R Google Earth Engine - Vol. 1

André Luiz Lima Costa

RGEE - created byr:

C Aybar, Q Wu, L Bautista, R Yali and A Barja (2020) rgee: An R package for interacting with Google Earth Engine Journal of Open Source Software URL https://github.com/r-spatial/rgee/

Index

INSTALLING

RGEE is a bridge between R and Google Earth Engine that elegantly and in a simple way makes possible the access to all capabilities of Google Earth Engine geospatial tool.

We will talk about the basic function as a starting point.

But first we have to install the main libraries that will support rgee, if they are not already present in your system. They are the **sf** and **mapview**. Also install the **remotes** package.

```
install.packages('sf')
install.packages('mapview')
install.packages('remotes')
```

Now we can install rgee, remember that you will need an active account with Google Earth Engine during the installation process.

```
library(remotes)
install_github("r-spatial/rgee")
```

After the install, execute the following two commands below just once to finalize the installation process.

```
library(rgee)
ee_install()
```

Execute the ee_Initialize() command. You will be directed to your Google Earth Engine and get access to it using rgee.

```
ee_Initialize()
```

That's it, rgee is ready to use now!!!

A BASIC OVERVIEW

We will make a quick tour showing how to execute basic process using rgee with Sentinel2 images. First we start rgee using:

```
library(rgee)
ee_Initialize()
── rgee 0.6.1 ──────────── earthengine-api 0.1.225 ──
    email: not_defined
    Initializing Google Earth Engine:  DONE!
    Earth Engine user: users/xxxxxxxx
──────────────────────────────────────────────────────
```

We are going to load a Sentinel2 RGB image using the bands 4, 3 and 2. This is done by creating an Image Collection that will be filtered by a date range and has a geometry point on it. Don't worry about these objects, we will cover them in detail as we move forward in this tutorial.
Loading the first image set that satisfy the filtering parameters defined by date (start, end) and **point coordinates** (longitude, latitude).

```
col<-ee$ImageCollection('COPERNICUS/S2_SR')
point <- ee$Geometry$Point(-44.366,-18.145)
start <- ee$Date("2019-07-11")
end <- ee$Date("2019-07-20")
filter<-col$filterBounds(point)$filterDate(start,end)
img <- filter$first()
```

Creating the visualization parameter list by listing the bands used, gamma correction for each band, minimum and maximum values used.

```
# RGB Visible
vPar <- list(bands = c("B4", "B3", "B2"),min = 100,max = 8000,
             gamma = c(1.9,1.7,1.7))
```

Finally we define the center of the map coordinates and the map scale before plotting the map with the predefined parameter.

```
Map$setCenter(-44.366,-17.69, zoom = 10)
Map$addLayer(img, vPar, "True Color Image")
```

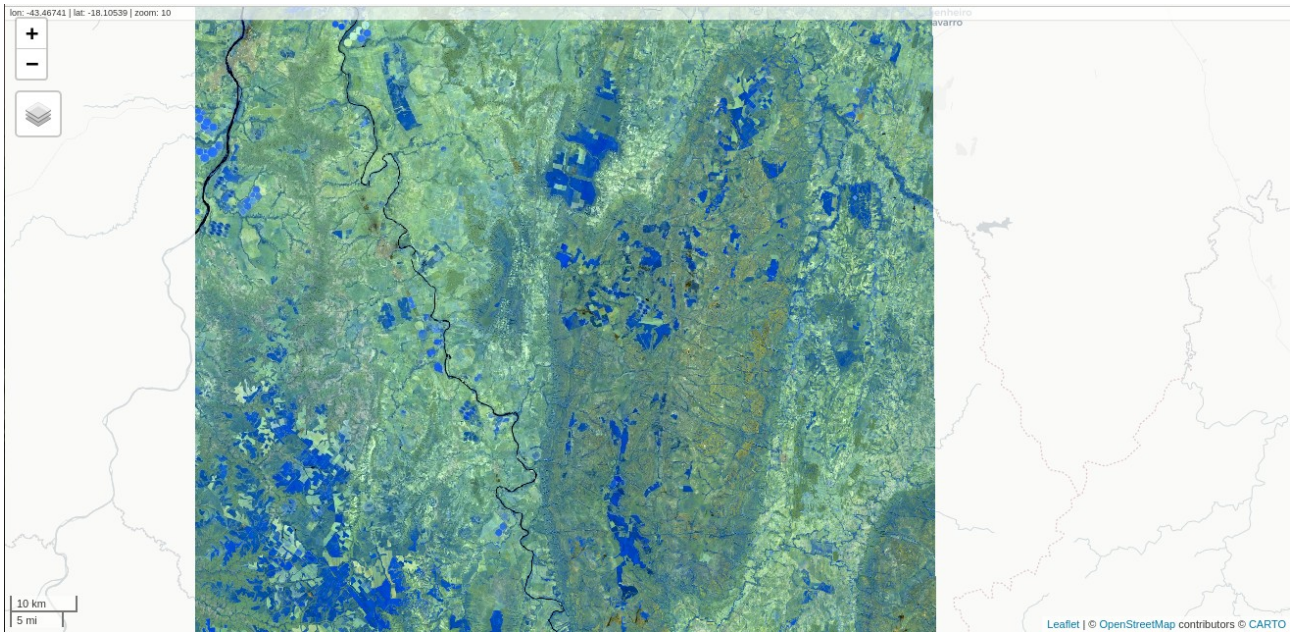
The resulting map will be:

By changing the visualization parameter we can change the image to be displayed. Below we show how to combine the bands 12, 11 and 8A into the RGB image that enhances rocks and soil responses.

```
# SWIR RGB Geology and soils
vPar2 <- list(bands = c("B12", "B11", "B8"),min = 500,max = 5000,
              gamma = 1.6)
```

and we plot it using:

```
Map$addLayer(img, vPar2, "Geology/Soil")
```



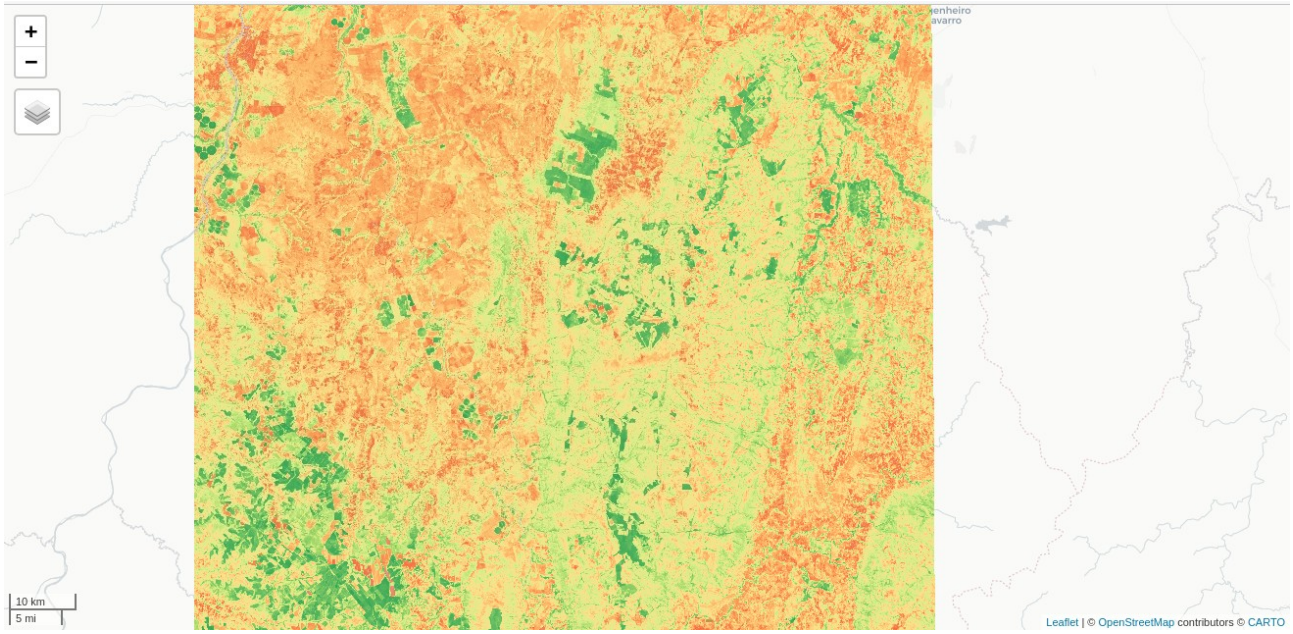
We will create now three normalized difference indexes for vegetation, water and moisture (NDVI, NDWI e NDMI) .

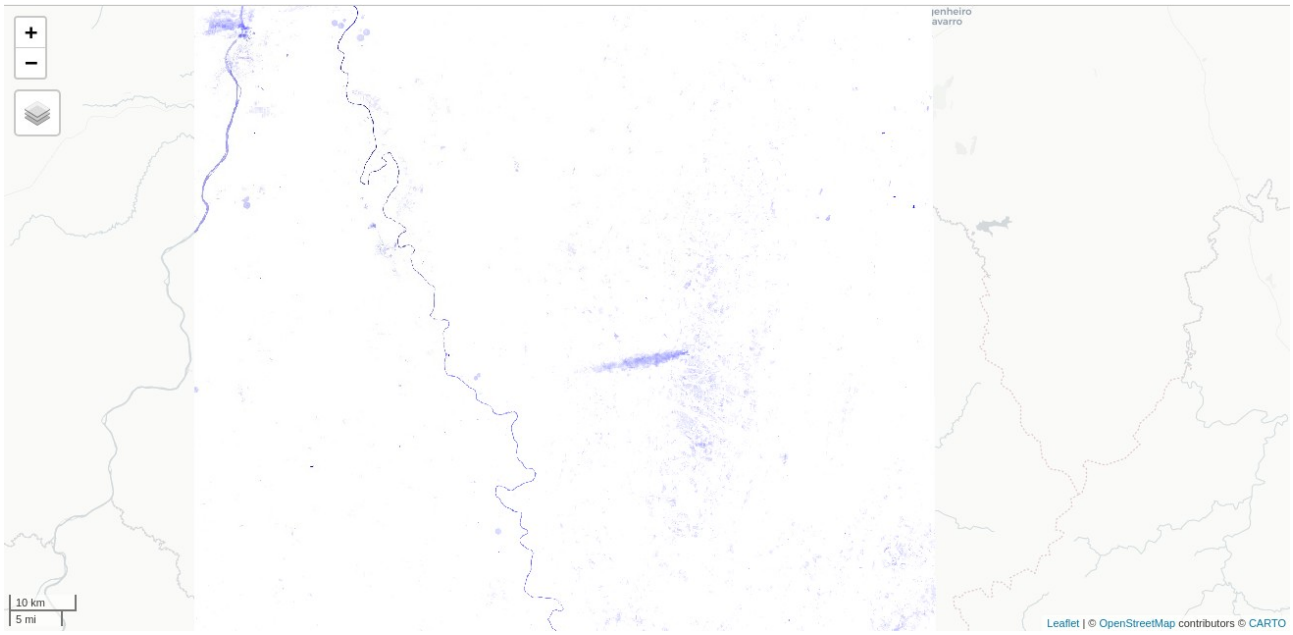
```
# NDVI
getNDVI <- function(image) {
  return(image$normalizedDifference(c("B8", "B4")))
}
ndvi1 <- getNDVI(img)
ndviPar <- list(palette = c(
  "#cccccc", "#f46d43", "#fdae61", "#fee08b",
  "#d9ef8b", "#a6d96a", "#66bd63", "#1a9850"
),min=0,max=1)


# NDWI
getNDWI <- function(image) {
  return(image$normalizedDifference(c("B3", "B5")))
}
ndwi <- getNDWI(img)
ndwiPar <- list(palette = c("#ffffff", "#0000ff","#0000ff"),min=-
0.25,max=0.75)

# NDMI
getNDMI <- function(image) {
  return(image$normalizedDifference(c("B8", "B11")))
}
ndmi <- getNDMI(img)
ndmiPar <- list(palette = c(
  "#d73027", "#f46d43", "#fdae61", "#fee08b",
  "#d9ef8b", "#a6d96a", "#66bd63", "#1a9850"
),min=-1,max=1)
```

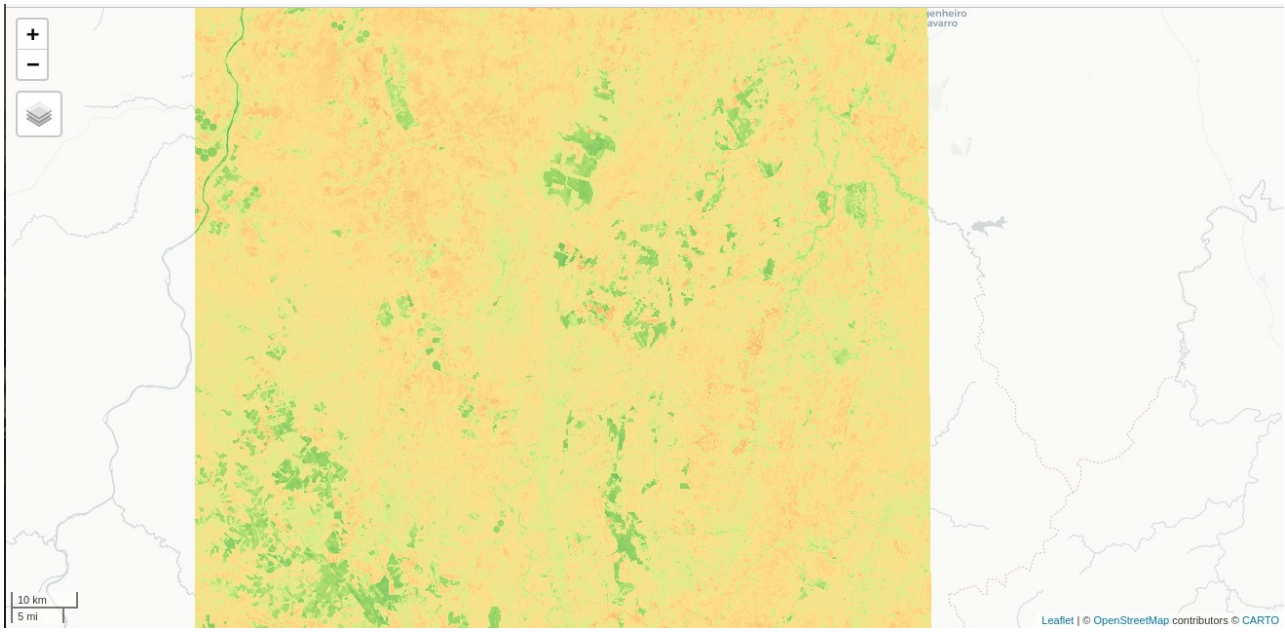
Creating the map with the three indexes:

```
#-----------------------------------------------------
# Centering the map and adding the images with the
#  parameter created above
#-----------------------------------------------------
Map$setCenter(-44.366,-17.69, zoom = 10)
Map$addLayer(ndvi1, ndviPar, "NDVI")+Map$addLayer(ndwi, ndwiPar,
"NDWI")+ Map$addLayer(ndmi, ndmiPar, "NDMI")
```

The result will be a map with three images representing the NDVI, NDWI and NDMI indexes.


10 km
5 mi
Leaflet | © OpenStreetMap contributors © CARTO


10 km
5 mi
Leaflet | © OpenStreetMap contributors © CARTO

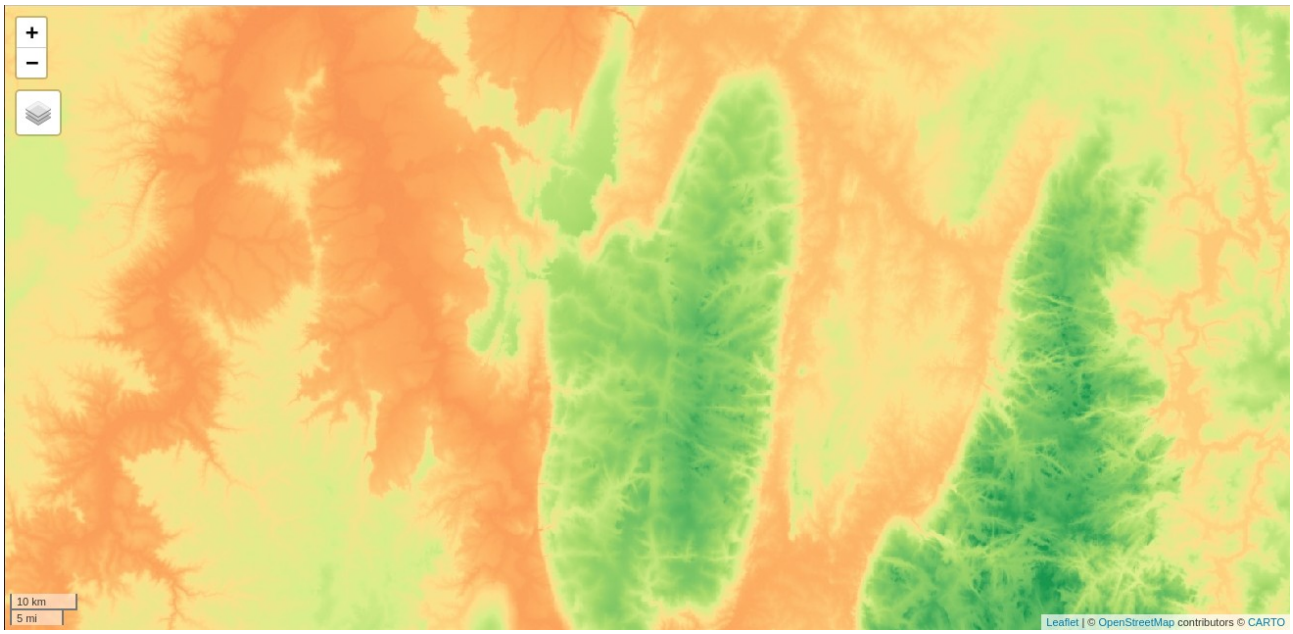

10 km
5 mi
Leaflet | © OpenStreetMap contributors © CARTO

Using Google Earth Engine is an advantage for us because most of the tasks are already algorithms/functions created within their system (and their huge data collections of different types of course). We just used one of these function above as an example, `image$nomalizedDifference(c(banda,banda))` to compute the normalized indexes, also we don't need to worry about different image resolutions of bands B5(20m) and B3(10m) used in NDWI, GEE takes care of it automatically.

Another example now using a DEM ALOS AW3D30 image.

```
library(rgee)
ee_Initialize()
imgData <- ee$Image("JAXA/ALOS/AW3D30_V1_1")
elev <- imgData$select("AVE")
Map$setCenter(-44.366,-17.69, zoom = 10)
Map$addLayer(eeObject=elev,visParams=list(palette = c(
 "#d73027", "#f46d43", "#fdae61", "#fee08b","#d9ef8b","#a6d96a",
 "#66bd63", "#1a9850"),min=200,max=1400),name="Elevation")
```

The resulting map is:



RGEE is very versatile and we will see a few more interesting examples before we deep in **ee$Image** and **ee$ImageCollection** in the following chapters.

EXAMPLES OF DATA FROM OTHER SOURCES

We can use images and features from different sources on a single map loading them from different resources and displaying them together.

We will use an Aster image with the bands B3N, B02 and B01 and also we will use a hill shade created from ALOS AW3D30 DEM and overlay it on top of the Aster image to create a topography perception on the final image.

```
library(rgee)
ee_Initialize()
# loading the Aster image collection
aster<-ee$ImageCollection("ASTER/AST_L1T_003")
point <- ee$Geometry$Point(-44.21722,-18.31933)
filter<-aster$filterBounds(point)$filterDate('2013-06-15', '2013-
09-15')

#selecting the bands
img <- filter$select(c('B3N','B02','B01'))
vPar <- list(min = 0,max = 255)

# degrees to radians transformation
Radians <- function(img) {
  img$toFloat()$multiply(base::pi)$divide(180)
}
# hillshade creation function
Hillshade <- function(az, ze, slope, aspect) {
  azimuth <- Radians(ee$Image(az))
  zenith <- Radians(ee$Image(ze))
  azimuth$subtract(aspect)$cos()$
    multiply(slope$sin())$
    multiply(zenith$sin())$
    add(zenith$cos()$multiply(slope$cos()))
}
# loading the ALOS DEM image
terrain <- ee$Algorithms$Terrain(ee$Image("JAXA/ALOS/AW3D30_V1_1")
$select("AVE"))
# extracting slope and aspect
slope_img <- Radians(terrain$select("slope"))
aspect_img <- Radians(terrain$select("aspect"))
# map central coordinate and zoom level
Map$setCenter(-44.21722,-18.31933, zoom = 11)
# Adding the 2 images -  hill shade 25%    opacity
Map$addLayer(img$median(), vPar, "Imagem Aster")+
Map$addLayer(Hillshade(135, -60, slope_img,
aspect_img),visParams=list(opacity=0.25),'Hill shade')
```

Image with hill shade.

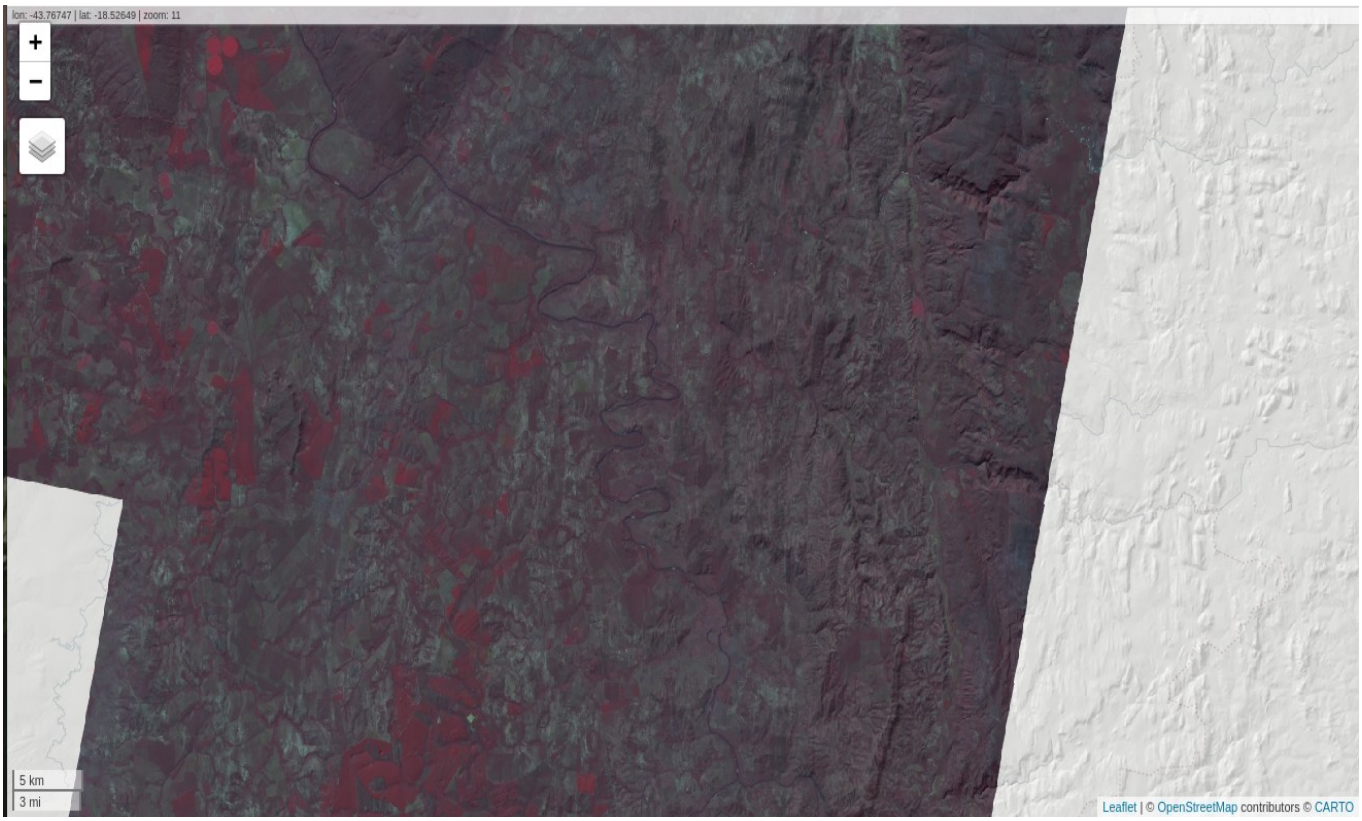
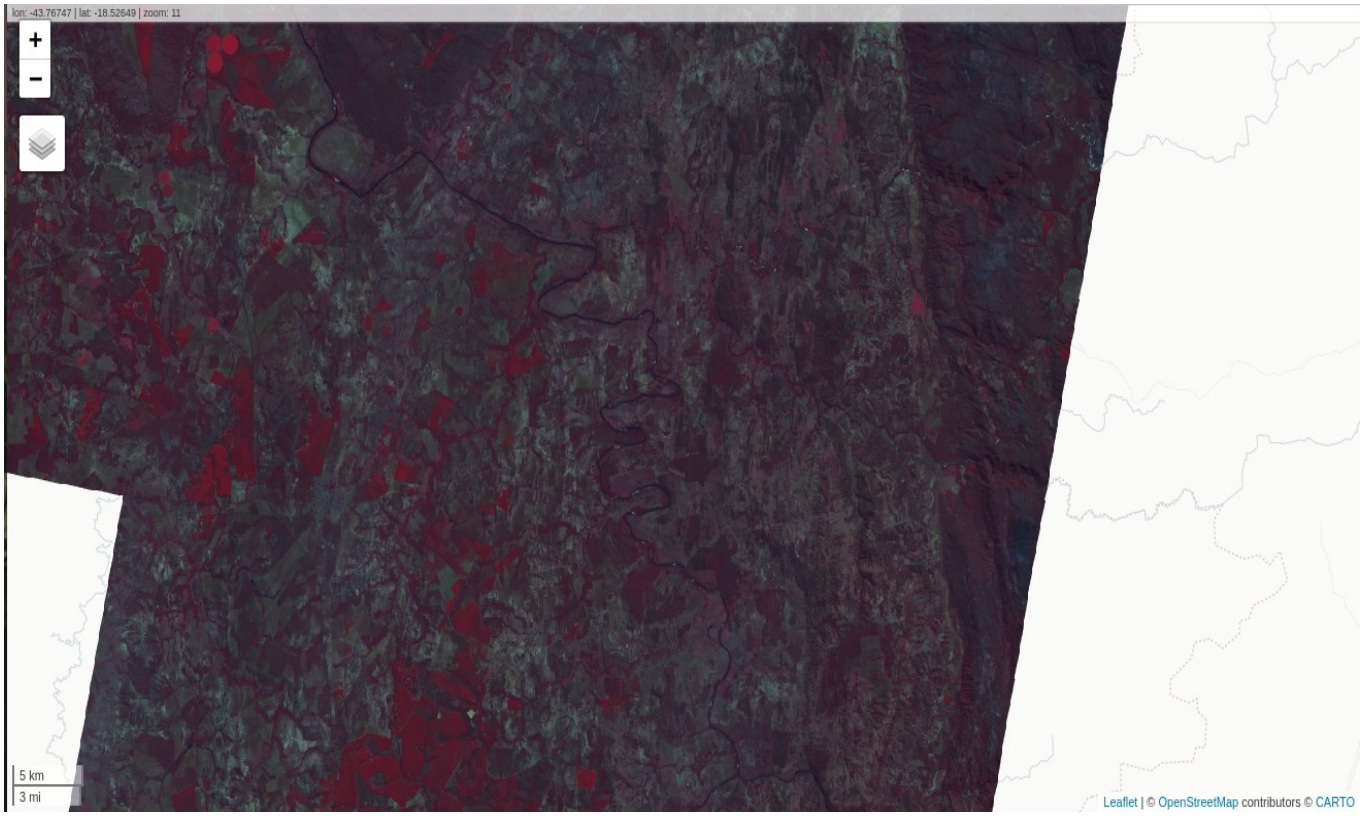
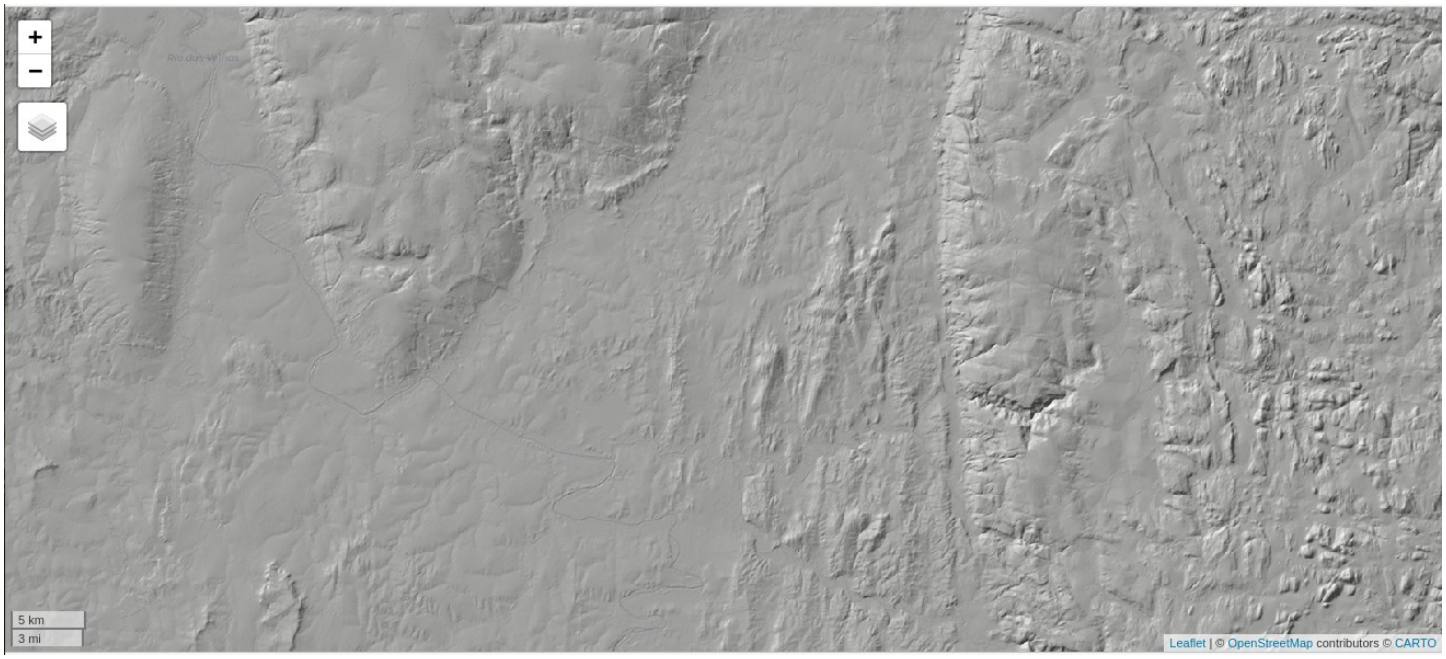


Image without hill shade.



Plotting only the hill shade image with 75% opacity:

```
Map$addLayer(Hillshade(135, -60, slope_img,aspect_img),visParams=
list(opacity=0.75),'Hill shade')
```

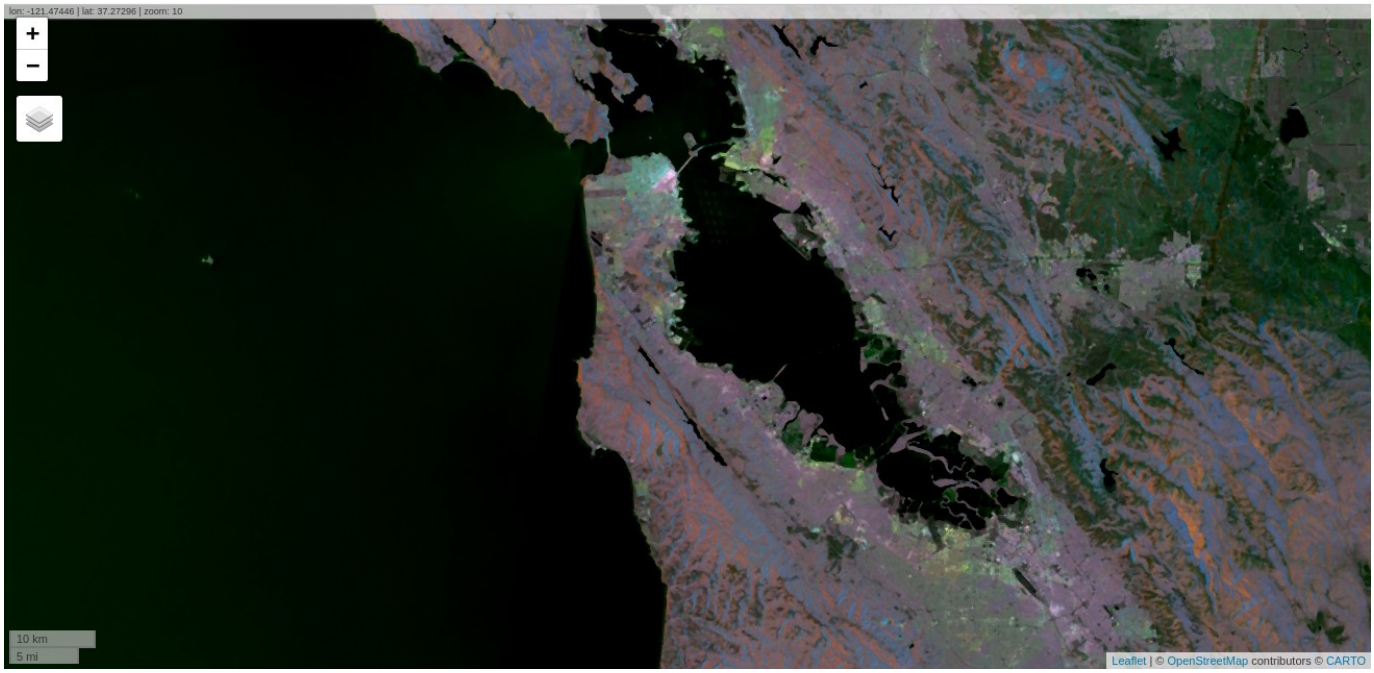


Using a Sentinel-1 image and creating a composite image composed by polarization and back scattering characteristics.

```
library(rgee)
ee_Initialize()

# loading Sentinel-1 into a ImageCollection.
sentinel1 <- ee$ImageCollection("COPERNICUS/S1_GRD")
$filterBounds(ee$Geometry$Point(-122.37383, 37.6193))

# Filtering using matadata information.
vh <-
sentinel1$filter(ee$Filter$listContains("transmitterReceiverPolari
sation", "VV"))
$filter(ee$Filter$listContains("transmitterReceiverPolarisation",
"VH"))$filter(ee$Filter$eq("instrumentMode", "IW"))

# Filtering to get images at different observation angle.
vhAscending <- vh$filter(ee$Filter$eq("orbitProperties_pass",
"ASCENDING"))
vhDescending <- vh$filter(ee$Filter$eq("orbitProperties_pass",
"DESCENDING"))

# Creating the final composite image
VH_Ascending_mean <- vhAscending$select("VH")$mean()
VV_Ascending_Descending_mean <- vhAscending$select("VV") %>%
ee$ImageCollection$merge(vhDescending$select("VV")) %>%
ee$ImageCollection$mean()
VH_Descending_mean <- vhDescending$select("VH")$mean()
composite <- ee$Image$cat(list(
  VH_Ascending_mean,
```

```
  VV_Ascending_Descending_mean,
  VH_Descending_mean
))$focal_median()

# Plotting the image composed by polarization and
# back scattering characteristics.
Map$setCenter(-122.37383, 37.6193, 10)
Map$addLayer(
  composite,
  list(min = c(-25, -20, -25), max = c(0, 10, 0)),
  "composite"
)
```

The resulting map is:



To conclude this introductory part, we will see how to download an image from Google Earth Engine. There is a limitation on the size of the image to be downloaded and the path generated is temporary.

```
library(rgee)
ee_Initialize()
# loading the image to be downloaded
image1 <- ee$ImageCollection('ASTER/AST_L1T_003')
$filterDate('2013-06-15', '2013-09-15')$median()
geom_params <-   list(
  scale = 30,
  crs = 'EPSG:4326',
  region = '[[-44.15, -18.25], [-44.35, -18.25], [-44.35, -18.35],
[-44.15, -18.35]]'
)
path <- image1$getDownloadUrl(geom_params)
print(path)
```

The path displayed is an url that will be used to download the image.

```
[1]
"https://earthengine.googleapis.com/v1alpha/projects/earthengine-
legacy/thumbnails/132ba605f16c0350a77b608984414295-
3dafacadd346ad41d9cfc658a765fd1f:getPixels"
```

Use wget at a command prompt (monitor) to download the compacted image and use unzip to extract it.

```
wget
https://earthengine.googleapis.com/v1alpha/projects/earthengine-
legacy/thumbnails/132ba605f16c0350a77b608984414295-
3dafacadd346ad41d9cfc658a765fd1f:getPixels

unzip -x "132ba605f16c0350a77b608984414295-
3dafacadd346ad41d9cfc658a765fd1f:getPixels"
```

Back into the R environment use the raster package that it is normally used to work with local images. Lets create a thermal and a color composite image with the downloaded data:

```
library(raster)
thermal<-
stack('download.B10.tif','download.B11.tif','download.B12.tif')
plotRGB(thermal,stretch='lin')
visible<-
stack('download.B3N.tif','download.B02.tif','download.B01.tif')
plotRGB(visible,stretch='lin')
```

Thermal image (90m resolution):

Visible range image (15m resolution):

A NOTE ON HOW GOOGLE EARTH ENGINE WORKS

Almost all transaction on google earth engine is basically the preparation of a JSON object that at the final stage is sent to google earth engine server and translated into a final image or feature that is presented on a map or viewed as properties using the function getinfo() or ee_print().

See the example below that illustrates how this is done:

```
library(rgee)
ee_Initialize()
── rgee 0.6.1 ──────────────────────── earthengine-api 0.1.225 ──
    email: not_defined
    Initializing Google Earth Engine:  DONE!
    Earth Engine user: users/?????????
──────────────────────────────────────────────────────────────────
aster<-ee$ImageCollection("ASTER/AST_L1T_003")
aster
ee.ImageCollection({
  "type": "Invocation",
  "arguments": {
    "id": "ASTER/AST_L1T_003"
  },
  "functionName": "ImageCollection.load"
})
```

```
point <- ee$Geometry$Point(-44.21722,-18.31933)
point
ee.Geometry({
  "type": "Point",
  "coordinates": [
    -44.21722,
    -18.31933
  ]
})
filter<-aster$filterBounds(point)$filterDate('2013-06-15', '2013-
09-15')
filter
ee.ImageCollection({
  "type": "Invocation",
  "arguments": {
    "collection": {
      "type": "Invocation",
      "arguments": {
        "collection": {
          "type": "Invocation",
          "arguments": {
            "id": "ASTER/AST_L1T_003"
          },
          "functionName": "ImageCollection.load"
        },
        "filter": {
          "type": "Invocation",
          "arguments": {
            "leftField": ".all",
            "rightValue": {
              "type": "Invocation",
              "arguments": {
                "geometry": {
                  "type": "Point",
                  "coordinates": [
                    -44.21722,
                    -18.31933
                  ]
                }
              },
              "functionName": "Feature"
            }
          },
          "functionName": "Filter.intersects"
        }
      },
      "functionName": "Collection.filter"
    },
    "filter": {
      "type": "Invocation",
      "arguments": {
        "rightField": "system:time_start",
        "leftValue": {
          "type": "Invocation",
          "arguments": {
            "start": "2013-06-15",
            "end": "2013-09-15"
          },
          "functionName": "DateRange"
        }
```

```
      },
      "functionName": "Filter.dateRangeContains"
    }
  },
  "functionName": "Collection.filter"
})
bands <- filter$select(c('B3N','B02','B01'))
bands
ee.ImageCollection({
  "type": "Invocation",
  "arguments": {
    "collection": {
      "type": "Invocation",
      "arguments": {
        "collection": {
          "type": "Invocation",
          "arguments": {
            "collection": {
              "type": "Invocation",
              "arguments": {
                "id": "ASTER/AST_L1T_003"
              },
              "functionName": "ImageCollection.load"
            },
            "filter": {
              "type": "Invocation",
              "arguments": {
                "leftField": ".all",
                "rightValue": {
                  "type": "Invocation",
                  "arguments": {
                    "geometry": {
                      "type": "Point",
                      "coordinates": [
                        -44.21722,
                        -18.31933
                      ]
                    }
                  },
                  "functionName": "Feature"
                }
              },
              "functionName": "Filter.intersects"
            }
          },
          "functionName": "Collection.filter"
        },
        "filter": {
          "type": "Invocation",
          "arguments": {
            "rightField": "system:time_start",
            "leftValue": {
              "type": "Invocation",
              "arguments": {
                "start": "2013-06-15",
                "end": "2013-09-15"
              },
              "functionName": "DateRange"
            }
          },
```

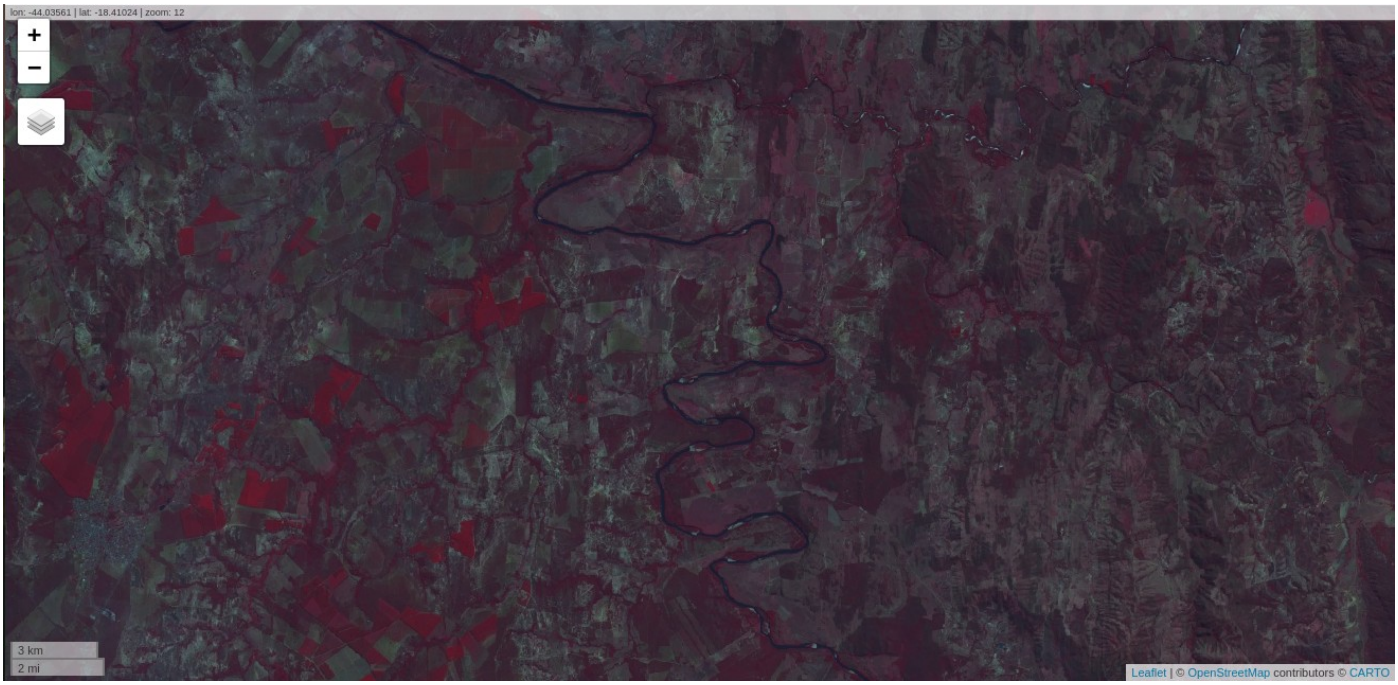
```
          "functionName": "Filter.dateRangeContains"
        }
      },
      "functionName": "Collection.filter"
    },
    "baseAlgorithm": {
      "type": "Function",
      "argumentNames": [
        "_MAPPING_VAR_0_0"
      ],
      "body": {
        "type": "Invocation",
        "arguments": {
          "input": {
            "type": "ArgumentRef",
            "value": "_MAPPING_VAR_0_0"
          },
          "bandSelectors": [
            "B3N",
            "B02",
            "B01"
          ]
        },
        "functionName": "Image.select"
      }
    }
  },
  "functionName": "Collection.map"
})
img <- bands$median()
img
ee.Image({
  "type": "Invocation",
  "arguments": {
    "collection": {
      "type": "Invocation",
      "arguments": {
        "collection": {
          "type": "Invocation",
          "arguments": {
            "collection": {
              "type": "Invocation",
              "arguments": {
                "collection": {
                  "type": "Invocation",
                  "arguments": {
                    "collection": {
                      "type": "Invocation",
                      "arguments": {
                        "id": "ASTER/AST_L1T_003"
                      },
                      "functionName": "ImageCollection.load"
                    },
                    "filter": {
                      "type": "Invocation",
                      "arguments": {
                        "leftField": ".all",
                        "rightValue": {
                          "type": "Invocation",
                          "arguments": {
                            "geometry": {
                              "type": "Point",
                              "coordinates": [
```

```
                            -44.21722,
                            -18.31933
                          ]
                        }
                      },
                      "functionName": "Feature"
                    }
                  },
                  "functionName": "Filter.intersects"
                }
              },
              "functionName": "Collection.filter"
            },
            "filter": {
              "type": "Invocation",
              "arguments": {
                "rightField": "system:time_start",
                "leftValue": {
                  "type": "Invocation",
                  "arguments": {
                    "start": "2013-06-15",
                    "end": "2013-09-15"
                  },
                  "functionName": "DateRange"
                }
              },
              "functionName": "Filter.dateRangeContains"
            }
          },
          "functionName": "Collection.filter"
        },
        "baseAlgorithm": {
          "type": "Function",
          "argumentNames": [
            "_MAPPING_VAR_0_0"
          ],
          "body": {
            "type": "Invocation",
            "arguments": {
              "input": {
                "type": "ArgumentRef",
                "value": "_MAPPING_VAR_0_0"
              },
              "bandSelectors": [
                "B3N",
                "B02",
                "B01"
              ]
            },
            "functionName": "Image.select"
          }
        }
      },
      "functionName": "Collection.map"
    }
  },
  "functionName": "reduce.median"
})
```

Now we center the map and plot it.

```
Map$setCenter(-44.21722,-18.31933, zoom = 12)
Map$addLayer(img, visParams=list(min=0,max=255),"Aster Image")
```

The resulting map is:



The HTML source code of this map is (Note where the map image is called in bold):

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<style>body{background-color:white;}</style>
...
script>
<script src="lib/clipboard-0.0.1/setClipboardText.js"></script>

</head>
<body>
<div id="htmlwidget_container">
  <div id="htmlwidget-56d53ad4c5aa44506bdd" style="width:100%;height:400px;"
class="leaflet html-widget"></div>
</div>
<script type="application/json" data-for="htmlwidget-56d53ad4c5aa44506bdd">{"x":
{"options":{"minZoom":1,"maxZoom":52,"crs":
{"crsClass":"L.CRS.EPSG3857","code":null,"proj4def":null,"projectedBounds":null,
"options":{}},"preferCanvas":false,"bounceAtZoomLimits":false,"maxBounds":[[[-
90,-370]],[[90,370]]]},"calls":[{"method":"addProviderTiles","args":
["CartoDB.Positron",1,"CartoDB.Positron",
{"errorTileUrl":"","noWrap":false,"detectRetina":false}]},
{"method":"addProviderTiles","args":
["CartoDB.DarkMatter",2,"CartoDB.DarkMatter",
{"errorTileUrl":"","noWrap":false,"detectRetina":false}]},
{"method":"addProviderTiles","args":["OpenStreetMap",3,"OpenStreetMap",
{"errorTileUrl":"","noWrap":false,"detectRetina":false}]},
{"method":"addProviderTiles","args":["Esri.WorldImagery",4,"Esri.WorldImagery",
```

{"errorTileUrl":"","noWrap":false,"detectRetina":false}]},
{"method":"addProviderTiles","args":["OpenTopoMap",5,"OpenTopoMap",
{"errorTileUrl":"","noWrap":false,"detectRetina":false}]},
{"method":"addLayersControl","args":
[["CartoDB.Positron","CartoDB.DarkMatter","OpenStreetMap","Esri.WorldImagery","O
penTopoMap"],[],{"collapsed":true,"autoZIndex":true,"position":"topleft"}]},
{"method":"addScaleBar","args":
[{"maxWidth":100,"metric":true,"imperial":true,"updateWhenIdle":true,"position":
"bottomleft"}]},**{"method":"addTiles","args":["https://
earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/maps/
bf13ded46a01aef419158c6a870b572c-be9082f1e8f40fb31898a4fd75c18ad2/tiles/{z}/{x}/
{y}",null,"Imagem Aster",
{"minZoom":0,"maxZoom":18,"tileSize":256,"subdomains":"abc","errorTileUrl":"","t
ms":false,"noWrap":false,"zoomOffset":0,"zoomReverse":false,"opacity":1,"zIndex"
:1,"detectRetina":false}]},**{"method":"addLayersControl","args":
[["CartoDB.Positron","CartoDB.DarkMatter","OpenStreetMap","Esri.WorldImagery","O
penTopoMap"],"Imagem Aster",
{"collapsed":true,"autoZIndex":true,"position":"topleft"}]},
{"method":"hideGroup","args":[null]}],"setView":[[-18.31933,-44.21722],11,
[]]},"evals":[],"jsHooks":{"render":[{"code":"function(el, x, data) {\n return
(\n   function(el, x, data) {\n   // get the leaflet map\n   var map =
this; //HTMLWidgets.find('#' + el.id);\n   // we need a new div element
because we have to handle\n   // the mouseover output separately\n   //
debugger;\n   function addElement () {\n   // generate new div Element\n
var newDiv = $(document.createElement('div'));\n   // append at end of
leaflet htmlwidget container\n $(el).append(newDiv);\n   //provide ID
and style\n   newDiv.addClass('lnlt');\n   newDiv.css({\n   'position':
'relative',\n   'bottomleft': '0px',\n   'background-color': 'rgba(255,
255, 255, 0.7)',\n   'box-shadow': '0 0 2px #bbb',\n   'background-clip':
'padding-box',\n   'margin': '0',\n   'padding-left': '5px',\n
'color': '#333',\n   'font': '9px/1.5 \"Helvetica Neue\", Arial, Helvetica,
sans-serif',\n   'z-index': '700',\n   });\n   return newDiv;\n   }\
n\n\n   // check for already existing lnlt class to not duplicate\n   var
lnlt = $(el).find('.lnlt');\n\n   if(!lnlt.length) {\n   lnlt =
addElement();\n\n   // grab the special div we generated in the beginning\n
// and put the mousmove output there\n\n   map.on('mousemove', function (e)
{\n   if (e.originalEvent.ctrlKey) {\n   if
(document.querySelector('.lnlt') === null) lnlt = addElement();\n
lnlt.text(\n   ' lon: ' + (e.latlng.lng).toFixed(5) +\n
' | lat: ' + (e.latlng.lat).toFixed(5) +\n   ' | zoom: '
+ map.getZoom() +\n   ' | x: ' +
L.CRS.EPSG3857.project(e.latlng).x.toFixed(0) +\n   ' |
y: ' + L.CRS.EPSG3857.project(e.latlng).y.toFixed(0) +\n
' | epsg: 3857 ' +\n   ' | proj4: +proj=merc +a=6378137
+b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m
+nadgrids=@null +no_defs ');\n   } else {\n   if
(document.querySelector('.lnlt') === null) lnlt = addElement();\n
lnlt.text(\n   ' lon: ' + (e.latlng.lng).toFixed(5) +\n
' | lat: ' + (e.latlng.lat).toFixed(5) +\n   ' | zoom: ' +
map.getZoom() + ' ');\n   }\n   });\n\n   // remove the lnlt div when
mouse leaves map\n   map.on('mouseout', function (e) {\n   var strip =
document.querySelector('.lnlt');\n   strip.remove();\n   });\n\n   };\
n\n   //$(el).keypress(67, function(e) {\n   map.on('preclick',
function(e) {\n   if (e.originalEvent.ctrlKey) {\n   if
(document.querySelector('.lnlt') === null) lnlt = addElement();\n
lnlt.text(\n   ' lon: ' + (e.latlng.lng).toFixed(5) +\n
' | lat: ' + (e.latlng.lat).toFixed(5) +\n   ' | zoom: ' +
map.getZoom() + ' ');\n   var txt =
document.querySelector('.lnlt').textContent;\n   console.log(txt);\n
//txt.innerText.focus();\n   //txt.select();\n   setClipboardText('\"' +
txt + '\"');\n   }\n   });\n\n   }\n   ).call(this.getMap(), el, x,
data);\n}","data":null}]}}</script>
<script type="application/htmlwidget-sizing" data-for="htmlwidget-
56d53ad4c5aa44506bdd">{"viewer":
{"width":"100%","height":400,"padding":0,"fill":true},"browser":
{"width":"100%","height":400,"padding":0,"fill":true}}</script>
</body>
</html>

On a very simplistic way, all the code we wrote was to prepare a JSON text with the specifications of the map we wanted. This JSON is sent to the guts of Google Earth Engine that follow the JSON instructions to prepare a map and send you back a temporary URL of {z}{x}{y} tiles of your requested map to its server.

https://earthengine.googleapis.com/v1alpha/projects/earthengine-legacy/maps/bf13ded46a01aef419158c6a870b572c-be9082f1e8f40fb31898a4fd75c18ad2/tiles/{z}/{x}/{y}

This url is used to dynamically load the tiles into your map (a leaflet map in our case).

We don't have to worry about the "JSONish" side, the rgee will take care of it in the same way that python and JS do. They create the JSON dialect that is sent to Google Earth Engine and receive the corresponding url or data.

We will cover now the main objects used by Google Earth Engine, what they are and what can we do with them. We will start with ee$Image.

ee$Image

An image (ee$Image object) is a raster data that it represented as an object in Earth Engine. Images are composed by one or more bands and each band has its own name, data type, scale (resolution), mask and projection. Each image also has a metadata with a set of image properties.

The image object (and the feature object as well) is the final goal in using Earth Engine and we are going to start by checking its characteristics and properties.

ee$Image Constructor

Images can be retrieved by using the data ID that it is present in Google Earth Engine as the ee$Image constructor parameter. A list of image dataset IDs can be found using the link https://developers.google.com/earth-engine/datasets.

Some datasets already generate a ee$Image object and others a ee$ImageCollection.

For example, to load a JAXA ALOS DEM of ee$Image we use:

```
library(rgee)
ee_Initialize()
#loading a global dem
dem<-ee$Image("JAXA/ALOS/AW3D30_V1_1")
```

Use ee_print() to retrieve the main information of this image object.

```
ee_print(dem,clean=TRUE)
```

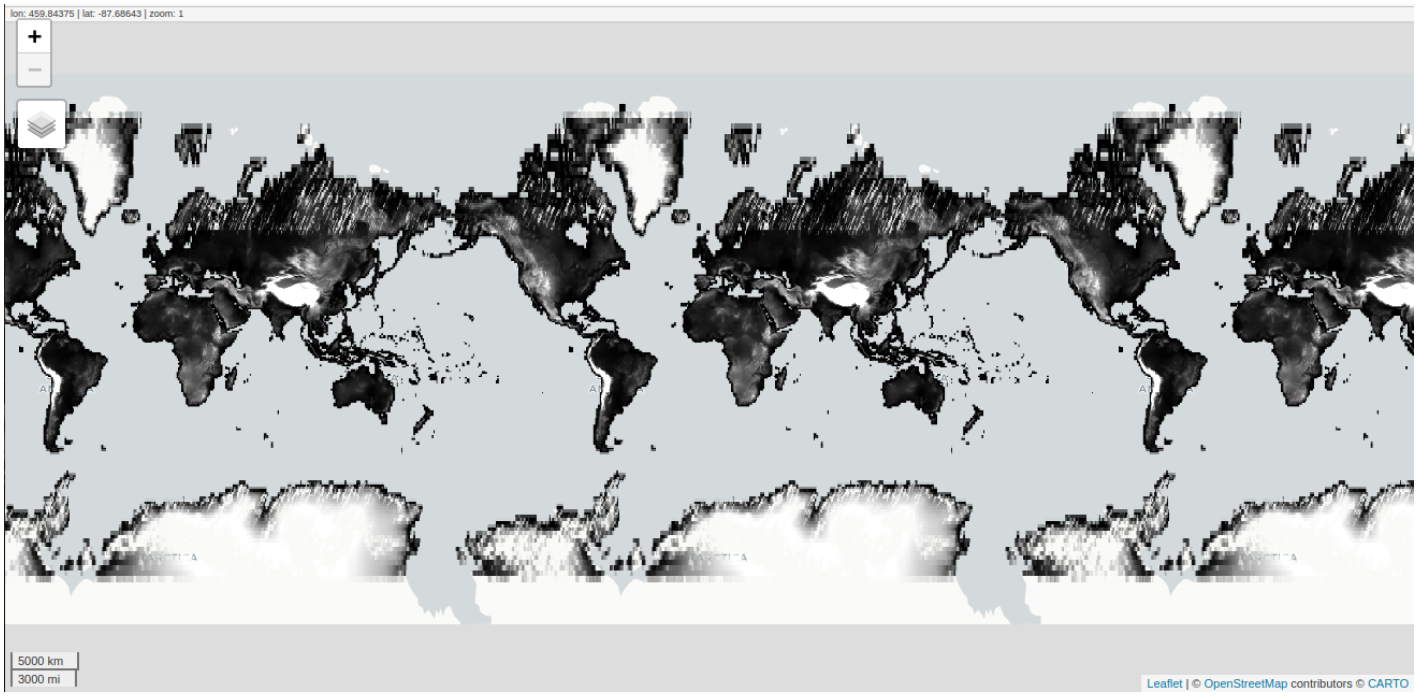
```
──────────────────────────────────────────────────────────── Earth Engine Image ──
Image Metadata:
 - Class                       : ee$Image
 - ID                          : JAXA/ALOS/AW3D30_V1_1
 - Number of Bands             : 6
 - Bands names                 : AVE MED AVE_STK MED_STK AVE_MSK MED_MSK
 - Number of Properties        : 25
 - Number of Pixels*           : 5.038848e+12
 - Approximate size*           : 2.20 TB
Band Metadata (img_band = AVE):
 - EPSG (SRID)                 : 4326
 - proj4string                 : +proj=longlat +datum=WGS84 +no_defs
 - Geotransform                : 0.0002777777778 0 -180 0 -0.0002777777778 83
 - Nominal scale (meters)      : 30.92208
 - Dimensions                  : 1296000 648000
 - Number of Pixels            : 8.39808e+11
 - Data type                   : INT
 - Approximate size            : 374.86 GB
──────────────────────────────────────────────────────────────────────────────────
```

In plotting this image we will see that it is composed by several pixels and by plotting the band AVE or MED, elevation values are associated with each one of the image pixels.

```
Map$addLayer(dem, visParams=list(),"DEM")
Map$addLayer(dem$select("AVE"),visParams=list(min=0,max=3000),
"DEM AVE BAND")
```

dem


```
dem$select("AVE")
```



Now creating an image from an ImageCollection of Sentinel2 type.

```
library(rgee)
ee_Initialize()
# Selecting a ee$ImageCollectionn
col<-ee$ImageCollection('COPERNICUS/S2_SR')
point <- ee$Geometry$Point(-44.366,-18.145)
start <- ee$Date("2019-07-11")
end <- ee$Date("2019-07-20")
filter<-col$filterBounds(point)$filterDate(start,end)
#Getting the first filtered image
#  ee$ImageCollection to ee$Image
img <- filter$first()
```

Inspecting the Image object values with ee_print().

```
ee_print(img,clear=TRUE)
```

```
── Earth Engine Image ──
Image Metadata:
 - Class               : ee$Image
 - ID                  : COPERNICUS/S2_SR/20190711T131251_20190711T131248_T23KNA
 - Time start          : 2019-07-11 13:17:13
 - Number of Bands     : 23
 - Bands names         : B1 B2 B3 B4 B5 B6 B7 B8 B8A B9 B11 B12 AOT WVP SCL
TCI_R TCI_G TCI_B MSK_CLDPRB MSK_SNWPRB QA10 QA20 QA60
 - Number of Properties: 81
 - Number of Pixels*   : 77066790
 - Approximate size*   : 38.39 GB
Band Metadata (img_band = B1):
 - EPSG (SRID)         : 32723
 - proj4string         : +proj=utm +zone=23 +south +datum=WGS84 +units=m
+no_defs
 - Geotransform        : 60 0 499980 0 -60 8100040
 - Nominal scale(m)    : 60
 - Dimensions          : 1831 1830
 - Number of Pixels    : 3350730
 - Data type           : INT
 - Approximate size    : 1.67 GB
```

Plotting a map with a RGB composed by bands 4 ,3 e 2.

```
Map$setCenter(-44.366,-17.69, zoom = 10)
Map$addLayer(img,visParams=list(bands = c("B4", "B3", "B2"),min =
100,max = 8000,gamma = c(1.9,1.7,1.7)), "True Color Image")
```

We can create an Image object from a geotiff cloud using ee$Image$loadGeoTIFF() from images stored at the Gloogle Cloud Storage. For example, this Landsat data stored at google cloud composed by band 5 extracted from a landsat 8 dataset can be loaded using:

```
library(rgee)
ee_Initialize()
uri<-'gs://gcp-public-data-landsat/LC08/01/001/002/
LC08_L1GT_001002_20160817_20170322_01_T2/
LC08_L1GT_001002_20160817_20170322_01_T2_B5.TIF'
cloudImage<-ee$Image$loadGeoTIFF(uri)
```

Visualizing the info with ee_print()

```
ee_print(cloudImage, clean = TRUE)
```

```
──────────────────────────────────────── Earth Engine Image ──
Image Metadata:
 - Class                      : ee$Image
 - ID                         : no_id
 - Number of Bands            : 1
 - Bands names                : B0
 - Number of Properties       : 0
 - Number of Pixels*          : 80658261
 - Approximate size*          : 123.07 MB
Band Metadata (img_band = B0):
 - EPSG (SRID)                : 32630
 - proj4string                : +proj=utm +zone=30 +datum=WGS84 +units=m
+no_defs
 - Geotransform               : 30 0 342000 0 -30 9016200
 - Nominal scale (meters)     : 30
 - Dimensions                 : 8991 8971
 - Number of Pixels           : 80658261
 - Data type                  : INT
 - Approximate size           : 123.07 MB
──────────────────────────────────────────────────────────────
```

The next example shows how to create and manipulate a constant Image object:

```
library(rgee)
ee_Initialize()
image<-ee$Image(1)
ee_print(image, clean = TRUE)
```

```
── Earth Engine Image ──
Image Metadata:
 - Class                      : ee$Image
 - ID                         : no_id
 - Number of Bands            : 1
 - Bands names                : constant
 - Number of Properties       : 0
 - Number of Pixels*          : 64800
 - Approximate size*          : 101.25 KB
Band Metadata (img_band = constant):
 - EPSG (SRID)                : 4326
 - proj4string                : +proj=longlat +datum=WGS84 +no_defs
 - Geotransform               : 1 0 0 0 1 0
 - Nominal scale (meters)     : 111319.5
 - Dimensions                 : 360 180
 - Number of Pixels           : 64800
 - Data type                  : INT
 - Approximate size           : 101.25 KB
NOTE:(*)Properties calculated considering a constant geotransform and data type.
```

Renaming the bands using

```
threebands <- ee$Image(c(1, 2, 3))
imagem2 <- threebands$select(
  opt_selectors = c("constant", "constant_1", "constant_2"),
  opt_names = c("band1", "band2", "band3")
)
ee_print(imagem2)
```

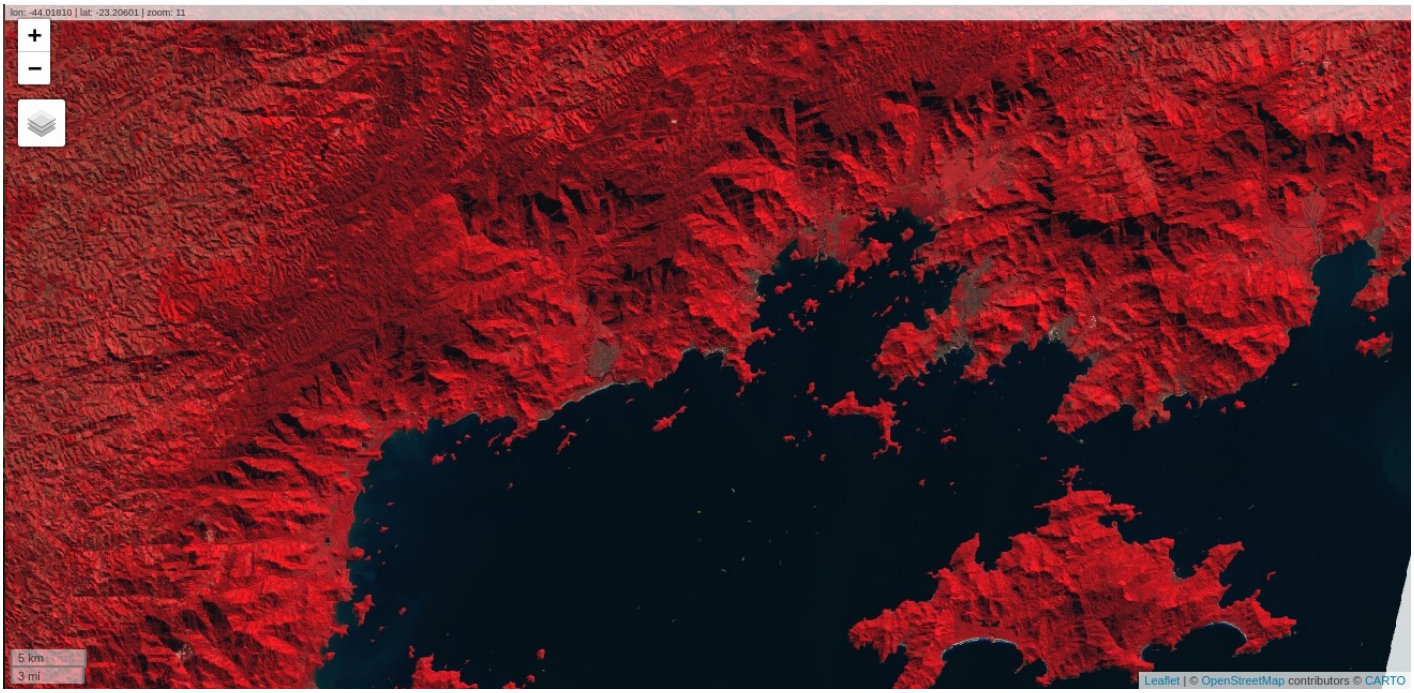
```
── Earth Engine Image ──
Image Metadata:
 - Class                      : ee$Image
 - ID                         : no_id
 - Number of Bands            : 3
 - Bands names                : band1 band2 band3
 - Number of Properties       : 0
 - Number of Pixels*          : 194400
 - Approximate size*          : 303.75 KB
Band Metadata (img_band = banda1):
 - EPSG (SRID)                : 4326
 - proj4string                : +proj=longlat +datum=WGS84 +no_defs
 - Geotransform               : 1 0 0 0 1 0
 - Nominal scale (meters)     : 111319.5
 - Dimensions                 : 360 180
 - Number of Pixels           : 64800
 - Data type                  : INT
 - Approximate size           : 101.25 KB
```

ee$Image visualization

When we call the function Map$addLayer the second parameter is a list of visualization properties and parameters that will be used to display the object into the map. They are:
bands : a vector of bands that will be associated with a RGB image.
min : a number or a three numbers vector that will be the minimum value used by the band (or bands).
max : a number or a three numbers vector that will be the maximum value used by the band (or bands).
gain : a number or a three number vector that will be the gain used by the band (or bands).
bias : a number or a three number vector that will be the gain added to each DN (Digital Number / pixel) of the band (or bands).
gamma : a number or a three number vector that will be the correction value of the band (or bands).
palette : a string vector of colors in CSS style that will be applied to a color ramp for the image. Only for single band images.
opacity : value use to adjust the image transparency (0 totally transparent 1 totally opaque).

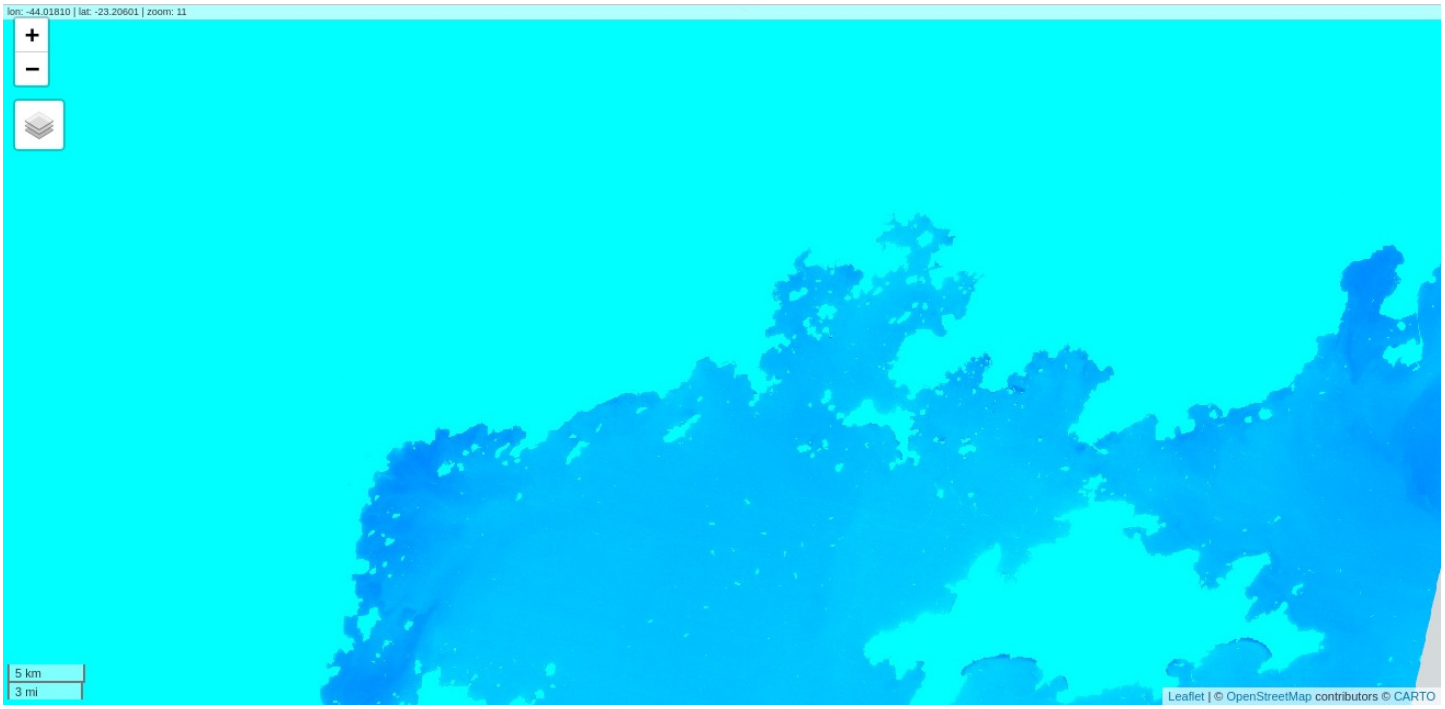
Creating a RGB composite based on visualization parameter:

```
library(rgee)
ee_Initialize()
col<-ee$ImageCollection('LANDSAT/LC08/C01/T1_TOA')
point <- ee$Geometry$Point(-44.4678,-23.0058)
start <- ee$Date("2019-06-11")
end <- ee$Date("2019-08-20")
filter<-col$filterBounds(point)$filterDate(start,end)
image <- filter$first()
imgViz <- list(
  min = 0,
  max = 0.5,
  bands = c("B5", "B4", "B3"),
  gamma = c(0.95, 1.1, 1)
)
Map$setCenter(-44.4678,-23.0058, 11)
Map$addLayer(image, imgViz,'Landsat 8 Image')
```



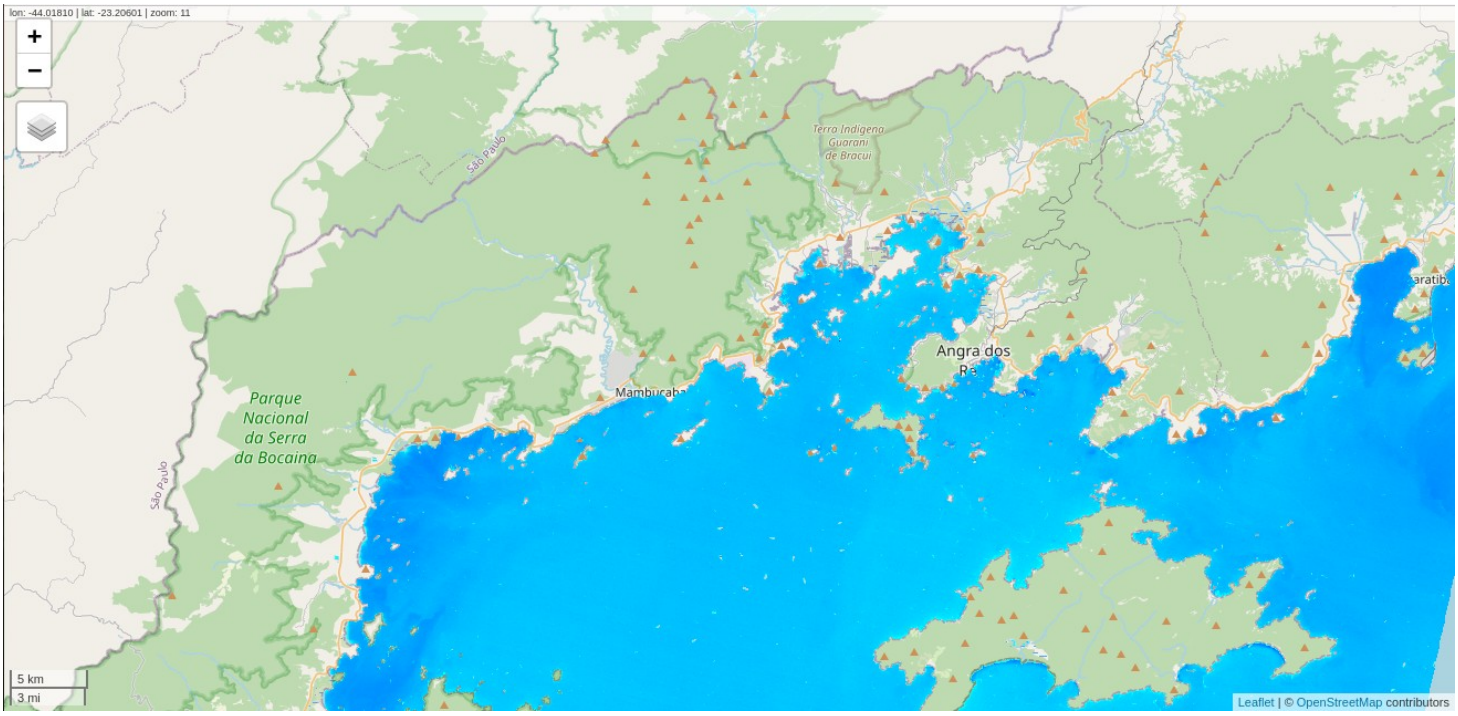
Now creating a single band NDWI using a color palette.

```
#Creating a ndwi
ndwi <- image$normalizedDifference(c("B3", "B5"))
ndwiViz <- list(
  min = 0.5,
  max = 1,
  palette = c("00FFFF",'0080FF', "0000FF")
)
Map$addLayer(ndwi, ndwiViz,'NDWI')
```



Creating a mask using image$updateMask() to make opaque (visible) the valid values (not null) below a certain value in the NDWI image, in this example, values smaller than 0.4, that represents values that are not water, will became transparent.

```
ndwiMask <- ndwi$updateMask(ndwi$gte(0.4))
Map$addLayer(ndwiMask, ndwiViz,'Máscara NDWI')
```

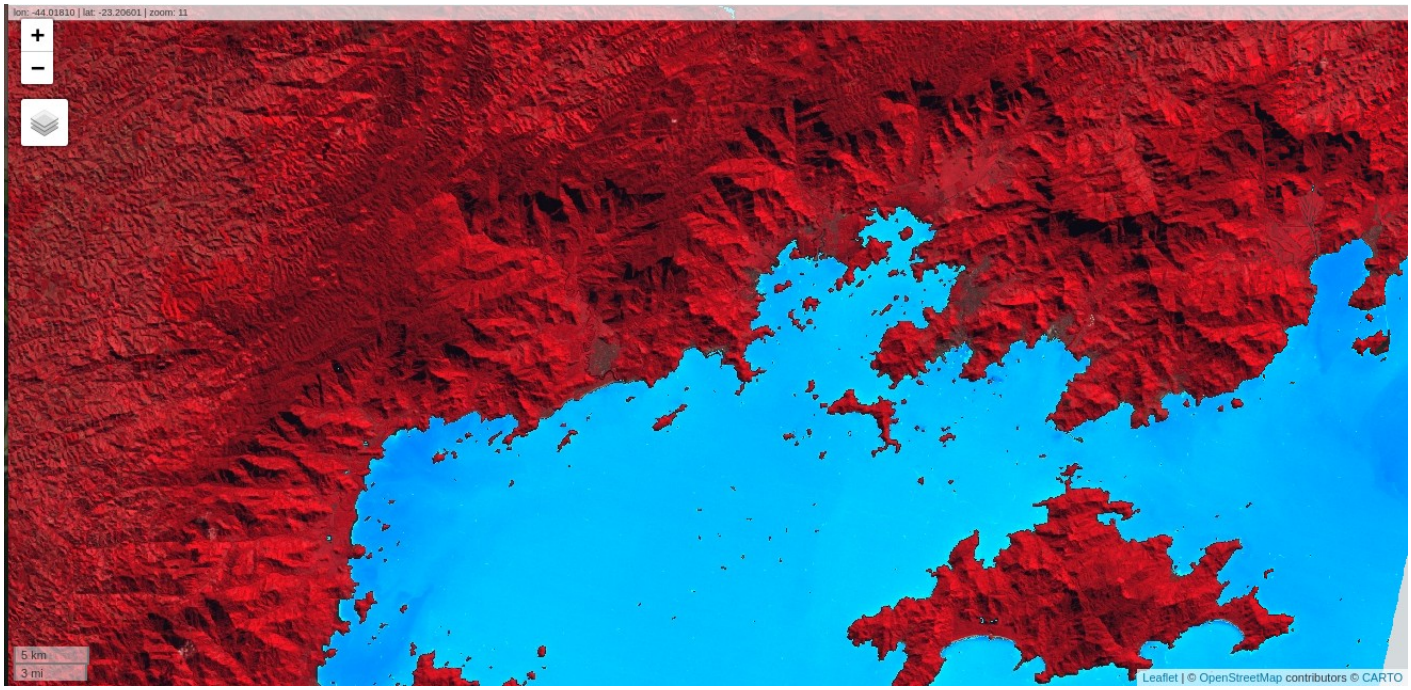


We can use the function image$visualize() to create a RGB image from any image.

```
imageRGB<-image$visualize(bands=c('B5','B4','B3'),max=0.5)
ndwiRGB<- ndwiMask$visualize(min=0.5,max=1,palette= c('#00FFFF',
'#0000FF'))
```

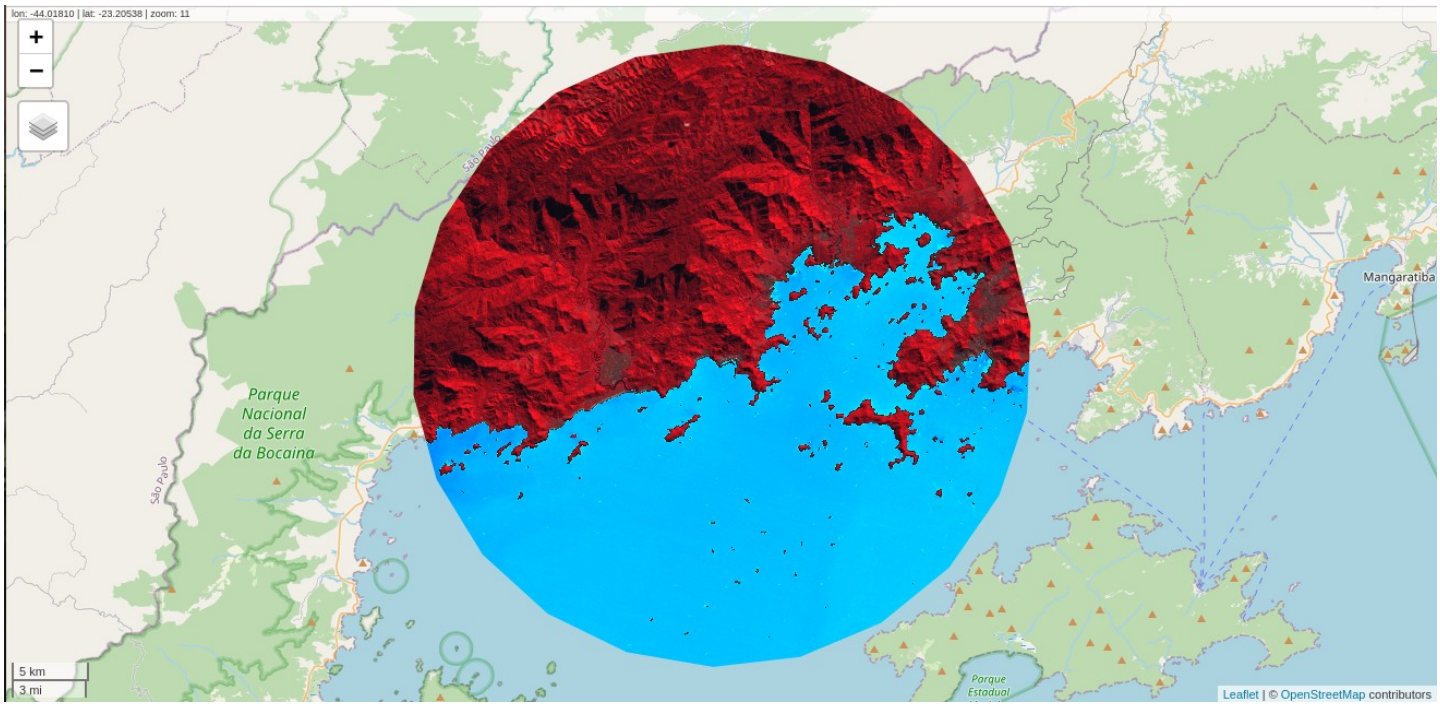
and create a mosaic of the two RGB images above using:

```
mosaic<-ee$ImageCollection(c(imageRGB, ndwiRGB))$mosaic();
Map$addLayer(mosaic ,list(), 'mosaic')
```



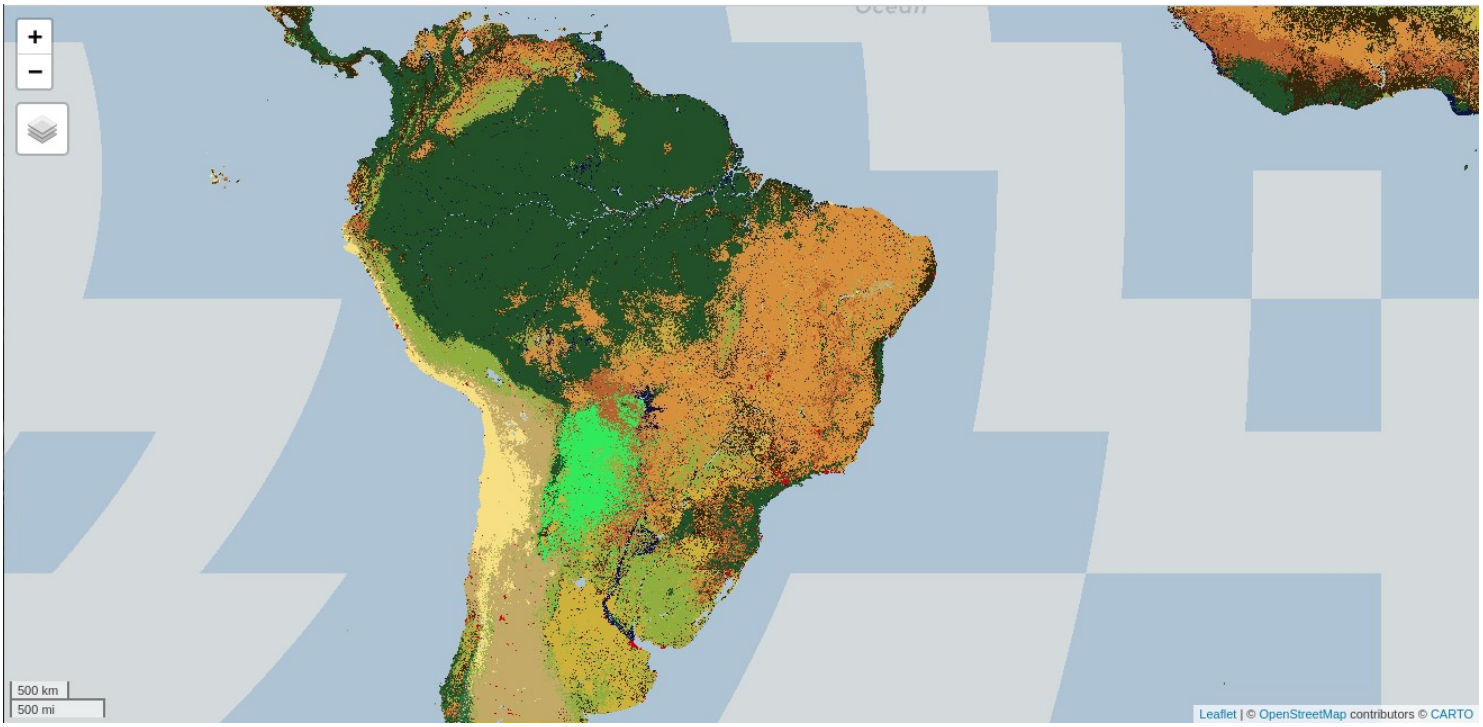
Extracting a region from the image using an geometric objet:

```
roi <- ee$Geometry$Point(c(-44.4678,-23.0058))$buffer(20000)
Map$addLayer(mosaic$clip(roi))
```



Now we are going to use a MODIS image to create a biome map:

```
biome<-ee$Image('MODIS/051/MCD12Q1/2012_01_01')
$select('Land_Cover_Type_1')
igbpPalette<-c('#aec3d4','#152106', '#225129', '#369b47',
       '#30eb5b', '#387242', '#6a2325', '#c3aa69', '#b76031',
       '#d9903d', '#91af40', '#111149', '#cdb33b', '#cc0013',
       '#33280d', '#d7cdcc', '#f7e084', '#6f6f6f')
Map$setCenter(-50.0, -15.0, 4)
Map$addLayer(biome,list(min=0, max=17, palette = igbpPalette),
             'Biomes')
```



ee$Image information and metadata

We already saw above how to get some image information and metadata using ee_print(). Now we are goin to get more information and metadata individually. First we load an image and use ee_print().

```
col<-ee$ImageCollection('COPERNICUS/S2_SR')
point <- ee$Geometry$Point(-44.366,-18.145)
start <- ee$Date("2019-07-11")
end <- ee$Date("2019-07-20")
filter<-col$filterBounds(point)$filterDate(start,end)
img <- filter$first()
ee_print(img)
```

```
──────────────────────────────────────────── Earth Engine Image ──
Image Metadata:
 - Class                : ee$Image
 - ID                   : COPERNICUS/S2_SR/20190711T131251_20190711T131248_T23KNA
 - Time start           : 2019-07-11 13:17:13
 - Number of Bands      : 23
 - Bands names          : B1 B2 B3 B4 B5 B6 B7 B8 B8A B9 B11 B12 AOT WVP SCL
TCI_R TCI_G TCI_B MSK_CLDPRB MSK_SNWPRB QA10 QA20 QA60
 - Number of Properties: 81
 - Number of Pixels*    : 77066790
 - Approximate size*    : 38.39 GB
Band Metadata (img_band = B1):
 - EPSG (SRID)          : 32723
 - proj4string          : +proj=utm +zone=23 +south +datum=WGS84 +units=m
+no_defs
 - Geotransform         : 60 0 499980 0 -60 8100040
 - Nominal scale(m)     : 60
 - Dimensions           : 1831 1830
 - Number of Pixels     : 3350730
 - Data type            : INT
 - Approximate size     : 1.67 GB
 ──────────────────────────────────────────────────────────────────────
 NOTE:(*)Properties calc. considering a constant geotransform and data type.
```

And now getting some information individually using cat.

Band names:

```
bandNames<-img$bandNames()
cat("Bands: ",paste(bandNames$getInfo(),"\n",collapse=" "))
```

```
Bands:  B1
 B2
 B3
 B4
 B5
 B6
 B7
 B8
 B8A
 B9
 B11
 B12
 AOT
 WVP
 SCL
 TCI_R
 TCI_G
 TCI_B
 MSK_CLDPRB
 MSK_SNWPRB
 QA10
 QA20
 QA60
```

band 1 projection info:

```
b1proj<- img$select('B1')$projection()
cat("B1 projection: ", paste(b1proj$getInfo(),"\n", collapse = "
"))
```

```
B1 projection:  Projection
EPSG:32723

c(60, 0, 499980, 0, -60, 8100040)
```

Band 1 Scale:

```
b1scale<-img$select('B1')$projection()$nominalScale()
cat("B1 Scale: ", paste(b1scale$getInfo(),"\n", collapse = " "))
```

```
B1 Scale:  60
```

Band 2 Scale:

```
b2scale<-img$select('B2')$projection()$nominalScale()
cat("B2 Scale: ", paste(b2scale$getInfo(),"\n", collapse = " "))
```
```
B2 Scale:  10
```

Banda 5 Scale:

```
b5scale<-img$select('B5')$projection()$nominalScale()
cat("B5 Scale: ", paste(b5scale$getInfo(),"\n", collapse = " "))
```
```
B5 Scale:  20
```

Showng all Metadata names:

```
metadata<-img$propertyNames()
cat("Metadata: ", paste(metadata$getInfo(),"\n",collapse = " "))
```
```
Metadata:
 DATATAKE_IDENTIFIER
 AOT_RETRIEVAL_ACCURACY
 SPACECRAFT_NAME
 SATURATED_DEFECTIVE_PIXEL_PERCENTAGE
 system:id
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B8A
 CLOUD_SHADOW_PERCENTAGE
 MEAN_SOLAR_AZIMUTH_ANGLE
 system:footprint
 VEGETATION_PERCENTAGE
 SOLAR_IRRADIANCE_B12
 system:version
 SOLAR_IRRADIANCE_B10
 SENSOR_QUALITY
 SOLAR_IRRADIANCE_B11
 GENERATION_TIME
 SOLAR_IRRADIANCE_B8A
 FORMAT_CORRECTNESS
 CLOUD_COVERAGE_ASSESSMENT
 THIN_CIRRUS_PERCENTAGE
 system:time_end
 WATER_VAPOUR_RETRIEVAL_ACCURACY
 system:time_start
 DATASTRIP_ID
 PROCESSING_BASELINE
 SENSING_ORBIT_NUMBER
 NODATA_PIXEL_PERCENTAGE
 SENSING_ORBIT_DIRECTION
 GENERAL_QUALITY
 GRANULE_ID
 REFLECTANCE_CONVERSION_CORRECTION
 MEDIUM_PROBA_CLOUDS_PERCENTAGE
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B8
 DATATAKE_TYPE
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B9
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B6
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B7
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B4
 MEAN_INCIDENCE_ZENITH_ANGLE_B1
 NOT_VEGETATED_PERCENTAGE
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B5
 RADIOMETRIC_QUALITY
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B2
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B3
 MEAN_INCIDENCE_ZENITH_ANGLE_B5
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B1
 MEAN_INCIDENCE_ZENITH_ANGLE_B4
 MEAN_INCIDENCE_ZENITH_ANGLE_B3
 MEAN_INCIDENCE_ZENITH_ANGLE_B2
```

```
 MEAN_INCIDENCE_ZENITH_ANGLE_B9
 MEAN_INCIDENCE_ZENITH_ANGLE_B8
 MEAN_INCIDENCE_ZENITH_ANGLE_B7
 DARK_FEATURES_PERCENTAGE
 HIGH_PROBA_CLOUDS_PERCENTAGE
 MEAN_INCIDENCE_ZENITH_ANGLE_B6
 UNCLASSIFIED_PERCENTAGE
 MEAN_SOLAR_ZENITH_ANGLE
 MEAN_INCIDENCE_ZENITH_ANGLE_B8A
 RADIATIVE_TRANSFER_ACCURACY
 MGRS_TILE
 CLOUDY_PIXEL_PERCENTAGE
 PRODUCT_ID
 MEAN_INCIDENCE_ZENITH_ANGLE_B10
 SOLAR_IRRADIANCE_B9
 SNOW_ICE_PERCENTAGE
 DEGRADED_MSI_DATA_PERCENTAGE
 MEAN_INCIDENCE_ZENITH_ANGLE_B11
 MEAN_INCIDENCE_ZENITH_ANGLE_B12
 SOLAR_IRRADIANCE_B6
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B10
 SOLAR_IRRADIANCE_B5
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B11
 SOLAR_IRRADIANCE_B8
 MEAN_INCIDENCE_AZIMUTH_ANGLE_B12
 SOLAR_IRRADIANCE_B7
 SOLAR_IRRADIANCE_B2
 SOLAR_IRRADIANCE_B1
 SOLAR_IRRADIANCE_B4
 GEOMETRIC_QUALITY
 SOLAR_IRRADIANCE_B3
 system:asset_size
 WATER_PERCENTAGE
 system:index
 system:bands
 system:band_names
```

Getting an specific matadata field value that was listed above:

```
cloudCover <- img$get('CLOUD_SHADOW_PERCENTAGE')
cat("Cloud Cover %:",paste(cloudCover$getInfo(),"\n",collapse="
"))
```

```
Cloud Cover %: 0.000116
```

ee$Image arithmetic operations

Google Earth Engine has two ways of executing mathematical operations: a weird one to R users that uses inline function for each operation that may be suitable for simple operation and another one using the function ee$Image$expression() which is more appropriated.

Executing the NDVI calculation using the first method (inline functions):

```
library(rgee)
ee_Initialize()
```

```
col<-ee$ImageCollection('LANDSAT/LC08/C01/T1_TOA')
point <- ee$Geometry$Point(-44.4678,-23.0058)
start <- ee$Date("2019-06-11")
end <- ee$Date("2019-08-20")
filter<-col$filterBounds(point)$filterDate(start,end)
image <- filter$first()
ndvi<-(image$select('B4')$subtract(image$select('B3')))
$divide(image$select('B4')$add(image$select('B3')))
```

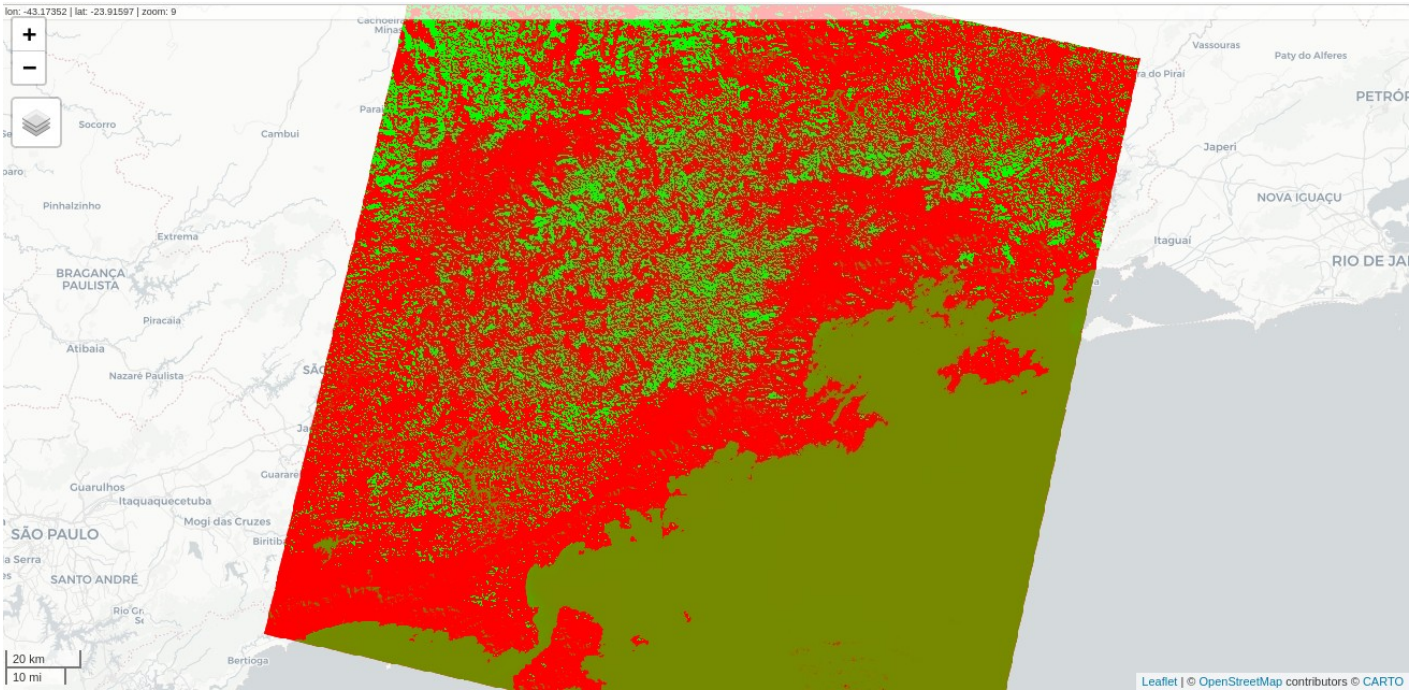
And now calculating the EVI using the second method using expression():

```
evi<-image$expression(
              '2.5 * ((NIR - RED) / (NIR + 6 * RED - 7.5 * BLUE))',
              list('NIR'=image$select('B5'),
              'RED'=image$select('B4'),
              'BLUE'=image$select('B2')))
```

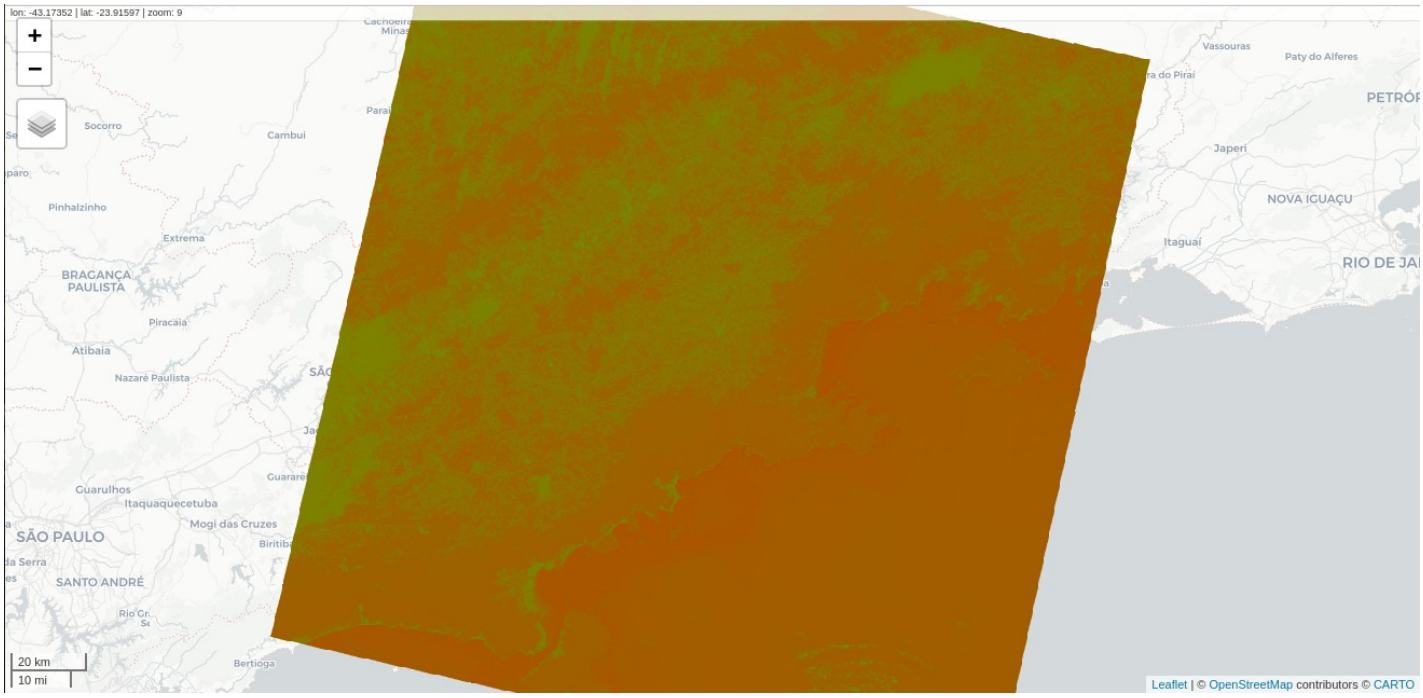
Plotting both images:

```
Map$centerObject(image,9)
Map$addLayer(ndvi,list(min=-1, max=1, palette=c('#FF0000','#00FF00')),
'NDVI') + Map$addLayer(evi,list(min=-1,max=1,palette=c('#FF0000',
'#00FF00')), 'EVI')
```

EVI



NDVI

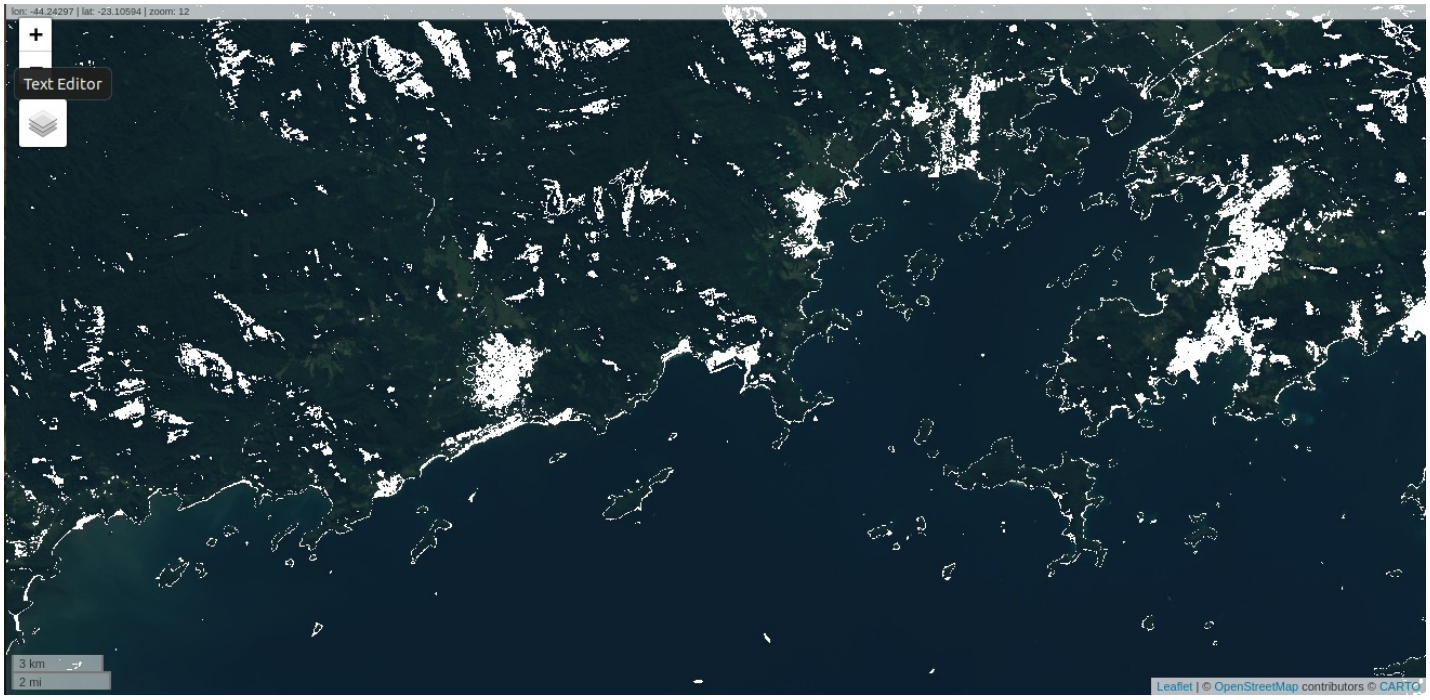


ee#Image relational, boolean and conditional operations

These operation are used a lot to create masks, filters and to generate heat maps. We will cover here on how to use these operations.

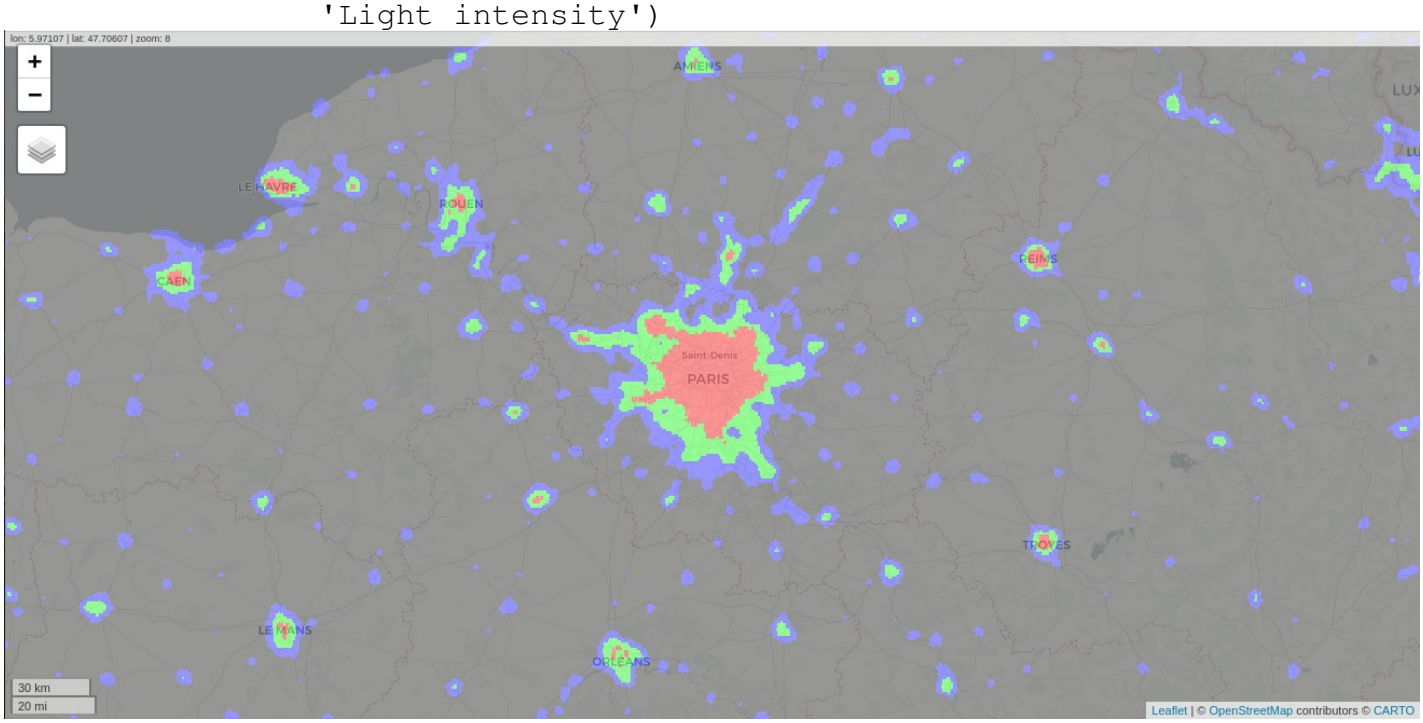
Creating a binary mask using two images (NDWI and NDVI) and selecting the ares not covered by water nor vegetation, the resulting image will be showing areas with rock, soil, beaches and urban zones.

```
library(rgee)
ee_Initialize()
col<-ee$ImageCollection('LANDSAT/LC08/C01/T1_TOA')
ponit <- ee$Geometry$Point(-44.4678,-23.0058)
start <- ee$Date("2019-06-11")
end <- ee$Date("2019-08-20")
filter<-col$filterBounds(point)$filterDate(start,end)
image <- filter$first()
ndvi<-image$normalizedDifference(c('B5', 'B4'))
ndwi<- image$normalizedDifference(c('B3', 'B5'))
exposed<-ndvi$lt(0.5)$And(ndwi$lt(0.2))
Map$setCenter(-44.4678,-23.0058, 12)
imgViz<-list(min=0,max=0.5,bands=c("B4","B3","B2"),gamma=c(0.95,
1.1,1))
Map$addLayer(image,imgViz, 'Landsat')+
Map$addLayer(exposed$updateMask(exposed),list(),'Exposed areas')
```



Paris is known as the 'city of lights', let's check it out using relational and conditional operations.

```
library(rgee)
ee_Initialize()
nl2012<-ee$Image('NOAA/DMSP-OLS/NIGHTTIME_LIGHTS/F182012')
pal<-c('#000000','#0000FF','#00FF00','#FF0000')
Map$setCenter(2.373,48.8683,8)
luz<-nl2012$expression("(b('stable_lights') > 62) ? 3  :
(b('stable_lights') > 55) ? 2 : (b('stable_lights') > 30) ? 1 : 0"
)
Map$addLayer(luz,list(min=0,max=3,palette=pal,opacity=0.4),
             'Light intensity')
```



Spectral transformation with ee$Image

There are several spectral transformation function In Google Earth Engine, including instances for images such as normalizedDifference(), unmix(), rgbToHsv() and hsvToRgb(). The last two can be used to execute a pensharpening process.

Pansharpening is a process used to join panchromatic images of high resolution with multispectral imates of lower resolution in order to create a colored image
Low-res color image + Hi-res gray band = Higher-res color image

This is done by decomposing a RGB image in HSV and replacing the V (value) band by the high resolution panchromatic and converting it back to a RGB image.

```
library(rgee)
ee_Initialize()
col<-ee$ImageCollection('LANDSAT/LC08/C01/T1_TOA')
point <- ee$Geometry$Point(-44.4678,-23.0058)
start <- ee$Date("2019-06-11")
end <- ee$Date("2019-08-20")
filter<-col$filterBounds(point)$filterDate(start,end)
image <- filter$first()
hsv<-image$select(c('B4', 'B3', 'B2'))$rgbToHsv()
sharpened<-ee$Image$cat(c(
   hsv$select('hue'), hsv$select('saturation'), image$select('B8')
   ))$hsvToRgb()
Map$setCenter(-44.4678,-23.0058, 13)
Map$addLayer(image,list(
    bands=c('B4', 'B3', 'B2'), min=0, max=0.25, gamma=c(1.3, 1.3,
1.3)),
    'rgb')+
Map$addLayer(hsv,list(
    bands=c('hue', 'saturation', 'value'), min=0, max=1,
gamma=c(1.3, 1.3, 1.3)),
    'hsv')+
Map$addLayer(sharpened,list(
             min=0, max=0.25, gamma=c(1.3, 1.3, 1.3)),
             'pensharpened')
```

The resulting images will be:

RGB

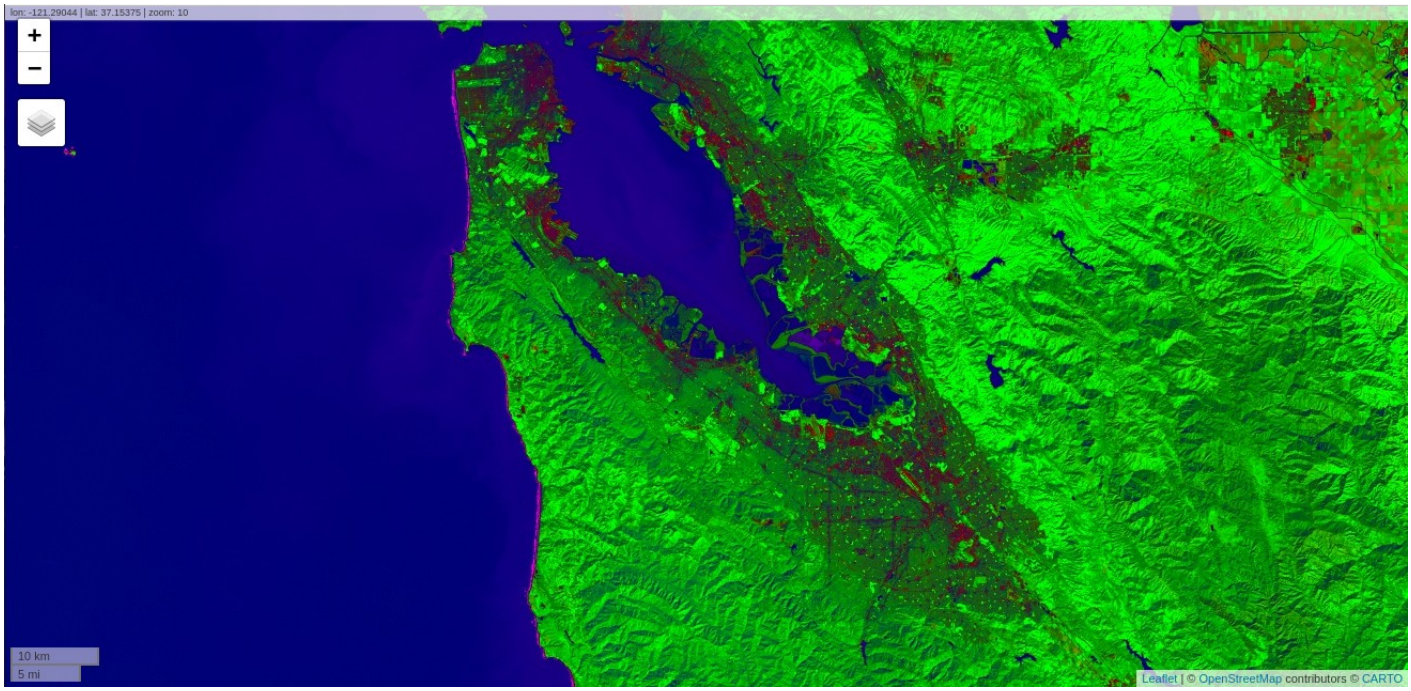
Pensharpened

Previously we saw how to use the function normilizedDifference(). Now we will use the function unmix() to extract and group regions with similar spectral responses

```
library(rgee)
ee_Initialize()
bands <- c("B1", "B2", "B3", "B4", "B5", "B6", "B7")
image <- ee$Image("LANDSAT/LT05/C01/T1/LT05_044034_20080214")
$select(bands)
# Defining the final spectral values for each defined class in the
# 7 bands
```

```
urban <- c(88, 42, 48, 38, 86, 115, 59)
veget <- c(50, 21, 20, 35, 50, 110, 23)
water <- c(51, 20, 14, 9, 7, 116, 4)
# Unmix a imagem.
fractions <- image$unmix(list(urban, veget, water))
Map$setCenter(lon=-122.1899,lat=37.5010)# San Francisco bay
Map$setZoom(zoom = 10)
Map$addLayer(
  eeObject = image,
  visParams = list(min = 0, max = 128, bands = c("B4", "B3",
"B2")),
  name = "Image"
) +
Map$addLayer(
  eeObject = fractions,
  visParams = list(min = 0, max = 2),
  name = "Unmixed"
)
```

The resulting image is (observ the separation between water, urban zoness and vegetated areas):

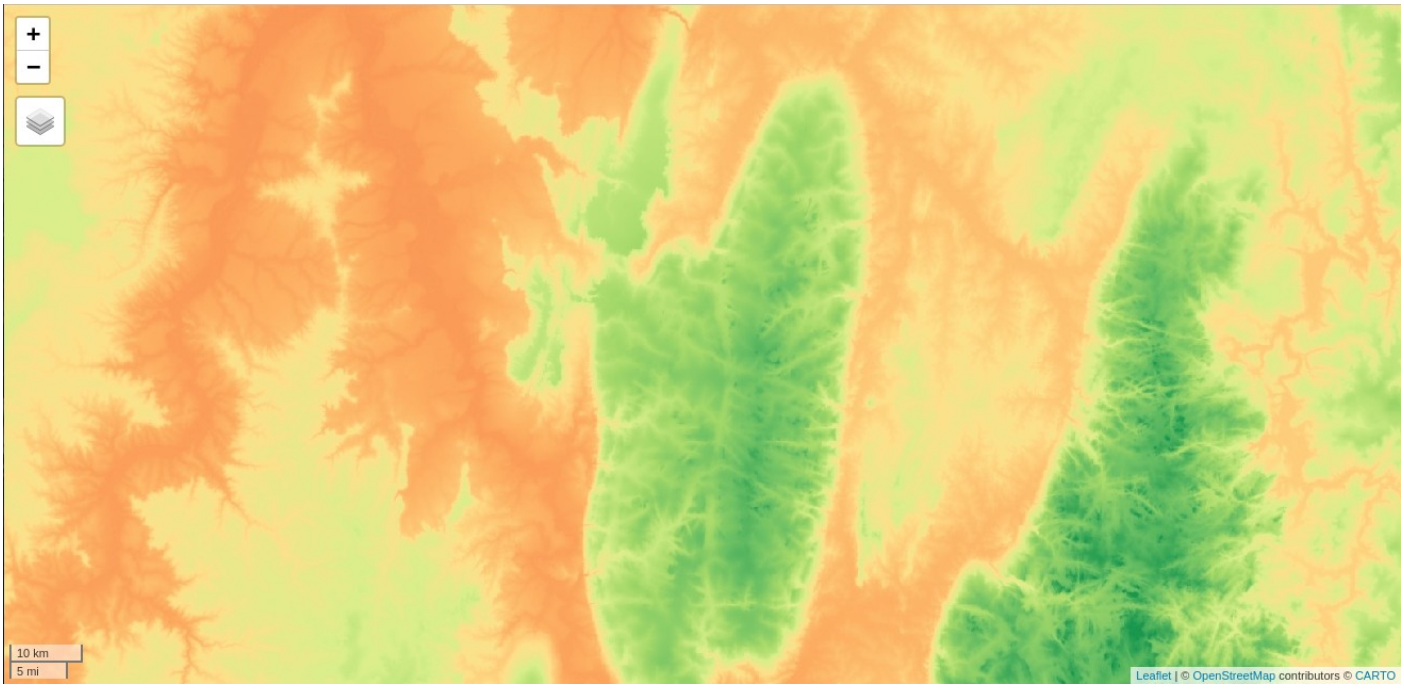


ee$image Gradient

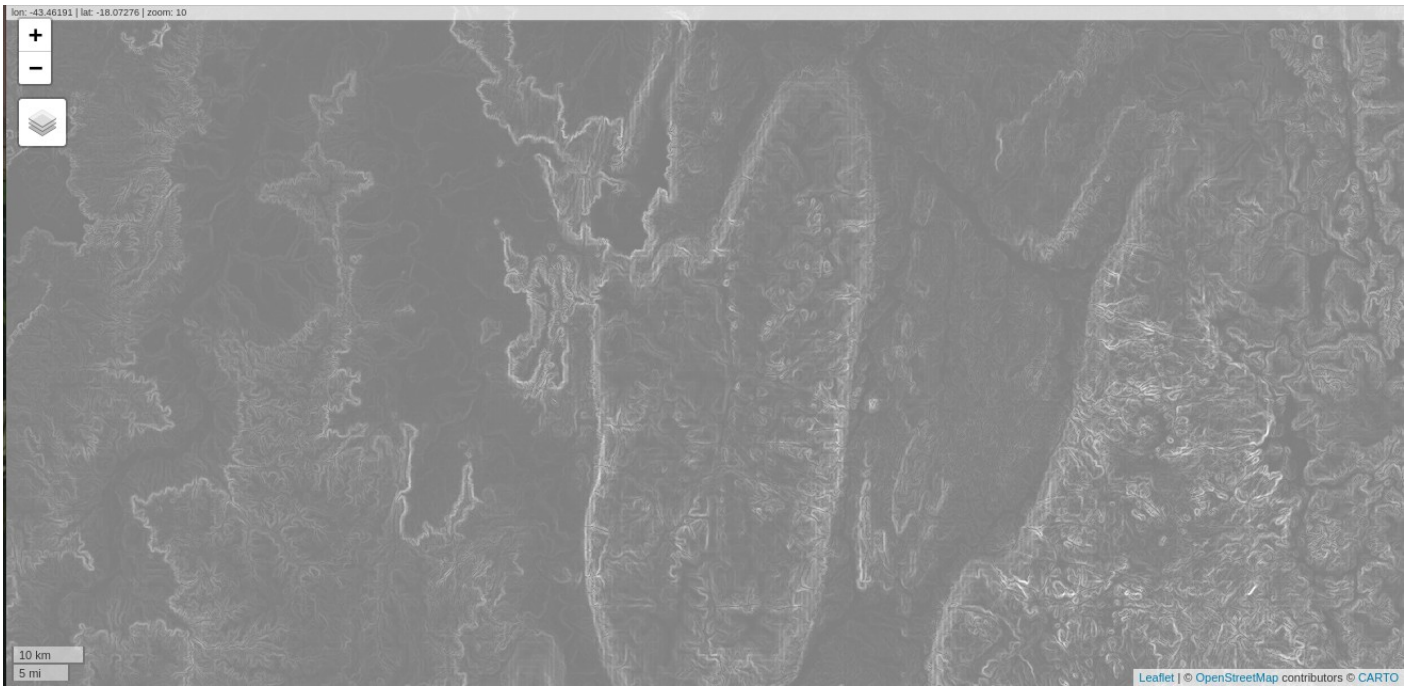
Extracting gradient information from an image can be done and we will use a DEM ALOS AW3D30 image to demonstrate it extracting the o gradient and aspect from it.

```
library(rgee)
ee_Initialize()
data <- ee$Image("JAXA/ALOS/AW3D30_V1_1")
elevation <- data$select("AVE")
xyGrad <- elevation$gradient()
gradient <- xyGrad$select("x")$pow(2)$add(xyGrad$select("y")
$pow(2))$sqrt()
direction <- xyGrad$select("y")$atan2(xyGrad$select("x"))
Map$setCenter(-44.366,-17.69, zoom = 10)
Map$addLayer(eeObject=direction,visParams=list(min=-2,max=2),
 name="Aspect")+
 Map$addLayer(eeObject=gradient,visParams=list(min=-1,max=1),
 name="gradient")+
 Map$addLayer(eeObject=elevation,visParams=list(palette = c(
 "#d73027", "#f46d43", "#fdae61", "#fee08b","#d9ef8b","#a6d96a",
 "#66bd63", "#1a9850"),min=200,max=1400),name="Elevation")
```

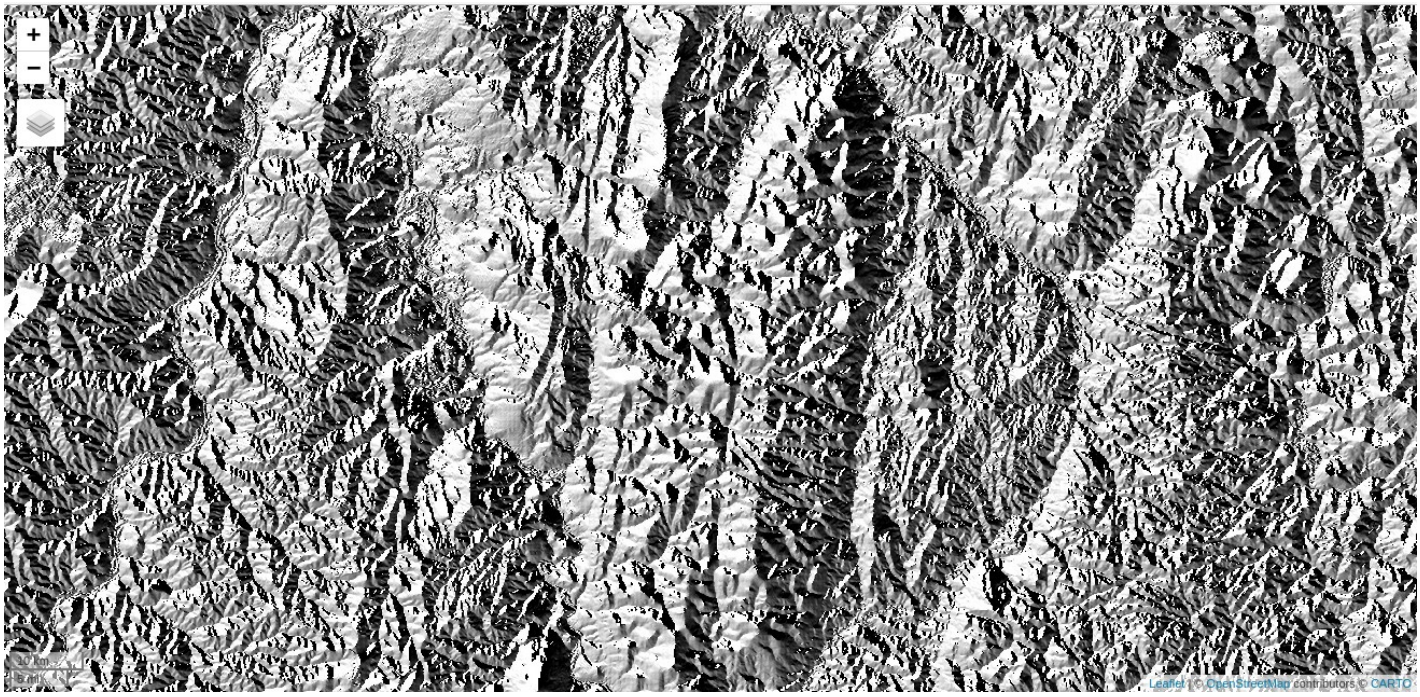
Elevation



gradient

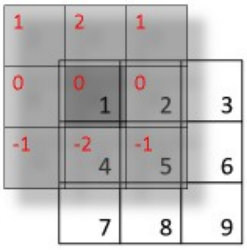


Aspect

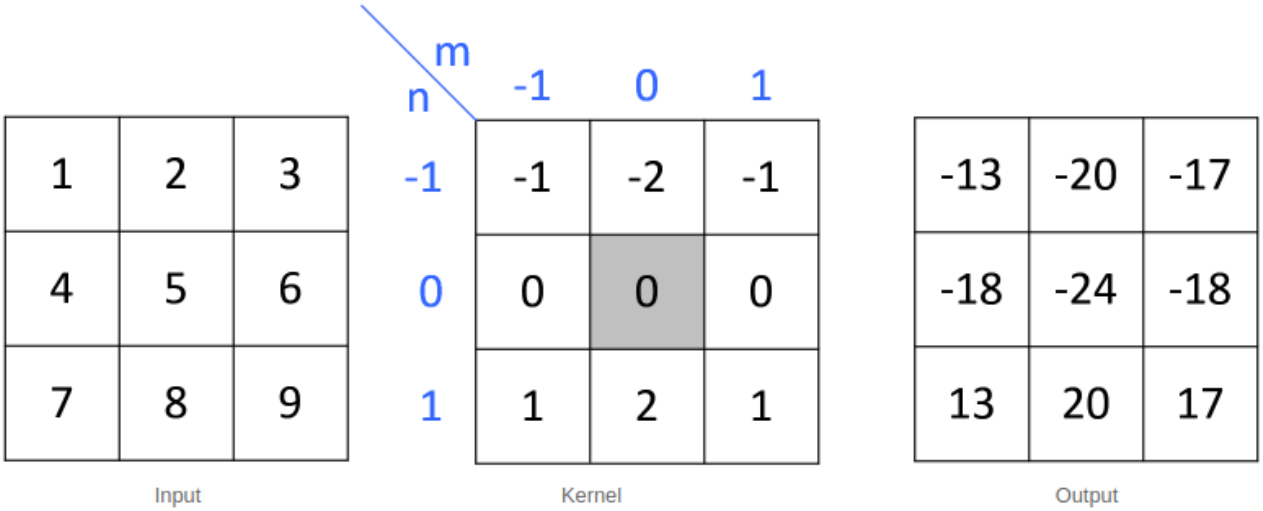


ee$Image convolution and edge detection

Na matemática convolução é uma operação envolvendo duas funções (f e g) que produz uma terceira função que expressa a forma que uma função é modificada pela outra. O termo se refere a ambos, a função resultante e processo de cálculo desta função. Em processamento digital de imagens a filtragem convolucional tem uma regra importante em algorítimos de detecção de bordas e processos relacionados. Ela é o resultado de um kernel (filtro modificador no formato de uma matriz NxM) sobre os valores dos pixels da imagem.



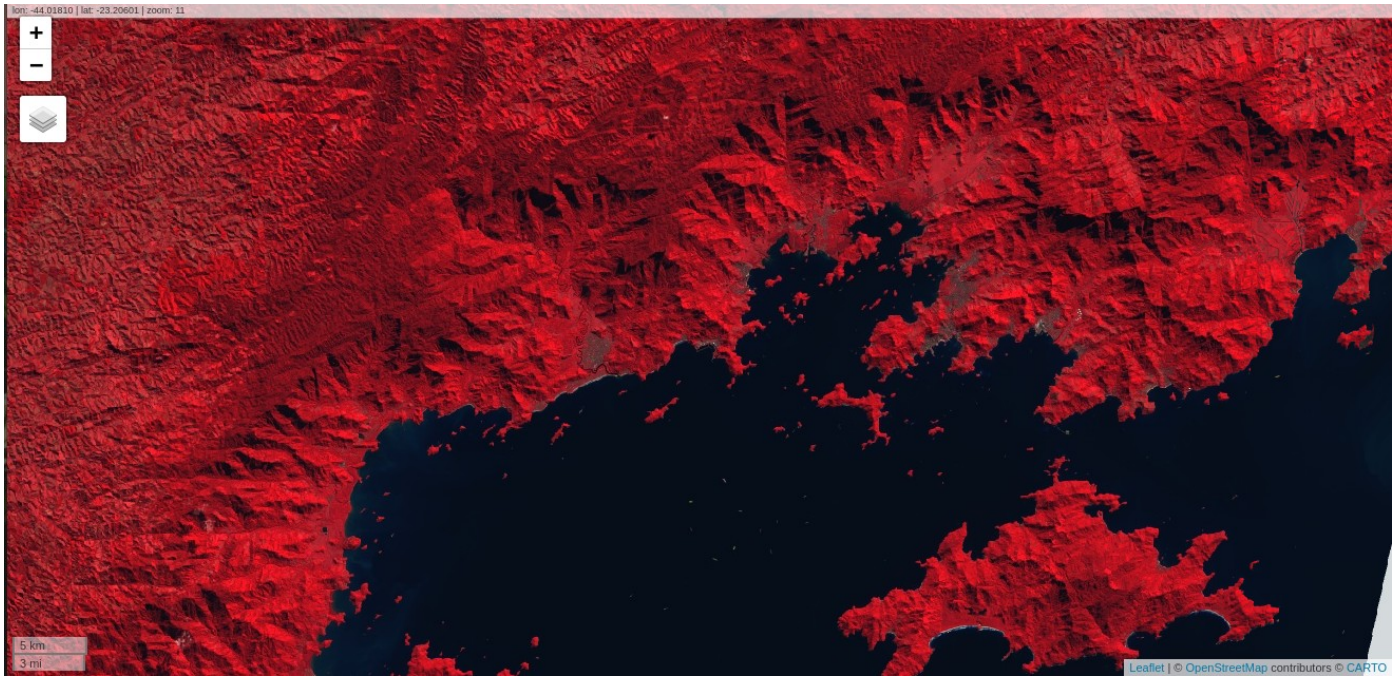
$$
\begin{aligned}
y[0,0] &= \sum_j \sum_i x[i,j] \cdot h[0-i, 0-j] \\
&= \quad x[-1,-1] \cdot h[1,1] + x[0,-1] \cdot h[0,1] + x[1,-1] \cdot h[-1,1] \\
&\quad + x[-1,0] \cdot h[1,0] \quad + x[0,0] \cdot h[0,0] \quad + x[1,0] \cdot h[-1,0] \\
&\quad + x[-1,1] \cdot h[1,-1] + x[0,1] \cdot h[0,-1] + x[1,1] \cdot h[-1,-1] \\
&= \quad 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 \\
&\quad + 0 \cdot 0 + 1 \cdot 0 + 2 \cdot 0 \\
&\quad + 0 \cdot (-1) + 4 \cdot (-2) + 5 \cdot (-1) \\
&= -13
\end{aligned}
$$



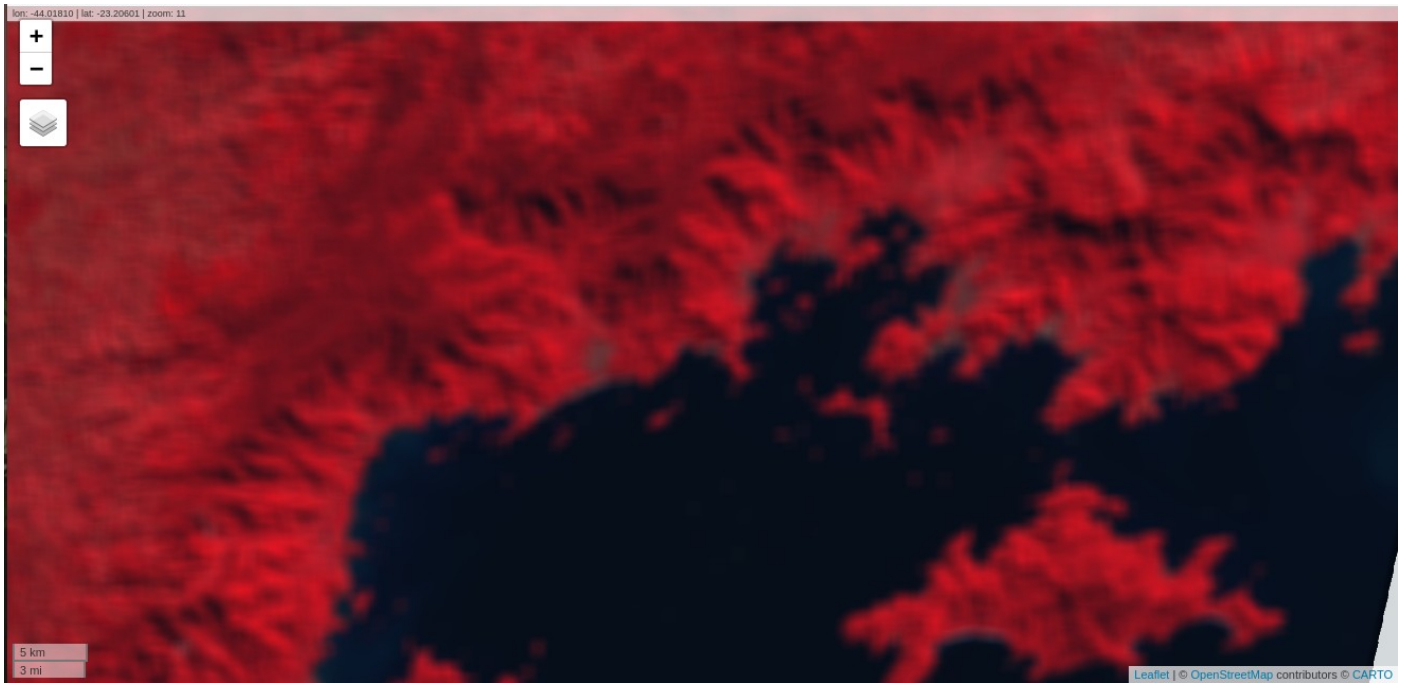
Different methods use different interaction equations than the one illustrated above. We will use convolution and edge detection in the next examples.

```
library(rgee)
ee_Initialize()
col<-ee$ImageCollection('LANDSAT/LC08/C01/T1_TOA')
point <- ee$Geometry$Point(-44.4678,-23.0058)
start <- ee$Date("2019-06-11")
end <- ee$Date("2019-08-20")
filter<-col$filterBounds(point)$filterDate(start,end)
image <- filter$first()

boxcar <- ee$Kernel$square(7, "pixels", T)
smoothed <- image$convolve(boxcar)

laplacian <- ee$Kernel$laplacian8(1, F)
sharpened <- image$convolve(laplacian)

Map$setCenter(-44.4678,-23.0058)
Map$setZoom(zoom = 11)
Map$addLayer(
  eeObject = image,
  visParams = list(bands = c("B5", "B4", "B3"), max = 0.5),
  name = "Image"
) +
Map$addLayer(
  eeObject = smoothed,
  visParams = list(bands = c("B5", "B4", "B3"), max = 0.5),
  name = "Smoothed"
) +
Map$addLayer(
  eeObject = sharpened,
  visParams = list(bands = c("B5", "B4", "B3"), max = 0.5),
  name = "Sharpened"
)
```

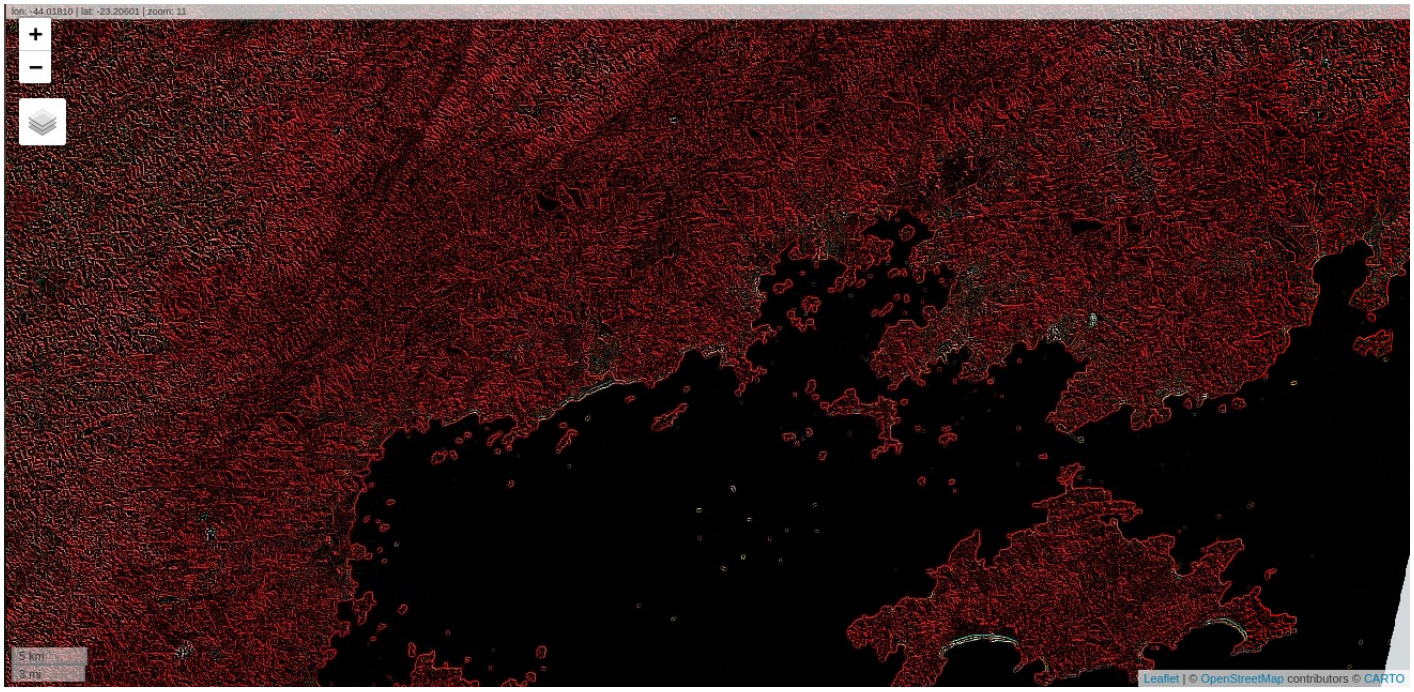
Original (Bands 5, 4 e 3)



Smoothed



Sharpened



We can create our own kernel with weights and shapes arbitrarily defined, to do it we use ee.Kernel.fixed(). For example, the code below creates a 9x9 kernel 9x9 with values of 1 and a zero in the middle:

```
list<-list(1, 1, 1, 1, 1, 1, 1, 1, 1)
listCenter<- list(1, 1, 1, 1, 0, 1, 1, 1, 1)
```

```
lists<-list(list, list, list, list, listCenter, list, list, list,
list)
kernel<-ee$Kernel$fixed(9, 9, lists, -4, -4, FALSE);
print(kernel)
```


```
ee.Kernel({
  "type": "Invocation",
  "arguments": {
    "width": 9.0,
    "height": 9.0,
    "weights": [
      [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
      ],
      [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
      ],
      [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
      ],
      [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
      ],
      [
        1.0,
        1.0,
        1.0,
        1.0,
        0.0,
        1.0,
        1.0,
        1.0,
        1.0
      ],
      [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
```

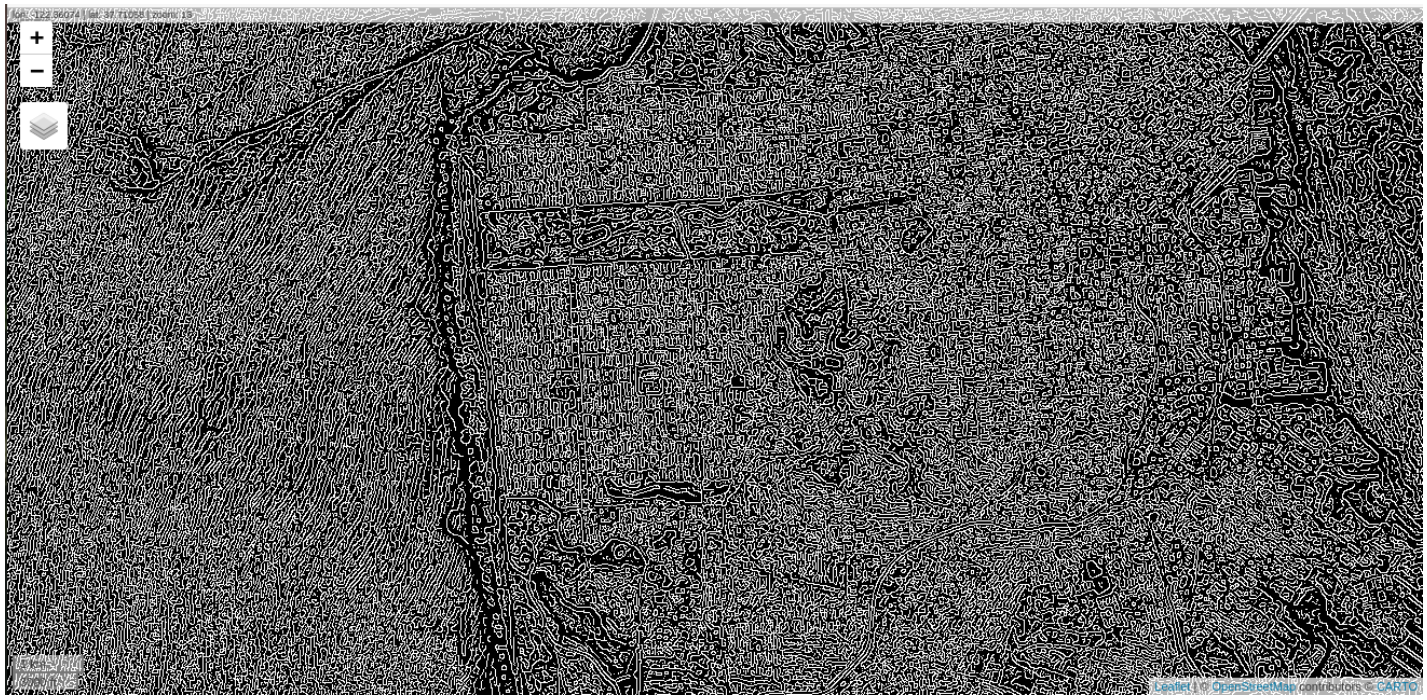
```
        1.0,
        1.0,
        1.0
      ],
      [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
      ],
      [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
      ],
      [
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0
      ]
    ],
    "x": -4.0,
    "y": -4.0,
    "normalize": false
  },
  "functionName": "Kernel.fixed"
})
```

Let's learn how to use some edge detection algorithms.

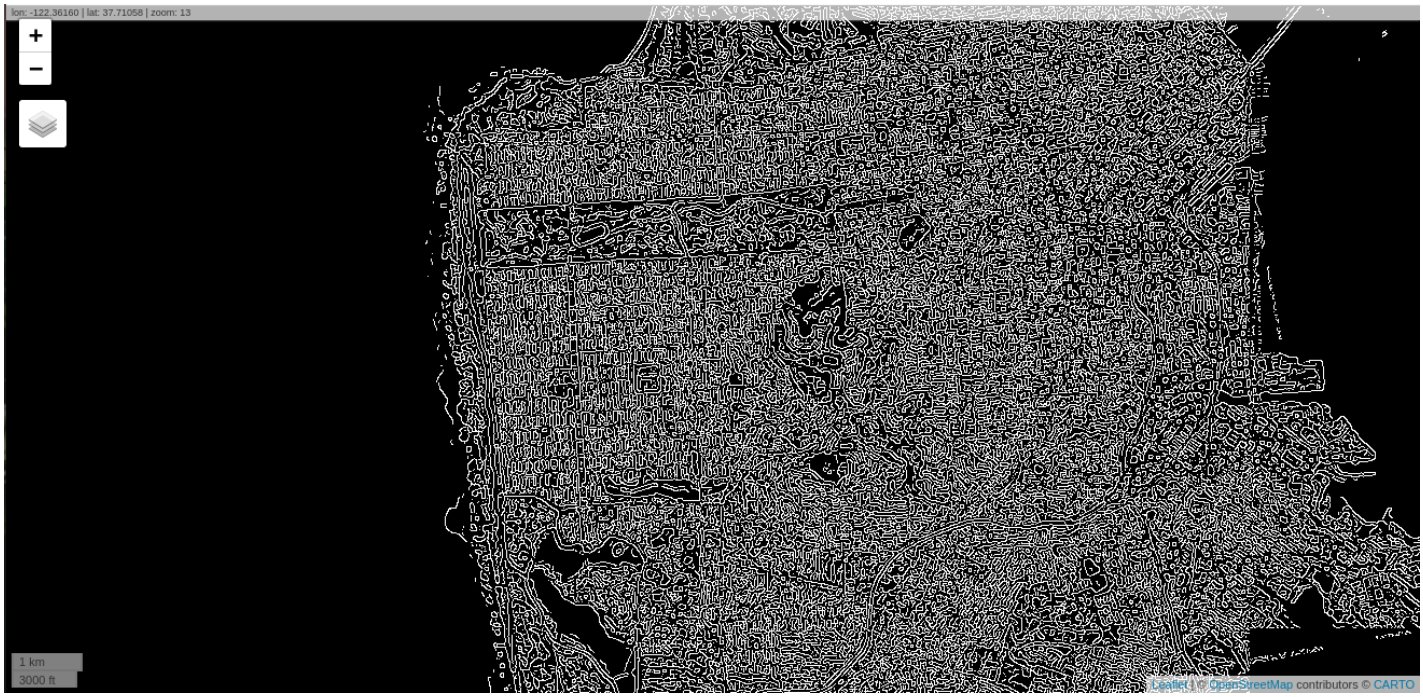
```
library(rgee)
ee_Initialize()
image    <-    ee$Image("LANDSAT/LC08/C01/T1/LC08_044034_20140318")
$select("B8")
canny  <-  ee$Algorithms$CannyEdgeDetector(image=image,  threshold  =
1, sigma = 1)
hough <- ee$Algorithms$HoughTransform(canny, 256, 600, 100)
fat <- ee$Kernel$gaussian(
  radius = 3,
  sigma = 3,
  units = "pixels",
  normalize = T,
  magnitude = -1
)
skinny <- ee$Kernel$gaussian(
  radius = 3,
  sigma = 1,
  units = "pixels",
  normalize = T
)
dog <- fat$add(skinny)
```

```
zeroXings <- image$convolve(dog)$zeroCrossing()

Map$setCenter(-122.054, 37.7295)
Map$setZoom(zoom = 10)
Map$addLayer(
  eeObject = canny,
  visParams = list(),
  name = "canny"
) +
  Map$addLayer(
    eeObject = hough,
    visParams = list(),
    name = "hough"
  ) +
  Map$addLayer(
    eeObject = image,
    visParams = list(gamma=1.5),
    name = "Landsat B8"
  ) +
  Map$addLayer(
    eeObject = zeroXings$updateMask(zeroXings),
    visParams = list(palette = "FF0000"),
    name = "zero crossings"
  )
```

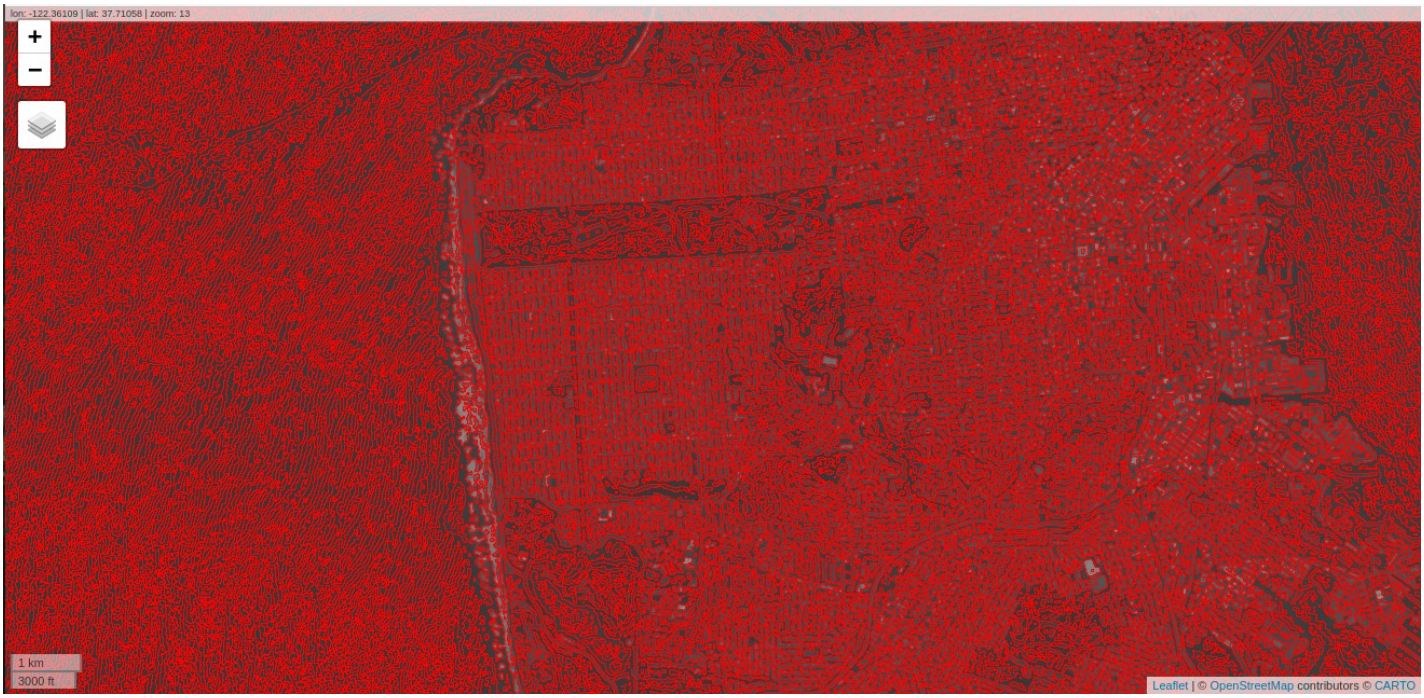
Canny



Hough



Zero Crossing



ee$ImageCollection

ee$ImageCollection constructors

A ee$ImageCollection is a stack or a sequence of images. This object can be loaded entering an ID of an asset as the parameter of an ee$ImageCollection object constructor.

```
library(rgee)
ee_Initialize()
col<-ee$ImageCollection('COPERNICUS/S2_SR')
```

This collection has a reference to all Sentinel2 in the public catalog and they are a lot of images, generally we use filters associated to image collections to reduce the number of images and get the ones we need.

```
point <- ee$Geometry$Point(-44.366,-18.145)
start <- ee$Date("2019-07-11")
end <- ee$Date("2019-07-20")
col.filt<-col$filterBounds(point)$filterDate(start,end)
```

Google Earth Engine also has functions to create an image collection object. The constructor ee$ImageCollection() and the method ee$ImageCollection$fromImages can create a new image collection object.

```
image1<-ee$Image(1)
image2<-ee$Image(2)
col1<-ee$ImageCollection(list(image1,image2))
ee_print(col1)
```

```
── Earth Engine ImageCollection ──
ImageCollection Metadata:
 - Class                    : ee$ImageCollection
 - Number of Images         : 2
 - Number of Properties     : 0
 - Number of Pixels*        : 129600
 - Approximate size*        : 202.50 KB
Image Metadata (img_index = 0):
 - ID                       : no_id
 - Number of Bands          : 1
 - Bands names              : constant
 - Number of Properties     : 1
 - Number of Pixels*        : 64800
 - Approximate size*        : 101.25 KB
Band Metadata (img_band = 'constant'):
 - EPSG (SRID)              : 4326
 - proj4string              : +proj=longlat +datum=WGS84 +no_defs
 - Geotransform             : 1 0 0 0 1 0
 - Nominal scale (meters)   : 111319.5
 - Dimensions               : 360 180
 - Number of Pixels         : 64800
 - Data type                : INT
 - Approximate size         : 101.25 KB
 NOTE: (*) Properties calculated considering a constant  geotransform and data
type.
```

Other form to create image collections is:

```
col2<-ee$ImageCollection$fromImages(list(ee$Image(3),
          ee$Image(4)))
ee_print(col2)
```

```
── Earth Engine ImageCollection ──
ImageCollection Metadata:
 - Class                       : ee$ImageCollection
 - Number of Images            : 2
 - Number of Properties        : 0
 - Number of Pixels*           : 129600
 - Approximate size*           : 202.50 KB
Image Metadata (img_index = 0):
 - ID                          : no_id
 - Number of Bands             : 1
 - Bands names                 : constant
 - Number of Properties        : 1
 - Number of Pixels*           : 64800
 - Approximate size*           : 101.25 KB
Band Metadata (img_band = 'constant'):
 - EPSG (SRID)                 : 4326
 - proj4string                 : +proj=longlat +datum=WGS84 +no_defs
 - Geotransform                : 1 0 0 0 1 0
 - Nominal scale (meters)      : 111319.5
 - Dimensions                  : 360 180
 - Number of Pixels            : 64800
 - Data type                   : INT
 - Approximate size            : 101.25 KB
 ──────────────────────────────────────────────────────────────────────
 NOTE: (*) Properties calculated considering a constant  geotransform and data
type.
```

We can join two image collections using $merge.

```
col3<-col2$merge(col1)
ee_print(col3)
```

```
── Earth Engine ImageCollection ──
ImageCollection Metadata:
 - Class                       : ee$ImageCollection
 - Number of Images            : 4
 - Number of Properties        : 0
 - Number of Pixels*           : 259200
 - Approximate size*           : 405.00 KB
Image Metadata (img_index = 0):
 - ID                          : no_id
 - Number of Bands             : 1
 - Bands names                 : constant
 - Number of Properties        : 1
 - Number of Pixels*           : 64800
 - Approximate size*           : 101.25 KB
Band Metadata (img_band = 'constant'):
 - EPSG (SRID)                 : 4326
 - proj4string                 : +proj=longlat +datum=WGS84 +no_defs
 - Geotransform                : 1 0 0 0 1 0
 - Nominal scale (meters)      : 111319.5
 - Dimensions                  : 360 180
 - Number of Pixels            : 64800
 - Data type                   : INT
 - Approximate size            : 101.25 KB
 ──────────────────────────────────────────────────────────────────────
 NOTE: (*) Properties calculated considering a constant  geotransform and data
type.
```

And we can create an image collection using existing images from geotiffs in the cloud storage.

```
uri1<-'gs://gcp-public-data-landsat/LT05/01/001/001/
LT05_L1GS_001001_19910509_20180220_01_T2/
LT05_L1GS_001001_19910509_20180220_01_T2_B5.TIF'
uri2<-'gs://gcp-public-data-landsat/LT05/01/001/001/
LT05_L1GS_001001_19910509_20180220_01_T2/
LT05_L1GS_001001_19910509_20180220_01_T2_B4.TIF'
uri3<-'gs://gcp-public-data-landsat/LT05/01/001/001/
LT05_L1GS_001001_19910509_20180220_01_T2/
LT05_L1GS_001001_19910509_20180220_01_T2_B3.TIF'
uri4<-'gs://gcp-public-data-landsat/LT05/01/001/001/
LT05_L1GS_001001_19910509_20180220_01_T2/
LT05_L1GS_001001_19910509_20180220_01_T2_B2.TIF'
image1<-ee$Image$loadGeoTIFF(uri1)
image2<-ee$Image$loadGeoTIFF(uri2)
image3<-ee$Image$loadGeoTIFF(uri3)
image4<-ee$Image$loadGeoTIFF(uri4)
col<-ee$ImageCollection$fromImages(list(image1,image2,image3,image4))
rgb<-col$toBands()$rename(c('B5', 'B4', 'B3', 'B2'))
Map$centerObject(rgb)
Map$addLayer(rgb, list(bands=c('B5', 'B3', 'B2'), min=0, max= 1000), 'rgb')
```

Filtering ee$ImageCollection

Filtering is a very important topic when working with Image Colletions. With it we can limit by area, date and type the ones we are going to use. The next example we fiter an image collection by date, area and an image quality metadata value.

```
col<-ee$ImageCollection('LANDSAT/LT05/C01/T2')$filterDate('1987-01-01','1990-05-
01')$filterBounds(ee$Geometry$Point(25.8544,-18.08874))
filtered <- col$filterMetadata('IMAGE_QUALITY','equals',9)
notSoGood<-ee$Algorithms$Landsat$simpleComposite(col,75,3)
good<-ee$Algorithms$Landsat$simpleComposite(filtered,75,3)
Map$setCenter(25.8544,-18.08874,13)
Map$addLayer(notSoGood,
             list(bands=c('B3','B2','B1'), gain=3.5),
             'Bad Composition')
Map$addLayer(good,
             list(bands=c('B3', 'B2', 'B1'), gain=3.5),
             'Good Composition')
```

ee$ImageCollection informations and metadata

We can use functions to extract general information and matadata from an image collection.

```
col<-ee$ImageCollection('LANDSAT/LC08/C01/T1_TOA')
$filter(ee$Filter$eq('WRS_PATH',44))$filter(ee$Filter$eq('WRS_ROW',34))
$filterDate('2014-03-01', '2014-08-01')
ee_print(col)
```

```
── Earth Engine ImageCollection ──
ImageCollection Metadata:
 - Class                  : ee$ImageCollection
 - Number of Images       : 9
```

```
 - Number of Properties   : 41
 - Number of Pixels*      : 6435271800
 - Approximate size*      : 111.20 GB
Image Metadata (img_index = 0):
 - ID               : LANDSAT/LC08/C01/T1_TOA/LC08_044034_20140318
 - Time start             : 2014-03-18 18:46:32
 - Number of Bands        : 12
 - Bands names            : B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 BQA
 - Number of Properties   : 118
 - Number of Pixels*      : 715030200
 - Approximate size*      : 12.36 GB
Band Metadata (img_band = 'B1'):
 - EPSG (SRID)            : 32610
 - proj4string : +proj=utm +zone=10 +datum=WGS84 +units=m +no_defs
 - Geotransform           : 30 0 460785 0 -30 4264215
 - Nominal scale (meters) : 30
 - Dimensions             : 7650 7789
 - Number of Pixels       : 59585850
 - Data type              : FLOAT
 - Approximate size       : 1.03 GB
```

ee_print() informs you a large number of metadata and parameters of an image collection. We can also get specifics information with functions as the ones below:

```
size<-col$size()
cat("Size: ", size$getInfo(),'\n')

range<-col$reduceColumns(ee$Reducer$minMax(),list("system:time_start"))
col_min <- eedate_to_rdate(range$get("min"))
col_max <- eedate_to_rdate(range$get("max"))
cat("Date range: ", as.character(col_min),as.character(col_max),'\n')

sunStats <- col$aggregate_stats("SUN_ELEVATION")
sunStats$getInfo()

image <- ee$Image(coleção$sort("CLOUD_COVER")$first())
image$getInfo()

recent <- col$sort("system:time_start", FALSE)$limit(10)
recent$getInfo()
```

Visualizing ee$ImageCollection

Image collection are good to create image timelines over a certain area or region. We will create a series of examples using animations to illustrate how it can be done.

In the first example we use the MODIS NDVI composition. We selected the Northeast region of Brazil and we created an animation using the dates from January the 1st 2013 to January the 1st 2014.

```
col<-ee$ImageCollection('MODIS/006/MOD13A2')$select('NDVI')
mask<-ee$FeatureCollection('USDOS/LSIB_SIMPLE/2017')
$filter(ee$Filter$eq('wld_rgn', 'South America'))
region<-ee$Geometry$Polygon(list(c(-48.0, 0.0),
    c(-48.0, -17.0),c(-34.0, -17.0),c(-34.0, 0.0)))
col<-col$map(function(img){
 doy<-ee$Date(img$get('system:time_start'))$getRelative('day', 'year')
 return (img$set('doy', doy))})
distinctDOY<-col$filterDate('2013-01-01', '2014-01-01')
```

```
filter<-ee$Filter$equals(leftField='doy', rightField= 'doy')
join<-ee$Join$saveAll('doy_matches')
joinCol<-ee$ImageCollection(join$apply(distinctDOY, col, filter))
comp<-joinCol$map(function(img) {
  doyCol<-ee$ImageCollection$fromImages(img$get('doy_matches'))
  return (doyCol$reduce(ee$Reducer$median()))})
images<-comp$map(function(img) {
  return (img$visualize('NDVI_median')$clip(mask))})
gifParams<-list('region'=region,'dimensions'=600,'crs'='EPSG:3857',
'framesPerSecond'=5,palette=c(
    '#FFFFFF', '#CE7E45', '#DF923D', '#F1B555', '#FCD163', '#99B718', '#74A901',
    '#66A000', '#529400', '#3E8601', '#207401', '#056201', '#004C00', '#023B01',
    '#012E01', '#011D01', '#011301'),min=110.0,max=250.0)
print(images$getVideoThumbURL(gifParams))
```

The result is a link to the animation.
Alternatively click this link to see it http://amazeone.com.br/barebra/pandora/ne.gif

Now we will create an animation with the temperature variation between 20 and 23 of June 2018 for the South America using NOAA data.

```
aoi<-ee$Geometry$Polygon(list(c(-90,15),c(-90,-55),c(-35,-55),c(-35,15)))
tempCol<-ee$ImageCollection('NOAA/GFS0P25')$filterDate('2018-06-20', '2018-06-
23')$limit(72)$select('temperature_2m_above_ground')
videoArgs<-list(dimensions=600, region=aoi,framesPerSecond=7,crs='EPSG:3857',
  min=-40.0, max=35.0,palette=c('blue','purple','cyan','green','yellow','red'))
print(tempCol$getVideoThumbURL(videoArgs))
```

The result is a link to the animation.
Alternatively click this link to see it http://amazeone.com.br/barebra/pandora/sa.gif

We can see each one of frames used by the animation above using.

```
filmArgs<-list(dimensions=256,region=aoi,crs='EPSG:3857',
  min=-40.0,max=35.0,palette=c('blue','purple','cyan','green','yellow','red'))
print(tempCol$getFilmstripThumbURL(filmArgs))
```

The result is a link to the frames image.
Use this link to see it http://amazeone.com.br/barebra/pandora/filmstrip.png

Advanced visualization techniques using ee$ImageCollection

In the next examples we will see how to achieve advanced and integrated results using image collection.

In this example we add a feature object to the animation.

```
tempCol<-ee$ImageCollection('NOAA/GFS0P25')$filterDate('2018-06-22','2018-06-
23')$limit(24)$select('temperature_2m_above_ground')
tempColVis<-tempCol$map(function(img){return (img)})
southAmCol<-ee$FeatureCollection('USDOS/LSIB_SIMPLE/2017')
$filterMetadata('wld_rgn','equals','South America')
southAmAoi<-ee$Geometry$Rectangle(-90.6,-54.8,-28.4,17.4)
```

by creating overlays associated to the ImageCollection.

```
empty<-ee$Image()$byte()
southAmOutline<-empty$paint(featureCollection=southAmCol,color=1,width=
1)$visualize(palette='#000000')
tempColOutline<-tempColVis$map(function(img) {return
(img$blend(southAmOutline))})
videoArgs<-list(dimensions=600,region=southAmAoi,framesPerSecond=7 ,
crs='EPSG:3857',bands=c('vis-red'),min=-40.0, max=35.0,palette=c('blue',
'purple', 'cyan', 'green', 'yellow', 'red'))
print(tempColOutline$getVideoThumbURL(videoArgs))
```

The result is a link to the animation.

Use this link to see it http://amazeone.com.br/barebra/pandora/sa_adv1.gif

Now we create a mask over the animation

```
empty<-ee$Image()$byte()
southAmOutline<-empty$paint(featureCollection=southAmCol,color=1,width=
1)$visualize(palette='#000000')
tempColOutline<-tempColVis$map(function(img) {return
(img$blend(southAmOutline))})
dullLayer<-
ee$Image$constant(200)$visualize(opacity=0.55,min=0,max=255,forceRgbOutput=TRUE)
finalVisCol<-tempColOutline$map(function(img){return (img$blend(dullLayer)
$blend(img$clipToCollection(southAmCol)))})
videoArgs<-
list(dimensions=600,region=southAmAoi,framesPerSecond=7,crs='EPSG:3857',bands=c(
'vis-red'),min=-40.0, max=35.0,palette=c('blue', 'purple', 'cyan', 'green',
'yellow', 'red'))
print(finalVisCol$getVideoThumbURL(videoArgs))
```

The result is a link to the animation.

Use this link to see it http://amazeone.com.br/barebra/pandora/sa_adv2.gif

Now adding a Hillshade to the animação.

```
hillshade<-ee$Terrain$hillshade(ee$Image('USGS/SRTMGL1_003')$multiply(100))
$clipToCollection(southAmCol)
finalVisCol<-tempColVis$map(function(img){
  return (hillshade$blend(img$clipToCollection(southAmCol)
$visualize(opacity=0.5,palette=c('blue', 'purple', 'cyan', 'green', 'yellow',
'red'),min=-40,max=35))})
videoArgs <-list(dimensions=600,region=southAmAoi,framesPerSecond=7,
crs='EPSG:3857')
print(finalVisCol$getVideoThumbURL(videoArgs))
```

The result is a link to the animation.

Use this link to see it http://amazeone.com.br/barebra/pandora/sa_adv3.gif

In this last example we will create a transition animation using image collections. The example will flicker from an image from the winter with another from the summer and we will use the country boundary as a mask.

```
aoi<-ee$Geometry$Polygon(list(c(-90,15),c(-90,-55),c(-35,-55),c(-35,15)))
temp<-ee$ImageCollection('NOAA/GFS0P25')
#summer in northern hemisphere
summerSolTemp<-temp$filterDate('2018-06-21', '2018-06-22')
$filterMetadata('forecast_hours', 'equals', 12)$first()
$select('temperature_2m_above_ground')
#winter in northern hemisphere
winterSolTemp<-temp$filterDate('2018-12-22', '2018-12-23')
$filterMetadata('forecast_hours', 'equals', 12)$first()
$select('temperature_2m_above_ground')
#combining the two collections
tempCol<-ee$ImageCollection(list(summerSolTemp$set('season',
'summer'),winterSolTemp$set('season', 'winter')))
countries<-ee$FeatureCollection('USDOS/LSIB_SIMPLE/2017')
tempColVis<-tempCol$map(function(img) {
  return (img$clipToCollection(countries)$unmask(0)$copyProperties(img,
img$propertyNames()))})
```

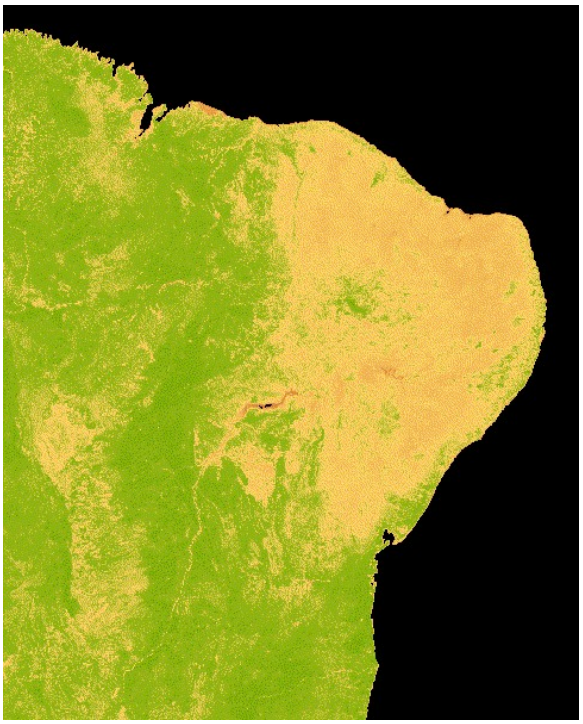
And creating the flicker visualization using:

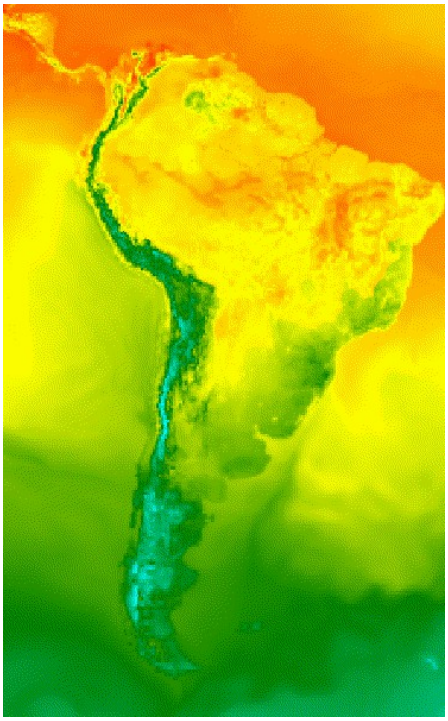
```
videoArgs<-list(dimensions=600,region=aoi,framesPerSecond=2,crs='EPSG:3857',
bands=c('temperature_2m_above_ground'),min=-40.0, max=35.0,palette=c('blue',
'purple', 'cyan', 'green', 'yellow', 'red'))
print(tempColVis$getVideoThumbURL(videoArgs))
```

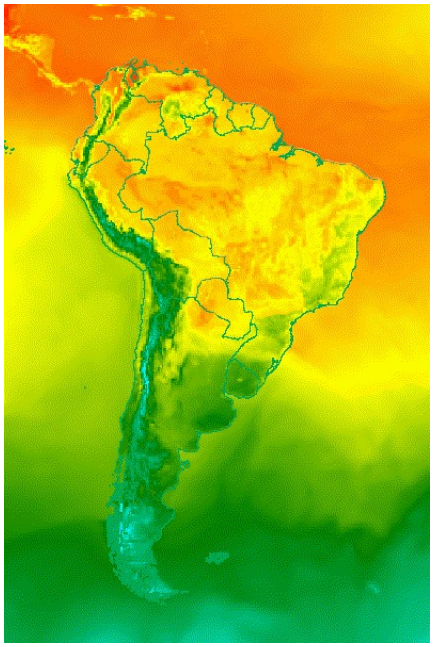
The result is a link to the animation.

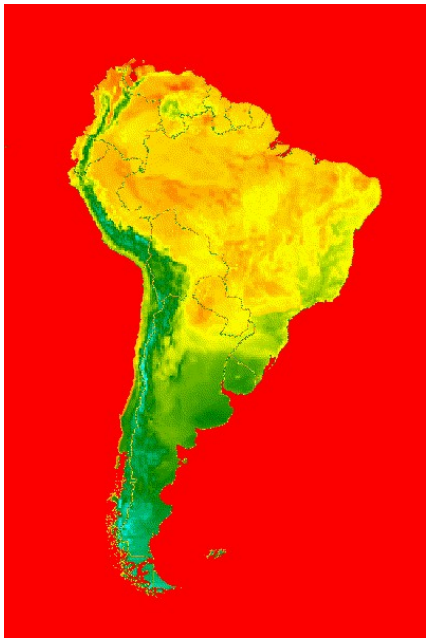
Use this link to see it http://amazeone.com.br/barebra/pandora/sa_trans1.gif

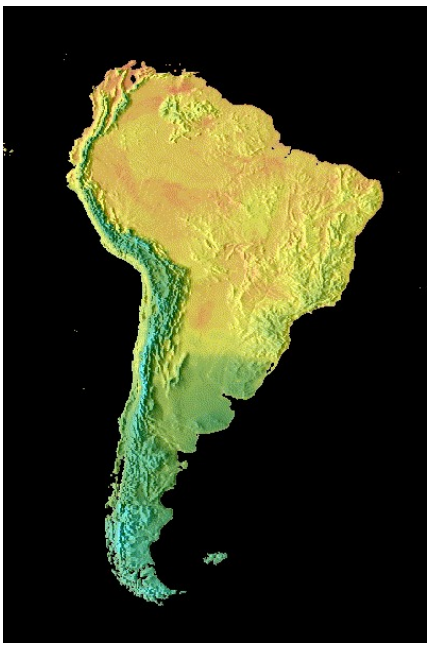
Some snapshots of the animations we created above:











ee$Geometry

Google earth engine deals with vector data using the ee$Geometry object. They are created in the geoJSON format. For more information about geoJSON please visit https://tools.ietf.org/html/rfc7946 .

```
point<-ee$Geometry$Point(c(1.5, 1.5))
point
ee.Geometry({
  "type": "Point",
  "coordinates": [
    1.5,
    1.5
  ]
})
lineString<-ee$Geometry$LineString(list(c(-35, -10), c(35, -10), c(35, 10), c(-
35, 10)))
lineString
ee.Geometry({
  "type": "LineString",
  "coordinates": [
    [
      -35.0,
      -10.0
    ],
    [
      35.0,
      -10.0
    ],
    [
      35.0,
      10.0
    ],
    [
      -35.0,
      10.0
    ]
  ]
})
linearRing<-ee$Geometry$LinearRing(list(c(-35,-10),c(35,-10),c(35,10),c(-
35,10),c(-35,-10)))
linearRing
ee.Geometry({
  "type": "LinearRing",
  "coordinates": [
    [
      -35.0,
      -10.0
    ],
    [
      35.0,
      -10.0
    ],
    [
      35.0,
      10.0
    ],
    [
      -35.0,
      10.0
    ],
    [
      -35.0,
      -10.0
    ]
  ]
})
```

```
retangle<-ee$Geometry$Rectangle(-40,-20,40,20)
retangle
ee.Geometry({
  "type": "Polygon",
  "coordinates": [
    [
      [
        -40.0,
        20.0
      ],
      [
        -40.0,
        -20.0
      ],
      [
        40.0,
        -20.0
      ],
      [
        40.0,
        20.0
      ]
    ]
  ],
  "evenOdd": true
})
polygon<-ee$Geometry$Polygon(list(c(-5,40),c(65,40),c(65,60),c(-5,60),c(-5,60)))
polygon
ee.Geometry({
  "type": "Polygon",
  "coordinates": [
    [
      [
        -5.0,
        40.0
      ],
      [
        65.0,
        40.0
      ],
      [
        65.0,
        60.0
      ],
      [
        -5.0,
        60.0
      ],
      [
        -5.0,
        60.0
      ]
    ]
  ],
  "evenOdd": true
})
```

A ee$Geometry object can be constructed using multiple geometries. To extract an individual part of this multi ee$Geometry object into each one of its element we can use the function `geometry.geometries()`. For example.

```
multiPoint<-ee$Geometry$MultiPoint(list(c(-121.68, 39.91), c(-97.38, 40.34)))
geometries<-multiPoint$geometries()
pt1<-geometries$get(0)
pt2<-geometries$get(1)
pt1
ee.ComputedObject({
```

```
  "type": "Invocation",
  "arguments": {
    "list": {
      "type": "Invocation",
      "arguments": {
        "geometry": {
          "type": "MultiPoint",
          "coordinates": [
            [
              -121.68,
              39.91
            ],
            [
              -97.38,
              40.34
            ]
          ]
        }
      },
      "functionName": "Geometry.geometries"
    },
    "index": 0.0
  },
  "functionName": "List.get"
})
pt2
ee.ComputedObject({
  "type": "Invocation",
  "arguments": {
    "list": {
      "type": "Invocation",
      "arguments": {
        "geometry": {
          "type": "MultiPoint",
          "coordinates": [
            [
              -121.68,
              39.91
            ],
            [
              -97.38,
              40.34
            ]
          ]
        }
      },
      "functionName": "Geometry.geometries"
    },
    "index": 1.0
  },
  "functionName": "List.get"
})
```

Planar x Geodesic

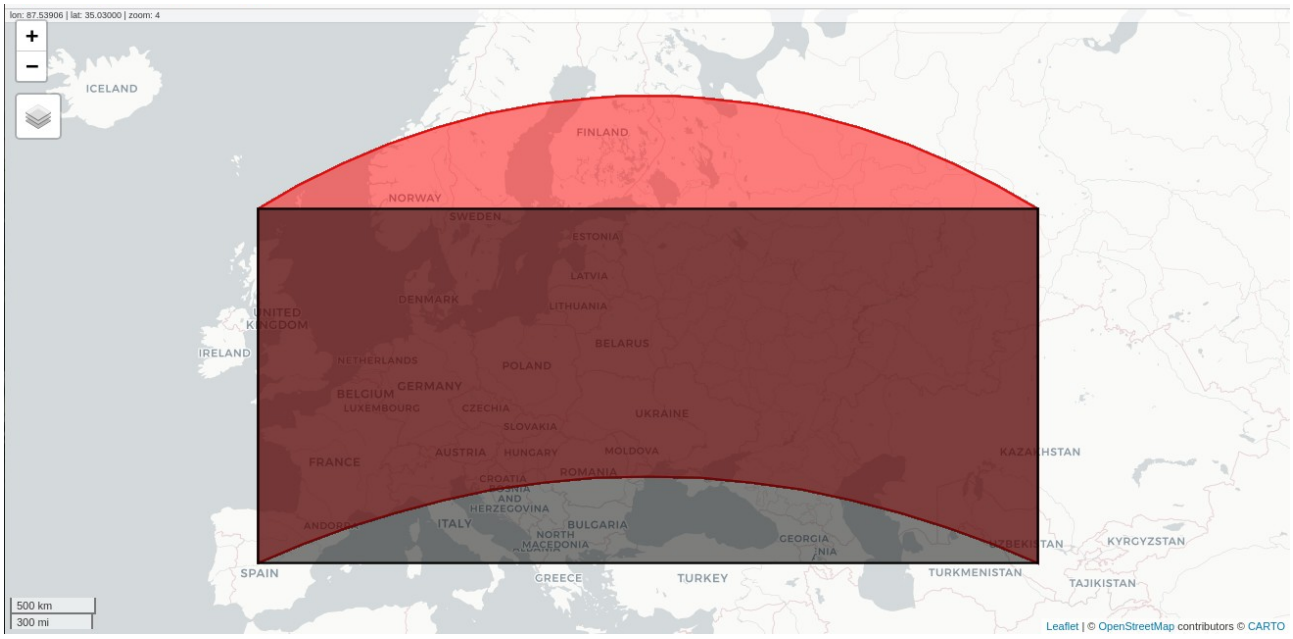
An ee$Geometry object geometry created using Earth Engine is geodesic (edges are the shortest path on the surface of a sphere) or planar (edges are the shortest path in a 2-D Cartesian plane). No one planar coordinate system is suitable for global collections of features, so Earth Engine's geometry constructors build geodesic geometries by default. To make a planar geometry, constructors have a geodesic parameter that can be set to false.

```
planarPolygon<-ee$Geometry(polygon, {}, FALSE)
```

In the next example we illustrate the difference between a geodesic (red) and a planar (black) ee$Geometry object.

```
polygon <- ee$Geometry$Polygon(list(c(-5, 40), c(65, 40), c(65, 60), c(-5, 60),
c(-5, 60)))
planarPolygon <- ee$Geometry(polygon, {}, FALSE)
polygon <- ee$FeatureCollection(polygon)
planarPolygon <- ee$FeatureCollection(planarPolygon)
Map$centerObject(polygon, zoom = 4)
Map$addLayer(polygon, list(color = "FF0000"), "Geodesic") +
Map$addLayer(planarPolygon, list(color = "000000"), "Planar")
```

The resulting map is:



ee$Geometry information and metadata

We just saw above how an ee$Geometry object is visualized in map, quite similar to how we do with image objects ee$image. Now we will see how to get information and metadata associated to a ee$Geometry object.

Using ee_print().

```
polygon <- ee$Geometry$Polygon(list(c(-5, 40), c(65, 40), c(65, 60), c(-5, 60),
c(-5, 60)))
ee_print(polygon)
```

```
── Earth Engine Geometry ──
Geometry Metadata:
 - EPSG (SRID)              : 4326
 - proj4string              : +proj=longlat +datum=WGS84 +no_defs
 - Geotransform             : 1 0 0 0 1 0
 - Geodesic                 : TRUE
 - WKT                      : POLYGON
```

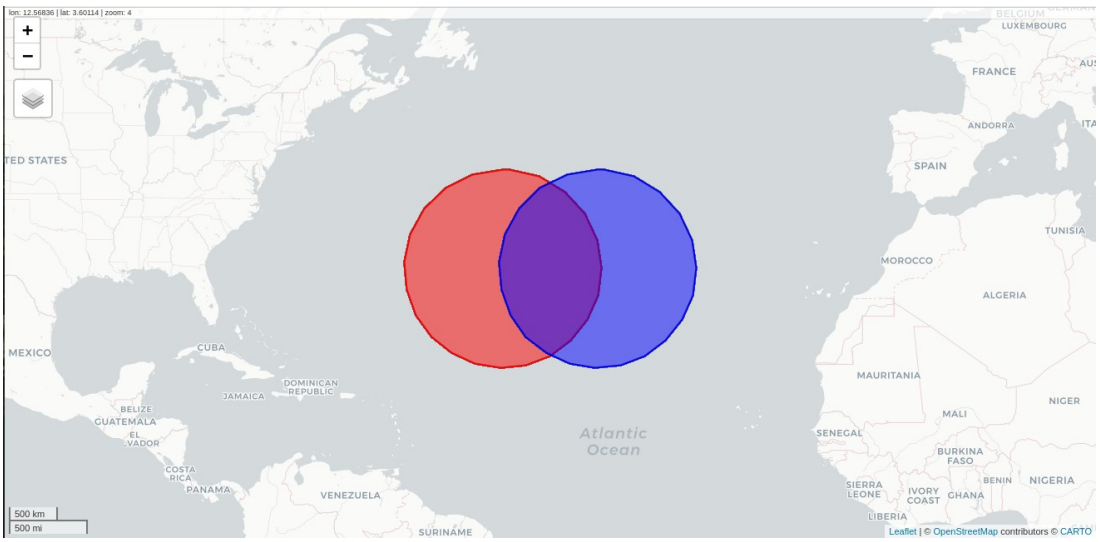
We can use the following to retrieve information about a ee$Geometry object:

```
polygon$area()$getInfo()
[1] 9.918986e+12
polygon$perimeter()$getInfo()
[1] 13979887
polygon$toGeoJSONString()
[1]  "{\"type\":  \"Polygon\",  \"coordinates\":  [[[-5.0,  40.0],  [65.0,  40.0],
[65.0, 60.0], [-5.0, 60.0], [-5.0, 60.0]]], \"evenOdd\": true}"
 polygon$type()$getInfo()
[1] "Polygon"
polygon$coordinates()$getInfo()
[[1]]
[[1]][[1]]
[1] -5 40

[[1]][[2]]
[1] 65 40

[[1]][[3]]
[1] 65 60

[[1]][[4]]
[1] -5 60

[[1]][[5]]
[1] -5 60

[[1]][[6]]
[1] -5 40

polygon$geodesic()$getInfo()
[1] TRUE
```

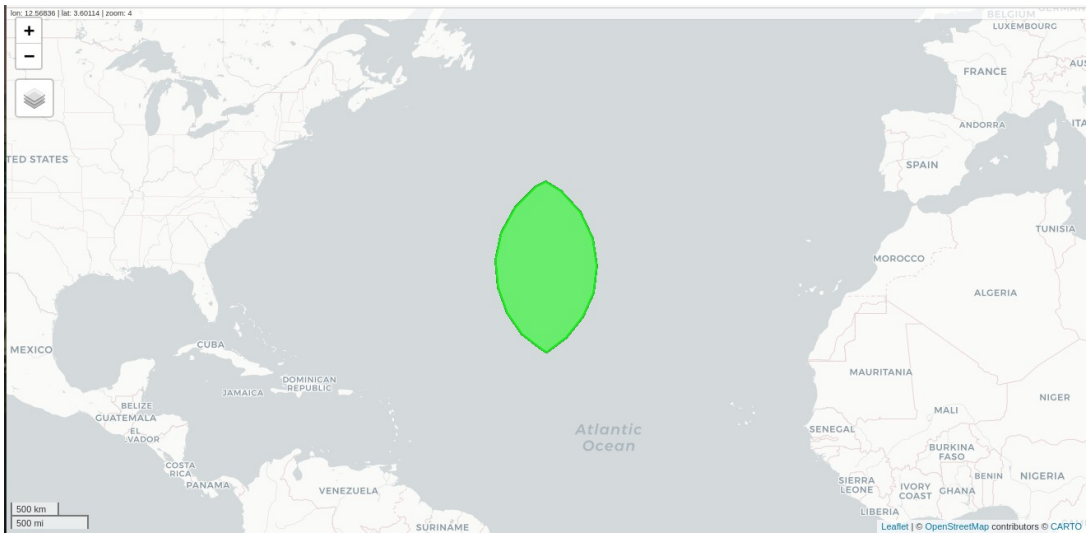
ee$Geometry operations

ee$Geometry objects have some ways to execute geospatial operations in Google Earth Engine. We will cover a few of them now using two circular geometries.

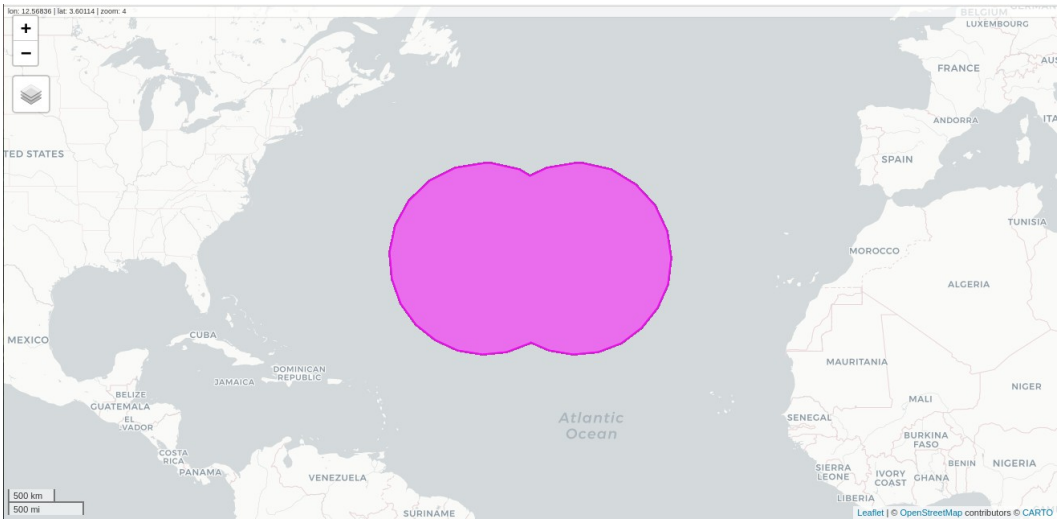
```
poly1 <- ee$Geometry$Point(c(-50, 30))$buffer(1e6)
poly2 <- ee$Geometry$Point(c(-40, 30))$buffer(1e6)
Map$setCenter(-45, 30,4)
Map$addLayer(poly1, list(color = "FF0000"), "poly1") +
```



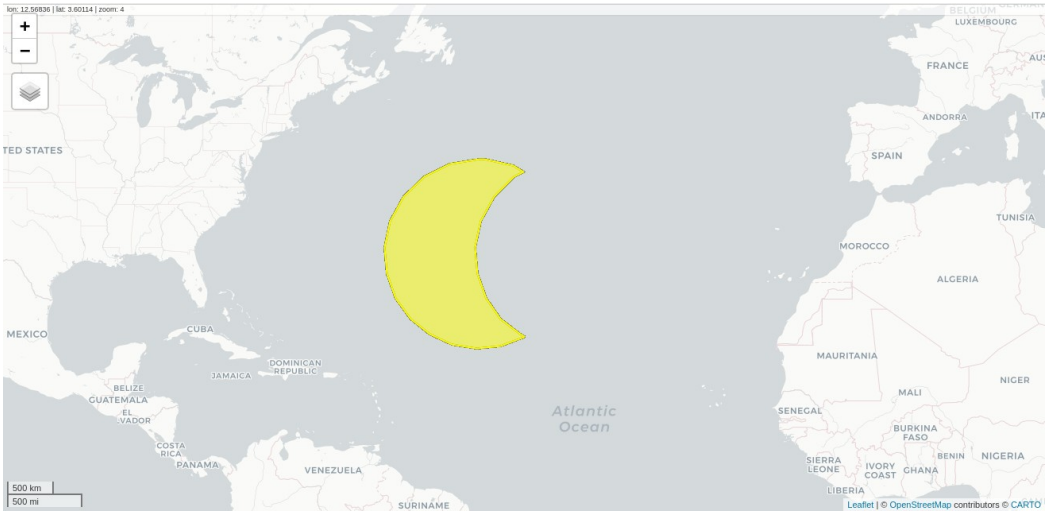
```
Map$addLayer(poly2, list(color = "0000FF"), "poly2")
intersection <- poly1$intersection(poly2, ee$ErrorMargin(1))
Map$addLayer(intersection, list(color = "00FF00"), "intersection")
```



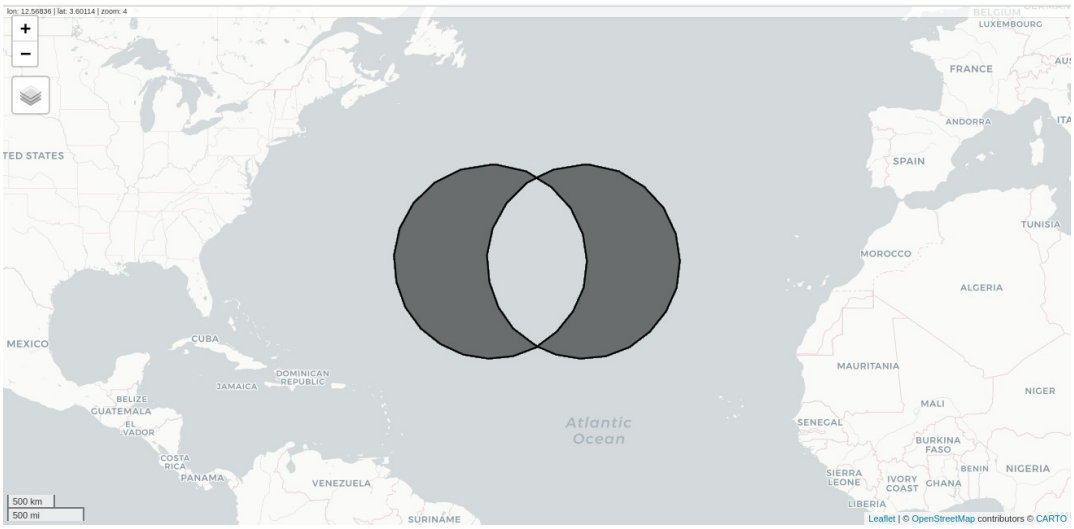

```
union <- poly1$union(poly2, ee$ErrorMargin(1))
Map$addLayer(union, list(color = "FF00FF"), "union")
```




```
diff1 <- poly1$difference(poly2, ee$ErrorMargin(1))
Map$addLayer(diff1, list(color = "FFFF00"), "difference")
```



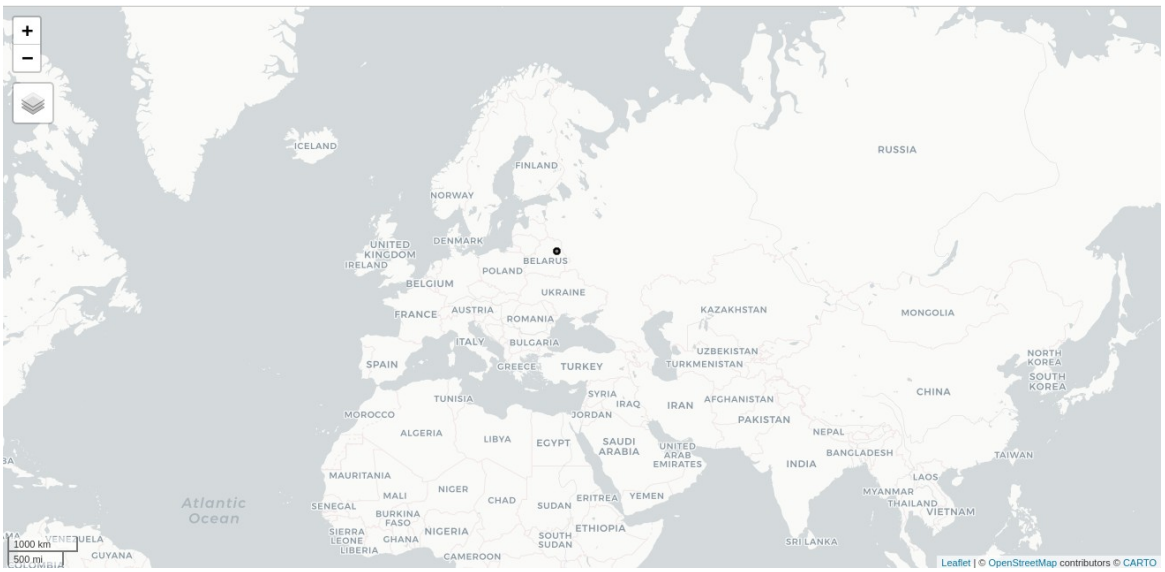
```
symDiff <- poly1$symmetricDifference(poly2, ee$ErrorMargin(1))
Map$addLayer(symDiff, list(color = "000000"), "symmetric difference")
```



Calculating buffer and centroid of a geometry.

```
polygon<-ee$Geometry$Polygon(list(c(-5,40),c(65,40),c(65,60),c(-5,60),c(-5,60)))
buffer<-polygon$buffer(1000000)
centroid<-polygon$centroid()
Map$setCenter(35,50,3)
Map$addLayer(buffer,list(),'buffer')
```

```
Map$addLayer(centroid,list(),'centroid')
```



ee$Feature

No Earth Engine um objeto Feature é definido com uma característica GeoJSON. Especificamente, uma característica ou Feature é um objeto composto por uma propriedade geométrica com um objeto Geometry (ou vazio) e uma propriedade property com um dicionário de outras propriedades não geométricas ou atributos.

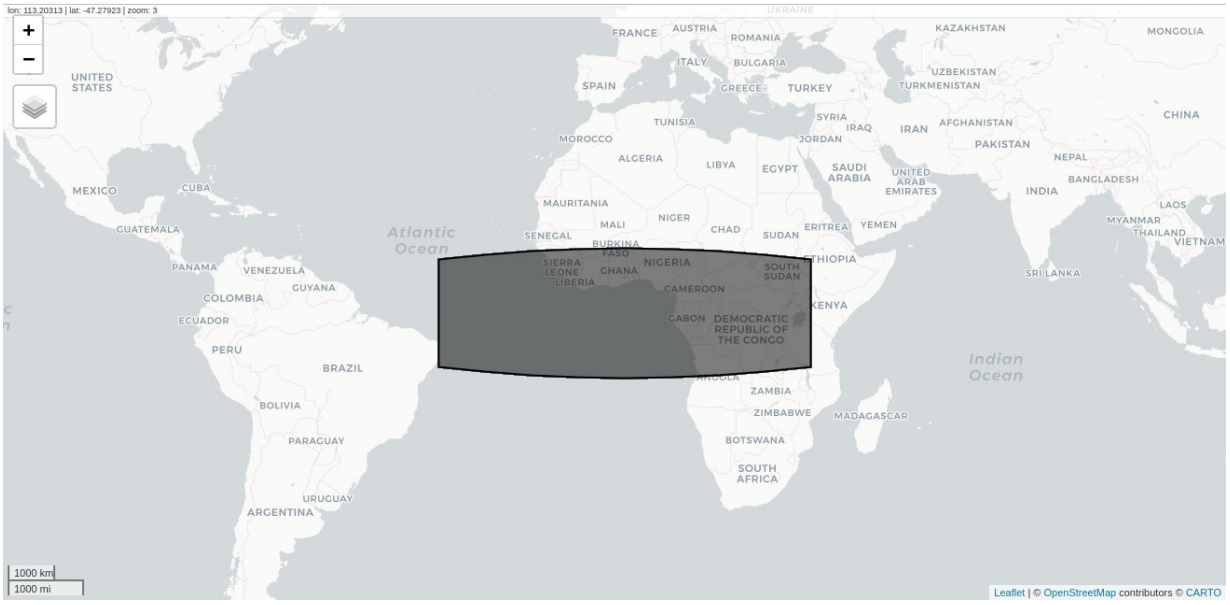
Neste exemplo criamos um objeto ee$Feature.

```
polygon<-ee$Geometry$Polygon(list(c(-35,-10),c(35,-10),c(35,10),c(-35,10),c(-35,-10)))
polyFeature<-ee$Feature(polygon,list(foo=42, bar='tart'))
ee_print(polyFeature)
```

```
──────────────────────────────────────────────────────────── Earth Engine Feature ──
Feature Metadata:
 - Number of Properties       : 2
Geometry Metadata:
 - EPSG (SRID)                : 4326
 - proj4string                : +proj=longlat +datum=WGS84 +no_defs
 - Geotransform               : 1 0 0 0 1 0
 - Geodesic                   : TRUE
 - WKT                        : POLYGON
──────────────────────────────────────────────────────────────────────────────────
```

Plotting on a map.

```
Map$addLayer(polyFeature, list(), 'feature')
```



Information of this object can be accessed using:

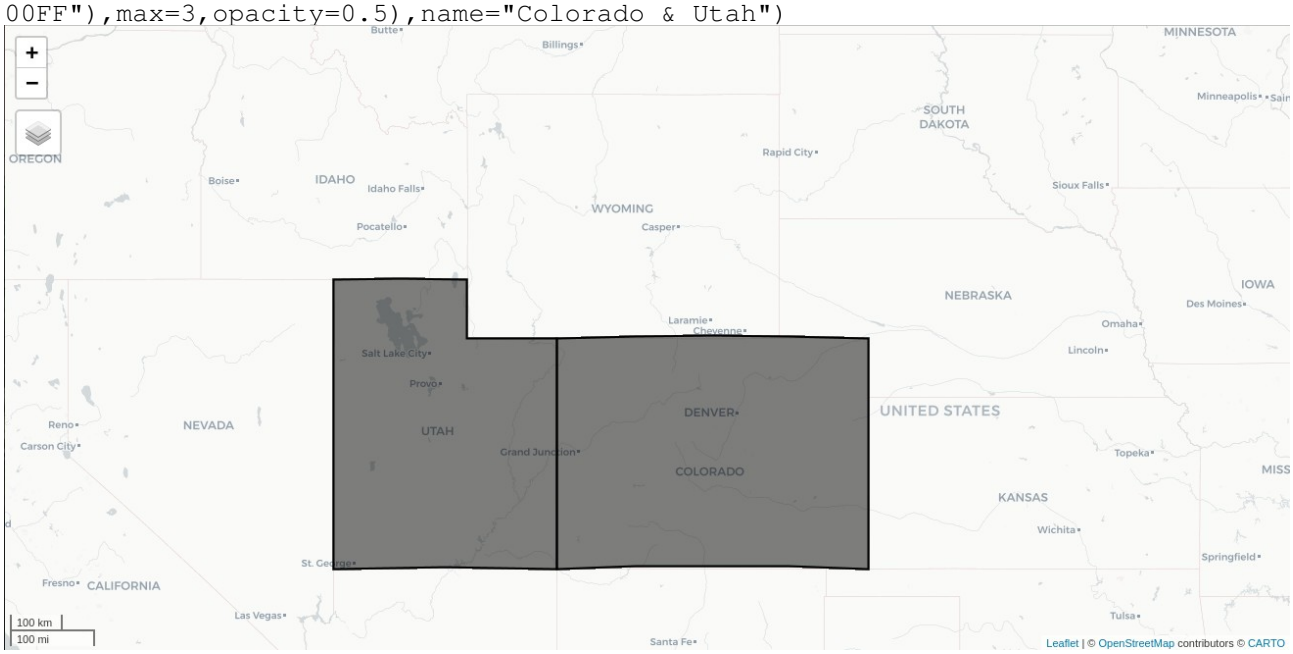
```
polyFeature$getInfo()
$type
[1] "Feature"

$geometry
$geometry$type
[1] "Polygon"
$geometry$coordinates
$geometry$coordinates[[1]]
$geometry$coordinates[[1]][[1]]
[1] -35 -10
$geometry$coordinates[[1]][[2]]
[1]  35 -10
$geometry$coordinates[[1]][[3]]
[1] 35 10
$geometry$coordinates[[1]][[4]]
[1] -35  10
$geometry$coordinates[[1]][[5]]
[1] -35 -10

$properties
$properties$bar
[1] "tart"

$properties$foo
[1] 42
```

ee$FeatureCollection

In the next example we will show how to create a eeFeatureCollection object and hoe to visualize it on a map.

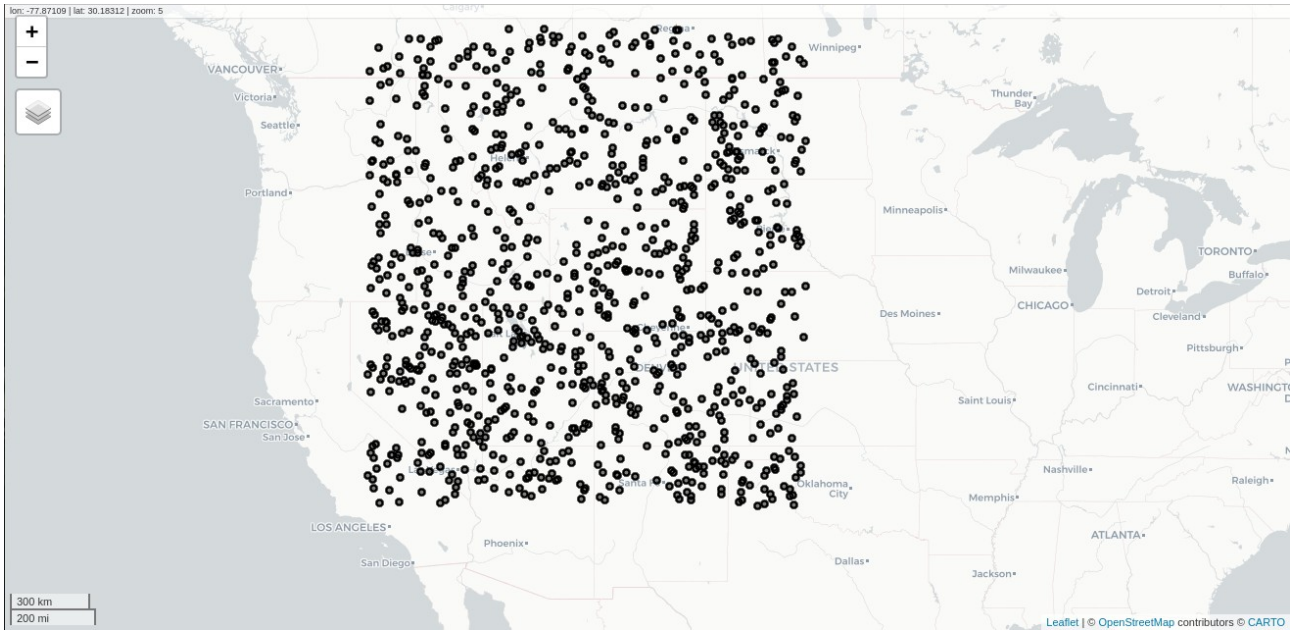
```
fc <- ee$FeatureCollection(list(
ee$Feature(ee$Geometry$Polygon(list(c(-109.05, 41),c(-109.05,37),c(-
102.05,37),c(-102.05,41))),list(name="Colorado",fill=1)),
ee$Feature(ee$Geometry$Polygon(list(c(-114.05,37.0),c(-109.05,37.0),c(-
109.05,41.0),c(-111.05,41.0),c(-111.05,42.0),c(-114.05,42.0))),list(name="Utah",
fill=2))
```

```
))
Map$setCenter(-107,41,6)
Map$addLayer(eeObject=fc,visParams=list(palette=c("000000","FF0000","00FF00","00
00FF"),max=3,opacity=0.5),name="Colorado & Utah")
```



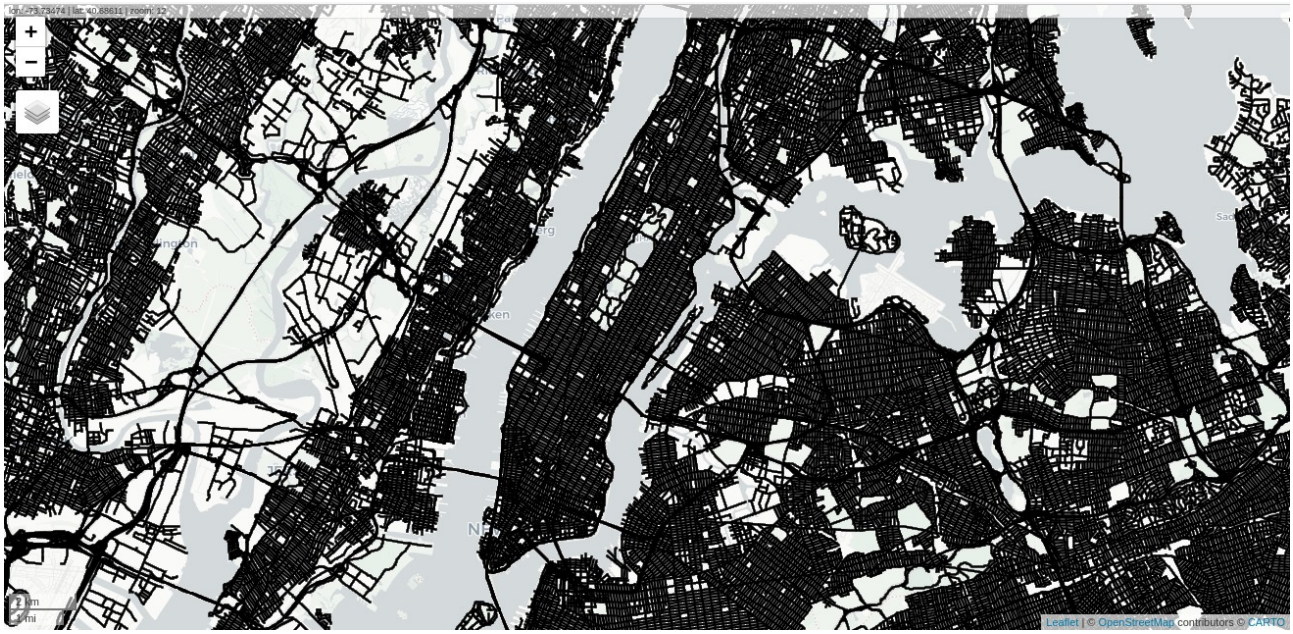
Now creating a ee$featureCollection using random points.

```
region<-ee$Geometry$Rectangle(-119.224, 34.669, -99.536, 50.064)
randomPoints<-ee$FeatureCollection$randomPoints(region)
Map$setCenter(-107,41,5)
Map$addLayer(randomPoints, list(), 'Random Points')
```



Creating a ee$FeatureCollection based on Google Earth Engine data.

```
fc<-ee$FeatureCollection('TIGER/2016/Roads')
Map$setCenter(-73.9596, 40.7688, 12)
Map$addLayer(fc, list(), 'NY Streets Census data')
```

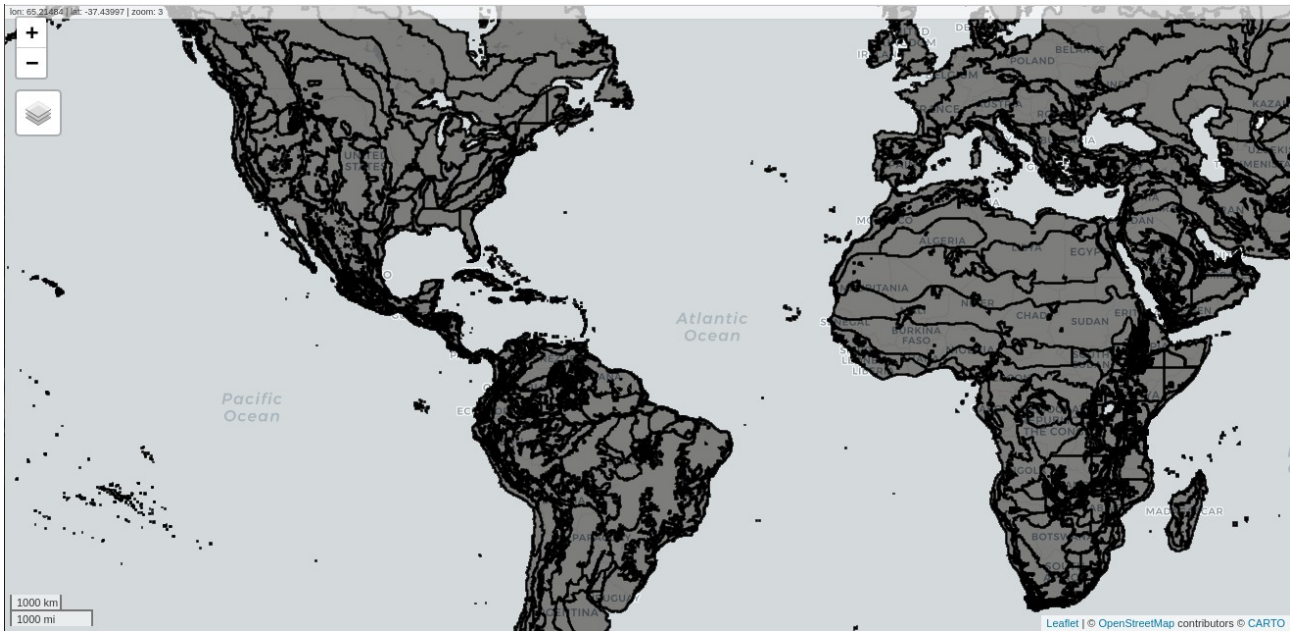


ee$FeatureCollection visualization

Visualizing an ee$FeatureCollection can the done in similar way that we do with other objects in GEE.

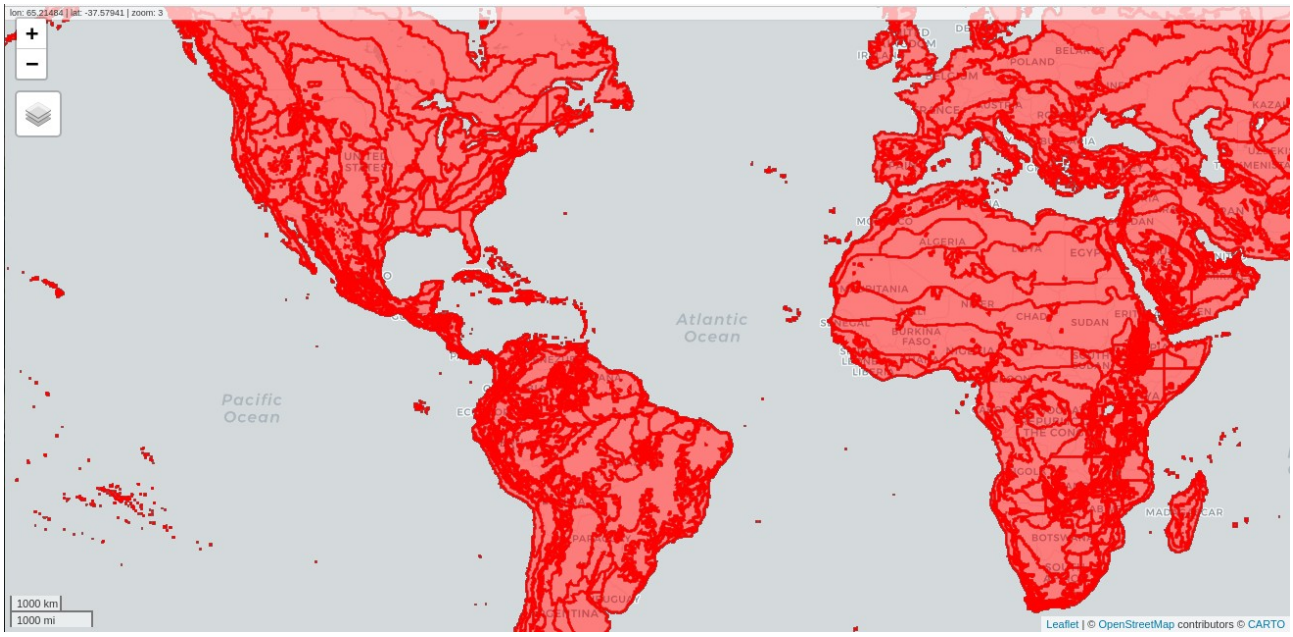
Creating a ee$Feature Collection objct and visualizing it using standard parameters.

```
Map$setCenter(-50,15,3)
ecoregions<-ee$FeatureCollection('RESOLVE/ECOREGIONS/2017')
Map$addLayer(ecoregions, list(), 'Standard')
```



Using visualization parameters:

```
Map$addLayer(ecoregions, list(color='FF0000'),'color')
```

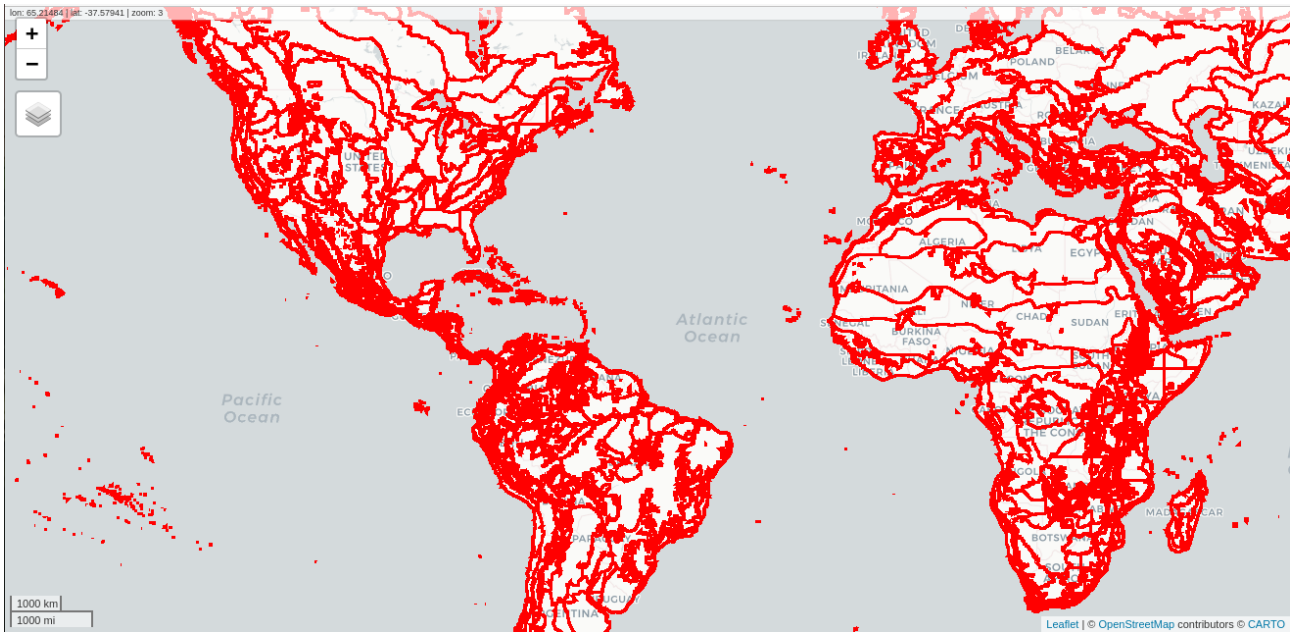


Using the draw function.

```
Map$addLayer(ecoregions$draw(color='green',strokeWidth=5),list(),'draw')
```

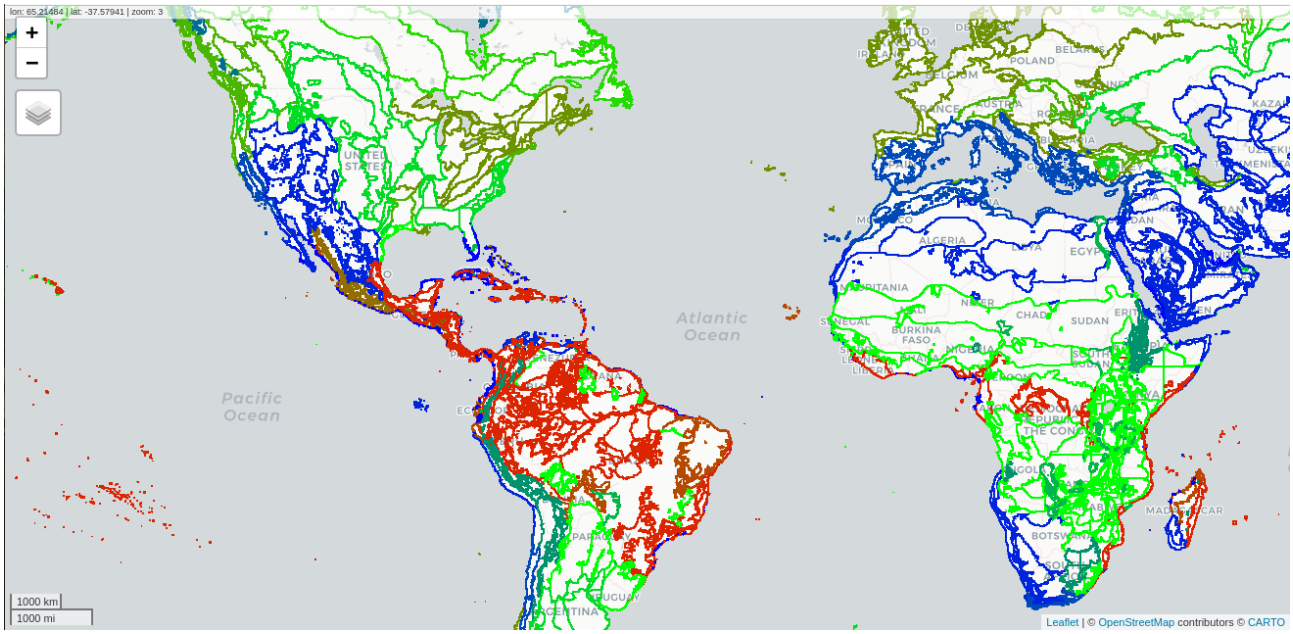
We can categorize efficiently what we want to see using the ee$FeatureCollection properties. This is done using the function `paint`. Initially we will use a fixed value for the edges.

```
empty<-ee$Image()$byte()
outline<-empty$paint(featureCollection=ecoregions,color=1,width=3)
Map$addLayer(outline, list(palette='#FF0000'),'edges')
```



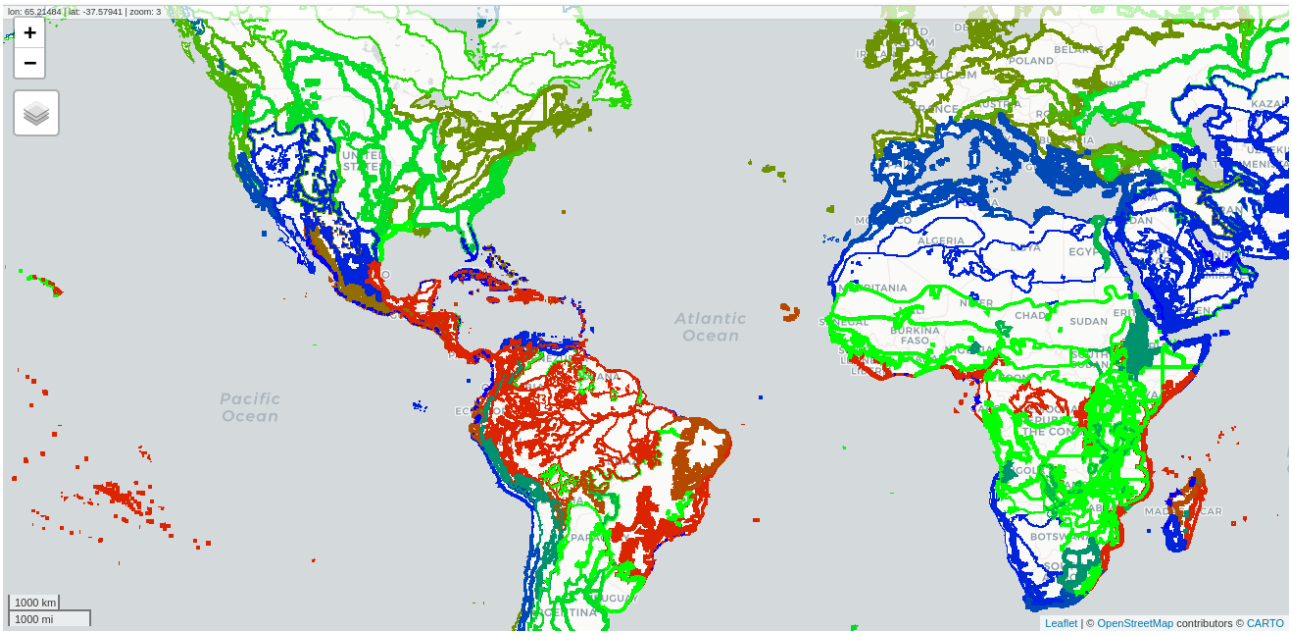
Using the parameter 'BIOME_NUM' to color the object edges creating an RGB palette.

```
outlines<-empty$paint(featureCollection=ecoregions,color='BIOME_NUM',width=2)
palette<-c('#FF0000', '#00FF00', '#0000FF')
Map$addLayer(outlines, list(palette=palette, max=14),'Color edges')
```



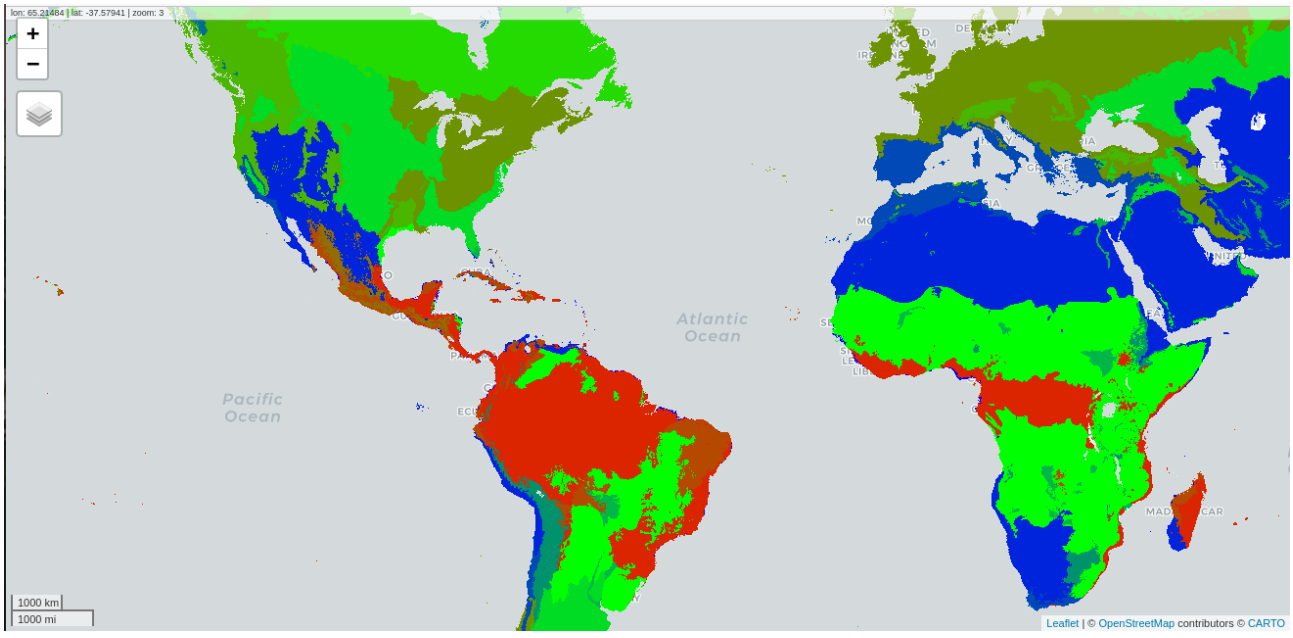
Using property 'NNH' to define the edge thickness.

```
outlines<-
empty$paint(featureCollection=ecoregions,color='BIOME_NUM',width='NNH')
Map$addLayer(outlines,    list(palette=palette,    max=14),'Edges    color    and
thickness')
```



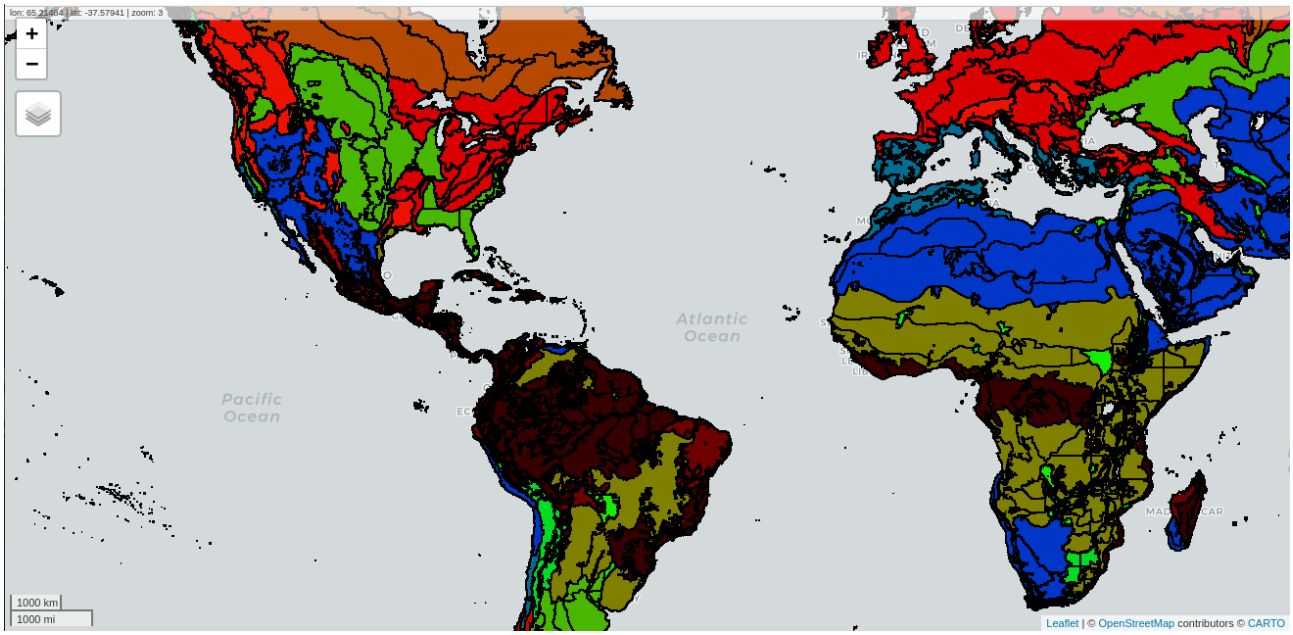
Now color filling the features using 'BIOME_NUM'.

```
fills<-empty$paint(featureCollection=ecoregions,color='BIOME_NUM')
Map$addLayer(fills, list(palette=palette, max=14), 'Color Fill')
```



To finalize, edges and color fill.

```
filledOutlines<-empty$paint(ecoregions, 'BIOME_NUM')$paint(ecoregions, 0, 2)
Map$addLayer(filledOutlines,list(palette= c('#000000',palette), max=14), 'Edges
and color fill')
```



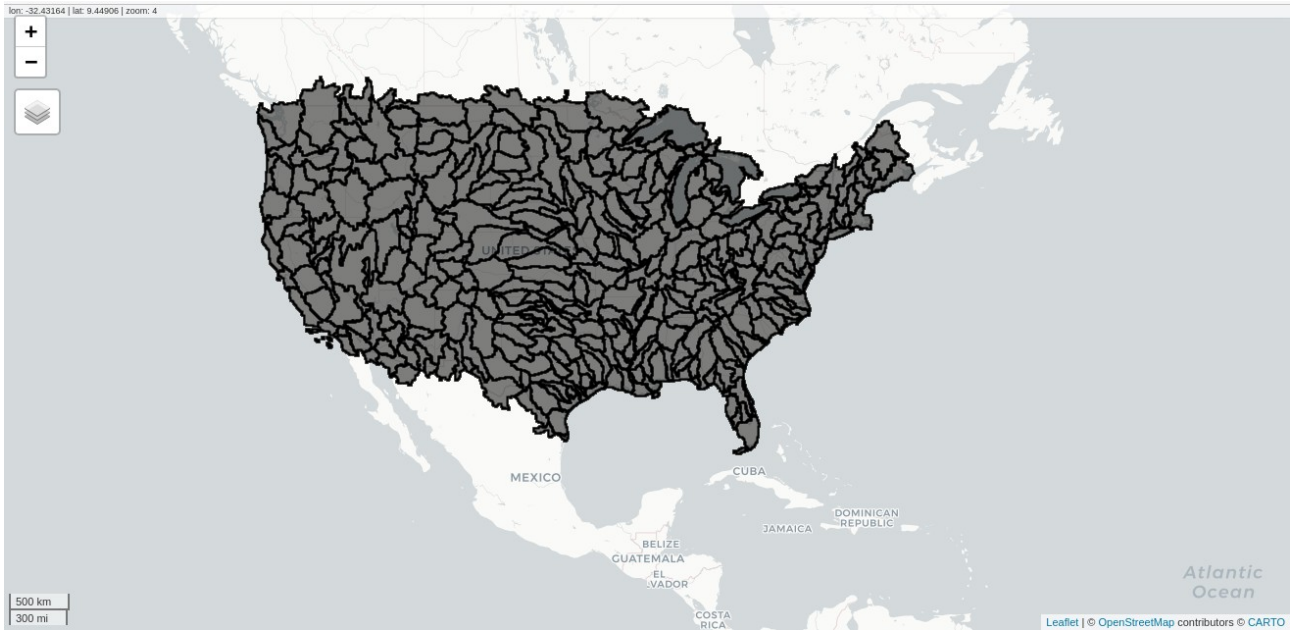
ee$FeatureCollection information and metadata

Retrieving information and metadata from a ee$FeatureCollection using ee_print().

```
sheds<-ee$FeatureCollection('USGS/WBD/2017/HUC06')
$filterBounds(ee$Geometry$Rectangle(-127.18,     19.39,     -62.75,     51.29))
$map(function(feature){
     num<-ee$Number$parse(feature$get('areasqkm'))
     return (feature$set('areasqkm', num))
   })
ee_print(sheds)
```

```
── Earth Engine FeatureCollection ──
FeatureCollection Metadata:
 - Class                      : ee$FeatureCollection
 - Number of Features         : 336
 - Number of Properties       : 14
Feature Metadata:
 - Number of Properties       : 14
Geometry Metadata (f_index = 0):
 - EPSG (SRID)                : 4326
 - proj4string                : +proj=longlat +datum=WGS84 +no_defs
 - Geotransform               : 1 0 0 0 1 0
 - Geodesic                   : TRUE
 - WKT                        : POLYGON
```

```
Map$setCenter(-90,35,4)
Map$addLayer(sheds, list(), 'watersheds')
```



Getting specific information from an ee$FeatureCollection object.

```
sheds$size()$getInfo()
  [1] 336
sheds$aggregate_stats('areasqkm')$getInfo()
$type
  [1] "DataDictionary"
$values
 $values$max
  [1] 123604.1
 $values$mean
  [1] 25513.32
 $values$min
  [1] 601.89
 $values$sample_sd
  [1] 17082.47
 $values$sample_var
  [1] 291810677
 $values$sum
  [1] 8572476
 $values$sum_sq
  [1] 316468933433
 $values$total_count
  [1] 336
 $values$total_sd
  [1] 17057.03
 $values$total_var
  [1] 290942193
 $values$valid_count
  [1] 336
 $values$weight_sum
  [1] 336
 $values$weighted_sum
  [1] 8572476
```

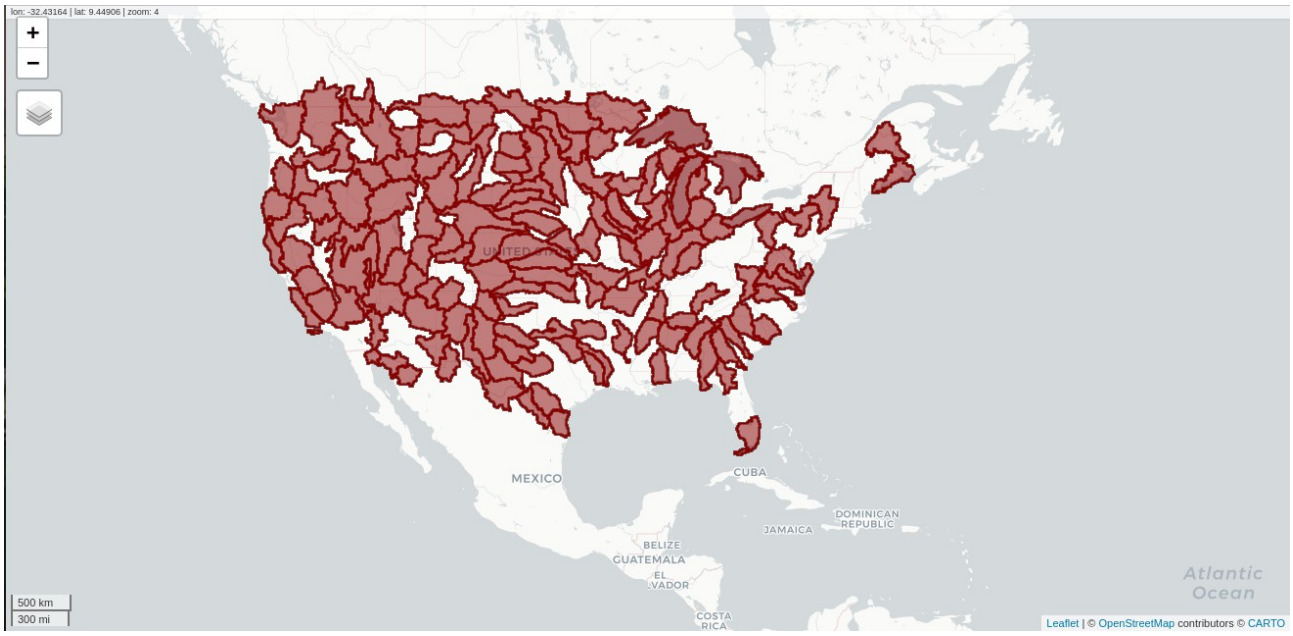
Getting columns names from a feature of an ee$FeatureCollection object.

```
wdpa<-ee$FeatureCollection("WCMC/WDPA/current/points")
columns<-wdpa$first()$getInfo()
names(columns$properties)
 [1] "DESIG"      "DESIG_ENG"  "DESIG_TYPE" "GOV_TYPE"   "INT_CRIT"
 [6] "ISO3"       "IUCN_CAT"   "MANG_AUTH"  "MANG_PLAN"  "MARINE"
[11] "METADATAID" "NAME"       "NO_TAKE"    "NO_TK_AREA" "ORIG_NAME"
[16] "OWN_TYPE"   "PARENT_ISO" "PA_DEF"     "REP_AREA"   "REP_M_AREA"
[21] "STATUS"     "STATUS_YR"  "SUB_LOC"    "VERIF"      "WDPAID"
[26] "WDPA_PID"
```

ee$FeatureCollection filtering

We will cover now how to filter information and data from an ee$FeatureCollection object. The filtering is done in a similar way as we did with the other objects so far.
In this example we will load data from USGS hydrographic basis and filter the ones inside continental US. We will check the number of basis first and the next step we will filter again to extract only the basins with an area grater than 25000 squared km.

```
sheds<-ee$FeatureCollection('USGS/WBD/2017/HUC06')$map(function(feature){
    num<-ee$Number$parse(feature$get('areasqkm'))
    return (feature$set('areasqkm', num))})
continentalUS<-ee$Geometry$Rectangle(-127.18, 19.39, -62.75, 51.29)
filtered<-sheds$filterBounds(continentalUS)
filtered$size()$getInfo()
[1] 336
largeSheds<-filtered$filter(ee$Filter$gt('areasqkm', 25000))
largeSheds$size()$getInfo()
[1] 140
Map$setCenter(-90,35,4)
Map$addLayer(largeSheds, list(color='#880000'), 'watersheds')
```



Map an ee$FeatureCollection object

We will demonstrate now how to change all the features inside an ee$FeatureCollection object using map. In the same way we used map with an ee$ImageCollection it will be used with ee$FeatureCollection passing a ee$Feature object as parameter instead an ee$ image object.

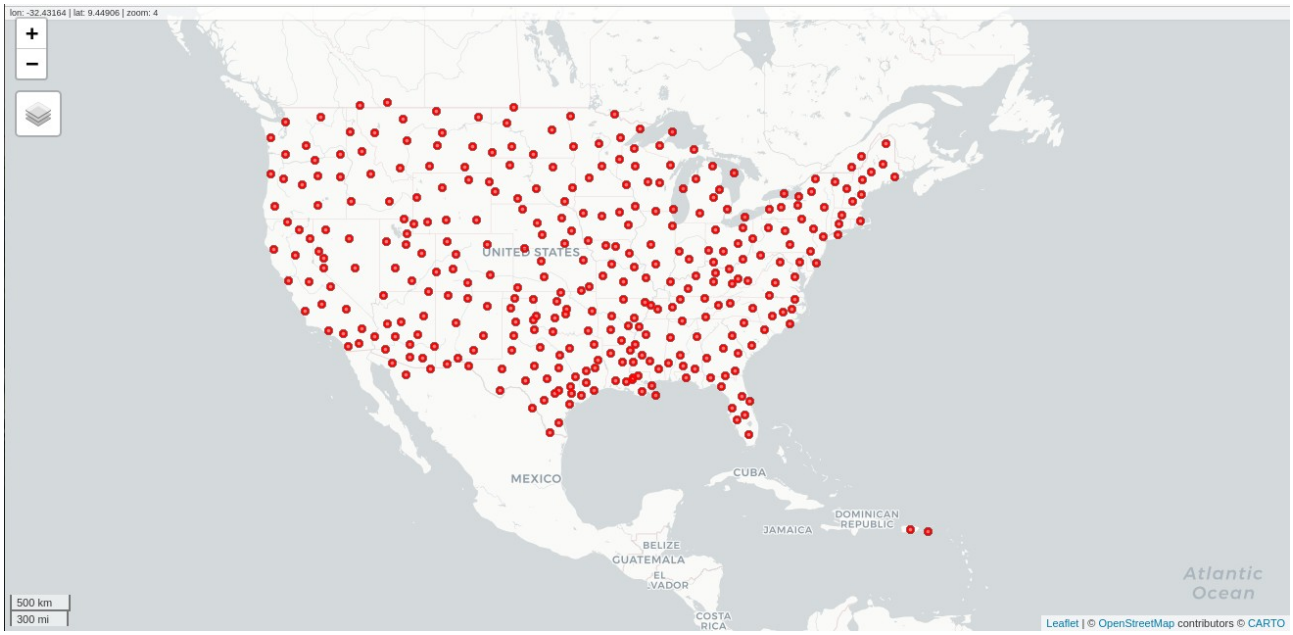
Adding a new column.

```
sheds<-ee$FeatureCollection('USGS/WBD/2017/HUC06')
columns<-sheds$first()$getInfo()
columns$properties
$areaacres
[1] "17935746.85"
$areasqkm
[1] "72583.46"
$gnis_id
[1] ""
$huc6
[1] "190301"
$loaddate
[1] "20120611075458"
$metasource
[1] ""
$name
[1] "Aleutian Islands"
$shape_area
[1] "9.9308571861444"
$shape_leng
[1] "81.5083456556245"
$sourcedata
[1] ""
$sourcefeat
[1] ""
$sourceorig
[1] ""
$states
[1] "AK"
$tnmid
[1] "{7D8E9EC1-6A0D-4FED-8063-94688560AC75}"
addArea<-function(feature){
     return   (feature$set(list(areaHa=   feature$geometry()$area()$divide(100   *
100))))
}
areaAdded<-sheds$map(addArea)
columns2<-areaAdded$first()$getInfo()
columns2$properties
$areaHa
[1] 7227401
$areaacres
[1] "17935746.85"
$areasqkm
[1] "72583.46"
$gnis_id
[1] ""
$huc6
[1] "190301"
$loaddate
[1] "20120611075458"
$metasource
[1] ""
$name
[1] "Aleutian Islands"
$shape_area
[1] "9.9308571861444"
$shape_leng
[1] "81.5083456556245"
$sourcedata
[1] ""
$sourcefeat
[1] ""
$sourceorig
[1] ""
$states
[1] "AK"
$tnmid
[1] "{7D8E9EC1-6A0D-4FED-8063-94688560AC75}"
```

In the example above we created a neu property (attribute) based on the ee$Feature object geometry. New attributes can be created also by computing the existing parameters.

A new ee$FeatureCollection can be entirely created using map(). The example below convert the 'watershed' object in an ee$FeatureCollection 'centroid' and select only some attributes of the original object.

```
getCentroid<-function(feature) {
 keepProperties = list('name', 'huc6', 'tnmid', 'areasqkm')
 centroid = feature$geometry()$centroid()
 return (ee$Feature(centroid)$copyProperties(feature, keepProperties))
}
centroids<- sheds$map(getCentroid)
Map$addLayer(centroids, list(color='FF0000'),'centroids')
```



And checking the columns created into the ee$FeatureCollection 'centroid'.

```
columns<-centroids$first()$getInfo()
names(columns$properties)
[1] "areasqkm" "huc6"     "name"     "tnmid"
columns$properties
$areasqkm
[1] "72583.46"
$huc6
[1] "190301"
$name
[1] "Aleutian Islands"
$tnmid
[1] "{7D8E9EC1-6A0D-4FED-8063-94688560AC75}"
```

In the next volume we will finalize covering ee$Reducer and ee$Join
Fortaleza July 20, 2020.