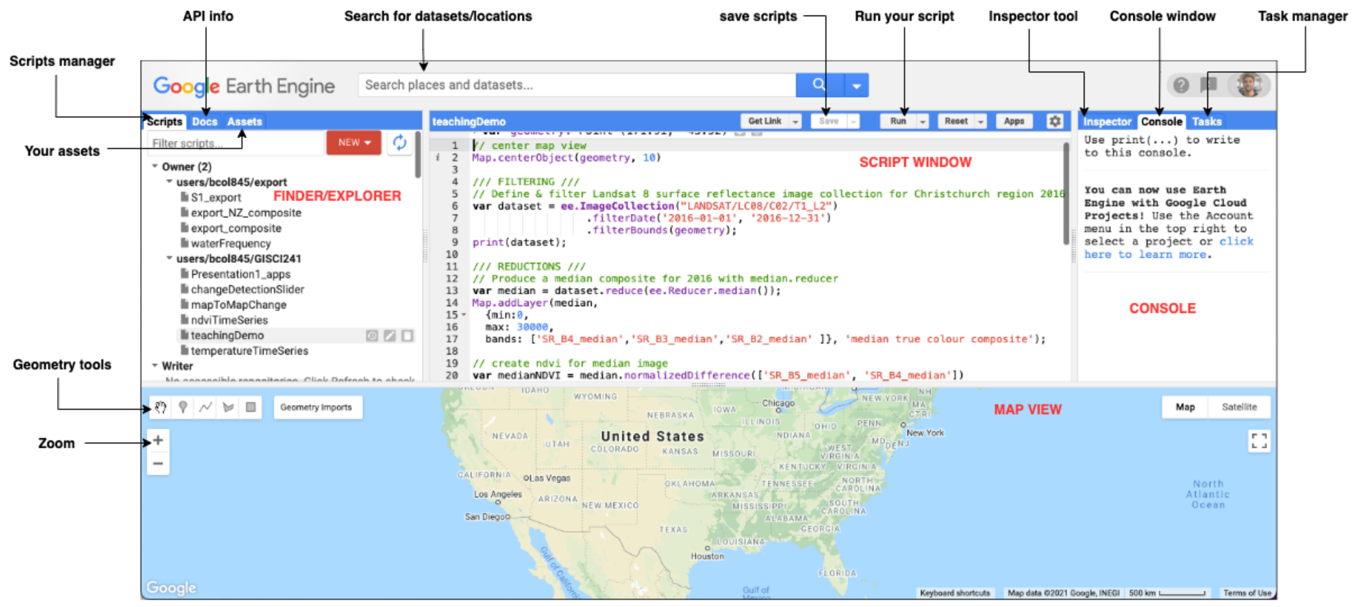
Introduction to Google Earth Engine

The code editor

This is the easiest way to run analysis and access the data available in GEE. The code editor is accessible through a web browser allowing us to write and execute scripts visualising the outputs almost instantaneously in the map view.



```
// center map view
Map.centerObject(geometry, 10)

/// FILTERING ///
// Define & filter Landsat 8 surface reflectance image collection for Christchurch region 2016
var dataset = ee.ImageCollection("LANDSAT/LC08/C02/T1_L2")
                  .filterDate('2016-01-01', '2016-12-31')
                  .filterBounds(geometry);
print(dataset);

/// REDUCTIONS ///
// Produce a median composite for 2016 with median.reducer
var median = dataset.reduce(ee.Reducer.median());
Map.addLayer(median,
  {min:0,
  max: 30000,
  bands: ['SR_B4_median','SR_B3_median','SR_B2_median' ]}, 'median true colour composite');

// create ndvi for median image
var medianNDVI = median.normalizedDifference(['SR_B5_median', 'SR_B4_median'])
```

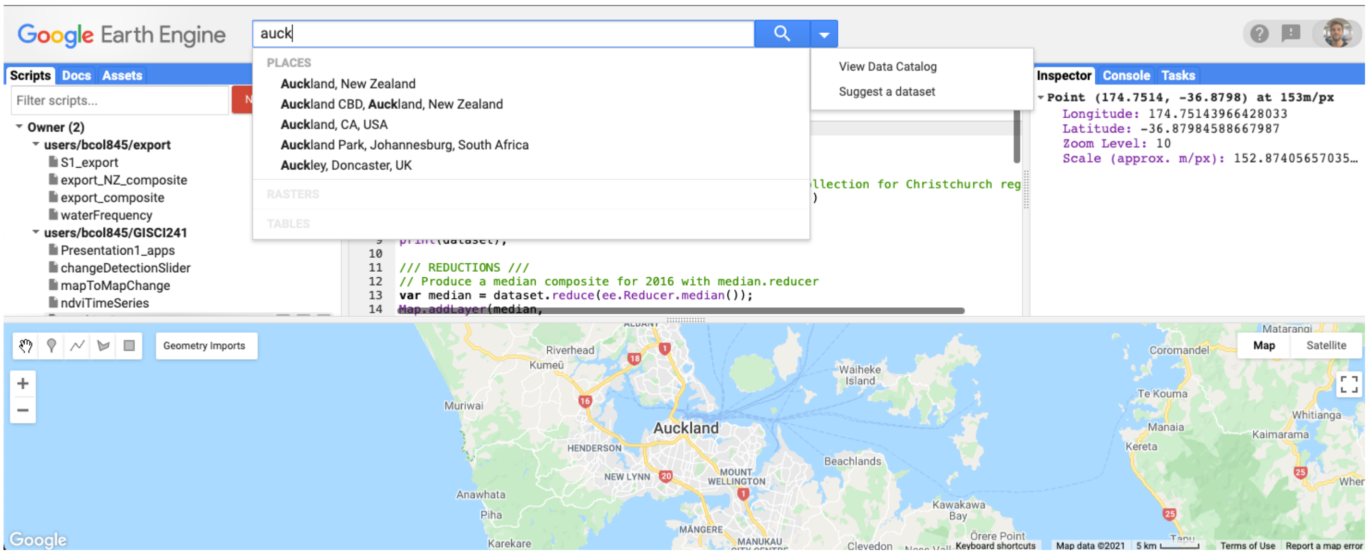
```
Use print(...) to write
to this console.

You can now use Earth
Engine with Google Cloud
Projects! Use the Account
menu in the top right to
select a project or click
here to learn more.
```

Searching for datasets, filtering and visualising images

> Objectives
>
> - Understand how to search for datasets using the code editor and import them into scripts.
> - Learn how to create a geometry and use it in a script.
> - Learn about image collections and how they can be filtered.
> - Understand how the docs tab can be used to search for functions.
> - Understand visualisation parameters and use them to visualise an image in the map view.

We can search for locations and available datasets that we want to use in our analysis using the search function. Interacting with data in the map view is achieved using the inspector tool. Clicking on the location in the map view prints this information to the console.



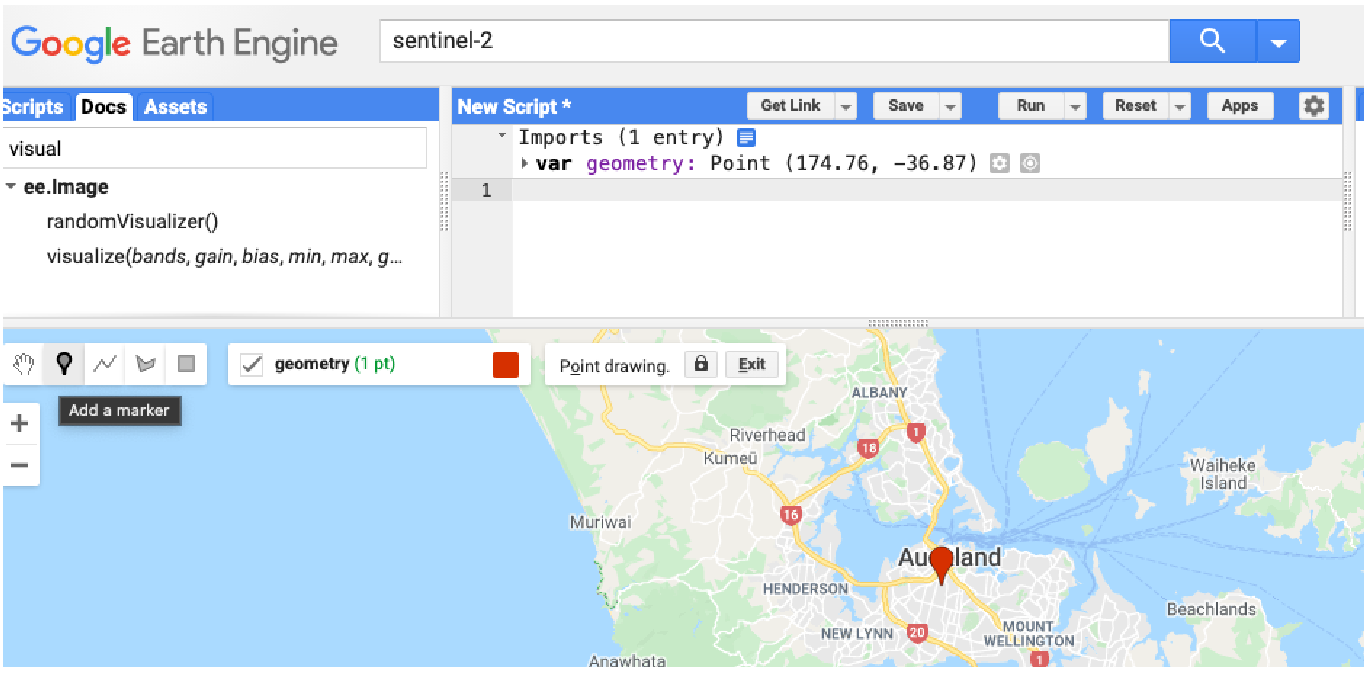
```
llection for Christchurch reg
)

print(dataset);

/// REDUCTIONS ///
// Produce a median composite for 2016 with median.reducer
var median = dataset.reduce(ee.Reducer.median());
Map.addLayer(median,
```

```
Point (174.7514, -36.8798) at 153m/px
  Longitude: 174.75143966428033
  Latitude: -36.87984588667987
  Zoom Level: 10
  Scale (approx. m/px): 152.87405657035…
```

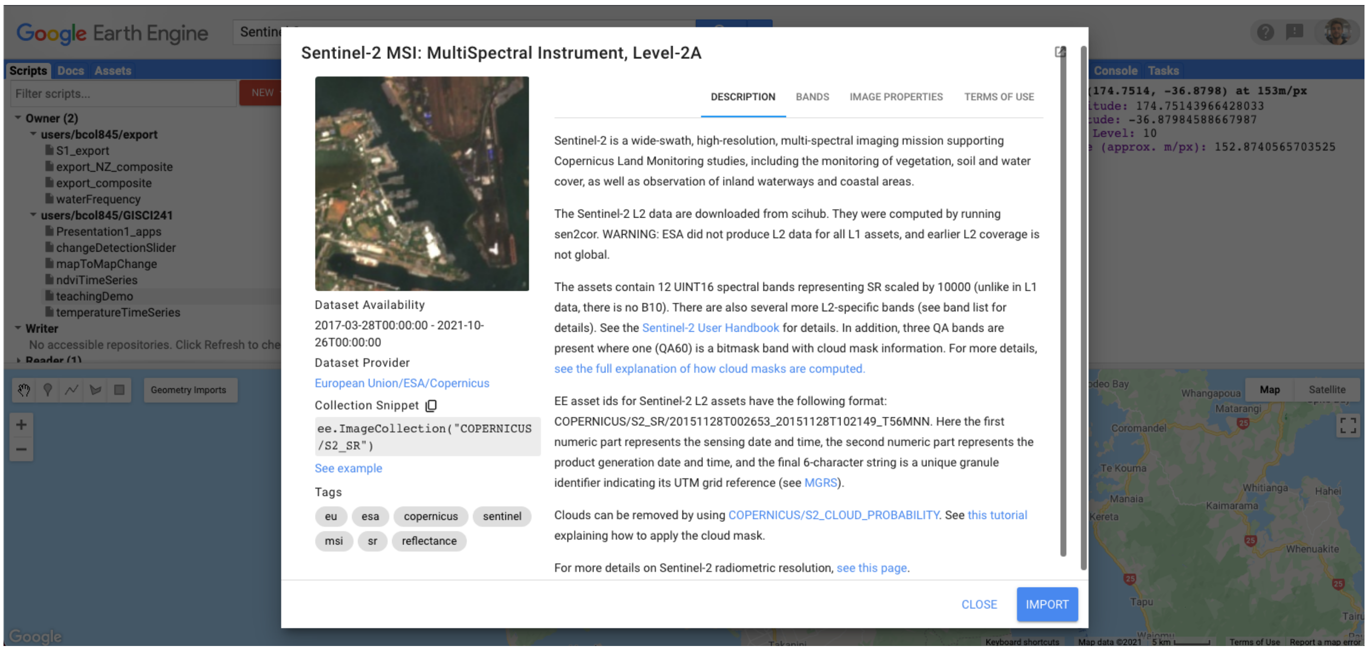
> **Step 1:** Search for Auckland and use the inspector tool to find out the coordinates of a point roughly in central Auckland. Make a note of your coordinates for later by creating a geometry that is a point of your coordinates using the geometry tools.



```
Imports (1 entry)
var geometry: Point (174.76, -36.87)
```

The inspector tool is very handy way of querying our data and extracting information that we can use in our scripts. The geometry tools are a quick easy way of generating a point, line or polygon that can be stored as an input to be used in our scripts.

> **Step 2:** Lets search for a dataset. We'll use the Sentinel-2 level 2A collection.



Sentinel-2 MSI: MultiSpectral Instrument, Level-2A

DESCRIPTION BANDS IMAGE PROPERTIES TERMS OF USE

Dataset Availability
2017-03-28T00:00:00 - 2021-10-26T00:00:00

Dataset Provider
European Union/ESA/Copernicus

Collection Snippet

```
ee.ImageCollection("COPERNICUS/S2_SR")
```

See example

Tags
eu esa copernicus sentinel msi sr reflectance

Sentinel-2 is a wide-swath, high-resolution, multi-spectral imaging mission supporting Copernicus Land Monitoring studies, including the monitoring of vegetation, soil and water cover, as well as observation of inland waterways and coastal areas.

The Sentinel-2 L2 data are downloaded from scihub. They were computed by running sen2cor. WARNING: ESA did not produce L2 data for all L1 assets, and earlier L2 coverage is not global.

The assets contain 12 UINT16 spectral bands representing SR scaled by 10000 (unlike in L1 data, there is no B10). There are also several more L2-specific bands (see band list for details). See the Sentinel-2 User Handbook for details. In addition, three QA bands are present where one (QA60) is a bitmask band with cloud mask information. For more details, see the full explanation of how cloud masks are computed.

EE asset ids for Sentinel-2 L2 assets have the following format: COPERNICUS/S2_SR/20151128T002653_20151128T102149_T56MNN. Here the first numeric part represents the sensing date and time, the second numeric part represents the product generation date and time, and the final 6-character string is a unique granule identifier indicating its UTM grid reference (see MGRS).

Clouds can be removed by using COPERNICUS/S2_CLOUD_PROBABILITY. See this tutorial explaining how to apply the cloud mask.

For more details on Sentinel-2 radiometric resolution, see this page.

CLOSE IMPORT

There are several ways we can access data in our scripts. In GEE a raster dataset is called an image collection. The simplest of these is using import. What happens if you click import?

This imports the image collection under the variable `imageCollection`

We can also copy the collection snippet to use in our code.

> **Step 3:** Copy the collection snippet and paste it to the script window and edit the line to match to code below and hit run.

> **Top-tip:** Typing your code rather than using copy & paste will help with your understanding of what each line is doing.

```
var image_collection = ee.ImageCollection("COPERNICUS/S2_SR")
print(image_collection)
```

We now have the Sentinel-2 level 2A dataset stored under the variable `image_collection`

What happens when this is printed to the console?

This collection contains all the level-2A data in the Sentinel-2 archive. We can only print 5000 elements to the console in one print command so we get an error. We need to make our collection managable by filtering it.

> **Step 4:** Filter the collection using our point we created earlier and the following code.

```
// define sentinel-2 level 2A image collection
var image_collection = ee.ImageCollection("COPERNICUS/S2_SR")
    .filterBounds(geometry)

// print image_collection to console
print(image_collection)
```

What happens with our print command this time?

We now get an ouput printed to console which contains a list of features showing us the number of elements/images in our collection that intersect with our geometry.

Lets add a filter to obtain the least cloudy image in the collection so we can visualise it.

> **Step 5:** Add a metadata filter to your script to include images that have less than 5% cloud cover.

```
// define sentinel-2 level 2A image collection
var image_collection = ee.ImageCollection("COPERNICUS/S2_SR")
    .filterBounds(geometry)
    .filterMetadata('CLOUDY_PIXEL_PERCENTAGE', "less_than", 5)

// print image_collection to console
print(image_collection)
```

What does this do to the number of elements in our collection?

We can use the metadata filter to filter a collection by any of ther properties in the metadata. Now we can select an image from our collection to visualise. There are several other useful filters that we can use. We can search for these in the docs tab.

> **Step 6:** Return the first image in the collection and print it to the console.

```
// define sentinel-2 level 2A image collection
var image_collection = ee.ImageCollection("COPERNICUS/S2_SR")
    .filterBounds(geometry)
    .filterMetadata('CLOUD_COVER', "less_than", 5)

// print image_collection to console
print(image_collection)

// define the first image in image_collection
var sentinel_image = image_collection.first()

// print sentinel_image to console
print(sentinel_image)
```

How does the output from the final print command differ from first one? We now have a single image which contains a list of 23 elements. These are the image bands in the raster. We are now ready to visualise the image in the map view.

> **Step 7:** Define some visualisation parameters and add the image to the map view.

```
// return the first image in image_collection
var sentinel_image = image_collection.first()

// print sentinel_image to console
print(sentinel_image)

// define visualisation parameters for sentinel_image
var visParams = {min: 0, max: 3000, bands:['B4', 'B3', 'B2']}

// add sentinel_image to map view
Map.addLayer(sentinel_image, visParams, 'sentinel-2 image')
```

Once we run the script we should see the image in the map view. Well done! You've visualised your first satellite image in GEE. The visulisation parameters are stored as a dictionary and can be defined as a variable or included in `addLayer()` function (e.g. `Map.addLayer(sentinel_image, {min: 0, max: 3000, bands:['B4', 'B3', 'B2']}, 'sentinel-2 image')`).

The `min` and `max` value are the pixels values used to apply the stretch and the `bands` define the image bands to be used in the Red, green, and blue colour channels respectively. In this case `['B4', 'B3', 'B2']` refer to the RGB bands for Sentinel-2 resulting in a true colour image.

> **A few more things to think about:**
>
> - What would you change in `visParams` to visualise your image as a false colour composite where the Red channel represents the near-infrared band?
> - What happens if you select the inspector tool and click on a pixel in your image?
> - How would you filter your image collection to return images in a given date range?

> **Key points**
>
> - An image collection contains all available data for a given sensor and this can be imported into a script or referenced using it's code snippet.
> - The search bar can be used to search for locations and any data available in GEE.
> - We use the filtering functions to obtain data for a given area, date range or filter a collection by metadata properties.
> - We must define visualisation parameters to view an image in the map view.
> - We can query information/data in the map view using the insepctor tool.
> - We can create lines, polygons or point geometries as imports to use in scripts.