



# Manejo de datos vectoriales

# Contenidos

Manejo de datos vectoriales: Manejo de geometrías y generación de features. Creación y administración de colecciones de features. Carga y visualización de vectores utilizando Google Fusion Table (FT). Manejo de iteraciones sobre colecciones de features. Exportar como tabla de datos. Realizar gráficos.

# Manejo de geometrías y generación de features

Earth Engine maneja datos vectoriales con el tipo `ee.Geometry`. La especificación `GeoJSON` describe en detalle el tipo de geometrías soportadas por Earth Engine, incluyendo:

- **Point**: una lista de coordenadas en alguna proyección,
- **LineString**: una lista de puntos,
- **LinearRing**: una LineString cerrada
- **Polygon**: una lista de LinearRings donde la primera es una cáscara y los anillos subsiguientes son agujeros.

Earth Engine también soporta **MultiPoint**, **MultiLineString** y **MultiPolygon**. GeoJSON GeometryCollection también es compatible, aunque tiene el nombre MultiGeometry dentro de **EE**.

# Creación de Geometrías

Vamos a crear y mapear una geometría

```
var point = ee.Geometry.Point([-60.54, -31.85]);
```

`ee.Geometry.Point` es la llamada a la API Earth Engine que recibe dos parámetros una lista `ee.List([])` y el segundo que es opcional una proyección que puede ser especificada como código EPSG [1].

El valor predeterminado es EPSG:4326 (WGS84 Lat/Lon)

# Creación de Geometrías

La definición de esa geometría (según la definición [GeoJSON](#)) sería:

```
{  
  "geodesic": true, // WGS84 Lat/Lon  
  "type": "Point",  
  "coordinates": [  
    -60.54,  
    -31.85  
  ]  
}
```

# Mapear e imprimir un objeto geometry

De esa forma los datos son **almacenados** y **utilizados** internamente por la librería.

Para poder ver en el [Code Editor](#) el formato nativo podemos hacer **print(point)**; y luego en la consola hacer clic en JSON.

También podemos agregar la geometría en el mapa utilizando la siguiente instrucción:

```
Map.addLayer(point, {'color': '00FF11'} , 'Punto');
```

**Tips:** Si quiere modificar el color de la geometría el formato hexadecimal de [colores html](#) puede obtenerse desde Google realizando la búsqueda con [picker color](#).

# Geometría de línea

El resto de las geometrías se construyen de la misma forma, veamos:

```
var lineString = ee.Geometry.LineString([
    [-63, -36],
    [-60.54, -31.85],
    [-58, -28],
    [-63, -27]]);
```

`ee.Geometry.LineString` recibe una lista de puntos y parámetros opcionales.

```
//Agreganos la geometría al mapa
Map.addLayer(lineString, {'color': 'CC0011'} , 'Linea');
```

## TIPS: Centrar el mapa

Podemos centrar el mapa en una geometría o *feature* particular, eso lo podemos hacer con `Map.centerObject` así:

```
// 1er elemento el objeto georeferenciado  
// 2do elemento el nivel de zoom 0 más lejos  
Map.centerObject(lineString, 7);
```

Ver ejemplo [aquí](#)



# Geometría LinearRing

```
var linearRing = ee.Geometry.LinearRing(  
  [[-63, -36.09], [-59.54, -31.85],  
   [-58, -28], [-63, -25],  
   [-64, -27], [-63, -36]]);
```

`ee.Geometry.LinearRing` recibe una lista de puntos que a diferencia de `LineString` comienza y termina con el mismo punto para poder cerrar el anillo. También tiene parámetros opcionales.

Y si, al mapa!!

```
Map.addLayer(linearRing, {'color': 'b227b0'} , 'Anillo');
```

# Geometría Rectángulo

```
var rectangle = ee.Geometry.Rectangle([-62, -33, -59, -31]);
```

`ee.Geometry.Rectangle` recibe una lista con esquinas mínimas y máximas del rectángulo, como una lista de dos puntos en formato de coordenadas GeoJSON 'Point' o una lista de dos `ee.Geometry` que describen un punto, o una lista de cuatro números en el orden **xMin, yMin , xMax, yMax**.

```
Map.addLayer(rectangle, {'color': 'fbff23'} , 'Rectángulo');
```

# Geometría Polígono

```
var vertices = [  
    [-62.935, -34.415],  
    [-61.745, -34.411],  
    [-61.388, -34.068],  
    [-62.663, -34.075] ];  
  
var poligono = ee.Geometry.Polygon( vertices );
```

[ee.Geometry.Polygon](#) recibe una lista de anillos que definen los límites del polígono. Puede ser una lista de coordenadas en el formato 'Polygon' de GeoJSON, o una lista de `ee.Geometry` que describe un `LinearRing`. El resto de los parámetros son similares al resto de las geometrías.

```
Map.addLayer(poligono, { 'color': '16a322' } , 'Polígono');
```

# Geometrías Multiparte

Una geometría individual puede consistir en múltiples geometrías.

```
var multiPoint = ee.Geometry.MultiPoint(  
    [  
        [-62.319, -32.856],  
        [-62.528, -32.944],  
        [-62.418, -33.109],  
        [-62.193, -33.008],  
        [-62.166, -32.805]  
    ]  
);  
  
Map.addLayer(multiPoint, {'color': '16a322'}, 'multiPoint');
```

Para dividir una Geometría de varias partes en sus geometrías constitutivas, use **geometry.geometries()**.

# Geometrías multiparte

Vamos a imprimir los puntos de la geometría, así que al objeto `multiPoint` le vamos a pedir todas las geometrías que lo componen.

```
var las_geometrias = multiPoint.geometries();  
print(las_geometrias);
```

1. ¿Qué tipo de dato es la variable **las\_geometrias**?
2. ¿Cómo puedo recuperar una de las geometrías contenidas en **las\_geometrias**?

# Desafío 1

Convierta los puntos existentes en multiPoint a un LinearRing.

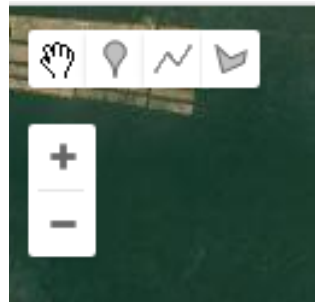


# Crear Geometrías desde el mapa

Existe una forma muy práctica de dibujar Geometrías desde el mismo mapa del Code Editor.

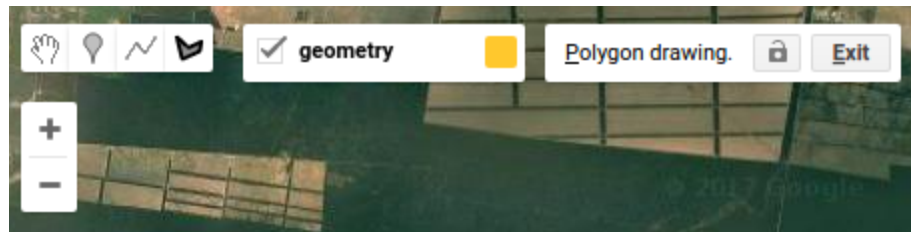
Veamos un ejemplo:

- Las opciones para dibujar están ubicadas en el sector superior izquierdo del mapa.
- Las herramientas disponibles permiten activar el dibujo de geometrías múltiples de: puntos, líneas y polígonos.  
Para dejar de dibujar se hace clic en la mano de la izquierda.



# Crear Geometrías desde el mapa

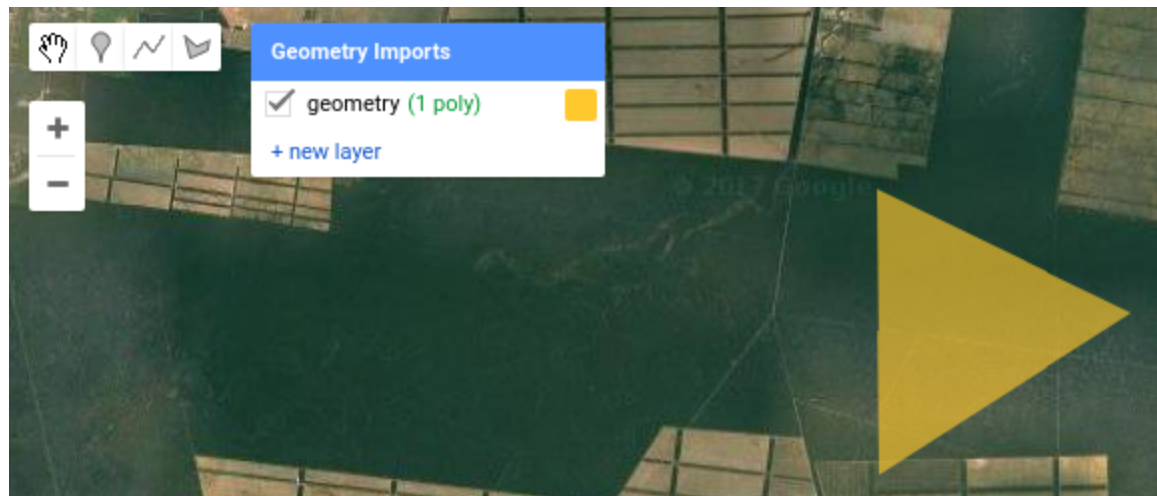
- Una vez que se activa la herramienta esta se habilita para poder dibujar.
- Se asigna un color al azar y cada figura que se trace formará parte de una geometría múltiple.





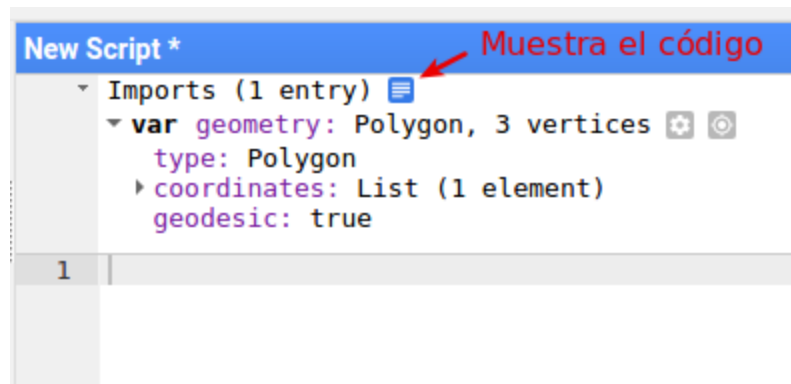
# Crear Geometrías desde el mapa

- Es posible incorporar desde la sección de **Geometry Imports** una nueva capa.
- Esta se instancia como una nueva variable de la clase Geometry.XXXX.



# Sección *Imports*

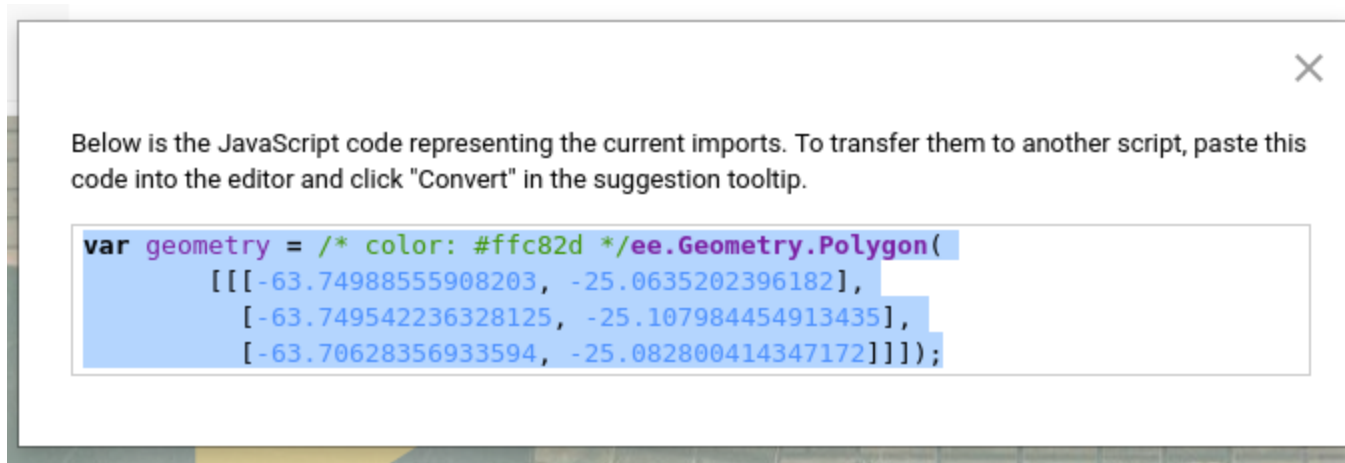
- Las capas de geometrías que se van incorporando serán ubicadas en la sección **Imports** del editor de código fuente.
- Estos objetos son mostrados de manera formateada. Pero haciendo clic en el ícono azul se muestra el código fuente correspondiente para la creación de la geometría.



Todo lo que esté en **Imports** lo puedo compartir con el link.

# Código fuente del *Import*

El código fuente generado puede copiarse y pegarse en el script que se está escribiendo.



**Ojo:** Solo funciona con Chrome!

# Geometrías Geodésicas y Planas

- Una geometría creada en Earth Engine es **geodésica** (*los bordes son la trayectoria más corta en la superficie de una esfera*) o **planar** (*los bordes son el camino más corto en un plano cartesiano bidimensional*).

En la configuración predeterminada de la instanciación de un objeto `ee.Geometry.XXXXX` este se crea como EPSG:4326, es decir, será una geometría geodésica.

# Geometrías Geodésicas y planas

```
var PoligonoGeo = ee.Geometry.Polygon([  
  [  
    [-71.411, -39.470], [-57.128, -39.402],  
    [-57.304, -33.394], [-70.751, -33.358],  
    [-71.411, -39.470]]  
]);
```

Esa misma geometría puede ser expresada en el plano, ya sea cuando se crea o puede ser convertida:

```
var PoligonoPlano = ee.Geometry(PoligonoGeo, null, false);
```

Veámos cómo se ven estas geometrías en el mapa:

```
Map.centerObject(PoligonoGeo);  
Map.addLayer(PoligonoGeo, {color: 'FF0000'}, 'Geodésico');  
Map.addLayer(PoligonoPlano, {color: '000000'}, 'Plano');
```

# Operaciones con Geometrías

Earth Engine admite una amplia variedad de operaciones en objetos Geometry. Estos incluyen operaciones unarias en geometrías individuales tales como calcular un buffer, centroide, bounding box, perímetro, envolvente convexa, etc.

# Operaciones con Geometrías

Utilizando la definición del polígono de la anterior vamos a realizar algunas operaciones de geometrías:

- Cálculo de área en KM<sup>2</sup>

```
print('Área: ', poligono.area().divide(1000 * 1000));  
// Todos los valores de mediciones de distancias  
// vienen expresados en metros.
```

# Operaciones con Geometrías

- Longitud de perímetro en KM

```
print('Perímetro: ', poligono.perimeter().divide(1000));
```

- Mostrar el GeoJSON 'type'.

```
print('Geometry type: ', poligono.type());
```

- Mostrar la lista de coordenadas.

```
print('Coordenadas: ', poligono.coordinates());
```

- Muestra true/false si las coordenadas son o no geodésicas.

```
print('¿Está en coordenadas geodésicas? ',  
      poligono.geodesic());
```



# Operaciones con Geometrías

- Muestra el BBOX de una geometría

```
print('Bounding Box', poligono.bounds());  
// bounds retorna el rectángulo que envuelve a la geometría
```

- Calcular el buffer de un polígono

```
var buffer = poligono.buffer(5000);  
// La distancia del buffer está expresada en metros.
```

- Calcular el centroide de un polígono..

```
var centroid = poligono.centroid();
```

# Operaciones con Geometrías

Ahora podemos mapear algunas de estas operaciones:

```
Map.addLayer(buffer, {'color': '0be51e'}, 'buffer');  
Map.addLayer(poligono, {}, 'el polígono');  
Map.addLayer(centroid, {'color': 'e5280b'}, 'centroide');  
Map.centerObject(buffer, 7);
```

# Operaciones con Geometrías: Binarias

Las operaciones que hemos realizado han sido todas con operadores unarios, donde a la geometría le pedimos (o le calculamos) algo. Ahora vamos a probar algunos operadores entre geometrías.

```
var poli1 = ee.Geometry.Polygon(  
  [[[-63.62113952636719, -25.129433436071757],  
    [-63.572044372558594, -25.128811779454853],  
    [-63.57135772705078, -25.054501051619468],  
    [-63.622169494628906, -25.05512308589585]]]);  
  
var poli2 = ee.Geometry.Polygon(  
  [[[-63.645172119140625, -25.061965254565465],  
    [-63.65753173828125, -25.13005508952488],  
    [-63.59367370605469, -25.128811779454853],  
    [-63.59367370605469, -25.061343255018567]]]);
```

# Operaciones con Geometrías

Calculamos la unión de las dos geometrías. Donde el primer parámetro es la geometría que se quiere unir y el segundo es un margen de error. **ErrorMargin** es la cantidad máxima de error tolerada al realizar cualquier reproyección necesaria, el valor está expresado en metros.

```
var poli1Upoli2 = poli1.union(poli2, ee.ErrorMargin(1));  
Map.addLayer(poli1Upoli2, {color: 'FF00FF'}, 'Unión');  
  
// Centramos el mapa en la unión  
Map.centerObject(poli1Upoli2, 12);
```

# Intersección de geometrías

Calculamos la **intersección** de los dos polígonos con la función `intersection`

```
var intersection = poli1.intersection(  
    poli2,  
    ee.ErrorMargin(1));  
Map.addLayer(intersection,  
    {color: '00FF00'}, 'Intersección');
```

# Diferencia entre geometrías

Calculamos la diferencia entre el polígono 1 y el polígono 2. Es el área de la primer geometría que no comparte con la segunda.

```
var diff1 = poli1.difference(poli2, ee.ErrorMargin(1));  
  
Map.addLayer(  
    diff1,  
    {color: 'FFFF00'},  
    'Diferencia( P1 - P2)');
```

# Diferencia Simétrica

Calculamos la diferencia simétrica, esta se define como el área de la geometría A y el área de la geometría B excepción el área común a ambas.

```
var dif_sim = poli1.symmetricDifference(
    poli2,
    ee.ErrorMargin(1));
Map.addLayer(dif_sim,
    {color: '000000'},
    'Diferencia Simétrica');

// Mapeamos las dos geometrías con las que
// hicimos los ejemplos.
Map.addLayer(poli1, {color: 'FF0000'}, 'Polígono 1');
Map.addLayer(poli2, {color: '0000FF'}, 'Polígono 2');
```

## Desafío 2

Verifique si la recta que pasan por los puntos  **$[-63.635, -25.051]$**  y  **$[-63.617, -25.146]$**  intersecta la intersección (valga la redundancia) de las geometrías utilizadas previamente (poli1 y poli2).



## Desafío 3:

Ahora compruebe si el punto definido abajo está contenido en la geometría que resultó de la diferencia simétrica.

```
var punto = ee.Geometry.Point([-63.6084366, -25.0803128]);
```

# Creación de Features

- Un Feature de Earth Engine se define como un GeoJSON Feature.
- En pocas palabras, es un objeto con una propiedad de tipo **Geometry** y una **propiedad** de properties que almacena un diccionario de otras propiedades.

# Creación de Features

Entonces, necesitamos un objeto **Geometry** y opcionalmente un diccionario con los atributos de ese Feature.

```
// La geometría  
var poligono = ee.Geometry.Polygon(  
    [[[-63.33892822265625, -25.150878651548442],  
      [-63.33824157714844, -25.17791290009134],  
      [-63.31043243408203, -25.17760219565173],  
      [-63.31043243408203, -25.15025710411473]]]);
```

# Declaración de Features

```
var miFeature = ee.Feature(poligono,  
    {variable_1: 100,  
     variable_2: 'Hola'});
```

Al igual que con las geometrías podemos enviarlos a la consola utilizando print o mostrarlos en el mapa.

```
print(miFeature);  
Map.addLayer(miFeature, {}, 'Mi Feature!');  
Map.centerObject(miFeature, 12);
```

La geometría del Feature puede ser nula y se podría crear el Feature solo con un diccionario:

```
var dict = {distancia: ee.Number(10).add(150),  
            lugar: 'Chivilcoy'};  
var featureSinGeo = ee.Feature(null, dict);
```

# Operaciones básicas sobre Features

Los Features tienen las mismas funcionalidades para gestionar sus geometrías que los objetos Geometry. Además, poseen otros métodos **setters & getters** para el manejo de las propiedades.

```
// Creamos un Feature y le incorporamos dos atributos  
// Nombre y Altura  
var feature_ejemplo =  
    ee.Feature( ee.Geometry.Point(  
        [-63.2951545715332, -25.163930416282465]))  
        .set( 'Nombre', 'Eldes Monte')  
        .set( 'Altura', 100);
```

# Operaciones básicas sobre Features II

```
// Recupero una propiedad del feature
var nombre = feature_ejemplo.get('Nombre');
print(nombre);

// Asigno una nueva propiedad
feature_ejemplo = feature_ejemplo.set('Población', 75000);

// Sobreescribo un nuevo diccionario
var newDict = {Nombre: 'El Lote', Altura: 300};
var feature_ejemplo = feature_ejemplo.set(newDict);

// Se muestran los resultados
print(feature_ejemplo);

Map.addLayer(feature_ejemplo, {}, 'Ejemplo 2');
```

# Creación y administración de FeatureCollection

- Los grupos de features relacionados se pueden combinar en una [FeatureCollection](#), para permitir operaciones adicionales en todo el conjunto tales como: filtrado, clasificación y renderizado.
- Además de simples features (geometría + propiedades), FeatureCollection también puede contener otras colecciones.

# FeatureCollection a partir de lista de Features

Podemos construir un FeatureCollection a partir de una lista de features.

```
// Las geometrias pueden ser distintas
var features = [
  ee.Feature(ee.Geometry.Point(-62.709, -31.428),
    {Estación: 'La Francia'}),
  ee.Feature(ee.Geometry.Point(-61.248, -31.475),
    {Estación: 'Pilar'}),
  ee.Feature(ee.Geometry.Point(-61.765, -31.840),
    {Estación: 'Sastre'}),
  ee.Feature(ee.Geometry.Point(-62.534, -31.858),
    {Estación: 'Las Varas'})];
var fc_desdeUnaLista = ee.FeatureCollection(features);

print(fc_desdeUnaLista);
Map.addLayer(fc_desdeUnaLista, {}, 'FC_Puntos');
Map.centerObject(fc_desdeUnaLista);
```



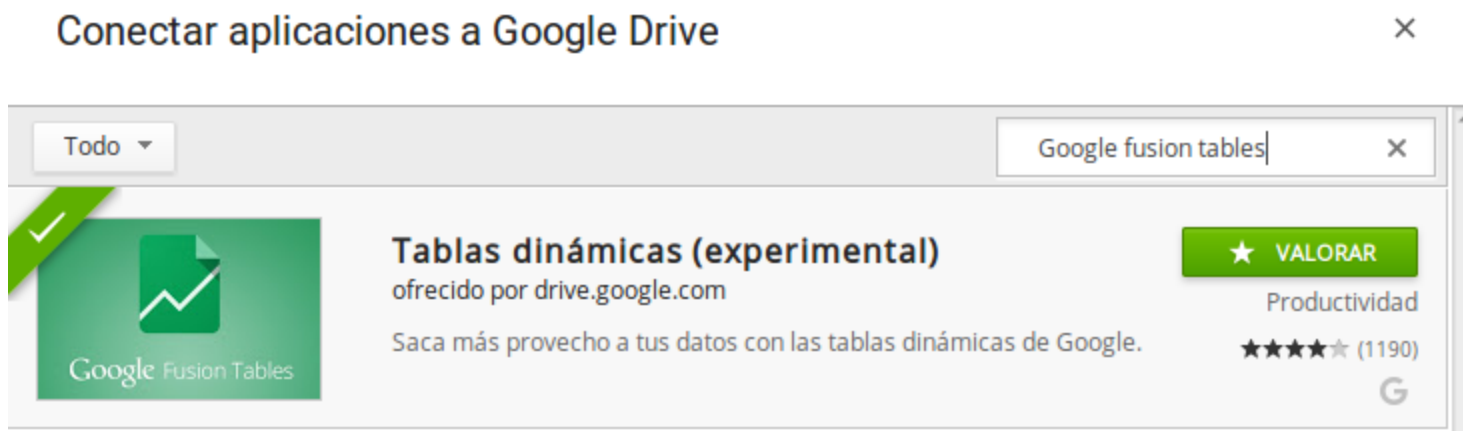
# FeatureCollection a partir de Google Fusion Table

También podemos incorporar un FeatureCollection a partir de un Google Fusion Table. Google Fusion Tables (GFT) es un servicio de Google que permite manejar tablas. Esto habilita la importación de archivos vectoriales a la plataforma Google Earth Engine (GEE) a través de archivos vectoriales en formato KML.

# Crear un nuevo Fusion Table I

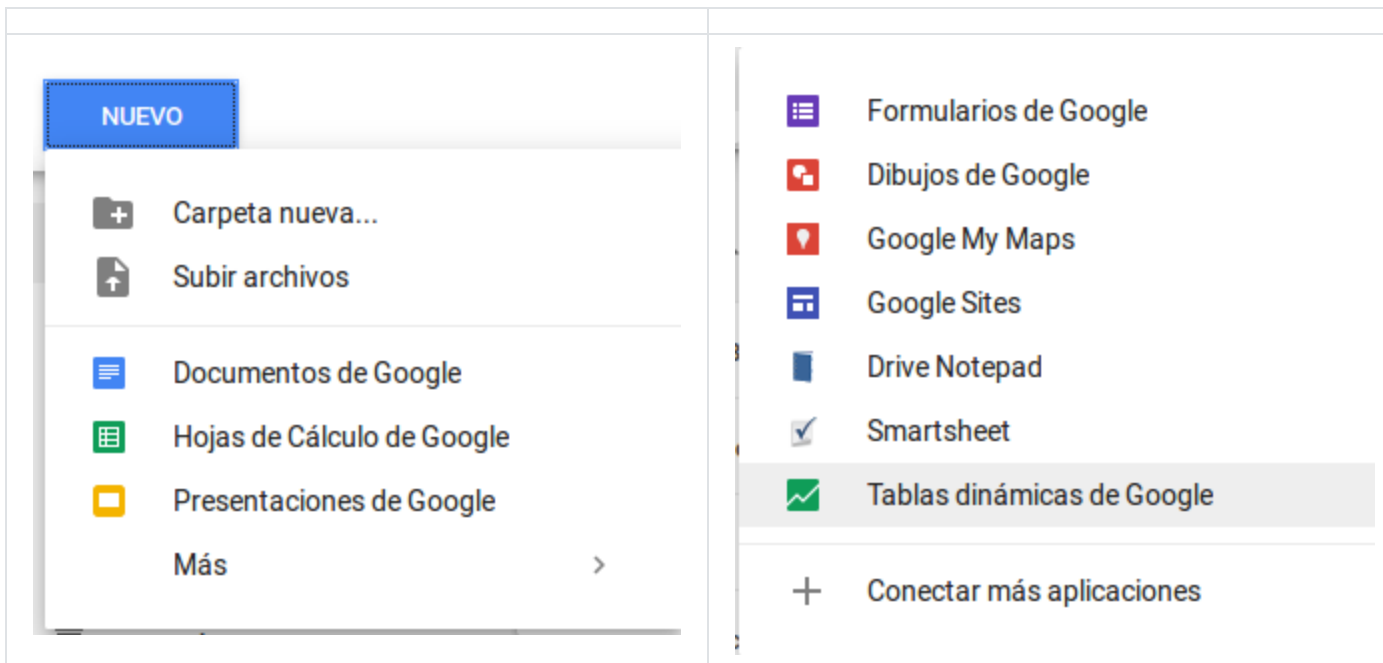
Estas tablas las gestionamos desde [Google Drive](#), la utilidad debe estar habilitada si no aparece en el menú NUEVO. La habilitación se realiza desde:

NUEVO > Más > + Conectar más aplicaciones



# Crear un nuevo Fusion Table II


Una vez habilitada vamos a poder subir una nueva tabla a través de la opción NUEVO > Más > Google Fusion Table (o Tablas dinámicas de Google en español).





# Crear un nuevo Fusion Table III

La creación del Fusion Table importamos desde el sistema de archivos un documento .kml y luego hacemos clic en Next.

Import new table

 From this computer


 Google Sheets

 Create empty table

Examinar... chacoo.kml

You can upload spreadsheets, delimited text files (.csv, .tsv, or .txt), and Keyhole Markup Language files (.kml) [Learn more](#)

Or search public data tables



New to Fusion Tables?

Take a peek! [Play with a data set](#) or [try a tutorial](#).

Cancel

« Back

Next »

# Crear un nuevo Fusion Table IV

Nos va a mostrar un preview del archivo donde podremos verificar los nombres de los atributos y el tipo de geometría que estamos subiendo.

Luego, Next.

Column names are in row 1

1	descr...	name	NOM...	ORG...	FECHA	geom...
2			Chaco Hmedo	Secretara de Recursos Naturales y Desarrollo Sust	Octubre 2007	<Polygon> <outerBo... <LinearRi... <coordin...
3			Chaco Hmedo	Secretara de Recursos Naturales y Desarrollo Sust	Octubre 2007	<Polygon> <outerBo... <LinearRi... <coordin...
4			Chaco Seco	Secretara de Recursos Naturales y	Octubre 2007	<Polygon> <outerBo... <LinearRi... <coordin...

Rows before the header row will be ignored.

---

**New to Fusion Tables?**

Take a peek! [Play with a data set](#) or [try a tutorial](#).

Cancel « Back Next »

# Crear un nuevo Fusion Table V

En esta sección podremos editar algunos parámetros de la nueva tabla como el nombre, la descripción y algunos permisos básicos. Damos clic en Finish.

Import new table

Table name

Región Chaqueña

Allow export



Attribute data to



Attribution page link

Description

Imported at Sat Jun 17 13:22:18 PDT 2017 from chaco.kml.

For example, what would you like to remember about this table in a year?

New to Fusion Tables?

Cancel

« Back

Finish

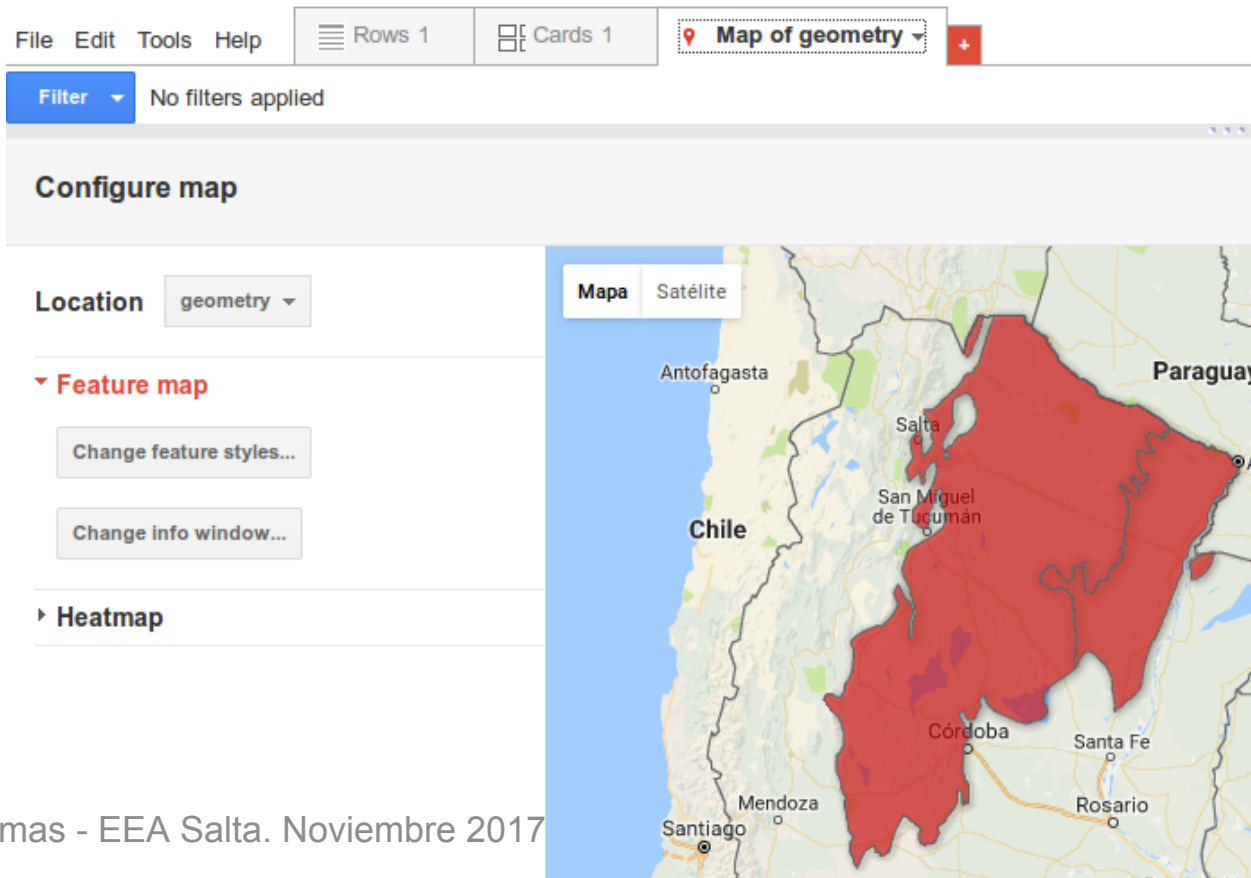
# Crear un nuevo Fusion Table VI

Ahora si ya está disponible la nueva tabla. Si vamos a la solapa Map of geometry podemos ver desplegada sobre un mapa de Google la columna geometría que se subió con el archivo kml.

## Región Chaqueña

Imported at Sat Jun 17 13:22:18 PDT 2017 from chaco.kml.

Edited at 17:22



# Crear un nuevo Fusion Table VII

En la opción Share se puede ajustar las opción de seguridad de la nueva capa que fue subida al GFT.

Compartir mediante enlace

☐  **Activado: público en la Web**  
Cualquier usuario de Internet puede encontrar el elemento y acceder a él sin necesidad de iniciar sesión.

☒  **Activado: cualquier usuario que tenga el enlace**  
Cualquier usuario que disponga del enlace puede acceder al elemento sin necesidad de iniciar sesión.

☐  **Desactivado: determinados usuarios**  
Compartido con ciertos usuarios.

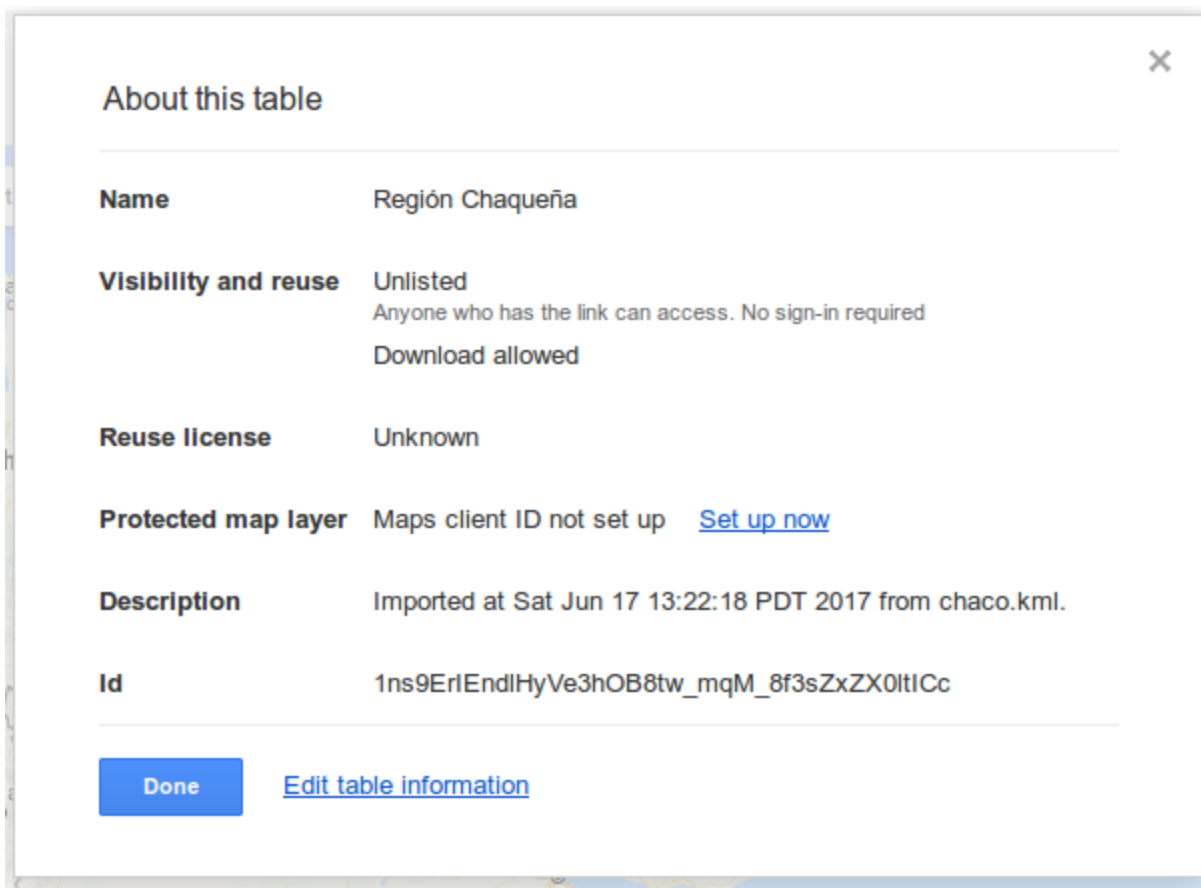
Acceso: Cualquier usuario (no requiere acceso) Puede ver

[Guardar](#) [Cancelar](#)



# Crear un nuevo Fusion Table VIII

Para poder importar este archivo a GEE necesitamos el ID de la tabla, para ello vamos a File y seleccionamos About this table. Ahí encontraremos el identificador (Id).



# Crear un nuevo Fusion Table IX (La última :D)

También en la opción “Enlace para compartir” disponible en Share, tenemos disponible el identificador de la GFT a través del cual vamos a vincular la tabla con nuestro script.

Configuración para compartir

Enlace para compartir

m/fusiontables/DataSource?docid=1ns9ErIEndlHyVe3hOB8tw\_mqM\_8f3sZxZX0ItICc

# Crear un FeatureCollection desde GFT

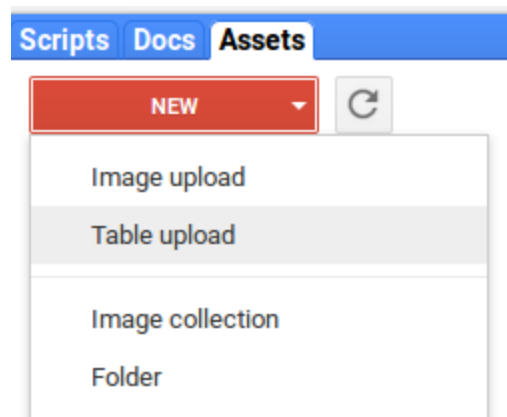
Para cargar una FeatureCollection desde una Fusion Table, proporcione al constructor (ee.FeatureCollection) el ID de tabla agregado a **ft:**. Por ejemplo:

```
var key = 'ft:1ns9ErIEndlHyVe3h0B8tw_mqM_8f3sZxZX0ltICc';  
var desdeFT = ee.FeatureCollection(key);  
print(desdeFT);  
Map.addLayer(desdeFT, {}, 'Región Chaqueña');  
Map.centerObject(desdeFT);
```

# Creación de un FeatureCollection a partir de Tablas

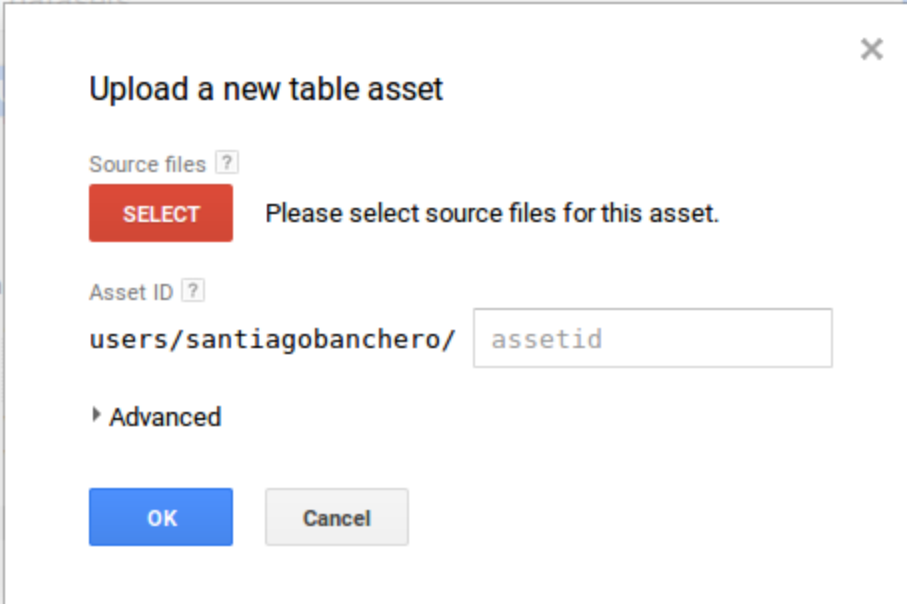
Es posible instanciar un FC utilizando tablas que tenemos almacenadas en un Assets. Para crear esta nueva tabla podemos utilizar un Shapefile o un .zip que contenga todos los archivos que componen el Shapefile. El tamaño máximo permitido es 10GB.

Los pasos para subir un Shapefile son seleccionar desde Assets NEW > Table Upload.



# Selección del Shapefile

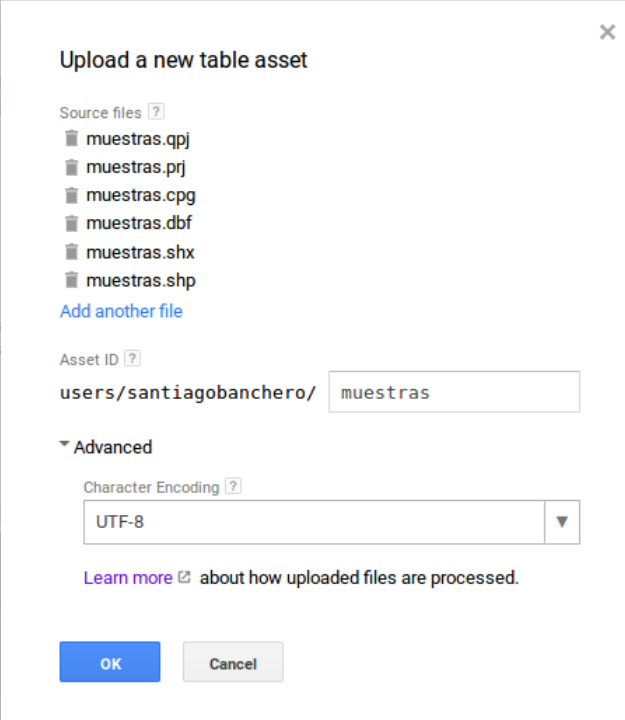
Luego seleccionamos desde nuestro sistema de archivo el shapefile y todos los archivos que lo componen (.dbf, .shp, .shx, .prj) (podemos seleccionar el .zip también).



The screenshot shows a dialog box titled "Upload a new table asset" with a close button (X) in the top right corner. Inside the dialog, there is a section for "Source files" with a help icon (?) and a red "SELECT" button. To the right of the button, it says "Please select source files for this asset." Below this, there is an "Asset ID" field with a help icon (?) and a text input box containing "assetid". The path "users/santiagobanchero/" is visible to the left of the input box. At the bottom, there is a section labeled "Advanced" with a right-pointing arrow. Below "Advanced" are two buttons: "OK" (blue) and "Cancel" (gray).

# Selección del Shapefile

Por default toma el nombre del shp para la tabla pero se puede editar. Además, podemos indicar la codificación de caracteres que posee el shp para no encontrarnos luego con caracteres mal interpretados. Esta opción está en Advanced. Ok para finalizar.



Upload a new table asset

Source files ?

- muestras.qpj
- muestras.prj
- muestras.cpg
- muestras.dbf
- muestras.shx
- muestras.shp

[Add another file](#)

Asset ID ?

users/santiagobanchero/

Advanced

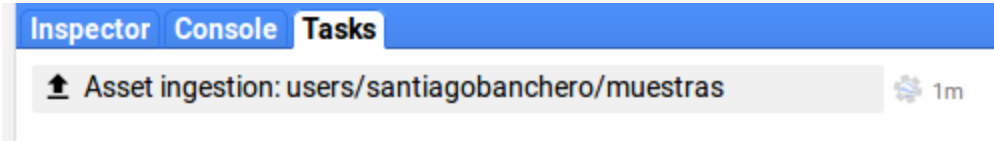
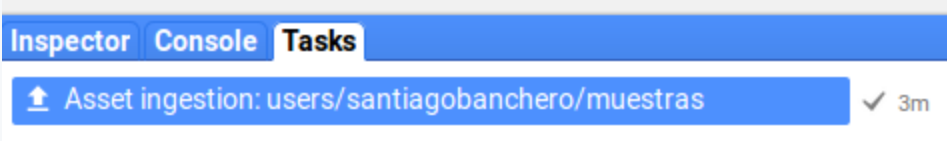
Character Encoding ?

[Learn more](#) about how uploaded files are processed.

# Upload/Task Panel

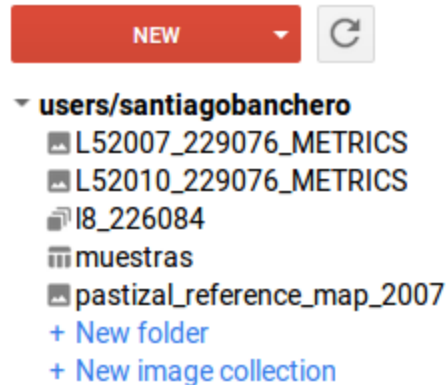
El upload no es instantáneo y puede demorar algunos minutos dependiendo de la congestión de la plataforma y el tamaño que tenga el archivo.

Podemos verificar el progreso desde la solapa TASK:

Tarea en proceso.	 The screenshot shows the 'Tasks' tab selected in a blue header bar. Below the header, there is a task entry with an upward arrow icon, the text 'Asset ingestion: users/santiagobanchero/muestras', and a gear icon followed by '1m'. The task bar has a light gray background, indicating it is in progress.
Tarea finalizada.	 The screenshot shows the 'Tasks' tab selected in a blue header bar. Below the header, there is a task entry with an upward arrow icon, the text 'Asset ingestion: users/santiagobanchero/muestras', and a checkmark icon followed by '3m'. The task bar has a blue background, indicating it is completed.

# Crear un FC desde una tabla

Finalizada la carga la Tabla estará disponible en el Assets de nuestro usuario.



```
var txt_assets = 'users/<usuario>/muestras';  
var muestreos = ee.FeatureCollection(txt_assets);  
print(muestreos);  
Map.addLayer(muestreos, {}, 'Muestras');
```



# FeatureCollection de una muestra al azar

Es posible crear un FeatureCollection a partir de generar una muestra al azar [ee.FeatureCollection.randomPoints](#) de puntos dada una región.

```
var region = ee.Geometry.Rectangle(-63.457, -25.155, -62.6
var randomPoints = ee.FeatureCollection.randomPoints(
    region,
    100, // cantidad de puntos
    123); // seed

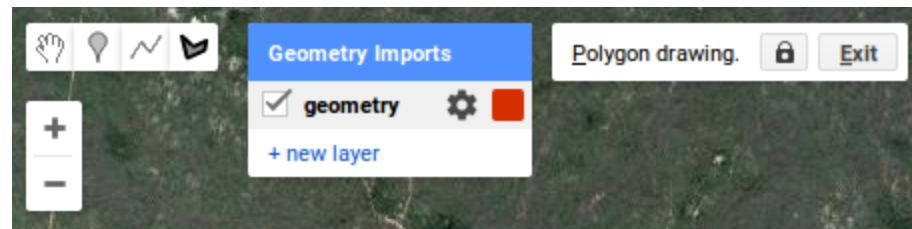
print(randomPoints)

Map.centerObject(randomPoints);

Map.addLayer(randomPoints, {}, 'Puntos al azar');
```

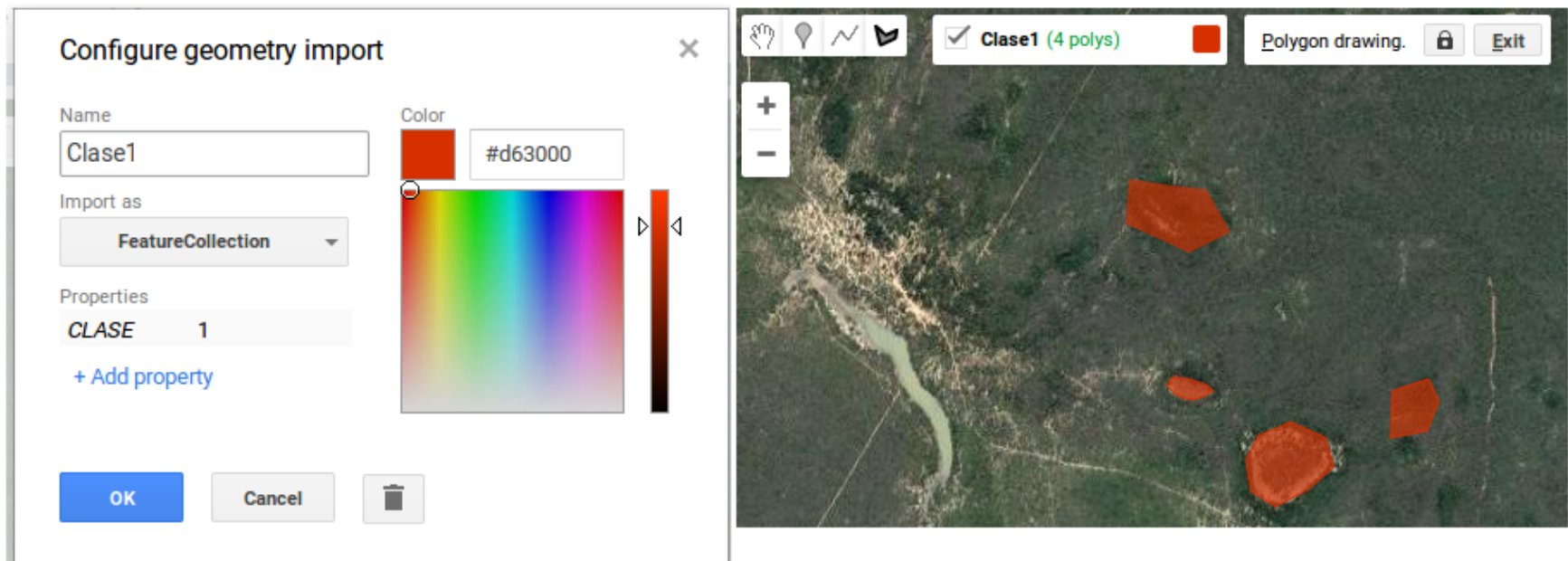
# Creación de FeaturesCollection desde el Mapa

Al igual que mostramos anteriormente como dibujar geometrías, es posible definir FeaturesCollection dibujando desde el mapa.



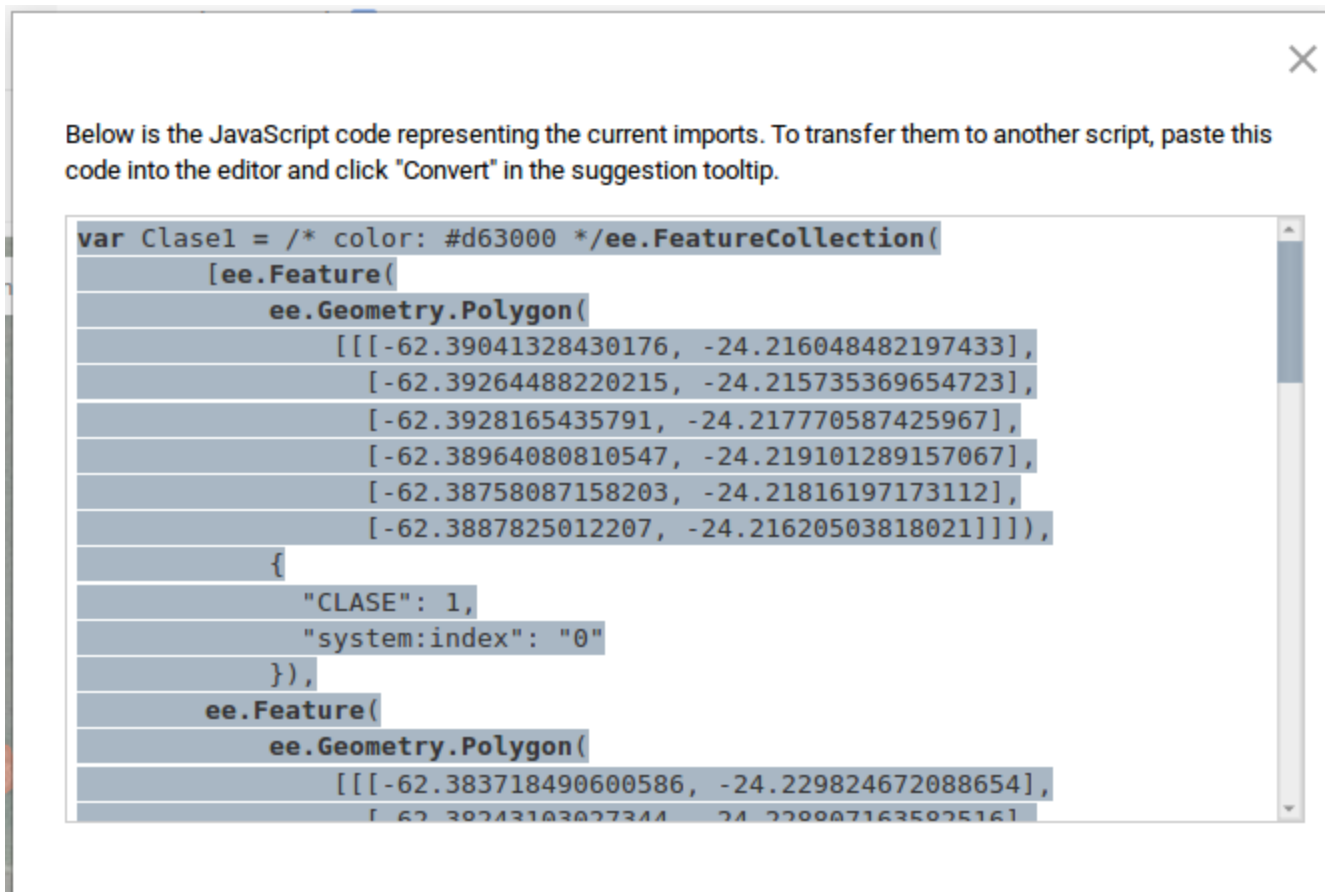
# Creación de FeaturesCollection desde el Mapa

En la configuración podemos darle un nombre al Layer y agregar propiedades con valores por defecto y definir el color.



# Creación de FeaturesCollection desde el Mapa

Podemos incorporar este FeaturesCollection desde la sección de Imports.



Below is the JavaScript code representing the current imports. To transfer them to another script, paste this code into the editor and click "Convert" in the suggestion tooltip.

```
var Clase1 = /* color: #d63000 */ee.FeatureCollection(  
  [ee.Feature(  
    ee.Geometry.Polygon(  
      [[[-62.39041328430176, -24.216048482197433],  
        [-62.39264488220215, -24.215735369654723],  
        [-62.3928165435791, -24.217770587425967],  
        [-62.38964080810547, -24.219101289157067],  
        [-62.38758087158203, -24.21816197173112],  
        [-62.3887825012207, -24.21620503818021]]]],  
    {  
      "CLASE": 1,  
      "system:index": "0"  
    })  
  ],  
  ee.Feature(  
    ee.Geometry.Polygon(  
      [[[-62.383718490600586, -24.229824672088654],  
        [-62.38243103027344, -24.228807163582516],  
        [-62.38243103027344, -24.228807163582516],  
        [-62.383718490600586, -24.229824672088654]]]]  
    )  
  )  
)
```

# Operaciones Básicas sobre FeatureCollection

Existen varios métodos para recuperar información y metadatos de un FC.

```
var key = 'ft:1ExULsxnCc7x8AJQmD7bsg9iQKrKMBvkb0Ji62XVy';

var muestreros = ee.FeatureCollection(key);
Map.addLayer(muestreros, {}, 'Áreas muestradas');
Map.centerObject(muestreros);

print(muestreros.limit(1)); // Limitamos el # de features
print('Lista de atributos:', muestreros.first().propertyName);
print('Cantidad: ', muestreros.size()); // # de features en
```

# Operaciones de agregación por columnas.

```
// Contar cuántas instancias de cada clase hay  
print('Clases Distintas:',  
      muestreos.aggregate_count_distinct('class'));  
  
// Suma toda la columna area_ha  
print('Superficie Total (Ha):',  
      muestreos.aggregate_sum('area_ha'));
```

## Desafío 4

Calcule el tamaño promedio de las parcelas muestreadas.

# Construcción de Filtros

Todos los objetos de tipo colección tienen un método de aplicación de filtros (.filter) que a menudo requieren como parámetro de un objeto del tipo [ee.Filter](#).



# Construcción de Filtros

**Importante:** los filtros aplican sobre las **instancias**, es decir, sobre las filas.

```
var key = 'ft:1ExULsxnCc7x8AJQmD7bsg9iQKrKMBvkb0Ji62XVy';
var muestras = ee.FeatureCollection(key);
Map.addLayer(muestras, {}, 'Áreas muestreadas');

var limites = ee.Geometry.Rectangle([-60.501708984375, -2
var filtrados = muestras.filterBounds(limites);

print('Cantidad de features después de filtrar:', filtrados

Map.addLayer(filtrados, {color: '1ae008'}, 'Filtrados por

var filtradasXarea = filtrados.filter(ee.Filter.gt('area_h

print('Parcelas de más de 10 ha:', filtradasXarea.size());
Map.addLayer(filtradasXarea, {color: 'f4df42'}, 'Más de 10
```

## Desafío 5

Construya un filtro que permita filtrar las clases de tipo 131.

## Desafío 6

¿Cuántas parcelas fueron etiquetadas como ARBUSTAL en la descripción?

# Operador Selection

Selección de propiedades de un Feature para generar un nuevo FeatureCollection.

**Importante:** las selecciones aplican sobre las **columnas**, es decir, sobre los atributos de una colección de features o las bandas cuando veamos colecciones de imágenes.

```
var fc_seleccion = filtrados.select(  
    ['area_ha', 'class'], ['AREA', 'CLASE']);  
  
print(fc_seleccion);
```

# Manejo de iteraciones sobre colecciones de features

Existen varias opciones para poder iterar sobre un FeatureCollection, una forma simple de modificar cada uno de los Features de un FeatureCollection es utilizando la instrucción [ee.FeatureCollection.map](#). Esta instrucción permite recorrer cada Feature y generar un FeatureCollection nuevo.

# Cómo funciona el Map

Veamos un ejemplo simple, supongamos que queremos incorporar al FeatureCollection de las muestras un atributo que sea perímetro. Esto requiere que para cada elemento (de tipo Feature) de la colección hagamos el cálculo, eso sería:

```
var get_perimetro = function(elemento){  
    return elemento.set(  
        {perimetro: elemento.geometry().perimeter()}  
    );  
}  
  
var m_con_perimetro = muestreros.map(get_perimetro);  
print(m_con_perimetro);
```

# Otro ejemplo de map

Supongamos que tenemos un CSV con las coordenadas expresadas en grados decimales y queremos generar la geometría para cada Feature del FeatureCollection:

```
var key = 'ft:1t-2SIDNQHJzi_6iSWww0pAbd_4i33l8o68NUh4an';
var muestras = ee.FeatureCollection(key);

// Muestreo sin geometría
print( muestras.limit(1) );

var agregar_geometria = function( elemento ){
  var geom = ee.Geometry.Point(
    [
      elemento.get('longitud'),
      elemento.get('latitud')
    ]
  );
  return elemento.setGeometry(geom);
};
Map.addLayer( muestras.map(agregar_geometria), {}, 'Muest'
```

## Desafío 7

Escriba una función de mapeo que para valores de class entre 20 y 23 completen un nuevo atributo llamado TIPO con el valor “Bosque” y en caso contrario complete con “No Bosque”.

**Solución.**



# Iteraciones con Iterate

Existe otra forma de recorrer un FeatureCollection que es con el método `iterate`.

```
var key = 'ft:1ExULsxnCc7x8AJQmD7bsg9iQKrKMVbkb0Ji62XVy';
var muestreros = ee.FeatureCollection(key);

var n_dict = ee.Dictionary({});

var contar_clases = function(feet, n_dict){
  return ee.Dictionary(n_dict).set(
    feet.get('class'),
    muestreros.filter(
      ee.Filter.eq('class', feet.get('class')))
      .aggregate_count('class')
  );
}

var n_class = muestreros.distinct(['class'])
  .iterate(contar_clases, n_dict);

print(n_class);
```

# Exportar como tabla de datos

Para exportar un FeatureCollection a Google Drive se requiere la instrucción `Export.table.toDrive()`

`Export.table.toDrive(collection, description, folder, fileNamePrefix, fileFormat)` ×

Creates a batch task to export a FeatureCollection as a table to Drive. Tasks can be started from the Tasks tab.

**Arguments:**

- **collection (FeatureCollection):**  
The feature collection to export.
- **description (String, optional):**  
A human-readable name of the task. Defaults to "myExportTableTask".
- **folder (String, optional):**  
The Google Drive Folder that the export will reside in.
- **fileNamePrefix (String, optional):**  
The filename prefix. Defaults to the task's description.
- **fileFormat (String, optional):**  
The output format: "CSV" (default), "GeoJSON", "KML", or "KMZ".

# KML

```
Export.table.toDrive({  
  collection: <Nombre del FC>,  
  description: 'Una descripción de la capa para enc  
  fileFormat: 'KML o KMZ'  
});
```

# CSV

```
Export.table.toDrive({  
  collection: <Nombre del FC>,  
  description: 'Una descripción de la tabla para encontra  
  fileFormat: 'CSV'  
});
```

# Un ejemplo picante! :D

```
var key = 'ft:1t-2SIDNQHJji_6iSWww0pAbd_4i33l8o68NUh4an';
var muestras = ee.FeatureCollection(key);

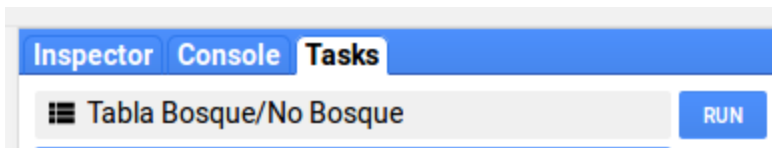
var mapear_clase = function( elemento ){
  return elemento.set('tipo',
    ee.Algorithms.If( ee.Number(elemento.get('class')).gte(20),
      'Bosque',
      'No Bosque' ));
};

var fc_tipo = muestras.map( mapear_clase );

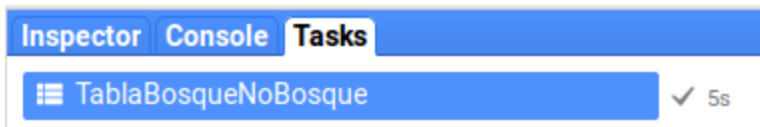
// Exportar a CSV
Export.table.toDrive({
  collection: fc_tipo,
  description: 'TablaBosqueNoBosque', //Es el nombre que ter
  fileFormat: 'CSV'
});
```

# Tareas de exportación

Luego de exportar se incluirá una nueva tarea que hay que poner a correr.



Dependiendo de la cantidad de features en la colección el proceso puede demorar mucho o poco y además también dependerá de la carga de trabajo de la plataforma.



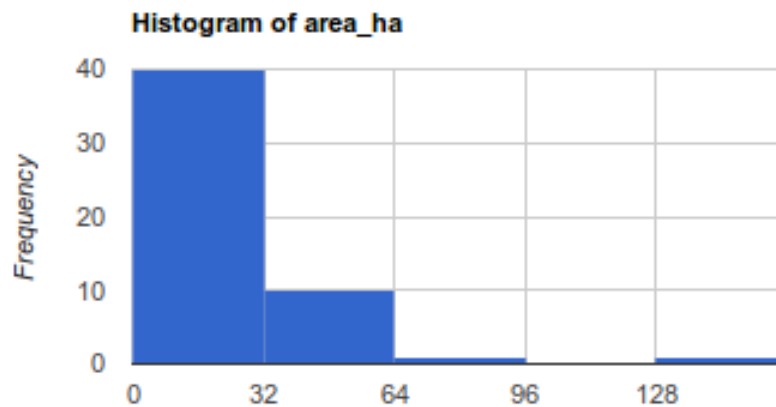
# Realizar gráficos

En GEE es posible realizar diferentes tipos de gráficos utilizando la librería `ui.Chart`. Las opciones disponibles para `FeaturesCollection` son:

- `ui.Chart.feature.byFeature(features, xProperty, yProperties)`
- `ui.Chart.feature.byProperty(features, xProperties, seriesProperty)`
- `ui.Chart.feature.groups(features, xProperty, yProperty, seriesProperty)`
- `ui.Chart.feature.histogram(features, property, maxBuckets, minBucketWidth, maxRaw)`

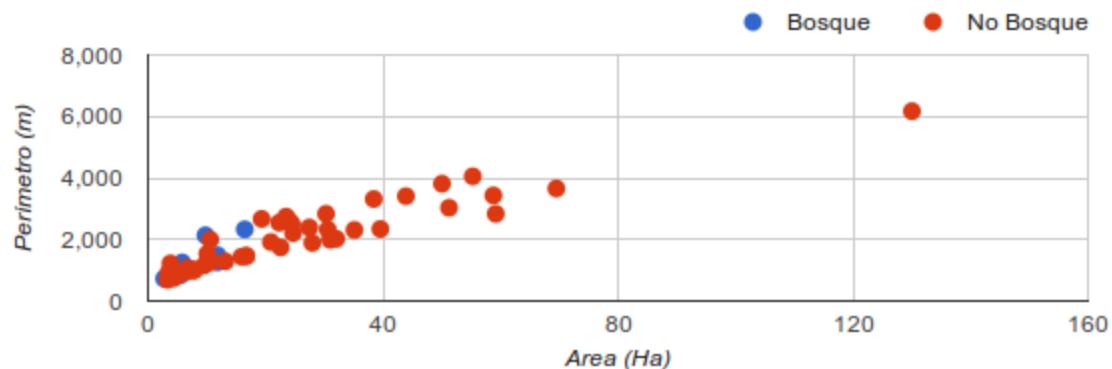
# Graficar Histogramas desde Features

```
var key = 'ft:1ExULsxnCc7x8AJQmD7bsg9iQKrKMBvkb0Ji62XVy';  
var muestras = ee.FeatureCollection(key);  
var histograma = ui.Chart.feature.histogram(muestras, 'area_ha',  
print(histograma);
```



# Graficar Features por Grupos

```
var chart = ui.Chart.feature.groups(fc_tipo, 'area_ha', 'p  
.setChartType('ScatterChart')  
.setOptions({  
  hAxis: {title: 'Area (Ha)'},  
  vAxis: {title: 'Perímetro (m)'},  
}).setSeriesNames(["Bosque", "No Bosque"]);  
  
print(chart);
```



Probar ejemplo completo [aquí](#)



# Bibliografía

API | Google Earth Engine API.

[https://developers.google.com/earth-engine/api\\_docs](https://developers.google.com/earth-engine/api_docs)

[1] European Petroleum Survey Group