tres elementos claves, que son los que permiten el funcionamiento de GEE: el catálogo de información, la capacidad computacional y las API.

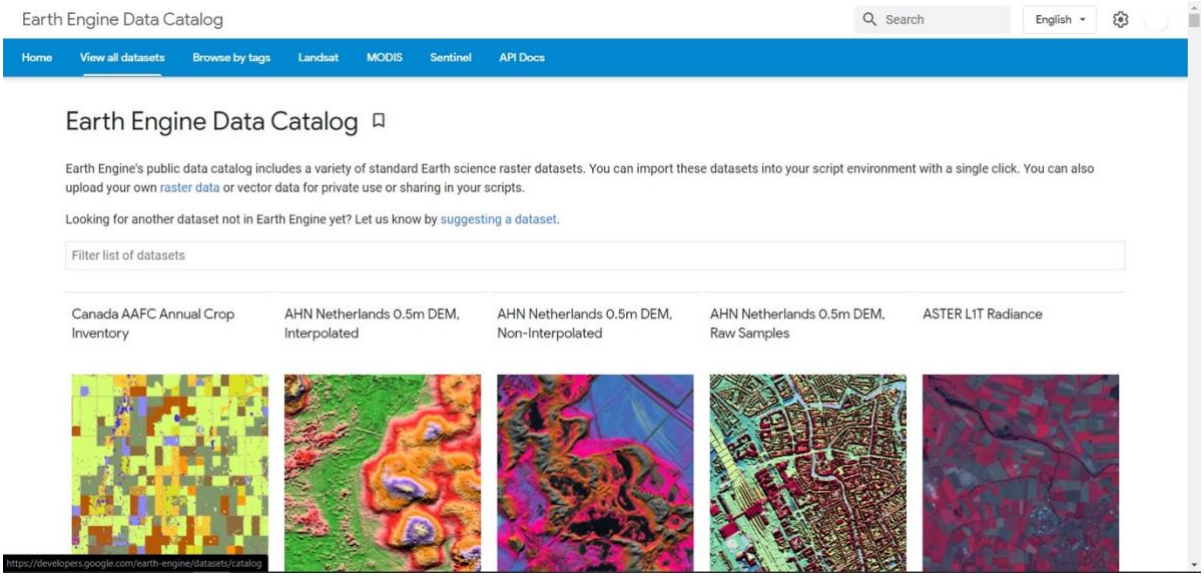


Figura 1.1: *Algunos ejemplos de fuentes de información disponibles en GEE.*

GEE recopiló la información geoespacial de diferentes fuentes alrededor del mundo y creó copias de esos datos en su propio *data center*, logrando así almacenar más de 20 petabytes (20 000 000 de gigabytes) de información en un solo lugar. El programa ofrece un amplio repositorio de información geográfica global que ya se encuentra cargada en su catálogo, lo cual facilita el acceso a diferentes fuentes de datos (Fig. 1.1). Entre la información que se puede consultar directamente en GEE se encuentran: acervos para todo el mundo de imágenes Landsat (1-9), MODIS, Sentinel (1-3, 5), SRTM, AVHRR, GOESS, ALOS, mosaicos anuales de PALSAR/PALSAR 2 y algunos productos derivados de estas imágenes u otros insumos como información climática global (WorldClim), capas de cambios en la cobertura forestal (Global Forest Cover Change), información de la altura del dosel global (Global Forest Canopy Height), mapas de densidad de carbono globales (Global Aboveground and Belowground

Biomass Carbon Density Map), datos de precipitación globales (CHIRPS), límites políticos (FAO GAUL: Global Administrative Unit Layers 2015, Country Boundaries), polígonos de áreas protegidas mundiales (WDPA: World Database on Protected Areas), densidad poblacional (GPWv411: Population Density —Gridded Population of the World Version 4.11—) y cuencas (WWF HydroSHEDS Hydrologically Conditioned DEM), entre otras.

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

Se estima que en 2010 GEE era capaz de ejecutar en cuestión de días procesamientos que en un computador personal habrían tardado 15 años.

De manera paralela al catálogo de información, GEE puso a disposición de la gente un conjunto de servidores para poder realizar rápida y eficazmente cálculos y computaciones sobre esos datos. En 2010, estos servidores tenían una capacidad más o menos equivalente a 10 000 computadores personales.

El tercer componente clave para el funcionamiento de GEE son las API. Una API es una interfaz que permite la comunicación entre nosotros (los usuarios) y los servidores de Google, proporcionando el acceso y el uso de la capacidad computacional de Google para nuestros análisis. Las API de GEE tienen el objetivo de facilitar la creación de programas, por lo cual cuentan con una serie de funciones, métodos y algoritmos preprogramados que se pueden llamar con una simple línea de código. La idea central de las API de GEE es brindar una interfaz que permita crear códigos y programas claros y concisos.

GEE ofrece dos API, una en JavaScript, a la cual se accede vía internet mediante un explorador y es la más conocida, actualizada y amigable con los usuarios. Además, es la que contiene más documentación y ayuda disponible. Por esta razón, el manual se enfocará en la API de JavaScript. Por otro lado, existe la API en Python, la cual se puede trabajar desde la consola de Python y permite —hasta cierto punto— usar complementariamente bibliotecas de Python, para así hacer procesamientos más complejos, o funcionalidades que la API de JavaScript no permite. Sin embargo, las desventajas de la API de Python incluyen el tener que instalar varias bibliotecas en la computadora para poder trabajar con GEE, el hecho de que existe mucho menos documentación y ayuda disponible para esta API, y que requiere de la actualización constante de algunas bibliotecas para su uso apropiado. A continuación se mencionan algunas de las principales ventajas de la API de JavaScript respecto de la de Python:

- Los datos pueden ser consultados directamente en la nube (no se requiere descargarlos para trabajar con ellos), lo que ahorra tiempo y espacio de almacenamiento para el usuario. En cambio, si se quiere usar alguna biblioteca de Python será necesario descargar la información.
- El procesamiento se hace en la nube, a través de internet, utilizando el poder de cómputo asignado para la API, lo cual reduce el gasto de memoria RAM (solo necesita una conexión estable y el consumo de RAM depende del consumo del navegador). En la API de Python hay un mayor consumo de memoria RAM (para ejecutar la consola de Python y sus bibliotecas) y requiere igualmente una conexión estable a internet.
- Su interfaz es mucho más amigable con los usuarios, ya que ofrece una plataforma más interactiva para programar y sencilla para enseñar.
- Se pueden realizar fácilmente consultas de las colecciones de imágenes y sus metadatos antes de decidir importarlos. En cambio, la API de Python no permite realizarlas de manera tan sencilla, de modo que hay que tener una absoluta claridad de las propiedades del conjunto de datos que se van a utilizar.
- Otra gran ventaja de la plataforma JavaScript 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 control de versiones de *Git*), lo cual facilita el seguimiento y comparación de cambios entre las versiones (y permite regresar a versiones anteriores). Además, se pueden generar repositorios compartidos para generar proyectos colaborativos.

Git es un programa de distribución libre que permite llevar un control de diferentes versiones de un código.

Ejemplos de aplicaciones realizadas con GEE

GEE se ha utilizado para diversos estudios enfocados en analizar la superficie terrestre mediante sensores remotos, especialmente imágenes multiespectrales (por ejemplo: Landsat, Sentinel-2, MODIS) y en algunos casos imágenes de radar de apertura sintética (por ejemplo, Sentinel-1). Algunos ejemplos de sus usos son:

- Estimar variables biofísicas de la superficie terrestre (Campos-Taberner et al., 2018).
- Monitorear asentamientos humanos (Trianni et al., 2014).
- Analizar la disponibilidad de imágenes y observaciones despejadas (Solórzano et al., 2020a; Solórzano et al., 2020b).
- Mapear campos de cultivo (Xiong et al., 2017; Dong et al., 2016).
- Identificar plantaciones de palma de aceite (Lee et al., 2016).

- Mapear cicatrices de incendios (Arruda et al., 2021).
- Evaluar el grado de sequía de la superficie terrestre (Sazib et al., 2018).
- Mapear características de humedales (Slagter et al., 2018).
- Identificar agricultura protegida a alta resolucion (Perilla y Mas, 2019)
- Caracterizar islas de calor (Ravanelli et al., 2018), entre muchos otros.

Además, es una herramienta cuyo uso ha ido en aumento gracias a sus enormes capacidades de procesamiento y su rapidez para realizar análisis con información geoespacial (Kumar *et al*., 2018). Por ultimo, cabe mencionar que dentro de Google Earth Engine se pueden programar aplicaciones que facilitan el uso y consulta de resultados de cualquier código, en una interfaz amigable para cualquier usuario. Aunque esto no se revisará en el material de este manual, en la siguiente lista se pueden consultar algunos de estos ejemplos:

- Realizar análisis de series de tiempo para detectar cambios de cobertura (Arévalo *et al*., 2020; Hamunyela *et al*., 2020).
  https://parevalo-bu.users.earthengine.app/view/advanced-tstools
  https://andreim.users.earthengine.app/view/bfastmonitor

- Evaluar los cambios de la cobertura boscosa global (Hansen *et al*., 2013).
  https://glad.earthengine.app/view/global-forest-change

- Enmascarar nubes de imágenes (Mateo-García *et al*., 2018).
  https://isp.uv.es/projects/cdc/viewer_l8_GEE.html

- Mapear el cambio de hábitat de las aves del 2000 al 2021 (Perilla *et al*., 2022)
  https://biomodelos-iavh.users.earthengine.app/view/biomodelos

- Mapear plantas de energía fotovoltaica (paneles solares) (Zhang, *et al*., 2021)
  https://xunhezhang.users.earthengine.app/view/ningxia-pv-power-plants

- Monitorear anomalías termales volcánicas globales (Genzano, *et al*., 2020)
  https://nicogenzano.users.earthengine.app/view/nhi-tool

- Monitorear estrés ambiental para corales (Williamson, *et al*., 2021)
  https://mjw1280.users.earthengine.app/view/coral-reef-stress-exposure-index

GEE sugiere que para citar su herramienta en cualquier tipo de trabajo se haga referencia a Gorelick *et al*. (2017).

Primeros Pasos

2

PRIMEROS PASOS

2.1 Registro para el uso de GEE JavaScript API

Lo primero que hay que hacer para poder utilizar la API de JavaScript de GEE es ingresar a su sitio web para registrarse como usuario, por medio de la siguiente liga: https://earthengine.google.com/. Después, le damos clic en la esquina superior derecha donde dice **Sign Up** (inscribirse; Fig. 2.1).

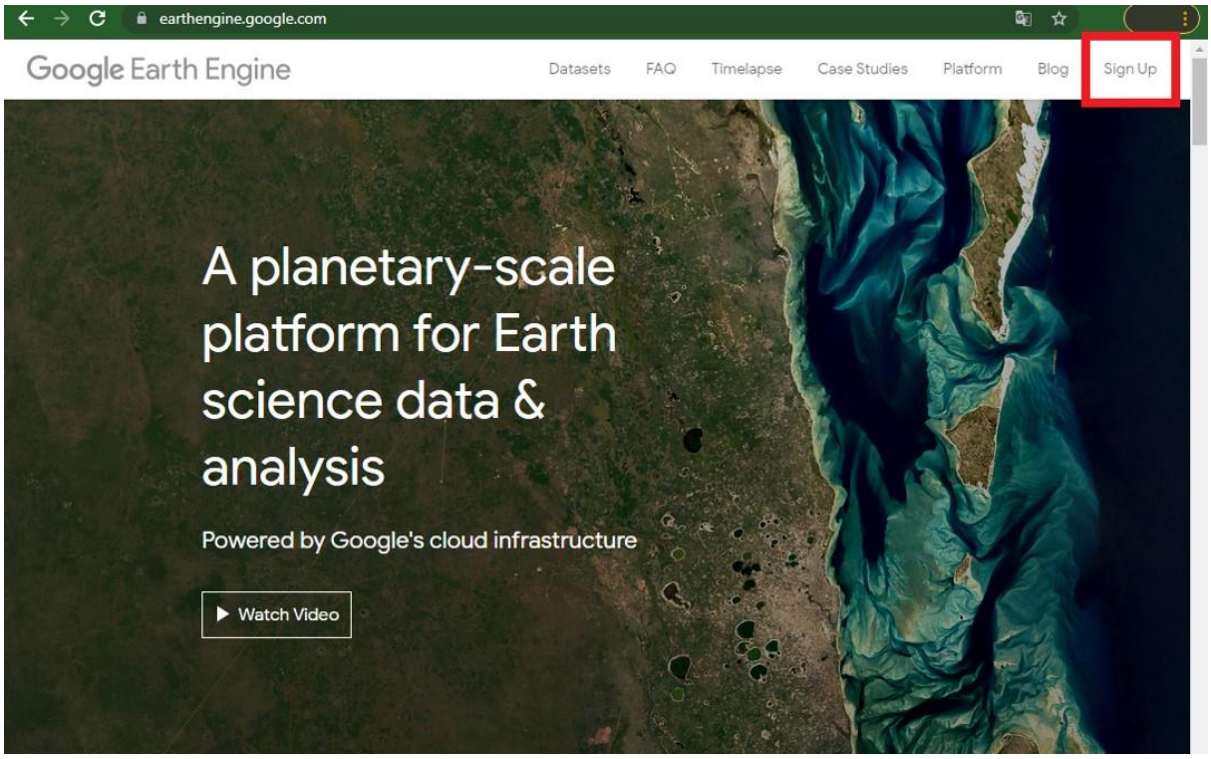


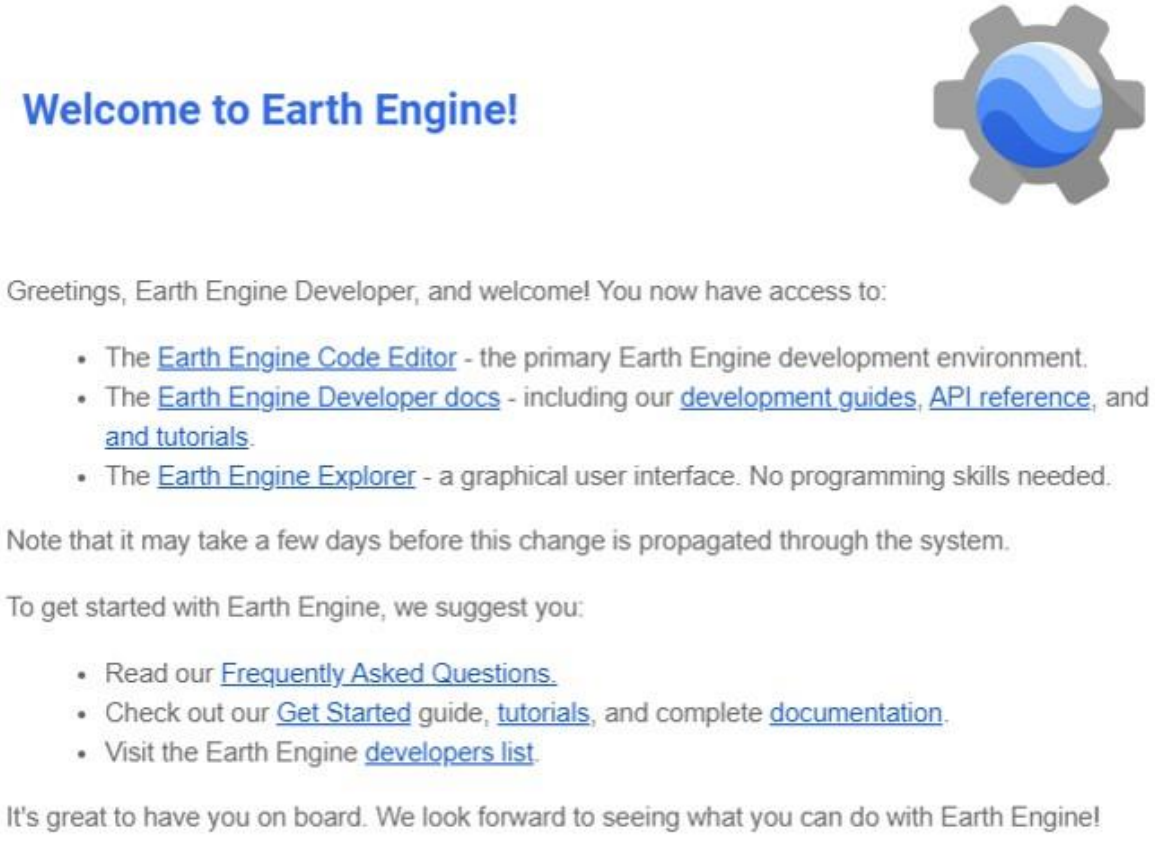
Figura 2.1: *Pantalla de inicio en la página de registro de GEE.*

Luego se ingresan todos los datos que pide el formulario. Recuerde que para utilizar GEE se requiere de una cuenta de Google (Fig. 2.2).

> Se recomienda usar 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.

Figura 2.2: Formulario de datos a llenar para registrarse en GEE.

Una vez realizado el registro, hay que esperar un tiempo (pueden ser desde minutos a días) hasta que llegue una confirmación por parte de GEE a nuestro correo informando que ya se puede hacer uso de la API (Fig. 2.3).



Welcome to Earth Engine!

Greetings, Earth Engine Developer, and welcome! You now have access to:

- The Earth Engine Code Editor - the primary Earth Engine development environment.
- The Earth Engine Developer docs - including our development guides, API reference, and and tutorials.
- The Earth Engine Explorer - a graphical user interface. No programming skills needed.

Note that it may take a few days before this change is propagated through the system.

To get started with Earth Engine, we suggest you:

- Read our Frequently Asked Questions.
- Check out our Get Started guide, tutorials, and complete documentation.
- Visit the Earth Engine developers list.

It's great to have you on board. We look forward to seeing what you can do with Earth Engine!

Figura 2.3: Ejemplo del correo de confirmación por parte de GEE para poder utilizar la API.

Una vez que tengamos dicha confirmación, podemos acceder a la API a través del siguiente enlace: https://code.earthengine.google.com/ y se accede con la cuenta de Google con la que nos registramos.

Una vez registrados, se podrá acceder al material de todos los ejercicios entrando al siguiente enlace:
https://code.earthengine.google.com/?accept_repo=users/JonathanVSV/GEE_manual

2.2 Elementos básicos de la interfaz gráfica

Una vez abierta la API de JavaScript, se observan cuatro pantallas (Fig. 2.4): la pantalla de repositorios, la pantalla de rutinas, la pantalla de mapa y la pantalla de control.

Figura 2.4: *Vista inicial de la API de GEE.*

Los elementos de la API se enlistan a continuación:

Pantalla de repositorios

Scripts

La pantalla de repositorios es el espacio donde se guardan y ordenan las rutinas o *scripts* del usuario. En ella se pueden crear repositorios y carpetas para organizar los archivos de código, como se muestra a continuación (Fig. 2.5).

Dentro de esta pantalla existen varias categorías. Las primeras tres se pueden utilizar para determinar distintos niveles de acceso para diversos usuarios (Fig. 2.6).

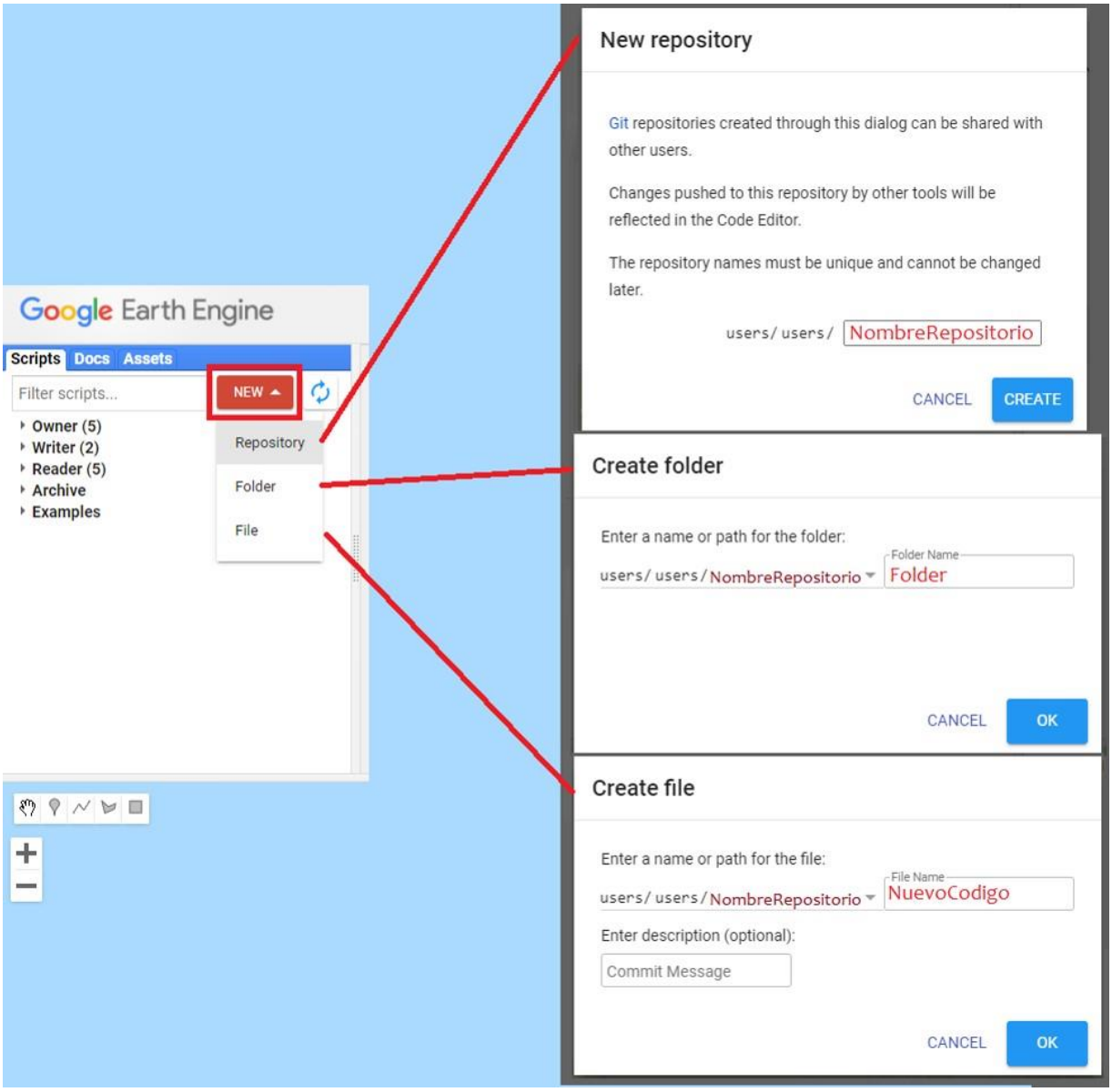


Figura 2.5: *Opciones disponibles dentro de la pestaña de Nuevo.*

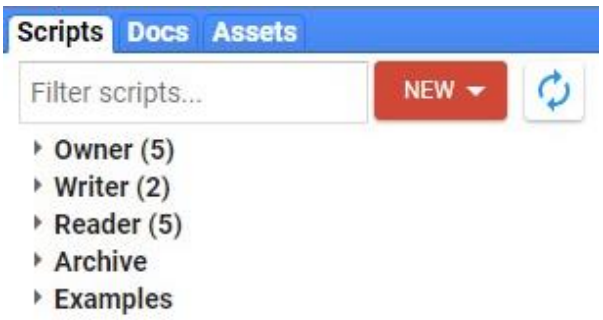


Figura 2.6: *Vista del panel de repositorios dentro de GEE.*

1. **Owner** (Propietario): en esta sección se guardan los códigos creados por el mismo usuario.
2. **Writer** (Editor): en este apartado se guardan códigos que pueden ser creados por otros usuarios, pero que estamos autorizados a modificar.
3. **Reader** (Lector): en esta parte aparecen códigos que otros usuarios nos han compartido, pero no estamos autorizados a modificar.
4. **Examples** (Ejemplos): se pueden consultar ejemplos de código para hacer algunas tareas específicas. Por ejemplo, existe un apartado específico para consultar las distintas maneras de enmascarar nubes utilizando diferentes colecciones de imágenes (por ejemplo: Landsat 4-7, Landsat 8-9, Sentinel-2, MODIS).
5. **Archive** (Archivo): se pueden guardar archivos de código que ya no se utilicen, pero que no se desea eliminar.

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 dar clic en el símbolo del engranaje a la derecha de cada repositorio (aparece una vez que se coloca el puntero sobre el nombre del repositorio; Fig. 2.7).

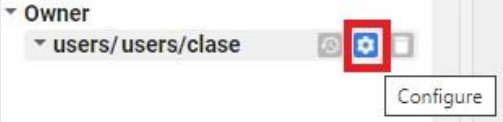


Figura 2.7: *Vista del espacio de propietario.*

Docs

Es un área donde se pueden consultar todos los métodos y algoritmos que se encuentran preprogramados y cargados en GEE (Fig. 2.8). Además, para cada método se indica lo que hace, la entrada que requiere y el tipo de objeto que se obtiene como salida, así como los argumentos del método (Fig. 2.9). Estos métodos se encuentran agrupados por objetos del servidor (los objetos se explican más adelante). Adicionalmente, tiene un buscador donde también se pueden consultar los métodos.

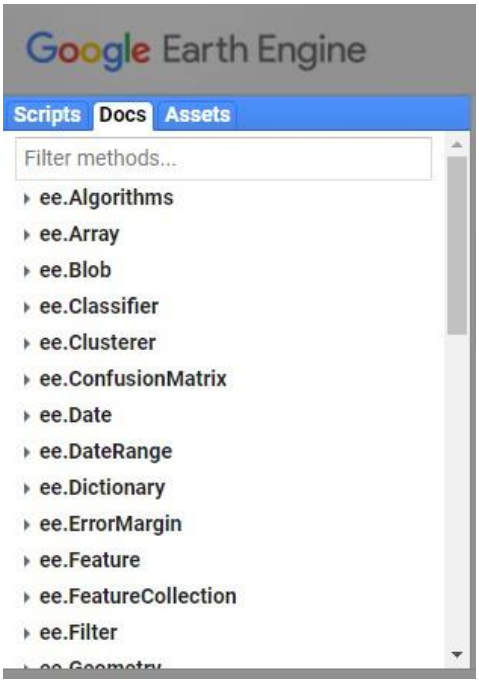


Figura 2.8: *Ejemplo de la pestaña de documentación dentro de GEE.*

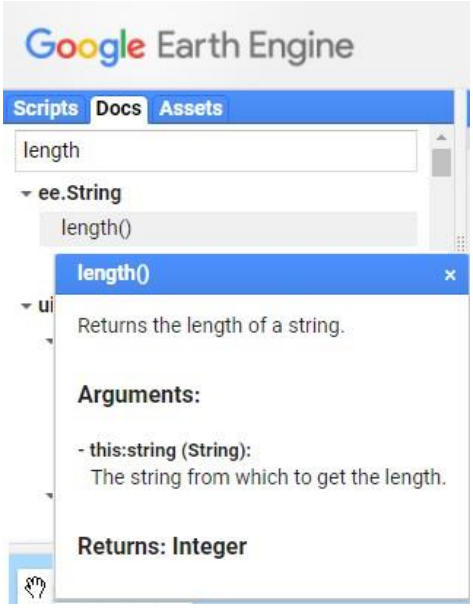


Figura 2.9: *Ejemplo de consulta de un método dentro de la sección de documentación.*

Assets

La primera vez que se accede a la pestaña de **Assets** hay que crear una carpeta principal (**Home Folder**) para almacenar la información que se desee importar a GEE. Se recomienda nombrar esta carpeta con el mismo nombre de usuario (Fig. 2.10).

Figura 2.10: *Carpeta de inicio dentro de la API.*

En este apartado el usuario puede subir y guardar su propia información para ser utilizada dentro de GEE (Fig. 2.11). Se pueden subir únicamente archivos en formato ráster, vector (***.shp*** con sus archivos auxiliares, o un ***.zip***) o separado por comas (ver sección 4 para consultar los detalles para realizar este proceso).

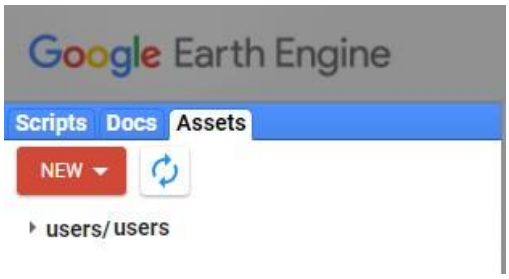


Figura 2.11: *Vista de la sección de información del usuario.*

Pantalla de rutinas

Esta ventana es donde se va a escribir el código en JavaScript (Fig. 2.12) y cuenta con varios botones en la parte superior. El primero, **Get Link**, sirve para compartir el código que se tenga abierto mediante una liga. Además, al darle clic en la flecha de despliegue (en la opción **Manage Links**) se puede hacer un manejo de los enlaces que se tengan activos, así como borrar los que ya no se utilicen (Fig. 2.13).

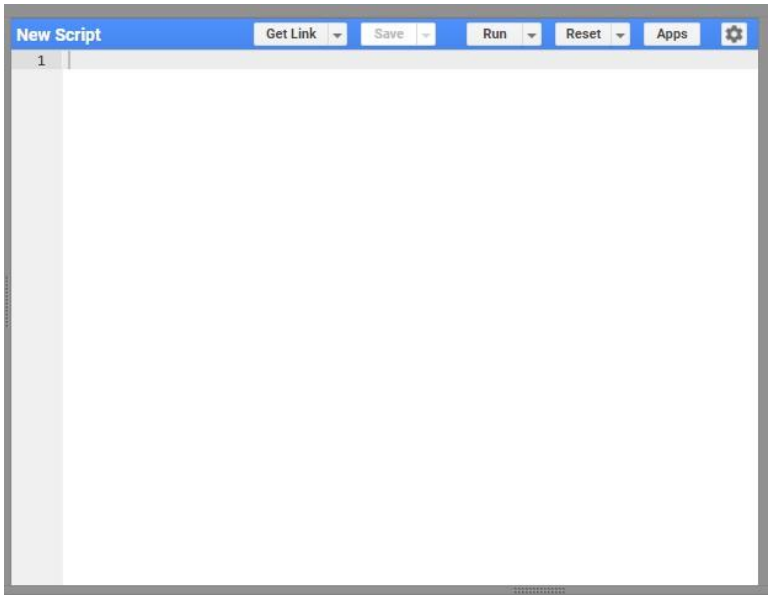


Figura 2.12: *Vista de la pantalla de rutinas o scripts.*

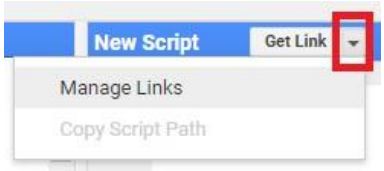


Figura 2.13: *Ubicación del botón para obtener enlaces para compartir rutinas.*

A su derecha se encuentra el botón de **Save**, el cual permite guardar el código con el que se esté trabajando (Fig. 2.14). Además, al darle clic en la flecha de despliegue se puede usar la opción **Save as** para crear una copia del código con otro nombre.

Figura 2.14: *Ubicación del botón para guardar el código.*

A continuación, está el botón **Run**, que sirve para correr, de principio a fin, el código que se muestra en la pantalla de rutinas (Fig. 2.15). Al darle clic a este botón se envía el código a los servidores de Google y se realiza el procedimiento indicado.

Figura 2.15: *Ubicación del botón para correr el código.*

A su derecha se encuentra el botón de **Reset**, el cual sirve para borrar todo el código que se tenga escrito en la pantalla de rutinas.

Después se encuentra el botón de **Apps**, que sirve para crear aplicaciones a partir del código que se encuentra en la pantalla. La creación de aplicaciones no se cubrirá en este manual, pero se refiere a programas que permiten crear una interfaz amigable con un usuario que no tenga conocimientos de GEE. De esta manera, se puede facilitar el uso de un código programado sin necesidad de acceder directamente al código. Para consultar algunos ejemplos de aplicaciones construidas en GEE se puede visitar el siguiente enlace: https://www.earthengine.app/. Por último, en el botón del engranaje se encuentran opciones para prender y apagar líneas de código como subrayar sugerencias en la pantalla de rutinas o autocompletar símbolos como ', ", (, [y {.

Cuando se corre un código en la consola con el botón de Run, este no corre directamente en los servidores de Google, sino que se transcriben a código GeoJSON, se envían a los servidores de Google y se espera una respuesta.

Para buscar un texto específico dentro del código, se debe dar clic en la pantalla de rutinas y presionar ctrl+F (algunos atajos pueden variar según la configuración del teclado). Esta pantalla permite buscar una cadena de caracteres determinada dentro del código. Además, al volver a presionar ctrl+F, también se activa el recuadro de sustitución, el cual facilita el sustituir una cadena de caracteres determinada por otra. Para consultar el listado completo de atajos se puede presionar ctrl+shift+H.

Search

Es una barra de búsqueda (**Search**) en la que se puede tratar de localizar fuentes de datos o sitios (Fig. 2.16). Resulta útil para encontrar la ruta de alguna fuente de datos o una colección en particular, así como para consultar sus metadatos, bandas y características.

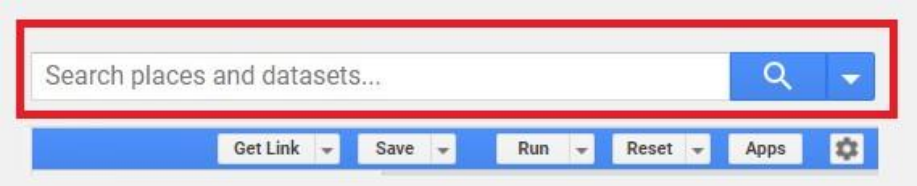


Figura 2.16: Vista de la barra de búsqueda.

Pantalla de Mapa

En esta pantalla (Fig. 2.17), se puede:

- Dibujar y visualizar puntos, polígonos, líneas o rectángulos.
- Mostrar la información que se haya indicado mediante la función `Map.addLayer`.
- Añadir un mapa base (Google Maps o Google Earth).

Figura 2.17: Vista de la pantalla de mapa, indicando las diferentes herramientas que se pueden utilizar en ella.

Gestor de capas

El gestor de capas permite prender y apagar las capas que se estén mostrando en el área del mapa (Fig. 2.18). Además, sirve para modificar las características de su visualización (por ejemplo, transparencia, color, distribución del histograma, compuesto de color). Este menú aparece una vez que se cargan capas a la pantalla de mapa.

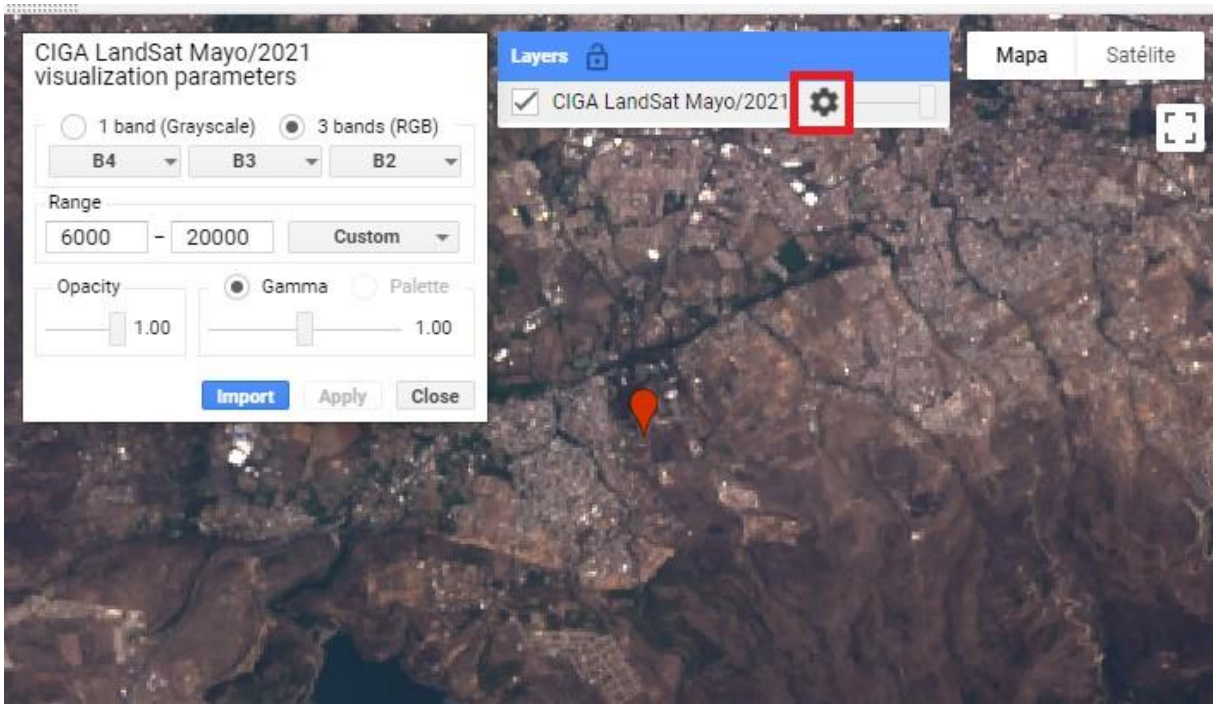


Figura 2.18: Ejemplo del uso del administrador de capas.

Pantalla de control

La pantalla de control contiene tres pestañas: **Console** (consola), **Inspector** (inspector) y **Tasks** (tareas). Esta pantalla le permite al usuario hacer algunas consultas sobre la información con la que se está trabajando o finalizar tareas de exportación de información. A continuación se describe cada una de estas pestañas.

Console

La consola consiste en la pantalla de comunicación con el servidor (Fig. 2.19). En ella se muestran los errores que se obtienen al correr un código o se puede mostrar la información indicada por la función `print`.

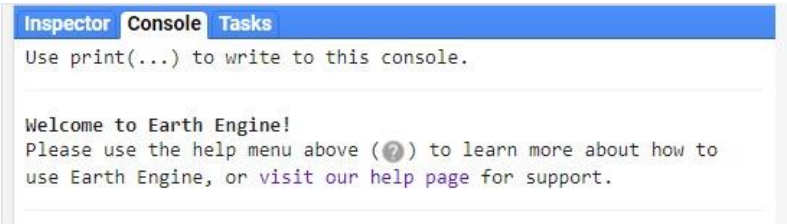


Figura 2.19: Vista de la consola en GEE.

Inspector

Permite consultar los valores de las capas que se muestran en el mapa, al dar clic sobre el punto de interés (Fig. 2.20).

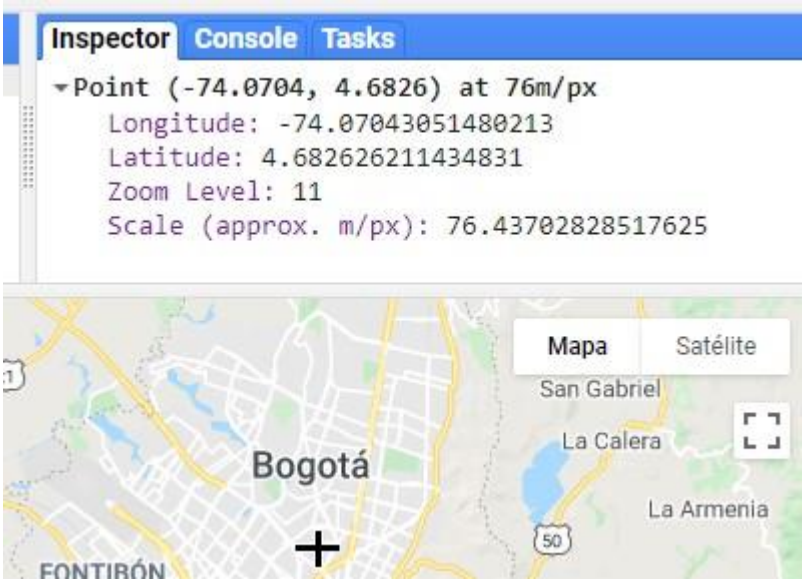


Figura 2.20: Vista de la pestaña de la herramienta de Inspector.

Tasks

En esta pestaña se muestran las tareas que se hayan exportado mediante la función `Export` y permite correr el trabajo de exportación al sitio donde el usuario le haya indicado (**drive**, **assets**; Fig. 2.21). Además, esta pestaña tiene otras funcionalidades:

- Mostrar las tareas para ejecutar (con el botón de **Run**, el cual permite especificar detalles de la exportación).
- Presentar las tareas en ejecución (color gris).
- Señalar el tiempo que tomó realizar en el trabajo.
- Indicar cuándo se ha finalizado algún trabajo (color azul).
- Cuando hay un error en algún objeto exportado, se verá el trabajo marcado en rojo. Por último, al dar clic sobre el signo de interrogación que aparece al colocar el cursor sobre algún trabajo, se puede obtener mayor información sobre este, como la ubicación del archivo exportado o más información sobre el error que arrojó el servidor al tratar de realizar una exportación.

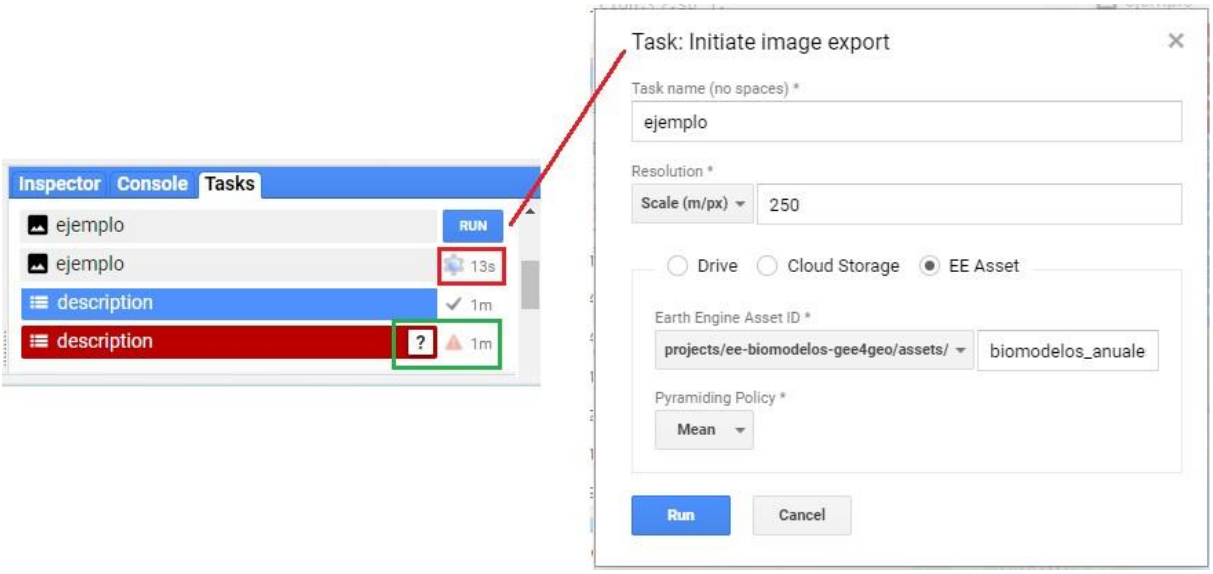


Figura 2.21: Vista de la pestaña de tareas dentro de GEE.

2.3 Programación en GEE

En la API de GEE, la programación sigue la sintaxis del lenguaje JavaScript y es orientada a objetos. Estos objetos se pueden interpretar como contenedores de información sobre los que se van a aplicar diferentes funciones o métodos para crear nuevos objetos o sobrescribir objetos preexistentes. Cada tipo de objeto tiene una serie de métodos propios que le permiten realizar diferentes tipos de operaciones, los cuales se revisan en el capítulo 5.

Simbología

- ‘ ’ : comillas simples.
- “ ” : comillas dobles.
- [ ] : corchetes.
- { } : llaves.
- ( ) : paréntesis.
- - : guion.
- _ : guion bajo.
- , : coma.

Google Earth Engine utiliza una versión anterior de JavaScript a ES6, de modo que para declarar variables no se utiliza la declaratoria ***const*** o ***let***, solo es válida la declaratoria ***var***.

Lenguaje JavaScript en GEE

El término que, para efectos de esta publicación, usaremos para llamar a los procesos aplicados con el operador `.` es el de método (*method*), lo cual quiere decir que son procedimientos ligados a un tipo de objeto. Por ejemplo, *en* ***`ee.Image.updateMask`***, el método ***`updateMask`*** es uno ligado a un objeto de tipo imagen (***`ee.Image`***). Por el contrario, las funciones en un sentido estricto se aplican a un objeto mediante la sintaxis: ***`function(objeto){}`***. Distintos tipos de objetos tienen diferentes métodos ligados a ellos.

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 (de igual forma iremos tocando la gramática del código a lo largo de los ejemplos):

- Es un lenguaje sensible a las mayúsculas y minúsculas, de tal manera que puede haber dos objetos diferentes, uno llamado ***a*** y otro llamado ***A***.
- Se recomienda cerrar cualquier comando con un ***;*** (de no hacerlo, el código funcionará igualmente; sin embargo, se recomienda utilizarlo para tener un código más claro y ordenado).
- Comúnmente en JavaScript se utiliza la notación *lowerCamelCase* para unir palabras (unir palabras sin espacio, pero cada una comenzando con su respectiva mayúscula), en lugar de guiones **-** o guion bajo _. Por ejemplo: `intervalMean` o `updateMask`.
- Todas las variables, funciones y objetos deben ser definidos mediante la función ***var***.
- El operador **=** asigna un valor a una variable.
- Los operadores matemáticos son: **+ - * /**.
- Para concatenar dos cadenas de caracteres se utiliza el símbolo **+**.
- Para realizar comentarios se puede utilizar `//` para comentarios de una línea, o también `/* ... */` para comentarios de varias líneas.
- Se suele utilizar el operador **.** (punto) para aplicar un método al objeto que lo precede. En este manual, todos los métodos aplicados a algún objeto se escribirán precedidos por el operador **.** para evitar errores de sintaxis.
- Son igualmente válidas tanto las comillas sencillas (' ') como las comillas dobles (" "), pero no deben mezclarse ambos tipos de comillas en una misma línea.

Interfaz Usuario

3

INTERFAZ DE USUARIO

Dentro de GEE hay un grupo de funciones que permiten interactuar con los objetos, y obtener información sobre sus características o propiedades, y visualizarlos o exportarlos para trabajar con ellos en un entorno distinto a GEE. A este conjunto de funciones le llamaremos funciones de interfaz de usuario. Estas funciones resultan muy útiles para ayudar al usuario a entender los procesos que está programando, así como para verificar que los procesos realicen la tarea deseada. Dentro de este grupo de funciones se distinguen, a continuación, cuatro grupos de funciones.

3.1 Impresión en consola

La función más sencilla de la interfaz es la de imprimir objetos en la consola, lo cual se realiza mediante la función `print` (imprimir). Al llamar esta función e indicar cualquier objeto como argumento, este se evalúa y se imprime su estructura en la consola. Esta función permite obtener información básica de los objetos como el tipo de objeto, número de registros, características y propiedades de los elementos que lo conforman. Además, esta función resulta muy útil para rastrear errores de programación (*bugs*) en los códigos. El resultado de la siguiente línea de código se puede apreciar en la Fig. 3.1.

```
print('Hola Mundo')
```

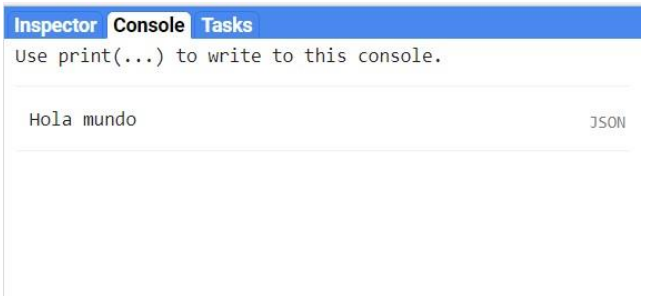


Figura 3.1: Ejemplo del uso de la función ***`print`*** *para imprimir en la consola.*

> Se recomienda utilizar varias impresiones en la consola (mediante ***`print`***), por programa, para verificar el resultado de procesos intermedios.

3.2 Interacción con la pantalla de mapa

Este grupo de funciones permiten agregar objetos del lado del servidor a la pantalla de mapa, así como controlar algunos parámetros de dicha pantalla. Todas las funciones encargadas del manejo de la pantalla de mapa se encuentran dentro de la biblioteca *`Map`*.

> Para revisar todas las funciones disponibles para la interfaz del usuario, ver la sección de Code Editor en la siguiente liga: https://developers. google.com/earth-engine/apidocs. Además, en esta sección se indican los argumentos que acepta cada función, así como el tipo de objeto de la salida.

Map.addLayer

`Map.addLayer` permite mostrar elementos en la pantalla de mapa de la API. Por lo tanto, permite mostrar objetos de tipo vector (`ee.Feature`) o imagen (`ee.Image`; estos objetos se presentan con mayor detalle más adelante). Los argumentos de la función `Map.addLayer` incluyen el objeto a añadir a la pantalla de mapa, seguido de un diccionario con información para el despliegue de la capa. Por último, se puede indicar el nombre de la capa con el que se desea mostrar la capa en la pantalla de mapa.

Al utilizar `Map.addLayer` sobre una imagen se pueden indicar los siguientes argumentos: ***bands*** (bandas), ***min*** y ***max*** (valores mínimos y máximos para visualizar la información). Estos se deben pasar dentro de un par de llaves {} como un diccionario (ver Capítulo 5 para una descripción de lo que es un diccionario). La función ofrece la opción de visualizar en modo monobanda o en compuesto RGB (del inglés Red, Green, Blue). En el caso de optar por el primero, permite asignar colores mediante el argumento de **palette** (paleta de colores); mientras que en el segundo se deben agregar las bandas en orden, es decir, primero la banda que corresponde al canal rojo, seguida del verde y del azul (Fig. 3.2).

```
Map.addLayer(image, {bands: ['B4', 'B3', 'B2'], min: 0,
                     max: 2000},
                     'RGB');
```

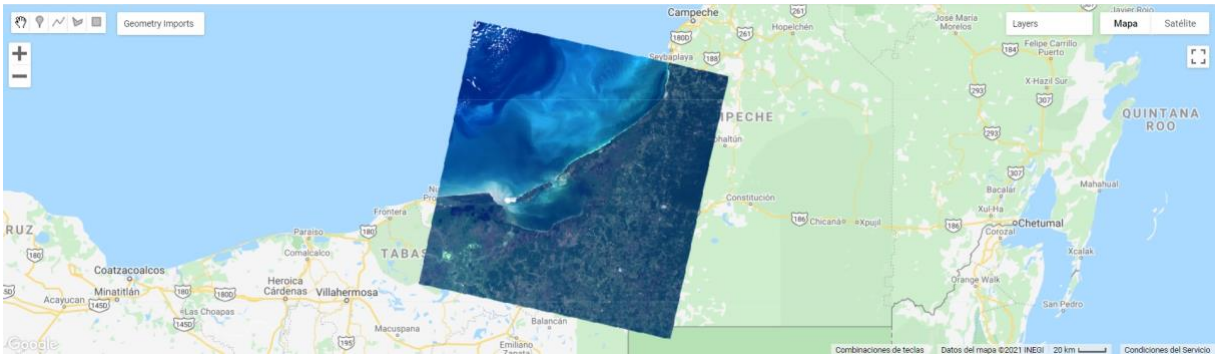


Figura 3.2: Ejemplo de visualización de un compuesto RGB de una imagen Landsat 8 sobre Laguna de Términos, México.

En algunas ocasiones, al intentar cargar objetos en la pantalla de mapa, que sean producto de procesamientos demandantes, la consola puede mostrar un error indicando que se superó el tiempo de espera o el límite de memoria disponible. En estos casos, se sugiere exportar el resultado en lugar de tratar de cargarlo en la pantalla de mapa y después visualizarlo de manera local en algún SIG o *software* similar (por ejemplo, *QGIS*).

La proyección por defecto para mostrar la información en la pantalla de mapa corresponde a *EPSG:3857*. Los códigos EPSG se pueden consultar en el siguiente enlace: https://epsg.io/. Dicha proyección únicamente se hace para poder visualizar la información, mas no transforma la información con la que el usuario trabaja.

Para el caso de los vectores (`ee.Feature`) se puede realizar una operación similar a las imágenes, aunque el único argumento válido es **`color`** (Fig. 3.3):

```
Map.addLayer(feature, {color: 'FF0000'}, 'featuresColored')
```

Figura 3.3: *Ejemplo de visualización de un vector con los límites políticos de los países de Sudamérica donde se indica su color.*

El código de los colores pasados al argumento de **`palette`** o **`color`** corresponden a códigos hexadecimales. Estos pueden ser consultados en el siguiente enlace: https://htmlcolorcodes.com/es/

Map.centerObject

Esta función permite centrar la pantalla de mapa en algún objeto o coordenada (longitud, latitud). Además, se puede indicar el zoom con el cual se quiere centrar el objeto.

```
Map.setCenter(39.86, 20.52, 5)
```

3.3 Creación de gráficos

Este grupo de funciones permiten realizar gráficos de diversos tipos; sin embargo, cada tipo de gráfico contiene argumentos válidos diferentes. Todas estas funciones se encuentran agrupadas dentro de la biblioteca `ui.Chart`. Los objetos básicos que permite graficar GEE son: vectores (`ee.Feature`), colecciones de vectores (`ee.FeatureCollection`), imágenes (`ee.Image`), colecciones de imágenes (`ee.ImageCollection`), arreglos (`ee.Array`) y listas (`ee.List`). Además, para indicar los colores a utilizar en el gráfico o el título del mismo se puede utilizar el método `.setOptions` sobre el objeto de tipo gráfico.

ui.Chart.feature

Este conjunto de funciones permite obtener distintos tipos de gráficos a partir de un vector o una colección de vectores y mostrar su resultado al imprimirlos en la consola (mediante la función `print`). Algunas de las funciones disponibles son:

```
ui.Chart.feature.byFeature.
ui.Chart.feature.byProperty.
ui.Chart.feature.groups.
ui.Chart.feature.histogram.
```

```
ui.Chart.feature.byProperty({
  // Indicar el vector o colección de atributos a partir del cual se
  // va a hacer el gráfico
  feature: coleccionAtributos,
  // La propiedad a colocar en el eje de las x
  xProperties: 'Clase',
})
```

ui.Chart.image

Este conjunto de funciones permite obtener distintos tipos de gráficos a partir de una imagen o colección de imágenes. Algunas de las funciones que se encuentran disponibles son:

```
ui.Chart.image.byClass
ui.Chart.image.byRegion
ui.Chart.image.doySeries
ui.Chart.image.histogram
ui.Chart.image.seriesByRegion
```

Por ejemplo, para graficar una serie de tiempo a partir de una colección de imágenes se utiliza la función `ui.Chart.image.seriesByRegion` (Fig. 3.4).

```
ui.Chart.image.seriesByRegion({
  // Definir la colección de imágenes a graficar
  imageCollection: coleccionImagenes,
  // Definir las regiones sobre las cuales se va a extraer la
  // información para el gráfico
  regions: areas.filter(ee.Filter.eq('Tipo','Bosque')),
  // El reductor a utilizar para resumir la información de la
  // colección de imágenes en las regiones indicadas
  reducer: ee.Reducer.mean(),
  // La banda sobre la cual se quiere construir el gráfico
  band: 'NDVI',
  // Tamaño en m del píxel para obtener la información
  scale: 30,
  // Propiedad para nombrar a cada serie
  seriesProperty: 'Tipo'
}).setOptions({
      title: 'Bosque',
      colors: ['#EE3A19']
  });
```

Figura 3.4: *Ejemplo de un gráfico que muestra los perfiles temporales de un índice de vegetación (NDVI) en tres tipos de áreas: deforestación, permanencia de bosque y permanencia de no bosque.*

> Los nombres 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.

3.4 Exportación de objetos fuera de GEE

Este grupo de funciones permite exportar algún resultado fuera de GEE para ser manejado en otro programa. Todas estas funciones se encuentran dentro de la biblioteca `Export`. Sin embargo, solo existen cuatro formatos válidos para exportar desde GEE: ráster (`ee.Image`), vector (`ee.FeatureCollection`), mapa y video.

Export.image

En el caso de las imágenes (ráster, `ee.Image`), la función para exportarlas es `Export.image.` Dentro de GEE hay tres opciones para exportar los resultados de una imagen:

```
Export.image.toAsset
Export.image.toDrive
Export.image.toCloudStorage
```

La primera permite exportar la imagen (ráster) a la sección de **Assets**, es decir, la sección donde el usuario puede subir su información a GEE. Esta opción es útil cuando el resultado se va a utilizar en otro procedimiento de GEE. La segunda opción permite exportarla al **Google Drive** de la cuenta con la que se tiene acceso a GEE. Esta opción es útil para trabajar con las imágenes (rásters) 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 Platform**. La opción que consideramos es más común para cualquier usuario será la de `Export.image.toDrive`. Al exportar información de GEE al **Google Drive**, se recomienda utilizar distintas carpetas para organizarla y definir su nombre en el argumento de folder.

> Solo se pueden exportar imágenes en formato **`ee.Image`**. Sin embargo, no siempre resulta obvio el formato de las imágenes, ya que algunas veces se pueden convertir a un formato de arreglo (**`ee.Array`**), tras lo cual hay que volver a convertir la información a formato de imagen (**`ee.Image`**) para poder exportarla. La aplicación permite exportar únicamente en formato GeoTiff o TFRecord (tensores), siendo la primera la opción por defecto.

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

```
Export.image.toDrive({
  // Definir la imagen a exportar
  image: imgDiff,
  // Especificar el nombre con el cual se va a guardar la imagen en
  // el Google Drive
  description: 'DiferenciaNDVI_2016-2017',
  // Determinar el tamaño del píxel en m de la imagen a exportar
  scale: 30,
  // Exportar la imagen en formato GeoTIFF
  fileFormat: 'GeoTIFF',
  // Definir la carpeta del Google Drive en la que se va a exportar
  // la imagen
  folder: 'DiferenciaNDVIL8'
});
```

> En GEE muchas operaciones utilizan el término de **`scale`**, sin embargo, este no se refiere a la escala de trabajo en un sentido tradicional (por ejemplo, 1:50000), sino que se refiere al tamaño de píxel expresado en metros.

En algunas ocasiones, al intentar exportar una imagen muy grande (mayor a 10 000 000 de píxeles), la consola puede mostrar un error indicando que el objeto a exportar tiene un número muy alto de píxeles. En este caso, se debe aumentar el número de píxeles 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 x 10^10 píxeles.

Export.table

Por su parte, el formato vector (`ee.Feature`) permite exportar información vector y tablas sin información geográfica (o sin geometría). De nuevo, para poder exportar en el formato vector, el objeto exportado debe estar en formato de colección de vectores (`ee.FeatureCollection`). Al igual que en el caso de las imágenes, dentro de GEE hay tres opciones para exportar objetos de tipo vector:

```
Export.table.toAsset
Export.table.toDrive
Export.table.toCloudStorage
```

Para el caso de las tablas sin información geográfica asociada, se debe definir el objeto como un objeto vector sin geometría (es decir, **null**) y en la tabla de atributos anexar la información deseada. Un ejemplo del uso de esta función se presenta a continuación:

```
Export.table.toDrive({
  // Definir la colección de vectores a exportar
  collection: feature,
  // Especificar el nombre con el que se va a guardar el
  // archivo en el Google Drive
   description: 'nombreArchivo',
  // Determinar la carpeta dentro de Google Drive donde
  // se va a guardar el archivo
  folder: 'mifolder',
  // Exportar la información en formato csv (valores
  // delimitados por comas)
  fileFormat: 'CSV'
});
```

GEE permite exportar información en formato vector, pero solo en formato CSV, GeoJSON, KML, KMZ, SHP o TFRecord. El primer formato es el predeterminado.

Export.video

Esta opción permite exportar una colección de imágenes como video, ya sea al almacenamiento de la nube de **Google** o al **Drive**.

```
Export.video.toCloudStorage
Export.video.toDrive
```

Esta función puede resultar atractiva para observar los cambios de un área de interés a través del tiempo. A continuación se muestra cómo exportar una colección de imágenes en video:

```
Export.video.toDrive({
  // Definir la colección de imágenes a exportar
  collection: coleccionImgs,
  // Especificar el nombre con el que se va a guardar el video
  // en el Google Drive
  description: 'videoTimelapse',
  // Definir la carpeta donde se va a exportar el video
  folder: 'mifolder',
  // Determinar el número de cuadros por segundo
  framesPerSecond:  24
});
```

Importación de información a GEE

4

IMPORTACIÓN DE INFORMACIÓN A GEE

En la sección de **Assets** se pueden importar varios tipos de archivos a GEE para utilizarlos en la API: imágenes (***GeoTIFF***), archivos vector (***shapefiles***), archivos separados por comas (***csv***), colecciones de imágenes exportadas por el mismo GEE y carpetas (*folders*; Fig. 4.1a, 4.1b, 4.1c). Para importar cualquiera de estos tipos de archivos simplemente se tiene que ingresar a la pestaña de **Assets** en el panel izquierdo y después dar clic en **New** (nuevo) e indicar el archivo que se quiere importar a GEE (Fig. 4.2).

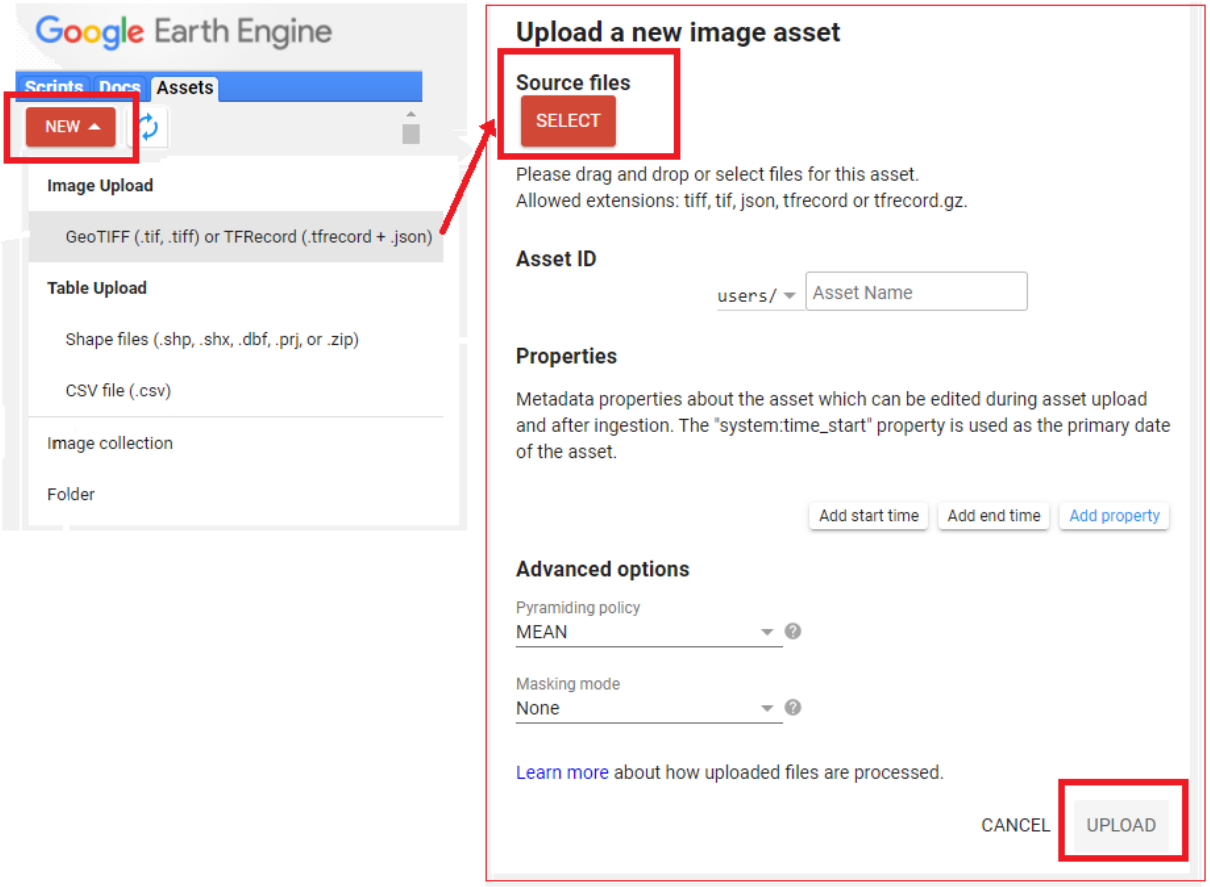


Figura 4.1a: *Opciones disponibles para importar datos tipo raster (GeoTIFF o TFRecord.*

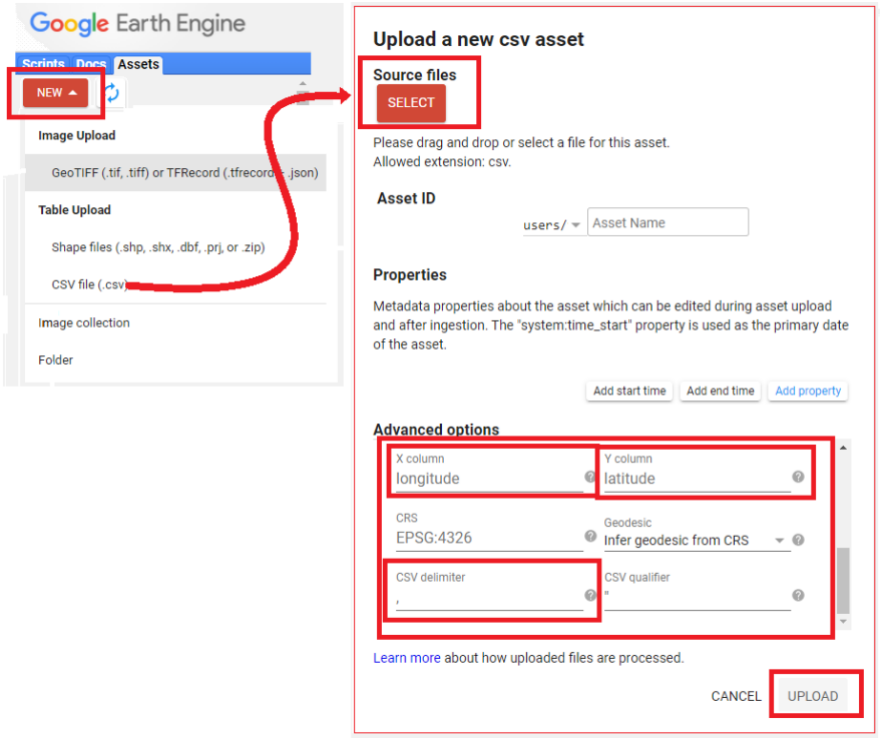


Figura 4.1b: Opciones disponibles para importar datos tipo CSV, especificar (si existe) columna de longitud y latitud (solo acepta punto decimal), y tipo de delimitador.

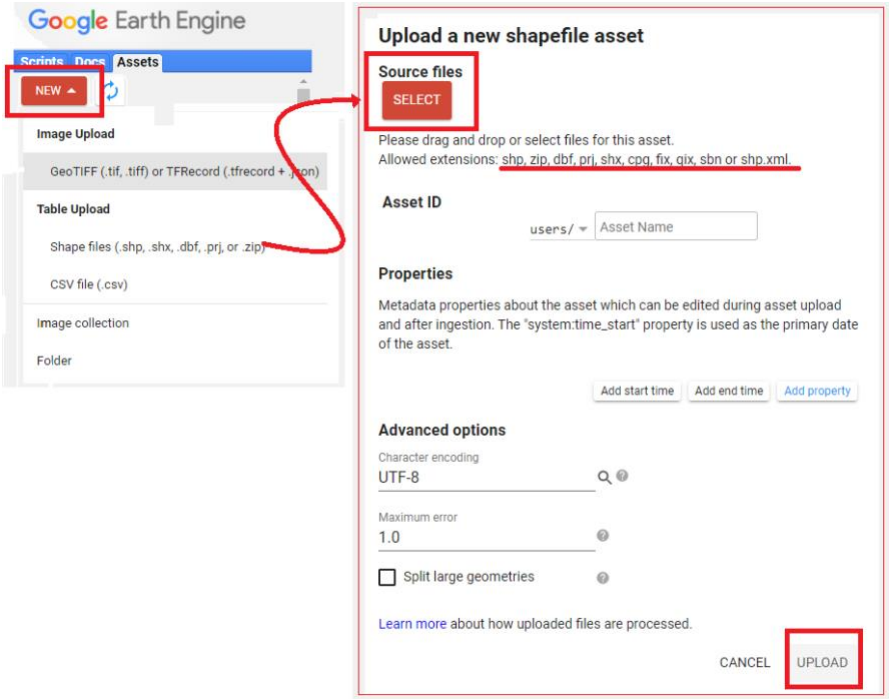


***Figura 4.1c:** Opciones disponibles para importar datos tipo Shapefile, incluir todos los archivos auxiliares.*

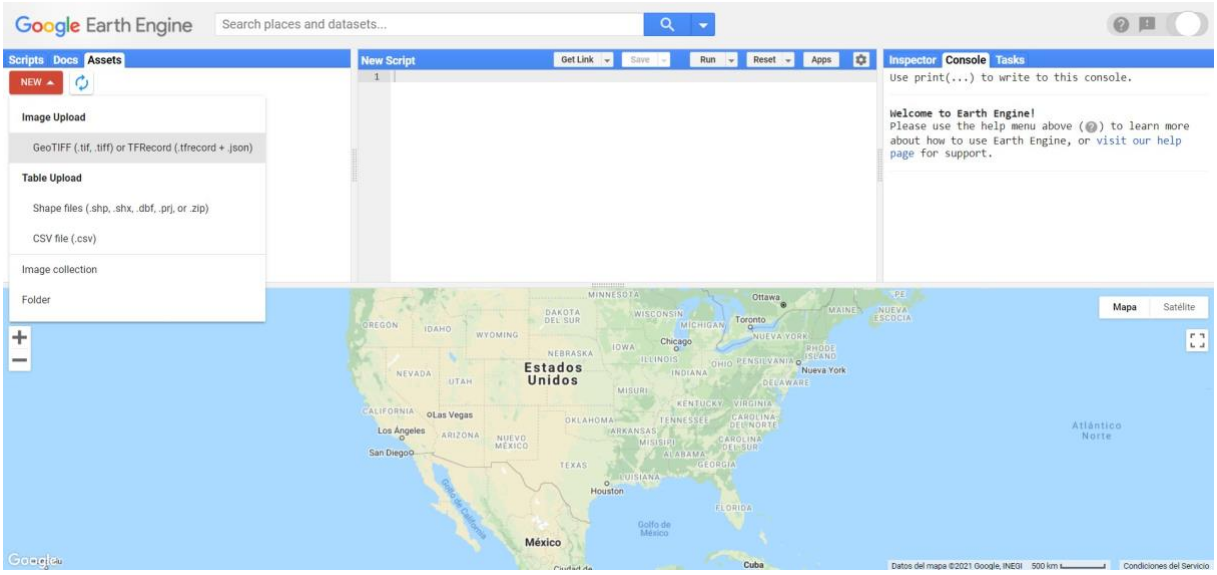


Figura 4.2: Ubicación de la pestaña de importación de datos a GEE, (botón rojo NEW, en la parte superior izquierda).

Posteriormente se da clic en **Select** y se seleccionan los archivos que se quieren subir (Fig. 4.3). Directamente en esta ventana se pueden agregar algunos campos a la información. Por último, se da clic 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**). Recuerde que para archivos shapefile solo se aceptan archivos con las extensiones **shp**, **zip**, **dbf**, **prj**, **shx**, **cpg**, **fix**, **qix**, **sbn** o **shp.xml**.

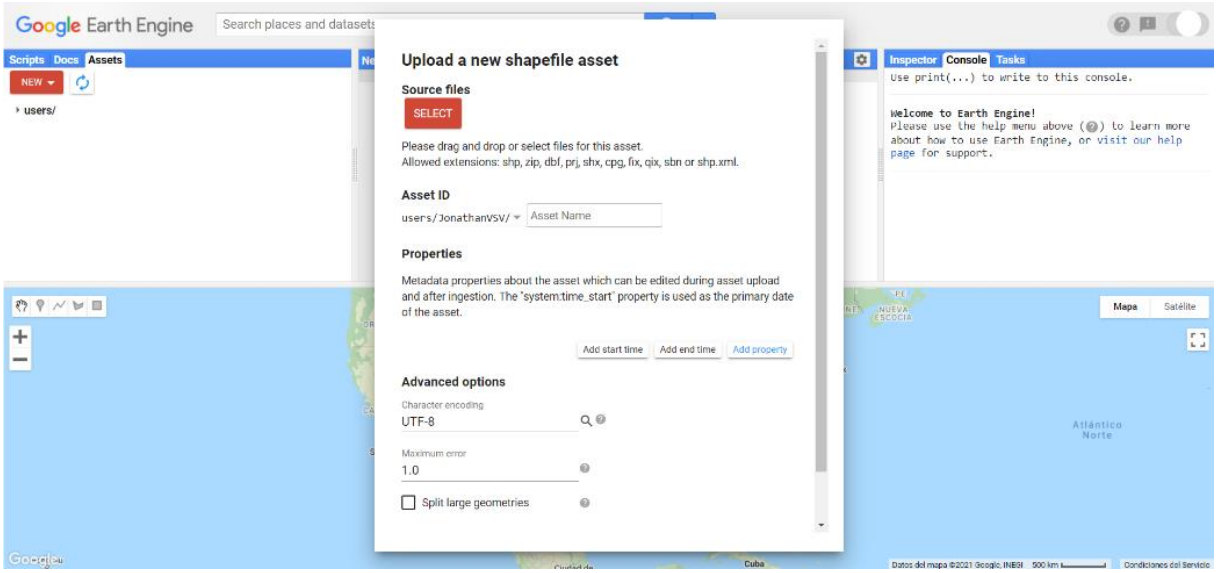


*Figura 4.3: ejemplo de menú para importar archivos, en este caso es el menu para importar un vector tipo **shapefile**, es necesario dar clic en el botón rojo SELECT, y escoger los archivos.*

Si se agrega algún otro archivo que tenga una extensión distinta a las aceptadas (shp, zip, dbf, prj, shx, cpg, fix, qix, sbn o shp.xml) la consola va a arrojar un error. En este caso hay que evitar seleccionar el archivo con la extensión que está provocando el error (por ejemplo, sbx).

Tras haber dado clic en **Upload**, el progreso en la subida del archivo a GEE se puede consultar en la pestaña de **Tasks** (Fig. 4.4). Una vez terminado, se puede tener acceso al archivo 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**, que contiene un par de flechas formando un círculo.

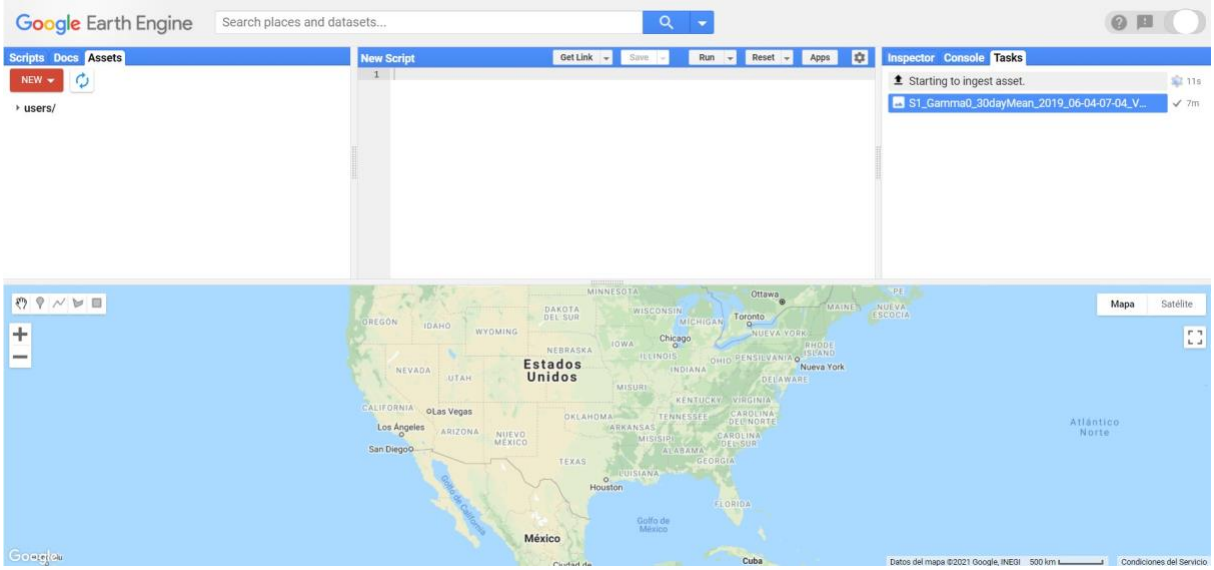


Figura 4.4: Ejemplo de la vista de una importación en proceso (Esquina superior derecha: barra gris importación en proceso, barra azul importación finalizada y exitosa).

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 carpetas.

Para subir un nuevo archivo hay que dar clic en el botón **New**, seleccionar el tipo de archivo y seguir las instrucciones:

- Para los archivos ráster, solo hace falta seleccionar el archivo correspondiente y dar clic en **Upload**.

- Para los archivos tipo **.shp** es necesario subir los archivos auxiliares (**dbf**, **prj**, **shx**, **cpg**, **fix**, **qix**, **sbn** o **shp.xml**) con el mismo nombre. Para los archivos **.zip** es necesario que contengan los archivos auxiliares.
- Para los archivos **.csv** es necesario que exista una columna de longitud y otra de latitud, las coordenadas deben ser decimales y estar en **EPSG:4326 (WGS 84).** El nombre de esas columnas se debe indicar en **Advanced options** / *X column - Y column* (longitud, latitud). Además, se debe tener absoluta claridad del tipo de delimitador (ejemplo: coma, punto y coma, u otro) y este debe ser especificado en **Advanced options** / *CSV delimited*.

El usuario puede compartir sus archivos con otros usuarios de GEE como lector o editor. Esta opción está disponible al darle clic en el símbolo de compartir a la derecha de cada archivo (aparece una vez que se coloca el puntero sobre el nombre del archivo; Fig. 4.5).

Figura 4.5: *Ubicación del botón para compartir información importada en GEE.*

Por último, hay dos maneras de utilizar cualquier información importada a GEE, pero primero hay que darle clic a la información importada en la sección de **Assets**, tras lo cual aparecerá la siguiente ventana (Fig. 4.6).

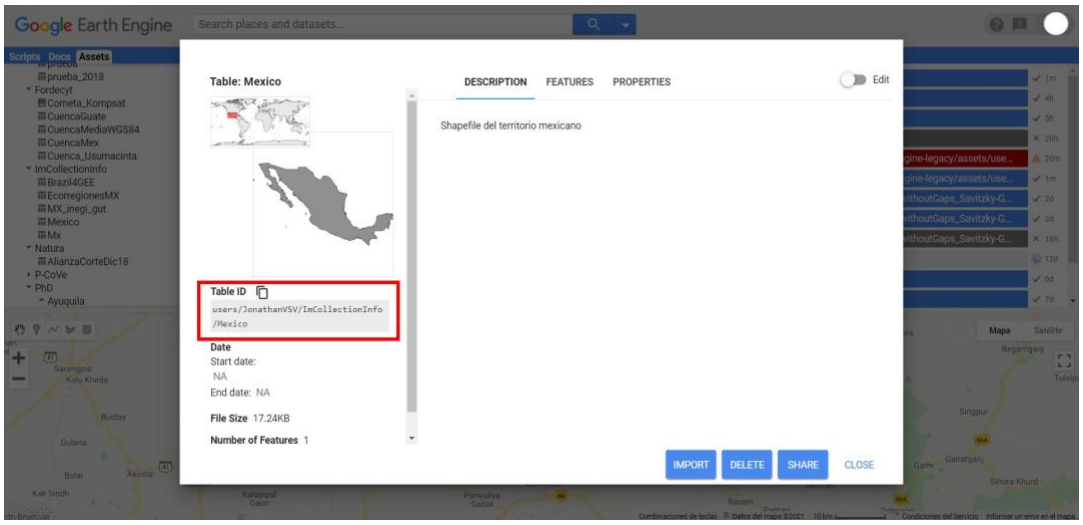


Figura 4.6: *Ejemplo de la ventana de información importada en GEE y ubicación de la ruta de la misma.*

La primera manera consiste en copiar la ruta de la información y utilizarla en cualquier código para definir un objeto (similar a lo que se hizo en ejercicios anteriores con información hospedada en la nube de GEE; Fig. 4.6).

La segunda consiste en acceder al código en el que estamos interesados en utilizar la información, buscar la información que se desea importar en la pestaña de **Assets**, darle clic, y al aparecer la misma ventana que en el ejemplo anterior, darle clic en el botón de **`Import`** (Fig. 4.7). De esa manera se importa la información en el código en el que se esté trabajando y se le asigna por defecto un nombre.

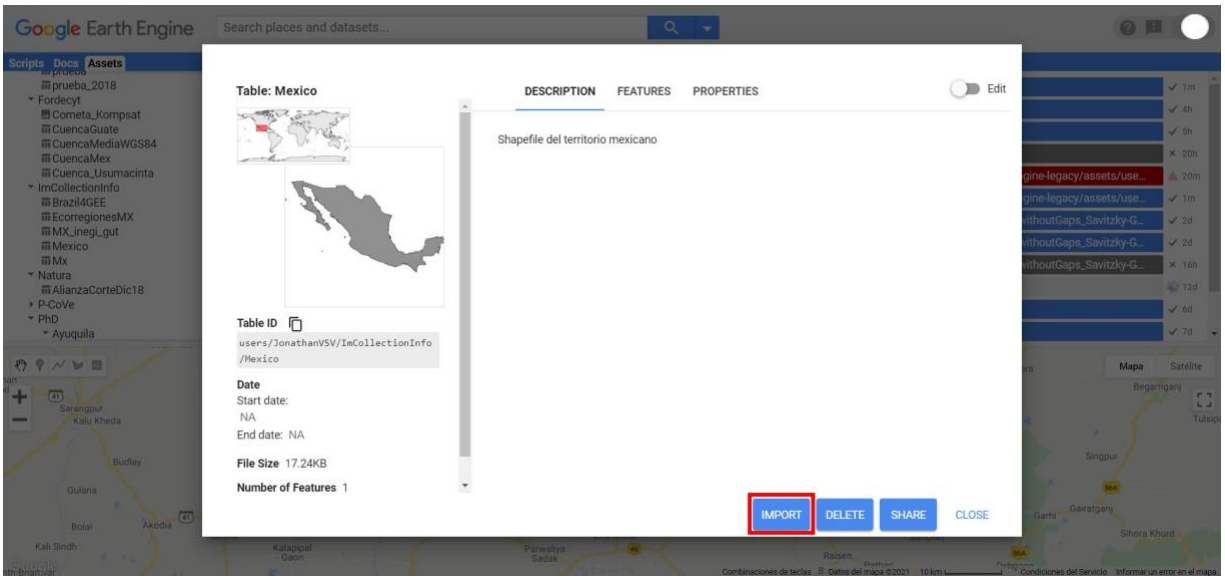


Figura 4.7: *Ejemplo de la ventana de información importada en GEE y ubicación del botón para importarla a un código previamente abierto.*

Tipos de Objetos

5

TIPOS DE OBJETOS

Dentro de GEE existe una amplia diversidad de tipos de objetos, por ejemplo: objetos de tipo numérico, listas, cadenas de caracteres, entre otros. Sin embargo, esta variedad de objetos se puede agrupar en dos grandes rubros: objetos del lado del cliente y objetos del servidor, según el lado donde se va a llevar a cabo el procedimiento deseado.

5.1 Objetos del cliente y del servidor

Existen dos lados de la programación de la API de GEE: el del servidor y el del cliente o usuario (Fig. 5.1). Así, un objeto puede ser convertido entre los dos tipos. 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')`. Adicionalmente, algunas operaciones se pueden hacer utilizando ambos tipos de sintaxis. Por ejemplo, una suma se puede realizar del lado del cliente mediante `1 + 2`, mientras que del lado del servidor se utilizará `ee.Number(1).add(ee.Number(2))`.

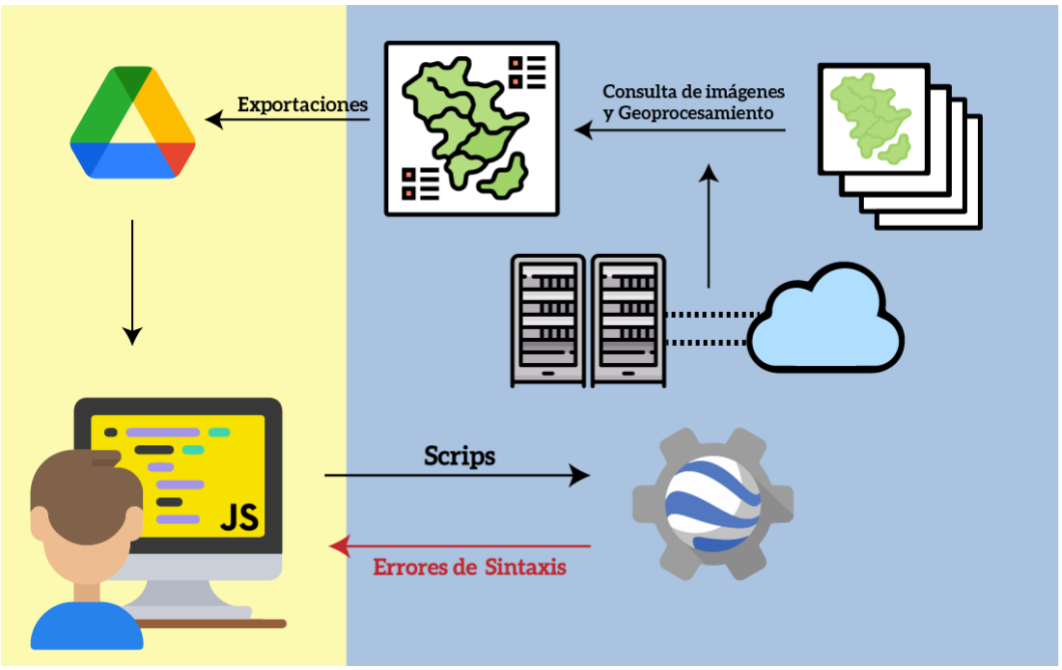


Figura 5.1: *Representación de los dos lados de programación en GEE: del cliente (amarillo) y el servidor (azul).*

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 las condiciones se sugiere utilizar en lugar de **if** y **else**, la función `ee.Algorithms.If`. Sin embargo, cabe aclarar que algunas funciones solo funcionan del lado del cliente. Por ejemplo, las funciones de la interfaz del usuario, utilizadas para exportar la información a algún archivo `ee.Export` (ya sea un ráster, un vector o una tabla), agregar una capa a la pantalla del mapa `Map.addLayer` o crear gráficos, así como imprimir información en la consola `print`. El siguiente diagrama permite visualizar el lado del usuario y del servidor en el funcionamiento de GEE.

5.2 Tipos de objetos del lado del cliente

Cadenas de texto

Se refiere a objetos de cadenas de símbolos de tipo alfanumérico. Estos se definen como cualquier cadena de caracteres (número, letras o símbolos) que se encuentren entre un par de comillas ya sean dobles “ ” o sencillas ’ ’. Por ejemplo (Fig. 5.2):

| EJERCICIO 0

```
// Cadena de texto de solo letras
var cadena = 'Esto es una cadena de caracteres';

// Cadena de texto de solo números
var telefono= '1234567890';

// Cadena de texto de letras, números y símbolos
var direccion = 'calle cuarta casa # 16';
```

| EJERCICIO 0.1

```
// Cadena de texto concatenada
var direccion = direccion + ' su número telefónico es:' + telefono;
```

Los números en una cadena de texto no serán interpretados como valores numéricos sino como texto.

```
Inspector Console Tasks
Use print(...) to write to this console.

Esto es una cadena de caracteres                                   JSON

1234567890                                                         JSON

calle cuarta casa # 16                                             JSON

calle cuarta casa # 16 su número telefónico es:1234567890          JSON
```

***Figura 5.2**: Salida de la consola de objetos de tipo cadena de caracteres.*

Números

Se refiere a objetos numéricos que indican un valor. Para números decimales se utiliza el punto decimal y no la coma decimal. Por ejemplo (Fig. 5.3):

| EJERCICIO 1

```
// Número entero
var numero = 1;

// Número decimal con punto decimal
var numero2 = 2.5;
```

```
Inspector Console Tasks
Use print(...) to write to this console.

1

2.5
```

***Figura 5.3:** Salida de la consola de objetos de tipo número.*

Para los objetos numéricos no se usan las comillas.

Listas

Las listas se refieren a objetos que contienen varias entradas, las cuales pueden ser numéricas (números), cadenas de texto (texto) o incluso otras listas. Las listas se definen mediante el uso de corchetes [ ] y cada entrada es separada mediante una coma , . Por ejemplo (Fig. 5.4):

| EJERCICIO 2

```
// Lista numérica
var lista = [1, 2, 3, 4, 5, 6, 7, 8];
// Lista de texto
var listaA = ['primero', 'segundo', 'tercero'];
// Lista de texto y números
var listaB = ['primero', 'segundo', 'tercero', 4];
// Lista de listas
var listaC = [lista, listaA, listaB];
```

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

| EJERCICIO 2.1

```
// Primer objeto de la lista llamada 'lista'
print(lista[0]);

// Segundo objeto de la lista llamada 'listaA'
print(listaA[1]);

// Tercer objeto de la lista llamada 'listaB'
print(listaB[2]);

// Lista completa con todos sus elementos
print(listaC);
```

```
Inspector  Console  Tasks
Use print(...) to write to this console.

▸[1,2,3,4,5,6,7,8]                                   JSON

▸["primero","segundo","tercero"]                     JSON

▸["primero","segundo","tercero",4]                   JSON

▾List (3 elements)                                   JSON
  ▸0: [1,2,3,4,5,6,7,8]
  ▸1: ["primero","segundo","tercero"]
  ▸2: ["primero","segundo","tercero",4]

 1

 segundo                                             JSON

 tercero                                             JSON

▾List (3 elements)                                   JSON
  ▸0: [1,2,3,4,5,6,7,8]
  ▸1: ["primero","segundo","tercero"]
  ▸2: ["primero","segundo","tercero",4]
```

Figura 5.4: *Salida de la consola de varios objetos de tipo lista.*

Diccionarios

Los diccionarios son objetos que contienen **claves** (**entradas**) y valores asociados a estas claves (**definiciones**). Los diccionarios se crean mediante el uso de llaves **{}** donde se indica cada clave seguida de dos puntos **:** y su definición, ya sea un valor, cadena de caracteres o lista asociada a esa clave. Para ingresar varias entradas, estas deben ir separadas por una coma.

Las claves o entradas siempre serán leídas como cadenas de texto. Se recomienda que no haya espacios dentro de ellas, en su lugar se aconseja usar guion bajo (_) o la notación de `LowerCamelCase`. Al hacer print se ordenarán alfabéticamente las claves (Fig. 5.5).

| EJERCICIO 3

```
// Se declara un nuevo diccionario.
// La primera entrada es el texto 'clave1' que está definido con
// el valor de 1 (tipo número) la segunda entrada es el texto
// 'clave2'y su definición es la letra 'A' (tipo cadena de texto)
var dicc = {
    clave1: 1,
    clave2: 'A'
};
```

Hay dos formas de consultar los valores dentro de un diccionario (Fig. 5.5):

Llamar directamente el nombre de la clave deseada dentro de comillas en un par de corchetes [' '].
Utilizar la notación de (punto) . seguida del nombre de la clave.

Por ejemplo:

| EJERCICIO 3.1

```
// Notación de lista, se mostrará en la consola la definición de
// 'clave1', es decir el valor 1
print(dicc['clave1']);

// Notación de lista, se mostrará en la consola la definición de
// 'clave2', es decir la letra 'A'
print(dicc['clave2']);

// Notación con punto, se mostrará en la consola la definición de
// 'clave1', es decir el valor 1
print(dicc.clave1);

// Notación con punto, se mostrará en la consola la definición de
// 'clave2', es decir la letra 'A'
print(dicc.clave2);
```

Figura 5.5: Salida de la consola de varios objetos de tipo diccionario.

> Para los diccionarios se recomienda utilizar la notación de (punto) . Seguida del nombre de la clave, ya que es la recomendada por GEE.

Funciones

Se refiere a objetos que contienen algún proceso que se realizará a alguna variable. Siempre comienzan con la palabra **`function`** (función) seguida por el objeto al que se le aplicará la función. Este objeto debe escribirse entre paréntesis y entre llaves se coloca el procedimiento que va a realizar la función. Por último, debe regresar un objeto mediante la función **`return`**.

Por ejemplo:

```
// MaskIm es una nueva función, su argumento de entrada es objeto
var maskIm = function(objeto){

    // Se define qué objeto es una imagen ráster y se guarda en la
    // variable image
    var image = ee.Image(objeto);

    // Se selecciona solamente la banda de nombre 'pixel_qa' de
    // la imagen image
    var qaImage = image.select('pixel_qa');

    // Se crea una nueva imagen binaria donde será 1 los
    // píxeles con valor igual a 322, y 0 los demás
    var clearData = qaImage.eq(322);

    // Se crea una máscara con la imagen binaria, y se entrega la
    // imagen original enmascarada
    return image.updateMask(clearData);
};
```

Aunque las funciones son propiamente objetos del lado del cliente, deben contener únicamente métodos del lado del servidor para que funcionen apropiadamente al trabajar en GEE. Esto permite que GEE interprete cualquier serie de procesos de manera adecuada en sus servidores. Esto quedará más claro después de repasar la siguiente sección.

> Las funciones que se ejecuten sobre una colección de imágenes o vectores solo funcionan si regresan un objeto de tipo `ee.Feature`, `ee.FeatureCollection`, `ee.Image` o `ee.ImageCollection`, por lo cual, a veces, se deben realizar ciertas conversiones para evitar un error.

5.3 Tipo de objetos del lado del servidor

Los objetos del lado del servidor, además de permitir llevar a cabo procesos en los servidores de GEE, tienen asociados una serie de métodos particulares por tipo de objeto. Esto quiere decir que al convertir un objeto del lado del cliente al del servidor, automáticamente se abre la posibilidad de utilizar los métodos precargados en GEE para ese tipo de objeto. Toda la información que se encuentra disponible en GEE corresponderá a objetos del servidor, entre los que destacan los objetos de tipo imagen (ráster), vector, colecciones de imágenes y colecciones de vectores.

Para saber cuáles objetos son del servidor, resulta útil recordar que todos ellos cuentan con el prefijo `ee.` (del servidor de Earth Engine) seguido del nombre del tipo de objeto con inicial mayúscula.

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. A continuación se describen los tipos de objetos que más se usan (Fig.5.6).

En algunos casos, los objetos que se obtienen a partir de ciertos métodos retornan un objeto de tipo indefinido (tipo objeto, `ee.Object`) como por ejemplo, al usar, en casos particulares, los métodos `.first` o `.get`, por lo cual se recomienda meter este objeto indefinido en un contenedor que indique el tipo de objeto del servidor. De no hacerlo, GEE mostrará un `error ... is not a function.`

Todos los métodos disponibles en GEE clasificados por tipo de objeto del servidor se pueden consultar en la sección de **Client libraries** en el siguiente enlace: https://developers.google.com/earth-engine/apidocs. En **Client libraries** se indican cuáles argumentos acepta cada método o función, así como el tipo de objeto de la salida.

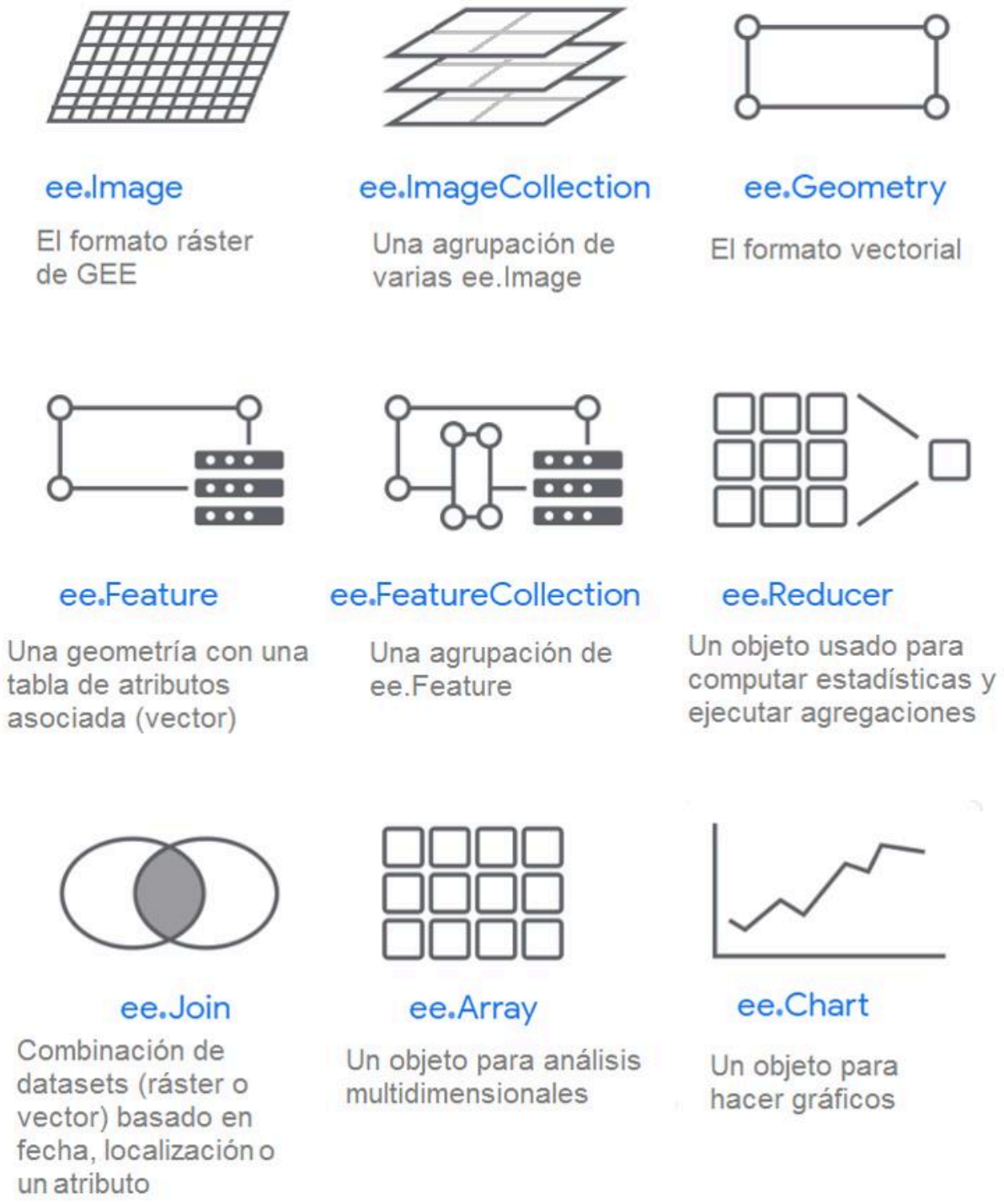


Figura 5.6: *Diagrama con algunos de los objetos del lado del servidor más utilizados en GEE. Tomado y modificado de https://developers.google.com/earth-engine/guides/objects_methods_overview.*

ee.String

Este tipo de objeto es equivalente a la cadena de texto pero del lado del servidor, ya que permite enviar un objeto como cadena de caracteres al servidor. Por ejemplo (Fig.5.7):

| EJERCICIO 4

```
// Esta es una cadena de texto del lado del cliente
var cadena2 = 'Esto es una cadena de caracteres';

// En este paso se convierte el texto del lado del cliente a
// un ee.String del lado del servidor
var cadenaServ = ee.String(cadena2);
```

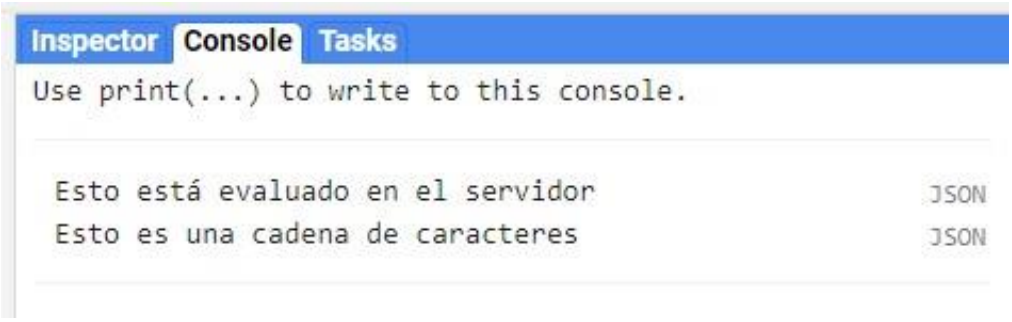


Figura 5.7: Salida de la consola de un objeto de tipo cadena de caracteres del lado del servidor.

ee.Number

Permite enviar un número como objeto al servidor. Por ejemplo (Fig. 5.8):

| EJERCICIO 5

```
// Se define un número como ee.Number del servidor
var numServ = ee.Number(1900);
```

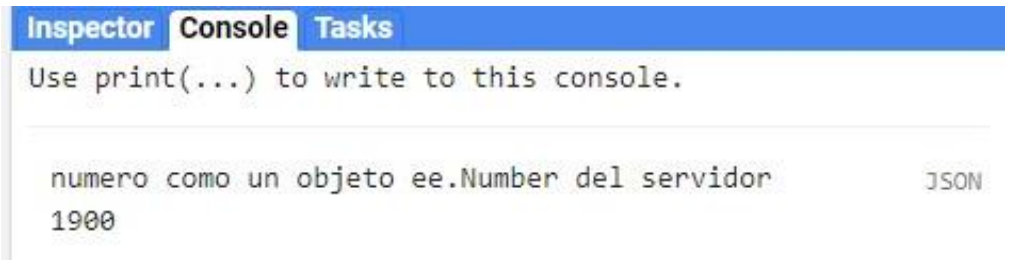


Figura 5.8: Salida de la consola de un objeto de tipo número del lado del servidor.

ee.List

Este tipo de objetos corresponden a listas del lado del servidor.
Por ejemplo (Fig. 5.9):

| EJERCICIO 6

```
// Se construye una nueva lista con diferentes tipos de objetos
// del lado del usuario y servidor
var lista = ['hola',    '12',    '5%',    ee.String('servidor'),
             ee.Number(64),      8.32];

// Se define una lista como ee.List del servidor
var listaServ = ee.List(lista);
```

Para acceder a un elemento de tipo `ee.List` se utiliza el método `.get` (Fig. 5.9).

| EJERCICIO 6.1

```
// Se llama solamente al primer objeto de la ee.List del servidor
print(listaServ.get(0));

// Se llama solamente al segundo objeto de la ee.List del
servidor
print(listaServ.get(1));
```

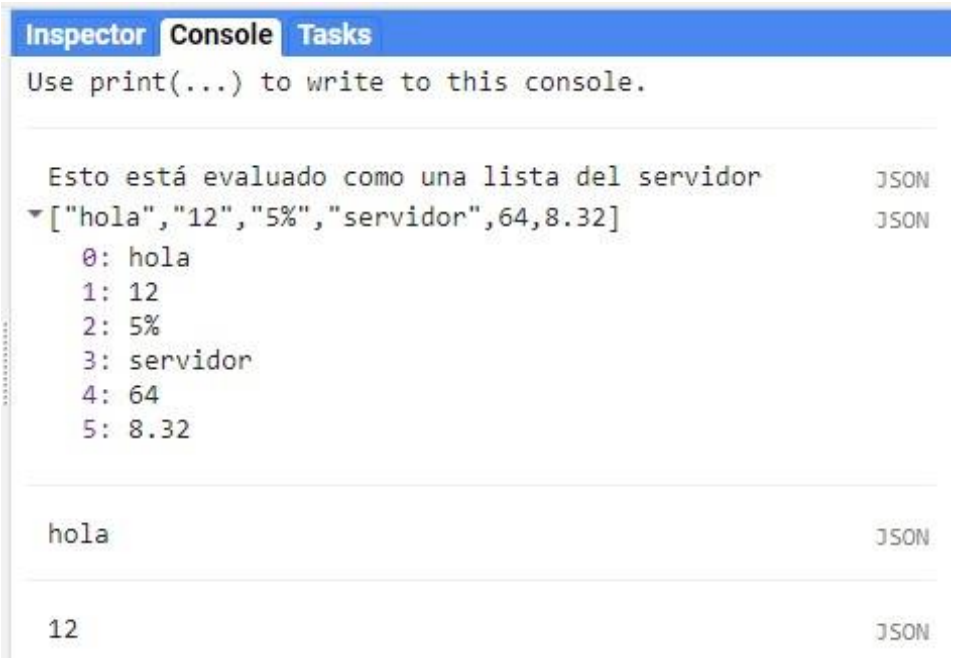


Figura 5.9: Salida de la consola de un objeto de tipo lista del lado del servidor.

ee.Dictionary

Permite enviar un objeto como diccionario al servidor. Por ejemplo (Fig. 5.10).

| EJERCICIO 7

```
// Se construye un diccionario de usuario con diferentes tipos de
// objetos del lado del usuario y servidor
var dicc = {
    'texto de usuario':'usuario',
    'string del servidor': ee.String('texto de servidor # 2'),
    'numero de usuario': 4,
    'numero de servidor': ee.Number(3.1416),
    'lista de usuario': [1, 'ejemplo'],
    'lista de servidor': ee.List(
         [ee.String('texto de servidor # 3'), ee.Number(1.44)])
};

// Se define el diccionario de usuario anterior como ee.Dictionary
// del servidor
var diccServ = ee.Dictionary(dicc);
```

Se puede consultar utilizando también el método .get y el nombre de la clave (Fig.5.10).

| EJERCICIO 7.1

```
// Se muestra el valor que contiene la clave 'lista_de_servidor'
print(diccServ.get('lista_de_servidor'));

// Se muestra el valor en el elemento ‘numero_de_usuario’
print(diccServ.get('numero_de_usuario'));
```

Por último, si se desea obtener las claves disponibles en un `ee.Dictionary` se utiliza el método `.keys`. Este método devuelve una lista. De igual forma, si se desea obtener los valores disponibles en un `ee.Dictionary` se utiliza el método `.values`, el cual devuelve una lista (Fig. 5.10).

| EJERCICIO 7.2

```
// Muestra una lista con todas las claves del ee.Dictionary
print('claves',diccServ.keys());

// Muestra una lista con todos los valores del ee.Dictionary
print('valores',diccServ.values());
```

```
Inspector  Console  Tasks
Use print(...) to write to this console.

 Esto está evaluado como una lista del servidor                    JSON
▾Object (6 properties)                                              JSON
  ▾lista_de_servidor: ["texto de servidor # 3",1.44]
     0: texto de servidor # 3
     1: 1.44
  ▾lista_de_usuario: [1,"ejemplo"]
     0: 1
     1: ejemplo
   numero_de_servidor: 3.1416
   numero_de_usuario: 4
   string_del_servidor: texto de servidor # 2
   texto_de_usuario: usuario

▾["texto de servidor # 3",1.44]                                     JSON
   0: texto de servidor # 3
   1: 1.44

 4

 claves                                                             JSON
▾List (6 elements)                                                  JSON
   0: lista_de_servidor
   1: lista_de_usuario
   2: numero_de_servidor
   3: numero_de_usuario
   4: string_del_servidor
   5: texto_de_usuario

 valores                                                            JSON
▾List (6 elements)                                                  JSON
  ▾0: ["texto de servidor # 3",1.44]
     0: texto de servidor # 3
     1: 1.44
  ▾1: [1,"ejemplo"]
     0: 1
     1: ejemplo
   2: 3.1416
   3: 4
   4: texto de servidor # 2
   5: usuario
```

Figura 5.10: *Salida de la consola de varios objetos de tipo diccionario del lado del servidor.*

ee.Date

Esta es la forma en la que GEE trabaja con fechas. Hay varias formas de construir una fecha específica (Fig. 5.11):

1. Se puede usar un texto de usuario indicando la fecha en el siguiente formato: **'AAAAMM-DD'**, esta fecha obligatoriamente debe:
    - Usar la función `ee.Date`.
    - Indicar la fecha en el orden de Año Mes Día (**AAAA-MM-DD**).
    - Usar guiones como separadores **-**.
    - Tener meses numéricos de dos dígitos (**01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12**).
    - Tener días de dos dígitos (**01, 02, 03 ... 09, 10, 11 ... 30**).
    - Estar entre comillas.

2. Usar números de usuario, en este caso se debe:
    - Usar el método `ee.DatefromYMD`.
    - Ingresar dentro del paréntesis el año, el mes y el día (en ese orden), separados por comas.

3. Usar un texto de usuario especificando el formato, para esto se debe:
    - Usar el método `ee.Date.parse`.
    - Especificar el formato de la fecha con la siguiente clave (AAAA = año, MM = mes, D = día).
    - Especificar el tipo de separador.
    - Especificar el orden en el que se ingresará el texto.
    - Especificar la cantidad de dígitos esperada para cada ítem (año, mes y día).
    - El formato debe estar entre comillas.
    - Luego de especificar el formato, se ingresa dentro de comillas el texto a convertir a `ee.Date` (entre el formato y el texto debe ir una coma).

Internamente los objetos ee.Date transforman el formato de fecha a formato UNIX (también llamado EPOCH o POSIX, que se define como la cantidad de segundos transcurridos desde la medianoche UTC del 1 de enero de 1970. Véase en la Fig. 5.11.

| EJERCICIO 8

```
// Fecha como un texto de cliente
var fechaString = '2001-10';

// Convierte la fecha de texto de cliente a ee.Date del servidor
var fecha = ee.Date(fechaString);

// Crea una ee.Date a partir de definir año, mes y día con números
// del cliente
var fecha2 = ee.Date.fromYMD(2015,03,28);

// Crea una fecha a partir de un formato específico y un texto que
// cumple con dicho formato, nótese que se especifica el orden del
// día año y mes, se especifica cada separador, y si se usan meses
// como texto deben ser las 3 primeras iniciales del mes en inglés
var fecha3 = ee.Date.parse('DD_YYYY/MMM', '14_1827/jul');
```

Figura 5.11: Salida de la consola de un objeto de tipo fecha del lado del servidor.

ee.Image

La definición de objetos ee.Image (imagen) habilita los métodos disponibles en GEE para el procesamiento de imágenes. Este va a ser el tipo de objetos para trabajar con cualquier elemento de tipo ráster en GEE. Más adelante se explica con mayor detalle.

ee.ImageCollection

Las colecciones de imágenes (`ee.ImageCollection`) están formadas por imágenes individuales (`ee.Image`). Más adelante se explica este tipo de objetos con mayor detalle.

ee.Feature

Este va a ser el tipo de objetos para trabajar con cualquier objeto de tipo vector o tabla. Más adelante se explica este tipo de objeto con mayor detalle.

ee.FeatureCollection

Las colecciones de vectores (`ee.FeatureCollection`) están formadas por varios vectores (`ee.Feature`). Más adelante se explican estos objetos 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, por ejemplo, `ee.Algorithms.If`, hasta algoritmos de segmentación temporal de una serie de imágenes, por ejemplo, `ee.Algorithms.TemporalSegmentation.Ccdc.`

ee.Array

Este tipo de objetos corresponden a arreglos multidimensionales, los cuales se pueden interpretar como matrices de más de dos dimensiones (por ejemplo, filas y columnas). Su uso más común se da en el análisis de series de tiempo o con ordenaciones espectrales. Este tipo de objetos cuentan con una serie de métodos, que se pueden consultar bajo la biblioteca de `ee.Array` (ver liga anterior).

ee.Classifier

Este tipo de objetos corresponden a algoritmos de clasificación supervisada de datos que se encuentran precargados en GEE. Por ejemplo, se encuentra el algoritmo de **random forest**, disponible mediante la función `ee.Classifier.smileRandomForest` o **MaxEnt**, disponible mediante la función `ee.Classifier.amnhMaxent`, entre otros.

ee.Clusterer

Este tipo de objetos corresponden a algoritmos de clasificación no supervisada de datos que se encuentran precargados 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 se utilizan para filtrar colecciones, ya sean de vectores o de imágenes. También permiten definir filtros de distintos tipos, ya sean espaciales, temporales o en función de características de las imágenes o vectores (por ejemplo, metadatos). Por último, contienen métodos para combinar filtros.

ee.Geometry

Este tipo de objetos corresponden a distintos tipos de **geometrías**, que incluyen **líneas**, **polígonos** y **puntos**. Además, en GEE se encuentran varios métodos precargados que se pueden aplicar a este tipo de objetos como, `ee.Geometry.MultiPolygon.Simplify` para simplificar polígonos múltiples o `ee.Geometry.Polygon.area` para calcular el área de un polígono, por ejemplo.

ee.Join

Este conjunto de métodos permite realizar uniones entre colecciones de vectores o imágenes, utilizando los campos de estos como las claves para realizar las uniones. Por ejemplo, se pueden unir dos colecciones de vectores mediante `ee.Join.merge` o unir los campos de una primera colección con los de una segunda mediante `ee.Join.inner`.

ee.Reducer

Los reductores permiten agregar datos basados en una regla o utilizando una operación matemática determinada. Este tipo de métodos son los utilizados para generar, por ejemplo, una sola imagen a partir de varias imágenes que comparten una misma extensión espacial. El tipo de reductor define el tipo de agregación que se desea aplicar, por ejemplo, se puede reducir con una sencilla estadística (mínimo, máximo, media, moda, mediana, etc.) o con reductores más complejos (histogramas, enlistar, regresión lineal). Las reducciones se pueden realizar sobre las bandas de las imágenes o los atributos de los vectores.

ee.Terrain

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

Los argumentos de cualquier método del servidor se pueden pasar de dos maneras:

1. siguiendo el orden por defecto de los argumentos del método y separando cada argumento por una coma.
2. como un diccionario (dentro de llaves `{ }`) indicando el nombre del argumento como la clave, seguido de dos puntos y el valor del argumento. En este último caso, también se separa cada argumento con una coma.

ee.Geometry

6

ee.Geometry

Las geometrías (`ee.Geometry`) son objetos que permiten leer y manejar formas geométricas, asociadas a un sistema de coordenadas geográficas. En GEE existen diversos tipos de geometrías que incluyen **punto**, **multipunto**, **línea**, **multilínea**, **perímetro**, **polígono** y **multipolígono**. Las geometrías en GEE corresponden, por defecto, a geometrías geodésicas.

6.1 Información y metadatos

Dependiendo del tipo de geometría, se pueden consultar algunas características de la información y los metadatos, como el área de un polígono mediante `.area`, el tipo de la geometría mediante `.type` o las coordenadas mediante `.coordinates`.

6.2 Creación de geometrías

GEE utiliza el objeto `ee.Geometry` para leer y manejar formas geométricas, estas incluyen geometrías sin área, como son (Fig. 6.1):

- Punto (`ee.Geometry.Point`): una coordenada en X & Y.
- Multipunto (`ee.Geometry.MultiPoint`): una lista de puntos.
- Línea (`ee.Geometry.LineString`): una línea.
- Multilínea (`ee.Geometry.MultiLineString`): una lista de líneas.
- Perímetro (`ee.Geometry.linearRing`): un perímetro (una línea cerrada).

| EJERCICIO 9

```
// Un punto definido con su coordenada
var punto = ee.Geometry.Point(-99.14, 19.47);


// Un multipunto definido a partir de una lista de coordenadas de
// puntos
var MultiPunto = ee.Geometry.MultiPoint([
    -74.072, 4.754,
    -75.545, 6.285,
    -76.533, 3.461,
    -74.819, 10.997,
    -75.501, 10.393
    ]);

// Una línea definida a partir de una lista de las coordenadas de
// sus vértices
var linea = ee.Geometry.LineString([
    -117.08, 32.04,
    -104.25, 31.74,
    -96.69, 25.58,
    -95.99, 19.74,
    -87.2, 21.39,
    -87.2, 16.91,
    -82.28, 14.88,
    -83.15, 10.07,
    -77.35, 8.86,
    -71.55, 12.66
    ]);

// Una multilínea definida a partir de una lista de líneas
// ee.LineString, que a su vez están definidas por una lista de
// las coordenadas de sus vértices
var MultiLinea = ee.Geometry.MultiLineString([
    ee.Geometry.LineString([-76.017, 26.173, -76.017, 16.173]),
    ee.Geometry.LineString([-73.017, 26.173, -73.017, 16.173]),
    ee.Geometry.LineString([-78.434, 20.926, -70.04, 24.401]),
    ee.Geometry.LineString([-78.434, 18.926, -70.04, 22.401])
    ]);

// Un perímetro definido a partir de una lista de las coordenadas
// de los vértices
var perimetro = ee.Geometry.LinearRing([
    -105.732, 20.627,
    -109.336, 26.544,
    -113.291, 31.274,
    -114.829, 31.274,
    -109.468, 23.317
    ]);
```

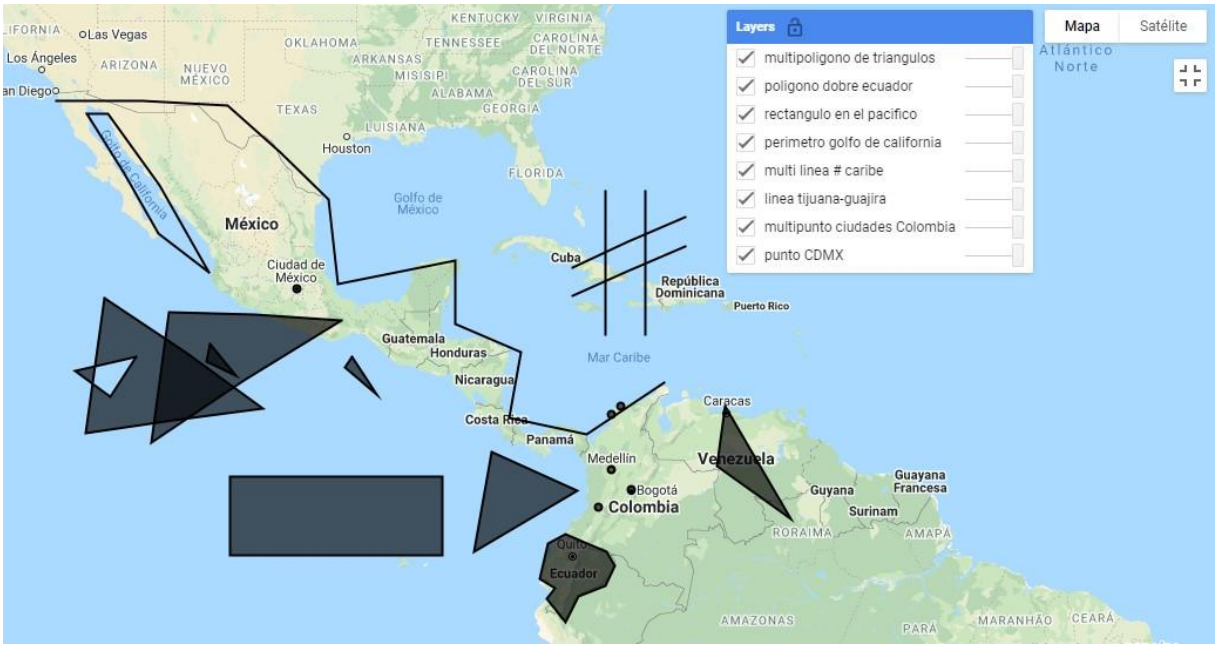


Figura 6.1: Visualización de los diferentes tipos de geometrías.

También hay geometrías que tienen el componente de área, como son (Fig. 6.1):

- Rectángulo (`ee.Geometry.Rectangle`): un rectángulo.
- Polígono (`ee.Geometry.Polygon`): un polígono.
- Multipolígono (`ee.Geometry.MultiPolygon`): una lista de polígonos.

| EJERCICIO 9.1

```
// Un rectángulo definido por las coordenadas de los vértices
// opuestos
var rectangulo = ee.Geometry.Rectangle(
    -104.12, 5.73,   -88.21, -0.15
    );

// Un polígono definido por las coordenadas de sus vértices
var poligono = ee.Geometry.Polygon(
    -80.348, -3.36,    -80.348, -3.36,
    -78.985, -5.113,   -78.019, -3.228,
    -75.997, -2.394,   -75.25, -0.901,
    -75.865, 0.11,     -78.941, 1.428,
    -80.26, 0.725,     -80.919, -1.911,
    -79.776, -2.614
    );
```

```
// Un multipolígono definido por una lista de polígonos
var Multipoligono = ee.Geometry.MultiPolygon([
    // El primer polígono está definido a partir de dos perímetros
    ee.Geometry.Polygon(
         // El primer perímetro define el polígono
         [ee.Geometry.LinearRing(-113.53, 18.81,
                                 -101.66, 10.83,
                                 -114.9, 8.93),
         // El segundo perímetro define los huecos dentro del
         // polígono
         ee.Geometry.LinearRing(-111.15, 14.52,
                                -113.09, 11.61,
                                -115.7, 13.58)
         ]),
    // El segundo polígono es una lista de coordenadas que
    // forman un triángulo
    [-95.77, 17.2,
    -108.7, 17.8,
    -110.0, 8.32],
    // El tercer polígono es una lista de coordenadas que forman
    // un triángulo
    [-105.6, 15.5,
    -105.9, 14.2,
    -103.7, 13.2],
    // El cuarto polígono es una lista de coordenadas que forman
    // un triángulo
    [-94.98, 14.6,
    -95.50, 13.8,
    -93.13, 11.8],
    // El quinto polígono es una lista de coordenadas que forman
    // un triángulo
    [-84.52, 7.62,
    -85.84, 0.17,
    -78.10, 4.74],
    // El sexto polígono es una lista de coordenadas que forman un
    // triángulo
    [-67.03, 11.0,
    -67.64, 6.49,
    -62.11, 2.63]
    ]);
```

Adicionalmente, se pueden crear geometrías directamente dibujando con el cursor en el mapa. Para ello se utilizan las herramientas de la esquina superior izquierda del panel del mapa (Fig. 6.2):

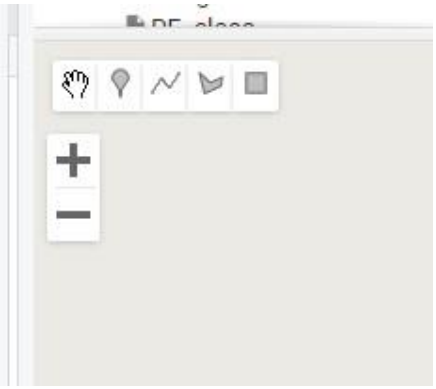


Figura 6.2: Herramienta de geometrías en la pantalla de mapa.

A continuación se describe cada función de las herramientas de geometrías de izquierda a derecha (Fig. 6.3). La primera, con el símbolo de una mano, permite moverse por el mapa. Las demás herramientas permiten crear geometrías de diferente tipo: la segunda, de tipo punto; la tercera, de tipo línea; la cuarta, de tipo polígono y la quinta, también de tipo polígono, pero se enfoca en crear rectángulos a partir de dos vértices.

Figura 6.3: Acercamiento a las herramientas de geometrías en la pantalla de mapa.

Todas las geometrías creadas con estas herramientas se importarán automáticamente al código y se podrán ver sobre las líneas del código (Fig. 6.4):

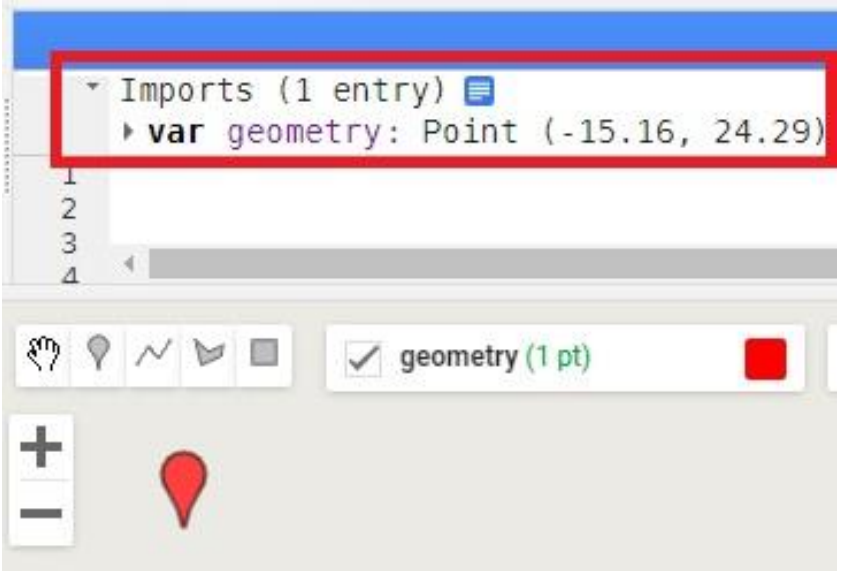


Figura 6.4: Visualización de la sección de Imports dentro de la pantalla de código.

A estas se les puede cambiar el nombre dando clic en el nombre de la geometría importada. Además, usando la mano se pueden mover y modificar las geometrías dibujadas (Fig. 6.3).

Todas las geometrías dibujadas se importarán como una sola colección, pero si se quiere tener geometrías separadas en diferentes colecciones se debe dar clic en **new layer**. Este menú aparecerá cuando se pase el cursor sobre la pestaña de geometrías dibujadas (Fig. 6.5).

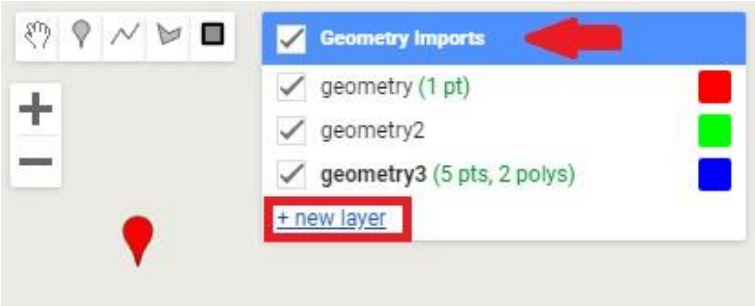


Figura 6.5: Herramienta de geometrías en la pantalla de mapas.

Pasando el cursor sobre cada una de las geometrías (en la pestaña de geometrías dibujadas) se podrá bloquear la capa para evitar modificaciones (con el candado) o cambiar la configuración (en el engranaje; Fig. 6.6).

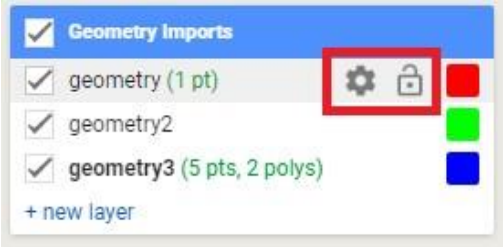


Figura 6.6: Ubicación de las herramientas de candado y engranaje.

Al dar clic en el engranaje se abrirá un menú donde se podrá cambiar el nombre de la geometría al escribir en la caja de texto nombrada **Name** (por defecto es '*geometry*'), elegir el tipo de geometría en el recuadro de **Import as** (geometría `ee.Geometry`, vector `ee.Feature` o colección de vectores `ee.FeatureCollection`), eliminar la geometría o cambiar el color de la geometría dibujada (al ingresar el código hexadecimal en la caja nombrada **Color** o al dar clic sobre un color del recuadro de colores) o añadir manualmente propiedades (al dar clic en el texto azul que dice **+ Property**; Fig. 6.7).

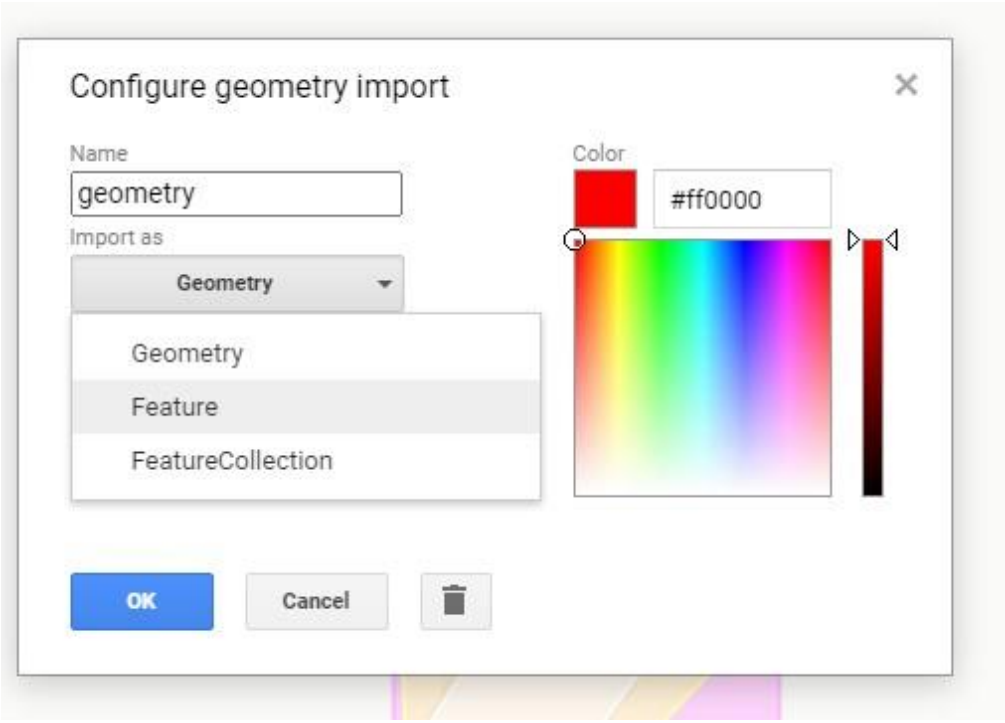


Figura 6.7: Ejemplo de modificación de las propiedades de las geometrías.

Recuerde que las geometrías no pueden contener propiedades o atributos, así que la opción de agregar propiedades únicamente estará disponible para objetos de tipo vector (`ee.Feature`) o colecciones de vectores (`ee.FeatureCollection`)

6.3 Métodos comunes

La mayoría de los métodos que se pueden utilizar sobre una geometría suelen ser los mismos que sobre los vectores, así que dichos métodos solo se describirán en la sección de vectores en el siguiente capítulo.

ee.Feature

7

ee.Feature

Los vectores (`ee.Feature`) en GEE son los objetos predeterminados para leer información vectorial o de tablas. Es importante recalcar que una `ee.Feature` es un vector individual, de modo que solo puede contener una sola geometría (**polígono**, **línea**, **punto**, **multipunto**, **multipolígono**).

7.1 Información y metadatos

Un vector estará compuesto por una geometría (**punto**, **línea**, **polígono**, **multipolígono**) y los atributos asociados a esa geometría (estos atributos son llamados propiedades y están almacenados en un diccionario; ver Olaya [2020] para una descripción más detallada de un vector). Para obtener información geométrica de los vectores se pueden utilizar los mismos métodos que un objeto `ee.Geometry`, mientras que para conocer características de las propiedades se puede utilizar el método `.propertyNames`.

7.2 Visualización de vectores

Las geometrías y vectores pueden ser visualizados en el mapa de GEE. El aspecto predeterminado de los vectores presenta líneas sólidas negras y un relleno semiopaco gris. Sin embargo, se puede especificar el color dentro de un diccionario en la función `Map.addLayer`, como código hexadecimal.

> Los cambios de los parámetros de visualización de los vectores no pueden hacerse en las herramientas de capa del mapa, solo pueden hacerse desde el editor de código. Para consultar el código hexadecimal del color deseado se puede acceder al siguiente enlace: https://htmlcolorcodes.com/es/

```
// Especificar el color de la capa
Map.addLayer(vector , {}, 'capa por default');
Map.addLayer(vector, {color: '2AF116'}, 'capa color verde');
```

7.3 Creación de vectores

Para crear un vector desde cero es necesario definir una geometría y opcionalmente un diccionario de propiedades (atributos) asociados a esa geometría. Por ejemplo (Fig. 7.1):

| EJERCICIO 10

```
// Crear un polígono
var poligono = ee.Geometry.Polygon([
    -119.24, 32.73, -115.02, -3.29,
    -66.5, -3.29,
    -68.43, 32.14 ]
    );

// Crear un vector a partir del polígono y un diccionario de
// atributos.
var vector = ee.Feature(
  // Polígono del vector
  poligono,
  // Diccionario de atributos con 3 atributos
  {anno: 1500,
  tamaño: '865 kilometros',
  nombres:['hugo','paco','luis']
  });
```

7.4 Métodos comunes

Selección de propiedades

Una vez revisados los nombres de las propiedades de un vector, se pueden seleccionar ciertos atributos mediante el método `.select`. Para seleccionar una única propiedad solo se requiere indicar el nombre de esta (entre comillas dentro de una lista), mientras que si se desea seleccionar varias propiedades, estas deben indicarse dentro de una lista separada por comas. Adicionalmente, se pueden renombrar las propiedades del vector. Para ello se inserta primero una lista que contenga las propiedades a las que se les desea cambiar el nombre, seguida de una segunda lista con los nuevos nombres. Por ejemplo (Fig. 7.1):

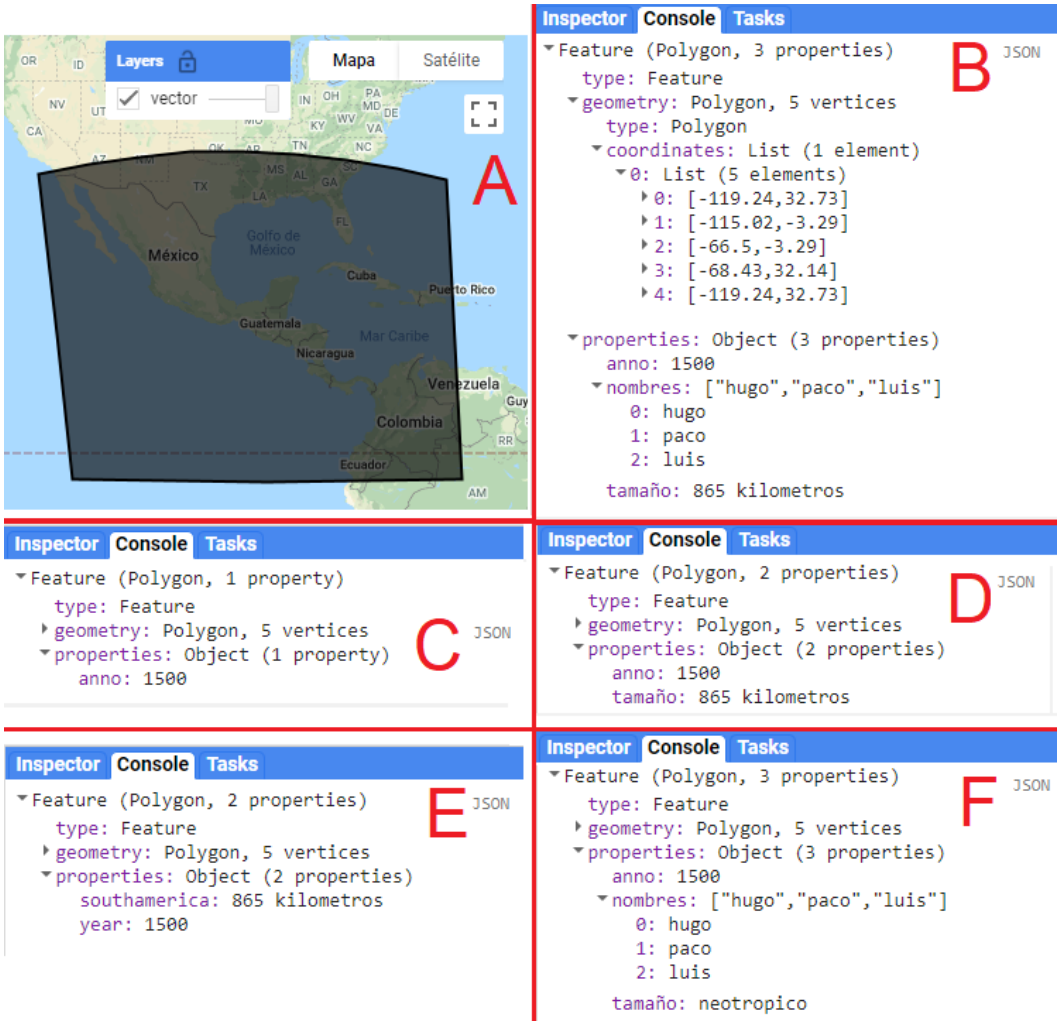


Figura 7.1: Visualización del vector (A). La salidas de la consola mostrando el vector creado con coordenadas y propiedades (B). La selección (ejercicio10.1) de una propiedad (C); de dos propiedades (D); renombrar esas propiedades (E) y reescribir propiedades (ejercicio 10.2) (F)..

| EJERCICIO 10.1

```
print(vector.select(['anno']))
print(vector.select(['anno','tamaño']))
print(vector.select(['anno','tamaño'],['year','southamerica']))
```

Adición de nuevas propiedades o modificación de propiedades preexistentes

Usando el método `.set` se pueden modificar las propiedades preexistentes o escribir nuevas propiedades sobre el vector. En este caso, primero se indica el nombre de la clave, seguido del valor correspondiente a la clave indicada. Por ejemplo (Fig. 7.1):

| EJERCICIO 10.2

```
var vector1 = vector.set('tamaño','neotropico');
```

Extracción y edición de propiedades de vectores

En este ejercicio primero se selecciona un vector para ejemplificar el uso de algunos métodos de los objetos `ee.Feature`. Para ello, se extrae el primer vector de una colección de vectores utilizando el método `.first`. Este procedimiento se verá con mayor detalle en el siguiente capítulo, en los métodos de las colecciones de vectores. A continuación, se consultarán algunas propiedades y atributos del vector seleccionado (Fig. 7.2).

1. Usamos el método `.propertyNames` para obtener una lista de los nombres de los atributos del vector.
2. Usamos el método `.toDictionary` para generar un diccionario con todos los atributos (claves) y sus valores del vector.
3. Usamos el método `.get` para obtener el valor del atributo `'COAST'` del vector.
4. Usando el método `.select` podemos seleccionar solamente una lista de atributos definidos, y si colocamos una segunda lista entonces los atributos de la primera lista serán renombrados con los nombres de la segunda.

| EJERCICIO 11

```
// Llamamos una capa de cuencas de alta resolución de GEE
// y se selecciona el primer vector de la colección
var cuenca = ee.Feature(
    ee.FeatureCollection('WWF/HydroSHEDS/v1/Basins/hybas_12')
    .first());

// Extraemos los nombres de los atributos del vector como una
// lista
var propiedades = cuenca.propertyNames();

// Extraemos los atributos del vector como un diccionario
var atributos = cuenca.toDictionary();

// Extraemos el atributo 'COAST'
var costa = cuenca.get('COAST');

// Extraemos el atributo 'COAST' y lo renombramos como
// 'costa maritima'
var renombre = cuenca.select(['COAST'],['costa maritima']);
```

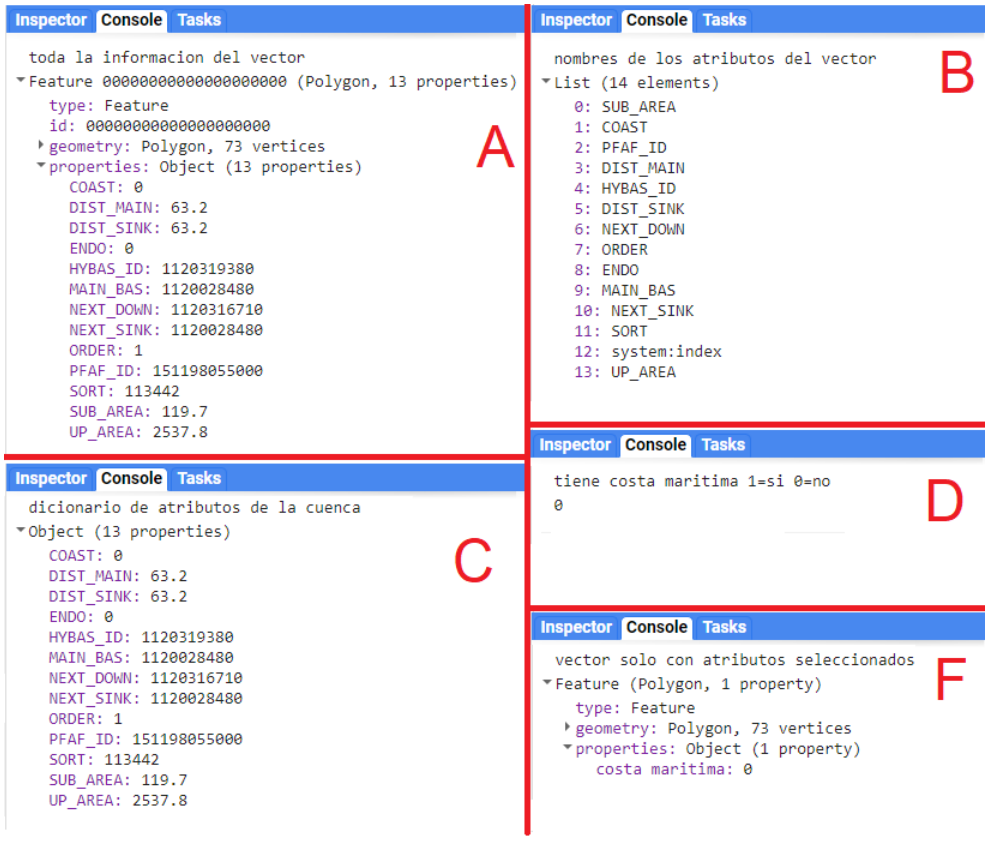


Figura 7.2: Salida de la consola de las propiedades (ejercicio 11) del vector "cuenca" (A), nombre de las propiedades de "cuenca" (B), propiedades como diccionario (C), valor del atributo 'COAST' (D), renombrar el atributo (F)

Intersección con otros vectores

Se pueden realizar cortes de un vector para quedarse con el área específica que intersecta otro vector utilizando el método `.intersection`. Por ejemplo (Fig. 7.3):

| EJERCICIO 12

```
// Crear dos vectores rectangulares
var rectangulo1 = ee.Feature(ee.Geometry.Rectangle(-92.0, 10.6,
    -82.0, 20.7));
var rectangulo2 = ee.Feature(ee.Geometry.Rectangle(-86.03, 15.6,
    -100.0, 40.7));

// Intersección
var interseccion = rectangulo1.intersection(rectangulo2);
```

Al realizar una intersección, el vector resultante heredará los atributos del vector al que se le aplicó dicho método (es decir, el objeto que queda del lado izquierdo del método .`intersection`).

Creación de un buffer

Para realizar un buffer a partir de un vector se utiliza el método .`buffer`, en el cual hay que indicar el valor en metros del tamaño del buffer y el error máximo aceptado (Fig. 7.3).

| EJERCICIO 12.1

```
// Crear un vector de un punto
var punto = ee.Feature(ee.Geometry.Point(-105.24, -2.73));

// Calcular un buffer de 500 km alrededor del punto
var bufferPol = punto.buffer(500000, 0.1);
```

Cálculo de área

Para calcular el área de un vector se utiliza el método .`area`. En este caso hay que definir el valor máximo de error aceptado. El área siempre se calcula en metros cuadrados (Fig. 7.3).

| EJERCICIO 12.2

```
// Calcular el área del buffer
var area = bufferPol.area(0.1);
```

Cálculo de perímetro

Podemos calcular el perímetro de un vector usando el método .`perimeter`, que hace el cálculo en metros (Fig. 7.3).

| EJERCICIO 12.3

```
// Calcular el perímetro del buffer
var perimetro = interseccion.perimeter();
```

Disolución de polígonos

El método .dissolve une todas las geometrías que se intersectan de un vector en un solo polígono y si no se intersectan, genera un multipolígono (Fig. 7.3).

| EJERCICIO 12.4

```
// Crear un vector de multipolígono
var multi = ee.Feature(ee.Geometry.MultiPolygon(
    [[[[ -137, 20],
       [ -128, 9 ],
       [ -116, 15],
       [ -124, 25]]],
     [[[ -123, 28],
       [ -129, 30],
       [ -139, 5 ]]],
     [[[ -113, 19],
       [ -143, 18],
       [ -123, 4 ]]],
     [[[ -133, 1 ],
       [ -118, 2 ],
       [ -116, 24]]],
     [[[ -109, 5 ],
       [ -103, 10],
       [ -147, 9 ]]],
     [[[ -110, 14],
       [ -160, 12],
       [ -161, 7 ]]],
     [[[ -117, -1],
       [ -117, -2],
       [ -136, -9]]],
     [[[ -121, -1],
       [ -123, -1],
       [ -126, -18]]]
    ]));

// Disolver sus geometrías
var disuelto = ee.Feature(multi.dissolve(0.1));
```

Unión de vectores

El método `.union` permite fusionar las geometrías de dos vectores en una sola geometría de un vector. Las geometrías que están intersectadas serán disueltas y las que no lo estén se convertirán en un multipolígono (Fig. 7.3).

| EJERCICIO 12.5

```
// Unir geometrías del buffer y un rectángulo
var union = rectangulo2.union(bufferPol)
```

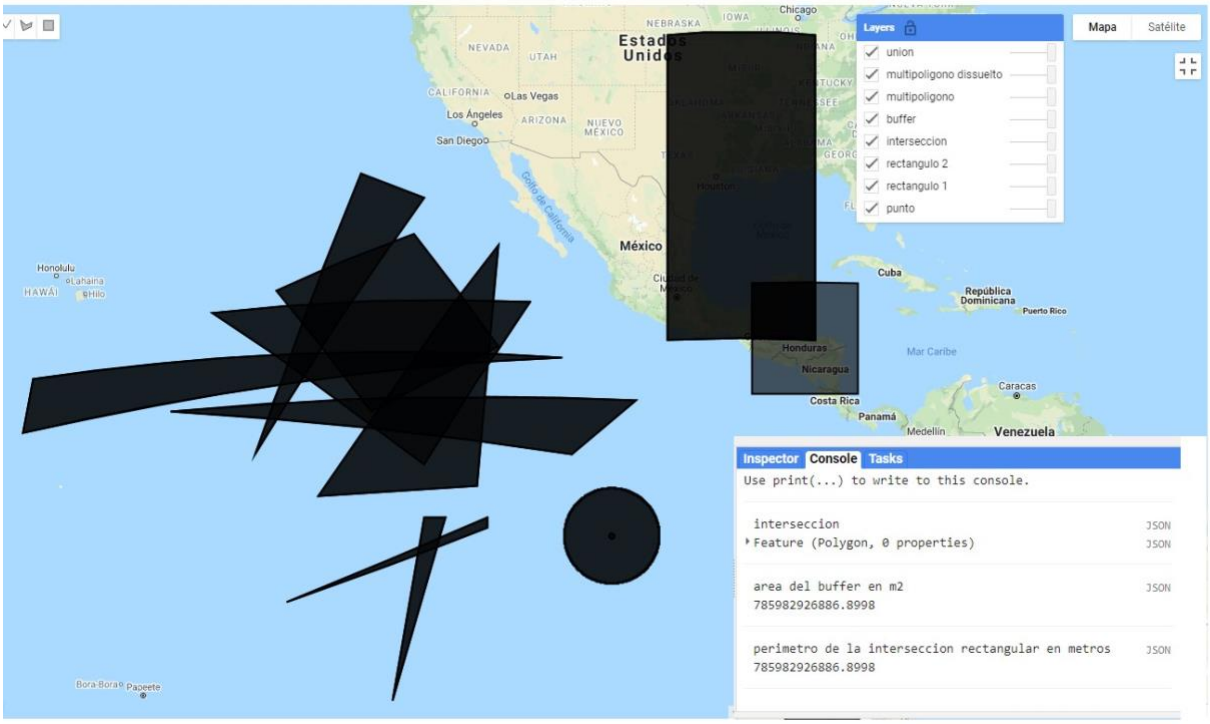


Figura 7.3: Visualización de la unión de diferentes geometrías, así como la salida de la consola de la intersección, área y perímetro calculados.

ee.FeatureCollection

8

ee.FeatureCollection

Las colecciones de vectores (`ee.FeatureCollection`) son objetos de GEE que contienen un conjunto de vectores, de modo que pueden contener objetos `ee.Feature` con geometrías diferentes (**polígono**, **línea**, **punto**, **multipunto**, **multipolígono**). La mayoría de los acervos de información vectorial disponibles en GEE van a ser definidos como colecciones de vectores. Para el manejo de varios vectores se recomienda utilizar esta estructura, en lugar de listas u otro tipo de objetos, ya que esto facilita ejecutar operaciones sobre todo el conjunto de vectores. Por ejemplo (Fig. 8.1):

EJERCICIO 13

```
// En el buscador se busca una tabla de países 'countries' y se
// escribe el ID de la tabla, en este caso es una colección de
// vectores del Departamento de Estado de Estados Unidos, que
// contiene las divisiones políticas de los países del mundo
var paises = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017');
```

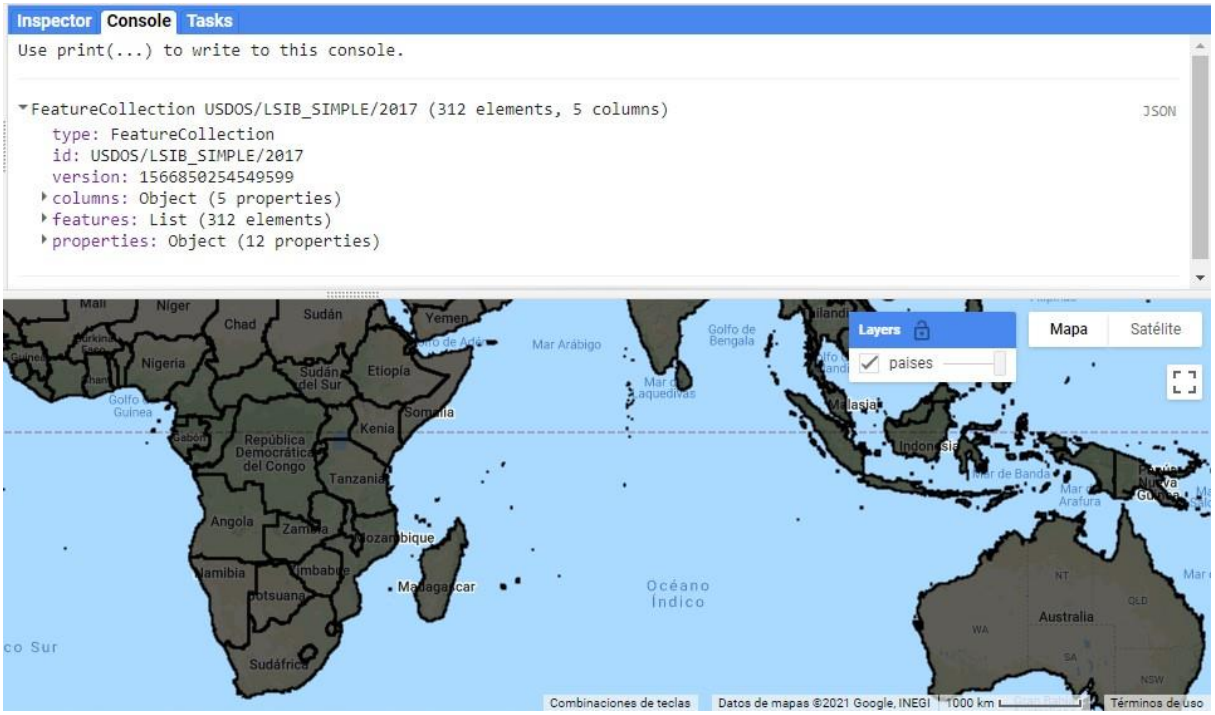


Figura 8.1: Visualización de la colección de vectores con los límites internacionales, así como su salida en la consola.

GEE ofrece un catálogo de colecciones de vectores que se puede buscar en la barra de búsqueda. Estas colecciones de vectores se llaman tablas (*tables*).

La búsqueda de información vectorial en los acervos de GEE es más difícil que la búsqueda de imágenes. Eso ocurre porque a pesar de que hay mucha información vectorial disponible (compartida por usuarios), solo está indexada la información compartida por Google.

8.1 Información y metadatos

En las colecciones de vectores se ubican los metadatos e información de todos los vectores que contienen. Gracias a ello se puede utilizar esta información para filtrar y utilizar únicamente los vectores que cumplen con ciertos criterios. Algunos de los métodos que permiten examinar una colección de vectores incluyen `.size` y `.limit` (Fig. 8.2).

| EJERCICIO 14

```
// Llamamos una capa de cuencas de alta resolución de GEE
var cuencas =
    ee.FeatureCollection('WWF/HydroSHEDS/v1/Basins/hybas_12');

// Cantidad de vectores en la colección (1034083)
var cantidad =cuencas.size();

// Seleccionamos solo los primeros 50 vectores
var primeras50cuencas= cuencas.limit(50);

// Este comando resultará en error porque excede los 5000 vectores
print(cuencas);

// Esta capa contiene todos los vectores 1034083. Nótese que en el
// mapa sí se pueden proyectar más de 5000 vectores, aunque esta
// capa es un poco pesada
Map.addLayer(cuencas,{},'todas las cuencas');

// Esta capa solo contiene las primeras 1000 cuencas y carga mucho
// más rápido (Noroccidente de África)
Map.addLayer(cuencas.limit(1000),{color:'00ff00'},
    '1000 cuencas filtradas');
```

Al usar `print` con las colecciones de vectores, la consola puede mostrar un máximo de 5000 vectores. En caso que uno necesite usar `print` en una colección de más de 5000 vectores, será necesario filtrar la colección previamente. Otra opción es limitar la cantidad de vectores que se vayan a mostrar. Para ello se puede usar el método `.limit`, el cual permite especificar la cantidad de vectores que queremos ver (siempre que sean <= 5000; Fig. 8.2).

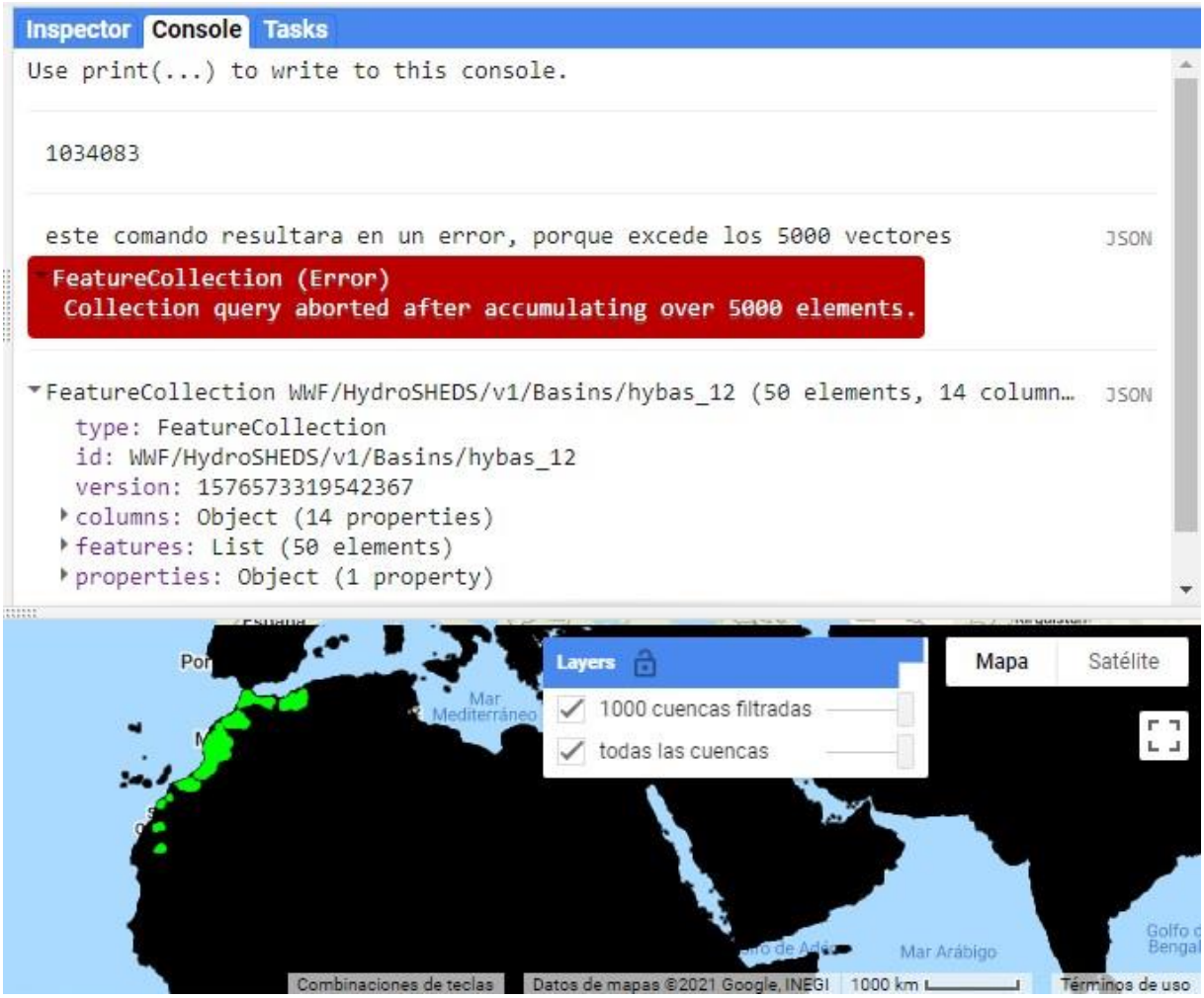


Figura 8.2: Visualización de las primeras 1000 cuencas (verdes) y todas las demás cuencas (negro), así como su salida en la consola.

8.2 Creación de colecciones de vectores

Una colección de vectores se define a partir del método `ee.FeatureCollection` y puede contener geometrías, vectores o inclusive otra colección de vectores. Una

colección de vectores puede contener un solo vector o una sola geometría y sus elementos pueden tener o no atributos. Por ejemplo (Fig. 8.3, 8.4):

| EJERCICIO 15

```
// Crear una colección de vectores con un solo punto, sin
// atributos
var coleccion = ee.FeatureCollection(ee.Geometry.Point(16.37,
                                                        48.225));
```

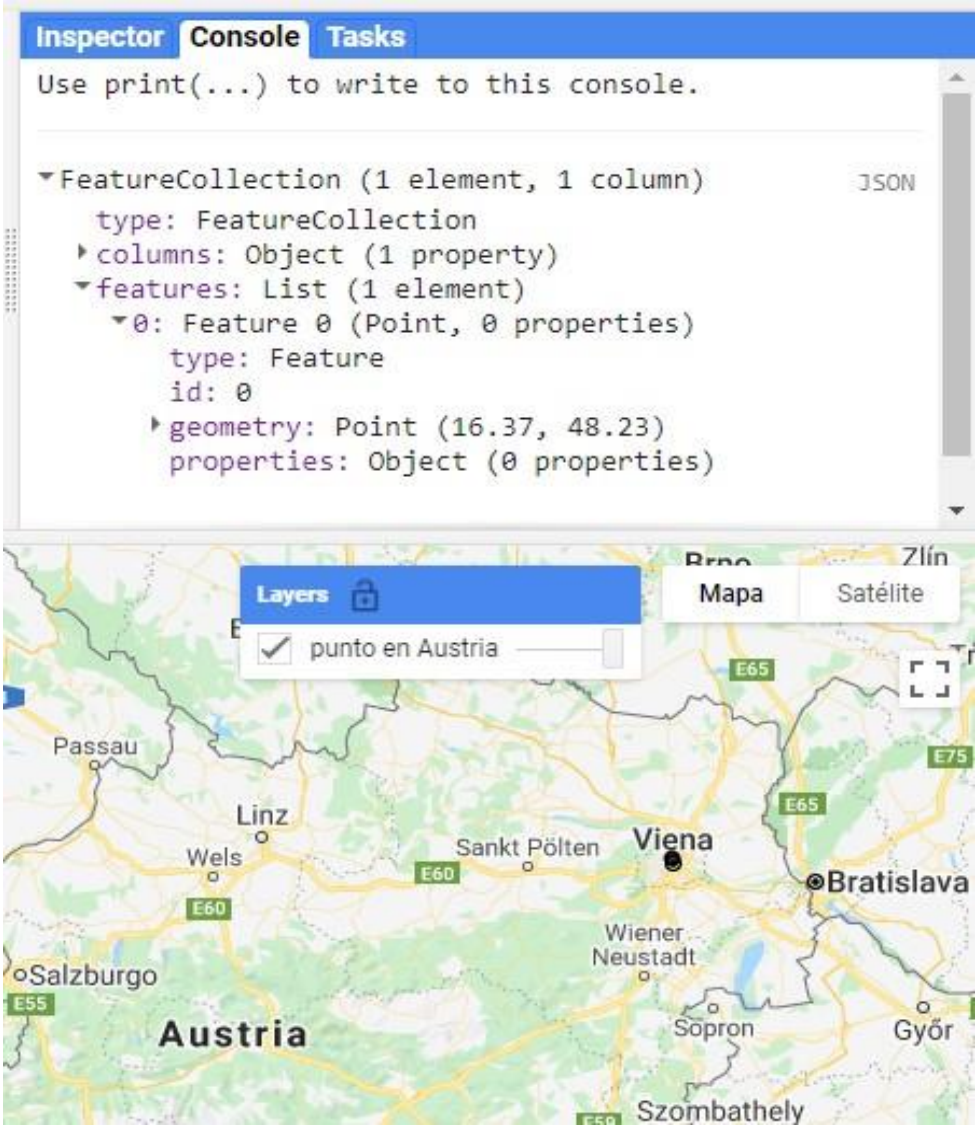


Figura 8.3: Visualización de una colección de vectores con un único objeto de geometría de tipo punto.

| EJERCICIO 16

```
// Información adicional para el ejemplo, creación de
// distintos vectores (ee.Feature),
// cada cual con diferentes tipos de geometrías y distintos
// atributos, esto para mostrar que una ee.FeatureCollection
// no requiere que todos sus elementos (ee.Feature)
// tengan los mismos atributos ni geometrías
var Punto = ee.Feature(ee.Geometry.Point( -99.1362, 19.4352),
    {pais:'Mexico',capital:'cdmx'});

var Multipunto = ee.Feature(ee.Geometry.MultiPoint(
    [ -90.496, 14.605,     -88.717, 17.245,     -87.157, 14.094,
      -89.2275, 13.6937,   -86.273, 12.115,     -84.097, 9.958,
      -79.527, 8.983]),
    { pais: [  'Guatemala',    'Belice',    'Honduras',
               'El Salvador',  'Nicaragua', 'Costa Rica',
               'Panama'],
      capital: ['cdad. de Guatemala',   'Belmopan', 'Tegucigalpa',
                'San Salvador',          'Managua',  'San Jose',
                'cdad. de Panama']});

var linea = ee.Feature(ee.Geometry.LineString(
    [ -66.865, 10.479, -74.072,4.732]),
    { pais:    'Venezuela - Colombia',
      capital: 'Caracas - Bogota'});

var Multilinea = ee.Feature(ee.Geometry.MultiLineString([
    ee.Geometry.LineString([ -76.803, 18.021, -82.362, 23.121]),
    ee.Geometry.LineString([ -72.31, 18.69, -69.98, 18.44]),
    ee.Geometry.LineString([ -66.103, 18.4597, -61.8558, 17.1315]),
    ee.Geometry.LineString([ -55.165, 5.861, -58.149, 6.822])]),
    {pais: [   'Jamaica - Cuba',
               'Haiti - Republica Dominicana',
               'Puerto Rico - Antigua y Barbuda',
               'Surinam - Guyana'],
     capital:[ 'Kingston - La Habana',
               'Puerto Principe - Santo Domingo',
               'San Juan - saint John',
               'Paramaribo - Georgetown']});

var perimetro = ee.Feature(ee.Geometry.LinearRing([
    -79.354, 27.32,
    -79.332, 23.661,
    -74.19, 21.161,
    -72.235, 23.165,
    -78.039, 27.859]),
    {pais:'Bahamas', capital: 'Nasau'});
```

```
var rectangulo = ee.Feature(ee.Geometry.Rectangle(
    -127.97, 50.25, -68.2, 27.63),
    {pais:'eeuu',capital:'Washington'});

var poligono = ee.Feature(ee.Geometry.Polygon(
    -80.348,  -3.36,   -80.348,   -3.36,   -78.985,   -5.11,
    -78.019,  -3.228,  -75.99,    -2.394,  -75.25,    -0.901,
    -75.865,  0.11,    -78.941,   1.428,   -80.26,    0.725,
    -80.919,  -1.911,  -79.776,   -2.614),
    {pais:'Ecuador', capital:'Quito'});

var Multipoligono = ee.Feature(ee.Geometry.MultiPolygon([
    ee.Geometry.Polygon([ -61.955,  10.146,  -61.713,   10.967,
                          -60.593,  10.881,  -60.68,    10.081]),
    ee.Geometry.Polygon([ -61.779,  12.269,  -61.504,   12.205,
                          -61.57,   11.968,  -61.845,   12.044]),
    ee.Geometry.Polygon([ -59.703,  13.042,  -59.417,   13.032,
                          -59.45,   13.32,   -59.736,   13.342]),
    ee.Geometry.Polygon([ -61.526,  15.671,  -61.087,   15.512,
                          -61.274,  15.194,  -61.438,   15.279])]),
    {pais:['Trinidad y Tobago',   'Granada',
           'Barbados',            'Dominica'],
    capital:['Puerto España',     'Chantimelle',
            'Bridgetown',         'Roseau']});

// Crear una lista que incluya todos los vectores
var coleccionVectores = ee.FeatureCollection([
    Multipoligono,    poligono,    rectangulo,    perimetro,
    Multilínea,       linea,       Multipunto,    Punto]);
```

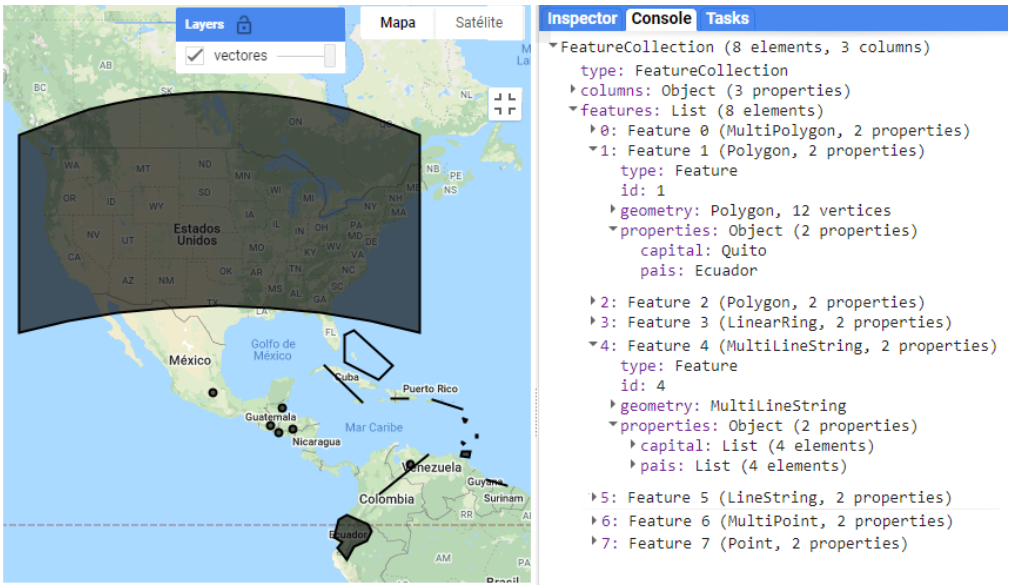


Figura 8.4: Visualización de una colección de vectores con objetos con diferentes geometrías, así como su salida en la consola y algunas propiedades desplegadas (nótese los tipos diferentes de geometrías).

8.3 Visualización de colecciones de vectores

Al igual que los vectores, las colecciones de vectores se pueden agregar a la pantalla de mapa mediante la función `Map.addLayer` (Fig. 8.5).

| EJERCICIO 17

```
var ecoregiones =
ee.FeatureCollection('RESOLVE/ECOREGIONS/2017');

// Especificar el color de la capa
Map.addLayer(ecoregiones, {}, 'capa por default');
Map.addLayer(ecoregiones, {color: '2AF116'}, 'capa color verde');
```

Además, solo para las colecciones de vectores se pueden especificar parámetros adicionales con el método `.draw` (el cual transforma la colección de vectores en una imagen y permite así especificar tamaño del punto **pointRadius** y grosor **strokeWidth** de la línea). Esto se logra con el método `.draw` (Fig. 8.5).

| EJERCICIO 17.1

```
Map.addLayer(ecoregiones.draw({color: 'FF6969',
    strokeWidth:10}),{}, 'capa color rosa, y línea gruesa');
```

Si queremos visualizar solo los contornos de los vectores se puede usar el siguiente código, el cual dibuja con el método `.paint` los contornos de los vectores sobre una imagen vacía (`ee.Image`), y luego se visualiza esa imagen (Fig. 8.5):

| EJERCICIO 17.2

```
// Crear una imagen vacía
var vacia = ee.Image().byte();

// Sobre la imagen vacía, dibujar los bordes de los vectores
var contornos = vacia.paint({
// Definir los vectores a dibujar
featureCollection: ecoregiones,
// Definir un atributo de los vectores para agrupar vectores por
// color según este atributo (puede omitirse)
color:'BIOME_NUM',
// Definir el grosor (si no se especifica este parámetro se
// dibujarán los vectores con color sólido)
width: 3 });
```

```
// Mostrar la imagen con los vectores dibujados y especificar la
// paleta de color (la cual en este caso es de un solo color, por
// lo tanto, todos los contornos serán del mismo color)
Map.addLayer(contornos, {palette: '6e00ff', max: 14},
    'bordes morados');

// Mostrar la imagen con los vectores dibujados y especificar la
// paleta de color en una lista [rojo, verde, azul]. Esto permite
// que los vectores se dibujen de diferentes colores según el
// atributo 'color' de la variable 'contorno'
Map.addLayer(contornos, {palette: ['FF0000', '00FF00', '0000FF'],
    max: 14}, 'bordes de colores');
```

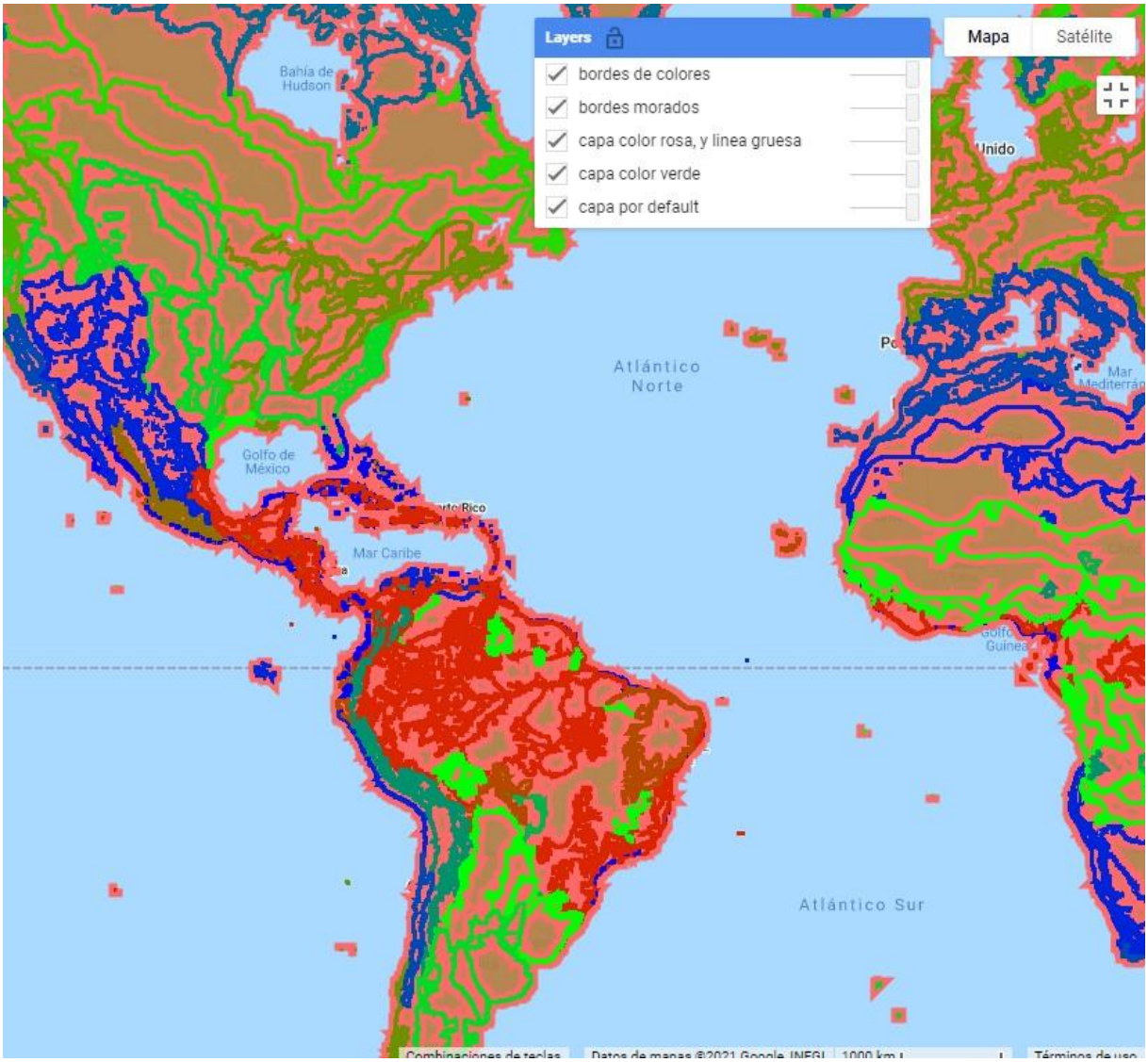


Figura 8.5: Diferentes formas de visualizar colecciones de vectores.

8.4 Métodos comunes

Filtración de una colección de vectores

Filtrar una colección de vectores permite seleccionar únicamente los vectores deseados, y se puede filtrar por fecha o por localización. De todas formas, si se necesitan filtros más avanzados se pueden configurar. Para filtrar por localización se usa el método .filterBounds, el cual permite seleccionar todos los vectores de la colección que se intersecten con la geometría indicada por el usuario. En el ejercicio se va a filtrar espacialmente una colección de vectores que contiene las áreas protegidas del mundo para seleccionar únicamente las áreas que se intersecten con un rectángulo que abarca el país de Costa Rica (Fig. 8.6).

| EJERCICIO 18

```
// Tabla dentro de GEE que contiene vectores de áreas protegidas
// globales
var areasProtegidas =
    ee.FeatureCollection('WCMC/WDPA/current/polygons');

// Dibujar una geometría (puede ser cualquier tipo de geometría),
// en este caso escogimos un rectángulo sobre Costa Rica)
var RecCostaRica = ee.Geometry.Rectangle( -86.16, 11.235,
                                          -82.601, 8.073);

// Áreas protegidas que se intersectan con el rectángulo dibujado
var filtro = areasProtegidas.filterBounds(RecCostaRica);
```

Para filtrar por atributos de los vectores se puede utilizar el método .filterMetadata. De esta forma, todos los vectores de la colección que cumplan con una condición deseada serán seleccionados. Para usar .filterMetadata se debe indicar dentro del paréntesis (separado por comas) el nombre entre comillas del atributo a evaluar y el tipo de comparación a hacer, de acuerdo con ciertos operadores lógicos. Los operadores lógicos aceptados en este método son:

- equals (igual a).
- less_than (menor que).
- greater_than (mayor que).
- not_equals (no igual a o diferente de).
- not_less_than (no menor que).
- not_greater_than (no mayor que).
- starts_with (empieza con).
- ends_with (termina con).

- not_starts_with (no empieza con).
- not_ends_with (no termina con).
- contains (contiene).
- not_contains (no contiene).

Y finalmente el valor con el que se hará la comparación.

En el siguiente ejercicio vamos a filtrar la colección de áreas protegidas según el atributo `'ISO3'`, que es un código de 3 letras para cada país del mundo, en el caso de Costa Rica su código es `'CRI'` (Fig. 8.6).

| EJERCICIO 18.1

```
// En este caso cada vector de la colección de áreas protegidas
// tiene un atributo llamado 'ISO3'; basándonos en ese atributo,
// vamos a buscar todos los vectores que tengan (para ese
// atributo) un valor exactamente igual ('equals') a 'CRI'
// (abreviación de Costa Rica)
var AreasProCRI =
areasProtegidas.filterMetadata('ISO3','equals','CRI');
```

El método para filtrar por fecha es `.filterDate`, sin embargo, este solo se puede utilizar si los vectores tienen un atributo llamado **'system:time_start'** y en ese atributo se encuentra la fecha en formato '**AAAA-MM-DD**'. En caso de que los vectores no tengan ese atributo pero sí tengan un atributo de fecha, entonces será necesario renombrar dicho atributo de fecha a '**system:time_start**', para ello se puede usar el método `.set` y el método `.map` (el método `.map` se explicará con mayor detalle más adelante en esta sección).

Funciones de usuario

En el siguiente ejercicio vamos a crear una función para extraer de cada vector de la colección el valor de año del atributo `'STATUS_YR'`, y guardarlo en un nuevo atributo llamado `'system:time_start'`. Una vez se haya realizado este proceso, procederemos a filtrar las áreas protegidas establecidas entre el 2020 y 2021 (Fig. 8.6).

| EJERCICIO 18.2

```
// Map es un método que nos permite ejecutar una operación sobre
// todos los vectores de la colección
var Areasfecha = AreasProCRI.map(function(vector){
    // Fecha extrae el valor del atributo 'STATUS_YEAR'
    var fecha = vector.get('STATUS_YR');
    // Cada vector se entrega con una nueva propiedad
    // 'system:time_start'que tiene fecha
    return vector.set('system:time_start',fecha)
    });

// Ahora ya se puede aplicar el filtro de fecha filtrando solo las
// áreas protegidas de Costa Rica declaradas en el año 2020
var filtroFechas = Areasfecha.filterDate(2020,2021);
```

Los filtros compuestos permiten combinar los filtros disponibles bajo la biblioteca de `ee.Filter`. En el ejercicio 18.3 se mostrará cómo construir un filtro combinado para filtrar vectores que cumplan con todas las condiciones. El ejercicio consta de los siguientes pasos:

1. El método `.filter` es para indicar que vamos a realizar un filtro.
2. `ee.Filter.and` es para indicar que se deben cumplir todas las condiciones para filtrar.
3. `ee.Filter.gte` es para seleccionar mediante la condición mayor o igual a (*greater than or equal*).
4. Después se utiliza una coma para indicar otro filtro.
5. `ee.Filter.stringContains` es un filtro para seleccionar de acuerdo a criterios de cadenas de texto que se encuentren como atributos de los vectores.
6. Después se usa otra coma para indicar otro filtro.
7. `ee.Filter.rangeContains` es para seleccionar todo lo que se encuentre en un rango de valores (Fig. 8.6).

| EJERCICIO 18.3

```
// Se declara que se ejecutará un filtro
var areasProMar = areasProtegidas.filter(
    // ee.Filter.and indica que se crea un filtro compuesto en
    // donde se deben cumplir TODAS las siguientes condiciones
    // para seleccionar un vector.
    ee.Filter.and(
          // Un filtro donde se indica el atributo a utilizar
          // ('MARINE'). Este es un atributo donde 1 es área
          // costera y 2 son áreas marinas. Por lo tanto al llamar
          // ee.Filter.gte con el valor 1, se seleccionarán todas
          // las áreas con un valor de 1 o 2, que indican áreas
          // que están protegiendo un área marina.
          ee.Filter.gte('MARINE',1),
          // Se filtran las áreas protegidas que en su atributo
          // 'NAME' (el nombre del área protegida) contengan la
          // palabra 'manglar', es así para seleccionar solo las
          // áreas protegidas cuyo nombre incluya la palabra
          // manglar.
          ee.Filter.stringContains('NAME', 'Manglar'),
          // 'GIS_M_AREA' es un atributo de las áreas protegidas
          // que indica el área marina protegida calculada con un
          // SIG. El valor mínimo del rango de área marina que
          // queremos filtrar es 10, mientras que 100 es el valor
          // máximo.
          ee.Filter.rangeContains('GIS_M_AREA', 10,100)
          )
);
```

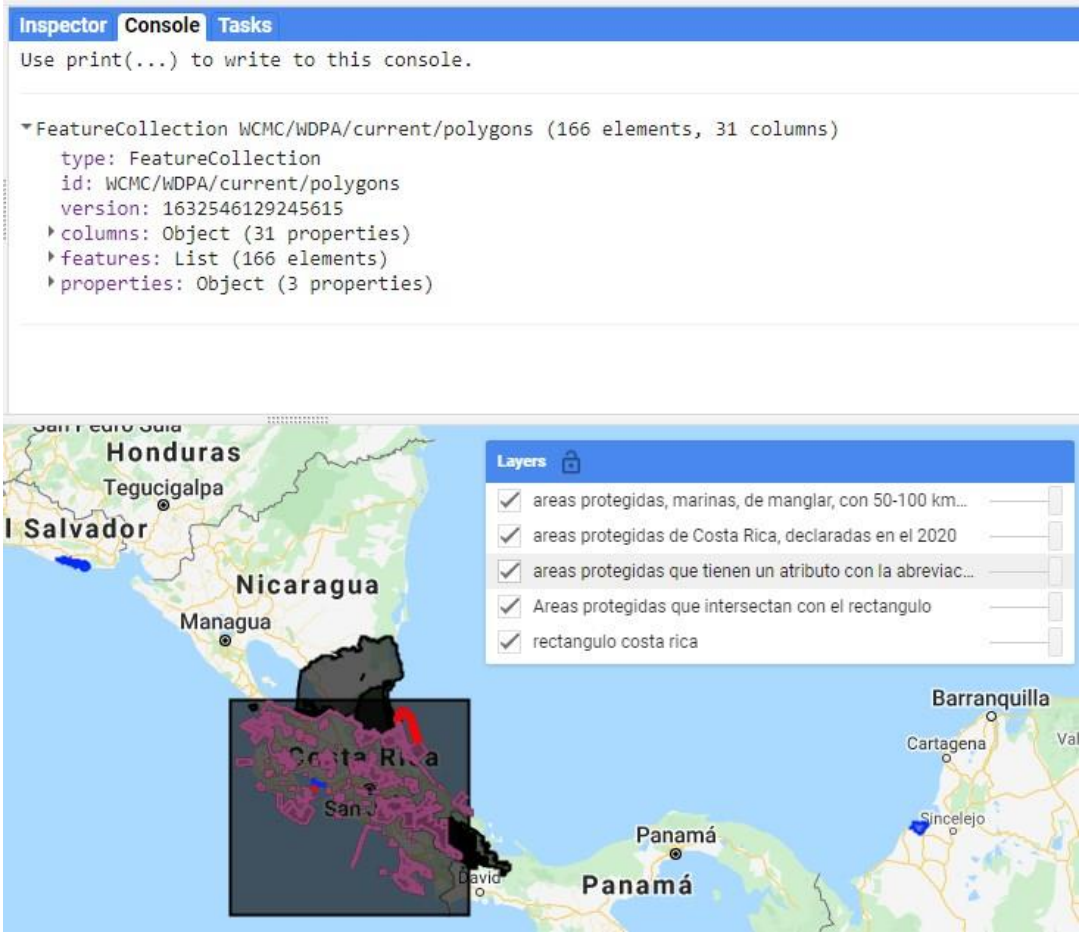


Figura 8.6: Visualización de los resultados de diferentes filtros sobre una colección de áreas protegidas.

Selección de atributos

Al igual que con los vectores, se puede utilizar el método `.select` para seleccionar un conjunto de atributos de los vectores en la colección. De igual manera que con los vectores individuales, para seleccionar un único atributo solo se requiere indicar su nombre, mientras que si se desea seleccionar varios atributos, estos deben indicarse dentro de una lista. Por ejemplo:

```
ee.FeatureCollection.select(['atributo'])
```

Creación de puntos aleatorios

GEE ofrece la herramienta para generar una colección de puntos aleatorios en una región especificada, mediante el método `.randomPoints`. En ella, especifica el área de interés y la cantidad de puntos deseados (Fig. 8.7).

| EJERCICIO 19

```
// Determinar una región específica donde se quiere lanzar puntos
// aleatorios
var region = ee.Geometry.Rectangle(-61.36, 31.2, -16.54, 1.93);

// Crear 1000 puntos aleatorios en la región especificada
var puntos = ee.FeatureCollection.randomPoints(region,1000);
```

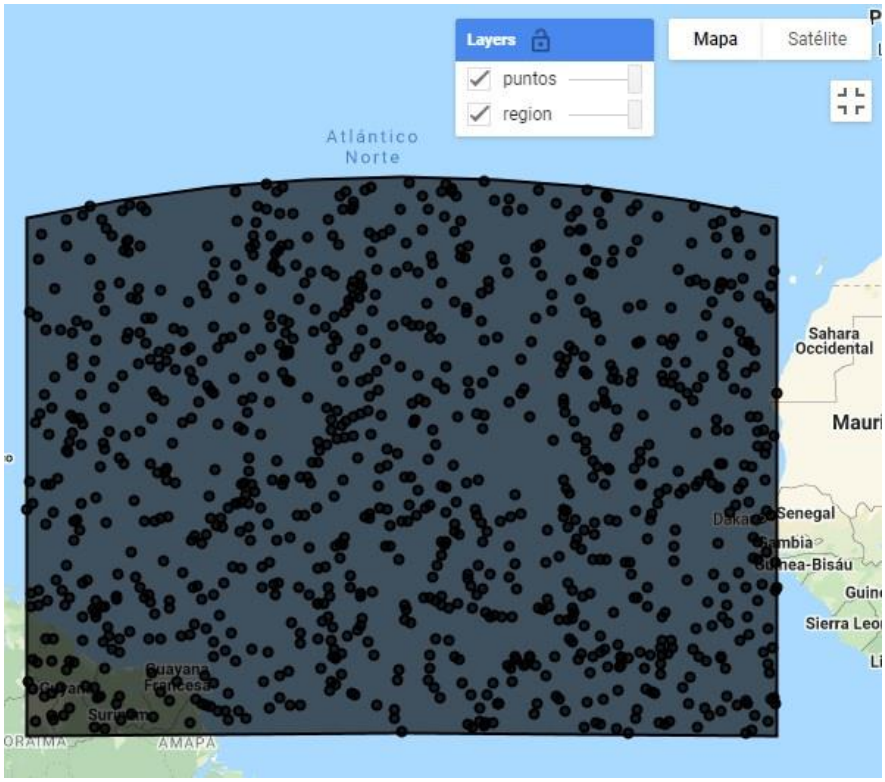


Figura 8.7: Visualización de los puntos aleatorios creados dentro de un área de interés.

Obtención del primer vector

En algunas ocasiones resulta útil obtener el primer vector de una colección de vectores para ver si se llevó a cabo el procedimiento deseado sobre la colección. Para ello, se utiliza el método `.first`, que devuelve el primer vector que se encuentra en la colección y permite agregarlo en la pantalla de mapa para analizarlo más a detalle. Al usar este método sobre una colección de vectores (`ee.FeatureCollection`), el resultado devuelto será un vector (`ee.Feature`, Fig. 8.8).

| EJERCICIO 20

```
// Llamamos una capa de cuencas de alta resolución de GEE
var CUENCAS_MUNDIAL = ee.FeatureCollection(
'WWF/HydroSHEDS/v1/Basins/hybas_12');

// Filtramos la colección solo al primer vector y lo convertimos
// de una colección de un solo elemento (ee.FeatureCollection) a
// un vector independiente (ee.Feature)
var cuenca = ee.Feature(CUENCAS_MUNDIAL.first());
```

Extracción de metadatos de una colección de vectores

Para extraer los nombres de los metadatos de una colección de vectores se usa el método `.propertyNames`, mientras que para extraer algún metadato en particular de una colección de vectores se debe utilizar el método de `.get`, indicando dentro de los paréntesis el nombre de la propiedad (Fig. 8.8).

> Los metadatos de una colección de vectores no son los mismos que los de los vectores que la conforman.

| EJERCICIO 20.1

```
// Mostrar un metadato de la colección de vectores
var IDcoleccion = CUENCAS_MUNDIAL.get('system:id')

// Se muestran los metadatos de la colección de vectores
var metadatosCOL = CUENCAS_MUNDIAL.propertyNames()
```

Extracción de listas con atributos de vectores

Para obtener una lista con los atributos de los vectores que conforman una colección de vectores se utiliza el método `.aggregate`. Este método tiene varias formas de obtener y resumir los atributos de una colección: extraer una propiedad de los vectores (`.aggregate_array`), extraer y calcular la media por propiedad (`.aggregate_mean`), extraer y calcular un histograma (`.aggregate_histogram`), o extraer y obtener estadísticas descriptivas de las propiedades (`.aggregate_stats`; Fig. 8.8).

| EJERCICIO 20.2

```
// Rectángulo que abarca las islas canarias
var canaria = ee.Geometry.Rectangle( -18.89,27.46, -13.26,29.93);

// Arroja una lista con todos los ID de las cuencas de esta zona
var IDcuencasCana = CUENCAS_MUNDIAL.filterBounds(canaria)
    .aggregate_array('HYBAS_ID');

// Arroja el dato del área subsuperficial promedio de las cuencas
// de esta zona
var areaCuencasCana = CUENCAS_MUNDIAL.filterBounds(canaria)
    .aggregate_mean('SUB_AREA');

// Arroja el dato de frecuencias de cada uno de los posibles
// valores de COAST (1=si tiene costa, 0=no tiene costa)
var costaCana = CUENCAS_MUNDIAL.filterBounds(canaria)
    .aggregate_histogram('COAST');

// Arroja las estadísticas descriptivas del área superficial de
// las cuencas de la zona
var estadisticas = CUENCAS_MUNDIAL.filterBounds(canaria)
    .aggregate_stats('UP_AREA');
```

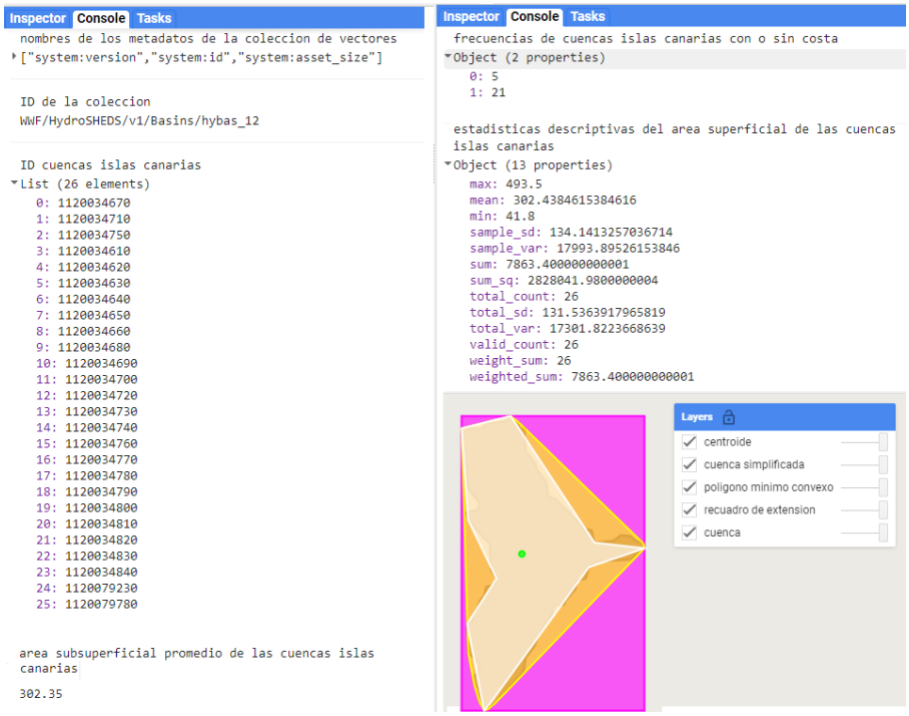


Figura 8.8: Ejemplos de la salida en la consola de diferentes propiedades de los vectores que forman una colección de vectores de cuencas.

Ejecución de una función sobre todos los vectores de una colección

Para aplicar una operación en todos los vectores de una colección se usa el método `.map`.

> Para poder ejecutar el método `.map` es necesario definir antes una función (del usuario). Para definir una función es necesario tener en cuenta varias cosas:
>
> Usar el comando **`function`**.
>
> Definir un nombre cualquiera, que representará el objeto de entrada de la función (insumo).
>
> Definir un diccionario dentro del cual se especifiquen los procedimientos a realizar sobre el insumo.
>
> Antes de cerrar el diccionario especificar el producto de salida (resultado de la función) usando el comando **`return`**.

En el siguiente ejercicio se mostrará cómo aplicar una función que permite crear un buffer sobre todos los vectores de una colección de vectores. En este ejercicio se creará un conjunto de puntos aleatorios para Perú, sobre los cuales se crearán los buffers y se agregarán algunos atributos. Para ello, se realizarán los siguientes pasos (Fig. 8.9):

- Llamamos a una colección con los límites políticos de los países.
- Seleccionamos a `'Peru'`.
- Generamos 100 puntos aleatorios en Perú.
- Le añadimos a cada punto un nuevo atributo llamado `'aleatorio'`.
- Definimos una función del usuario en la que para cada punto se genera un buffer con tamaño aleatorio.
- Ejecutamos la función sobre todos los puntos aleatorios.
- Usando el método `.map` ejecutamos la función **bufferAleatorio** sobre todos los puntos aleatorios.

| EJERCICIO 21

```
// Se llama una tabla de GEE con colección de vectores de países
var paises = ee.FeatureCollection('USDOS/LSIB/2017');

// Seleccionamos el vector cuyo atributo 'COUNTRY_NAME' es igual
// ('equals') a 'Peru', filtrando así solo a Perú
var peru = paises.filterMetadata('COUNTRY_NA','equals','Peru');

// Creamos una colección de vectores de 100 puntos aleatorios en
// Perú, y usando el método .randomColumn le asignamos a cada
// punto un nuevo atributo llamado 'aleatorio' cuyo valor es
// aleatorio (entre 0-1)
var puntos = ee.FeatureCollection.randomPoints(peru, 100)
    .randomColumn('aleatorio');

// Definimos una función del usuario (con el método function)
// llamada 'bufferAleatorio' en la que el dato de entrada (insumo)
// será un vector (y el nombre genérico que le asignamos al insumo
// es (punto)): Abrimos un diccionario donde especificaremos los
// procedimientos a realizarle al vector (punto).
var bufferAleatorio = function(punto){
    // Extraemos el número aleatorio del vector que se encuentra en
    // el atributo 'aleatorio'.
    var numero = punto.get('aleatorio');
    // Convertimos el número extraído en el paso anterior en un
    // ee.Number del servidor.
    var NumServidor = ee.Number(numero);
    // Multiplicamos ese ee.Number por 100 000.
    var multiplicar = NumServidor.multiply(100000);
    // Redondeamos el resultado de la multiplicación en el número
    // entero mayor o igual más cercano.
    var redondear = multiplicar.ceil();
    // Creamos un buffer (area) alrededor del vector de entrada de
    // longitud definida según el número redondeado (redondear).
    var area = punto.buffer(redondear);
    // Le asignamos al buffer (area) un nuevo atributo llamado
    // 'radio del buffer' con el número redondeado (metros usados
    // para calcular el buffer).
    var tamaño = area.set('radio del buffer',redondear);
    // Le asignamos al buffer (tamaño) otro atributo llamado
    // 'area', en donde calculamos el área del buffer usando el
    // método .area.
    var circulo = tamaño.set('area',tamaño.area());
    // Usando el método return especificamos que el buffer
    // (circulo) con los dos nuevos atributos será el resultado de
    // esta función.
return circulo;
};
```

```
// Ejecutamos la función 'bufferAleatorio' sobre todos los
vectores
// de la colección, cada punto de la colección entonces entrará
// como insumo de la función bufferAleatorio
var circulos = puntos.map(bufferAleatorio);
```

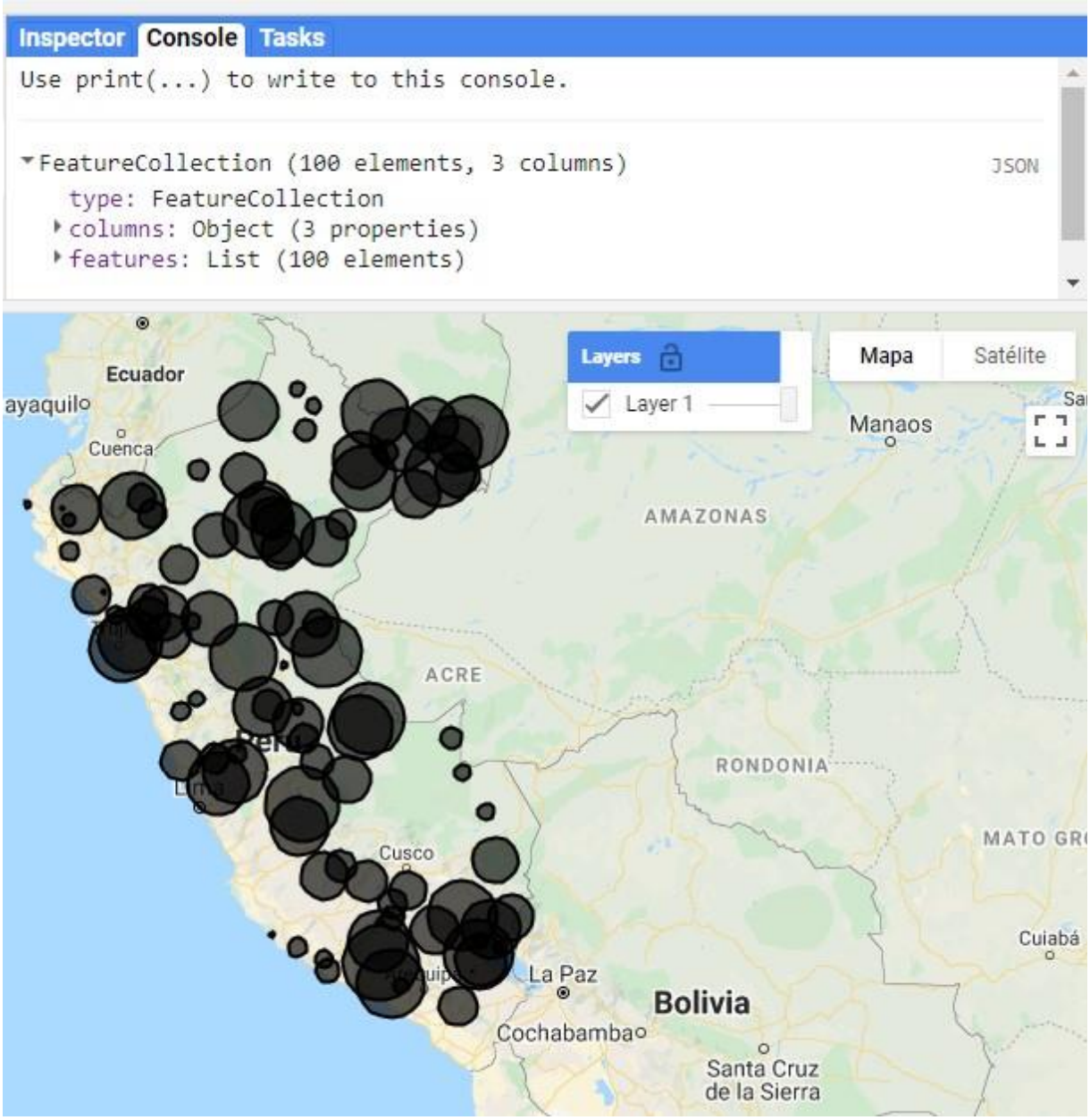


Figura 8.9: Visualización de varios buffers, así como la salida de la consola.

Reducción de una colección de vectores

Las colecciones también permiten agregar atributos de sus vectores en los metadatos de la colección, esta agregación se llama reducir (reduce). Los reductores de atributos tomarán los valores de un atributo de todos los vectores y los convertirán en un solo valor para toda la colección. Depende del usuario escoger la operación matemática que se utilizará para la reducción, ya sea promedio, mínimo, máximo, mediana, entre otras. Estos reductores entregarán como resultado un diccionario. En el siguiente ejercicio se calculará el área de un conjunto de cuencas en Hawái y se comparará con el área reportada para cada cuenca de acuerdo a la USGS (United States Geological Survey). Por último, se calculará el error cuadrático medio de la diferencia para la colección de vectores. Para ello, se realizarán los siguientes pasos (Fig. 8.10):

- Importamos una capa de cuencas, la filtramos con un rectángulo sobre Hawái.
- Definimos una función del usuario (**`function`**) llamada 'areaDife' en la que el dato de entrada (insumo) será un vector (y el nombre genérico que le asignamos al insumo es ('cuenca')).
- Abrimos un diccionario donde especificaremos los procedimientos a realizarle al vector (cuenca).
- Calculamos el área del vector (con el método `.area`) y lo dividimos por `1 000 000` para convertir a km^2.
- Con `.get` extraemos el valor del atributo `'areasqkm'` (el cual es una cadena de texto). Con `.parse` convertimos el valor extraído anteriormente a un número del servidor (`ee.Number`).
- Usando el método `.subtract` restamos el área calculada con el método `.area` menos el área reportada según el atributo del vector.
- Usando el método **`return`** especificamos que el vector sea devuelto con un nuevo atributo llamado `'diferencia'` (el cual contiene el valor de la resta elevado al cuadrado usando el método `.pow`).
- Ejecutamos la función 'areaDife' sobre todos los vectores de la colección cuencasHawaii. Usando el método `.reduceColumns` declaramos que usaremos un reductor sobre toda la colección enfocándonos en un atributo de los vectores.
- Con `ee.Reducer.mean` declaramos que el reductor a utilizar será uno de promedio, por lo que el resultado será el promedio del atributo `'diferencia'`.
- Con `.get` extraemos el valor del reductor que por defecto se llama `'mean'`. Con `.sqrt` calculamos la raíz cuadrada del promedio calculado.

| EJERCICIO 22

```
// Importar una capa de cuencas
 var cuencasHawaii = ee.FeatureCollection('USGS/WBD/2017/HUC06')
    .filterBounds(ee.Geometry.Rectangle(-160.969, 23.104,
                                        -154.136, 18.733));

// Definir una función que ejecute la diferencia entre el área
// reportada en el atributo vector y la calculada en GEE
var areaDife = function(cuenca) {
    var area = cuenca.area().divide(1000000);
    var dife = area.subtract(ee.Number.parse(
         cuenca.get('areasqkm')));
    return cuenca.set('diferencia', dife.pow(2));
};

// Calcular el error cuadrático medio de toda la colección.
var RECM = ee.Number(cuencasHawaii.map(areaDife)
    .reduceColumns(ee.Reducer.mean(), ['diferencia'])
    .get('mean')).sqrt();
```

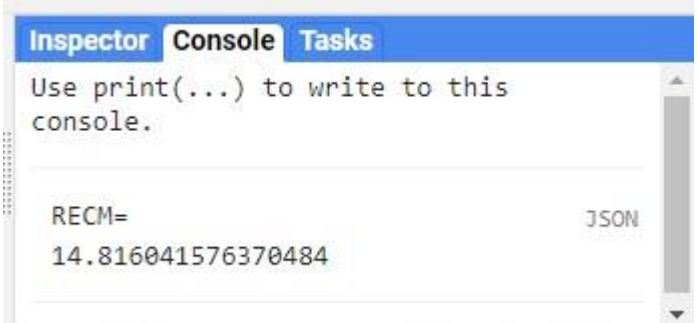


Figura 8.10: Vista en la consola del error cuadrático medio calculado entre el área calculada directamente en GEE y el reportado por la fuente de datos utilizada.

También se puede usar una colección de vectores para reducir la información de una imagen. Este procedimiento se revisa con mayor detalle en el capítulo 11. Sin embargo, por el momento se ejemplifica cómo obtener el valor de la suma de algunas variables climáticas (datos de **WorldClim**) en formato ráster para dos cuencas en Puerto Rico (objetos de tipo vector). En este caso se realizará el siguiente procedimiento (Fig. 8.11):

- Filtrar las cuencas de Puerto Rico.
- Obtener la suma de los valores de todos los píxeles que caigan en cada cuenca.

| EJERCICIO 23

```
// Importar una imagen de clima de WorldClim
var clima = ee.Image('WORLDCLIM/V1/BIO');

// Importar una capa de cuencas y filtrar las cuencas con un
// rectángulo sobre Puerto Rico.
var cuencasPuertorRico =
ee.FeatureCollection('USGS/WBD/2017/HUC06')
    .filterBounds(ee.Geometry.Rectangle(-79.906, 25.196,
                                        -59.6, 9.776));

// Usando el método .reduceRegions declaramos que usaremos un
// reductor espacial en el área de las cuencas de Puerto Rico.
// Con ee.Reducer.sum especificamos que se desea obtener la suma
// de los valores de los atributos de todos los píxeles que caigan
// en cada vector.
var cuencaClima = clima.reduceRegions(cuencasPuertorRico,
    ee.Reducer.sum());
```

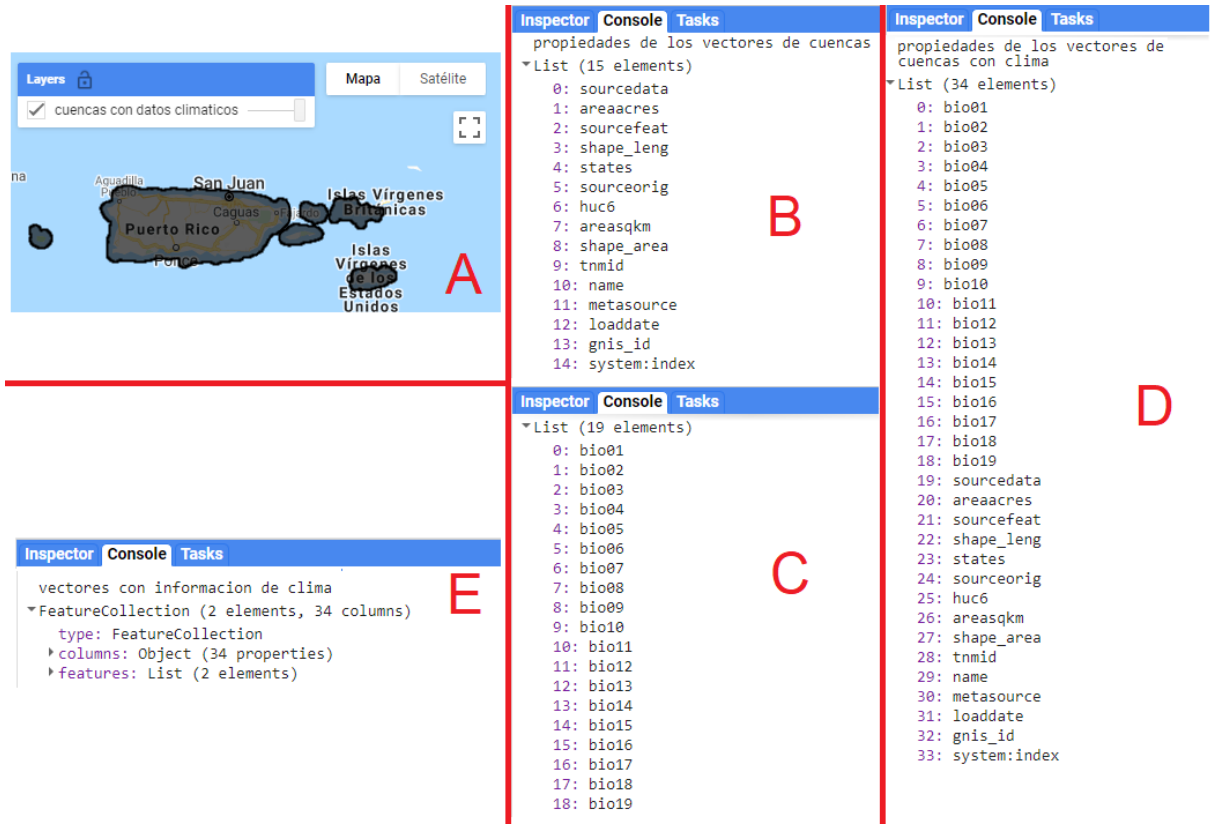


Figura 8.11: Visualización y salida de la consola (ejercicio 23). Colección de vectores con variables climáticas (mapa A, consola E), propiedades de los vectores (B), variables climáticas de un raster (C), propiedades de vectores adicionando las variables climáticas (D), vectores con variables climáticas (E).

Conversión de una colección de vectores en una imagen

Para rasterizar un vector se usa el método .reduceToImage. Este método asigna bajo cada vector los píxeles del valor del atributo especificado. Para usar .reduceToImage indicamos el atributo que corresponderá al valor de los píxeles del ráster como 'area'. Por último, con el método ee.Reducer.first especificamos que el primer valor que encuentre en ese atributo sea asignado al valor del píxel (Fig. 8.12).

| EJERCICIO 24

```
// Definir un área en la Antártida
var rectangulo = ee.Geometry.Rectangle(-158.63, -81.47,
                                       -40.51, -59.12);

// Importar una colección de vectores de GEE con datos de
// glaciares
var hielo = ee.FeatureCollection('GLIMS/current')
    .filterBounds(rectangulo);

// Crear una imagen ráster de las áreas de los glaciares.
var areaGlaciar = hielo
    // Seleccionar vectores cuyo atributo área no esté vacío
    .filter(ee.Filter.notNull(['area']))
    // Rasterizar
    .reduceToImage({
         // Atributo del vector para rasterizar
         properties: ['area'],
         // Tomar el primer valor
         reducer: ee.Reducer.first()
    });
```

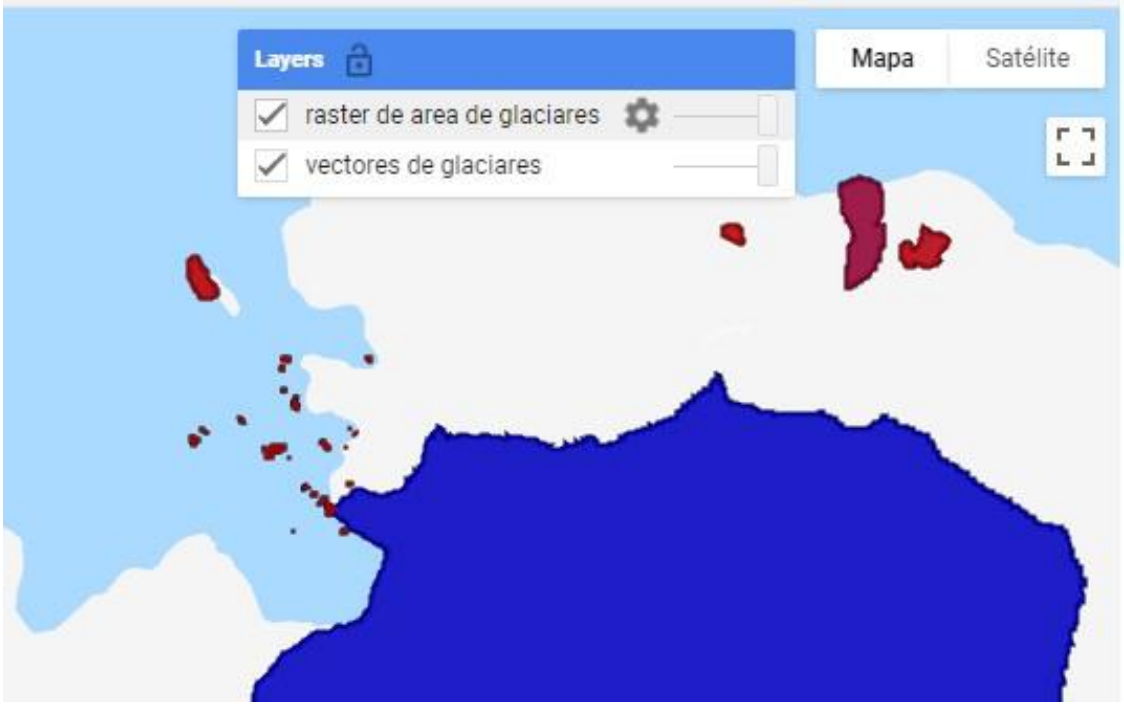


Figura 8.12: Visualización de la transformación de una colección de vectores a imagen.

Unión de colecciones de vectores

Para unir diferentes colecciones de vectores mediante los valores de sus atributos se usan las funciones de `ee.Join`. El resultado de estas funciones suele constar de una colección de vectores que contendrá los atributos de otra colección de vectores. Para determinar el tipo de relación entre los atributos de los vectores de ambas colecciones se utiliza un filtro (`ee.Filter`).

Adicionalmente, se debe definir el tipo de unión a realizar entre los vectores. Los diferentes tipos de uniones se pueden consultar en la biblioteca `ee.Join`. En el siguiente ejercicio realizaremos un tipo de unión espacial de tipo **saveALL**, pero lo usaremos con dos filtros diferentes (uno de intersección y otro de distancia).

En este ejercicio se busca saber cuáles áreas protegidas se encuentran a menos de 10 km de una planta de energía y ver cuáles áreas protegidas tienen una planta de energía dentro de sus límites (Fig. 8.13).

| EJERCICIO 25

```
// Importar una colección de vectores: áreas protegidas de
// Guatemala
var APGuatemala =
ee.FeatureCollection('WCMC/WDPA/current/polygons')
    .filter(ee.Filter.eq('PARENT_ISO', 'GTM'));

// Importar otra colección de vectores plantas de energía
var plantas = ee.FeatureCollection('WRI/GPPD/power_plants');

// Define un filtro espacial con distancia de 10 km. El .geo se
// utiliza para indicar que se comparan geometrías
var distFiltro = ee.Filter.withinDistance({ distance: 10000,
    // Geometría de la primera colección
     leftField: '.geo',
    // Geometría de la segunda colección
    rightField: '.geo', maxError: 10 });

// Define un filtro espacial de intersección, el .geo se utiliza
// para indicar que se compararán geometrías
var interFiltro = ee.Filter.intersects({
    // Geometría de la primera colección
    leftField: '.geo',
    // Geometría de la segunda colección
    rightField: '.geo', maxError: 0.1
});

// Definir la forma de unión el Join
var Union = ee.Join.saveAll({ matchesKey: 'plantas_energia_PA',
    measureKey: 'metros'});


// Ejecutar la unión por distancia
var AP_cerca_PE = Union.apply(APGuatemala, plantas, distFiltro);

// Ejecutar la unión por intersección
var AP_dentro_PE = Union.apply(APGuatemala, plantas, interFiltro);
```

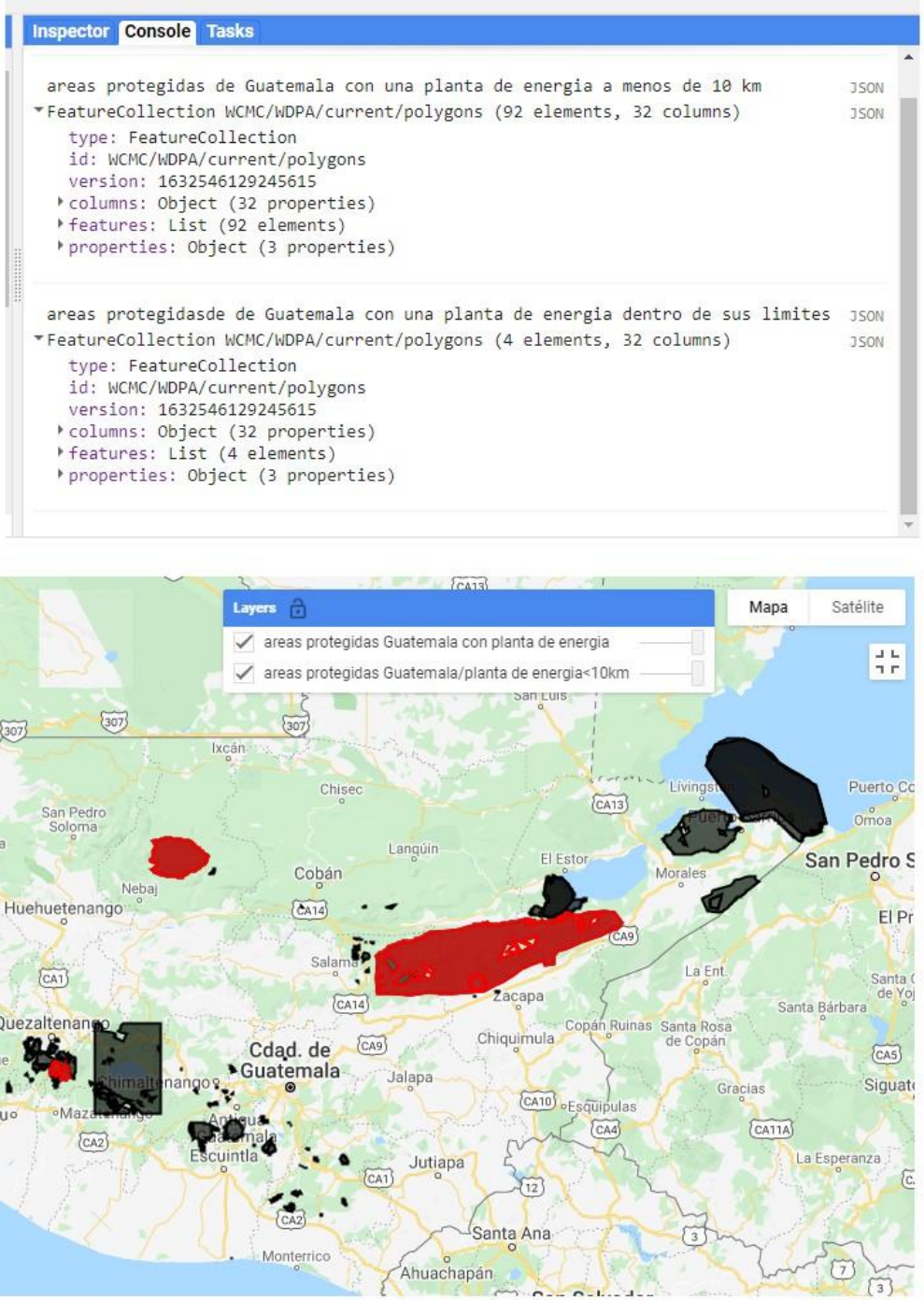


Figura 8.13: Visualización y salida de la consola del resultado de dos uniones distintas (rojo y negro) de dos colecciones de vectores (áreas protegidas y plantas de energía) mediante sus propiedades.

Exportación de colecciones de vectores

GEE permite exportar vectores a la sección de **Assets**, al **Google Drive** o al **Google Cloud**. Consideramos que las dos primeras son las que serán más utilizadas, ya que son

gratuitas y permiten fácilmente exportar datos. En cambio, la opción de **Google Cloud** es para exportar información a Google Cloud, que suele implicar un costo extra y un registro en dicha plataforma (que no es el mismo registro que GEE).

Para exportar vectores es necesario usar la función `Export.table` y especificar si va a ser para **drive** (`.toDrive`) o para los **Assets** (`.toAsset`). Sus argumentos en común son:

- **Collection** (colección): obligatoriamente una colección de vectores `ee.FeatureCollection`. Si se quiere exportar un solo vector se deberá convertir en una colección de un solo elemento.
- **Description** (descripción): es una cadena de texto de usuario (no debe contener espacios, ni símbolos; solo letras, números y guion bajo _). Esta cadena de texto es una descripción que el usuario quiera darle a la tarea de exportar.

> Es importante recordar que no se podrán exportar vectores a los Assets sin geometrías, ni vectores con geometría NULL. Siempre que se quiera exportar una colección de vectores a la sección de Assets será necesario especificar alguna geometría por vector de la colección.

Los argumentos específicos para exportar a **drive** `Export.table.toDrive` son:

- **Folder** (carpeta): cadena de texto del usuario que será el nombre de la carpeta donde se guardará el vector dentro del **drive**. Si no se especifica este argumento se guarda en el **drive** sin carpeta, aunque si se especifica un nombre de una carpeta que no existe, dicha carpeta se creará automáticamente.
- **FileNamePrefix** (prefijo del archivo): el prefijo a utilizar para nombrar al vector guardado en **drive**.
- **FileFormat** (formato del archivo): el tipo de formato en el que se exportará la colección de archivos. GEE permite las siguientes opciones:
  - `'CSV'`, valor por defecto, vectores separados por comas. Este formato consta de valores separados por comas, en el que cada coma indica una columna distinta. Este formato es el más utilizado para exportar colecciones de vectores sin geometrías (es decir, una simple tabla de atributos).
  - `'GeoJSON'` (*Geographic JavaScript Object Notation*). Formato basado en el lenguaje JSON (*JavaScript Object Notation*) para almacenar datos vectoriales y sus atributos.

- ‘KML’ (*Keyhole Markup Language*). Formato utilizado sobre todo por Google para almacenar datos espaciales y tablas de atributos asociados a ellos.
- ‘KMZ’ (*KML zipped*). Formato que consta de un KML comprimido.
- ‘SHP’ (*Shapefile*). Formato utilizado por ESRI para almacenar vectores y tablas de atributos asociados a ellos.
- ‘TFRecord’ (*Tensorflow Record*). Formato utilizado por Tensorflow para almacenar datos tanto vectoriales como ráster.

El argumento adicional para exportar a los **Assets** `Export.table.toAssets` es ‘AssetID’, que corresponde el nombre que le pondremos al nuevo archivo (Fig. 8.14).

| EJERCICIO 26

```
// Determinamos un área de estudio con un polígono
var poligono = ee.Geometry.Polygon(
    [-79.59, 7.00,    -85.02, 8.14,    -88.04, 10.68,
     -93.69, 9.44,    -98.51, 4.69,    -94.24, -2.93,
     -92.70, 0.56,    -89.13, 0.82,    -87.57, -1.55,
     -82.31, -0.08,   -79.64,  3.24]);

// Creamos la colección de vectores de puntos de ejemplo
var puntos = ee.FeatureCollection.randomPoints(poligono, 4000);

// Exportamos a los assets dentro de la nube
Export.table.toAsset({
    collection: punto,
    description: 'exportar_puntos_asset',
    assetId: 'puntos_exportados'
});

// Exportamos al drive personal
Export.table.toDrive({
    collection:punto,
    description:'exportar_puntos_drive',
    folder:'carpeta_clase_GEE',
    fileNamePrefix:'puntos_ejemplo',
    fileFormat:'SHP'
});
```

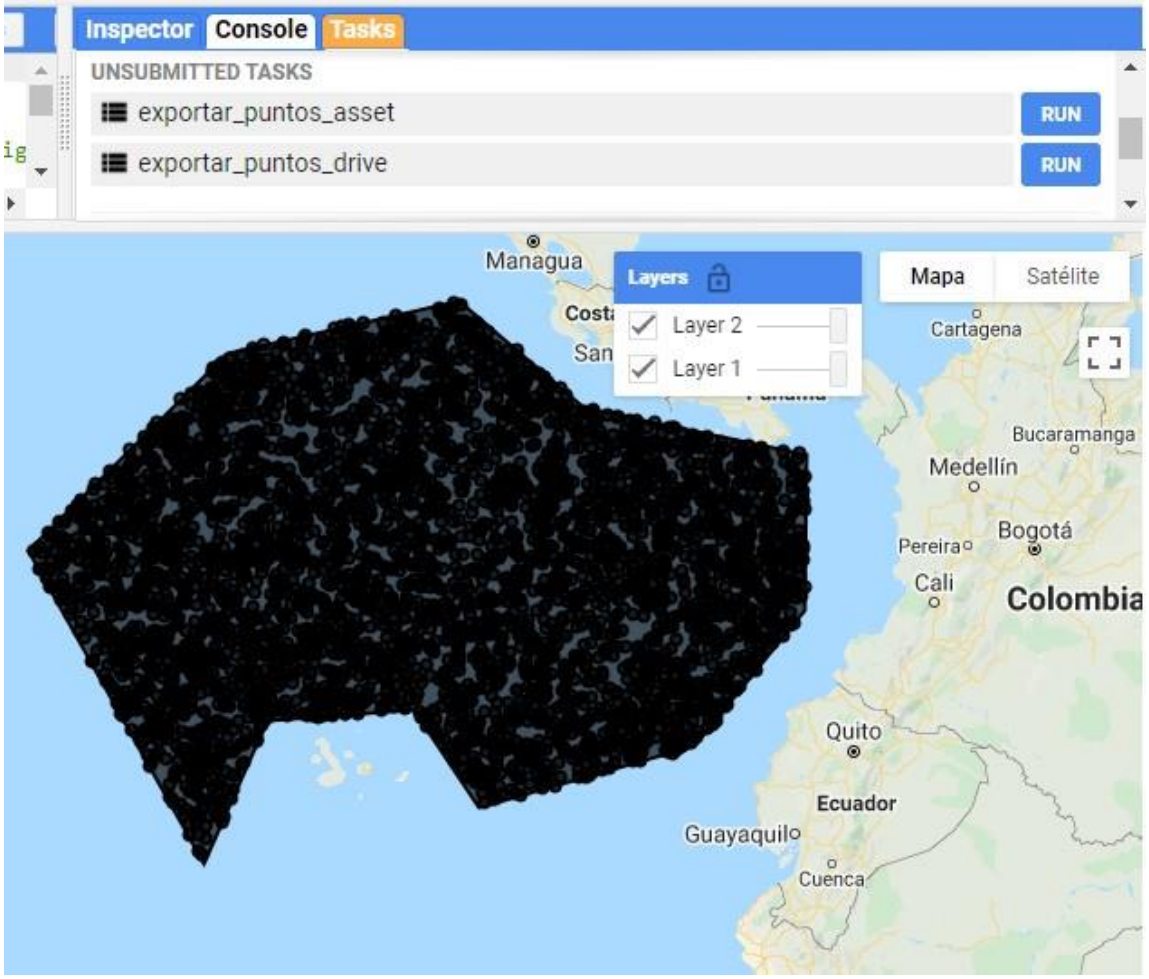


Figura 8.14: Visualización de la información a exportar, así como su salida en la pestaña de Tasks (tareas).

Es importante recordar que es necesario dar clic al botón de **Run** en la pestaña de **Tasks** (tareas) para que se lleve a cabo la exportación (Fig. 8.15).

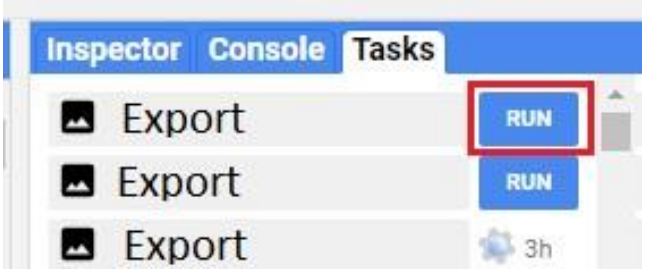


Figura 8.15: *Ubicación del botón Run para ejecutar una tarea de exportación.*

ee.Image

9

ee.Image

Las imágenes (`ee.Image`) en GEE generalmente se encuentran disponibles en colecciones de imágenes, aunque también existen fuentes de datos particulares que constan de una sola imagen. Las imágenes corresponden a información en formato ráster que cuenta con una resolución espacial, espectral y radiométrica particular (ver Olaya [2020] para una descripción más detallada de un ráster). Los metadatos de cada imagen en GEE se encuentran como propiedades de esta. A diferencia de las imágenes que posiblemente el usuario esté acostumbrado a manejar, GEE permite que cada banda tenga una resolución de píxel o tipo de datos distinta (**Integer**, **Float**, **16 bits**, etc.). Esto no afecta el uso de estas imágenes, pero es importante considerarlo si se van a exportar imágenes cuyas bandas tienen distinta resolución espacial (por ejemplo, Sentinel-2), ya que al exportar una imagen multibanda, todas sus bandas deben tener la misma resolución y tipo de datos. Al final de este capítulo se presenta un ejercicio integrador de varios de los procedimientos revisados aquí.

Para ejemplificar esta sección se cargará una imagen global de la altura del dosel para áreas con cobertura arbórea en el 2005 (Fig. 9.1).

EJERCICIO 27

```
// Cargar la imagen global de la altura del dosel para áreas con
// cobertura arbórea en el 2005
 var CHM = ee.Image('NASA/JPL/global_forest_canopy_height_2005');
```

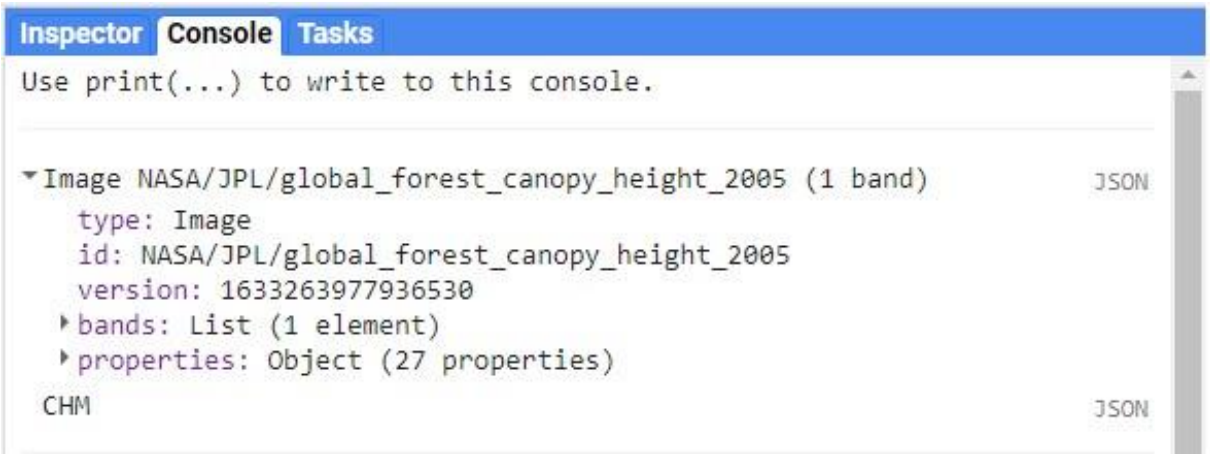


Figura 9.1: Ejemplo de la información desplegada en la consola al imprimir una imagen.

Para realizar los ejercicios de este apartado, se cargarán dos fuentes de información adicionales para aplicar algunos de los métodos disponibles para imágenes. La primera consta de una simple geometría de una región de interés, mientras que la segunda es un vector que contiene el polígono de la superficie de México.

```
// Cargar un polígono con el área de interés
 var roi1 = ee.Geometry.Polygon(
    [[[ -101.90832388457699, 20.455157626721274],
      [ -101.90832388457699, 19.422397223570666],
      [ -100.56799185332699, 19.422397223570666],
      [ -100.56799185332699, 20.455157626721274]]], null, false);

// Cargar la colección de atributos de los polígonos de la
// superficie de los países
var MX = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017')
    // Filtrar por la propiedad country_co para elegir México
    .filter(ee.Filter.eq('country_co','MX'));
```

Adicionalmente, se cargará la primera imagen de una colección de imágenes (Landsat 7). Este procedimiento se verá más a detalle en el siguiente capítulo sobre colecciones de imágenes. Por el momento, cabe aclarar que el método `.first` permite extraer la primera imagen de una colección de imágenes, tras lo cual el objeto resultante es una imagen (`ee.Image`). Ya que las imágenes multiespectrales solo se encuentran dentro de colecciones de imágenes, el uso de esta imagen de Landsat 7 nos permitirá ejemplificar algunos de los métodos más utilizados sobre este tipo de objetos.

```
// Obtener la primera imagen Landsat 7 de una colección de
// imágenes. Primero cargar la colección de imágenes de Landsat 7
var landsat7im = ee.ImageCollection('LANDSAT/LE07/C01/T1_SR')
    // Filtrar por fecha
    .filterDate('2020-01-01','2020-02-01')
    // Filtrar imágenes para mantener únicamente las que se
    // sobrepongan con el área de interés
    .filterBounds(roi1)
    // Obtener la primera imagen de la colección
    .first();
```

9.1 Información y metadatos

Para consultar las propiedades de una imagen (metadatos) se puede utilizar el siguiente comando: `.propertyNames` (Fig. 9.2). Otros comandos que facilitan la consulta de algunos elementos particulares son: `.bandNames`, `.projection`,

.projection().nominalScale, para conocer el nombre de las bandas de la imagen, su proyección y el tamaño de píxel en m que presenta, respectivamente.

| EJERCICIO 27.1

```
// Obtener las propiedades de la imagen
CHM.propertyNames();
```

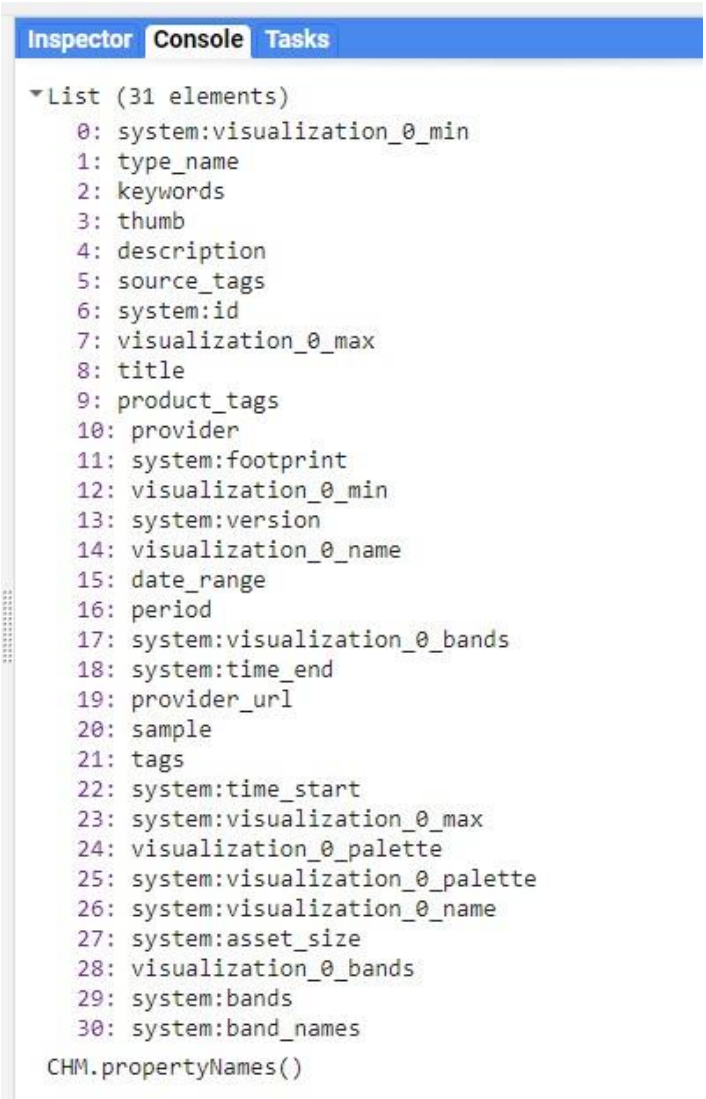


Figura 9.2: Propiedades de una imagen desplegadas en la consola.

El sistema de coordenadas de referencia de trabajo para todas las capas es EPSG:4326. Si se requiere cambiar de proyección se puede utilizar el método .reproject, aunque no se recomienda hacerlo, a menos que sea estrictamente necesario.

Por otro lado, la información almacenada en un ráster, usualmente está organizada en distintas bandas. Para consultar los nombres de las bandas de una imagen se utiliza el método `.bandNames` (Fig. 9.3).

| EJERCICIO 27.2

```
// Obtener los nombres de las bandas de la imagen
CHM.bandNames();
```

Figura 9.3: Nombre de las bandas de la imagen CHM.

9.2 Visualización de una imagen

De igual manera que los vectores y las colecciones de vectores, las imágenes se pueden agregar a la pantalla de mapa mediante la función `Map.addLayer` (Fig. 9.4).

| EJERCICIO 27.3

```
// Cargar la imagen a la pantalla de mapa
Map.addLayer(CHM,
    // Definir los valores mínimos y máximos de la
    // imagen para su visualización
    {min: 0, max: 32},
    // Definir el nombre con el que se desea cargar la imagen a la
    // pantalla de mapa
    'primera imagen');
```

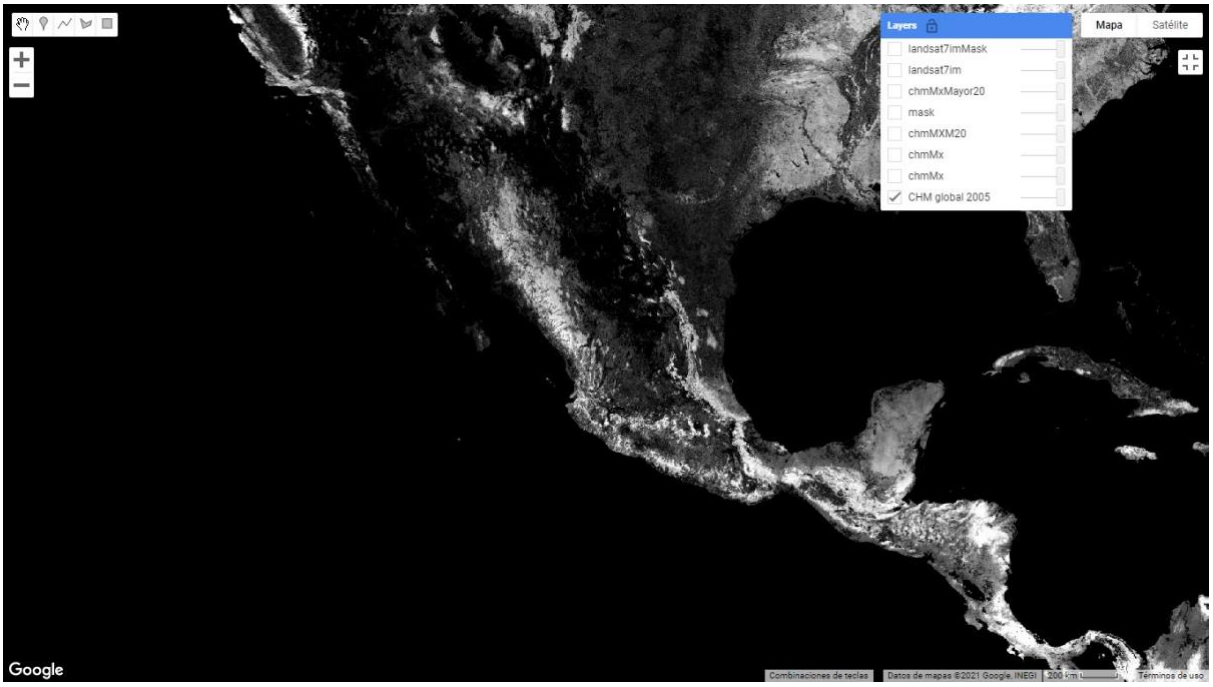


Figura 9.4: Visualización de la capa de la altura del dosel

La información ráster en GEE se almacena en un esquema piramidal, es decir, que existen varias copias de la imagen, en la que cada una corresponde a una escala distinta. Al visualizar una imagen en la pantalla de mapa, la imagen mostrada será la de la escala que se encuentre más cercana al nivel del zoom de la pantalla de mapa. Por ello, en algunas ocasiones al usar el Inspector y mover el zoom, el valor de un píxel puede cambiar, ya que se están consultando valores de distintas imágenes (cada una asociada a una escala distinta). Para mayores detalles visitar: https://developers.google.com/earth-engine/guides/scale

9.3 Métodos comunes

Selección de bandas

Una vez revisados los nombres de las bandas de una imagen, se pueden seleccionar algunas de ellas mediante el método `.select`. Para seleccionar una única banda solo se requiere indicar el nombre de la banda, mientras que si se desea seleccionar varias bandas, estas deben indicarse dentro de una lista. Por ejemplo (Fig. 9.5):

| EJERCICIO 27.4

```
// Seleccionar la banda 1 de la imagen (que es la única)
var banda1 = CHM.select('1');

// Seleccionar solo las primeras cuatro bandas de la imagen
var landsat7imBands = landsat7im.select(['B1','B2','B3','B4']);
```

Adición de nuevas propiedades o modificación de propiedades preexistentes

Usando el método .set se pueden modificar las propiedades preexistentes, o escribir nuevas propiedades sobre la imagen. En este caso, primero se indica el nombre de la clave, seguido de su valor correspondiente. Por ejemplo (Fig. 9.5):

| EJERCICIO 27.5

```
// Agregar una nueva propiedad
var CHMprop1 = CHM.set('miPropiedad','miImagen');

// Modificar una propiedad preexistente
var CHMprop2 = CHM.set('title','miTítulo');
```

Se puede sobrescribir una misma variable o una propiedad de un elemento, pero hay que tener cuidado porque la información anterior se pierde.

```
Inspector  Console  Tasks
▾Image NASA/JPL/global_forest_canopy_height_2005 (1 band)        JSON
    type: Image
    id: NASA/JPL/global_forest_canopy_height_2005
    version: 1633263977936530
  ▸bands: List (1 element)
  ▸properties: Object (27 properties)
 banda1                                                           JSON

▾Image LANDSAT/LE07/C01/T1_SR/LE07_027046_20200115 (11 ban…      JSON
    type: Image
    id: LANDSAT/LE07/C01/T1_SR/LE07_027046_20200115
    version: 1581545686430677
  ▸bands: List (11 elements)
  ▸properties: Object (23 properties)
 landsat7im                                                       JSON

▾Image LANDSAT/LE07/C01/T1_SR/LE07_027046_20200115 (4 band…      JSON
    type: Image
    id: LANDSAT/LE07/C01/T1_SR/LE07_027046_20200115
    version: 1581545686430677
  ▸bands: List (4 elements)
  ▸properties: Object (23 properties)
 landsat7imBands                                                  JSON

▾Image NASA/JPL/global_forest_canopy_height_2005 (1 band)        JSON
    type: Image
    id: NASA/JPL/global_forest_canopy_height_2005
    version: 1633263977936530
  ▸bands: List (1 element)
  ▸properties: Object (28 properties)
 CHMprop1                                                         JSON

▾Image NASA/JPL/global_forest_canopy_height_2005 (1 band)        JSON
    type: Image
    id: NASA/JPL/global_forest_canopy_height_2005
    version: 1633263977936530
  ▸bands: List (1 element)
  ▸properties: Object (27 properties)
 CHMprop2                                                         JSON
```

Figura 9.5: Salida de la consola tras haber seleccionado algunas bandas o agregar propiedades. Nótense los valores en las entradas de **bands** *y* **properties**.

Sustitución de nombres de bandas

Por defecto, el nombre de las bandas de las imágenes suele corresponder a la letra `'B'` más un número que indica el número de banda. Por ejemplo, `'B1'`, `'B2'`, `'B3'`, etc. En algunos casos, el usuario puede preferir sustituir estos nombres con letras que indiquen la longitud de onda que contiene cada banda. Para ello, se puede utilizar el método `.rename`. En caso de que la imagen de interés conste de una sola banda y un solo nombre, se puede indicar directamente el nuevo nombre, si no, se debe pasar una lista con el nuevo nombre para cada banda. Por ejemplo (Fig. 9.5):

| EJERCICIO 27.6

```
// Cambiar el nombre de la banda
banda1 = banda1.rename('chm');

var landsat7imBandsRename = landsat7imBands.rename(
    ['B','G','R','NIR']);
```

Realización de cortes

Para obtener el área de una imagen únicamente de una región de interés se puede utilizar el método `.clip`. Por ejemplo (Fig. 9.6):

| EJERCICIO 27.7

```
// Cortar la imagen a la extensión del polígono de México
var chmMx = banda1.clip(MX);
```

Esto permite trabajar únicamente con la región de interés (en el ejemplo anterior, especificada en el objeto roi).

> Se sugiere evitar el uso del método `.clip` a menos que sea estrictamente necesario, ya que aumenta el tiempo de procesamiento. En muchos casos se puede indicar el área de interés hasta el último paso del procesamiento, que suele implicar la exportación del resultado fuera de GEE (ver última sección de este capítulo: Exportación de una imagen).

Cálculo de operaciones matemáticas

Dentro de GEE se puede realizar cualquier operación matemática entre bandas o entre imágenes. Los operadores para realizar estas operaciones son: `.subtract`, `.add`, `.divide`, `.multiply` para restar, sumar, dividir y multiplicar, respectivamente (Fig. 9.6).

| EJERCICIO 27.8

```
// Multiplicar la imagen por 100 para obtener el valor en cm
var chmCmMx = chmMx.multiply(100);
```

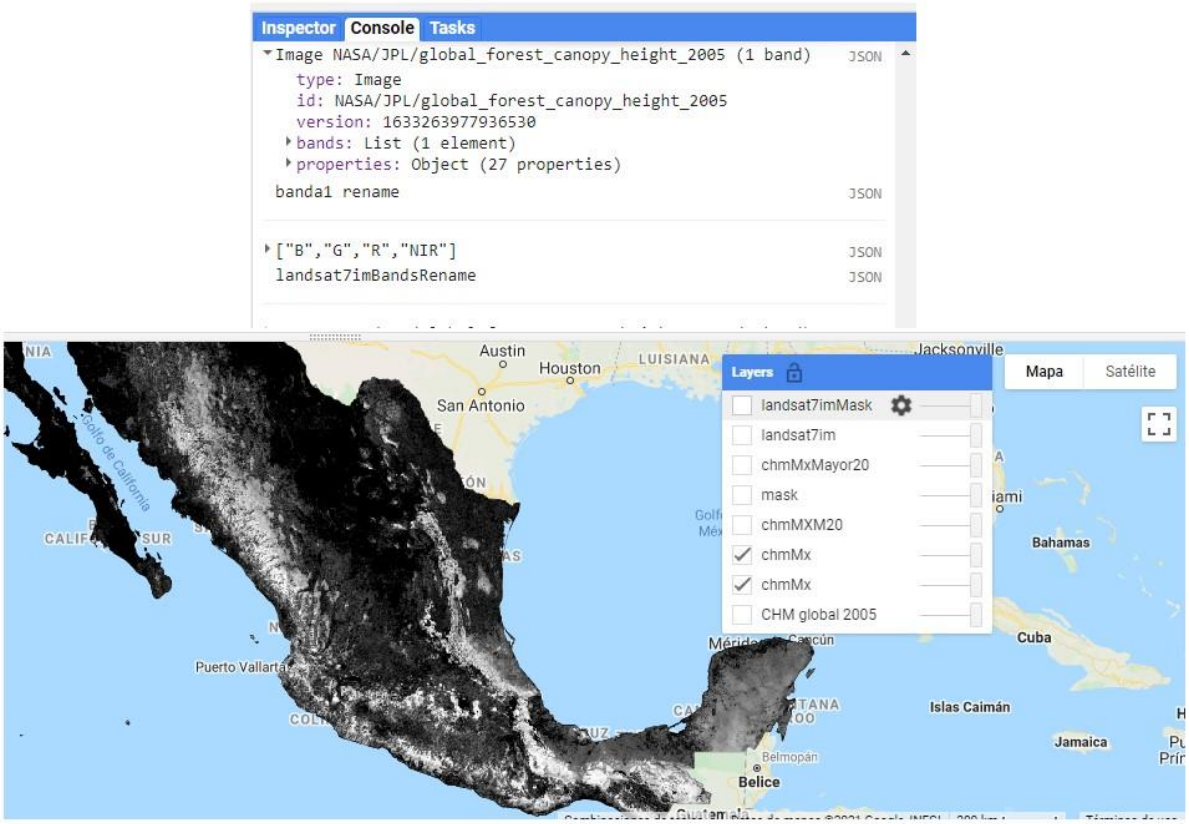


Figura 9.6: Visualización de la capa de altura del dosel en México.

Además, GEE cuenta con un método de `.normalizedDifference` para calcular un índice normalizado a partir de las bandas que se indiquen. Por ejemplo, para calcular el índice de vegetación de diferencia normalizada (**NDVI**, por sus siglas en inglés), 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 el método `.expression`. En este método se indica primero la fórmula del cálculo que se va a realizar (en formato de cadena de texto) 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. Las siguientes dos formas de calcular el NDVI obtienen el mismo resultado:

```
// Calcular el ndvi con la imagen de landsat 7 obtenida
// previamente
var ndvi= landsat7imBands.expression('(NIR - R) / (NIR + R)', {
    'NIR' = landsat7imBands.select('B4'),
    'R' = landsat7imBands.select('B3')
    })
    // Renombrar la banda como 'ndvi'
    .rename('ndvi');

var ndvi2 = landsat7imBands.normalizedDifference('B4','B3')
    // Renombrar la banda como 'ndvi'
    .rename('ndvi');
```

> Si no se cambia el nombre de la banda producida por `.normalizedDifference`, por defecto, esta va a recibir el nombre de `'nd'`.

Para las fórmulas utilizadas dentro de `.expression` se pueden utilizar operadores como la notación estándar matemática (+, -, /, *), módulo (%), exponente (**), relacionales (>, <, >=, <=, !=, ==), lógicos (&&, ||, !, ^) u operadores ternarios ***if then else*** (?:).

Uso de operaciones relacionales, condicionales y booleanas

Se pueden utilizar operadores relacionales, condicionales o booleanos sobre las imágenes para crear clases o máscaras. Para ello, se utilizan las siguientes notaciones: `.lt`, `.lte`, `.gt`, `.gte`, `.eq`, `.neq`, que equivalen a menor que (*less than*), menor que o igual (*less than or equal*), mayor que (*greater than*), mayor que o igual (*greater than or equal*), igual (*equal*) o no igual (*not equal*), respectivamente. Además, se pueden utilizar operadores relacionales para unir dos condiciones mediante `.and` u `.or`, es decir, y u o. Por ejemplo, la siguiente línea permite crear una máscara de las áreas que tengan valores menores o iguales a `0.2` y mayores o iguales a `0` (Fig. 9.7):

| EJERCICIO 27.9

```
// Crear una imagen de los píxeles que cumplen con la condición de
// tener valores mayores o iguales a 20
// Nótese que se utiliza la imagen con valores en m
var chmMXMask20 = chmMx.gte(20);
```

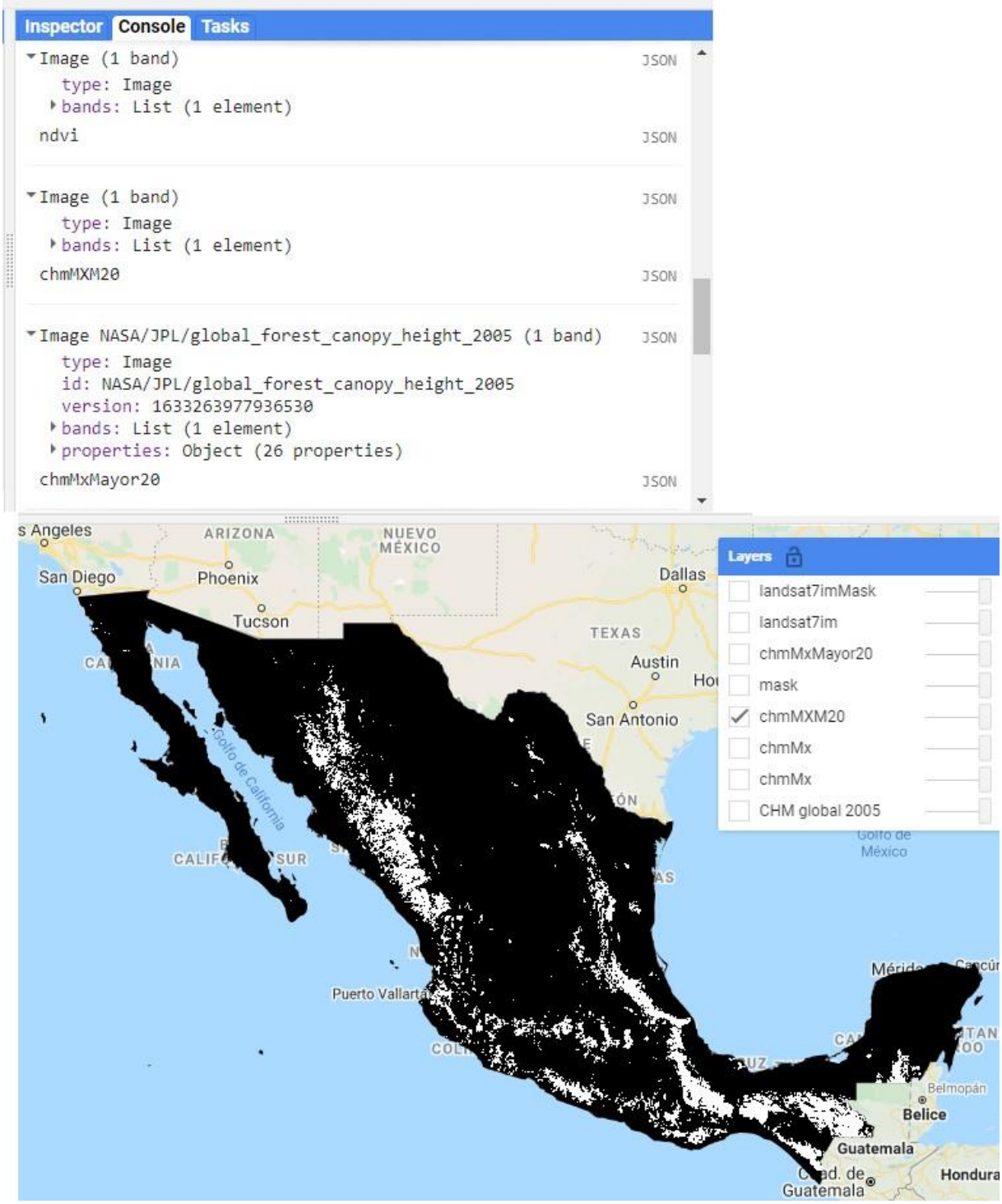


Figura 9.7: Resultado de la máscara booleana de altura del dosel.

Otra forma de crear máscaras es indicando valores por bits. En este caso, se indica el valor del bit de interés y después se evalúa una condición sobre una banda determinada. Por ejemplo, para Landsat 8, la banda de evaluación de la calidad de la

imagen corresponde a la banda `'pixel_qa'`. En esta banda, el **bit 5** indica una cobertura de nubes, así que primero se define una variable que indique que el **bit 5** es el que será evaluado. Después se selecciona la banda `'pixel_qa'` para evaluar qué píxeles tienen un valor de `0` en el **bit 5** (valores a conservar) y cuáles no (valores a eliminar; Fig. 9.11).

| EJERCICIO 27.10

```
// Definir la regla en bits de la imagen de los valores que se
// desea enmascarar
var nubesBit = (1 << 5);

// Seleccionar la banda de evaluación de la calidad de la imagen
var qa = landsat7im.select('pixel_qa');

// Definir una máscara donde las áreas a mantener (1) serán las
// áreas donde el bit 5 tenga valores iguales a cero
var mask = qa.bitwiseAnd(nubesBit).eq(0);
```

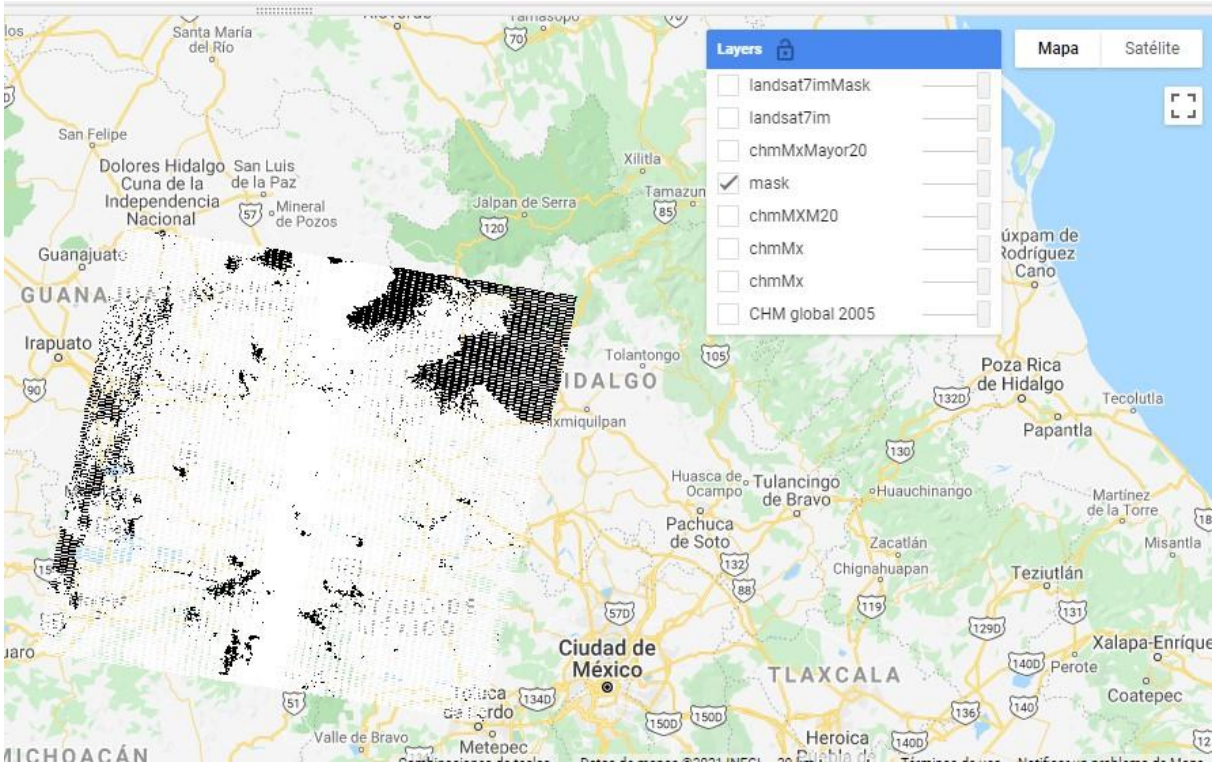


Figura 9.8: Visualización de la máscara de Landsat 8 indicando píxeles con nubes y sin nubes.

Enmascaramiento

El método .updateMask permite enmascarar los píxeles de una imagen, utilizando otra imagen como máscara que indique cuáles píxeles se deben conservar (valores de 1) y cuáles enmascarar (valores de 0). En la figura 9.9 se puede apreciar la visualización del enmascaramiento realizado a la capa que contiene la altura del dosel. Por su parte, las figuras 9.10 y 9.11 muestran la imagen Landsat 8 antes y después de realizar el enmascaramiento de nubes, respectivamente.

| EJERCICIO 27.11

```
// Utilizar la máscara de valores que tienen alturas >= 20 y
// aplicarla a la imagen con los valores de alturas
var chmMxMayor20 = chmMx.updateMask(chmMXMask20);

// Aplicar la máscara a la imagen original
var landsat7imMask = landsat7im.updateMask(mask);
```

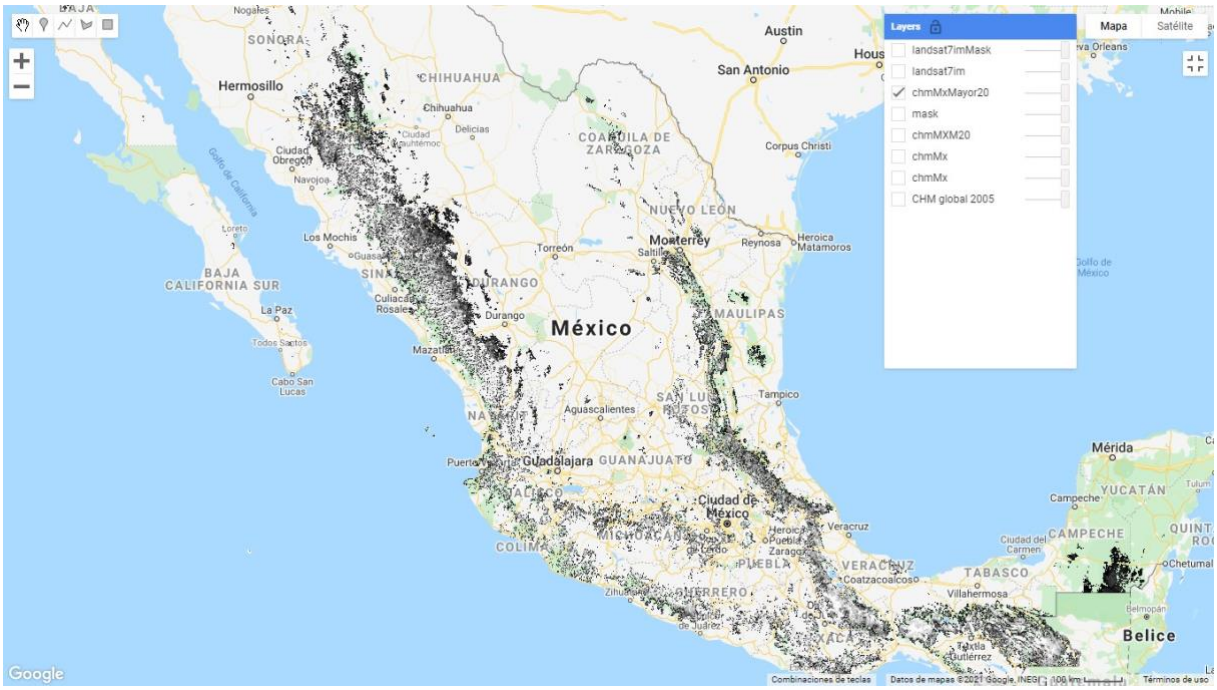


Figura 9.9: Resultado del enmascaramiento de la capa de altura del dosel.

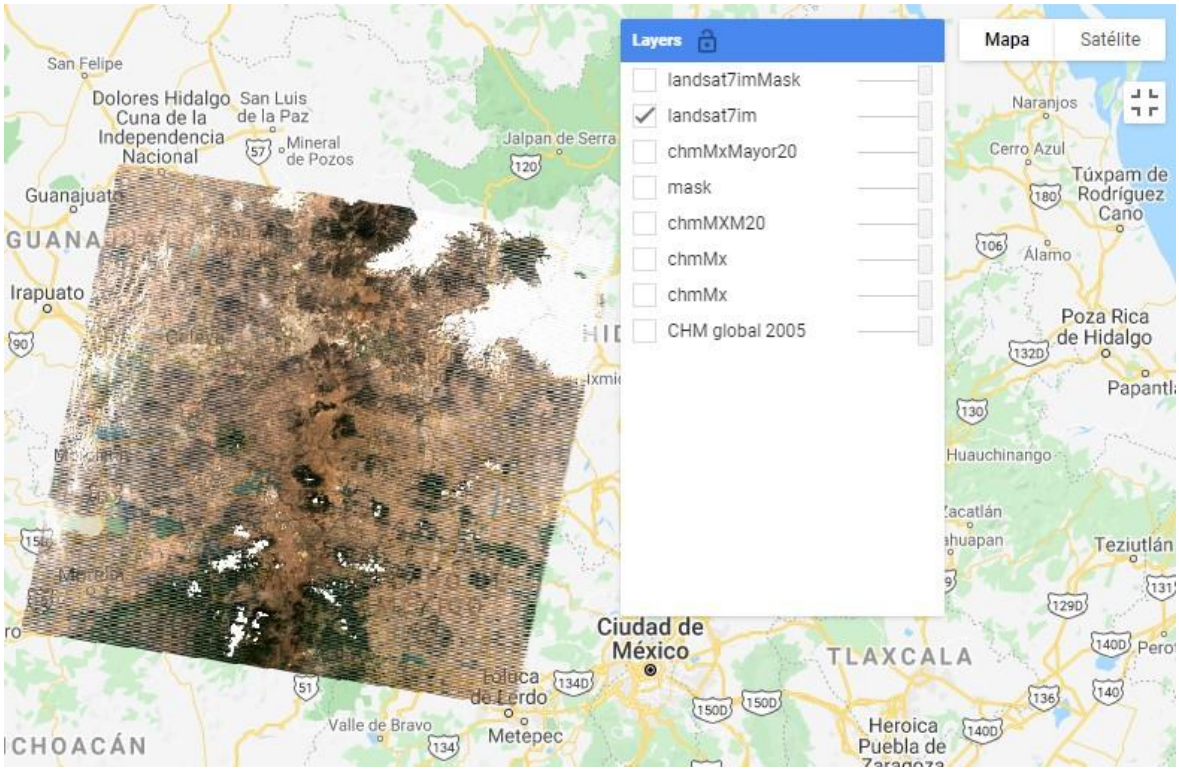


Figura 9.10: Imagen Landsat antes del enmascaramiento de nubes.

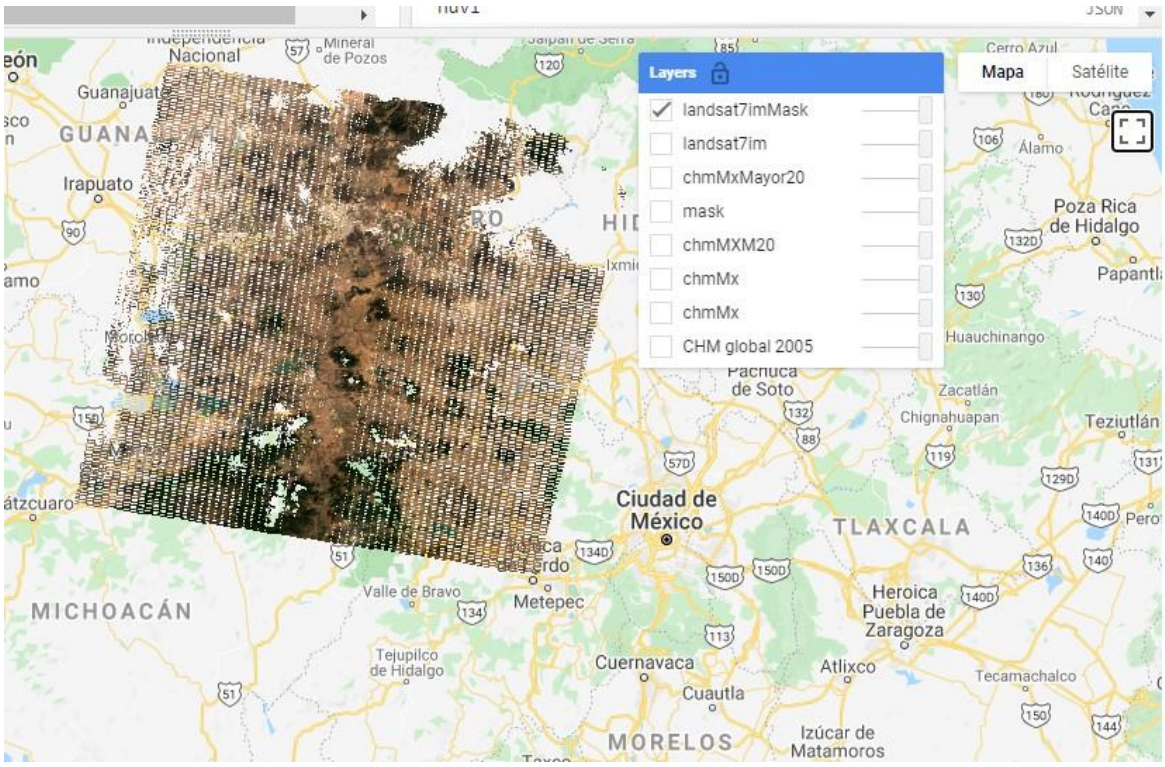


Figura 9.11: Imagen Landsat después del enmascaramiento de nubes. Nótese que las áreas enmascaradas ahora son transparentes.

Aunque en GEE existe el método .mask para enmascarar áreas de una imagen, se sugiere siempre utilizar .updateMask. El método .mask simplemente sustituye la máscara original de la imagen, mientras que .updateMask combina la máscara anterior con la nueva, lo que evita desenmascarar áreas que seguramente no son de interés para el usuario.

Adición de bandas

El usuario puede agregar una banda al conjunto de bandas que contiene por defecto una imagen utilizando el método .addBands. Así, se podría agregar una banda que contenga algún índice calculado con las bandas originales de la imagen. Por ejemplo:

| EJERCICIO 27.12

```
// Agregar la banda de alturas en cm a la banda original en m
chmMx2Bands = chmMx.addBands(chmCmMx.rename('chmCm'));

// Agregar la banda de ndvi a la imagen original
landsat7im = landsat7im.addBands(ndvi);
```

Conversión de una imagen en un vector

En algunos casos puede ser conveniente transformar una imagen en un vector, o parte de una imagen en un vector. El método para convertir una imagen en vector es .reduceToVectors y se utiliza de la siguiente manera (Fig. 9.12):

| EJERCICIO 27.13

```
// Convertir la máscara de valores que tienen valores de altura
// >= 20 a vector
var chmMxMayor20Vec = chmMXMask20.reduceToVectors({
    // Usar el polígono de México para definir el área a convertir
    // a vector
    geometry: MX,
    // Definir el tipo de geometría que se desea
    geometryType: 'polygon',
    // Definir el tamaño de píxel en m con el cual se va a crear
    // el vector
    scale: 1200,
    // Definir el tamaño máximo de píxeles a procesar
    maxPixels: 1e11
    });
```

Figura 9.12: Visualización de la capa vector producida a partir de una imagen.

En operaciones como la conversión de ráster a vector, donde se debe indicar el tamaño del píxel para realizar el proceso, el argumento de scale le indica a GEE la escala de la representación piramidal de una imagen de la cual va a tomar los valores para realizar la operación. Ver sección 9.3.

Exportación de una imagen

En GEE se pueden exportar imágenes a los **Assets**, al **Google Drive** o al **GoogleCloud**. Debido a que el servicio más común será el de **Drive**, el cual este es el que se explicará. La función para realizar dicha exportación es `Export.imageToDrive` (ver Sección 3.4). Los argumentos de esta función son: **image** (la imagen a exportar), **description** (el nombre con el que se desea guardar el archivo), **folder** (la carpeta donde se desea guardar la imagen), **scale** (la escala para exportar la imagen), **maxPixels** (el número máximo de píxeles permitido), **format** (el formato de imagen deseado), **region** (región que se desea exportar, suele corresponder al área de interés) y **crs** (sistema de coordenadas de referencia). La función es capaz de aceptar otros argumentos que se pueden consultar en la sección de **Reference** o en la pestaña de **Docs**, así como los

valores por defecto de cada uno de estos parámetros. Cabe mencionar que todos los argumentos son opcionales, excepto la imagen a exportar, sin embargo, se recomienda siempre definir por lo menos los argumentos de descripción, carpeta, escala y región. En el caso de la exportación a **Assets** el argumento más importante es **assetId**, que corresponde al nombre con el que se guardará el archivo en la sección de **Assets**.

| EJERCICIO 27.14

```
// Exportación al drive personal
Export.image.toDrive({
    // Nombre de la imagen a exportar en GEE
    image:chmMx,
    // Nombre con el que se va a guardar la imagen en el drive
    description:'CanopyHeightModelMX',
    // Carpeta dentro del drive en el cual se exportará la imagen
    folder:'carpeta_clase_GEE',
    // Tamaño en m del píxel de la imagen a exportar
     scale: 30,
    // Tamaño en m del píxel de la imagen a exportar
     maxPixels:1e12,
    // Formato en el cual se va a exportar la imagen
    fileFormat: 'GeoTIFF',
    // Región de la imagen a exportar
    region: roi1,
    // Definir sistema de coordenadas de referencia
    crs: 'EPSG:4326'});
```

En la experiencia de los autores, el valor máximo de píxeles que permite exportar GEE es 1e13, es decir, un uno seguido de trece ceros.

El sistema de coordenadas de referencia (CRS, por sus siglas en inglés) usado en el ejemplo anterior se definió de acuerdo a un código EPSG. Estos pueden consultarse en el siguiente enlace: https://epsg.io/. Por defecto, el sistema de coordenadas de referencia de la información almacenada en GEE corresponde a coordenadas geográficas WGS84 (EPSG:4326) con un tamaño de píxel de un grado. Recuerde que para mostrar la información en la pantalla de mapa, esta se proyecta a un CRS con la clave EPSG:3857, sin embargo, dicha proyección únicamente se hace para desplegar la información, mas no transforma la información con la que el usuario está trabajando. Por lo tanto, puede haber pequeñas diferencias entre la información visualizada en la pantalla de mapa y la información exportada.

Es importante recordar que es necesario dar clic a las tareas en la pestaña **Tasks** para que se lleve a cabo la exportación.

| EJERCICIO A: CONSULTA Y PROCESAMIENTO DE UNA IMAGEN

En GEE hay algunos objetos que constan de imágenes únicas, por lo tanto, se pueden utilizar directamente sobre ellas los métodos para el procesamiento de imágenes. Un ejemplo muy utilizado de este tipo de objetos es el modelo digital de elevación (**DEM**) producido por la misión SRTM (*Shuttle Radar Topography Mission*). Para buscar cuál es la localización para acceder a esta información se puede ir a la barra de búsqueda (**Search**) y escribir SRTM. Debería aparecer una ventana así (Fig. 9.13):

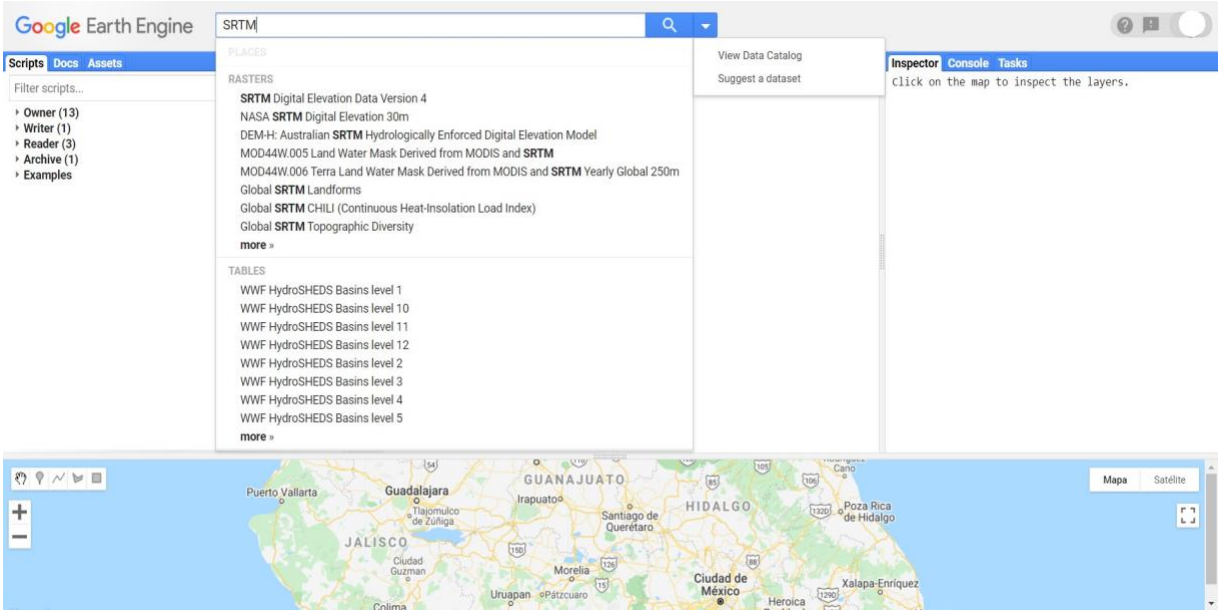


Figura 9.13: Ejemplo de búsqueda de información en la barra de búsqueda.

Posteriormente se da clic sobre el primer resultado y debería abrirse la siguiente ventana (Fig. 9.14).

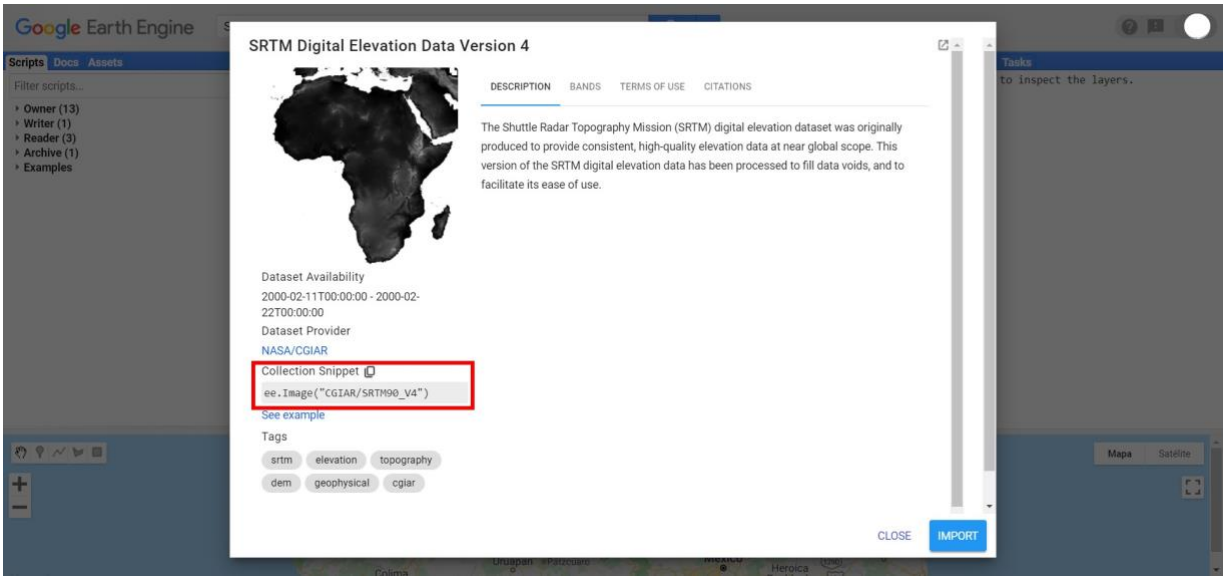


Figura 9.14: Consulta de información de una imagen del repositorio de GEE.

Aquí, se puede copiar la ubicación de la imagen del **DEM** de SRTM en los servidores de Google. Posteriormente, en la pantalla de rutinas se puede escribir la siguiente línea para guardar la imagen SRTM en un objeto nuevo llamado dem.

```
// Cargar la imagen con el DEM de SRTM
var dem = ee.Image('CGIAR/SRTM90_V4');
```

Posteriormente, se corta el DEM a la extensión que nos interesa. En este caso se utilizará la colección con los límites políticos del mundo que se encuentra cargada en GEE. Esta se puede buscar de igual manera que la capa de SRTM, en la barra de búsqueda. Posteriormente, se puede consultar la información que contiene la capa en la pestaña de **TABLE SCHEMA**. Por último, se aplica un filtro para seleccionar el país que nos interesa, en este caso México.

```
// Cargar la colección de atributos con polígonos de la superficie
// de los países
var MX = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017')
    // Filtrar para seleccionar únicamente México
    .filter(ee.Filter.eq('country_co','MX'));
```

Y se corta la imagen a la extensión del polígono de interés.

```
// Cortar el DEM con el polígono de México
var demCortado = dem.clip(MX);
```

Para comparar el DEM antes y después de cortar la imagen a la extensión del polígono de interés, se cargan los dos objetos: dem y demCortado.

```
// Cargar el DEM global y el cortado (solo MX), indicando valores
// máximos y mínimos en m
Map.addLayer(dem, {min: 0, max:4000}, 'dem');
Map.addLayer(demCortado, {min: 0, max:4000}, 'demMX');
```

Posteriormente, se define una máscara de todas las áreas que tengan una elevación menor o igual a `2500` m s.n.m. y mayor o igual a `1000` m s.n.m. Después, se aplica esta máscara al **DEM** cortado para mantener únicamente las áreas que cumplan con el criterio anterior.

```
// Crear máscara de zona de interés
var mascara = demCortado.lte(2500).and(demCortado.gte(1000));

// Aplicar máscara al DEM de México
var imgEnmascarada = demCortado.updateMask(mascara);
```

A continuación, se renombrará la banda de la imagen de `'elevation'` a `'elevacion'` y se agrega la imagen a la pantalla de mapa.

```
// Cambiar el nombre de la banda de la imagen enmascarada a
// elevación
imgEnmascarada = imgEnmascarada.rename('elevacion');

// Agregar imagen enmascarada a la pantalla de mapa
Map.addLayer(imgEnmascarada,{min: 1000, max: 2500},
'imgEnmascarada');
```

Figura 9.15: Visualización de las tres capas generadas: DEM sin corte, DEM con corte, DEM con corte y enmascarada.

A continuación, se generará un gráfico para obtener un histograma de los valores de elevación del DEM. Para hacer esto, se utiliza la función `ui.Chart.image.histogram` y se indica la imagen de la cual se quiere obtener el

histograma (**image**), la escala en metros a la que se quiere obtener los valores del histograma (**scale**) y el número máximo de clases (**maxBuckets**). Después, se agrega este gráfico a la consola con un `print` (Fig. 9.16).

```
// Crear gráfico de histograma de los valores del DEM enmascarado
var graficoHistograma = ui.Chart.image.histogram({
    // Indicar imagen de la cual se va a crear el histograma
    image: imgEnmascarada,
    // Indicar la escala en m a la que se va a muestrear la
    // información para obtener el histograma
    scale: 9000,
    // Indicar el número máximo de categorías (en eje x) para crear
    // el histograma
    maxBuckets: 10 });

    print(graficoHistograma,'gráfico Histograma');
```

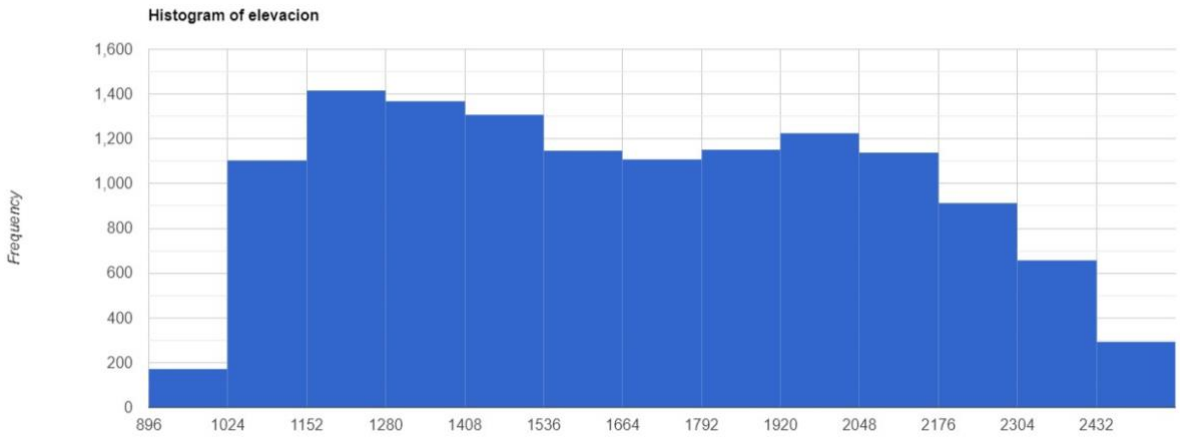


Figura 9.16: Histograma de valores del DEM.

Por último, se puede exportar el resultado al **Google Drive** del usuario. En este caso se definirá un valor de **scale** de `9000` para acelerar el tiempo que requiere la exportación.

```
Export.image.toDrive({
    image:imgEnmascarada,
    description:'MX_DEM1000-2500msnm',
    folder:'carpeta_clase_GEE',
    scale: 9000,
    maxPixels:1e12,
    fileFormat: 'GeoTIFF',
    region: MX,
    crs: 'EPSG:4326'
    });
```

Una vez corrido el código, hay que irse a la pestaña de **Tasks** para darle clic en **Run** (correr) y comenzar a exportar la imagen al **Google Drive** (Fig. 9.17). A continuación aparece una pantalla, en la cual podemos confirmar los parámetros utilizados para la exportación de la imagen, tras lo cual, se vuelve a dar clic en **Run**. Por último, el producto exportado aparecerá en la pestaña de **Tasks** con un símbolo de un engranaje girando, el cual indica que el proceso está corriendo. Además, en esta pestaña se puede consultar el tiempo de demora del trabajo de exportación. Ya que la imagen haya sido exportada al **Drive**, la tarea cambiará a color azul y el engranaje en movimiento cambiará a una palomita (chulo) indicando el tiempo demorado en exportar la imagen al **Drive** (ver Fig. 2.22).

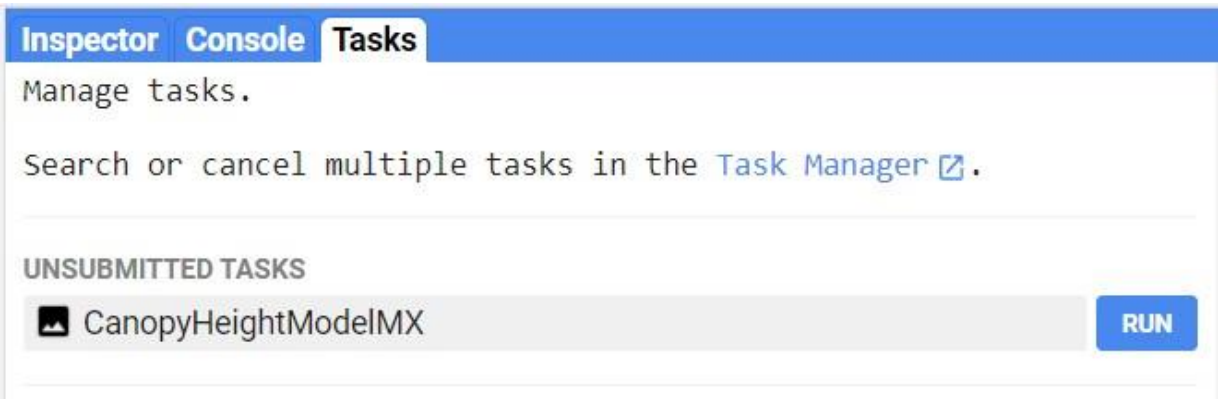


*Figura 9.17: Vista de la pestaña de **Tasks** (tareas) para correr el trabajo de exportación.*

ee.ImageCollection

10

ee.ImageCollection

Las colecciones de imágenes (`ee.ImageCollection`) son objetos de GEE que contienen un conjunto de imágenes. La mayoría de los acervos de imágenes disponibles en GEE van a ser definidos como este tipo de objetos. Para el manejo de varias imágenes se recomienda utilizar esta estructura, en lugar de listas u otro tipo de objetos. La mayoría de las colecciones de imágenes disponibles en GEE incluyen acervos de imágenes gratuitas como Landsat 1-5 y 7-9, MODIS, Sentinel 1-3 y 5, ASTER, entre otras (Gorelick *et al.*, 2017; Radočaj *et al.*, 2020). Al final de este capítulo se presentan ejercicios integradores de varios de los procedimientos revisados en este capítulo.

Por ejemplo, para cargar la colección de reflectancia de la superficie de imágenes Sentinel-2 se usaría el siguiente código:

EJERCICIO 28

```
// Cargar la colección de Sentinel-2 reflectancia de la
// superficie
var S2 = ee.ImageCollection('COPERNICUS/S2_SR');
```

Como información adicional para este capítulo se definirá una geometría de un área de interés.

```
// Definición de un área de interés
var roi = ee.Geometry.Polygon(
    [[[ -101.80582759131487, 20.274503120947937],
      [ -101.80582759131487, 19.35462236718103],
      [ -100.61930415381487, 19.35462236718103],
      [ -100.61930415381487, 20.274503120947937]]], null, false);
```

10.1 Información y metadatos

Las colecciones de imágenes contienen los metadatos e información de todas las imágenes que incluyen, lo cual permite filtrar las imágenes y utilizar únicamente las que cumplen con ciertos criterios. Es importante recalcar que los metadatos y propiedades de una colección de imágenes no corresponden a las propiedades de las imágenes que la componen. Para consultar las propiedades de una colección de imágenes se usa el método `.propertyNames` (Fig. 10.1).

| EJERCICIO 28.1

```
// Obtener las propiedades de la colección de imágenes
var S2Prop = S2.propertyNames();
```

```
List (23 elements)                                    JSON
  0: date_range
  1: period
  2: system:visualization_0_min
  3: type_name
  4: keywords
  5: system:visualization_0_bands
  6: thumb
  7: description
  8: source_tags
  9: system:id
  10: visualization_0_max
  11: provider_url
  12: title
  13: sample
  14: tags
  15: system:visualization_0_max
  16: product_tags
  17: provider
  18: visualization_0_min
  19: system:version
  20: system:visualization_0_name
  21: visualization_0_name
  22: visualization_0_bands
S2Prop                                                JSON
```

Figura 10.1: Salida de la consola indicando las propiedades de la colección de imágenes de Sentinel-2 2A.

> No se puede escribir o cambiar los objetos que se encuentran directamente hospedados en GEE. Por ejemplo, si se quiere mapear (`.map`) una función que sobrescriba alguna característica de las colecciones de imágenes en GEE, se debe hacer una copia de estas para que se pueda sobrescribir la propiedad de interés.

10.2 Creación de colecciones de imágenes

En ciertas ocasiones, el usuario puede estar interesado en crear nuevas colecciones de imágenes a partir de imágenes generadas dentro del mismo GEE. En este caso, se

utiliza el método `ee.ImageCollection.fromImages`, indicando como argumentos la lista de las imágenes a partir de la que se quiere crear dicha colección (Fig. 10.2).

| EJERCICIO 28.2

```
// Filtrar colección de Sentinel-2 por fechas
var im1 = S2.filterDate('2019-01-01','2019-05-01')
    // Aplicar filtro espacial
    .filterBounds(roi)
    // Obtener la primera imagen
    .first();

// Crear otra imagen de igual manera que la imagen anterior, pero
// con fechas distintas
var im2 = S2.filterDate('2020-01-01','2020-05-01')
    .filterBounds(roi)
    .first();

// Crear colección a partir de dos imágenes
var imColl = ee.ImageCollection.fromImages([im1, im2]);
```

```
▾Image COPERNICUS/S2_SR/20190105T171709_20190105T172443_T13QHB (23 bands)        JSON
    type: Image
    id: COPERNICUS/S2_SR/20190105T171709_20190105T172443_T13QHB
    version: 1556514341039576
  ▸bands: List (23 elements)
  ▸properties: Object (81 properties)
 im1                                                                              JSON

▾Image COPERNICUS/S2_SR/20200105T171711_20200105T171930_T13QHB (23 bands)        JSON
    type: Image
    id: COPERNICUS/S2_SR/20200105T171711_20200105T171930_T13QHB
    version: 1578382574569888
  ▸bands: List (23 elements)
  ▸properties: Object (81 properties)
 im2                                                                              JSON

▾ImageCollection (2 elements)                                                     JSON
    type: ImageCollection
    bands: []
  ▾features: List (2 elements)
    ▸0: Image COPERNICUS/S2_SR/20190105T171709_20190105T172443_T13QHB (23 bands)
    ▸1: Image COPERNICUS/S2_SR/20200105T171711_20200105T171930_T13QHB (23 bands)
```

Figura 10.2: Salida de la consola indicando el contenido de las dos imágenes obtenidas mediante un filtrado temporal distinto, así como la combinación de ambas en una colección de imágenes.

10.3 Visualización de colecciones de imágenes

Así como ocurre con los vectores y las colecciones de vectores, las colecciones de imágenes se pueden visualizar en la pantalla de mapa mediante la función `Map.addLayer`. Usualmente, al agregar una colección de imágenes a la pantalla de mapa se van a mostrar los elementos con fecha de registro más reciente. Por ejemplo, en este caso muestra la imagen más reciente por cada escena dentro de la colección (Fig. 10.3).

| EJERCICIO 28.3

```
// Filtrar colección de Sentinel-2 utilizando fechas
var imCol3 = S2.filterDate('2019-01-01','2019-05-01')
    // Aplicar filtro espacial
    .filterBounds(roi);

// Agregar colección de imágenes a la pantalla de mapa
Map.addLayer(imCol3, {bands:['B4','B3','B2'], min:200, max: 2000},
    'S2 Mich');
```

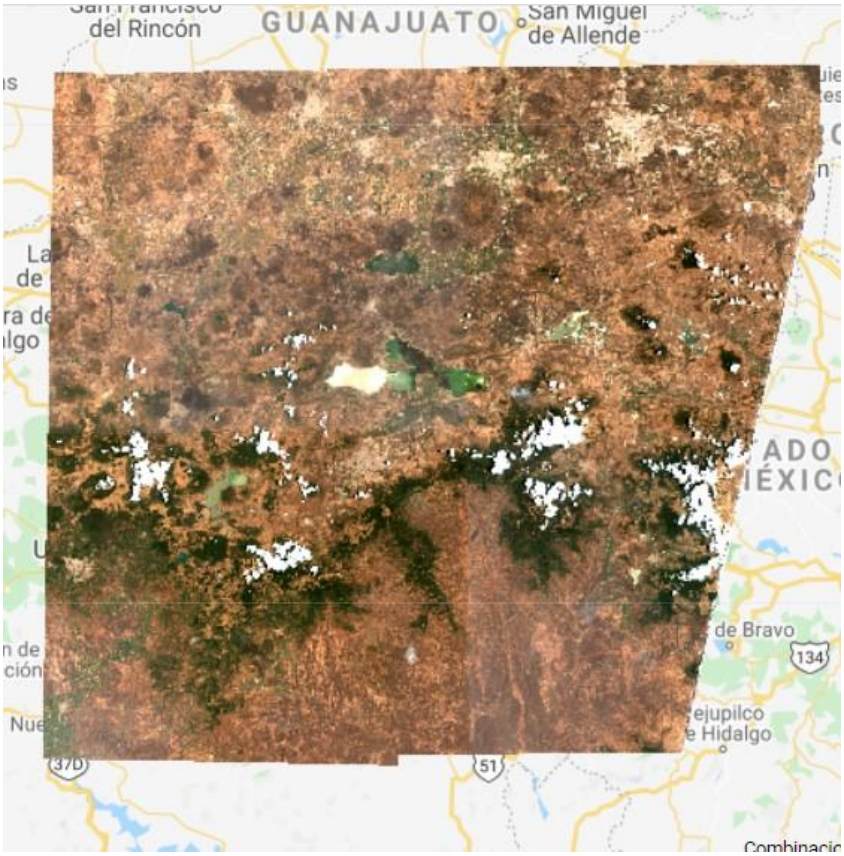


Figura 10.3: Visualización de la colección de imágenes de Sentinel-2 2A.

10.4 Métodos comunes

Filtración de colecciones de imágenes

Las colecciones de imágenes se pueden filtrar a partir de los metadatos de las imágenes que contienen. De tal manera, se puede filtrar por: superposición con algún polígono de interés, fecha de registro, metadatos 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 puede utilizar `.filterDate` (Fig. 10.4):

| EJERCICIO 28.4

```
// Filtrar colección de imágenes por fecha
var imCol4 = S2.filterDate('2019-01-01','2019-05-01');
```

Por otro lado, mediante `.filterBounds` se puede filtrar espacialmente con un polígono:

```
// Filtrar colección de acuerdo a sobreponga con polígono de
// región de interés
imCol4 = imCol4.filterBounds(roi);
```

Si se desea filtrar por alguna característica de los metadatos de las imágenes se puede hacer mediante `.filter` y después definiendo el filtro del metadato con `ee.Filter`. Otra forma de realizar este proceso es mediante `.filterMetadata`. Las siguientes dos líneas son equivalentes ya que obtienen el mismo resultado.

```
// Obtener el nombre de las propiedades de una imagen de la
// colección de imágenes
var imProperties = im1.propertyNames();

// Filtrar a partir de las propiedades de las imágenes
imCol4 = imCol4.filter(ee.Filter.lt('CLOUD_COVERAGE_ASSESSMENT',
    50));
imCol4 = imCol4.filterMetadata('CLOUD_COVERAGE_ASSESSMENT',
    'less_than', 50);
```

Recuerde que para filtrar una colección de imágenes utilizando metadatos se debe filtrar de acuerdo a las propiedades de las imágenes, no de la colección.

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`).

Para combinar varios filtros se utiliza `ee.Filter.and` o `ee.Filter.or` dependiendo de si se desea que se cumplan todas las condiciones indicadas (`and`) o alguna de ellas (`or`).

```
▾ ImageCollection COPERNICUS/S2_SR (148 elements)                    JSON
    type: ImageCollection
    id: COPERNICUS/S2_SR
    version: 1633349976941267
    bands: []
  ▸ features: List (148 elements)
  ▸ properties: Object (21 properties)
  filtro espacial y temporal                                         JSON

▸ List (85 elements)                                                 JSON
  imProperties                                                       JSON

▾ ImageCollection COPERNICUS/S2_SR (134 elements)                    JSON
    type: ImageCollection
    id: COPERNICUS/S2_SR
    version: 1633349976941267
    bands: []
  ▸ features: List (134 elements)
  ▸ properties: Object (21 properties)
  im4 filtros op1                                                    JSON

▾ ImageCollection COPERNICUS/S2_SR (134 elements)                    JSON
    type: ImageCollection
    id: COPERNICUS/S2_SR
    version: 1633349976941267
    bands: []
  ▸ features: List (134 elements)
  ▸ properties: Object (21 properties)
  im4 filtros op2                                                    JSON
```

Figura 10.4: *Salida de la consola indicando los resultados de la colección de imágenes después de realizar diferentes tipos de filtros.*

Selección de bandas

Al igual que con las imágenes, se puede utilizar el método `.select` para seleccionar un conjunto de bandas de todas las imágenes que se encuentran en la colección. De igual manera que como ocurre con las imágenes individuales, para seleccionar una única banda solo se requiere indicar el nombre de esta, mientras que si se desea seleccionar varias bandas, estas deben indicarse dentro de una lista. Por ejemplo (Fig.10.5):

| EJERCICIO 28.5

```
// Obtener el nombre de las bandas de una imagen de una colección
// de imágenes
var imBandNames = im1.bandNames();

// Seleccionar ciertas bandas de todas las imágenes que conforman
// la colección
var imCol5 = imCol4.select(['B4','B3','B2']);
```

```
▾List (23 elements)                                                    JSON
    0: B1
    1: B2
    2: B3
    3: B4
    4: B5
    5: B6
    6: B7
    7: B8
    8: B8A
    9: B9
    10: B11
    11: B12
    12: AOT
    13: WVP
    14: SCL
    15: TCI_R
    16: TCI_G
    17: TCI_B
    18: MSK_CLDPRB
    19: MSK_SNWPRB
    20: QA10
    21: QA20
    22: QA60
  imBandNames                                                          JSON

▾ImageCollection COPERNICUS/S2_SR (134 elements)                       JSON
    type: ImageCollection
    id: COPERNICUS/S2_SR
    version: 1633349976941267
    bands: []
  ▾features: List (134 elements)
    ▾0: Image COPERNICUS/S2_SR/20190105T171709_20190105T172443_T13QHB (3 bands)
        type: Image
        id: COPERNICUS/S2_SR/20190105T171709_20190105T172443_T13QHB
        version: 1556514341039576
      ▾bands: List (3 elements)
        ▸0: "B4", unsigned int16, EPSG:32613, 10980x10980 px
        ▸1: "B3", unsigned int16, EPSG:32613, 10980x10980 px
        ▸2: "B2", unsigned int16, EPSG:32613, 10980x10980 px
      ▸properties: Object (81 properties)
    ▸1: Image COPERNICUS/S2_SR/20190105T171709_20190105T172443_T13QHC (3 bands)
    ▸2: Image COPERNICUS/S2_SR/20190105T171709_20190105T172443_T14QKG (3 bands)
    ▸3: Image COPERNICUS/S2_SR/20190105T171709_20190105T172443_T14QKH (3 bands)
    ▸4: Image COPERNICUS/S2_SR/20190105T171709_20190105T172443_T14QLG (3 bands)
```

Figura 10.5: *Salida de la consola indicando los nombres de las bandas de las imágenes Sentinel-2, así como de la selección únicamente de las bandas 4, 3 y 2.*

Recuerde que para seleccionar ciertas bandas en una colección de imágenes, estas corresponden a bandas de las imágenes, no de la colección.
Esto es útil para revisar los nombres de las bandas.

Ordenación de una colección

El método .sort permite ordenar una colección de imágenes por alguna propiedad de estas. Por defecto, las colecciones de imágenes están ordenadas cronológicamente. Sin embargo, se puede cambiar este orden para que las presente por alguna característica de las imágenes, por ejemplo, por orden de menor a mayor cobertura de nubes. Por defecto, el método .sort ordena la colección de imágenes de forma ascendente, de acuerdo al valor de la característica indicada. Si se desea ordenar de manera descendente, esto se puede indicar utilizando el argumento ascending: **false**. En el siguiente ejemplo, primero se ordena la colección en orden ascendente y en el segundo, descendente.

| EJERCICIO 28.6

```
// Cambiar el orden de la colección en función de la propiedad de
// las imágenes de cobertura de nubes en sentido descendente
imCol5 = imCol5.sort({property: 'CLOUD_COVERAGE_ASSESSMENT',
    ascending: false});
```

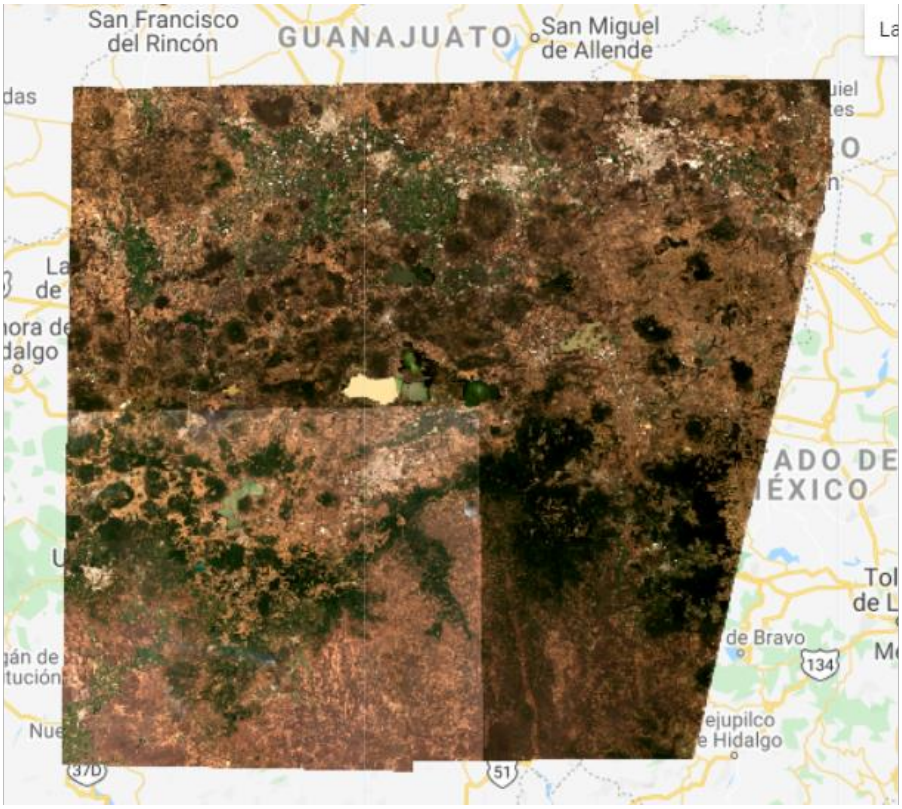


Figura 10.6: Visualización de la primera imagen de la colección.

Obtención de la primera imagen

En algunas ocasiones resulta útil obtener la primera imagen de una colección de imágenes para ver si se llevó a cabo el procedimiento deseado sobre la colección. Para ello, se utiliza el método `.first`, que devuelve la primera imagen que se encuentra en la colección y permite visualizarla en la pantalla de mapa para analizarla más a detalle. Al utilizar el método `.first` sobre una colección de imágenes (`ee.ImageCollection`) el resultado corresponderá a una imagen (`ee.Image`; Fig. 10.7).

| EJERCICIO 28.7

```
// Obtener la primera imagen de una colección
var primera = imCol3.first()
```

```
▾ Image COPERNICUS/S2_SR/20190105T171709_20190105T172443_T13QHB (23 …   JSON
    type: Image
    id: COPERNICUS/S2_SR/20190105T171709_20190105T172443_T13QHB
    version: 1556514341039576
  ▾ bands: List (23 elements)
    ▸ 0: "B1", unsigned int16, EPSG:32613, 1830x1830 px
    ▸ 1: "B2", unsigned int16, EPSG:32613, 10980x10980 px
    ▸ 2: "B3", unsigned int16, EPSG:32613, 10980x10980 px
    ▸ 3: "B4", unsigned int16, EPSG:32613, 10980x10980 px
    ▸ 4: "B5", unsigned int16, EPSG:32613, 5490x5490 px
    ▸ 5: "B6", unsigned int16, EPSG:32613, 5490x5490 px
    ▸ 6: "B7", unsigned int16, EPSG:32613, 5490x5490 px
    ▸ 7: "B8", unsigned int16, EPSG:32613, 10980x10980 px
    ▸ 8: "B8A", unsigned int16, EPSG:32613, 5490x5490 px
    ▸ 9: "B9", unsigned int16, EPSG:32613, 1830x1830 px
    ▸ 10: "B11", unsigned int16, EPSG:32613, 5490x5490 px
    ▸ 11: "B12", unsigned int16, EPSG:32613, 5490x5490 px
    ▸ 12: "AOT", unsigned int16, EPSG:32613, 10980x10980 px
    ▸ 13: "WVP", unsigned int16, EPSG:32613, 10980x10980 px
    ▸ 14: "SCL", unsigned int8, EPSG:32613, 5490x5490 px
    ▸ 15: "TCI_R", unsigned int8, EPSG:32613, 10980x10980 px
    ▸ 16: "TCI_G", unsigned int8, EPSG:32613, 10980x10980 px
    ▸ 17: "TCI_B", unsigned int8, EPSG:32613, 10980x10980 px
    ▸ 18: "MSK_CLDPRB", unsigned int8, EPSG:32613, 5490x5490 px
    ▸ 19: "MSK_SNWPRB", unsigned int8, EPSG:32613, 5490x5490 px
    ▸ 20: "QA10", unsigned int16, EPSG:32613, 10980x10980 px
    ▸ 21: "QA20", unsigned int32, EPSG:32613, 5490x5490 px
    ▸ 22: "QA60", unsigned int16, EPSG:32613, 1830x1830 px
  ▸ properties: Object (81 properties)
primera                                                                JSON
```

Figura 10.7: Salida de la consola indicando la primera imagen de la colección.

Extracción de listas con atributos de imágenes

Para obtener una lista con los atributos de las imágenes que conforman una colección de imágenes se utiliza el método `.aggregate`. Este método tiene varias

formas de obtener y resumir los atributos de los vectores que conforman una colección: extraer una propiedad de los vectores (.aggregate_array), extraer y calcular la media por propiedad (.aggregate_mean), extraer y calcular un histograma (.aggregate_histogram) o extraer y obtener estadísticas descriptivas de las propiedades (.aggregate_stats). Por ejemplo, una lista con la propiedad 'CLOUD_COVERAGE_ASSESSMENT' de todas las imágenes contenidas en una colección de imágenes se puede obtener de la siguiente manera (Fig. 10.8):

| EJERCICIO 28.8

```
// Obtener una lista de con los valores de la propiedad de
// cobertura de nubes de todas las imágenes de la colección
var listMetadatos =
imCol5.aggregate_array('CLOUD_COVERAGE_ASSESSMENT');
```

```
▾List (134 elements)                                    JSON
  0: 48.075073
  1: 43.1642
  2: 42.381954
  3: 38.767011
  4: 36.728684
  5: 28.089517
  6: 25.068042
  7: 24.11147
  8: 24.061825
  9: 23.849605
  10: 22.640179
  11: 19.898208
  12: 12.417454
  13: 12.202027
  14: 11.50309
  15: 11.04683
  16: 10.283798
  17: 10.132086
  18: 9.412513
  19: 9.068769
  20: 8.785343
  21: 8.436566
  22: 8.40476
  23: 8.178108
  24: 8.074684
  25: 7.383551
  26: 7.055483
  27: 7.008205
  28: 6.844971
  29: 5.985491
  30: 5.601502
  31: 5.587979
  32: 5.100831
  33: 5.088097
```

Figura 10.8: Salida de la consola indicando la lista de propiedades de las imágenes.

Conteo de observaciones por píxel

Para conocer el número de valores sin enmascarar por píxel en la colección de imágenes se utiliza el método `.count`. El resultado de este procedimiento consta de una imagen con las mismas bandas que las imágenes a partir de las cuales se calculó el número de observaciones por píxel (Fig. 10.9).

```
// Obtener el número de observaciones válidas para un píxel dentro
// de una colección de imágenes
var conteoImgs = imCol5.count();
```

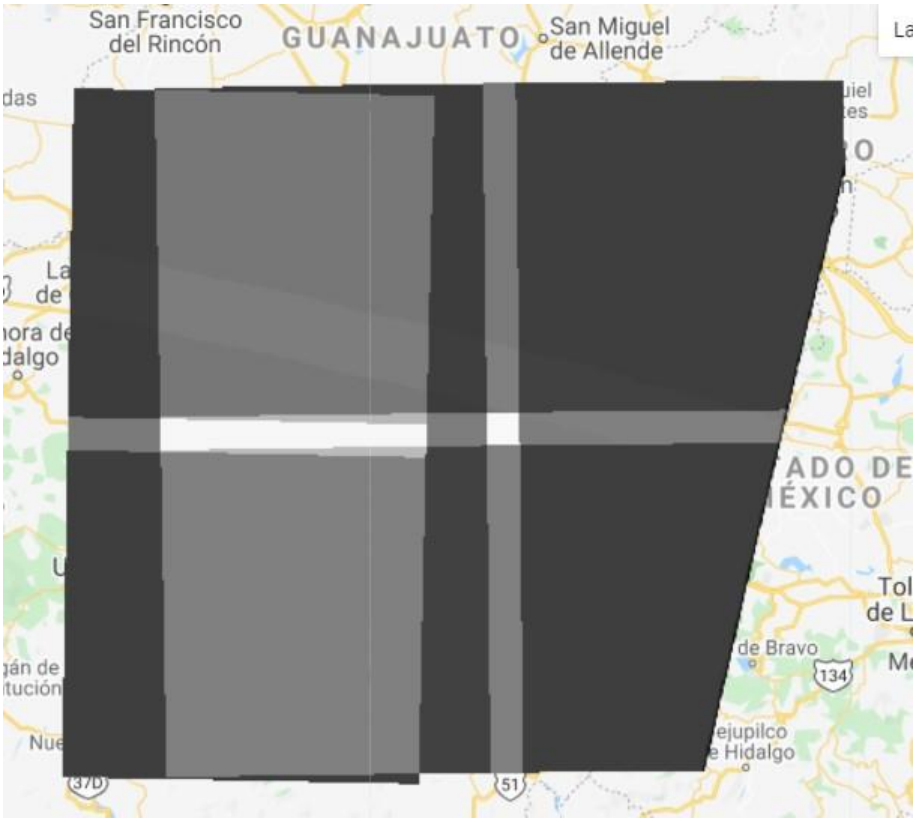


Figura 10.9: Visualización de la imagen del conteo de observaciones.

Conocer el tamaño de la colección

Para conocer el tamaño de la colección en número de imágenes incluidas se utiliza el método `.size`.

```
// Obtener el número de observaciones válidas para un píxel dentro
// de una colección de imágenes
var numImgs = imCol5.size();
```

Unión de más de una colección

Para unir una colección de imágenes con otra, se utiliza el método `.merge`. Un caso típico de tal operación sería unir dos colecciones de Landsat, por ejemplo, Landsat7 y 8. La unión de una colección de Sentinel-2 se podría hacer de la siguiente manera (Fig.10.10):

| EJERCICIO 28.9

```
// Aplicar un filtro por fechas a la colección de Sentinel-2
var imCol6 = S2.filterDate('2019-01-01','2019-05-01')
    // Aplicar filtro espacial
    .filterBounds(roi);

// Hacer lo mismo que el caso anterior, pero cambiando las fechas
var imCol7 = S2.filterDate('2020-06-01','2020-11-01')
    .filterBounds(roi);

// Unir las dos colecciones de imágenes en una sola
var imCol8 = imCol6.merge(imCol7);
```

```
▾ ImageCollection COPERNICUS/S2_SR (592 elements)          JSON
    type: ImageCollection
    id: COPERNICUS/S2_SR
    version: 1633349976941267
    bands: []
  ▸ features: List (592 elements)
  ▸ properties: Object (21 properties)
imCol6                                                     JSON

▾ ImageCollection COPERNICUS/S2_SR (632 elements)          JSON
    type: ImageCollection
    id: COPERNICUS/S2_SR
    version: 1633349976941267
    bands: []
  ▸ features: List (632 elements)
  ▸ properties: Object (21 properties)
imCol7                                                     JSON

▾ ImageCollection (1224 elements)                          JSON
    type: ImageCollection
    bands: []
  ▸ features: List (1224 elements)
imCol8                                                     JSON
```

Figura 10.10: Salida de la consola indicando el número de elementos en cada colección, así como el número de elementos en la colección combinada.

Las colecciones Landsat 7 y 8 no tienen las mismas bandas, ni los nombres de las bandas corresponden entre sí. Por ejemplo, la banda 1 de Landsat 7 corresponde al espectro del azul (*blue*), mientras que en Landsat 8, la banda 1 corresponde al azul de costa (*coastal blue*). En este caso, conviene primero filtrar y renombrar las bandas a unir para asegurarse de que el nombre de las bandas a unir en una colección corresponda con el de la otra.

Ejecución de una función sobre todas las imágenes de una colección

Para realizar alguna operación sobre todas las imágenes de una `ee.ImageCollection` se utiliza el método `.map`. Esto va a aplicar el procedimiento o función que se defina en el interior de ella, a cada imagen dentro de la colección. Por ejemplo, si quisiéramos agregar una banda con un índice de vegetación a todas las imágenes, se podría hacer de la siguiente manera (Fig. 10.11):

| EJERCICIO 28.10

```
// Definir una función que calcule el ndvi de una imagen
var ndviCalc = function(image){
    // Calcular el ndvi utilizando las bandas del NIR: B8 y R: B4
    var ndvi = image.normalizedDifference(['B8','B4']);
    // Regresar la imagen con la nueva banda de ndvi
    return image.addBands(ndvi);
};

// Aplicar la función sobre todas las imágenes de una colección
var imCol9 = imCol4.map(ndviCalc);
```

Nótese que el método `.map` es distinto de la función `Map` que permite agregar objetos a la pantalla de mapa. Recuerde que JavaScript es un lenguaje sensible a mayúsculas y minúsculas.

GEE recomienda el uso de `.map` para realizar operaciones de tipo ciclo, en lugar de otras funciones (por ejemplo, for).

El método `.map` requiere del uso de funciones del lado del servidor, así que se recomienda evitar el uso de funciones del lado del cliente como `print`, `Map`, `Export` y `Chart`.

```
▾ImageCollection COPERNICUS/S2_SR (134 elements)                    JSON
    type: ImageCollection
    id: COPERNICUS/S2_SR
    version: 1633384966582513
    bands: []
  ▾features: List (134 elements)
    ▾0: Image COPERNICUS/S2_SR/20190105T171709_20190105T172443_…
        type: Image
        id: COPERNICUS/S2_SR/20190105T171709_20190105T172443_T13Q…
        version: 1556514341039576
      ▾bands: List (24 elements)
        ▸0: "B1", unsigned int16, EPSG:32613, 1830x1830 px
        ▸1: "B2", unsigned int16, EPSG:32613, 10980x10980 px
        ▸2: "B3", unsigned int16, EPSG:32613, 10980x10980 px
        ▸3: "B4", unsigned int16, EPSG:32613, 10980x10980 px
        ▸4: "B5", unsigned int16, EPSG:32613, 5490x5490 px
        ▸5: "B6", unsigned int16, EPSG:32613, 5490x5490 px
        ▸6: "B7", unsigned int16, EPSG:32613, 5490x5490 px
        ▸7: "B8", unsigned int16, EPSG:32613, 10980x10980 px
        ▸8: "B8A", unsigned int16, EPSG:32613, 5490x5490 px
        ▸9: "B9", unsigned int16, EPSG:32613, 1830x1830 px
        ▸10: "B11", unsigned int16, EPSG:32613, 5490x5490 px
        ▸11: "B12", unsigned int16, EPSG:32613, 5490x5490 px
        ▸12: "AOT", unsigned int16, EPSG:32613, 10980x10980 px
        ▸13: "WVP", unsigned int16, EPSG:32613, 10980x10980 px
        ▸14: "SCL", unsigned int8, EPSG:32613, 5490x5490 px
        ▸15: "TCI_R", unsigned int8, EPSG:32613, 10980x10980 px
        ▸16: "TCI_G", unsigned int8, EPSG:32613, 10980x10980 px
        ▸17: "TCI_B", unsigned int8, EPSG:32613, 10980x10980 px
        ▸18: "MSK_CLDPRB", unsigned int8, EPSG:32613, 5490x549…
        ▸19: "MSK_SNWPRB", unsigned int8, EPSG:32613, 5490x549…
        ▸20: "QA10", unsigned int16, EPSG:32613, 10980x10980 px
        ▸21: "QA20", unsigned int32, EPSG:32613, 5490x5490 px
        ▸22: "QA60", unsigned int16, EPSG:32613, 1830x1830 px
        ▸23: "nd", float ∈ [-1, 1], EPSG:32613, 10980x10980 px
      ▸properties: Object (81 properties)
```

Figura 10.11: Salida de la consola con la información de la colección de imágenes con la banda NDVI agregada a cada imagen.

Reducción de una colección de imágenes

Los reductores permiten crear un compuesto (una sola imagen) a partir de una colección de imágenes. Por ello, el resultado obtenido tras una reducción sobre una colección de imágenes (`ee.ImageCollection`) será una imagen (`ee.Image`). Los reductores son muy útiles para hacer mosaicos multitemporales u obtener información de la colección de imágenes. Para reducir una colección de imágenes se utilizará el método `.reduce`, indicando la operación matemática 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`. Otra forma de realizar estas mismas reducciones es utilizando directamente los métodos de `.min`, `.max`, `.mean`, `.mode`, `.median`. Las siguientes dos líneas realizan exactamente el mismo proceso (Fig. 10.12):

| EJERCICIO 28.11

```
// Calcular el valor promedio de reflectancia para cada una de las
// bandas de las imágenes que conforman la colección
var imRedMean = imCol9.mean();

// Realizar lo mismo, pero utilizando explícitamente la definición
// del reductor
var imRedMean2 = imCol9.reduce(ee.Reducer.mean());
```

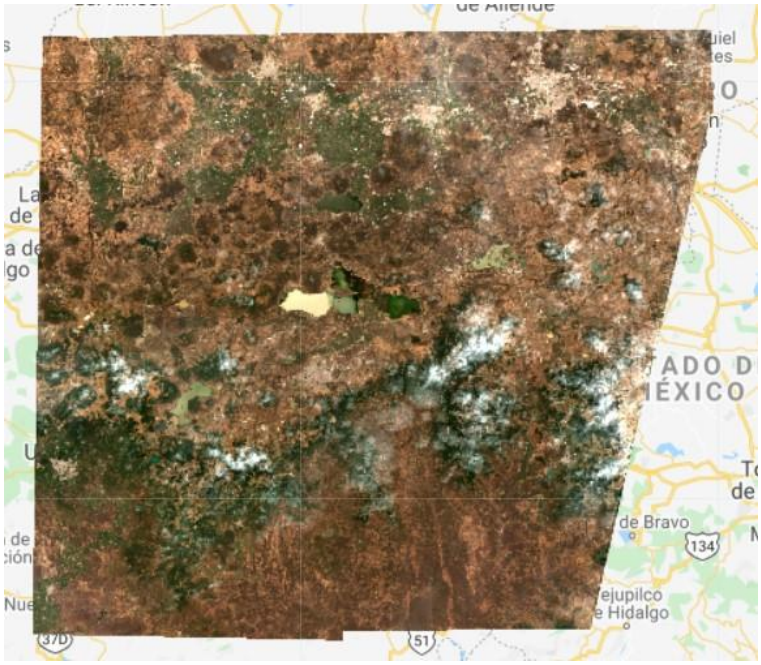


Figura 10.12: Visualización de la imagen de la reflectancia promedio (compuesto RGB).

Al aplicar una reducción a una colección de imágenes mediante el método `.reduce`, el nombre de las bandas será el mismo que el de las bandas de cada imagen con un sufijo que indica la operación matemática 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'`.

Creación de compuestos y mosaicos

Si se desea hacer mosaicos a partir de varias imágenes se puede realizar en GEE mediante dos métodos: `.mosaic` y `.qualityMosaic`. La principal diferencia entre estos dos procesos 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, el

método de .qualityMosaic permite priorizar el píxel 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 píxel con mayor valor de NDVI. En el caso de .qualityMosaic hay que indicar la banda que se utilizará como la banda de calidad para crear el mosaico (Fig. 10.13).

| EJERCICIO 28.12

```
// Crear un mosaico de las imágenes contenidas en una colección.
// Generalmente en áreas de sobreposición se tomarán los valores
// de las imágenes más recientes
var imRedMosaic = imCol9.mosaic();

// Crear un mosaico utilizando como banda de calidad 'nd' que en
// este caso corresponde al ndvi
// Este método permite obtener mosaicos, por ejemplo, del valor
// del píxel de mayor verdor o mayor valor de ndvi
var imRedQualMosaic = imCol9.qualityMosaic('nd');
```

Figura 10.13: Visualización de la imagen de la reflectancia promedio con el método de mosaico de calidad (compuesto RGB).

EJERCICIO B: FILTRO DE COLECCIÓN DE IMÁGENES Y VISUALIZACIÓN

Este ejercicio integra elementos de los ejercicios **28.4**, **28.5**, **28.6** y **28.7**, de modo que sea posible remitirse a estos ejercicios si se tienen dudas.

En un solo paso, aplicando múltiples filtros, se puede 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*. Para ello, primero se indica la ruta de la colección que se desea utilizar. Para encontrar esta ruta se puede buscar la colección de interés en la barra de búsqueda (**Search**), después se le da clic y del lado izquierdo aparecerá la ruta de la colección. Por otro lado, en esta misma ventana, en la pestaña de **Band** se pueden 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 (Fig. 10.14).

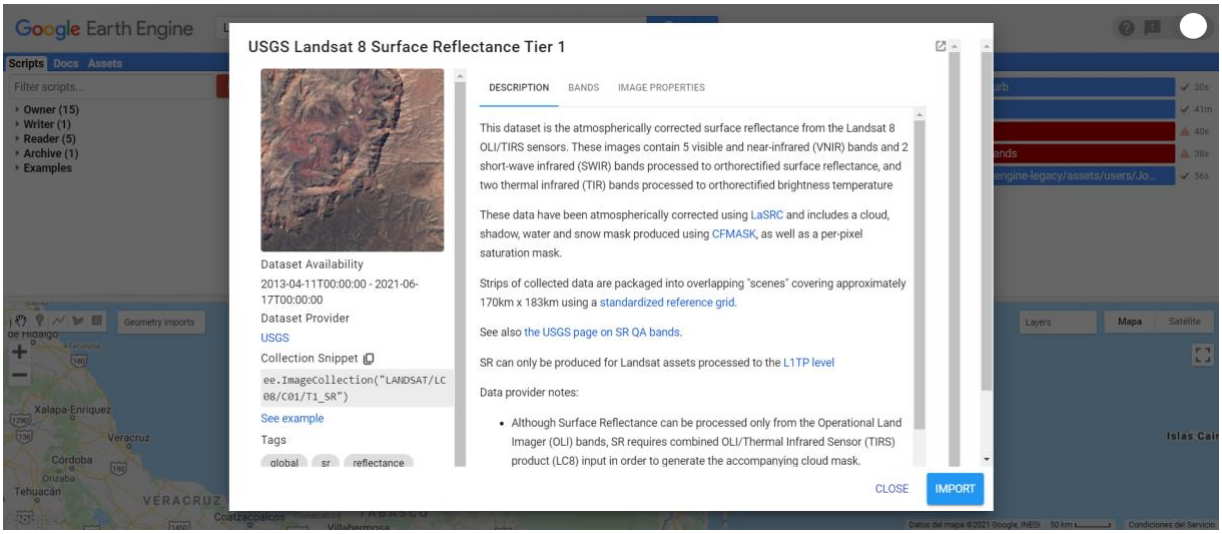


Figura 10.14: Consulta de propiedades de las imágenes contenidas en una colección de imágenes.

Una vez que se conoce la ruta de la colección de interés, se llama dicha colección y se especifican los filtros. En este caso, primero se filtra por fecha, en formato **AAAA-MM-DD** (año, mes, día), seguido de filtros de los metadatos de las imágenes:

- `'CLOUD_COVER_LAND'`.
- `'WRS_PATH'`.
- `'WRS_ROW'`.

Para realizar este filtro se debe escribir exactamente igual el nombre de las propiedades (por ejemplo, ‘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 igual (.lte).

```
// Filtrado de la colección
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 y ver sus características en la consola.

```
print(L8imgCol);
```

Una vez pasado 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. Después, se pueden consultar las características de las imágenes. Para ello, en la consola se le da:

1. Un clic a la colección en la consola.
2. Luego un clic a **features**.
3. Después un clic a cualquier imagen (**Image**).
4. 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. Esta información es la misma que se puede obtener en la barra de búsqueda (Fig. 10.15).

A continuación se ordena la colección de manera ascendente de acuerdo a la propiedad de ‘CLOUD_COVER_LAND’. 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, este va a ser un objeto ee.Image.

```
// Ordenar imágenes y obtener la primera
L8imgCol = L8imgCol.sort('CLOUD_COVER_LAND');
var L8imgFirst = L8imgCol.first();
print(L8imgFirst);
```

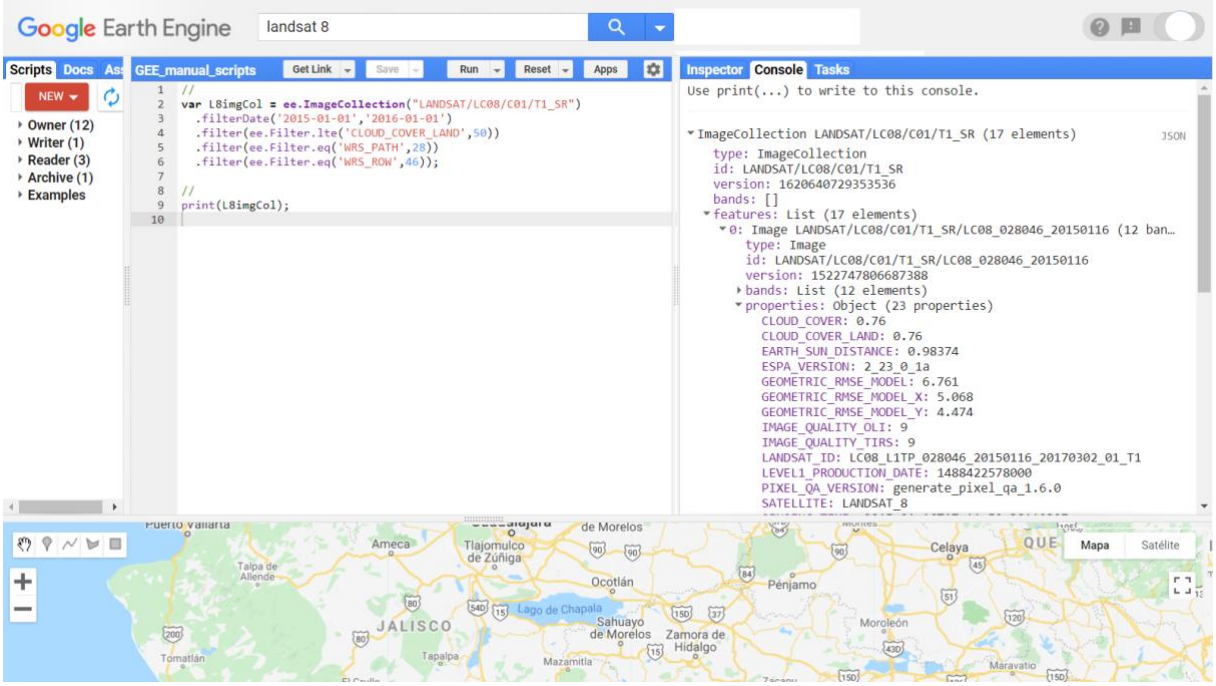


Figura 10.15: Ejemplo de una consulta de información de la colección de imágenes en la consola.

Además, se puede agregar esta imagen a la ventana de mapas para visualizar su información (Fig. 10.16). Esto se puede realizar de dos maneras: 1) agregando la información sin pasar más argumentos, o 2) indicando las bandas y el orden que se desea cargar, así como los valores mínimos y máximos del histograma y un nombre para la capa.

```
// Agregar a la pantalla de mapa
Map.addLayer(L8imgFirst);

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

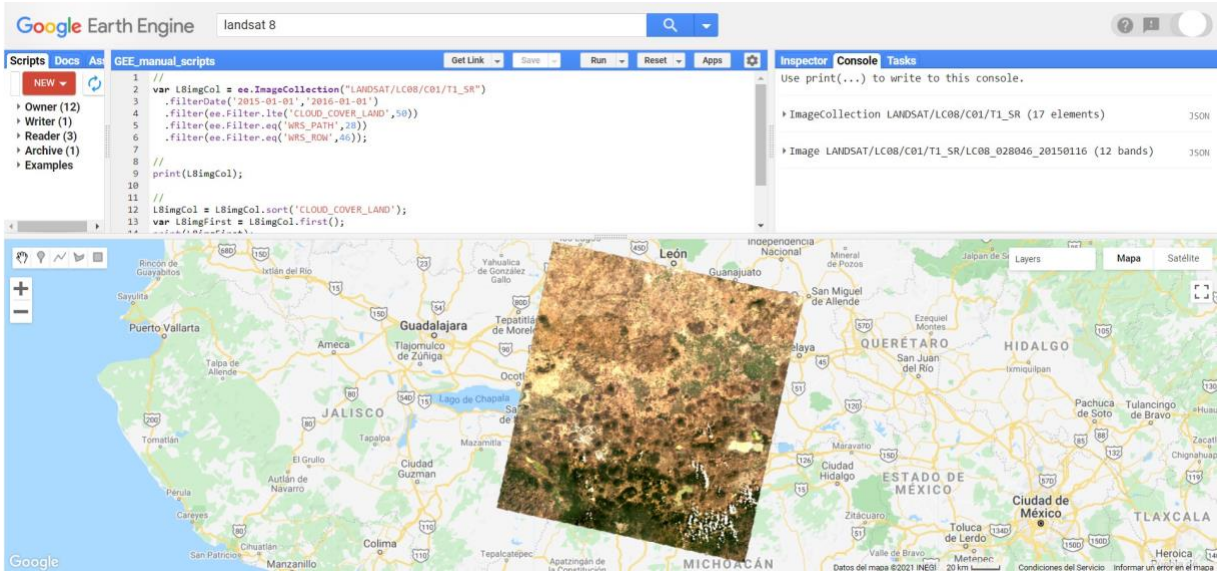


Figura 10.16: Visualización de la primera imagen de la colección de imágenes.

EJERCICIO C: ENMASCARAMIENTO DE NUBES, CÁLCULO DE ÍNDICES Y REDUCCIÓN

Este ejercicio integra elementos de los ejercicios **28.8** y **28.9**, de modo que puede remitirse a estos ejercicios si necesita de una mayor aclaración.

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 ello, 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 píxeles cuya información provenga de nubes o sombras. Después, se definirá una función para calcular el NDVI de cada imagen contenida en la colección de imágenes, utilizando el método `.normalizedDifference` e indicando las bandas del infrarrojo cercano (**NIR**) y rojo (**R**). También, se utilizará el método `.rename` para cambiar el nombre dado por defecto a la nueva banda calculada. Posteriormente, se agrega esta banda a la imagen con el método `.addBands` y se utiliza el método `.map` para aplicarla a cada una de las imágenes.

En este procedimiento, primero se tiene que definir la función para enmascarar las nubes. Para ello, 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'`. Se selecciona esta banda y se confirma que en los dos bits antes mencionados el valor sea igual a cero, es decir, que sea un píxel 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).

```
// Crear función para enmascarar nubes
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);
    };
```

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

```
// Crear función para calcular NDVI
var ndviCalc = function(image){
var ndvi = image.normalizedDifference(['B5','B4']);
ndvi = ndvi.rename('ndvi');
return image.addBands(ndvi);
};
```

Para obtener una muestra de cómo se ve la primera imagen con y sin la máscara de nubes, primero se agregará la imagen con nubes a la pantalla de mapa (Fig. 10.17).

```
// Agregar la primera imagen a la pantalla de mapa
Map.addLayer(L8imgCol.first(), {bands: ['B4','B3','B2'], min: 0,
    max: 1300},'Imagen con nubes');
```

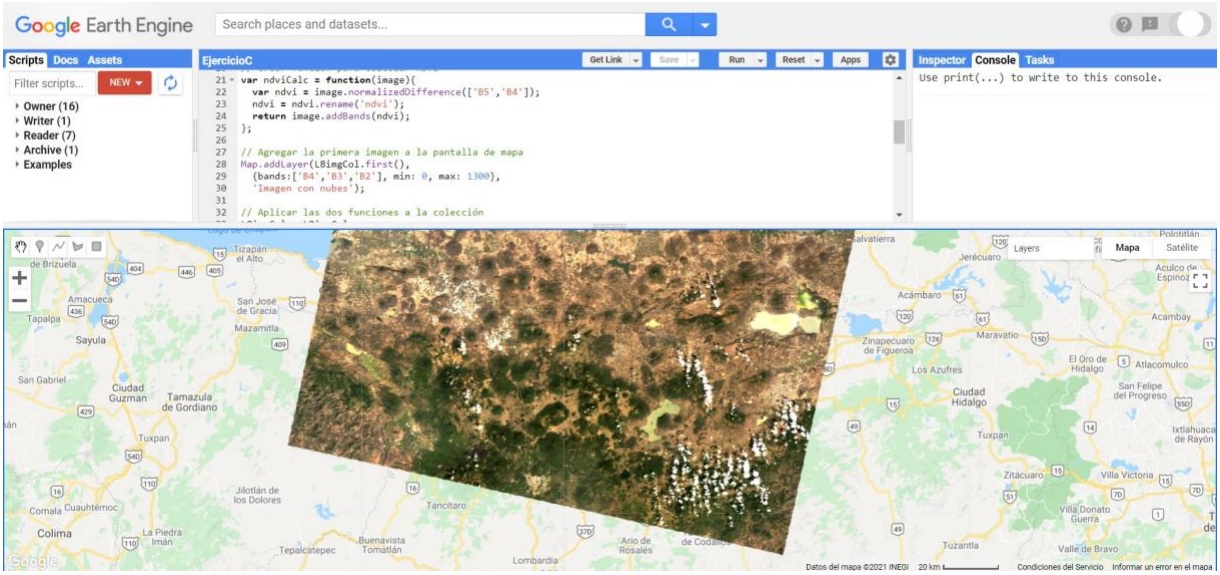


Figura 10.17: Ejemplo de una imagen con nubes.

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

```
// Aplicar las dos funciones a la colección
L8imgCol = L8imgCol
    .map(maskL8sr)
    .map(ndviCalc);

print(L8imgCol);
```

Se muestra la primera imagen sin nubes (Fig. 10.18).

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

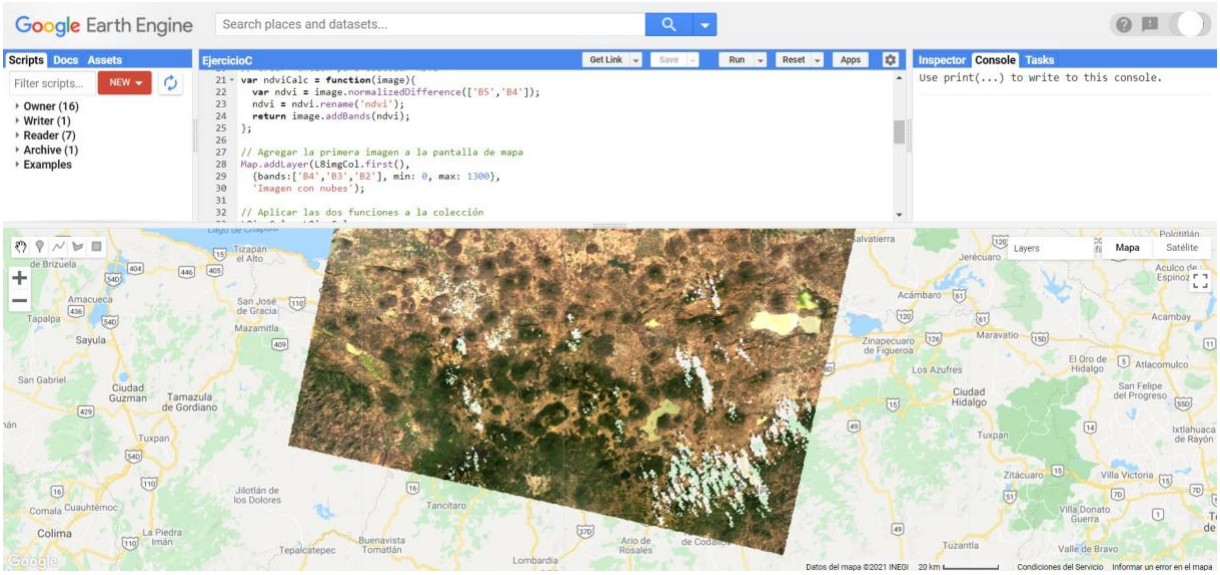


Figura 10.18: Ejemplo de una imagen donde se enmascararon las nubes.

En la consola se puede observar que todas las imágenes ahora contienen una nueva banda llamada `'ndvi'` (Fig. 10.19).

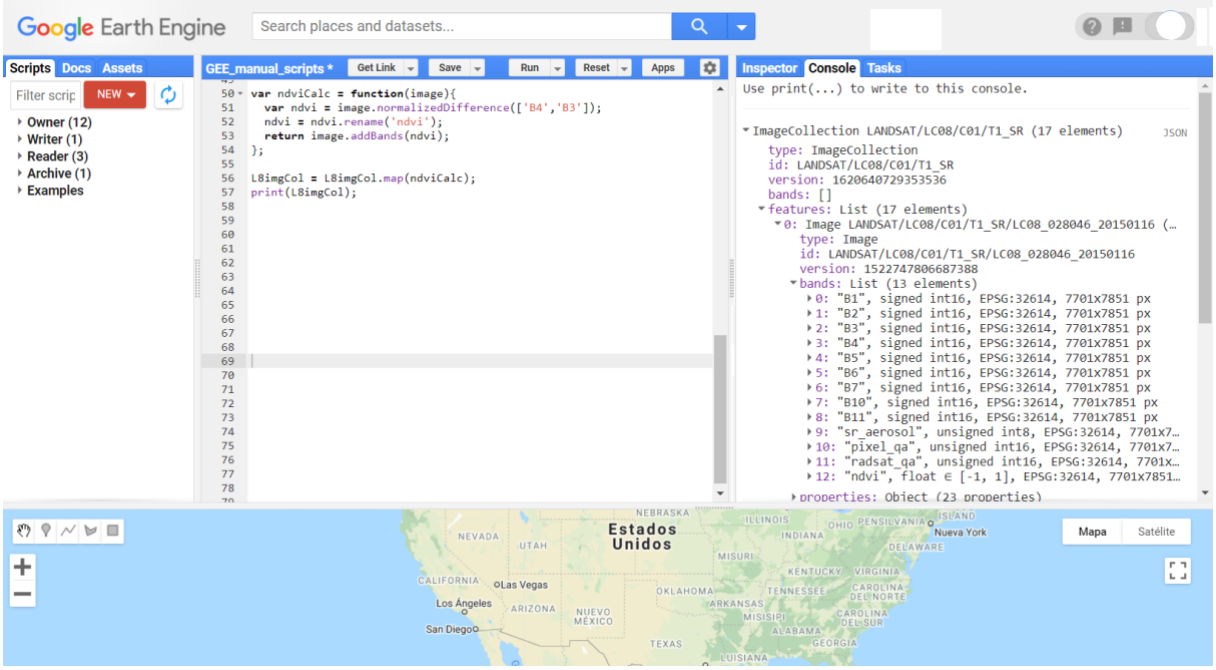


Figura 10.19: Consulta de información en la consola para verificar que la colección de imágenes contiene la banda NDVI.

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

```
// Seleccionar la banda ndvi y hacer dos reducciones
L8imgCol = L8imgCol.select('ndvi');

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

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

A continuación, agregamos los dos resultados a la pantalla de mapa y visualizamos el compuesto de media (Fig. 10.20).

```
// Visualizar las imágenes
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');
```

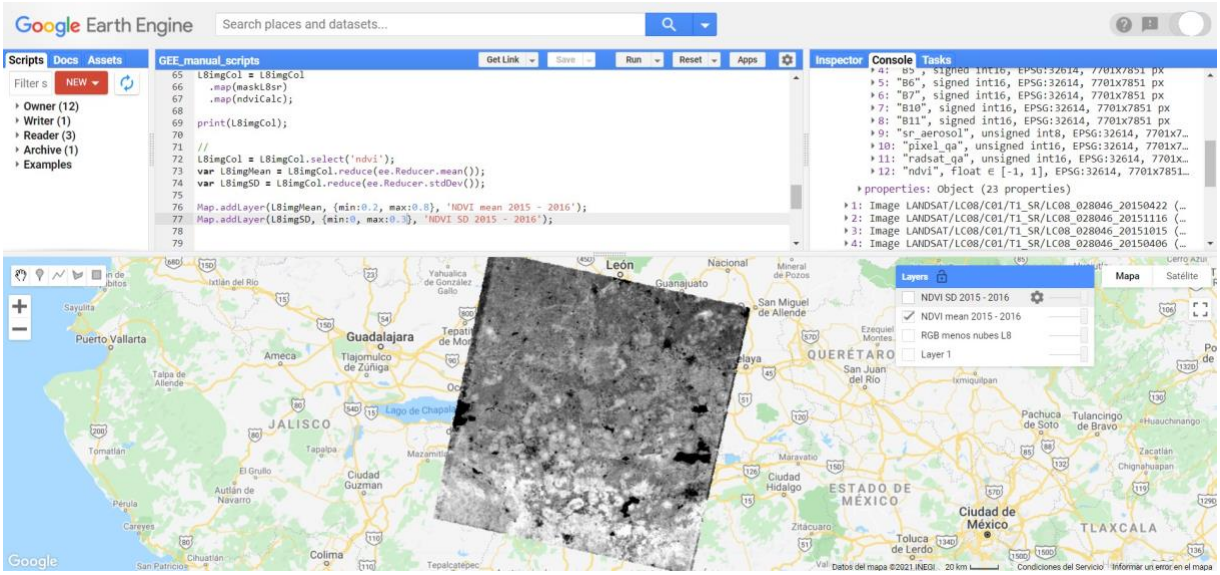


Figura 10.20: Visualización de la imagen de NDVI promedio.

Se puede conocer el número de observaciones sin enmascarar por píxel utilizando el método `count`. Debido a que ya se enmascararon las nubes en todas las imágenes de la colección, esta imagen contendrá el número de observaciones sin nube por píxel. Esta información nos permite conocer a partir de cuántas observaciones por píxel se calcularon las imágenes anteriores, es decir, las imágenes con el promedio y desviación estándar de NDVI.

```
// Obtener la imagen del número de observaciones por píxel sin
//enmascarar
var numObsPixel = L8imgCol.count();

Map.addLayer(numObsPixel,{min: 1, max: 17},
    'Número de observaciones sin enmascarar por píxel');
```

La imagen con el número de observaciones por píxel sin enmascarar se ve así (Fig.10.21):

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

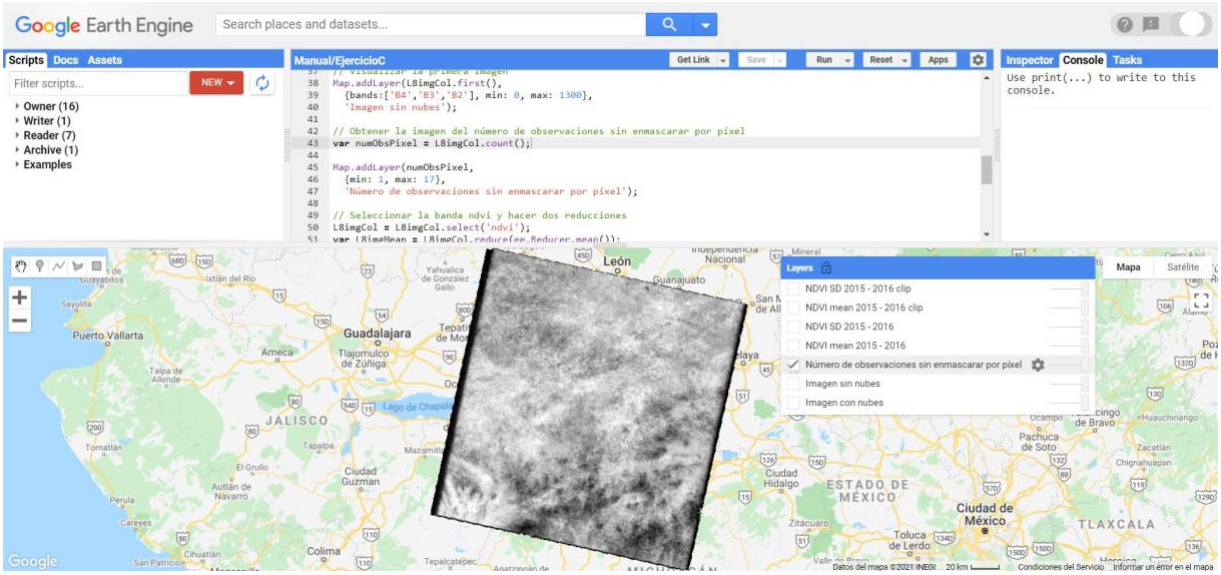


Figura 10.21: Visualización de la imagen con el número de observaciones sin enmascarar por píxel.

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

```
// Cargar el polígono del área de interés
var geometry = ee.Geometry.Polygon(
    [[[ -101.82737418916153, 19.836437094032178],
      [ -101.82737418916153, 19.368119068204525],
      [ -101.15171500947403, 19.368119068204525],
      [ -101.15171500947403, 19.836437094032178]]], null, false);
```

Para después cortar las dos imágenes y mostrarlas en la pantalla de mapa (Fig.10.22).

```
 // Cortar la imágenes
L8imgMean = L8imgMean.clip(geometry);

L8imgSD = L8imgSD.clip(geometry);

// Visualizarlas
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');
```

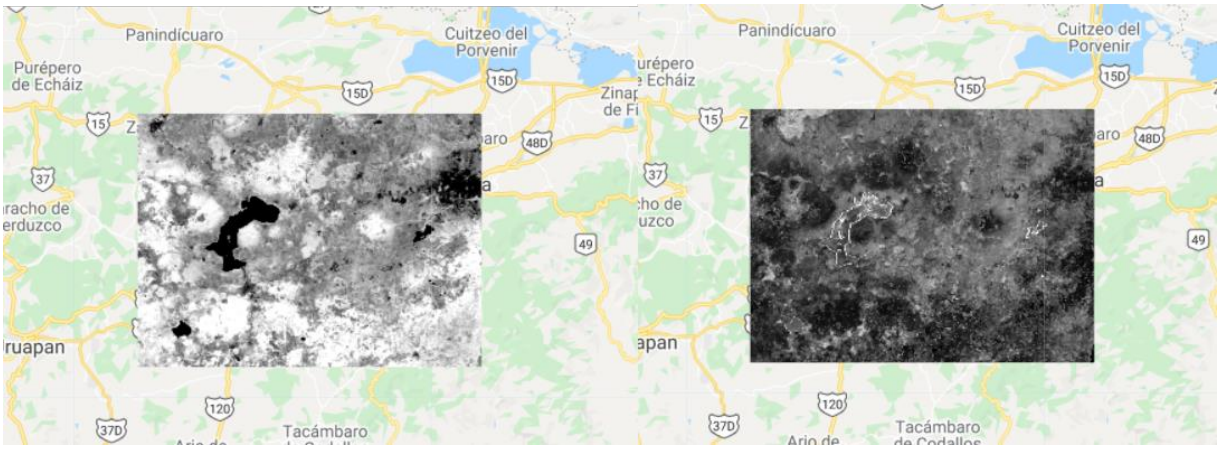


Figura 10.22: Visualización de los compuestos de media y desviación estándar cortados.

EJERCICIO D: CÁLCULO DE DIFERENCIA DE DOS IMÁGENES Y EXPORTACIÓN DE RESULTADO

Se utilizarán 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 enmascarar las nubes y agregar el NDVI a las imágenes.

```
// Filtrado en tiempo 2
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);
```

Después, se seleccionan únicamente las bandas de NDVI y se reduce la colección de imágenes para generar una imagen del valor promedio de NDVI. Posteriormente, se recorta la imagen a la extensión del polígono del área de interés. 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.

```
// Seleccionar NDVI y reducción
L8imgColT2 = L8imgColT2.select('ndvi');
var L8imgMeanT2 = L8imgColT2.reduce(ee.Reducer.mean());

// Cortar imagen a área de interés
L8imgMeanT2 = L8imgMeanT2.clip(geometry);

// Restar media T1 a T2
var imgDiff = L8imgMeanT2.subtract(L8imgMean);
```

Posteriormente se agrega el resultado a la pantalla de mapa (Fig. 10.23).

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

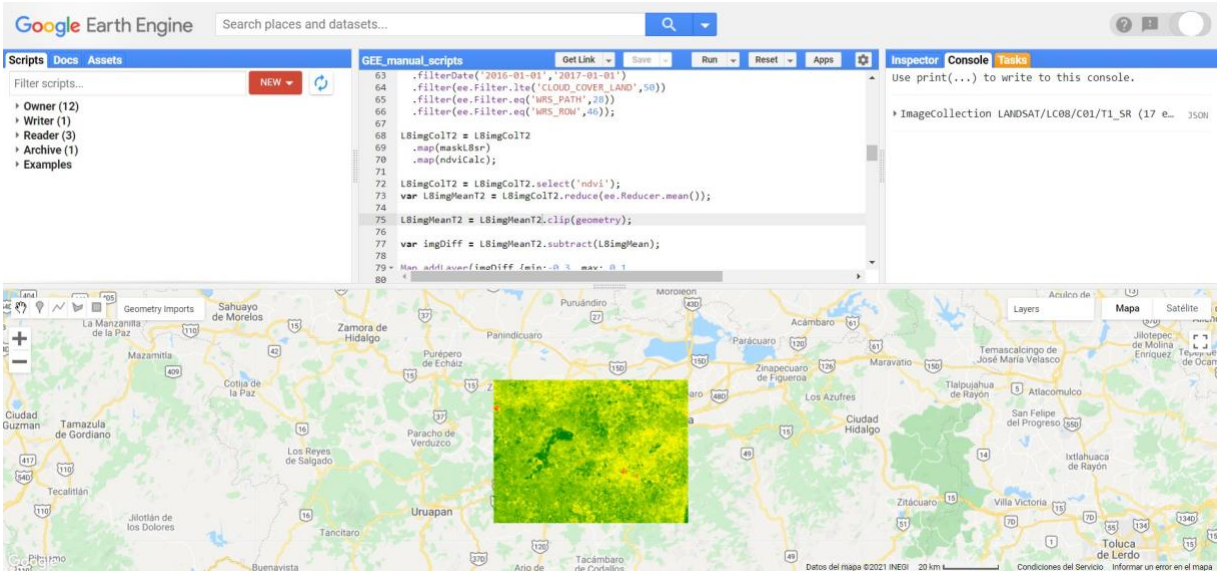


Figura 10.23: Imagen de la diferencia de NDVI entre dos años.

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 píxel en metros), formato a utilizar y si se desea guardar dentro de una carpeta del **Google Drive**. Estos argumentos, al igual que otras funciones en GEE, se pasan dentro de un diccionario.

```
// Exportar imagen a Google Drive
Export.image.toDrive({
    image: imgDiff,
    description: 'DiferenciaNDVI_2016-2017',
    scale: 30,
    maxPixels: 1e12,
    fileFormat: 'GeoTIFF',
    folder: 'DiferenciaNDVIL8',
    region: roi,
    crs: 'EPSG:4326'
});
```

Uso conjunto de vectores e imágenes

11

USO CONJUNTO DE VECTORES E IMÁGENES

Además de los métodos que existen en GEE para realizar operaciones sobre imágenes, colecciones de imágenes, vectores o colecciones de vectores, existen algunos métodos que permiten combinar estos dos tipos de información para realizar ciertos procesos. Los dos tipos de procesos más comúnmente utilizados son el muestreo de una imagen por vectores y la reducción de una imagen por vectores.

11.1 Métodos comunes

Muestreo por vector(es)

Este tipo de operaciones normalmente se utilizan para muestrear los datos de una imagen y obtener, por ejemplo, muestras por clases de coberturas para después entrenar un clasificador o para obtener los valores de un ráster en formato de vector (como puntos). En GEE, existen tres métodos para hacer este proceso: para un ráster completo (`.sample`), para un rectángulo (`.sampleRectangle`) o para varias regiones (`.sampleRegions`). Por ejemplo, el método `.sample` toma como argumentos el vector utilizado para determinar el área de muestreo, así como la escala a la que se desea realizar el muestreo. En este ejemplo, primero se define una geometría y después se hace el muestreo (Fig. 11.1).

EJERCICIO 29

```
// Definir región a muestrear
var bosque1 = ee.Geometry.Rectangle( -101.53892, 19.74148,
                                     -101.51906,19.72362);

// Muestrear imagen
var muestreo = L8imgMean
    .sample({
    region: bosque1,
    scale:30
    });
```

```
▾FeatureCollection (4891 elements, 1 column)                    JSON
  type: FeatureCollection
 ▾columns: Object (1 property)
    ndvi_mean: Float<-1.0, 1.0>
 ▾features: List (4891 elements)
  ▾0: Feature 0
     type: Feature
     id: 0
     geometry: null
    ▾properties: Object (1 property)
       ndvi_mean: 0.6194936633110046
  ▸1: Feature 1
  ▸2: Feature 2
  ▸3: Feature 3
  ▸4: Feature 4
  ▸5: Feature 5
  ▸6: Feature 6
  ▸7: Feature 7
  ▸8: Feature 8
  ▸9: Feature 9
```

Figura 11.1: Ejemplo de la salida de la consola de una colección de vectores obtenida mediante el método `sample`.

> Si se desea obtener la información de un muestreo de una imagen con su correspondiente información espacial, se puede utilizar el argumento geometries: true. Esto dará como resultado una tabla con la misma información, pero con las coordenadas geográficas de los centros de los píxeles de la imagen muestreada.

Reducción de una imagen por región(es)

Este tipo de operaciones normalmente se utilizan para resumir los valores de algún ráster en ciertas áreas de interés. En GEE, existen dos métodos para realizar este proceso: para una única región (`.reduceRegion`) o para varias regiones (`.reduceRegions`). Por ejemplo, el método `.reduceRegion` permite obtener estadísticas de la imagen en la extensión indicada por un área de interés (**geometry**). En este caso, además hay que indicar el reductor (**reducer**) y el tamaño de píxel para realizar la operación (**scale**), (Fig. 11.2).

```
// Reducción por región
var reduccion = ee.Image(L8imgMean)
    .reduceRegion({
    // Reductor a utilizar
    reducer: ee.Reducer.mean(),
    // Definir el área de interés a resumir
    geometry: bosque1,
    // Tamaño de píxel
    scale: 30
    });
```

```
▾Object (1 property)                                                    JSON
    ndvi_mean: 0.7900092091142787
 reduccion                                                              JSON
```

Figura 11.2: Ejemplo de la salida de la consola del promedio obtenido mediante el método .`reduceRegion`.

> En algunas ocasiones, los procesos de `.reduceRegions` o `.sampleRegions`, pueden demandar muchos recursos computacionales y arrojar un error de `'computation timed out'`. Para evitarlo, se pueden utilizar los argumentos de scale o tileScale, ya que ambos argumentos permiten reducir los costos computacionales del proceso. Scale permite aumentar el tamaño de píxel al que se va a resumir la información, mientras que tileScale permite definir un factor de escalamiento.

11.2 Interpolación de un vector a una imagen

Este procedimiento consiste en tomar los datos numéricos de un vector e interpolarlos a áreas fuera de los vectores, lo cual resulta en una imagen con valores en toda el área de interés.

Ponderación de distancia inversa

Este método interpola un valor para los píxeles de una imagen de acuerdo a los puntos con información y los pondera según su distancia, generando un promedio de esos valores. Esto provoca que los píxeles más cercanos tengan más influencia sobre el resultado de la interpolación que los píxeles más alejados. Para consultar más detalles sobre el procedimiento de interpolación, revisar Basso *et al*. (1999). Esta interpolación se realiza con el método `.inverseDistance`, donde hay que especificar al menos:

- El radio a interpolar alrededor de cada vector.
- El atributo (la propiedad) numérica del vector, que se interpolará.
- El promedio global de la variable.
- La desviación estándar global de la variable.

El siguiente ejercicio tiene como objetivo interpolar valores de metano (CH_4) para el norte de México. La imagen Sentinel-5 que se utilizará no tiene los valores de CH_4 para toda el área de estudio, sino que muchos de los píxeles se encuentran sin valores o enmascarados, es por ello que queremos interpolar la imagen, para poder tener

valores continuos en toda el área. Para ello se generarán 10000 muestras (puntos), que consisten en el valor de CH_4 de 10000 píxeles aleatorios. Con la ayuda de unos reductores extraemos el promedio y la desviación estándar de todas las muestras (puntos) y finalmente interpolamos las muestras en buffers de 70 km, usando como argumento el promedio y la desviación estándar (Fig. 11.3).

| EJERCICIO 30

```
// Importamos datos de metano de 5 días, los filtramos y sacamos
// el promedio.
var ch4 = ee.ImageCollection('COPERNICUS/S5P/OFFL/L3_CH4')
    .select('CH4_column_volume_mixing_ratio_dry_air')
    .filterDate('2020-01-01', '2020-01-05')
    .mean().rename('ch4');

// Definimos un rectángulo de área de estudio al norte de México.
var norteMex = ee.Geometry.Rectangle( -117.33, 32.753,
                                      -97.773, 22.634);

// Crear una colección de vectores para interpolar.
// Se crean dos bandas adicionales para la imagen indicando la
// longitud y latitud de cada píxel
var muestras = ch4.addBands(ee.Image.pixelLonLat())
    // Se crea una colección de vectores (sin geometría) en la
    // región norteMex y aproximadamente cuántos puntos generar
    .sample({region: norteMex, numPixels: 10000,
          // resolución de la imagen en metros por píxel
          scale:1000})
    // Función para cada punto generado
    .map(function(sample) {
          // Tomar el valor del punto de latitud
          var lat = sample.get('latitude');
          // Tomar el valor del punto de Longitud
          var lon = sample.get('longitude');
          // Tomar el valor del punto de metano
          var ch4 = sample.get('ch4');
          // Crear una geometría para el vector asignándole el
          //valor de metano
          return ee.Feature(ee.Geometry.Point([lon, lat]),
               {ch4: ch4});
    });

// Combinar un reductor de promedio y desviación estándar
var Reductores = ee.Reducer.mean().combine({
    reducer2: ee.Reducer.stdDev(),
    sharedInputs: true});
```

```
// Estimar el promedio y la desviación estándar de metano de los
// puntos.
var stats = muestras.reduceColumns({ reducer: Reductores,
    selectors: ['ch4']});

// Interpolar para 70 km.
var interpolado = muestras.inverseDistance({
    // Rango de interpolación para cada punto
    range: 70000,
    // Atributo numérico a interpolar
    propertyName: 'ch4',
    // Promedio (parámetro para el método)
    mean: stats.get('mean'),
    // Desviación estándar (parámetro para el método)
    stdDev: stats.get('stdDev') });
```

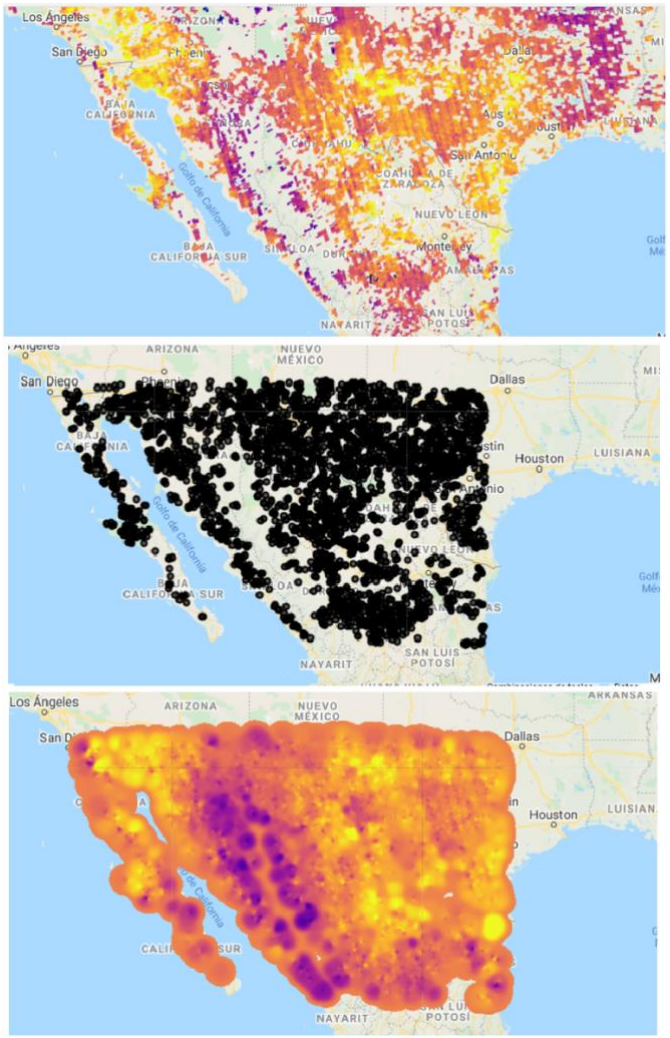


Figura 11.3: Visualización de la capa inicial de CH_4, (superior), puntos de muestreo (medio) e imagen interpolada (inferior).

Kriging

Es un método de interpolación que utiliza un estimado de la semivarianza para crear una imagen de valores interpolados que sea una combinación óptima de los datos conocidos. En este método hay que especificar algunos parámetros que describen la forma de la función de semivarianza ajustada a los datos conocidos, los cuales incluyen: el intervalo de la varianza (**range**), la meseta (**sill**), el valor del efecto pepita (**nugget**), la distancia máxima a analizar (**maxDistance**) y la función a utilizar como reductor (**reducer**). Este ejercicio mostrará cómo interpolar la información de la temperatura superficial del mar (imagen), a partir de un muestreo de puntos (vectores; Fig. 11.4).

| EJERCICIO 31

```
// Importar una imagen de temperatura marina superficial
var temperatura =
    ee.Image('NOAA/AVHRR_Pathfinder_V52_L3/20120802025048')
    .select('sea_surface_temperature').rename('temperatura')
    .divide(100);

// Definir una geometría donde muestrear puntos
var geometria = ee.Geometry.Rectangle([ -65.60, 31.75,
                                        -52.18, 43.12]);

// Muestrear las temperaturas en 1000 localizaciones aleatorias.
var muestras = temperatura.addBands(ee.Image.pixelLonLat())
    .sample({region: geometria, numPixels: 1000})
    .map(function(sample) {
          var lat = sample.get('latitude');
          var lon = sample.get('longitude');
          var temperatura1 = sample.get('temperatura');
          return ee.Feature(ee.Geometry.Point([lon, lat]),
                {temperaturaMarina: temperatura1});
          });

// Interpolar a partir de las muestras
var interpolar = muestras.kriging({
    propertyName: 'temperaturaMarina',
    // Forma de la función de la varianza de los datos
    shape: 'exponential',
    // Rango de la varianza
    range: 100 * 1000,
    sill: 1.0,
    nugget: 0.1,
    maxDistance: 100 * 1000,
    reducer: 'mean',
});
```

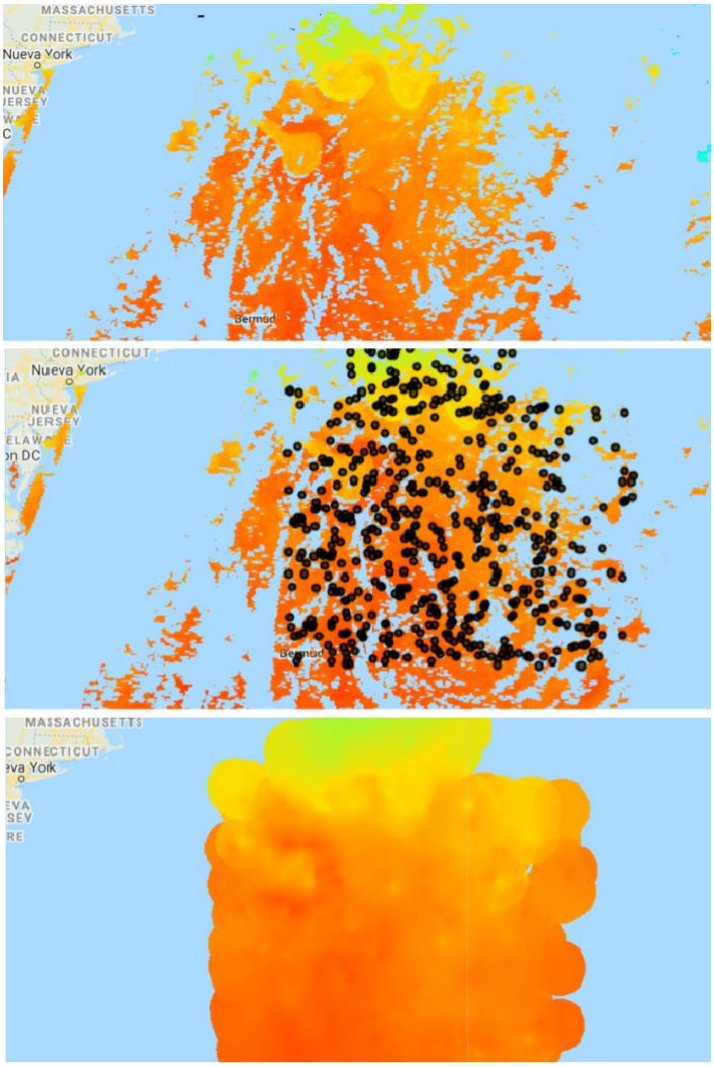


Figura 11.4: Visualización de la capa inicial de temperatura de la superficie marina (superior), puntos de muestreo (medio) y capa interpolada (inferior).

Clasificación Supervisada

12

CLASIFICACIÓN SUPERVISADA

Una clasificación supervisada se refiere al proceso en el cual se obtiene un mapa temático a partir de imágenes satelitales u otro tipo de información generada mediante sensores remotos. Este tipo de procedimiento tiene el objetivo de asociarle una etiqueta a cada píxel de una imagen, que puede constar de, por ejemplo, una clase de tipo de vegetación o uso de suelo (Chuvieco, 1995). Las clasificaciones se pueden ver como una interpretación de la reflectancia de una imagen en un sistema de clases de interés y normalmente constan de procedimientos en los cuales se entrena un algoritmo a reconocer estas clases, a partir de muestras otorgadas por el usuario (Chuvieco, 1995). Generalmente, el proceso de una clasificación supervisada implica los siguientes pasos:

1. Obtención de los datos de entrenamiento.
2. Selección del clasificador.
3. Entrenamiento del clasificador basado en los datos de entrenamiento.
4. Obtención de la clasificación.
5. Registro de los datos de verificación.
6. Verificación de la clasificación.

En este capítulo, primero se describirá cada uno de estos pasos junto con las funciones y métodos utilizados y al final se integrarán dichos pasos en un ejercicio práctico.

Dentro de GEE se pueden hacer clasificaciones supervisadas con algunos algoritmos populares como **CART** (Classification and Regression Trees; Breiman *et al.*, 1984), **Random Forests** (Breiman, 2001), **Naive Bayes** (Webb, 2011), **Gradient Tree Boost** (Friedman, 2001), **Support Vector Machine** (Cortes y Vapnik, 1995), **MaxEnt** (Maximum Entropy; Phillips *et al.*, 2004) y **Decision Trees** (Quinlan, 1986). Como todos los demás objetos dentro de GEE, tienen su propio tipo de objetos del servidor.

12.1 Clasificadores

Los objetos `ee.Classifier` son los 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, así como los argumentos que aceptan, se pueden consultar en la página de ayuda, en la pestaña de referencia (**Reference**) en el apartado de bibliotecas del cliente (**Client libraries**), o también en la pestaña **Docs** bajo `ee.Classifier`. Cabe destacar que cada clasificador tiene diferentes argumentos. Por ejemplo, para el clasificador de random forests, los argumentos

incluyen: el número de árboles (**numberOfTrees**), las variables utilizadas por árbol aleatorio (**variablesPerSplit**), la población mínima de hoja (**minLeafPopulation**), la fracción de los datos de entrada para entrenar y verificar el algoritmo (**bagFraction**), el número máximo de nodos (**maxNodes**) y el número para hacer replicables los resultados (**seed**). Para consultar una descripción más detallada de este algoritmo y sus argumentos ver Breiman (2001).

| EJERCICIO 32

```
var rfClassif = ee.Classifier.smileRandomForest(500)
```

Usando al final el método `.setOutputMode` sobre un clasificador se puede cambiar la forma en la que el clasificador entrega los resultados. Para ello, dentro de los paréntesis del método hay que escribir cualquiera de las siguientes opciones:

- `'CLASSIFICATION'` (clasificación, por defecto): devuelve una variable nominal (clases).
- `'REGRESSION'` (regresión): devuelve una variable numérica.
- `'PROBABILITY'` (probabilidad): devuelve la probabilidad de que la clasificación sea correcta.
- `'MULTIPROBABILITY'` (multiprobabilidad): devuelve un arreglo de probabilidades para cada una de las clases.
- `'RAW'` (crudo): devuelve un arreglo con la representación interna del proceso de clasificación. Por ejemplo, los votos crudos de un modelo de múltiples árboles de decisión como random forests.
- `'RAW_REGRESSION'` (regresión cruda): devuelve un arreglo con la representación interna del proceso de regresión. Por ejemplo, las predicciones crudas de los múltiples árboles de regresión.

Resulta importante recordar que, dependiendo del clasificador, algunas de estas opciones de resultado estarán disponibles o no.

12.2 Realización de la fase de entrenamiento

Debido a que estos clasificadores trabajan en un esquema supervisado, requieren de la provisión de muestras de entrenamiento para entrenar al algoritmo. 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, por ejemplo, **QGIS**, y luego importar el archivo vector a GEE (ver sección de importación de información a GEE). Una vez definido el clasificador que se va a

utilizar, se utiliza el método .train para entrenarlo, utilizando las muestras provistas. Los argumentos de este método indican la colección de vectores que contiene las áreas de entrenamiento (**features**), el nombre del campo que indica el esquema de clasificación dentro de la colección de vectores (**classProperty**), así como las bandas a utilizar de la imagen (**inputProperties**).

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

12.3 Obtención de la clasificación

Una vez que se tiene entrenando el clasificador, el siguiente paso consta de clasificar la imagen completa en las clases de interés. Para ello, se utiliza el método .classify, el cual permite hacer predicciones sobre la clase de cada píxel a partir de las predicciones realizadas por el algoritmo entrenado. Por ejemplo:

```
var classifiedImg = imagen.classify(trainedClassifier)
```

12.4 Evaluación de la clasificación

La última fase de una clasificación consta de la evaluación de los resultados utilizando otro conjunto de datos que no haya sido empleado en la fase de entrenamiento. A estos datos se les conoce como los datos de validación o verificación. Para evaluar una clasificación, primero se extrae la información de las clases predichas para los datos de validación y después se compara con la información de referencia. Comúnmente, se utiliza una matriz de confusión para realizar dicha evaluación, la cual se puede obtener mediante el método .errorMatrix. Además, para calcular la precisión total de la clasificación sobre los datos de validación se utiliza el método .accuracy. Por ejemplo:

```
var validErrMat = validacion.errorMatrix('claseRef', 'clasePred');

var validAcc = validacion.accuracy();
```

EJERCICIO E: CLASIFICACIÓN

En este ejercicio se hará una clasificación supervisada utilizando el algoritmo random forests. Para ello se utilizarán siete polígonos como sitios de entrenamiento, pertenecientes a cuatro 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 vector (`ee.Feature`) y se le asigna la propiedad de `'clase'` como número a través de un diccionario. Por último, se hace una colección de vectores (Fig.12.1).

```
// Definición de datos de entrenamiento
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,
                                       -101.34509,19.64990);

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);

// Definición de colección de vectores
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}),
    ]);

// Agregar a pantalla de mapas
Map.addLayer(poligonos, {}, 'polígonosEntrenamiento');
```

Figura 12.1: Visualización de los sitios de entrenamiento.

Antes de pasar al filtrado de la colección de Landsat 8 y enmascarar las nubes, habrá que volver a definir la función **maskL8sr**, que es la misma que se definió en el capítulo 10 en el Ejercicio C: Enmascaramiento de nubes, cálculo de índices y reducción:

```
// Crear función para enmascarar nubes
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);
    };
```

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 usa el método `.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`.

```
// Filtrar colección y aplicar máscara de nubes
var L8imgCol = ee.ImageCollection('LANDSAT/LC08/C01/T1_SR')
    .filterBounds(poligonos)
    .filterDate('2015-01-01','2016-01-01')
    .filter(ee.Filter.lte('CLOUD_COVER_LAND',50))
    .map(maskL8sr);
```

El siguiente paso consiste en reducir la colección para generar una sola imagen con el valor promedio para todas las bandas. 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 una.

```
// Reducir, cortar, seleccionar y renombrar bandas
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']);
```

Posteriormente se corta la imagen para quedarnos únicamente con el área de interés. Para ello, primero se define un área de interés y después se corta la imagen con ese objeto.

```
// Definir el área de interés
var geometry = ee.Geometry.Polygon(
    [[[ -101.82737418916153, 19.836437094032178],
      [ -101.82737418916153, 19.368119068204525],
      [ -101.15171500947403, 19.368119068204525],
      [ -101.15171500947403, 19.836437094032178]
    ]]);

// Cortar la imagen a la geometría del área de interés
L8imgMean = L8imgMean.clip(geometry);
```

Una vez que ya se tiene la imagen sobre la que se va a hacer la clasificación, se muestrean los valores de los píxeles que se encuentren dentro de cada polígono de los datos de entrenamiento utilizando el método `.sampleRegions`. Para este proceso hay que indicar la colección de vectores que se va a utilizar, así como su propiedad que indica la clase y la escala a la que se van a muestrear los píxeles en metros. El resultado de este procedimiento corresponde a una colección de vectores, en la cual cada vector corresponde a un píxel dentro de un polígono de entrenamiento. De tal manera, para cada vector se indica la clase a la que pertenece, así como el valor en cada una de las bandas en la imagen muestreada (Fig. 12.2).

```
// Muestrear regiones
var training = L8imgMean.sampleRegions({
    // colección de vectores a utilizar
    collection: poligonos,
    // propiedad que se desea mantener en los vectores de la
    // colección
    properties: ['clase'],
    // tamaño de píxel
    scale: 30
});
```

El siguiente paso es entrenar al clasificador utilizando la colección de vectores anterior. En este caso, se va a utilizar el algoritmo random forests (Breiman, 2001), el cual se llama utilizando `ee.Classifier.smileRandomForest`. En este ejercicio se utilizarán `30` árboles de clasificación, que es el único argumento obligatorio. Los demás argumentos opcionales pueden ser consultados en la sección de **Reference** o en la pestaña de **Docs** e incluyen parámetros como el número de variables a utilizar por división y el número máximo de nodos, entre otros. Posteriormente, se entrena este algoritmo mediante el método `.train` y se pasan algunos argumentos adicionales dentro de un diccionario, como: la colección de vectores a utilizar para el entrenamiento (**features**), el nombre de la clase objetivo (**classProperty**) y el nombre de las propiedades de la imagen (**inputProperties**), es decir, el nombre de las bandas (Fig. 12.2).

```
// Entrenamiento del clasificador
var trainedClassifier = ee.Classifier.smileRandomForest(30)
    .train({
          // Vectores con la información extraída del paso anterior
          features: training,
          // Propiedad que indica la clase de cobertura
          classProperty: 'clase',
          // Nombres de las propiedades de la imagen
          inputProperties: ['CB','B','G','R','NIR','SWIR1','SWIR2']
    });
```

```
training                                                    JSON
▾FeatureCollection (20 elements, 0 columns)                 JSON
  type: FeatureCollection
  columns: Object (0 properties)
  ▾features: List (20 elements)
    ▾0: Feature 0_0
      type: Feature
      id: 0_0
      geometry: null
      ▾properties: Object (8 properties)
        B: 325.16129032258067
        CB: 253.80645161290323
        G: 577.6129032258065
        NIR: 2772.2903225806454
        R: 527.1290322580645
        SWIR1: 1975.8387096774193
        SWIR2: 1120.8387096774193
        clase: 0
    ▸1: Feature 0_1
    ▸2: Feature 0_2
    ▸3: Feature 0_3
    ▸4: Feature 0_4
    ▸5: Feature 0_5
    ▸6: Feature 0_6
    ▸7: Feature 0_7
    ▸8: Feature 0_8
    ▸9: Feature 0_9
    ▸10: Feature 0_10
    ▸11: Feature 0_11
    ▸12: Feature 0_12
    ▸13: Feature 0_13
```

```
trainedClassifier                                           JSON
▾Classifier.train                                           JSON
  type: Classifier.train
  ▸features: FeatureCollection (0 columns)
  classProperty: clase
  ▾classifier: Classifier.smileRandomForest
    type: Classifier.smileRandomForest
    numberOfTrees: 30
  ▾inputProperties: ["CB","B","G","R","NIR","SWIR1","SWIR2"]
    0: CB
    1: B
    2: G
    3: R
    4: NIR
    5: SWIR1
    6: SWIR2
```

*Figura 12.2: Salida de la consola de los primeros veinte puntos de entrenamiento con la información del ráster, así como del clasificador entrenado. Nótese que el total de puntos de validación es de más de 5000 puntos, pero se muestran solo los primeros **20** para ejemplificar el tipo de salida.*

> Generalmente se menciona que el utilizar un mayor número de árboles en el algoritmo Random Forest permite obtener mejores resultados, sin embargo, dependiendo del conjunto de datos utilizado existe un umbral a partir del cual aumentar el número de árboles no resulta en una mejoría significativa de la clasificación. Consultar Oshiro *et al.* (2012) para ver una evaluación del efecto del número de árboles sobre las capacidades de clasificación.

Posteriormente, se puede clasificar la imagen completa utilizando el método `.classify` e indicando como argumento al algoritmo entrenado. El resultado de `.classify` va a ser un objeto de tipo `ee.Image` que contiene la clase predicha en la banda `'classification'`. Por último, se visualiza el resultado en la pantalla de mapas (Fig. 12.3).

```
// Usar el clasificador entrenado sobre toda la imagen
var classifiedImg = L8imgMean.classify(trainedClassifier);

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

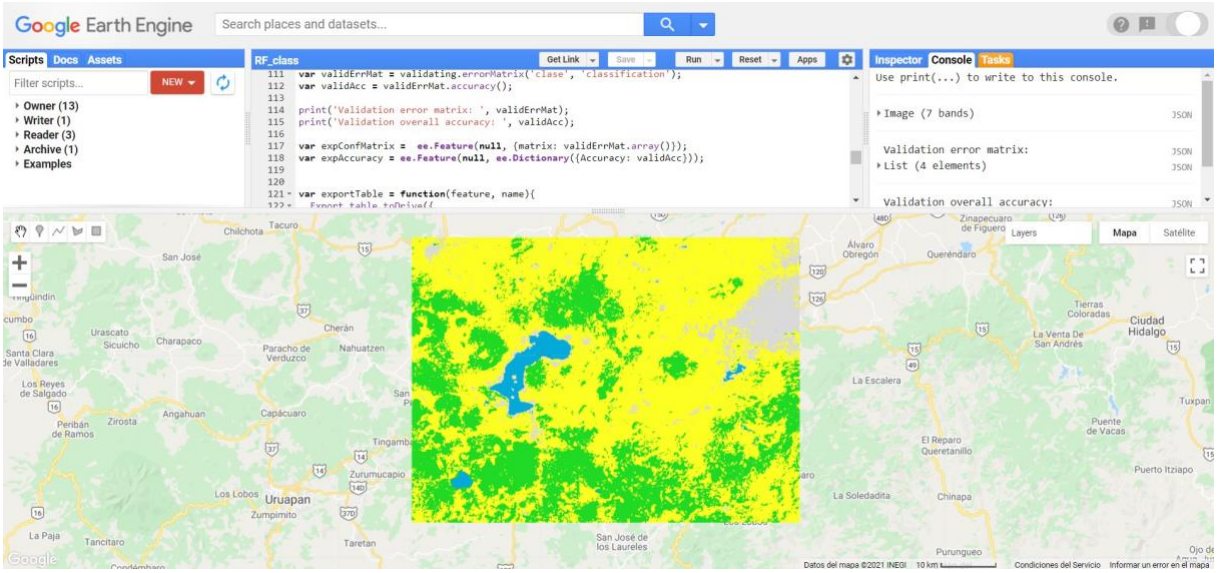


Figura 12.3: Visualización de la clasificación.

Después, se puede hacer un gráfico para analizar el perfil espectral de las clases de interés. Para hacer esto, primero se agrega la banda con la clasificación a la imagen que contiene la media de la reflectancia de la superficie (**L8imgMean**). Después, se utiliza la función `ui.Chart.image.byClass` para crear un gráfico por clase. Como argumentos a esta función se indica la imagen a utilizar (**image**), el nombre de la banda que contiene las categorías de la clasificación (**classBand**), el reductor que se va a utilizar para resumir los valores por clase (**reducer**), las regiones de las cuales se va a obtener esta información (**region**), las etiquetas de las clases (**classLabels**), la escala en m de trabajo para aplicar el reductor (**scale**) y algunas opciones para ponerle título y asignar los colores al gráfico (argumentos dentro de `.setOptions`). Por último, hay que imprimir el gráfico con la función de `print` y el gráfico aparecerá en la consola (Fig.12.4).

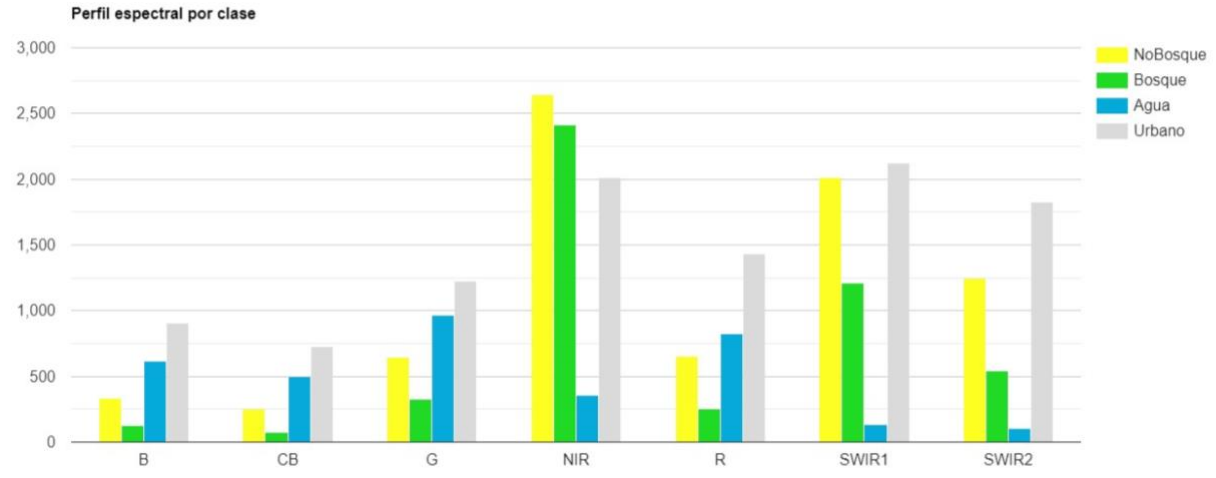


Figura 12.4: Perfil espectral promedio por clase.

```
// Agregar clasificación como banda
var imgConClass = L8imgMean.addBands(classifiedImg);

// Hacer gráfico de barras
var chartClassSpec = ui.Chart.image.byClass({
    // Imagen a utilizar para construir el gráfico
    image: imgConClass,
    // Banda que contiene las clases
    classBand: 'classification',
    // Reductor a utilizar
    reducer: ee.Reducer.mean(),
    // Regiones a partir de las cuales se va a construir el
    // gráfico
    region: poligonos,
    // Etiquetas de las clases
    classLabels: ['NoBosque','Bosque','Agua','Urbano'],
    // Tamaño de píxel
    scale: 240
    // Definir más opciones para el gráfico
    }).setOptions({
          // Título del gráfico
          title: 'Perfil espectral por clase',
          // Colores a utilizar
          colors: ['#fcff21','#20da25','#05a9da','#dadada']
          });

print(chartClassSpec);
```

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.

```
// Definción de áreas 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);
```

```
// Creación de colección de verificación
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 vectores (Fig. 12.5).

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

A continuación, se calcula la matriz de error mediante el método `.errorMatrix` indicando, en primer lugar, el campo de la clase de referencia (`‘clase’`) y, en segundo lugar, el campo de la predicción (`‘classification’`). Posteriormente, se calcula la precisión total de clasificación a partir de esta matriz de error. Después, se van a mostrar estos dos objetos en la consola mediante `print` para inspeccionar su contenido (Fig. 12.5).

```
// Obtener matriz de confusión y precisión
var validErMat = validating.errorMatrix('clase','classification');

var validAcc = validErrMat.accuracy();

print('Validation error matrix: ', validErMat);

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

```
validating                                                    JSON
▾FeatureCollection (10 elements, 0 columns)                   JSON
  type: FeatureCollection
  columns: Object (0 properties)
 ▾features: List (10 elements)
   ▾0: Feature 0_0
      type: Feature
      id: 0_0
      geometry: null
     ▾properties: Object (2 properties)
        clase: 0
        classification: 0

   ▸1: Feature 0_1
   ▸2: Feature 0_2
   ▸3: Feature 0_3
   ▸4: Feature 0_4
   ▸5: Feature 0_5
   ▸6: Feature 0_6
   ▸7: Feature 0_7
   ▸8: Feature 0_8
   ▸9: Feature 0_9
 ▸properties: Object (1 property)

Validation error matrix:                                      JSON
▾List (4 elements)                                            JSON
 ▸0: [726,0,0,2]
 ▸1: [1050,1810,0,0]
 ▸2: [0,0,2,0]
 ▸3: [295,6,0,399]

Validation overall accuracy:                                  JSON
0.6846153846153846
```

Figura 12.5: Salida de la consola de los primeros diez puntos de validación, matriz de error (error matrix) y precisión total (overall accuracy). Nótese que el total de puntos de validación es de 4290, pero se muestran solo los primeros diez para ejemplificar el tipo de salida.

Por defecto, la imagen que se obtiene de un método `.classify` lleva el nombre de 'classification', por eso la propiedad que se muestrea en el paso de `.sampleRegions` y con la que se hace la matriz de error `.errorMatrix` se llama 'classification'.

Para exportar estos resultados primero hay que pasar los datos a tipo vector. Para ello, se va a crear un vector 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 desea exportar (que podrían interpretarse como una tabla de atributos). En el caso de la matriz de error, primero se debe convertir a un arreglo mediante el método `.array`.

```
// Transformarlos en vectores
var expConfMatrix = ee.Feature(null, {
    matrix: validErMat.array()});

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

Posteriormente, se define una función (`ExportTable`) para exportar la matriz de error y el valor de la 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` solo 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 la carpeta dentro de la que se quiere guardar la tabla exportada.

```
// Definir función de exportación
var exportTable = function(feature, name){
    Export.table.toDrive({
          // Colección de vectores a exportar
          collection: feature,
          // Nombre para guardar el archivo
          description: name,
          // Carpeta donde se guardará el archivo
           folder: 'Clasificacion_RF',
          // Formato del archivo
          fileFormat: 'CSV'
    });
};

// Exportar a Google Drive
// Usar la función exportTable
exportTable(ee.FeatureCollection(expConfMatrix), 'confMatrix');

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

Reflexiones Finales

13

REFLEXIONES FINALES

Seguramente el aprendizaje de la programación en GEE será lento en un inicio, pero como cualquier otro lenguaje de programación, una vez aprendida la lógica y sintaxis básica usada en la API, el avance será mucho más rápido. Esperamos que este manual haya motivado a los usuarios a adentrarse en el uso de GEE, haya brindado herramientas para entender las cuestiones básicas sobre su funcionamiento, y que se hayan ofrecido buenos ejemplos de diferentes funcionalidades y métodos disponibles en la API, para que puedan ser utilizados en los futuros proyectos de nuestros lectores.

A pesar de que GEE es una herramienta muy poderosa para realizar análisis geomáticos, hay que tener presente que, como cualquier otro recurso, tiene sus propias limitaciones. Por lo tanto, habrá procesos que sea muy conveniente realizar en GEE, pero otros que no, tales como crear mapas (inclusive con leyendas), ya que consideramos que sus capacidades son bastante limitadas y se pueden obtener resultados personalizados más sencillamente en otros Sistemas de Información Geográfica (SIG), como QGIS. Por eso, consideramos que no todos los procesos se podrán realizar más fácilmente ni más rápidamente en GEE; sin embargo, con la práctica el usuario podrá aprender cuáles conviene realizar en GEE y cuáles no.

Referencias

14

REFERENCIAS

Amani, M., Ghorbanian, A., Ahmadi, S. A., Moghaddam, A., Mahdavi, S., Ghahremanloo, M., ... Brisco, B. (2020). **Google Earth Engine Cloud Computing Platform for Remote Sensing Big Data Applications: A Comprehensive Review.** IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 13, 5326–5350. https://doi.org/10.1109/JSTARS.2020.3021052

Arévalo, P., Bullock, E. L., Woodcock, C. E., & Olofsson, P. (2020). **A Suite of Tools for Continuous Land Change Monitoring in Google Earth Engine.** Frontiers in Climate, 2, 576740. https://doi.org/10.3389/fclim.2020.576740

Arruda, V. L. S., Piontekowski, V. J., Alencar, A., Pereira, R. S., & Matricardi, E. A. T. (2021). **An alternative approach for mapping burn scars using Landsat imagery, Google Earth Engine, and Deep Learning in the Brazilian Savanna.** Remote Sensing Applications: Society and Environment, 22, 100472. https://doi.org/10.1016/j.rsase.2021.100472

Basso, K., De Avila Zingano, P. R., & Dal Sasso Freitas, C. M. (1999). **Interpolation of scattered data: investigating alternatives for the modified Shepard method, XII Brazilian Symposium on Computer Graphics and Image Processing** (Cat. No.PR00481), 39-47. https://doi.org/10.1109/SIBGRA.1999.805606.

Breiman, L. (2001). **Random Forests.** Machine Learning, 45, 5-32.

Breiman, L., Friedman, J. H., Olshen, R. A., Stone, C. J. (1984). ***Classification and Regression Trees.*** Wadsworth, Belmont, EUA.

Campos-Taberner, M., Moreno-Martínez, Á., García-Haro, F. J., Camps-Valls, G., Robinson, N. P., Kattge, J., & Running, S. W. (2018). **Global estimation of biophysical variables from Google Earth Engine platform.** Remote Sensing, 10, 1167. https://doi.org/10.3390/rs10081167

Chuvieco, E. (1995). ***Fundamentos de Teledetección Espacial.*** **Segunda edición.** Ediciones RIALP, Madrid, España.

Cortes, C., Vapnik, V. (1995). **Support-vector networks.** Machine Learning, 20, 273–297.

Dong, J., Xiao, X., Menarguez, M. A., Zhang, G., Qin, Y., Thau, D., ... Moore, B. (2016). **Mapping paddy rice planting area in northeastern Asia with Landsat 8 images, phenologybased algorithm and Google Earth Engine.** Remote Sensing of Environment, 185, 142–154. https://doi.org/10.1016/j.rse.2016.02.016

Friedman, J. H. (2001). **Greedy function approximation: a gradient boosting machine.** Annals of Statistics *29*, 1189–1232.

Genzano, N., Pergola, N., & Marchese, F. (2020). **A Google Earth Engine tool to investigate, map and monitor volcanic thermal anomalies at global scale by means of mid-high spatial resolution satellite data.** Remote Sensing, 12(19), 3232. https://doi.org/10.3390/rs12193232

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). **Google Earth Engine: Planetary-scale geospatial analysis for everyone.** Remote Sensing of Environment, 202, 18–27. https://doi.org/10.1016/j.rse.2017.06.031

Hamunyela, E., Rosca, S., Mirt, A., Engle, E., Herold, M., Gieseke, F., & Verbesselt, J. (2020). **Implementation of BFASTmonitor Algorithm on Google Earth Engine to Support Large-Area and Sub-Annual Change Monitoring Using Earth Observation Data.** Remote Sensing, 12, 2953. https://doi.org/10.3390/rs12182953

Hansen, M. C., Potapov, P. V, Moore, R., Turubanova, S. A., Tyukavina, A., Thau, D., ... Townshend, J. R. G. (2013). **High-Resolution Global Maps of 21st-Century Forest Cover Change.** Science, 342, 850–853. https://doi.org/http://doi.org/10.1126/science.1244693

Kumar, L., Mutanga, O., Kumar, L., & Mutanga, O. (2018). **Google Earth Engine Applications Since Inception: Usage, Trends, and Potential.** Remote Sensing, 10, 1509. https://doi.org/10.3390/rs10101509

Lee, J. S. H., Wich, S., Widayati, A., & Koh, L. P. (2016). **Detecting industrial oil palm plantations on Landsat images with Google Earth Engine.** Remote Sensing Applications: Society and Environment, 4, 219–224. https://doi.org/10.1016/j.rsase.2016.11.003

Mateo-García, G., Gómez-Chova, L., Amorós-López, J., Muñoz-Marí, J., & Camps-Valls, G. (2018). **Multitemporal cloud masking in the Google Earth Engine.** Remote Sensing, 10, 1079. https://doi.org/10.3390/rs10071079

Olaya, V. (2020). **Sistemas de Información Geográfica.** CreateSpace Independent Publishing Platform, España. https://volaya.github.io/libro-sig/

Oshiro, T. M., Perez, P. S., Baranauskas, J. A. (2012). **How Many Trees in a Random Forest?.** En: Perner, P. (eds) Machine Learning and Data Mining in Pattern Recognition. MLDM 2012. Lecture Notes in Computer Science, 7376. Springer, Berlin, Alemania. https: //doi.org/10.1007/978-3-642-31537-4_13

Perilla, G. A., & Mas, J. F. (2020). **Google Earth Engine (GEE): una poderosa herramienta que vincula el potencial de los datos masivos y la eficacia del procesamiento en la nube.** Investigaciones Geográficas, 101, e59929. https://doi.org/10.14350/rig.59929

Perilla, G. A., & Mas, J. F. (2019). **High-resolution mapping of protected agriculture in Mexico, through remote sensing data cloud geoprocessing.** European Journal of Remote Sensing, 52:1, 532-541. https://doi.org/10.1080/22797254.2019.1686430

Perilla, G. A., Cruz-Rodríguez, C. A., Muñoz, C. J., Olaya-Herrera, H, Jimenez, L. R., Arango, H. M., Escobar, D. A., Suarez-Valencia, E., & Noguera-Urbano, E. A. (2022). **Multi-temporal habitat models using Machine learning in Google Earth Engine.** Manuscrito sin publicar.

Phillips, S. J., Dudík, M., Schapire, R. E. (2004). **A maximum entropy approach to species distribution modeling.** En: Proceedings of the twenty-first international conference on Machine learning (p. 83).

Quinlan, J. R. (1986). **Induction of Decision Trees.** Machine Learning 1, 81–106. https://doi.org/10.1023/A:1022643204877

Radočaj, D., Obhođaš, J., Jurišić, M., Gašparović, M. (2020). **Global open data remote sensing satellite missions for land monitoring and conservation: A review.** Land, 9, 1–24. https://doi.org/10.3390/land9110402

Ravanelli, R., Nascetti, A., Cirigliano, R., Di Rico, C., Leuzzi, G., Monti, P., & Crespi, M. (2018). **Monitoring the Impact of Land Cover Change on Surface Urban Heat Island through Google**

Earth Engine: Proposal of a Global Methodology, First Applications and Problems. Remote Sensing, 10, 1488. https://dx.doi.org/10.3390/rs10091488

Sazib, N., Mladenova, I., & Bolten, J. (2018). **Leveraging the google earth engine for drought assessment using global soil moisture data.** Remote Sensing, 10, 1265. https://doi.org/10.3390/rs10081265

Slagter, B., Tsendbazar, N.-E., Vollrath, A., & Reiche, J. (2020). **Mapping wetland characteristics using temporally dense Sentinel-1 and Sentinel-2 data: A case study in the St. Lucia wetlands, South Africa.** International Journal of Applied Earth Observation and Geoinformation, 86, 102009. https://doi.org/10.1016/j.jag.2019.102009

Solórzano, J. V., Gallardo-Cruz, J. A., & Peralta-Carreta, C. (2020). **Potencial del acervo de imágenes Landsat disponible en Google Earth Engine para el estudio del territorio mexicano.** Investigaciones Geográficas, 101, e59821. https://doi.org/10.14350/rig.59821

Solórzano, J. V., Mas, J. F., Gao, Y., & Gallardo-Cruz, J. A. (2020). **Patrones espaciotemporales de las observaciones de Sentinel-2 a nivel de imagen y píxel sobre el territorio mexicano entre 2015 y 2019.** Revista de Teledetección, 56(Número especial), 103–115. https://doi.org/10.4995/raet.2020.14044

Trianni, G., Angiuli, E., Lisini, G., & Gamba, P. (2014). **Human settlements from Landsat data using Google Earth Engine.** Geoscience and Remote Sensing Symposium, 1473–1476. https://doi.org/10.1109/IGARSS.2014.6946715

Wagle, N., Acharya, T. D., Kolluru, V., Huang, H., & Lee, D. H. (2020). **Multi-Temporal Land Cover Change Mapping Using Google Earth Engine and Ensemble Learning Methods.** Applied Sciences, 10, 8083. https://doi.org/10.3390/app10228083

Webb, G. I. (2011). ***Naïve Bayes****. En: Sammut, C., Webb, G. I. (eds)* **Encyclopedia of Machine Learning.** Springer, Boston, EUA. https://doi.org/10.1007/978-0-387-30164-8_576

Williamson, M. J., Tebbs, E. J., Thompson, H. J., Dawson, T. P., Head, C. E., & Jacoby, D. M. (2021). **Application of Earth Observation Data and Google Earth Engine for Monitoring Coral Reef Exposure to Environmental Stressors.** https://doi.org/10.20944/preprints202106.0473.v1

Xiong, J., Thenkabail, P. S., Gumma, M. K., Teluguntla, P., Poehnelt, J., Congalton, R. G., . . . Thau, D. (2017). **Automated cropland mapping of continental Africa using Google Earth Engine cloud computing** ISPRS Journal of Photogrammetry and Remote Sensing, 126, 225–244. https://doi.org/10.1016/j.isprsjprs.2017.01.019

Zhang, X., Zeraatpisheh, M., Rahman, M. M., Wang, S., & Xu, M. (2021). **Texture is important in improving the accuracy of mapping photovoltaic power plants: A case study of Ningxia Autonomous Region, China.** Remote Sensing, 13(19), 3909. https://doi.org/10.3390/rs13193909

ÍNDICE ALFABÉTICO

accuracy, 173, 181
add, 119
addBands, 125, 154
aggregate, 95, 142
and, 130, 139, 154
apply, 105
area, 64, 77, 100

bandNames, 114
bitwiseAnd, 122, 154
buffer, 77

ceil, 98
classify, 173, 178
clip, 118, 129, 158, 176
coordinates, 64
count, 144, 157

dissolve, 78
divide, 101, 119
draw, 87

ee.Algorithms, 60
ee.Array, 60
ee.Classifier, 61, 171
ee.Classifier.smileRandomForest, 177
ee.Clusterer, 61
ee.Date, 58
ee.Dictionary, 56
ee.Feature, 60, 72
ee.FeatureCollection, 60, 81
ee.Filter, 61
ee.Filter.and, 91
ee.Filter.eq, 105, 112, 151
ee.Filter.gte, 91
ee.Filter.intersects, 105
ee.Filter.lte, 151
ee.Filter.notNull, 103
ee.Filter.rangeContains, 91
ee.Filter.stringContains, 91
ee.Filter.withinDistance, 105
ee.Geometry, 61, 64
ee.Image, 60, 111
ee.ImageCollection, 60, 134
ee.Join, 61, 104
ee.Join.saveAll, 105
ee.List, 55
ee.Number, 54, 100
ee.Reducer, 62, 100, 148
ee.Reducer.first, 103
ee.Reducer.mean, 100, 156, 176
ee.Reducer.stdDev, 156
ee.Reducer.sum, 102
ee.String, 54
ee.Terrain, 62
eq, 120
errorMatrix, 173, 181
Export, 126
Export.image, 32
Export.image.toDrive, 131, 161
Export.table, 34, 107
Export.table.toAsset, 107
Export.table.toDrive, 108, 183
Export.video, 35
expression, 119

filter, 91, 129, 138, 150
filterBounds, 89, 102, 175

filterDate, 90, 138, 151
filterMetadata, 89, 138
first, 94, 136, 142, 151, 155
fromImages, 135
function, 50, 90, 97, 101, 154, 175

get, 75, 100
gt, 120
gte, 120, 130

intersection, 76
inverseDistance, 165

kriging, 168

limit, 82
lt, 120
lte, 120, 130

map, 97, 146, 155, 175
Map.addLayer, 28, 72, 87, 114, 137, 154
Map.centerObject, 30
mask, 123, 125
merge, 145
mosaic, 148
multiply, 98, 119

neq, 120
normalizedDifference, 119, 153

or, 139

paint, 87
parse, 100
perimeter, 77
pow, 100
print, 27
projection, 112
propertyNames, 75, 95, 112, 135

qualityMosaic, 148

randomPoints, 93, 98
reduce, 100, 147, 156, 176
reduceColumns, 100
reduceRegion, 164
reduceRegions, 102, 164
reduceToImage, 103
reduceToVectors, 125
rename, 118, 130, 153

sample, 163
sampleRectangle, 163
sampleRegions, 163, 177, 181
select, 74, 75, 93, 115, 139, 160
set, 74, 90, 116
size, 82, 144
sort, 141
sqrt, 100
subtract, 100, 101, 119

toDictionary, 75
train, 172, 177
type, 64

ui.Chart.feature, 30
ui.Chart.image, 31
ui.Chart.image.byClass, 179
ui.Chart.image.histogram, 130
union, 78
updateMask, 123, 130, 154

Cómo usar Google Earth Engine y no fallar en el intento, editado por el **Centro de Investigaciones en Geografía Ambiental** y el **Instituto de Investigación de Recursos Biológicos Alexander von Humboldt** se publicó el 20 de octubre de 2022. Para su formación se utilizaron las **tipografías** Aleo y Consolas en 9, 10, 10.5 y 12 pt. El cuidado de la **edición** estuvo al cuidado de Jonathan Vidal Solórzano Villegas, Gabriel Alejandro Perilla Suárez, Mónica Braun y Jorge Andrés Trinidad González. El **diseño editorial y diagramación** estuvo a cargo de Jonathan Vidal Solórzano Villegas y Laura Perilla Suárez.

Google Earth Engine es una herramienta enfocada en facilitar el procesamiento de grandes volúmenes de información geoespacial. Por ello, ha permitido realizar análisis previamente imposibles de ejecutar en una computadora personal y acortar enormemente el tiempo de procesamiento requerido para realizar diferentes tipos de análisis. Este libro pretende fomentar el uso de esta poderosa herramienta entre usuarios hispanoparlantes y brindar una traducción al español de los principales tipos de objetos y métodos de esta plataforma. Este manual será de utilidad tanto para personas sin conocimiento de Google Earth Engine hasta aquellas con un nivel intermedio de uso, ya que aborda desde aspectos fundamentales hasta algunos consejos. También, ofrece ejemplos, códigos comentados, guías ilustradas, descripciones de la lógica de programación, detalles del funcionamiento interno, identificación de los errores más comunes y sus soluciones, y enlaces de utilidad que ayudarán al lector a aprender a utilizar Google Earth Engine o mejorar en sus capacidades de uso. Se recomienda enérgicamente que la lectura de este manual se haga en paralelo a la exploración de la plataforma y ejecutando todos los ejercicios. Finalmente, se espera que este libro permita aumentar el uso de Google Earth Engine y promover su uso para el desarrollo de diferentes tipos de investigaciones y aplicaciones.

UNAM

INSTITUTO HUMBOLDT COLOMBIA

ciga
centro de investigaciones en geografía ambiental · UNAM