SINERGISE

SENTINEL Hub

CUSTOM SCRIPT TUTORIAL

Create Useful and Beautiful Satellite Visualizations in EO Browser

Prepared by:

Sinergise laboratory for geographical information systems, Ltd.
Cvetkova ulica 29, 1000 Ljubljana
27.6.2019, Ljubljana

Contents

1 Preface 3

2 A gentle theoretical introduction 4

2.1 Optical satellites 4

2.2 Band calculations using map algebra 5

2.3 Color 6

3 Your first custom script 7

3.1 Create a composite in EO Browser 7

3.2 Start exploring custom scripts 11

3.3 Visualize the Betsiboka river delta 13

4 Implement a remote sensing index 15

5 Custom color scales 18

5.1 Discrete color scale 18

5.2 Continuous color scale 20

6 Concluding remarks 23

7 References 24

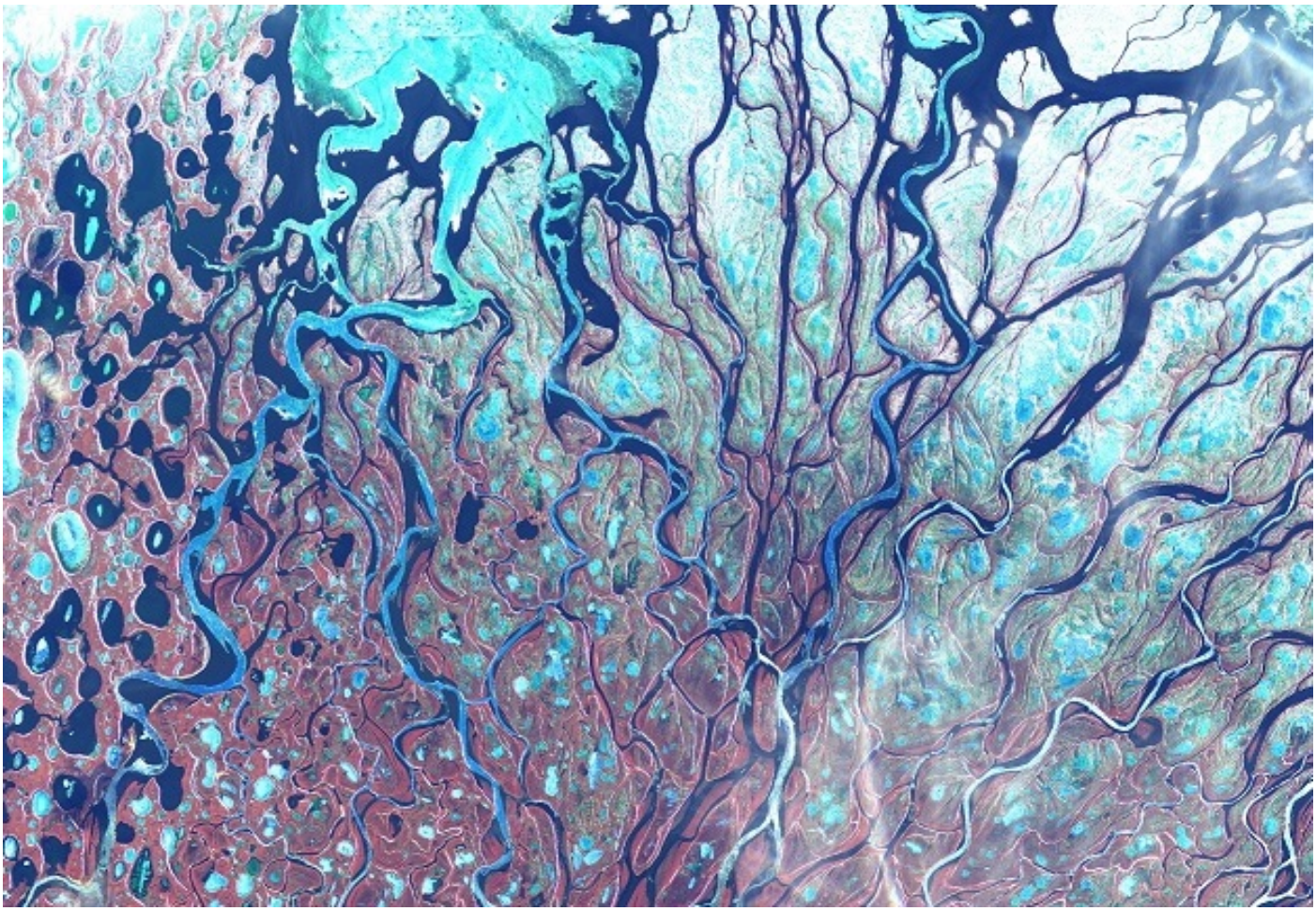
1 Preface

Custom scripts are javascript code, used to control the visualization and processing of satellite images with Sentinel Hub. They allow the user to quickly calculate different band combinations and visualize them, without previously downloading data.

Sentinel Playground and **EO Browser** are web applications, that allow easy access to free satellite data provided by the **European Space Agency (ESA)** and **The National Aeronautics and Space Administration (NASA).** They also provide an environment for creating custom scripts.

It only takes a minute to create a beautiful image, such as the image below and 5 minutes to implement the visualization of a simple remote sensing index, such as the **Normalized difference vegetation index (NDVI).**

This is a tutorial for those interested in creating custom scripts, who are not familiar with programming. If you are not confident in understanding the basic remote sensing concepts, we provided you with a short theory introduction, that should prepare you for the process of scripting in EO Browser. Feel free to skip ahead to *Your first custom script*, if you feel you don't need it.

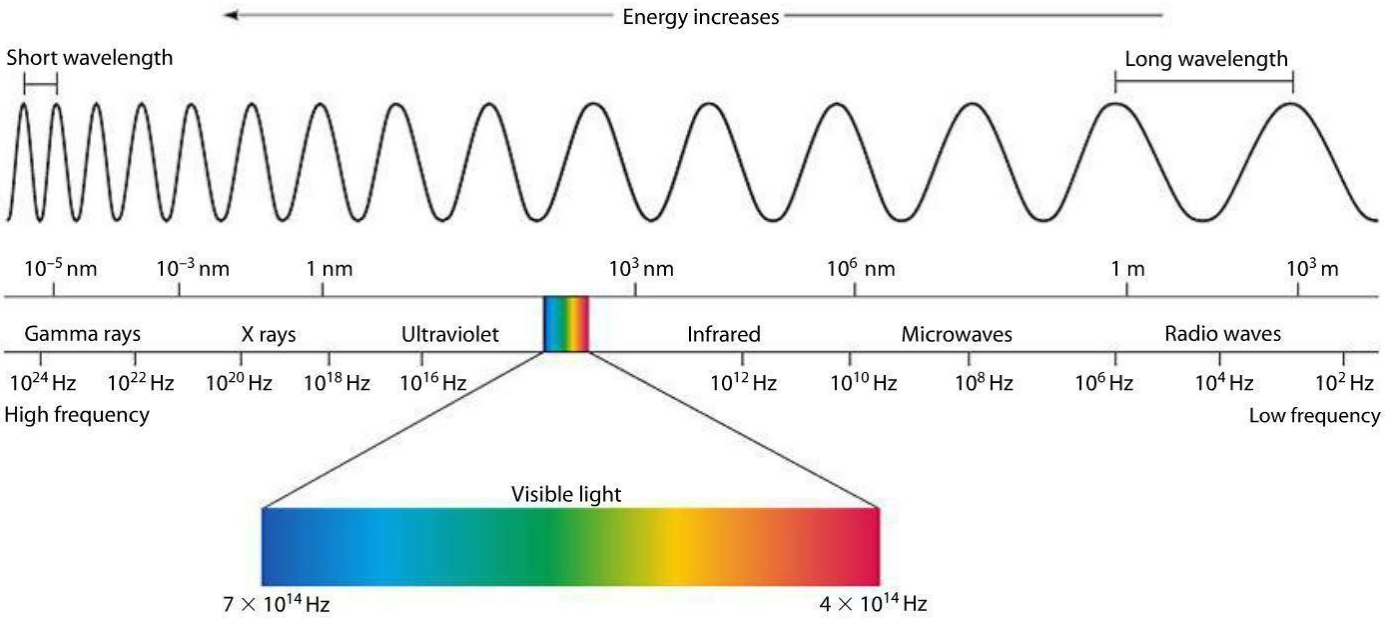


Lena river delta in Russia (modified Copernicus Sentinel-2 data processed by Sentinel Hub, acquired on 10.6.2019). ***Inspect in EO Browser.***

2 A gentle theoretical introduction

2.1 Optical satellites

To start creating beautiful and useful images, the first thing we need to understand is the electromagnetic spectrum. You can see the representation of it on the image below. Electromagnetic energy travels in the form of waves with different frequencies. The frequency of a wave is related to its wavelength; the shorter the wavelength, the higher the frequency and also the energy of a wave. Frequency is the main characteristic of waves and thus determines how the wave is going to interract with matter.

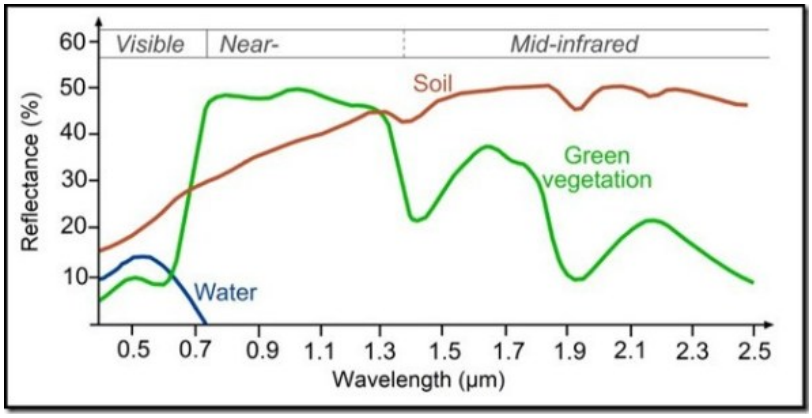


*The electromagnetic spectrum (**Miniphisycs.com**)*

Objects absorb specific wavelengths of light and reflect others, based on the material. For example, plants absorb blue and red light, while they reflect green light. That is why they appear green to the naked eye. However, they also strongly reflect some parts of the infrared spectrum. If we were to look at an infrared image, we would clearly see vegetation.

Satellites orbit the Earth and capture images of it, similar to cameras. They carry light sensors, each sensitive to a specific electromagnetic wavelength range. Optical satellites have sensors sensitive to visible (red, green, blue) light, infrared light and sometimes UV light. Radar satellites work in the microwave part of the spectrum. Images captured by satellite sensors are called **bands**.

Each object has its own **spectral signature**, as is depicted on an image below. Here we can see spectral signatures for water, soil and green vegetation. It shows which wavelengths are absorbed (low reflectance values) and which are reflected.



*Soil, water and green vegetation spectral signatures (**GrindGIS.com**)*

For example, if we look at the image above, we can see, that at wavelength around 2.1 μm, both soil and green vegetation are reflective, while water is not. If we look at a satellite band, that captures 2.1 μm wavelength, we will see soil as bright, since its reflectance is high (around 50 %), green vegetation as darker grey, since its reflectance is lower (around 20 %) , and water as black, since its reflectance is 0 %. At other wavelengths the image would be different, since soil, vegetation and water reflect different amounts at different wavelengths, as is evident from spectral signatures.
For example, if we were to look at a satellite band displaying reflectance at around 0.5 μm, differences between water, soil and vegetation would be barely noticeable, since they all have low and similar reflectance. If we wish to see the differences between them, it would thus be best to choose the wavelength (satellite band), where differences are higher. In general, every material type has its own spectral signature, meaning its reflectance at different wavelengths is variable. It is thus possible to see the differences between different types of land use and land cover, if we choose to observe the Earth using suitable bands.

To learn more about remote sensing basics, visit **this remote sensing tutorial.**

2.2 Band calculations using map algebra

Each satellite band is an image. An image is represented as a grid of values in computers (also called a **raster**). Every grid cell is called a **pixel** and is square shaped.

We can use map algebra to perform calculations between satellite bands. The way we do that is by calculating between the corresponding pixels . For example, on the graphics below, we can see how we would subtract one band from another. The value of a pixel in row 1 and column 1 of the first raster will be subtracted from the value of a pixel in row 1, column 1 of the second raster and so on.

Band 1				Band 2				Output band		
41	71	105	−	15	18	92	=	26	53	13
46	29	40		52	48	32		-6	-19	8
50	41	31		42	38	27		8	3	4

*Map algebra (**Humboldt State University**)*

We can add, subtract, multiply and divide bands with other bands or with scalar values.
As you will see later, we can use map algebra to calculate remote sensing indices, such as NDVI.

In the table below you can see bands for the Sentinel-2 satellite, along with the corresponding wavelengths and spatial resolutions. Since we will work with Sentinel-2 in this tutorial, this table will come in handy. If you want to work with other satellites, check out the bands for them **here.**

BAND		WAVELENGTH (min-max in micrometers)	SPATIAL RESOLUTION (meters)[1]
Band 1	Coastal Aerosol	0.421 – 0.457	60
Band 2	Blue	0.439 – 0.535	10
Band 3	Green	0.537 – 0.582	10
Band 4	Red	0.646 – 0.685	10
Band 5	Vegetation red edge	0.694 – 0.714	20
Band 6	Vegetation red edge	0.731 – 0.749	20
Band 7	Vegetation red edge	0.768 – 0.796	20
Band 8	NIR (near infrared)	0.767 – 0.908	10
Band 8a	NIR (near infrared)	0.848 – 0.881	20
Band 9	Narrow NIR	0.931 – 0.958	60
Band 10	Cirrus	1.338 – 1.414	60
Band 11	SWIR (Short wave infrared)	1.539 – 1.681	20
Band 12	SWIR (Short wave infrared)	2.072 – 2.312	20

2.3 Color

Next, we need to understand color. The human eye has 3 types of color receptors: red, blue and green. Computers use the RGB color model to represent colors. R in RGB stands for red, G for green and B for blue. We also call those **color channels**. Every color a computer can display, contains certain amounts of red, blue or green light. Values range from 0 to 255 for every channel, adding up to 16,7 million different colors for most modern computer screens.
For example, a "pure red" color will have a value 255 in the red channel, and 0 in both green and blue channels. If we add 255 into the blue channel instead of 0, red and blue colors will mix and we will get purple. The higher the channel value, the more of the color will be mixed in. If all three channels have equal values, we get a color in black and white range; white, if all three values are 255 and black, if they are all 0.

Visit this page and try to input different RGB values, to get a sense of how the colors mix.

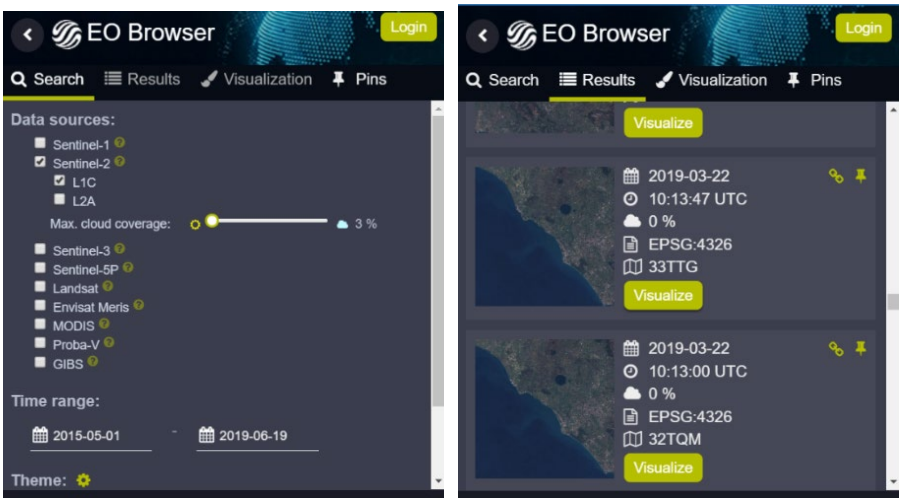
[1] Spatial resolution depends on the real world length, that one pixel represents. For example, if one pixel represents 30 meters in the real world, the resolution of an image will be 30 meters. Spatial resolution corresponds to how much information an image can convey, i.e. how precise or how small the objects it can display.

3 Your first custom script

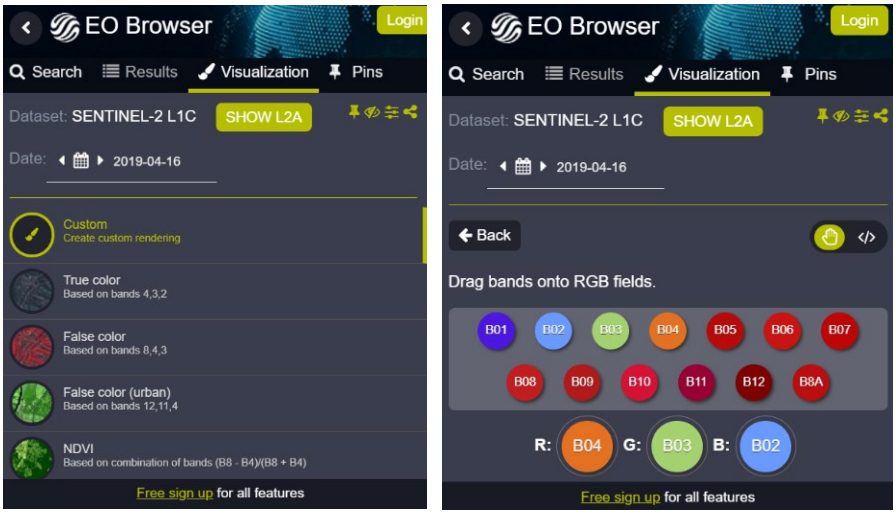
3.1 Create a composite in EO Browser

First, we will choose the data for the visualization. It is very easy, just follow the short steps below.

1. First, open **EO Browser**.
2. Next you can pan the window to your area of interest, but we recommend you choose vegetated areas, since we will be working with vegetation. You can also simply stay in Italy, as is default.
3. In the search tab, choose the data by selecting the Sentinel-2 satellite (leave the L1C option on), low cloud coverage (2-5 %) and a time range (we recommend you change the first date to a specific day in 2015), like on the image below (left).
4. Scroll down and confirm by clicking *search*.
5. Choose one of the results, that covers your area well and has low cloud coverage by selecting *visualize*, like on the image below (right).
6. For more information on how to use EO Browser, consult the **Sentinel Hub user guide**.

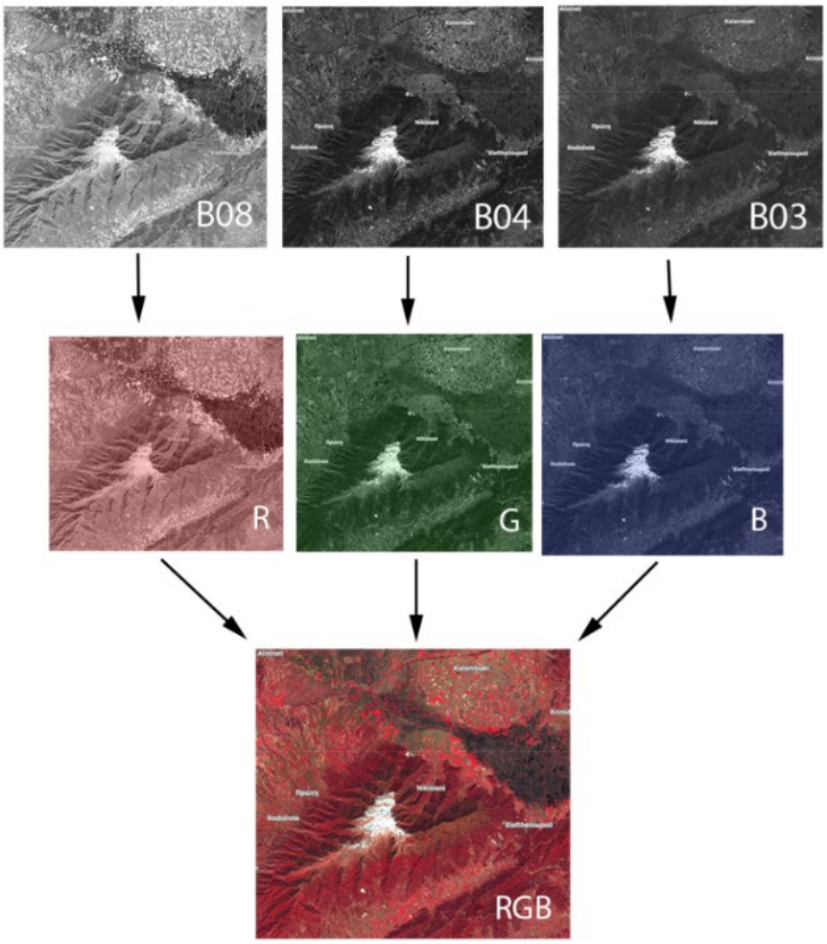


By default, the *True color* option is selected under the *Visualization* tab. Since we are interested in creating custom composites ourselves, choose the *Custom* option on top.
The custom visualization panel opens, where you can manually drag and drop satellite bands into the RGB channels. As you do that, the satellite image is updated automatically.



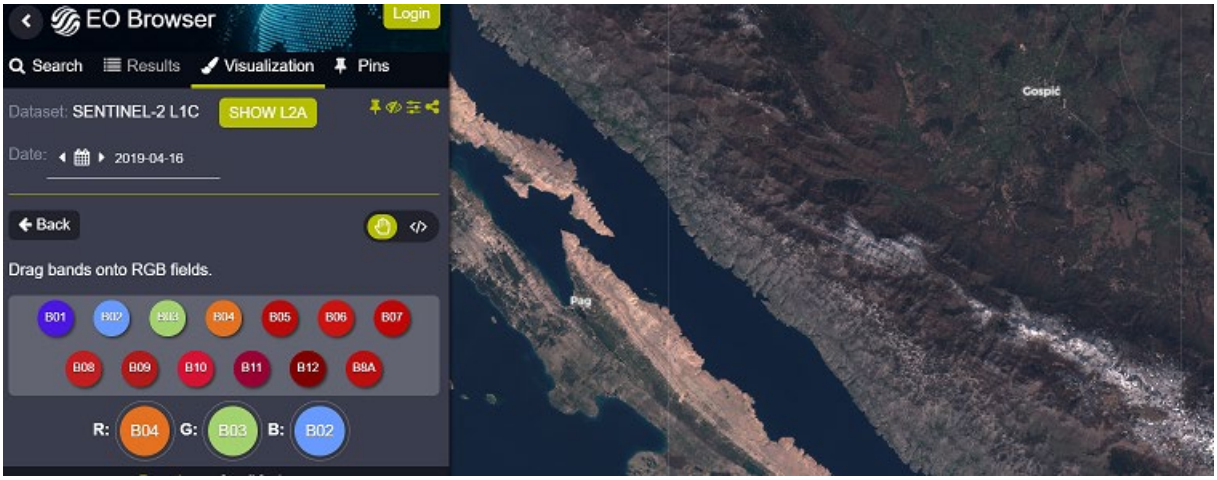
In EO Browser, Sentinel-2 bands are named with B and a number, e.g. B01, B08 or B12. If we look at a single satellite band, it will appear in the black and white color range, called ***grayscale***. Lighter values represent higher reflectance values of objects on the ground, while darker values represent lower radiance (the light sensed by the sensor) values.

To get a color image, we must create it by combining 3 bands by inputting them into the red (R), green (G) and blue (B) channels. This means, that the band we choose to input into the red channel will appear red on the image, the one in the green channel will appear green and the one in the blue channel will appear blue. The resulting image is called a ***composite***.

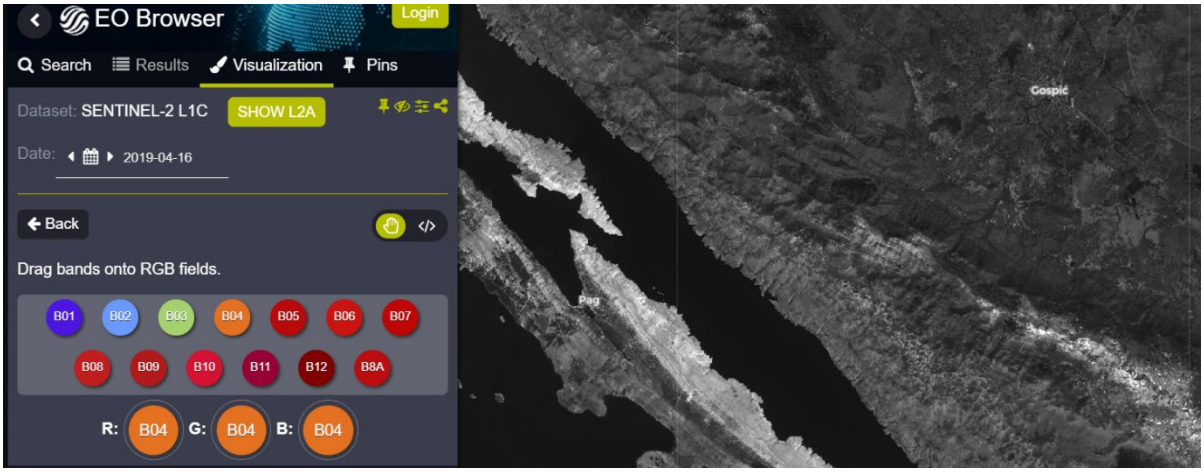


If we input a band, that displays reflectance of red wavelength (called a red band) into the red channel, green band into the green channel and blue band into the blue channel, we get an image, that corresponds to colors, as humans see naturally. In Sentinel-2, bands that correspond to red, green and blue light are B04, B03 and B02 respectively. Such composites are called ***true color*** composites.

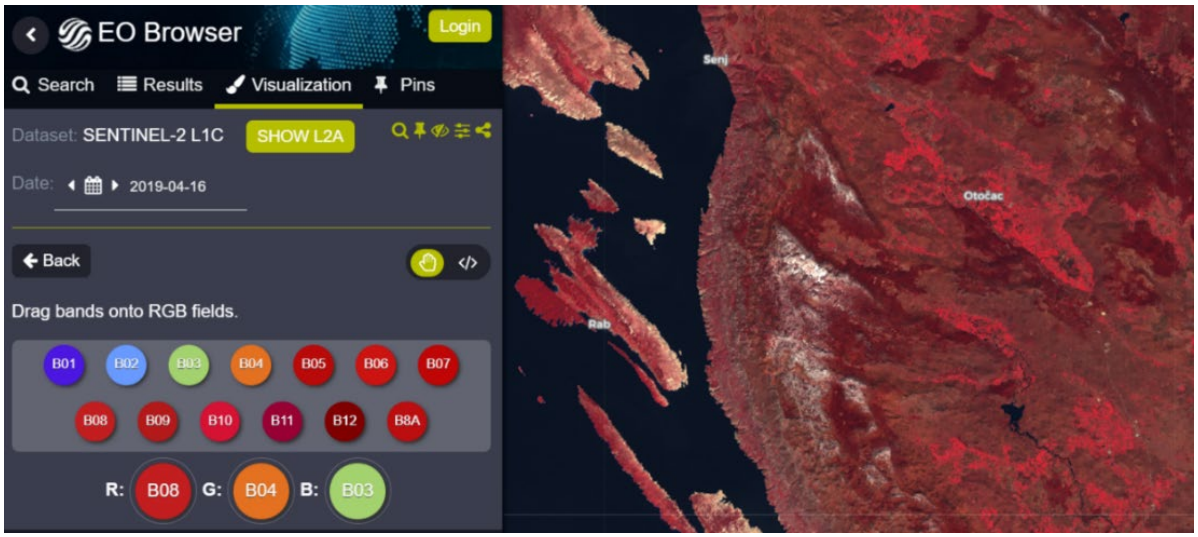
The image below is an example of a true color composite for Sentinel-2. Click on it to open it in EO Browser.



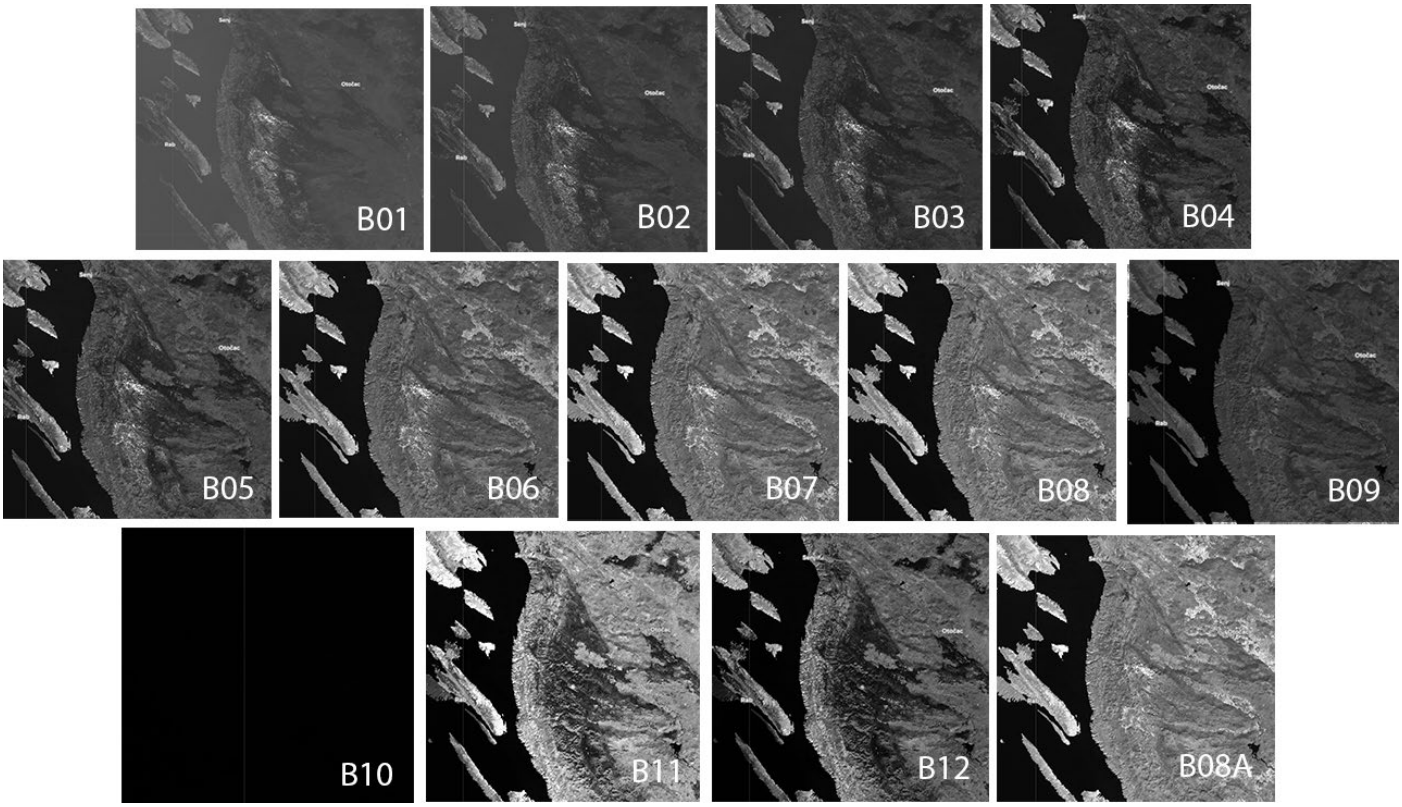
We can input the same band into multiple channels; in case we put the same band into all 3 color channels, we lose the RGB properties and get a grayscale image of the band, as is shown on the image below.



We can put any band in any one of the 3 channels. A composite, that is not a true color composite, is called a **false color composite.** For example, to create a composite, that highlights vegetation, we could input band 8 (near infrared) into the red channel, band 4 (red) into the green and band 3 (green) into the blue channel, as shown on the image below.



Since the image above is mostly red, it means, that the values in band 8 are higher than values in bands 4 and 3. If we were to input it into the blue channel, we would get a similar image, but with blue color. You can see the differences in band brightness in the images below. Compare for example B01 and B07.

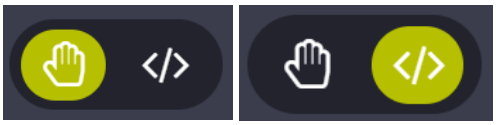


The reasons some bands are brighter than others are that bands have different ***bandwidths*** (band wavelength range from min to max), different spatial resolutions and that the amount of electromagnetic energy reflected from the Earth's surface and absorbed through the atmosphere varies with wavelength.

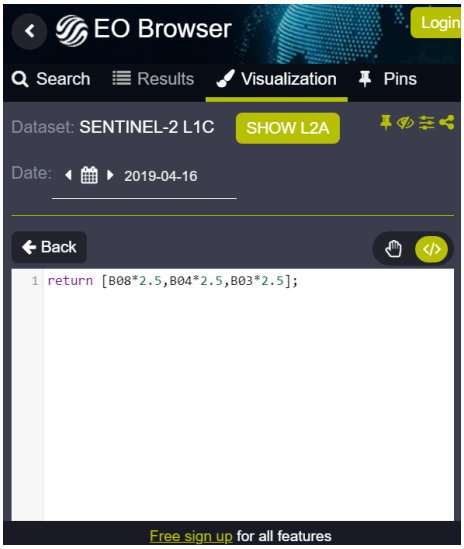
If we wanted to make one of the less dominant bands stronger, we could use custom scripts to multiply them.

3.2 Start exploring custom scripts

To access custom scripts, click on the green hand symbol under *custom*, to toggle between draggable bands and custom scripts (</>).



As the white javascript window opens, you can start changing the code. When you're done, scroll down and click *refresh*, since the image won't update automatically.



To make a simple true color composite, write the following code:

```
return [ B04, B03, B02 ]
```

The array of values in a return statement defines the colors for visualization. An array in javascript is a collection of elements, separated by a comma and contained within square brackets. The first number in the array will be used to visualize red color, the second number in the array to visualize green color and the third number to visualize blue color. We can define either one or three elements in the array.

If we use a single band 3 times, the result is the same, as if we only call it once – we get a grayscale image.

```
return [ B04, B04, B04 ]  // One band in all 3 RGB channels – grayscale image of the band
return [ B04 ]            // Same as above
```

Note, that we can use // to comment in javascript. Comments will not be executed. To create a multiline comment, we use / to start, and */ to end a comment.*

To manipulate the result, we can multiply the bands by numerical values. Multiplying each band by 2.5 proved to be useful for Sentinel-2 true color composites, as it improves the appearance of images. The true color composite we get by dragging and dropping bands is also multiplied, which you can check by looking at the script.

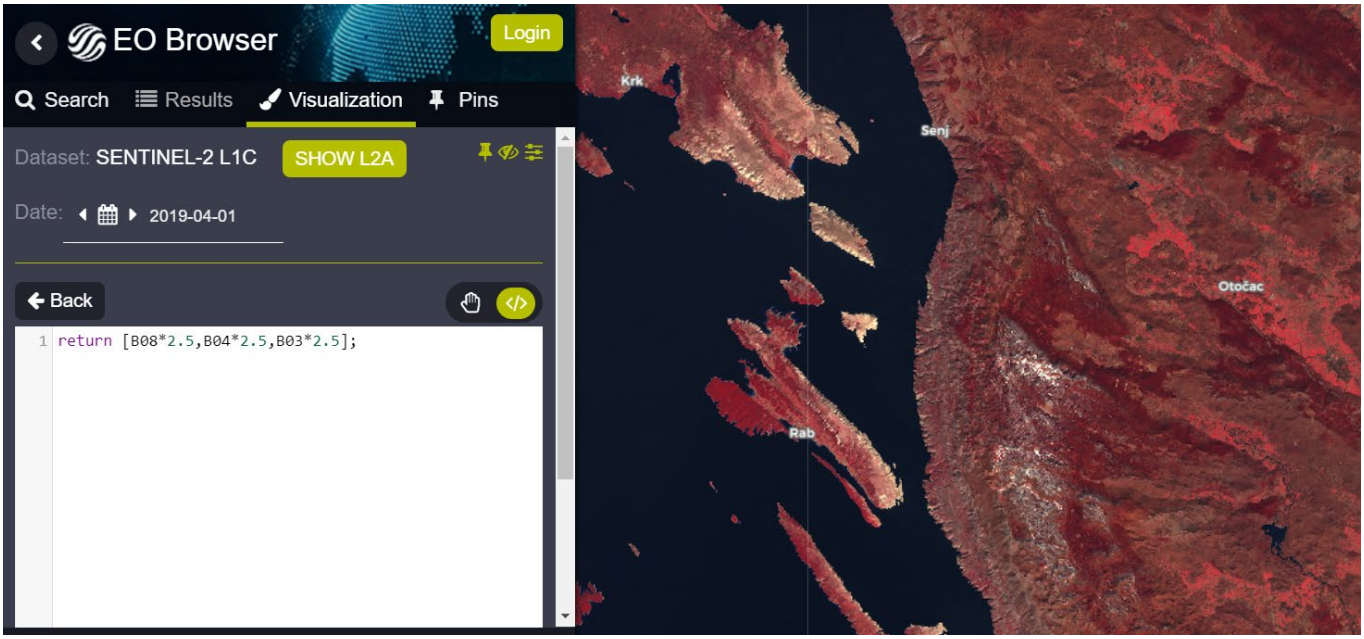
```
return [ B04 * 2.5, B03 * 2.5, B02 * 2.5 ]
```

Below is an example of a true color composite without the multiplication (left) and with the multiplication (right). Click on the images to open them in EO Browser.

Let's create two simple false color composites, where vegetation appears either red or green.

The common false color composite inputs the band 8, which displays vegetation, into the red (first) channel and bands 4 and 3, which have high values for non-vegetated areas, into the other two channels. The result is shown below. Vegetation is displayed in red, color darkness indicating depth of vegetation; deep red for forests and light red for grasslands. In white, brown and sandy colors we see non-vegetated areas, such as rock, bare soil or snow.

```
return [B08 * 2.5, B04 * 2.5, B03 * 2.5] // Vegetation in red
```



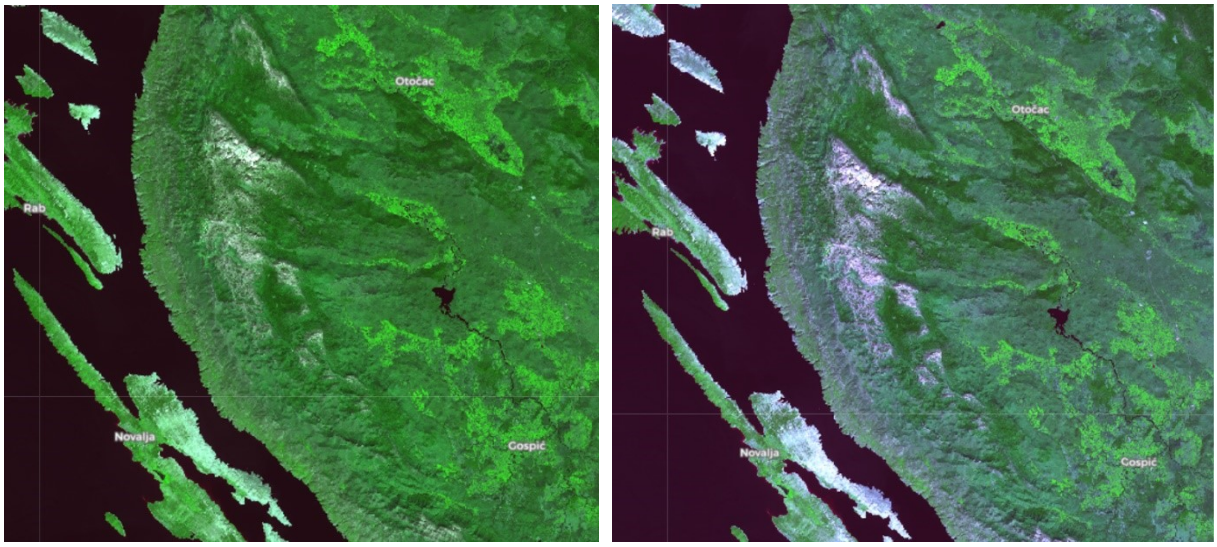
To instead show vegetation in green, we should input band 8 into the green (second) channel.

However, the result (on the left below) is not as clear as before, since non-vegetated areas appear bright green. We should tweak the values to show bare ground more clearly.

We can choose to multiply each channel differently, to manipulate the brightness of each color. We will multiply bands 3 and 4, to make them brighter. Bare ground areas now appear bluish-purple, since we input higher values to red and blue channels, and the image is clearer (image on the right below).

```
return [B03 * 2.5, B08 * 2.5, B04* 2.5]  // Vegetation in green

return [B03 * 3.5, B08 * 2.5, B04* 4.5]  // Vegetation in green, tweaked values
```



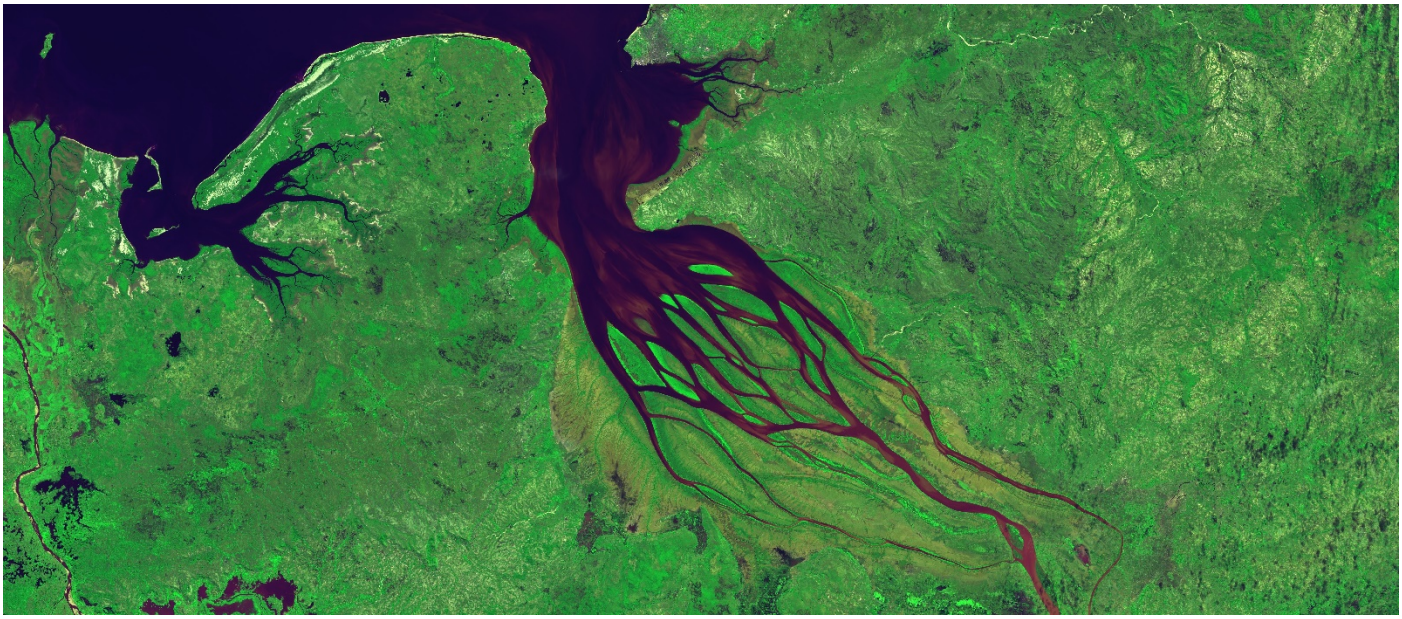
3.3 Visualize the Betsiboka river delta

Let's try out a fun challenge. We will visualize the Madagascar Betsiboka river delta, in an informative and appealing way.

Below is the true color composite, we will try to enhance. Click on the image below to open it in EO Browser.

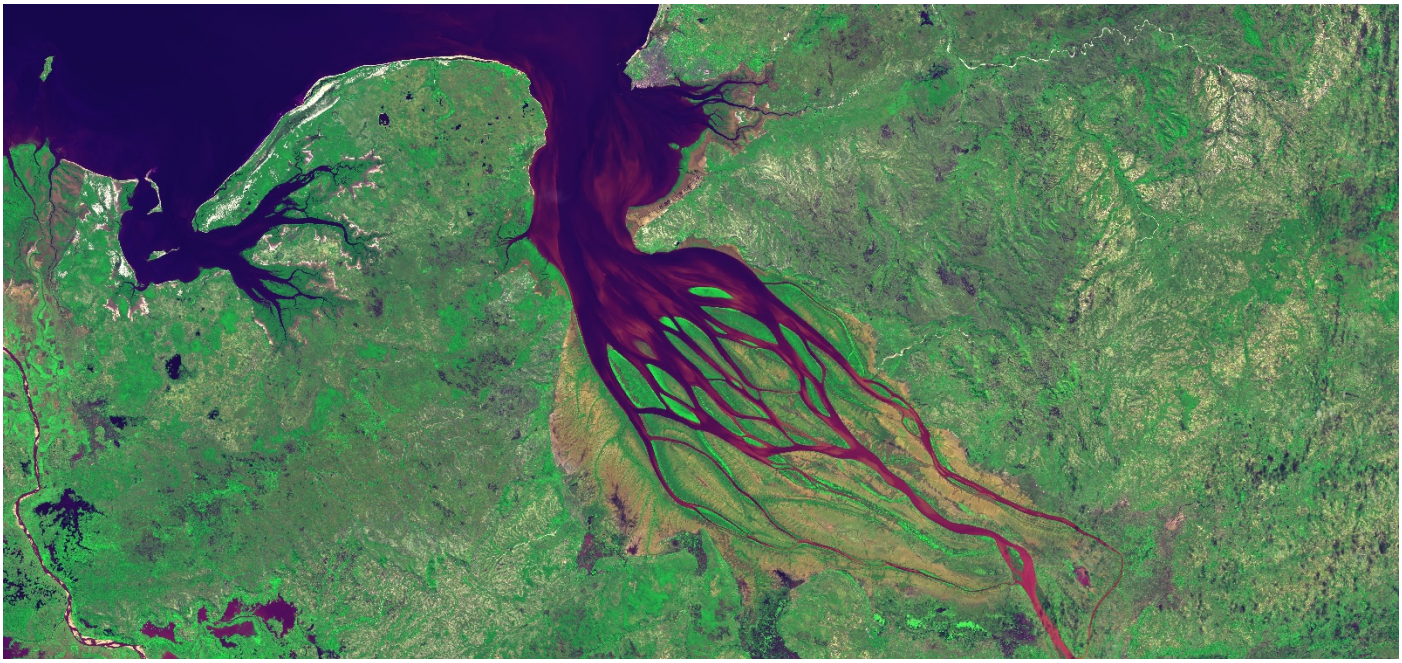
Suppose we want to separate vegetation from non-vegetation and clearly show sedimentation in the river. To achieve that, we can input band 8 (vegetation) into the green channel, band 4 (bare ground) into the red channel and band 2 (which is used for deep water analysis) into the blue channel. We also multiply them by 2.5.

```
return [B04 * 2.5, B08 * 2.5, B02 * 2.5]
```



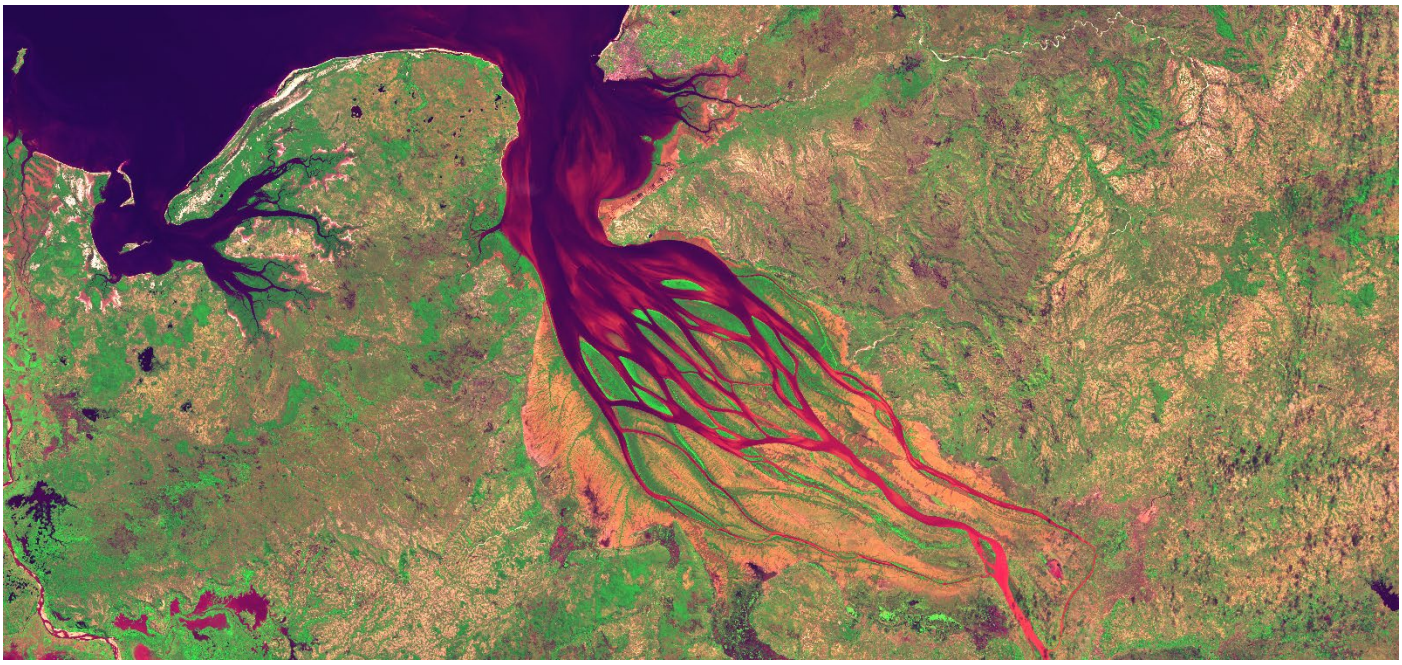
We can see, that band 8 is very strong and we can't see sedimentation or non-vegetated areas well. Let's try to increase the brightness of the red and blue channels, by multiplying them by 3.5.

```
return [B04 * 3.5, B08 * 2.5, B02 * 3.5]
```



Although the result we got is informative, the image would be more expressive, if we increased the red values further. This is now only a matter of aesthetics and experimentation. Let's increase the value of the red channel to 5.5.

```
return [B04 * 5.5, B08 * 2.5, B02 * 3.5]
```



We have created a composite, that displays deeper water in dark purple, sedimentation and shallow waters in red, bare ground in orange and pink and vegetation in green. You can continue tweaking the values, if you'd like.

Congratulations! You have created your first custom script! Feel free to experiment with different bands and RGB values, to get a sense for how they work together.

4 Implement a remote sensing index

Let's try to do something even more useful. We will use map algebra to calculate the NDVI index based on two bands. First, we assign our calculation to the variable, using **let**, which literally means, *let something be…* The variable we are assigning our index to can have any name we want, but can only be a single word (excluding numbers, spaces and signs). Here, we name it **value**. Then we set it equal to (=) and write our calculation.
Next, we simply call our variable **value** inside a **return** array. This way we get a grayscale result.

*Note that in Javascript the equal sign (=) implies, that we are assigning a value. It does not imply that the two are identical. If we want to check if **val** is identical to, let say 2, we would use double equal sign (==).*

The simple example of a band combination would be to add two bands:

```
let value = B11 + B02;

return [value]
```

We can input any calculation we like. For example, we will calculate the NDVI index, which shows vegetation health:

$$\text{NDVI} = \frac{(NIR-RED)}{(NIR+RED)}$$

NIR stands for "near infrared" and for Sentinel-2 it corresponds to band 8, which shows vegetation best and "RED" corresponds to band 4.
An NDVI index would thus look like this:

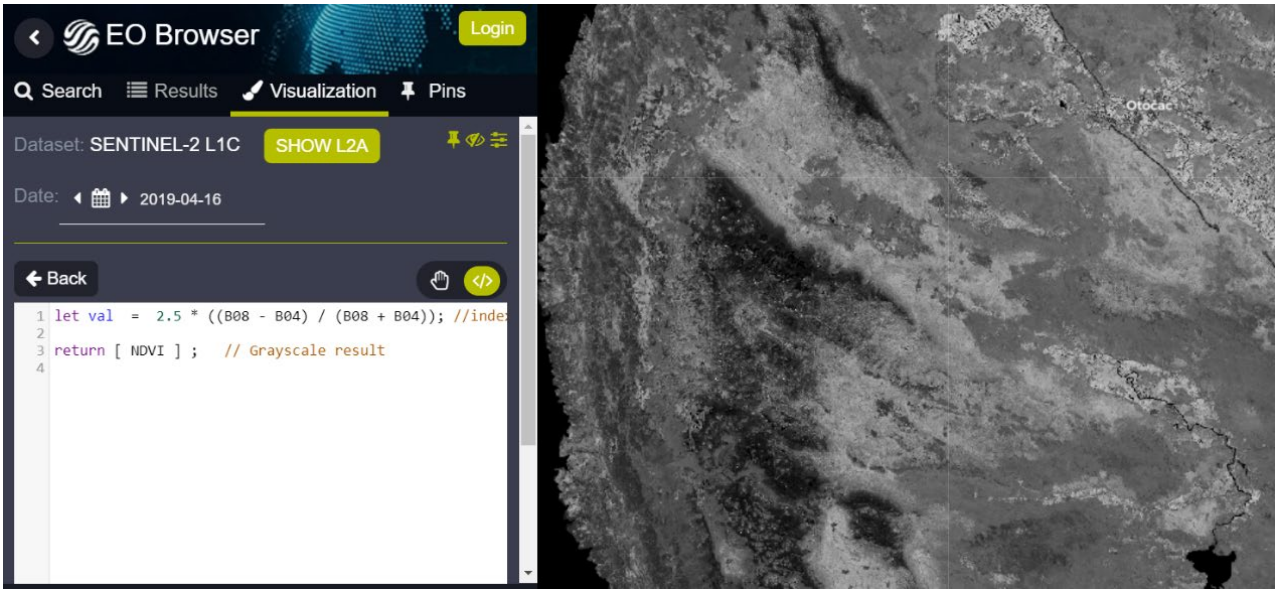
$$\text{NDVI} = \frac{(B08-B04)}{(B08+B04)}$$

We can simply input the calculation into our script (we multiply it by 2.5 to increase the brightness) as follows:

```
let NDVI  =  2.5 * ((B08 - B04) / (B08 + B04));

return [ NDVI ]   // Grayscale result
```

Click on the image below to open it in EO Browser:



We can visualize most indices this way. Check out **this webpage** for a list of indices, you could use.

Many indices share the structure of dividing an addition of two bands from the difference of two bands, as is shown in the equation below, where A and B represent two satellite bands.

$$\text{Index} = \frac{(A-B)}{(A+B)}$$

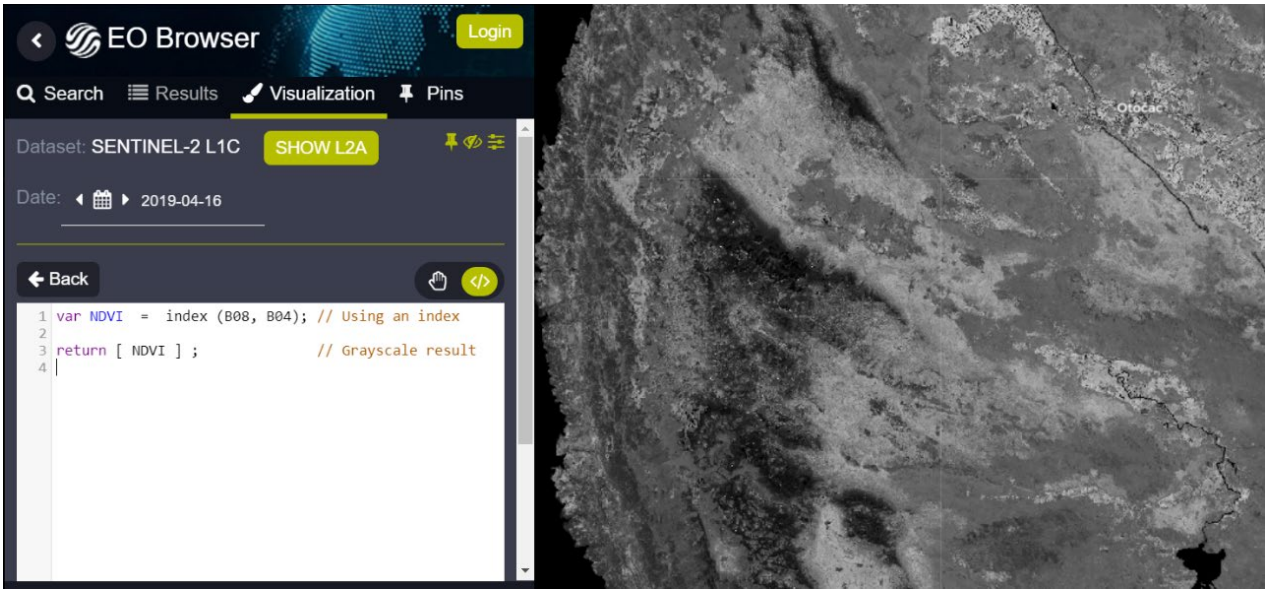
For this type of index, you can use a Sentinel Hub inbuilt function, that simplifies the process. We could simply calculate NDVI like in the previous example, but it would be simpler to use the index function, since it is used precisely for an index like this.

To use the index function, we should define a variable as an index function of the 2 bands we wish to use.

```
let NDVI = index (B08, B04); // Using an index function

return [ NDVI ]              // Grayscale result
```

When we define variables, we can also use **var** instead of **let** (see **here** to understand the difference).



For now, we returned the result in grayscale. When we calculate an index, we have to consider how to best visualize it.

We need to use a color scale, which will help us correctly interpret NDVI (or any other data). It will help us distinguish between low and high values, understand how the values of NDVI change spatially and through time, and emphasize values of special importance. For this, we need to learn new methods for displaying color.

5 Custom color scales

5.1 Discrete color scale

The easiest way to create custom color classes is by using if **statements**.
In Javascript, we can use an if statement to specify the conditions for something to happen. In our case, we want to decide which value ranges will get which colors. This way we create discrete classes, each of its own color. For example, if we know that certain values represent forest and others urban areas, we can create classes, to display each of them in a separate color.

An if statement has the following structure:

```
If (this is true) {
  that should happen
}
```

In the brackets after an if, we specify the condition. Then we wrap our command into **curly brackets**.

For example:

```
if ( NDVI < 0.2 ) {
  return [ 0, 1, 0 ]
}
```

The range of RGB color values here is 0 – 1, where the [0, 0, 0] array represents black color and the [1, 1, 1] array represents white color.
The above if statement will use green color ([0, 1, 0]) to visualize pixels with NDVI values less than 0.2.

Instead of smaller (<), we can use any other **JavaScript logical operator.**

We can specify multiple if conditions. If we want to specify a command in case if is not true, for example for all the values that do not equal 0.2, we could follow the if statement with an else statement. It is important, that the classes do not overlap.

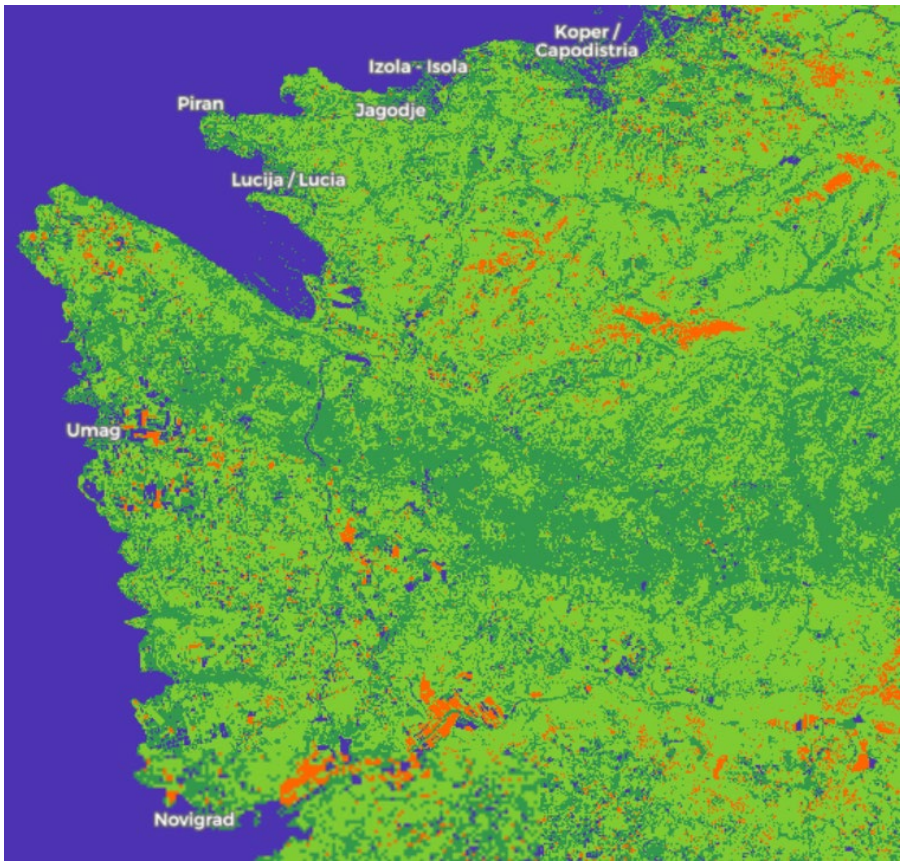
```
if (NDVI < 0.2) {
  return [0.3, 0.3, 0.3]
}
else {
  return [0.9, 0.2, 0.2]
}
```

For example, here is an NDVI visualization, using if statements:

```
var NDVI = index (B08, B04); // calculate the index

if (NDVI < 0.2) {
  return [0.3, 0.2, 0.7]
}
if (NDVI < 0.5) {
  return [0.2, 0.6, 0.3]
}
if (NDVI < 0.7) {
  return [0.5, 0.8, 0.2]
}
else {
  return [1, 0.4, 0]
}
```

On the resulting image below we can see, that we have successfully classified NDVI values into 4 custom classes. Click on the image to open it in EO Browser.



5.2 Continuous color scale

What if we wanted to display index values continuously? In that case, we can use the Sentinel Hub colorBlend function to display continuous color between the chosen values.
ColorBlend will map every pixel value of an index or a band to a new scale between the chosen minimum and maximum input values. In the second step it will map those values to the colors. For values which are not explicitly defined, it will linearly interpolate colors, which will result in a continuous color scale. This will become clearer as we explore the function structure and examples.

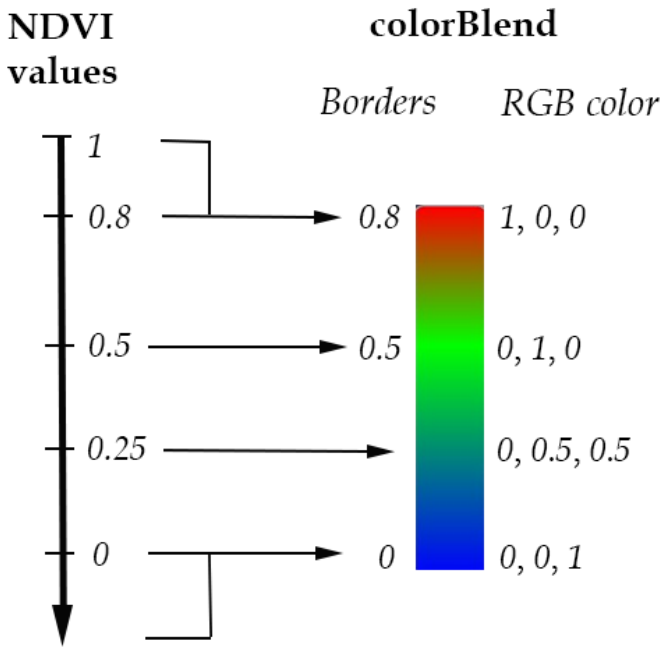
The colorBlend function structure is:

colorBlend(pixel value[2] [**Array of border values**] [[**RGB for border value 1**] [**RGB for border value 2**]… [**RGB for border value n**]])

We will try the colorBlend function on the NDVI index. First, we calculate an index, like before. Then we call the colorBlend function. All the parameters of the function are in brackets and separated with a comma.

```
var NDVI = index (B08, B04); // calculate the index

return colorBlend   // call the colorBlend function
(NDVI,              // Pixel value
  [0, 0.5, 1],      // Define the borders
  [ [0, 0, 1],      // Define the RGB colors for each border
    [0, 1, 0],
    [1, 0, 0],
  ]);
```



[2] **E.g., NDVI index or band 1.**

The first input for the function is the pixel we want to visualize. In our case, it is the NDVI index, but it could also be a band, for example.

The second parameter is an array of border values (the green brackets), which will be visualized with colors specified in the third input to the function. We are only using 3 border values. The first border value represents the minimum value and, in our case, equals 0. Index values smaller than 0 will be represented with the same color as the border value 0. The second border value equals 0.5 and the third border value, the maximum, equals 1. Index values greater than 1 will be represented with the same color as the border value 1.

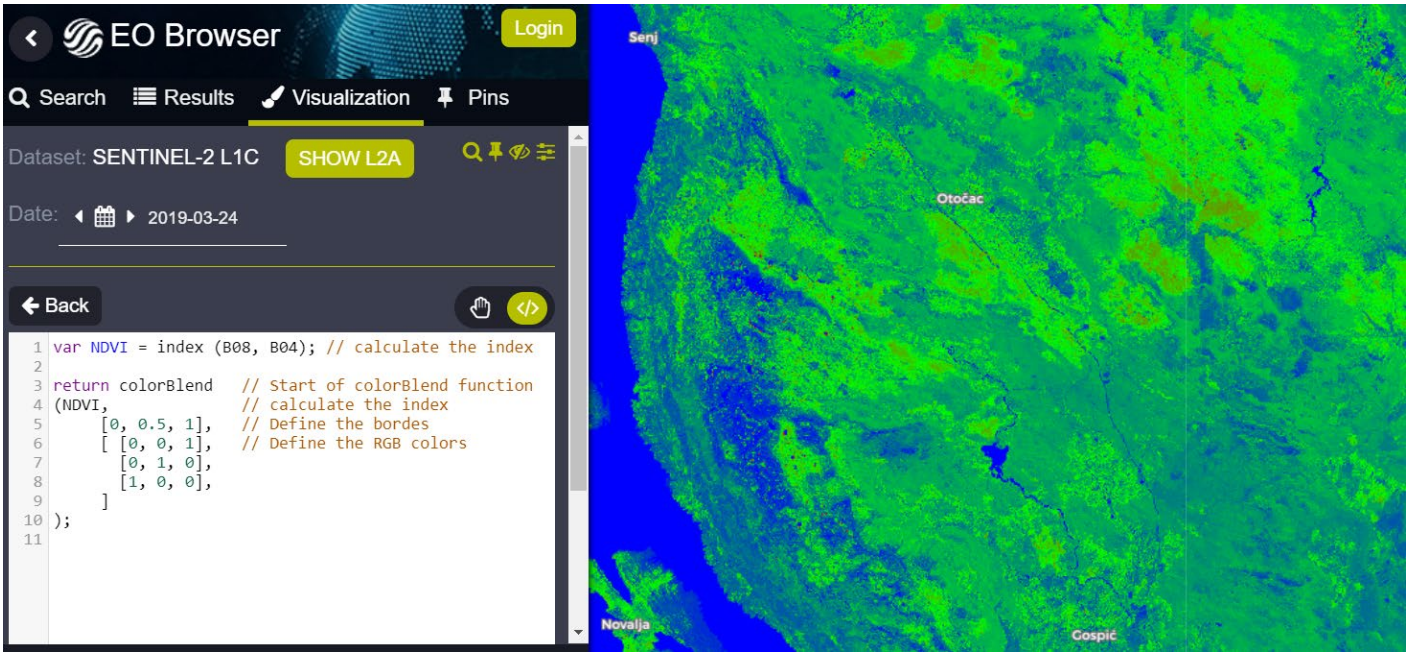
The size of the green array (how many elements it has) defines the number of colors we need to specify next. Since we have 3 border values, we need to specify three colors.

The third parameter is an array of colors (the blue array). Each sub-array (the pink arrays) contains RGB color values for the border, to which the color is assigned. Here, we made a very simple color selection, where the first border will appear blue, the second will appear green and the third will appear red.

All the pixel values smaller or equal to 0 will appear blue, all values equal to 0.5 will become green and all values grater or equal to 1 will appear red.

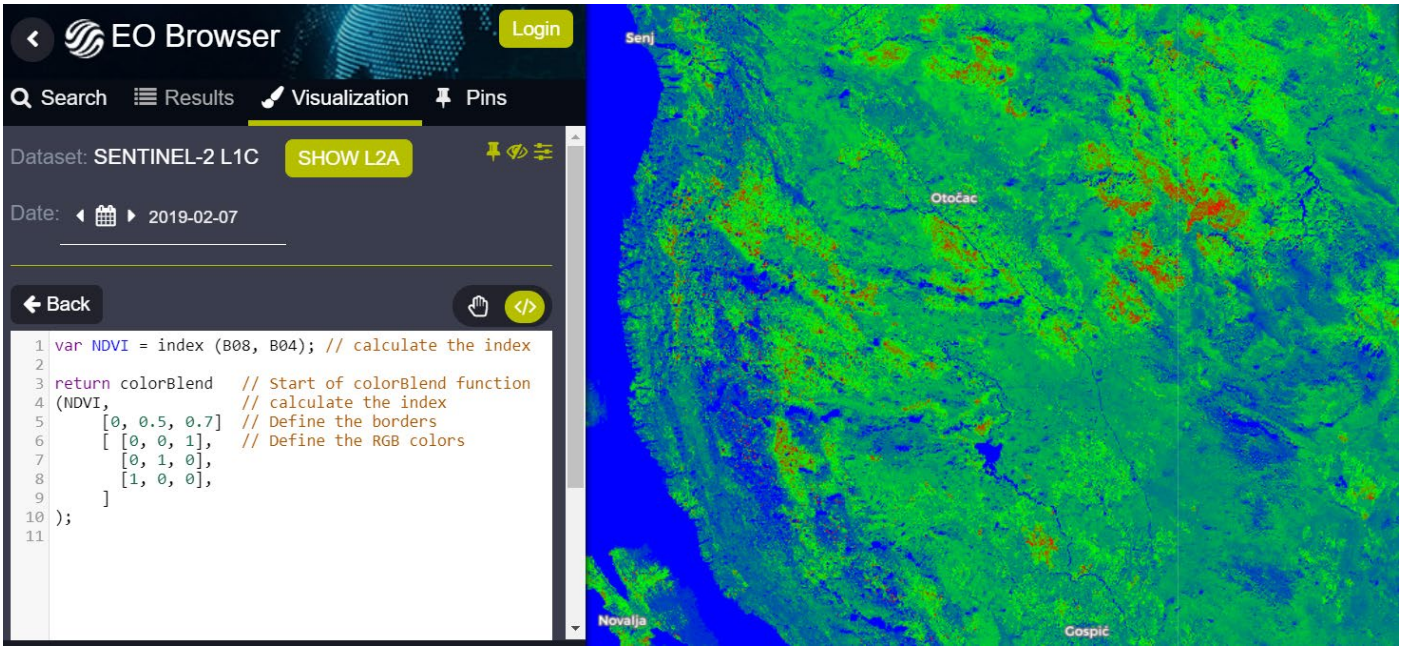
For all pixel values in between, colorBlend will interpolate the color between neighboring colors. For example, to the NDVI pixel value 0.2, colorBlend will assign a color [0, 0.4, 0.6], which is linearly interpolated between blue ([0, 0, 1] for the minimum border 0) and green ([0, 1, 0] for the second border 0.5).

Similarly, to the pixel value of 0.7, for example, colorBlend will assign a color [0.4, 0.6, 0]. This way we get a continuous color scale. Click on the image below to open it in EO Browser.



On the image above we can see the blue areas, where the NDVI index is 0, or the values were negative. There are apparently few values, where the NDVI would be 1, which is expected. There are also few high values, since we don't see much color between red and green at all. However, we can clearly see the NDVI values continuously.

If we would like to see more high values, we should change the maximum (upper) border to something a bit lower than 1, for example 0.7. What will happen is that the difference between green and red will be smaller, thus more values will appear red. We can see the result on an image below.



How we should choose the borders depends on the histogram of our NDVI for a specific area. Where are the majority of values located? On the lower end, or on the upper? Are they distributed normally? This is difficult to set properly just from experimenting and looking at the result. It would we best to calculate a histogram and set borders accordingly.
For example, you may have to set the border values as [0, 0.1, 0.2, 0.3] for the vegetation index in a low vegetation area, such as Australia for example. Otherwise, you will not see much color variation. For Slovenia on the other hand, which has lots of high vegetation areas, you may set the border values as [0, 0.5, 0.7, 0.9].

You should set the border values as fits your data best. For example, we can't set the border values for the NDVI index as ranging from -50 to 50, since the NDVI values range from -1 to 1 and the colorBlend function would "remove" most index values (all values lower than 0 and all higher than 1). If your data values range from -20 to 20, for example, you should use similar border minimum and maximum to display the whole range of values [-20, 20]. If you would only like to display positive values or values above 10, you should set the minimum and maximum as [0, 20] or as [10, 20].

You can also set as many intermediate border values as you would like, such as [0, 0.2, 0.4, 0.6, 0.8, 1], but you have to also assign them additional color arrays below, so that each border has its own RGB array.

A colorBlend function is implemented in Sentinel Hub and is not a general JavaScript function. It can thus be used and found only within Sentinel Hub. Check its documentation **here**.
Other functions implemented in Sentinel Hub, which can be used in custom scripts are described **here.**

When using colorBlend, be careful not to forget commas between the array elements. Also note, that letter case is important. Colorblend won't work; only colorBlend will.

6 Concluding remarks

Congratulations! You have finished the basic custom script tutorial!

You have learned how to:

- use EO Browser to create custom color composites,
- use multiplication to manipulate the color brightness,
- choose the bands to display the phenomena of interest,
- implement a remote sensing index,
- create custom discrete and continuous color scales.

You should take some time to practice and experiment. Try out different values and bands. Note that custom scripts can be used for scientific, as well as aesthetic purposes. Think about whether your script is useful to you.

If you would like to learn more, visit the **Sentinel Hub Evalscript documentation**.

Start creating new, useful and beautiful Earth visualizations, and have fun!

Agricultural fields of Italy (modified Copernicus Sentinel-2 data processed by Sentinel Hub, acquired on 16.4.2019). ***Inspect in EO Browser.***

7 References

Colors RGB. (n.d.). Retrieved 7 4, 2019, from w3schools.com: https://www.w3schools.com/colors/colors_rgb.asp

Index DataBase. (n.d.). Retrieved 7 1, 2019, from IndexDataBase.com: https://www.indexdatabase.de/db/i.php

JavaScript Comparison and Logical Operators. (n.d.). Retrieved 7 6, 2019, from w3schools.com: https://www.w3schools.com/js/js_comparisons.asp

JavaScript Let. (n.d.). Retrieved 7 8, 2019, from w3schools.com: https://www.w3schools.com/js/js_let.asp

Raster Analysis and Raster Math. (n.d.). Retrieved 7 3, 2019, from Humboldt State University: http://gsp.humboldt.edu/OLM/Courses/GSP_216_Online/lesson5-2/raster-calc.html

Remote sensing tutorials. (n.d.). Retrieved 7 5, 2019, from Natural resources Canada: https://www.nrcan.gc.ca/maps-tools-and-publications/satellite-imagery-and-air-photos/tutorial-fundamentals-remote-sensing/9309

Sentinel 2 L2A. (n.d.). Retrieved 7 6, 2019, from Sinergise.com: https://docs.sentinel-hub.com/api/latest/#/API/data/Sentinel-2-L2A?id=pixel-properties

Sentinel Hub documentation. (2019, 7 6). Retrieved from Sinergise.com: https://docs.sentinel-hub.com/api/latest/#/

Sentinel Hub User Guide. (n.d.). Retrieved 6 25, 2019, from Sinergise.com: https://www.sentinel-hub.com/explore/eobrowser/user-guide

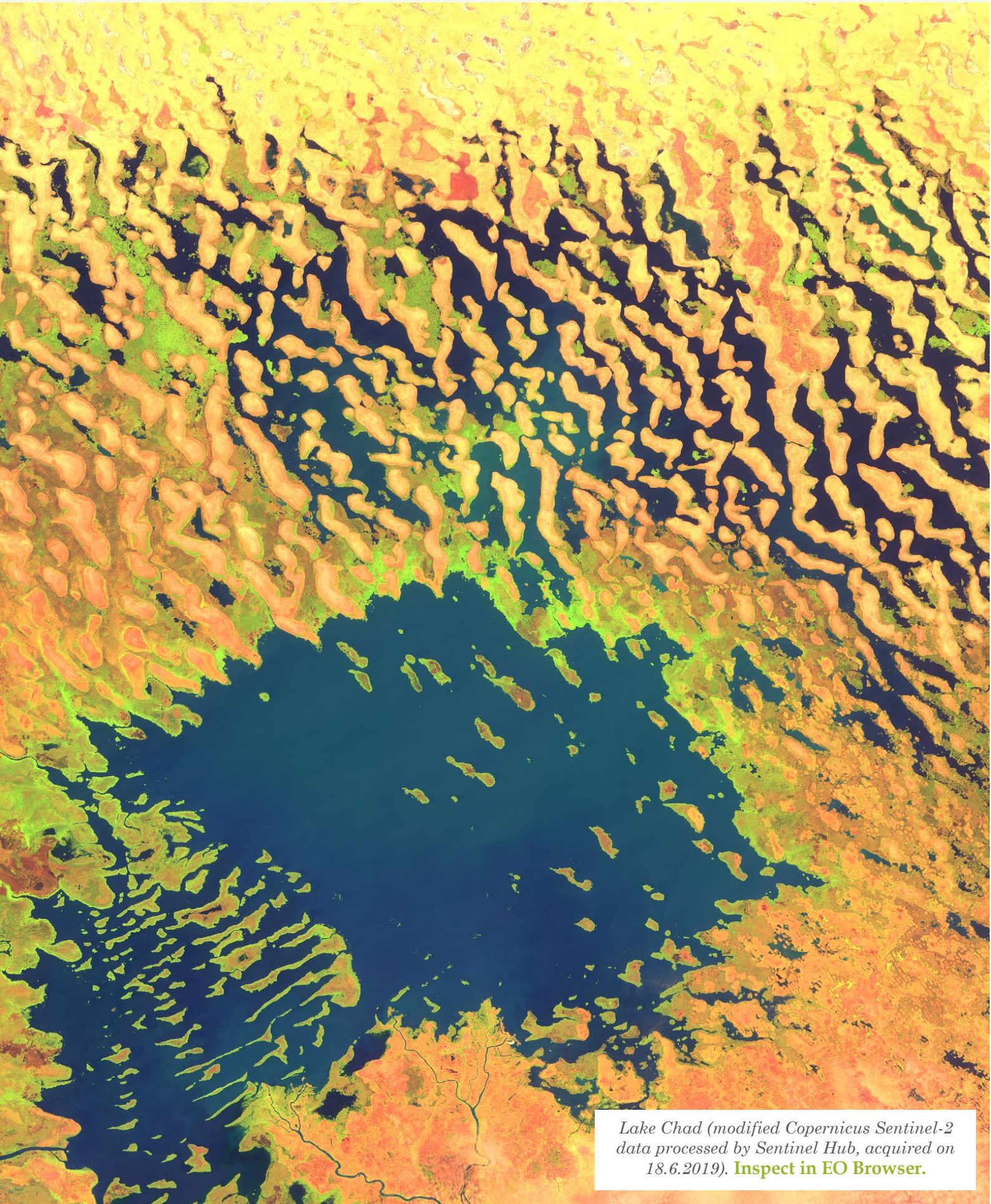
Sentinel Playground. (2019, 7 4). Retrieved from Sinergise.com: https://sentinel-hub.com/explore/sentinel-playground

The Electromagnetic Spectrum. (n.d.). Retrieved 7 8, 2019, from Mini Physics: https://www.miniphysics.com/electromagnetic-spectrum_25.html

Vegetation Spectral Signature Cheat Sheet. (n.d.). Retrieved 7 8, 2019, from Grind GIS: easy to learn GIS, love geography.

SINERGISE

SENTINEL Hub



Lake Chad (modified Copernicus Sentinel-2 data processed by Sentinel Hub, acquired on 18.6.2019). **Inspect in EO Browser.**