Remotely Sensing Cities and Environments

Lecture 5: An introduction to Google Earth Engine

28/10/2022 (updated: 06/02/2023)

a.maclachlan@ucl.ac.uk

andymaclachlan

andrewmaclachlan

Centre for Advanced Spatial Analysis, UCL

PDF presentation

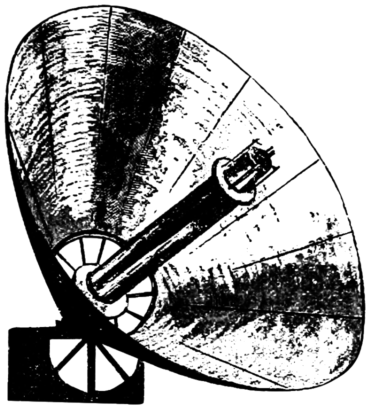
How to use the lectures

- Slides are made with xaringan
- 🔍 In the bottom left there is a search tool which will search all content of presentation
- Control + F will also search
- Press enter to move to the next result
- ✏ In the top right let's you draw on the slides, although these aren't saved.
- Pressing the letter o (for overview) will allow you to see an overview of the whole presentation and go to a slide
- Alternatively just typing the slide number e.g. 10 on the website will take you to that slide
- Pressing alt+F will fit the slide to the screen, this is useful if you have resized the window and have another open - side by side.

Lecture outline

Part 1: The set up of GEE

- Terms / jargon specific to GEE
- Relating spatial data formats we have seen to GEE
- Client vs server side
- Scale (resolution)
- Projections

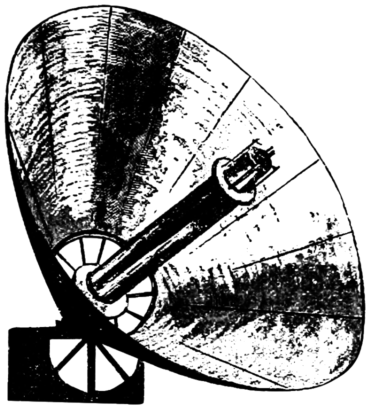


Source:Original from the British Library. Digitally enhanced by rawpixel.

Lecture outline (part 2)

Part 2: GEE in action (how we use it)

- How we use Google Earth Engine
- Buildings blocks of data (the data in GEE)
- Collections, geometries and features
- Reducing images (e.g. zonal statistics)
- Regression (over time)
- Joins
- Machine learning (video intro, more in next few weeks)



Source:Original from the British Library. Digitally enhanced by rawpixel.

Part 1: The setup of GEE

What is Google Earth Engine

Google Earth Engine

- "Geospatial" processing service
    - recall week 1 of CASA0005!
    - Geo = Earth's surface (and near surface)
    - Spatial = Any space (not necessarily geographic)
- It permits geospatial analysis **at scale**
- Scale ?
    - massive datasets
    - planetary scale analysis
    - really quickly (within seconds)
- Stores data on servers
- Takes the code you have written and applies it for you

- Can be used to make **queryable** online applications

GEE terms (jargon?)

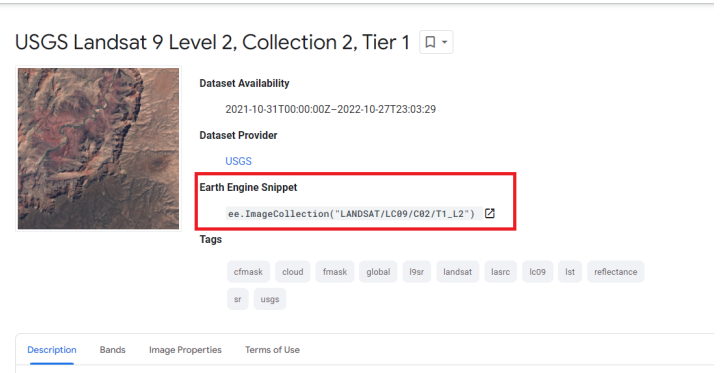
Remember raster and vector data?

GEE has special names for them:

- Image = raster
  - Has bands
- Feature = vector
  - Has geometry and attributes (dictionary of properties in GEE)

A stack of images (remember from R) is also called something new:

- Image stack= ImageCollection
- Feature stack (lots of polygons) = FeatureColletion



Example of loading data quickly

GEE Javascript

- GEE uses Javascript
  - Website programming language
- You will see some similarities to python and R but there are some notable differences
  - Variables (or objects) as defined with `var`..
  - A specific part of your code ends with a `;`
  - Objects are dictionaries in Javascript...

```
// Use curly brackets {} to make a dictionary of key:value pairs.
var object = {
  foo: 'bar',
  baz: 13,
  stuff: ['this', 'that', 'the other thing']
};

print('Print foo:', object['foo']);
```

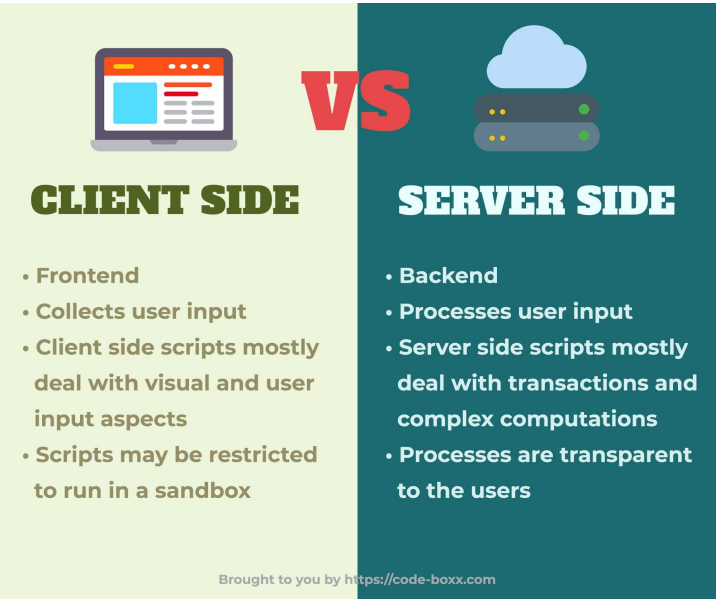
Introduction to JavaScript for Earth Engine. Source: GEE

Client vs server side

- Within GEE we have code that runs on the client side
    - This is the browser
- We have code that runs on the server side
    - This is **on the server where data is stored**...

In GEE we have **Earth Engine Objects** = starting with ee, for example...

```
ee.ImageCollection("LANDSAT/LC09/C02/T1_L2'
```



Client vs Server side. Source: pintrest/codeboxx

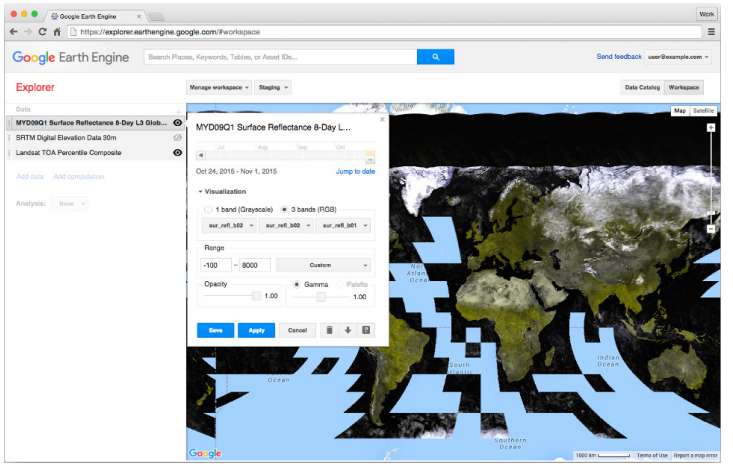
Client vs server side...what does this mean...

In practice i don't think about this much...

- Anything that has ee **in front of it is stored on the server**
- It has no data in your script. Recall in R the data environments, this would be empty.
- They are termed "proxy objects"

> the agency, function, or office of a deputy who acts as a substitute for another

- Any pre-loaded data product will be on the server side...



Data Explorer. Source: GEE

Client vs server side...problems...

- Looping
  - You can't (or shouldn't) use a loop for something on the server
  - The loop doesn't know what is in the ee object!
- Mapping functions
  - Instead we can create a function and save it into an object (or variable here)
  - Then apply it to everything on the server
  - Same idea as `map()` in R from the purrr package.
- Server side **functions**
  - Same as the data.
  - `ee.Thing.method()`
  - **Saved** function on the sever that can be run without mapping

Loop vs Map...an example

Loop

- Run some code on each element in an array
- Save it in a new array (or update existing)

```
names = ['andrew', 'james', 'crites']

# manually
capitalize(names[0])
capitalize(names[1])
capitalize(names[2])

#loop
x = 0;
for x < names.length:
  capitalize(names[x])
  x = x + 1
```

map vs. for loop. Source: Andrew Crites

Map

- Code is applied to each element
- But...the conditions are dealt with
    - No indexing at the end!

```
# map
names.map(name => capitalize(name))
```

OR

```
names.map(capitalize)
```

Why map in GEE?

- Otherwise you might load the complete image collection many, many times when looping
- The loop doesn't know what is inside the collection
- Mapping lets GEE allocate the the processing to different machines with map. I assume as it knows how many images we have and the function it must apply...With a loop it doesn't know until the next interation...

Scale on the first slide referred to the volume of analysis

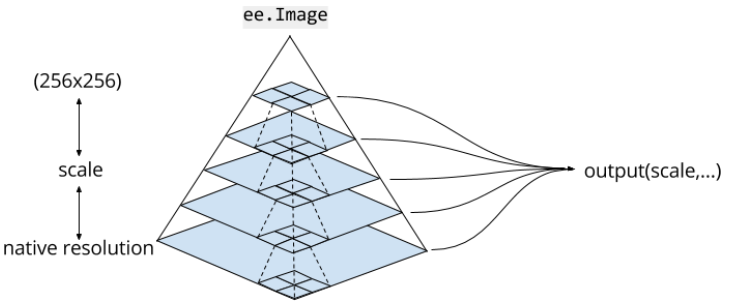
Scale here refers to pixel resolution

Scale

- Image scale in GEE refers to pixel resolution.
- In GEE the scale (resolution) is set by the **output not input**
- When you are doing analysis
    - GEE aggregates the image to fit a 256x256 grid
    - Earth Engine selects the pyramid with the closest scale to that of your analysis (or specified by it) and resamples as needed
    - resampling uses nearest neighbor by default

For example... the zoom level of the map can dictate the scale used...

Set your scale



Scale. Source: GEE

Scale example

Take an image and have a look at it...

```
var image = ee.Image('LANDSAT/LC08/C01/T1/LC08_044034_20140318');

var rgbVis = {
  bands: ['B4', 'B3', 'B2'],
  min: 5964.56,
  max: 11703.44
};

Map.addLayer(image, rgbVis, "Landsat 8");
```

Scale. Source: GEE

Scale example 2

- Select a band and then change the scale
- Explore the values...

```
var band_4 = image.select('B4');

var printAtScale = function(scale) {
  print('Pixel value at '+scale+' meters scale',
    band_4.reduceRegion({
      reducer: ee.Reducer.first(),
      geometry: band_4.geometry().centroid(),
      // The scale determines the pyramid level from which to pull the input
      scale: scale
  }).get('B4'));
};

printAtScale(10); // 8883
printAtScale(30); // 8883
printAtScale(50); // 8337
printAtScale(70); // 9215
printAtScale(200); // 8775
printAtScale(500); // 8300
```

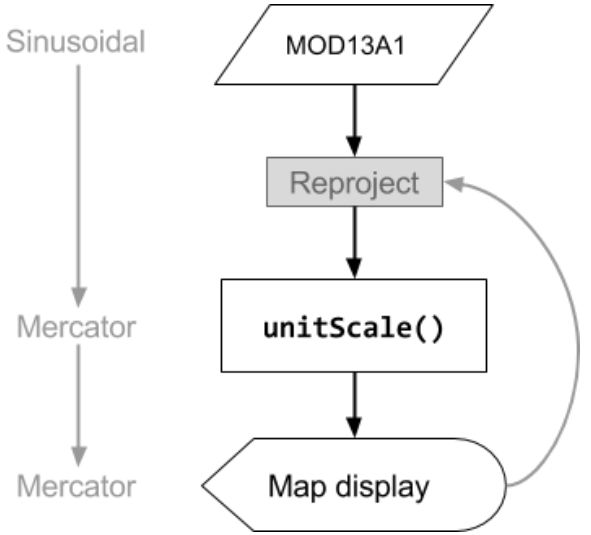
Code example

Why are the values different

Earth engine is aggregating the values based on your analysis (or specified scale)

Projections

- You do not need to worry about projects in GEE
- GEE converts all data into the Mercator projection (EPSG: 3857) when displayed
    - Specifically: WGS 84 / Pseudo-Mercator -- Spherical Mercator, Google Maps, OpenStreetMap, Bing, ArcGIS, ESRI
- The operations of the proejction are determined by the output - so it works out what you need and gives it to you!
- You force set the projection, but there is no real reason to do this.



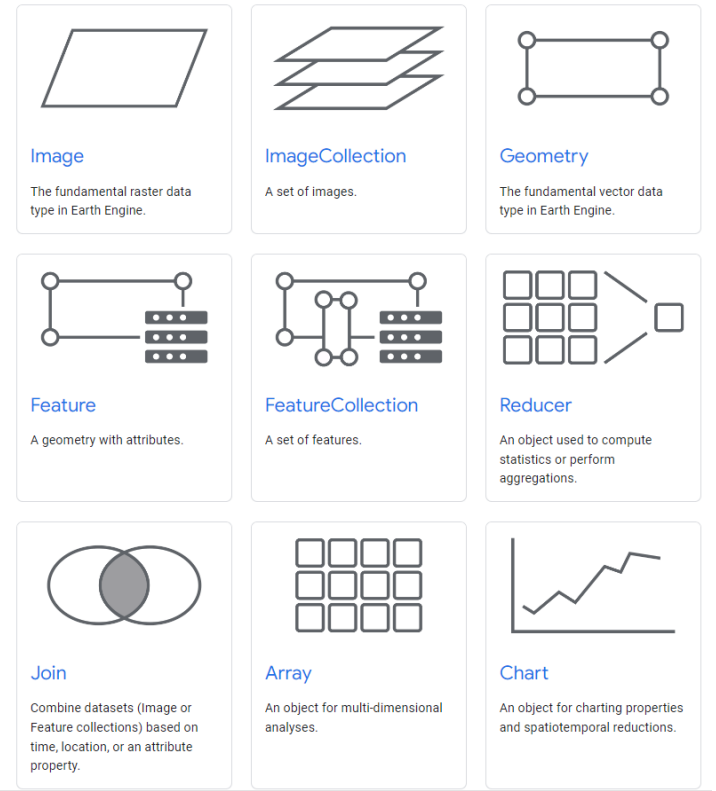
Flow chart of operations corresponding to the display of a MODIS image in the Code Editor map. Projections (left side of flow chart) of each operation are determined from the output. Curved lines indicate the flow of information to the reprojection: specifically, the output projection and scale. Source: GEE

Part 2: GEE in action (how we use it)

Building blocks of GEE

Objects

- Object = vector, raster, feature, string, number
- Each of these belongs to a **class**
- Each **class** has specific GEE functions (or **methods**) for it...



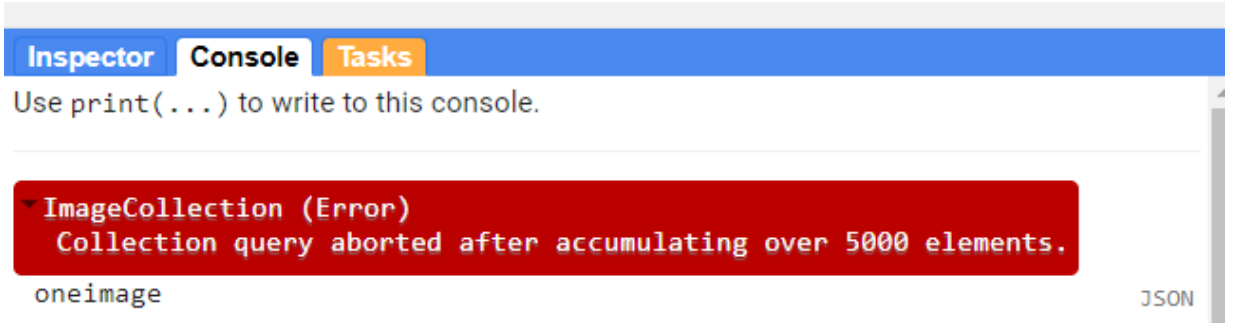
Object classes. Source: GEE

An example...image collection

- An image collection is a **stack** or rasters...
- It is loaded with the function specific to image collections...

```
var dataset = ee.ImageCollection('LANDSAT/LC09/C02/T1_L2');

print(oneimage, "oneimage");
```

- But this is too many images and gives...

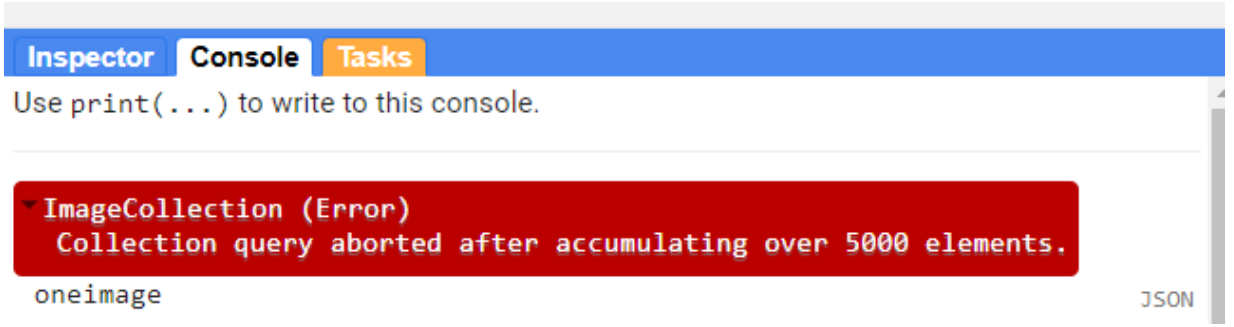


An example...image collection

- We could filter on the dates....but this is still providing Landsat data for the **whole world**

```
var oneimage = ee.ImageCollection('LANDSAT/LC09/C02/T1_L2')
  .filterDate('2022-01-03', '2022-04-04');
```

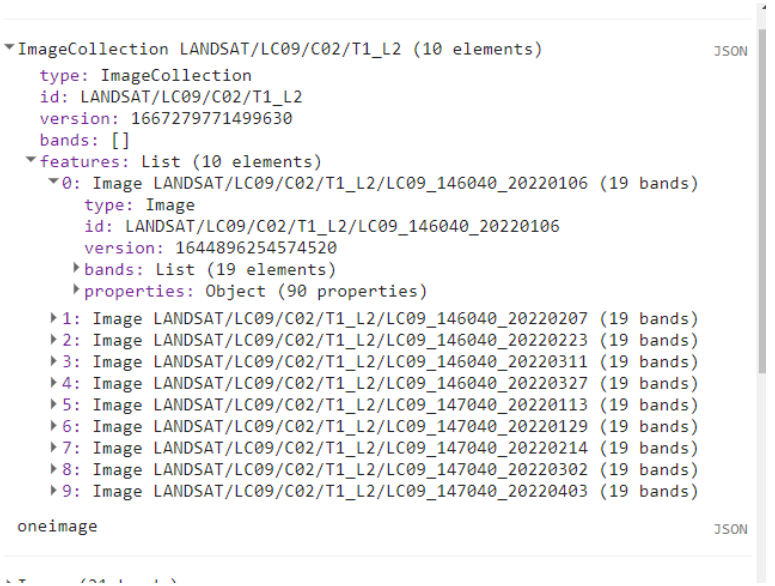
- So we get the same error!



An example...image collection

- We can add a dates and region filter....! This is where it intersects the bounds..

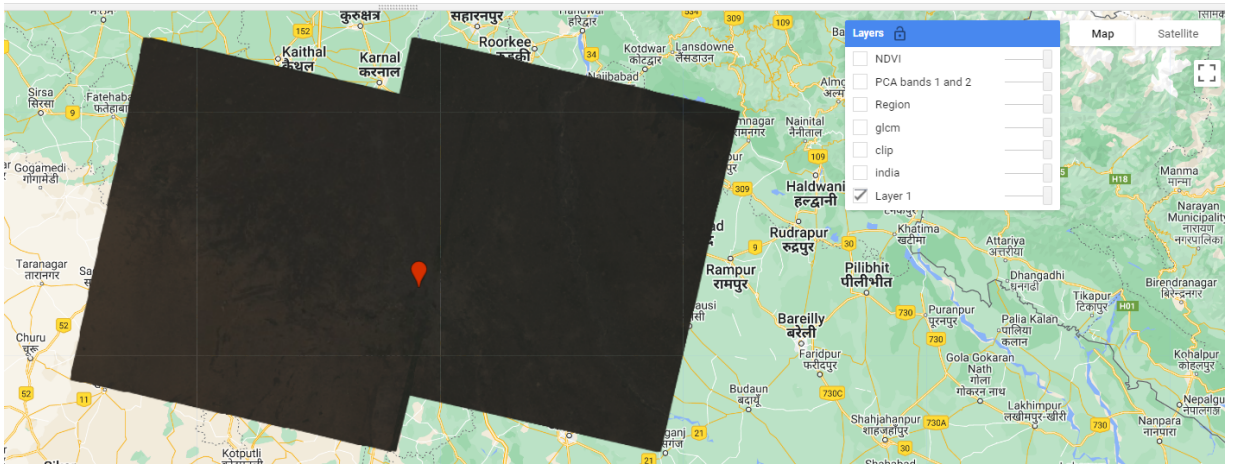
```
var oneimage = ee.ImageCollection('LANDSAT/LC09/C02/T1_L2')
  .filterDate('2022-01-03', '2022-04-04')
  .filterBounds(Dheli);  // Intersecting ROI
```



An example...image collection

Now, we can also add this to the map to see it

```
var oneimage = ee.ImageCollection('LANDSAT/LC09/C02/T1_L2')
  .filterDate('2022-01-03', '2022-04-04')
  .filterBounds(Dheli);  // Intersecting ROI

Map.addLayer(oneimage, {bands: ["SR_B4", "SR_B3", "SR_B2"]})
```



So here we have:

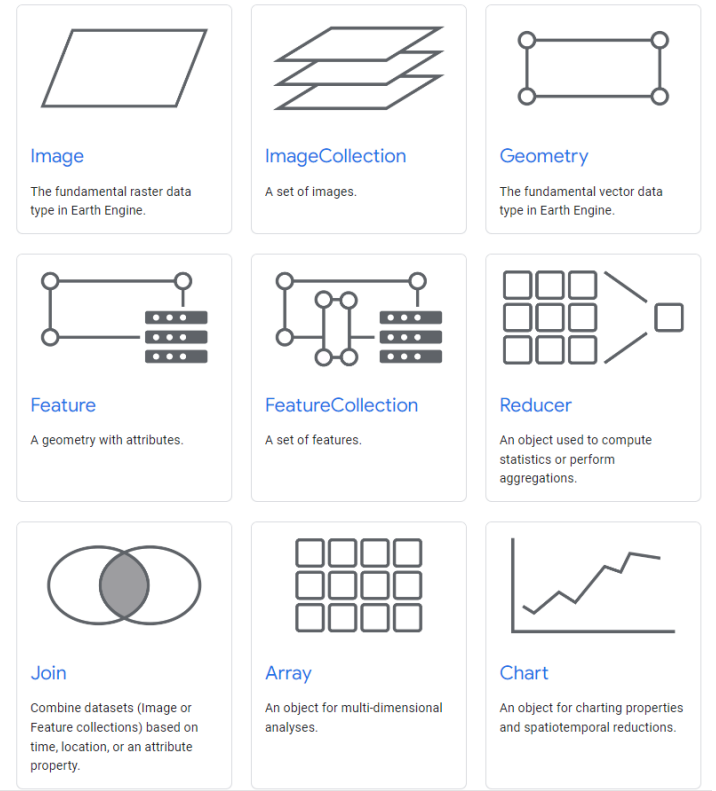
Raster data (lots of images)

They belong to an imagecollection (as there are lots of images)

We use the specific function (method or "constructor") to load and manipulate

An example...geometries and features

- Geometry = point/line/polygon with **no attributes**
    - Note you can also have MultiPolygon or MultiGeometry
- Feature = geometry **with attributes**
- Feature collection = **several features with** attributes

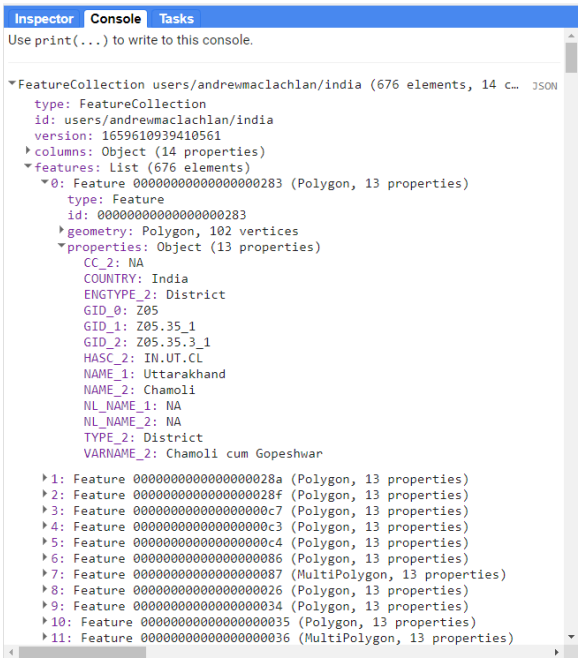


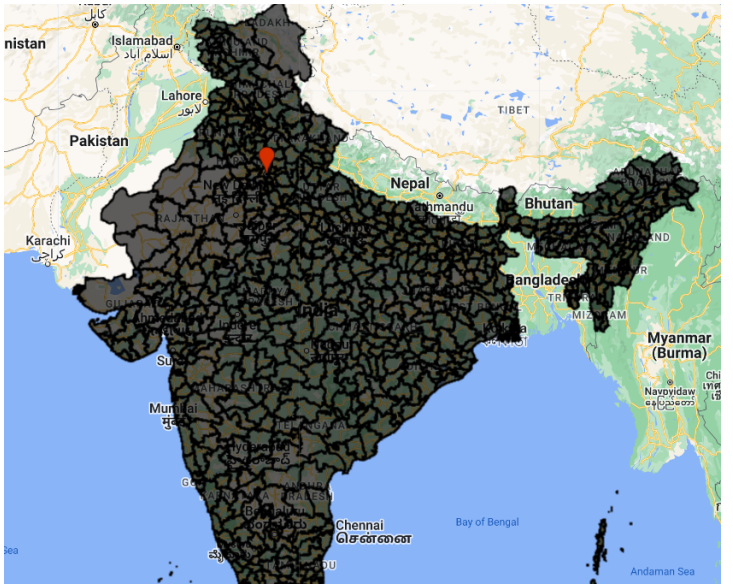
Object classes. Source: GEE

An example...geometries and features

To load a feature collection....print the summary and then add it to the map..

```
var india = ee.FeatureCollection('users/andrewmaclachlan/india');

print(india, "india");

Map.addLayer(india, {}, "india");
```



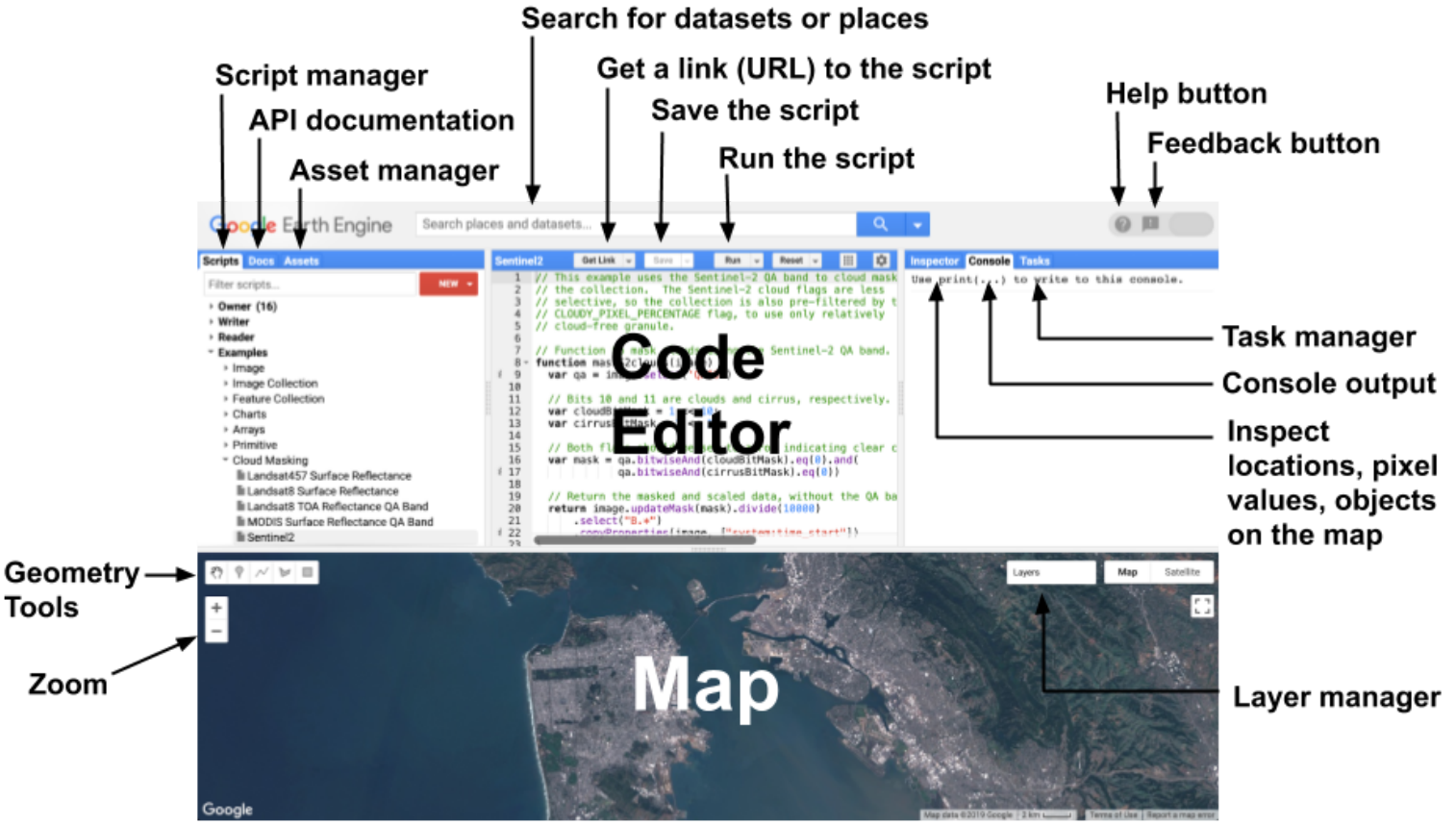


An example...geometries and features

We can filter the collection to just one polygon, in a similar method to the image collection...

```
var india = ee.FeatureCollection('users/andrewmaclachlan/india')
    .filter('GID_1 == "IND.25_1"');
```

What does GEE look like



What typical processes can i do in GEE?

- Geometry operations (e.g. spatial operations)
    - Joins
    - Zonal statistics (e.g. average temperature per neighbourhood)
    - Filtering of images or specific values
- Methods
    - Machine learning
    - Supervised and unsupervised classification
    - Deep learning with Tensor Flow
    - Exploring relationships between variables

> Machine learning and deep learning are both types of AI. Deep learning is a subset of machine learning that uses artificial neural networks to mimic the learning process of the human brain

- Applications/outputs
    - Online charts
    - Scalable geospatial applications with GEE data
    - These let us query the data with a user inteface that then updates the results

Reducing images

This is combines the previous two ideas....

- In the first instance we load an image collection from a dates and place
  - Think about how we could do this in R (select, filter from `dplyr`)
- We want to reduce the collection to the extreme values for each pixel
  - Provide me the median value for each pixel from the collection
  - `collection.reduce` is a client side function but it always calls a server side function in `ee.Reducer`

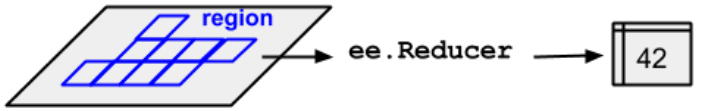
```
// Load an image collection, filtered so it's not too much data.
var collection = ee.ImageCollection('LANDSAT/LT05/C01/T1')
  .filterDate('2008-01-01', '2008-12-31')
  .filter(ee.Filter.eq('WRS_PATH', 44))
  .filter(ee.Filter.eq('WRS_ROW', 34));

// Compute the median in each band, each pixel.
// Band names are B1_median, B2_median, etc.
var median = collection.reduce(ee.Reducer.median());
```

Reducing images by region

One of the most useful functions we can use here is termed **zonal statistics** --- we saw this briefly in CASA0005.

In GEE this is termed `reduceRegion()`

in the first instance I might want to take an image and generate some statistics for it - e.g. the average reflectance for each band within the study area



Statistics of an Image Region. Source: GEE

Reducing images by region

GEE example...

```
// Load input imagery: Landsat 7 5-year composite.
var image = ee.Image('LANDSAT/LE7_TOA_5YEAR/2008_2012');

// Load an input region: Sierra Nevada.
var region = ee.Feature(ee.FeatureCollection('EPA/Ecoregions/2013/L3')
  .filter(ee.Filter.eq('us_l3name', 'Sierra Nevada'))
  .first());

// Reduce the region. The region parameter is the Feature geometry.
var meanDictionary = image.reduceRegion({
  reducer: ee.Reducer.mean(),
  geometry: region.geometry(),
  scale: 30,
  maxPixels: 1e9
});

// The result is a Dictionary.  Print it.
print(meanDictionary);
```

Reducer. Source: GEE

Reducing images by region(s)

What if we want to use a **feature collection (with many polygons)**...same idea...but with `image.reduceRegions()`

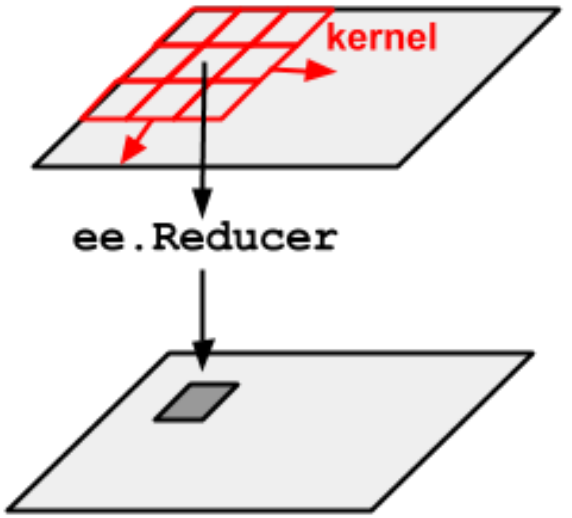
Note, the scale parameter.

```
// Load input imagery: Landsat 7 5-year composite.
var image = ee.Image('LANDSAT/LE7_TOA_5YEAR/2008_2012');

// Load a FeatureCollection of counties in Maine.
var maineCounties = ee.FeatureCollection('TIGER/2016/Counties')
  .filter(ee.Filter.eq('STATEFP', '23'));

// Add reducer output to the Features in the collection.
var maineMeansFeatures = image.reduceRegions({
  collection: maineCounties,
  reducer: ee.Reducer.mean(),
  scale: 30,
});
```

Reducers. Source: GEE

Reducing images by neighbourhood

- Instead of using a polygon to reduce our collection we can use the image neighbourhood
  - A window of pixels surrounding a central pixel
  - Like a filter or texture measure
  - Although texture (from previous weeks) has its own function



Statistics of Image Neighborhoods. Source: GEE

Reducing images by neighbourhood 2

```
// Define a region in the redwood forest.
var redwoods = ee.Geometry.Rectangle(-124.0665, 41.0739, -123.934, 41.2029);

// Load input NAIP imagery and build a mosaic.
var naipCollection = ee.ImageCollection('USDA/NAIP/DOQQ')
  .filterBounds(redwoods)
  .filterDate('2012-01-01', '2012-12-31');
var naip = naipCollection.mosaic();

// Compute NDVI from the NAIP imagery.
var naipNDVI = naip.normalizedDifference(['N', 'R']);

// Compute standard deviation (SD) as texture of the NDVI.
var texture = naipNDVI.reduceNeighborhood({
  reducer: ee.Reducer.stdDev(),
  kernel: ee.Kernel.circle(7),
});
```

Reducing images by neighbourhood 3

Figure 2a. NAIP imagery of the Northern California coast.

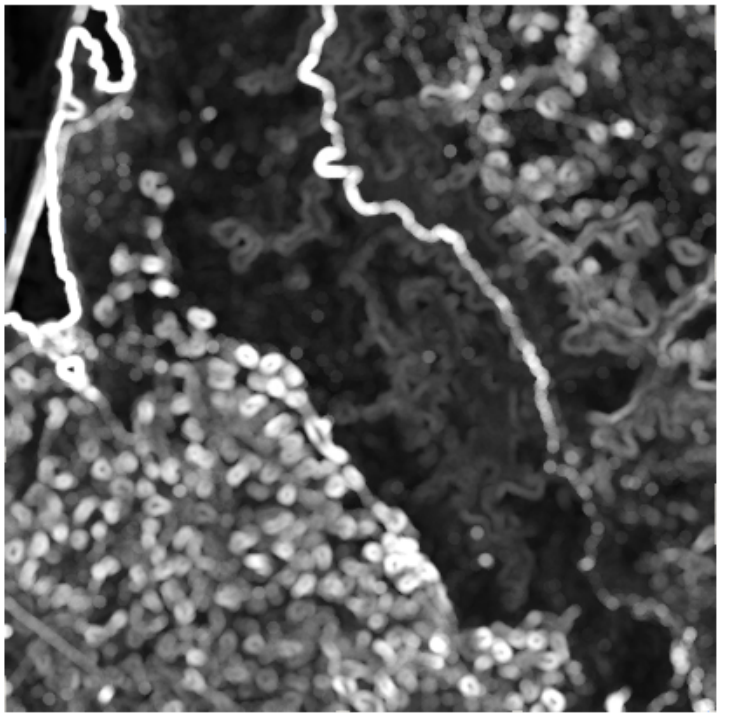


Figure 2b. `reduceNeighborhood()` output using a standard deviation reducer.

Statistics of an Image Region. Source: GEE

Recall regression....

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

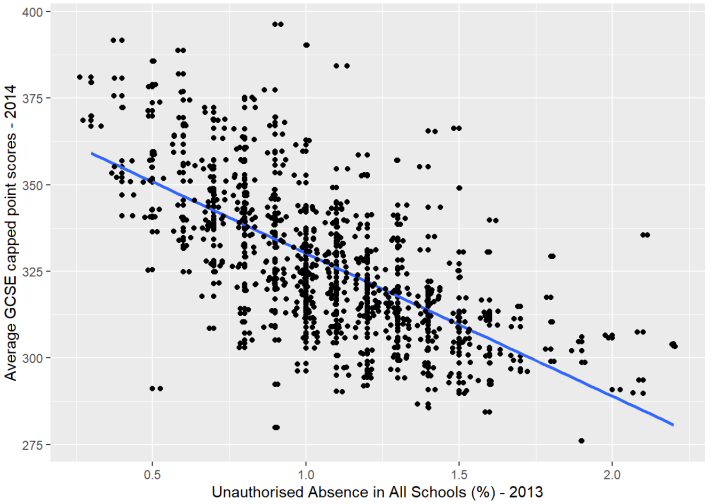
where:

β_0 is the intercept (the value of y when $x = 0$)

β_1 the 'slope' the change in the value of y for a 1 unit change in the value of x (the slope of the blue line)

ϵ_i is a random error term (positive or negative) - if you add all of the vertical differences between the blue line and all of the residuals, it should sum to 0.

Any value of y along the blue line can be modeled using the corresponding value of x



Source:CASA0005

what are the factors that might lead to variation in Average GCSE point scores:

- independent **predictor** variable (unauthorised absence)
- dependent variable (GCSE point score)

Linear regression 1

The real benefit of GEE is being able to access **all** imagery for multiple sensors...

What if we wanted to see **the change over time in pixel values** - `linearFit()`

`linearFit()` takes a least squares approach of one variable. 2 bands:

- Band 1: dependent variable
- Band 2: independent variable (often time)

This runs of a **per pixel** basis

This is still considered a **reducer** as we are reducing all of the data to two images:

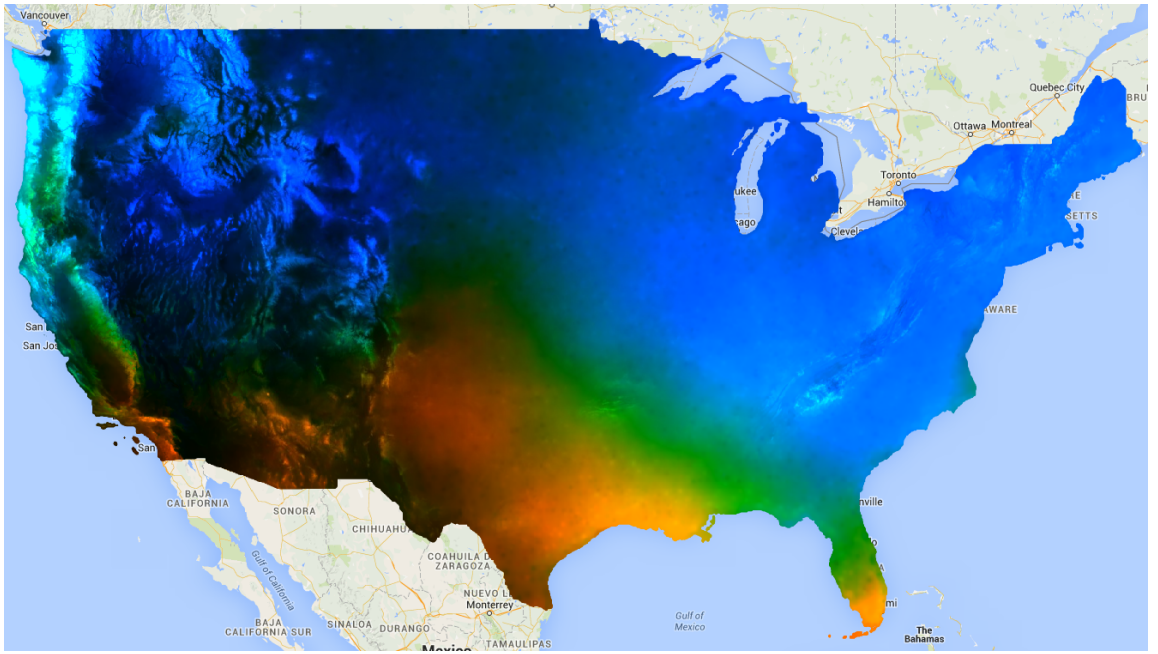
- Offset - intercept
- Scale - line of the slope

Linear regression 2

```
// This function adds a time band to the image.
var createTimeBand = function(image) {
  // Scale milliseconds by a large constant to avoid very small slopes
  // in the linear regression output.
  return image.addBands(image.metadata('system:time_start').divide(1e18));
};

// Load the input image collection: projected climate data.
var collection = ee.ImageCollection('NASA/NEX-DCP30_ENSEMBLE_STATS')
  .filter(ee.Filter.eq('scenario', 'rcp85'))
  .filterDate(ee.Date('2006-01-01'), ee.Date('2050-01-01'))
  // Map the time band function over the collection.
  .map(createTimeBand);

// Reduce the collection with the linear fit reducer.
// Independent variable are followed by dependent variables.
var linearFit = collection.select(['system:time_start', 'pr_mean'])
  .reduce(ee.Reducer.linearFit());
```

Linear Regression. Source: GEE

Linear regression 3



Increased precipitation are shown in blue and decreased precipitation in red (from the slope of the line)

Linear Regression. Source: GEE

Linear regression 4

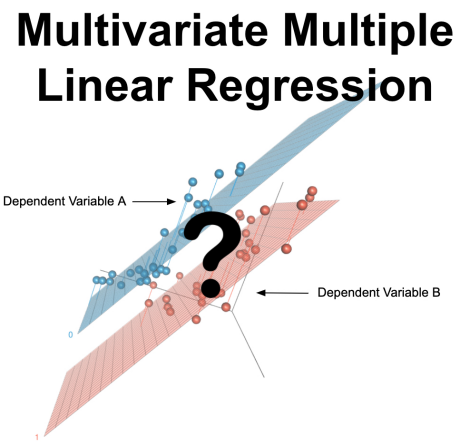
We can use additional variables like we have seen before, including multiple dependent variables...this is termed **Multivariate Multiple Linear Regression**

This just does the same as OLS for both of the dependent variables

In the GEE example...

precip + temp *are modelled by* a control variable (this means the intercept) and time.

The only difference is with a covariance matrix, see this towardsdatascience article from section 2



Multivariate Multiple Linear Regression. Source: statstest

Linear regression 5

You can combine reducers for regression, for example:

- Regression per pixels (typically with an image collection over several years)
- Regression of all the values within a polygon (taking an image of 1 date, extracting all the pixels and then running regression)

In **GEE** you must add a constant as an independent variable for **the intercept (unless you want it to be 0)**

In the following example, precip + temp *are modelled by* a control variable (this means the intercept) and time.

So where predicted temp and time increase per pixel then the coefficient will be positive

Linear Regression. Source: GEE

Linear regression 6

```
// Load the input image collection: project
var collection = ee.ImageCollection('NASA/N
  .filterDate(ee.Date('2006-01-01'), ee.Dat
  .filter(ee.Filter.eq('scenario', 'rcp85')
  // Map the functions over the collection,
  .map(createTimeBand)
  .map(createConstantBand)
  // Select the predictors and the response
  .select(['constant', 'system:time_start',

// Compute ordinary least squares regressio
var linearRegression = collection.reduce(
  ee.Reducer.linearRegression({
    numX: 2,
    numY: 2
}));
```

```
model1 <- Regressiondata %>%
  lm(average_gcse_capped_point_scores_2014
                unauthorised_absence_in_all_
     data=.)
```

Joins

Joins in GEE are similar to joins in R (think left_join()) BUT

- We can join image collections (e.g. satellite data from January with data from October)
- We can join feature collections (e.g. different polygons)

To use joins we have to put them within a filter (`ee.Filter`)

- The `leftField` is the index (or attribute) in the primary data
- The `rightField` is the secondary data
- We set the **type of join** (e.g.
  - simple: primary matches any in secondary
  - inverted: retain those in primary that are not in secondary
  - inner: shows all matches between collections as a feature collection
  -)
- We then combine (or join) with `join.apply()`

Joins 2

We can also do a spatial join....remember back to CASA0005

In the following code I have added to the GEE example:

- Select a national park
- Add all power plants within 100km of the park as a property **to the feature** (Yosemite here)...
- The joined attributes are in: Features > properties > points (100 elements)

Code

Joins 3

We can also intersect!

- Select a polygon (national park)
- Return the intersection of the points within each polygon(s) (e.g. all national parks)

And finally spatially subset

- Select the power plants
- Subset power plants within a polygon.

Code

Machine learning (we cover this next week)

Summary

- How GEE is setup
    - Raster = image
    - Collection = several images or polygons
    - Javascript into
    - Client vs server side
    - Loops and Mapping
- GEE functions and tools
    - Loading image collections
    - Reducing images (by region or regions or neighbourhoods)
    - Regression
    - Joins and filtering