
Digital Earth Africa Training

Release 0.1

Digital Earth Africa

Jun 01, 2021

CONTENTS

1 Session 1: Introduction	3
1.1 What is Digital Earth Africa?	3
1.2 Create a Digital Earth Africa account	4
1.3 Navigating the Sandbox	7
1.4 Running a Notebook	9
1.5 Session 1 Quiz	19
2 Session 2: Datasets	21
2.1 Digital Earth Africa products	21
2.2 Digital Earth Africa Map	24
2.3 Data availability	32
2.4 Loading data in the Sandbox	38
2.5 Session 2 Quiz and Solution	51
3 Session 3: Composites	53
3.1 Introduction to cloud-free composites	53
3.2 Cloud masking with <code>load_ard()</code>	59
3.3 Create a geomedian composite	63
3.4 Session 3 Quiz and Solution	67
4 Session 4: Indices	69
4.1 Band indices	69
4.2 Calculating NDVI: Part 1	73
4.3 Calculating NDVI: Part 2	78
4.4 Session 4 Quiz and Solution	84
5 Session 5: Vegetation Analyses	85
5.1 Building a case study: vegetation analysis	85
5.2 Exercise: Vegetation change detection	88
5.3 Session 5 Quiz and Solution	95
6 Session 6: Water Analyses	97
6.1 Introduction to water analyses	97
6.2 Exercise: Determining the extent of water bodies	100
6.3 Session 6 Quiz and Solution	108
7 Course conclusion & wrap-up	109
7.1 Congratulations! You have reached the end of the Digital Earth Africa training course.	109
7.2 What next?	110
7.3 Keep in contact	112

8 Extra session: Python basics	113
8.1 Python basics 1: Jupyter	113
8.2 Python basics 2: Numpy	116
8.3 Python basics 3: Matplotlib	122
8.4 Python basics 4: Cleaning data	127
8.5 Python basics 5: Xarray	132
8.6 Exercise solutions	141
9 Latest Updates	153
9.1 Contact us	153
9.2 Frequently asked questions	154
9.3 Download course content	160
9.4 Events	160
9.5 Sandbox help documentation	160
9.6 Quiz index	164
9.7 Geomedian widget	165
9.8 Map portal user guide	166
9.9 GIS web services	172
9.10 Import external datasets	177
9.11 License information	182

Note: As of June 2021, DE Africa Landsat data has been upgraded to Collection 2. Datacube names have been updated to `ls5_sr`, `ls7_sr` and `ls8_sr`. Deprecated naming conventions such as `ls8_usgs_sr_scene` will no longer work. For more information on Landsat Collection 2, visit the [DE Africa Landsat documentation](#). Additionally, the `deafrica_tools` package has replaced the deprecated `sys.path.append('../Scripts')` file import. For more information on `deafrica_tools`, visit the [DE Africa Tools module documentation](#).

Welcome to the Digital Earth Africa training course!

This is a 6-week, self-paced online course. It should take around 2 hours per week to complete. Participants who successfully complete the course will receive a Certification of Completion at the end of training.

To get started, watch the short introductory video below. Then click **Session 1: Introduction** to begin. Other sessions will be added as the course progresses.

Should you require help, see the [Frequently asked questions](#) and [Contact us](#) pages.

<https://youtu.be/-jGkL9kx6rg>

**CHAPTER
ONE**

SESSION 1: INTRODUCTION

Welcome to the first session of the Digital Earth Africa training course!

This session introduces the Digital Earth Africa program and shows you how to sign up to the Digital Earth Africa Sandbox, our data analysis platform. We will then walk through an exercise exploring crop health data.

It is not required to have previous experience in Earth observation data or Python coding. Every section includes instructions and screenshots on how to perform each task.

To get started, click on **What is Digital Earth Africa?** or select the **Next** button below.

1.1 What is Digital Earth Africa?

1.1.1 Introduction

Africa is a continent with a rich and diverse environment that faces many challenges such as ready access to drinking water, rapid urban development, active deforestation and food insecurity. Our understanding of these challenges connects directly to how well we understand its natural resources, and the human and climate impact on them.

The Digital Earth Africa (DE Africa) program aims to increase this understanding through three main tenets:

- **Explore:** Allow access to satellite data in ways that address Africa's needs
- **Learn:** Provide avenues for learning about Earth observation technology and its significance for policymaking and improving the lives of Africans
- **Engage:** Stimulate participation and collaboration in events that will help implement, scale and sustain the program

As such, Digital Earth Africa is a platform designed to:

- Catalogue large amounts of Earth observation data
- Allow users the ability to easily perform exploratory data analysis
- Allow scalable continent-wide processing of stored data
- Monitor contained data for quality control and updates

For more details on the Digital Earth Africa program, visit the [Digital Earth Africa homepage](#). If you would like to learn more about Earth observations in general, take a look at [this summary document](#) by the OECD.

1.2 Create a Digital Earth Africa account

1.2.1 Overview

The Digital Earth Africa Sandbox is a learning and analysis environment for accessing Digital Earth Africa data. During this course, you will learn how to use this environment to write and run analyses using Earth observation data.

The Sandbox is free to use. Watch the video for a quick introduction. Then follow the step-by-step instructions below to create an account.

If you have an existing account, you can sign in at <https://sandbox.digitalearth.africa/>.

1.2.2 Video: Signing up to the Sandbox

This video contains a short introduction to the Sandbox. After watching the video, follow the written steps below for detailed instructions on how to sign up.

<https://youtu.be/ZPml2s-VoWY>

1.2.3 Create an account

1. Visit <https://sandbox.digitalearth.africa/> and click **Login or Sign up**.
2. Click **Sign up**.
3. Fill in the **Username**, **Name**, **Email** and **Password** fields. Passwords must be at least 8 characters, and contain at minimum a lower case letter, an upper case letter and a number. Record your username securely as it will be required to reset your password should you forget it.



Sign up with a new account

Username

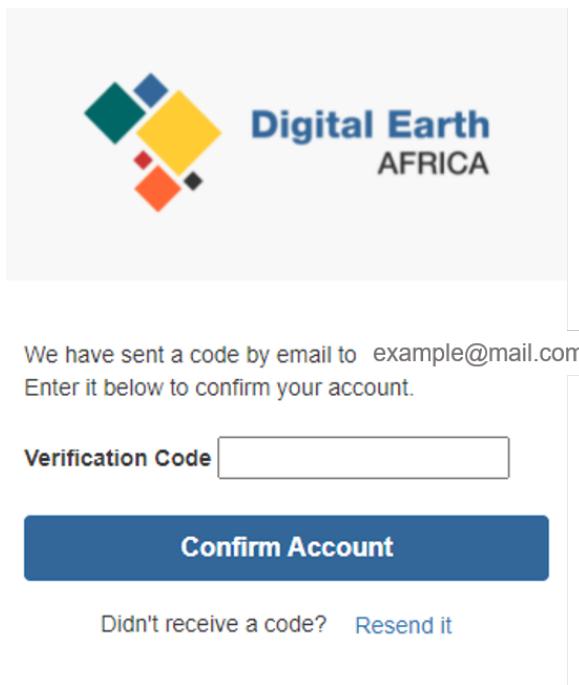
Name

Email

Password

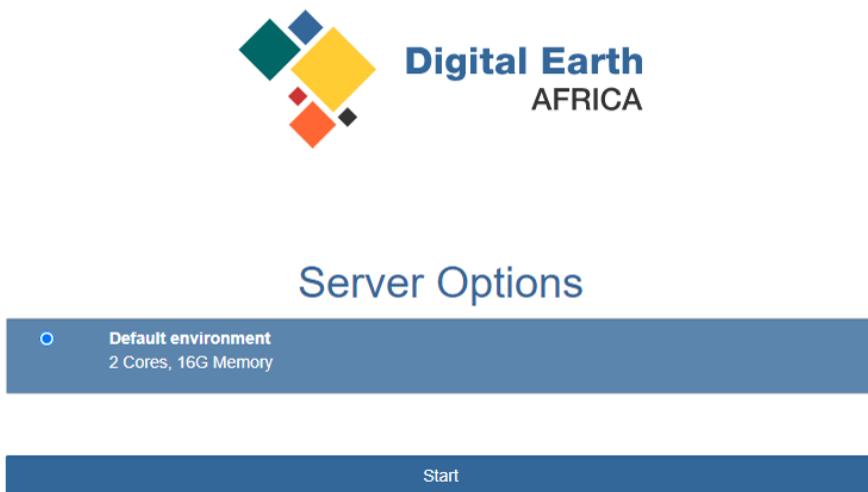
Sign up

4. Click **Sign up**.
5. A verification code will have been sent to your nominated email address. Enter the verification code and click **Confirm Account**. This will automatically log you in.



Note: If you have not received the verification code email, please check your spam or junk mail folders.

6. You will be prompted to select a sever from Server Options. For regular users, only the default environment is available. Select **Default environment** and click **Start**.



The server will start up; this can take a few minutes. You will automatically be redirected to the Sandbox environment. The Sandbox runs completely in-browser and all necessary software is provided. No additional installation or configuration is required.

1.2.4 Conclusion

You now have a user account on the Digital Africa Sandbox. The next section will cover how to open the existing Jupyter notebook files that are available in the Sandbox.

1.3 Navigating the Sandbox

Now you have created a user account and successfully logged in, this session will teach you about the main elements of the Sandbox.

If you do not yet have a Digital Earth Africa account, follow the steps in *Create a Digital Earth Africa account*.

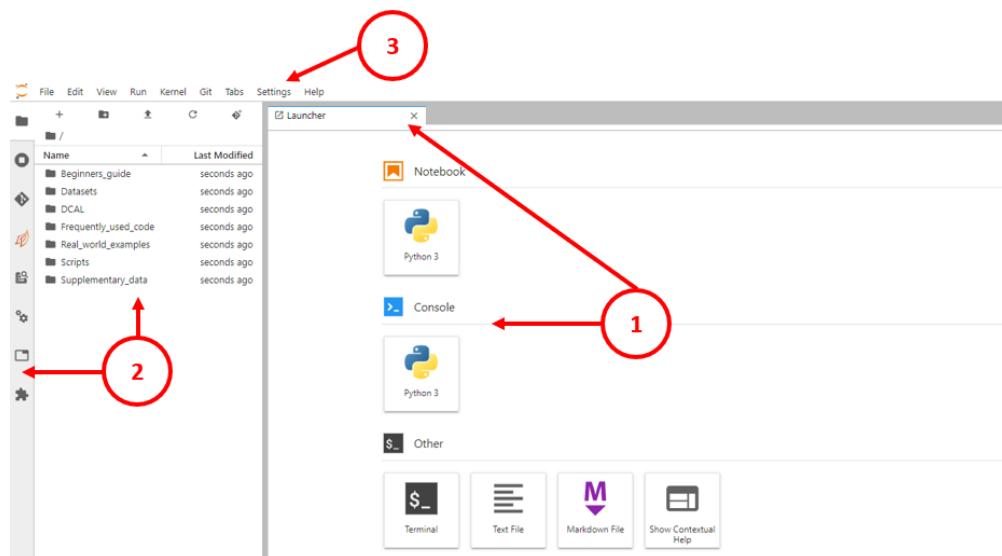
https://youtu.be/6x_S9Kp1Dqs

1.3.1 Overview

The Digital Earth Africa Sandbox uses a JupyterLab interface. JupyterLab is a web-based interactive coding environment. It has a user-friendly interface to make file navigation and workflows more intuitive. Within the Sandbox, we create Jupyter Notebooks to contain and execute code.

1.3.2 The JupyterLab interface

When the Sandbox starts up, you will be presented with three main sections, as numbered in the diagram below:



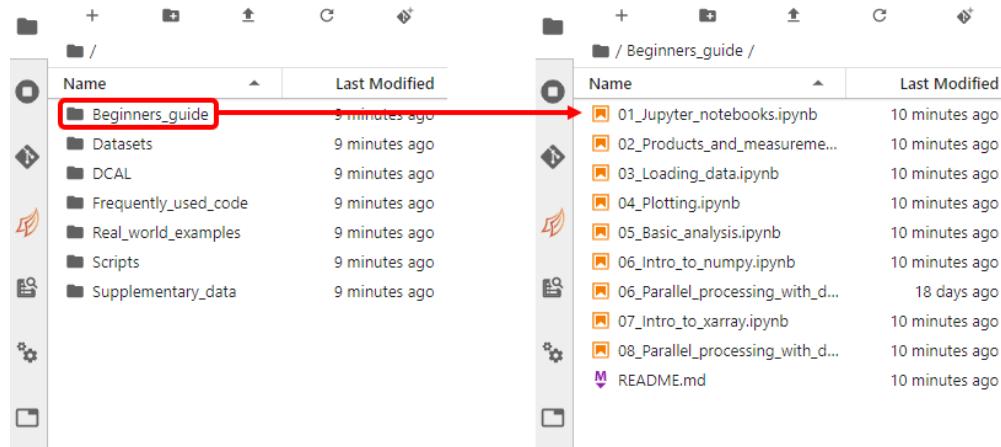
1. **Main work area:** Enables you to arrange documents and other windows (such as terminals or code consoles) into panels of tabs that can be resized or subdivided. Open files will be displayed in this area. Most of your work will be done here.
2. **Left sidebar:** Contains a file browser, the list of running kernels and terminals, the command palette, the notebook cell tools inspector, and the tabs list. By default, the file browser is selected; this will be the most useful toolbar for new users.

3. **Horizontal menu bar:** Exposes actions available in JupyterLab, including actions related to saving, editing, viewing and running notebooks. When you are done, you can select **File -> Log Out** to quit your Sandbox session and free up server space.

1.3.3 Open a Jupyter Notebook

The Sandbox comes with many example notebooks that you can learn from and use. Follow the instructions in this section to learn how to open a notebook.

1. In JupyterLab, double-clicking in the left sidebar file browser opens folders and files. Double-click the **Beginners_guide** folder in the left sidebar to open the folder and view its contents. This is shown in the figure below.



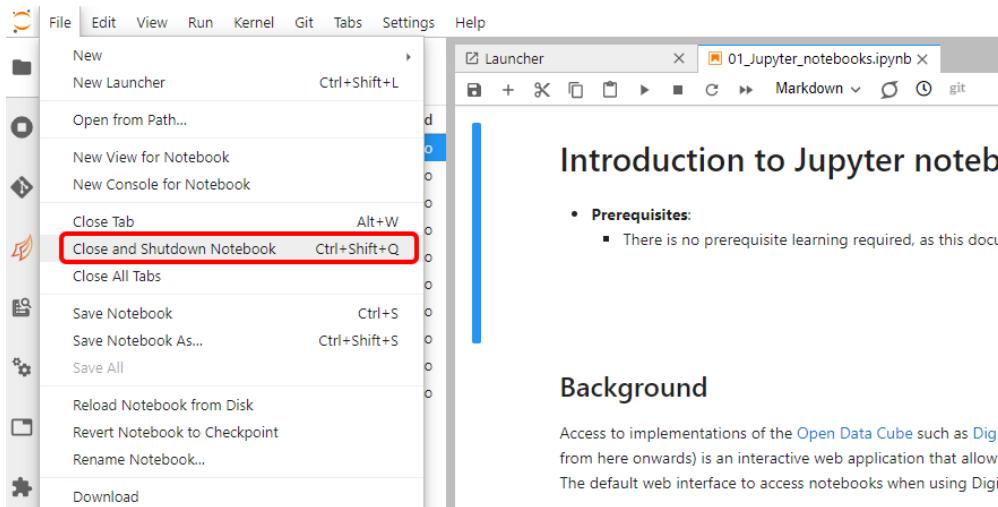
2. Jupyter notebooks can be identified by the file extension `.ipynb`. Choose a notebook (for example, `01_Jupyter_notebooks.ipynb`) and double-click to open it in the main work area.
3. To navigate back to the home folder, click the folder icon at the start of the folder path. The folder path is near the top of the left sidebar.

For more information, refer to this section of the JupyterLab documentation on [Working with Files](#).

1.3.4 Close a Jupyter Notebook

Closing a notebook browser tab will not shut down its “computational engine” (called the kernel). It is good practice to shut down the kernel of any notebooks you aren’t using, as it will free up memory in your Sandbox session. Follow the instructions below to completely shut down the notebook you opened in the previous section.

1. With the notebook open, click on the menu option **File -> Close and Shutdown Notebook**.



- When asked whether you want to shut down the notebook, click **OK**.

1.3.5 Conclusion

You can now open and close existing Jupyter notebook files. The next section will cover how to run an existing Jupyter notebook.

1.4 Running a Notebook

Now that you have learned to open and close notebooks, this section will teach you how to run an analysis using one of the existing Jupyter notebooks.

1.4.1 Overview

Jupyter is an interactive coding environment compatible with the programming language Python. Digital Earth Africa is based on a Python library called the [Open Data Cube](#), so the Sandbox contains Python-based notebooks.

It is helpful, but not required, to be familiar with Python programming. More detail on Python can be found in [Extra session: Python basics](#), which is an **optional** module that provides greater context into common Python commands used in the Digital Earth Africa platform. It is recommended to first follow the steps below to set up a folder in the Sandbox environment before completing the Python basics module.

1.4.2 Video: Working with notebooks

This video gives an overview of the notebook we will be working with. Watch the video, and then follow the written instructions below to complete the exercise.

https://youtu.be/ecVjImPy2_A

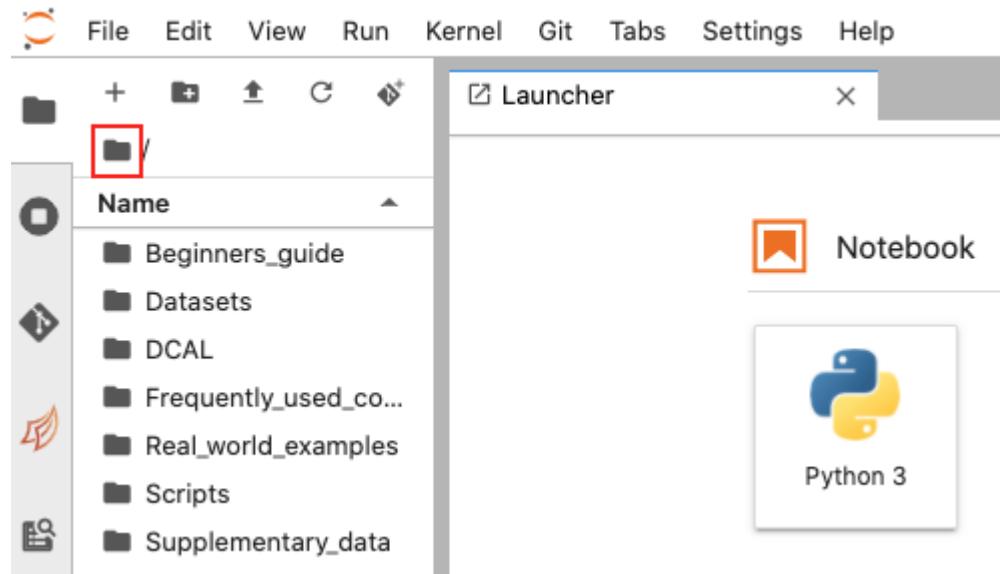
1.4.3 Exercise: Run the Crop Health notebook

This activity will demonstrate how to run one of the Digital Earth Africa Sandbox notebooks. The notebook is a real world example showing how to measure crop health. Follow the instructions below to run the notebook.

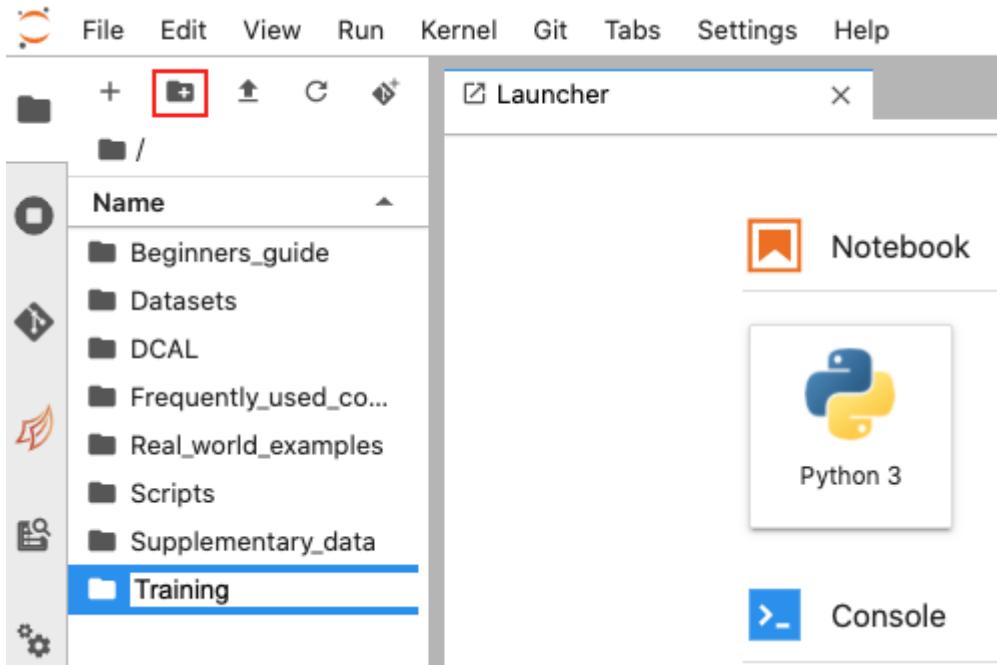
Create a copy of the notebook

The Sandbox comes with a large number of existing notebooks, which you can use as reference material. To keep these in their original format, we recommend making a copy of a notebook when you want to run it. For the rest of this training material, we will ask you to make copies of notebooks and store them in a new **Training** folder. Follow the instructions below to set up this folder and add a copy of the notebook for this session.

1. Click the **Folder icon** in the folder path to make sure you're in the **Home** folder.

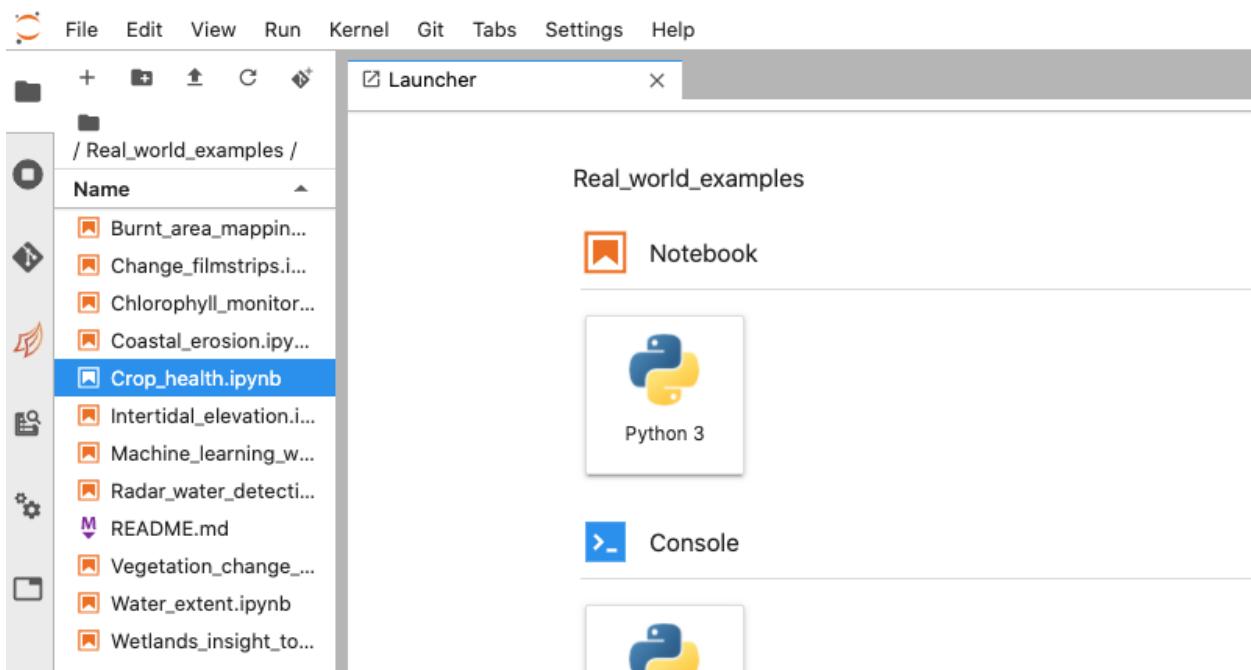


2. Click the **New Folder icon** and name the new folder **Training**. Press **Enter** to finish creating the folder.

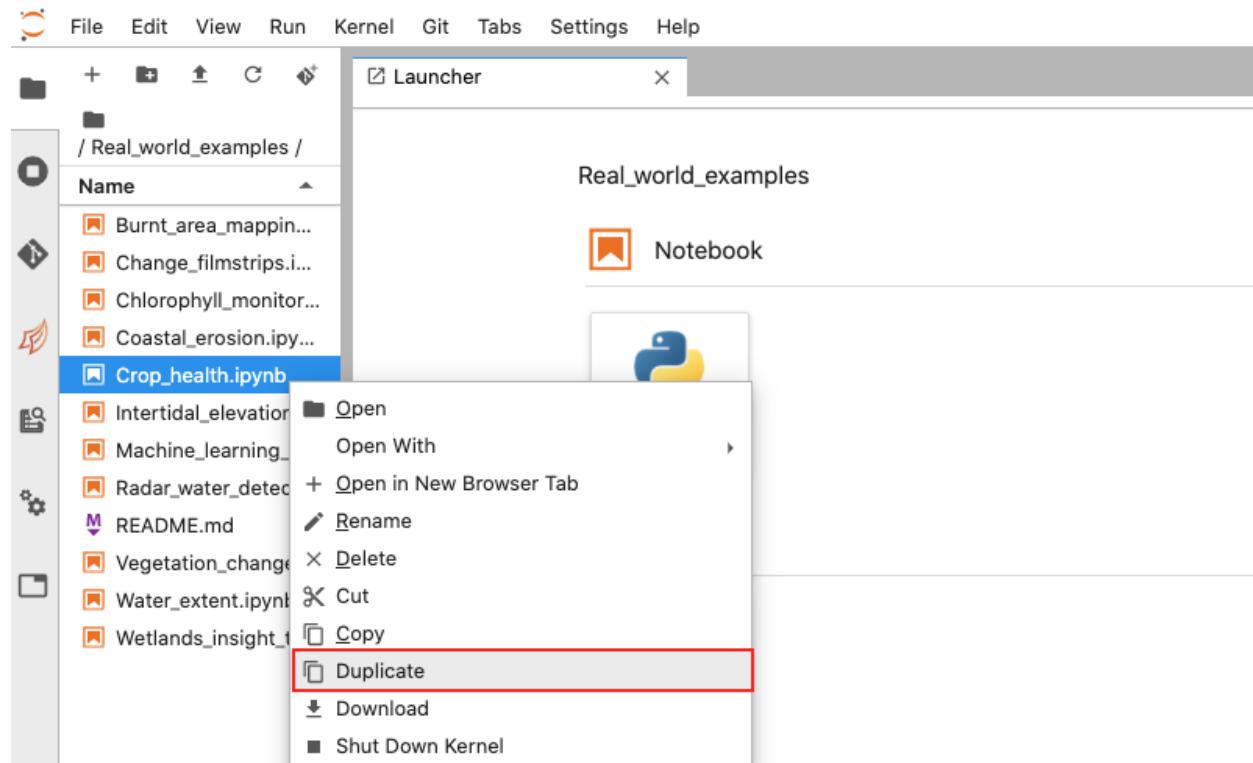


The next step is to identify and copy the notebook you want to work on. For this exercise, this will be the `Crop_health.ipynb` notebook, which is in the **Real_world_examples** folder. Follow the steps below to copy it.

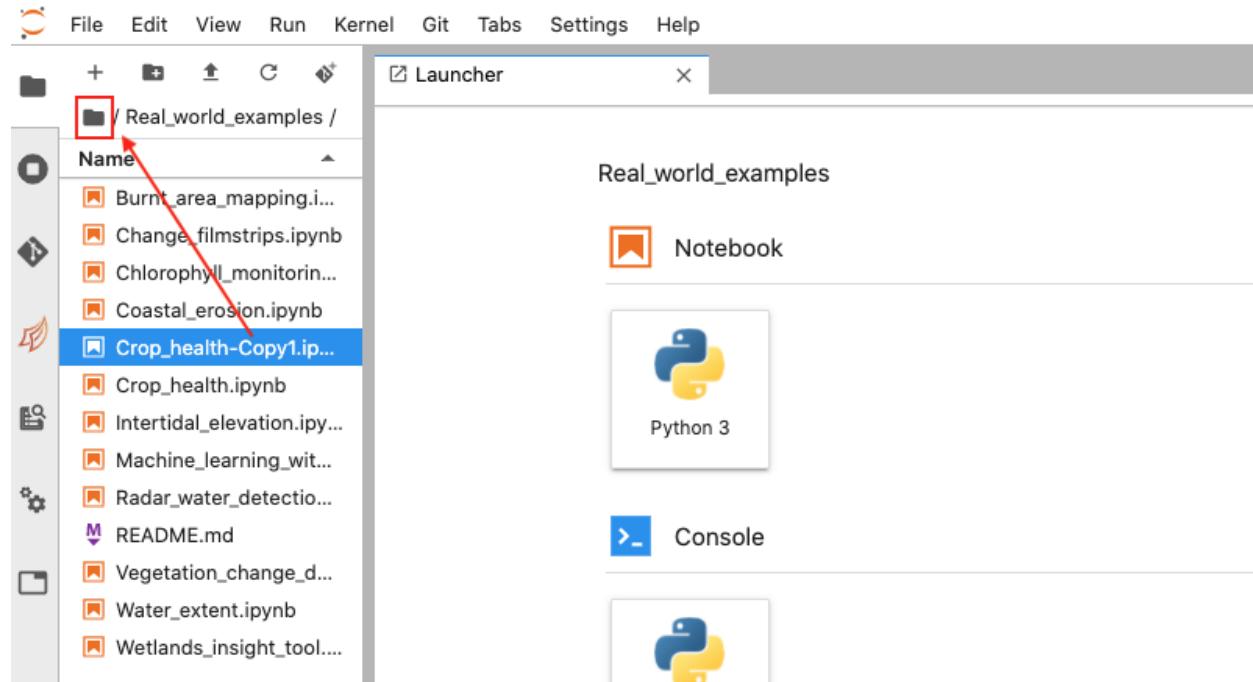
3. Double-click the **Real_world_examples** folder to open it. The `Crop_health.ipynb` notebook is selected in the image below.



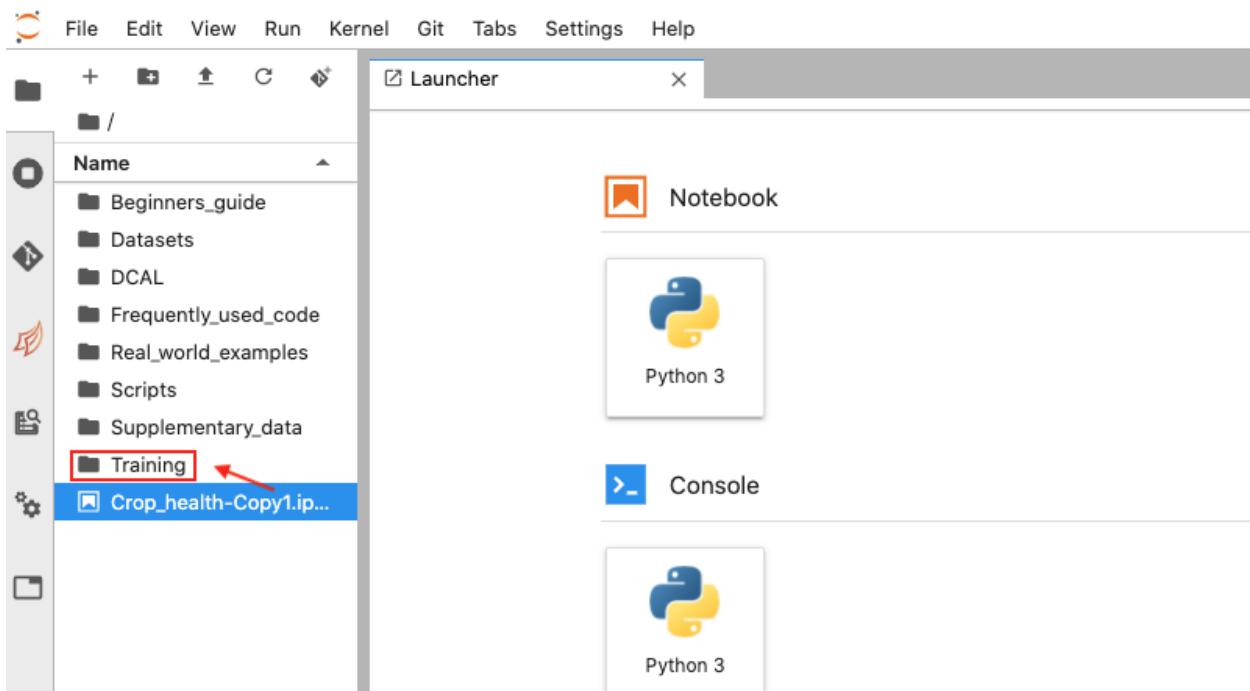
4. Next, right-click on the notebook and click **Duplicate** to create a copy. By default, this will be called `Crop_health-Copy1.ipynb`.



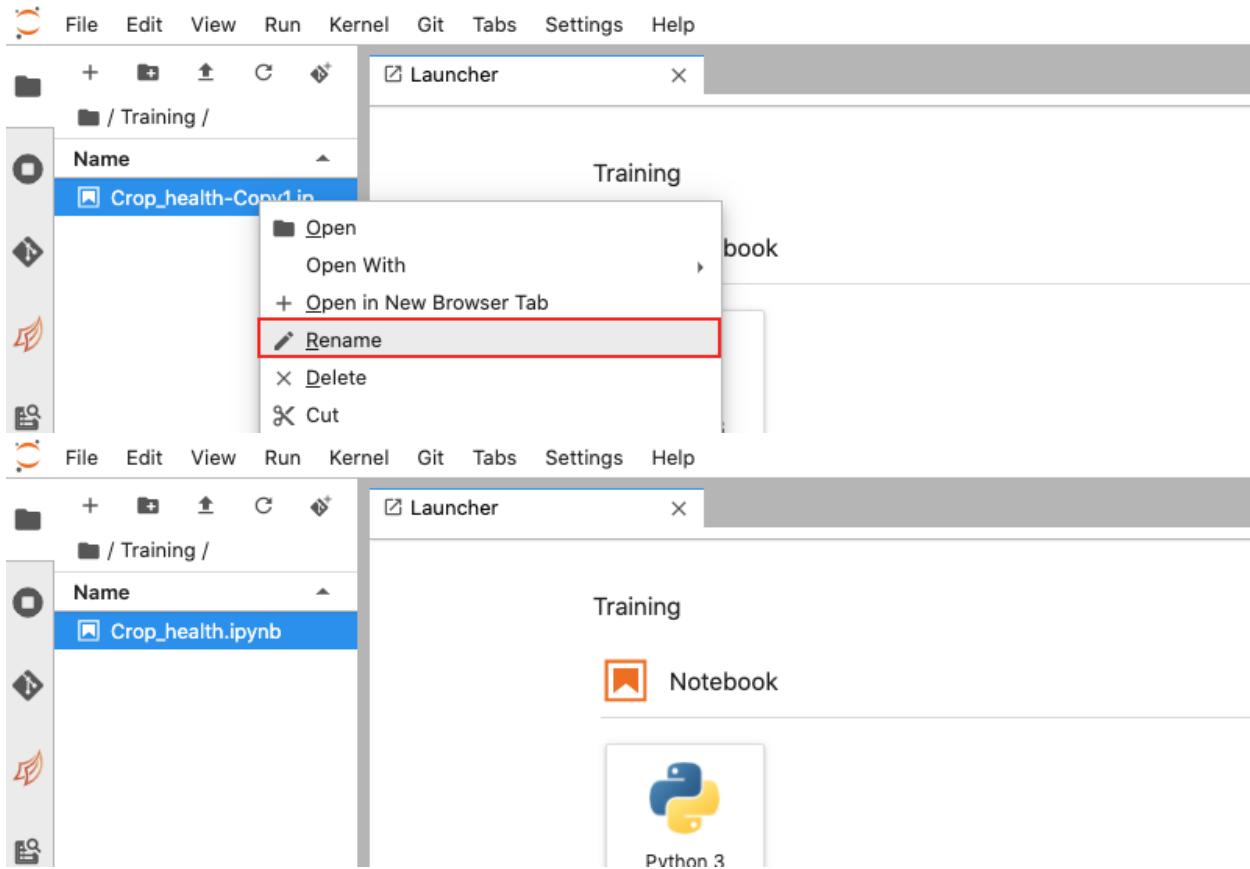
5. Click and drag **Crop_health-Copy1.ipynb** to the **Folder icon** to move it to the **Home** folder.



6. Return to the **Home** folder by clicking the **Folder icon**. You should see the **Crop_health-Copy1.ipynb** notebook. Click and drag this file to the **Training** folder you created.



- Double-click the **Training** folder. Right-click the copied notebook and click **Rename**. Rename the notebook to `Crop_health.ipynb`.

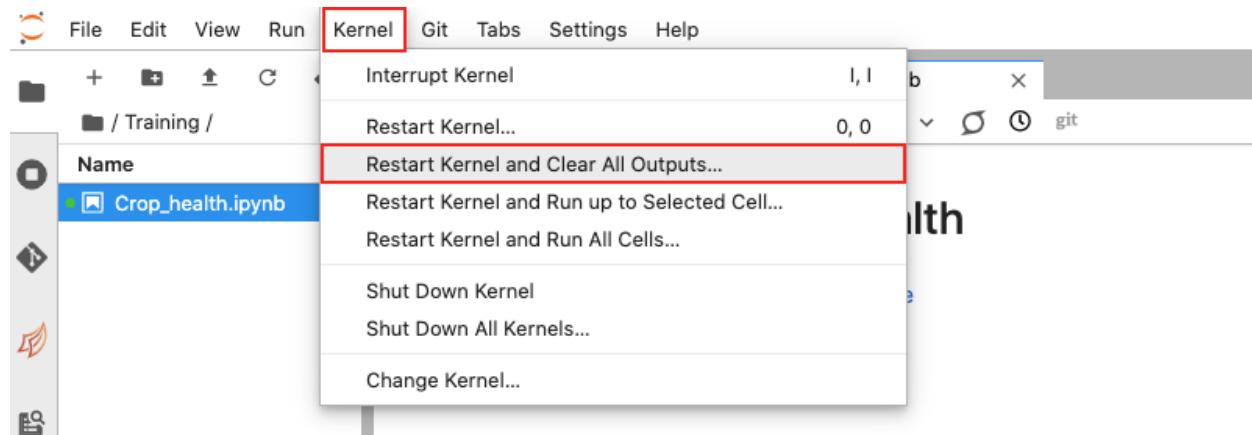


You can now run this notebook, as well as make edits to it.

Starting the notebook

Notebooks on the Sandbox come pre-run. It is best practice to restart and clear the notebook before running.

1. Double-click your copied `Crop_health.ipynb` notebook to open it.
2. In the main menu, click **Kernel -> Restart Kernel and Clear All Outputs**. When asked whether you want to restart the kernel, click **Restart**.



3. Read the **Background** and **Description** sections to learn more about the notebook.

Loading packages and functions

This notebook uses Python packages and functions to conduct the analysis. These are contained in the first code cell, which looks like:

```
[1]: %matplotlib inline
import sys
import datacube
sys.path.append("../Scripts")
from notebookapp_cropehealth import load_cropehealth_data
from notebookapp_cropehealth import run_cropehealth_app
```

To load the necessary packages and functions:

1. In the `Crop_health.ipynb` notebook, click on the code cell that matches the one above.
2. Press **Shift + Enter** on your keyboard to run the cell.
3. When the cell has finished running, the [] icon to the left of the cell should change to [1], as shown below. This indicates that the cell has finished running. The number 1 indicates that this was the first cell run in the notebook.

```
[1]: %matplotlib inline
import sys
import datacube
sys.path.append("../Scripts")
from notebookapp_cropehealth import load_cropehealth_data
from notebookapp_cropehealth import run_cropehealth_app
```

Pick a study site

Next, we have to provide the area of interest using latitude, longitude, and a buffer, which describes how many square degrees to load around the set latitude and longitude values. For now, keep the values that have been set. The notebook will turn this into a bounding box, which will be used to load the data.

1. Click on the cell corresponding to the image below in the notebook.

```
[ ]: # Define the area of interest for the analysis
lat = 14.789064
lon = -17.065202
buffer = 0.005
```

2. Press **Shift + Enter** on your keyboard to run the cell.
3. When the cell has finished running, the [] icon to the left of the cell should change to [2].

Loading the data

This notebook uses a special function called `load_crophealth_data` to load the required data. Later in this training course, you will learn how to write your own data loading commands.

The cell containing this function is shown in the image below:

```
[ ]: dataset = load_crophealth_data(lat, lon, buffer)
```

The `load_crophealth_data` function goes through a number of steps. It identifies the available Sentinel-2 data over the last two years, then masks the bad quality pixels such as cloud and cloud shadow. After the masking step, the function only keeps images where more than half the pixels are good. Finally, it calculates the Normalised Difference Vegetation Index (NDVI) and returns the data. The returned data are stored in the `dataset` object.

To run the cell:

1. Click on the cell corresponding to the image above in the notebook.
2. Press **Shift + Enter** on your keyboard to run the cell. Outputs should start to appear below the cell.
3. When the cell has finished running, the [] icon to the left of the cell should change to [3].

As the cell is running, you should see outputs describing the function's actions. The output is shown in the image below:

```
Using pixel quality parameters for Sentinel 2
Finding datasets
  s2_l2a
Counting good quality pixels for each time step
Filtering to 100 out of 145 time steps with at least 50.0% good quality pixels
Applying pixel quality/cloud mask
Loading 100 time steps
```

In this case, the function identified 145 observations over the last 2 years, then kept and loaded 100 that met the quality requirements.

Run the Crop Health App

After loading the data, you can visualize the health of various fields using the `run_cropherealth_app` function. This is a special function for this notebook, which takes the `dataset` you loaded, as well as the latitude, longitude and buffer parameters that define the area of interest. It then starts an interactive app that you can use to measure the average NDVI in different fields.

The cell containing this function is shown in the image below:

```
[ ]: run_cropherealth_app(dataset, lat, lon, buffer)
```

1. Click on the cell corresponding to the image above in the notebook.
2. Press Shift + Enter on your keyboard to run the cell. An interactive app should appear, as shown in the image below:



3. The left-hand side of the app contains a map, which will allow you to draw a polygon around a field of interest. Click the **Polygon icon**, then click points on the map to outline an area, as shown below:

```
[4]: run_crophealth_app(dataset, lat, lon, buffer)
```

Draw a polygon within the red box to view a plot of average NDVI over time in that area.



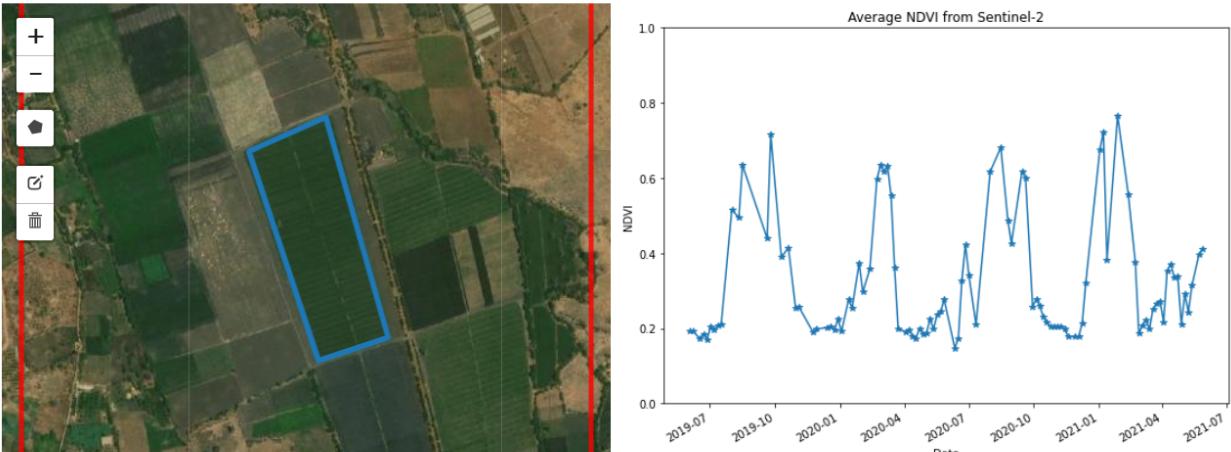
Leaflet | Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community

Plot status:

Note: The map in this app shows an ESRI basemap which is higher resolution than the Sentinel-2 data used to calculate the NDVI index. The purpose of the map is to guide you in drawing field boundaries.

4. Click the first point of the polygon to finish selecting the area. The app will then calculate the average NDVI for that area and display it on the right-hand side of the app, as shown below:

Draw a polygon within the red box to view a plot of average NDVI over time in that area.



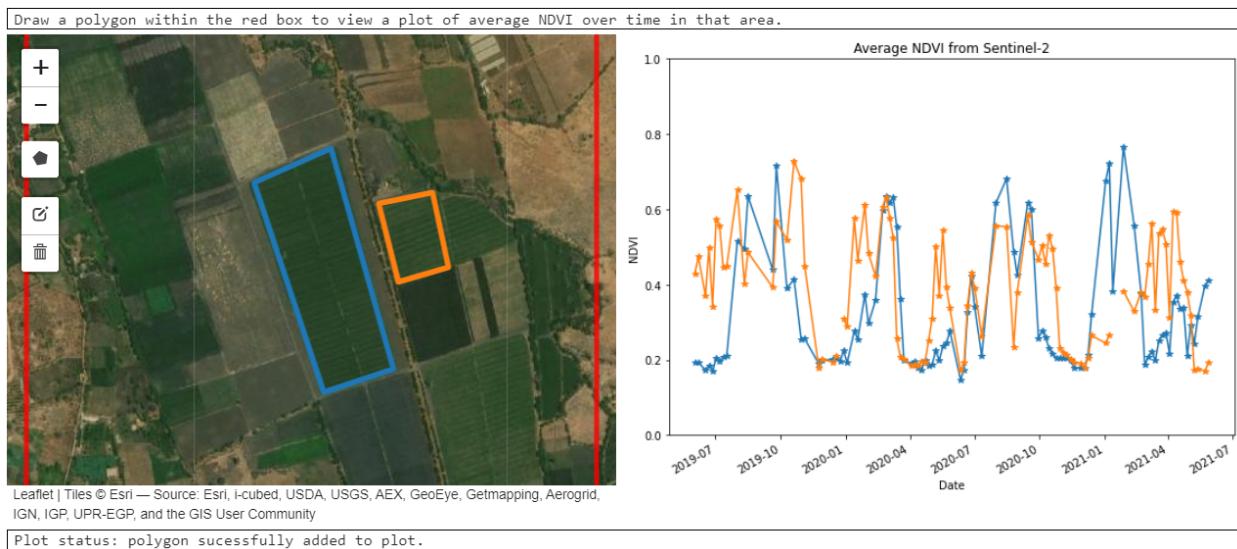
Leaflet | Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community

Plot status: polygon sucessfully added to plot.

The cycles in the graph are likely related to growth and harvesting events.

5. It is possible to draw multiple polygons and compare the vegetation index over time of different fields. Click the

Polygon icon to draw a second polygon:



1.4.4 Conclusion

Congratulations! You have finished the first session of this training! You now know how to open and close notebooks, make copies of notebooks and run them.

1.4.5 Optional activity

If you would like to re-run this notebook using a different area of interest, follow these steps:

1. Restart the notebook and clear all outputs.
2. Run the cell that imports packages and functions.
3. Change the latitude, longitude and buffer values in the second code cell. You can either use the values suggested above the cell, or provide your own. Then run the cell.

Note: If you're going to use your own area of interest, make sure data is available by looking at the Sandbox Metadata Explorer for Sentinel-2 at https://explorer.digitalearth.africa/s2_l2a. You will also need to make sure you provide latitude and longitude values (rather than Easting and Northing values).

4. Run the rest of the notebook as described in the tutorial, drawing in areas of interest using the **Polygon icon** on the map. Did you learn anything interesting about the fields for your area of interest?
5. Investigate the **optional Python basics extra session** before continuing on to Session 2. If you are new to Python, it may be helpful for context and reference. If you have previous experience in Python, it provides a review of the major concepts used in the Digital Earth Africa training course.

1.5 Session 1 Quiz

In Session 1, you learned about the mission and purpose of Digital Earth Africa. You also created an account on the Digital Earth Africa Sandbox, a platform where you can retrieve and analyse Earth observation data over Africa. The exercise was to copy, open and run the Crop Health notebook.

1.5.1 Quiz

If you would like to be awarded a certificate of achievement at the end of the course, we ask that you [complete the quiz](#). You will need to supply your email address to progress towards the certificate. After you complete the quiz, you can check if your answers were correct by pressing the **View Accuracy** button.

The quiz will ask you to use the Crop Health notebook you developed for this session's exercise. To review the exercise, visit the section **Running a Notebook** or click the **Previous** button.

**CHAPTER
TWO**

SESSION 2: DATASETS

This session is all about the datasets available through Digital Earth Africa. These datasets can be visualised through the Digital Earth Africa Map, or sorted by availability in the Digital Earth Africa Metadata Explorer. This session will cover how to use the Digital Africa Sandbox to load datasets for a given area of interest and make a colour image.

Click on **Digital Earth Africa products** to get started. If you would like to review Session 1 content first, click **Session 1: Introduction** in the side menu.

2.1 Digital Earth Africa products

The Digital Earth Africa Sandbox allows users to access a variety of Earth observation products. These products are comprehensive datasets that form the basis of all analysis conducted in the Sandbox. As a Sandbox user, it is helpful to know more about the available products: each has its own strengths and limitations that can impact results.

The Digital Earth Africa products can be split into two types:

- Satellite input products: datasets of measurements taken by satellites
- Derived products: datasets created by applying algorithms to satellite products

For now, we will focus on satellite input products. This section describes the two main datasets we will see in this training course — **Landsat 8** and **Sentinel-2**.

2.1.1 Video: Satellite products

Watch this short video for an overview of Digital Earth Africa satellite products. Then, read the text below for more detail.

Download the video slides as a PDF

2.1.2 Satellite products



An artistic rendering of one of the Sentinel-2 satellites. [Source]

Satellite products are datasets output from satellites orbiting Earth. These datasets have undergone some pre-processing by the satellite operators, but are otherwise ‘raw’ data, measuring quantities such as reflectance, radiation and surface temperature.

These products are fundamental to Earth observation analytics. They are distinguished by a few key attributes:

- **Spatial resolution (how detailed):** Spatial resolution refers to the pixel size of the satellite product. It is the smallest thing the satellite can ‘see’. A $30\text{ m} \times 30\text{ m}$ spatial resolution means the satellite product cannot pick up any details smaller than $30\text{ m} \times 30\text{ m}$.
- **Temporal resolution (how often):** Satellites pass over different parts of the Earth at different times of the day and month. The frequency of measurements depends on how many satellites are contributing to the product.
- **Spectral resolution (how versatile):** Spectral resolution refers to the sensors on the satellite. The sensors are calibrated to collect measurements in certain wavelengths of the electromagnetic spectrum, known as ‘bands’. Different bands are used for identifying different features on Earth, so the more bands a product has, the more versatile it is.

Digital Earth Africa hosts several satellite products available to Sandbox users. The two products used most in this training course are the Landsat 8 product and the Sentinel-2 product.

2.1.3 Satellite product: Landsat 8

Landsat 8 is the latest iteration of the Landsat program run jointly by the United States Geological Survey (USGS) and National Aeronautics and Space Administration (NASA). It consists of one satellite in a polar orbit, launched on February 11, 2013, and is still in operation today.

The Landsat data is organised by collections. Recent pre-processing methodology upgrades by USGS have allowed the release of ‘Collection 2’. This supersedes the older ‘Collection 1’ of Landsat data. Collection 2 is the collection used in Digital Earth Africa.

Each collection contains the data at two levels; Level-1 and Level-2. Level-1 data is surface reflectance measured at the top of the atmosphere. Level-2 has been pre-processed for atmospheric corrections, and gives surface reflectance at bottom-of-atmosphere. Since the corrections increase data quality and usability, Level-2 data is generally used.

The Digital Earth Africa Landsat 8 product uses Collection 2 Level-2 data, and holds the following characteristics:

- **Sandbox alias:** ls8_sr
- **Data type:** Surface reflectance
- **Data timespan:** March 2013 – present
- **Available regions:** Entire African continent
- **Spatial resolution:** 30×30 m (size of one pixel)
- **Temporal resolution:** 16 days (one flyover every 16 days, or approximately twice a month)
- **Spectral resolution:** 7 spectral bands — Coastal/Aerosol, Blue, Green, Red, Near-Infrared, Short Wavelength Infrared 1, Short Wavelength Infrared 2

For more details, see the following resources:

- Digital Earth Africa dataset specifications: [Landsat Collection 2 Level-2](#)
- `Landsat.ipynb` in the **Datasets** folder of the Sandbox
- [USGS Landsat Collections website](#)

These sources also contain information about previous Landsat missions; together, Landsat 5/7/8 data covers the time period 1984 – present.

2.1.4 Satellite product: Sentinel-2

The Sentinel-2 mission is part of the European Union ‘Copernicus’ Earth observation programme. Sentinel-2 consists of twin satellites, Sentinel-2A (launched 23 June 2015) and Sentinel-2B (launched 7 March 2017). Their combined data is used in the Digital Earth Africa Sentinel-2 product.

Like the Landsat datasets, Sentinel-2 data is tiered by level of pre-processing. Level-0, Level-1A and Level-1B data contain raw data from the satellites, with little to no pre-processing. Level-1C has been corrected to top-of-atmosphere reflectance, while Level-2A is bottom-of-atmosphere reflectance. As with Landsat, the Level-2A bottom-of-atmosphere measurements are most ideal for further research activities.

The Digital Earth Africa Sentinel-2 product uses Sentinel-2 Level-2A data, and holds the following characteristics:

- **Sandbox alias:** s2_12a
- **Data type:** Surface reflectance
- **Data timespan:** July 2017 – present
- **Available regions:** Entire African continent
- **Spatial resolution:** 10×10 m, 20×20 m, 60×60 m (depending on band)
- **Temporal resolution:** 5 days
- **Spectral resolution:** 12 spectral bands — Coastal/Aerosol, Blue, Green, Red, Red Edge 1, Red Edge 2, Red Edge 3, Near-Infrared 1, Near-Infrared 2, Water Vapour, Short Wavelength Infrared 1, Short Wavelength Infrared 2

Note that standalone Sentinel-2A data exists for the period of 2015 – 2017, before Sentinel-2B was launched, but as such has a reduced temporal resolution.

For more details, see the following resources:

- Digital Earth Africa dataset specifications: Sentinel-2 Level-2A
- `Sentinel_2.ipynb` in the **Datasets** folder of the Sandbox
- ESA Sentinel-2 mission website

2.1.5 Which satellite product is ‘best’?

This question depends on the analysis you are doing. It helps to consider the available resolutions of the satellite products. For example:

Analysis timespan: Recent short-term changes (for example, over a couple of months) are best captured by the Sentinel-2 product, which has good temporal resolution. However, if it were changes over several years (or decades), then the Landsat 8 product might be more suitable.

Analysis region size and level of detail: Sentinel-2 data has better native resolution than Landsat 8, so it is often preferred as it can distinguish smaller artifacts. However, if your region of analysis is expansive and you are looking for larger landmarks (for example, large bodies of water), then Landsat 8 may provide the level of detail required. Using Landsat 8 in that case may significantly reduce computation time while still providing comparable results.

2.1.6 Conclusion

Now you know more about the main satellite products covered in this training course. We will use both Landsat 8 and Sentinel-2 in the upcoming data manipulation exercises.

2.2 Digital Earth Africa Map

Satellite products, as a dataset, are just a lot of numbers — not the most intuitive format when you are deciding how to analyse the data. It is much more helpful to see what the data looks like. The [Digital Earth Africa Map](#) portal helps users visualise Digital Earth Africa products by offering an interactive map.

2.2.1 Overview

In this section, we will show you how to navigate the [Digital Earth Africa Map](#). This includes finding the longitude and latitude coordinates of a location. We will then walk through a short exercise on displaying Landsat 8 data on the Map.

2.2.2 Video: Introduction to the DE Africa Map

This video will show you the basics of the Digital Earth Africa Map portal. In the video, a derived product (called ‘Water Observations from Space’) is loaded.

Watch the video to see the product loading process, then follow the written instructions in the exercise below to load and visualise Landsat 8 data.

<https://youtu.be/44mEA6GjugE>

2.2.3 Navigate the Digital Earth Africa Map

To access the Map, open <https://maps.digitalearth.africa/>. This will bring you to the Digital Earth Africa Map user interface.

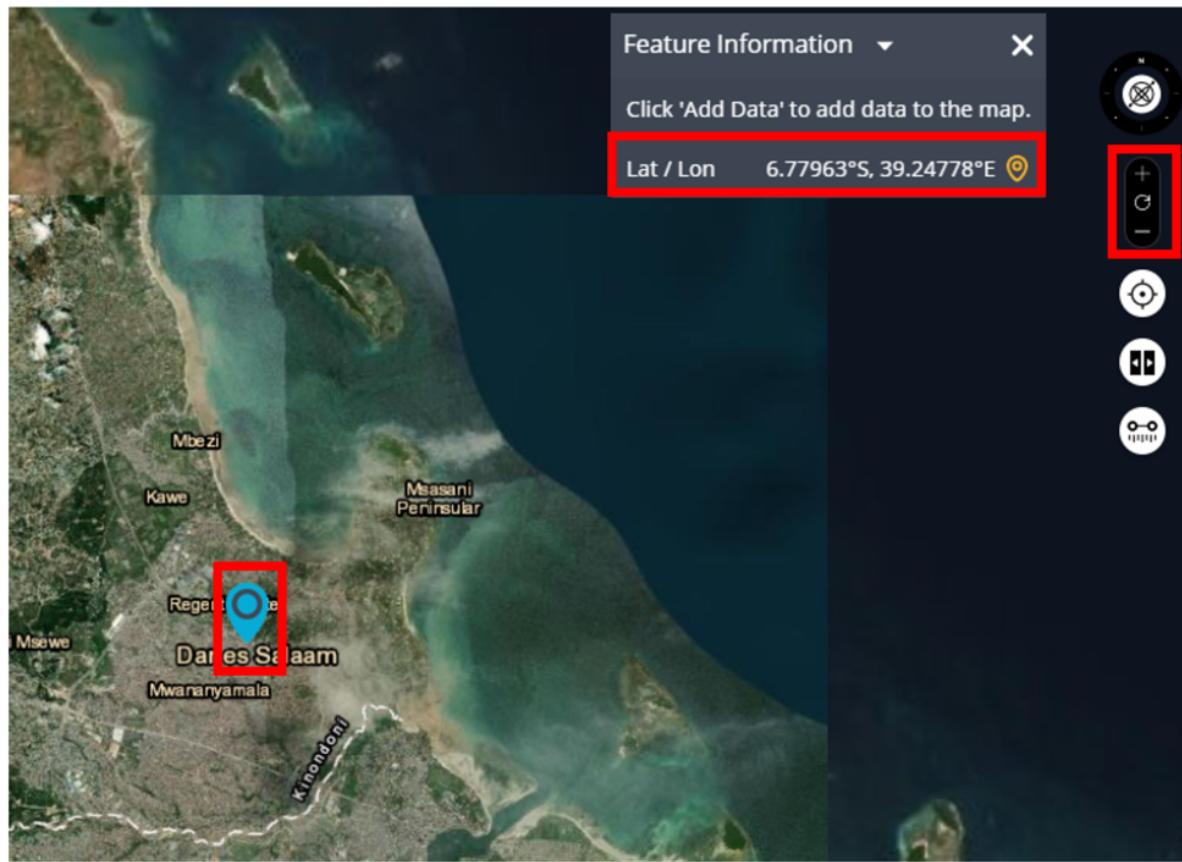


The Digital Earth Africa Map can be split into two sections. The exercise below will cover how to use both.

1. **Data workbench (left side):** Allows users to select data they want to view on the Map. By default, no data is selected.
2. **Map interface (right side):** Displays the data selected in the workbench, and allows users to customise how the data is viewed. By default, it shows an Esri World Imagery Basemap.

Longitude and latitude coordinates

Click anywhere on the map interface to display the longitude and latitude of the selected point.



- The selected point is shown by a blue location marker.
- Zoom in to the shaded area by scrolling up on your mouse or pressing the floating + button on the side of the map interface. Left-click and drag to pan the map.
- The latitude and longitude coordinates are shown in the **Feature Information** window. You can highlight and copy them with your mouse.

This function is useful when you want to perform analysis on a certain area, and need to input the longitude and latitude to load the data. We do not need coordinates for the exercise below.

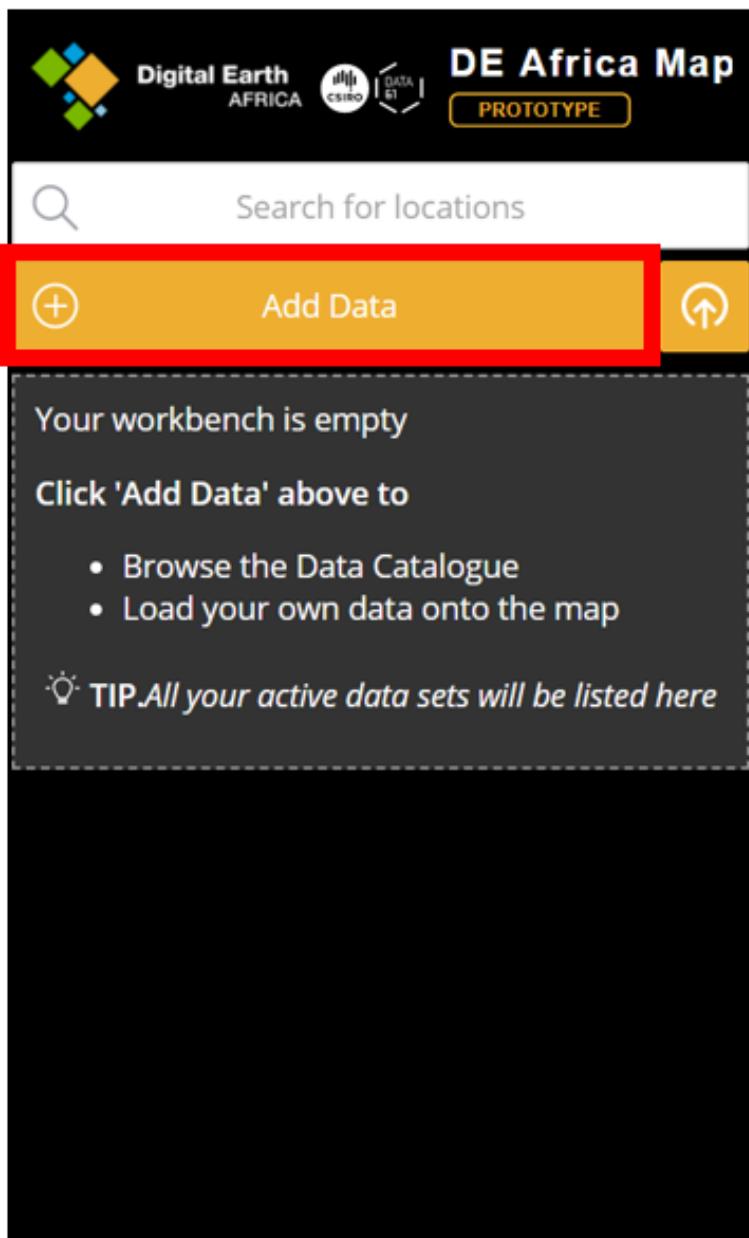
Note: The coordinates from Digital Earth Africa Maps are given in terms of degrees in the cardinal directions (North, East, South, West). Latitude coordinates may be North (N) or South (S). Longitude coordinates may be East (E) or West (W). If copying these values for use in the Sandbox, you must add a negative (-) sign to the latitude value if it is given as South, and you must add a negative (-) sign to the longitude value if it is given as West.

2.2.4 Exercise: Load Landsat 8 imagery

This activity will demonstrate how to load Landsat 8 imagery onto the Digital Earth Africa Map. Follow the instructions below to open the dataset in the workbench and view it in the map interface.

Note: The video above refers to the dataset for Water Observations from Space, however this exercise will use the Landsat 8 dataset.

1. First, we need to select the Landsat 8 dataset. To add data to the data workbench, click **Add Data** to open the Data Catalogue.



2. The datasets are organised by category. Landsat 8 is a satellite product, so select **Satellite images**. Click **Surface**

reflectance and **Daily surface reflectance** to open the folder of Landsat surface reflectance products. Click **Surface reflectance (Landsat 8)**.

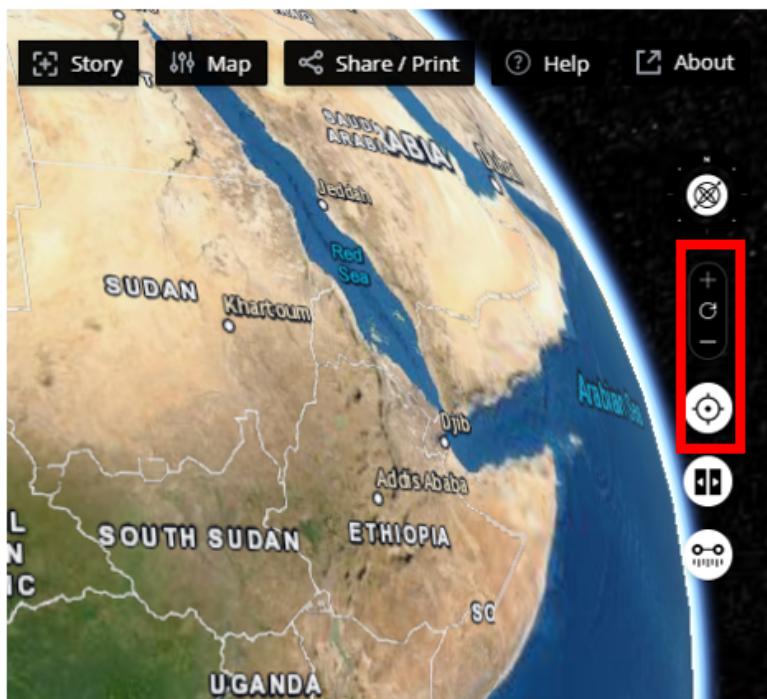
The right panel will show information on the selected dataset. Click **Add to the map**. This will add the dataset to the workbench and close the Data Catalogue.

The screenshot shows the 'Data Catalogue' interface with the 'My Data' tab selected. In the left sidebar, under 'Satellite images', the 'Surface reflectance' category is expanded, and 'Daily surface reflectance' is selected. Under 'Surface reflectance', 'Surface reflectance (Landsat 8)' is highlighted with a red box. On the right, a preview window for 'Surface reflectance (Landsat 8)' shows a map of Port Elizabeth, South Africa, with an 'Add to the map' button highlighted with a red box. A 'Done' button is at the top right of the preview window.

3. The Landsat 8 data will be displayed in the map interface as blue shaded areas. By default, the data shows the latest available timestep. In this example, this is 25 January 2020.



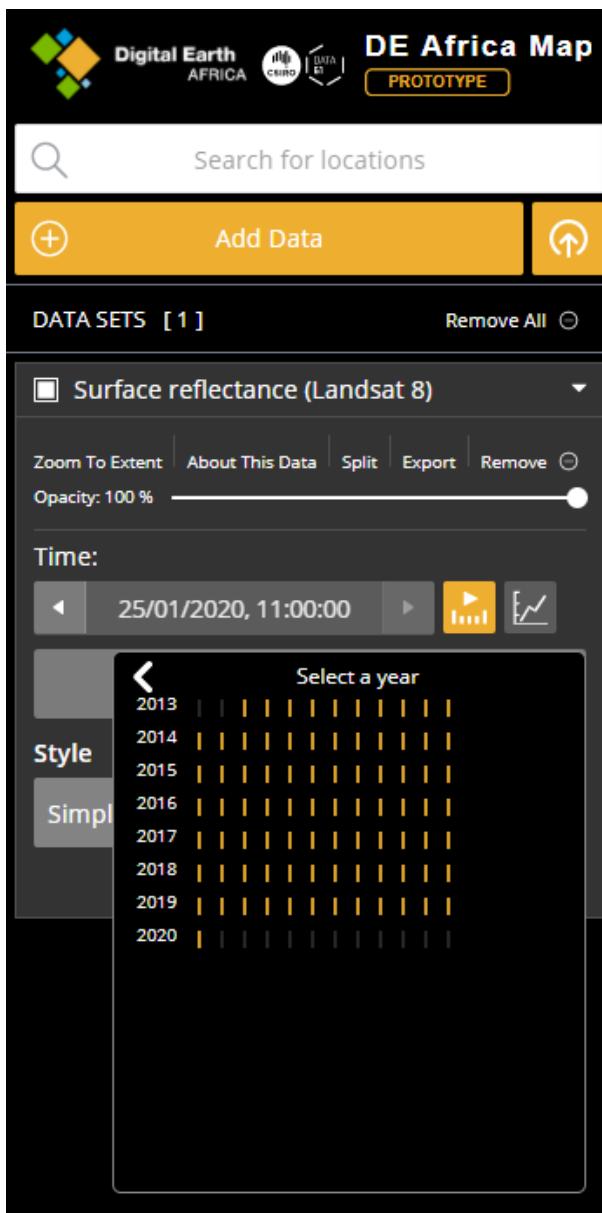
4. Zoom in to the shaded area by scrolling up on your mouse or pressing the floating + button on the side of the map interface. Left-click and drag to pan the map.



5. Zooming in on the shaded area will show the Landsat 8 data rendered as true-colour imagery. This is only one day of data, so not all of Africa is covered in the satellite flyover. Only the area inside the rectangle is Landsat 8 imagery.



6. To see other days of data, click the date shown in the workbench dataset **Time** bar. This will open a menu that shows all available timesteps for the Landsat 8 product. Select a different time and see how the map shaded area changes.



Congratulations — you have used the Digital Earth Africa Map to view Landsat 8 data.

2.2.5 Optional activity

Now you have successfully loaded Landsat 8 imagery on the Digital Earth Africa Map, try loading imagery for Sentinel-2.

Hints:

- The Sentinel-2 product is also a surface reflectance dataset
- You can have multiple datasets in the workbench

Other useful functions:

- Click on the white square to deselect a dataset. This will keep it in the workbench but remove it from the map interface display

- Click on **Remove** to remove the selected dataset
- Click on **Remove All** to clear all datasets from the workbench

Details on other functions of the Digital Earth Africa Map can be found in the *Map portal user guide*, but are not necessary to progress in the training course.

2.2.6 Conclusion

Using the [Digital Earth Africa Map](#), we can see the satellite data in a visual way. This makes it easier to understand the scope of the data we are analysing. It is also a useful tool for pinpointing longitude and latitude coordinates of areas of interest.

2.3 Data availability

In the [previous section](#), we loaded the Landsat 8 dataset onto the [Digital Earth Africa Map](#) and saw it only covers certain parts of Africa on each day. The [section on Digital Earth Africa products](#) tells us Landsat 8 data is only available at certain times in some countries.

How do we know specifically **where** and **when** data is available? Before we start any analysis, we can answer this question by verifying existing data on the [Digital Earth Africa Metadata Explorer](#). The Metadata Explorer can be found at <https://explorer.digitalearth.africa/>.

2.3.1 Overview

In this section, we will use the [Digital Earth Africa Metadata Explorer](#) to check where data is available for a selected time period. The place we will investigate is Tanzania, for the time period of 2018.

2.3.2 Map or Explorer?

The [Digital Earth Africa Metadata Explorer](#) and [Digital Earth Africa Map](#) look similar, but they are designed for different purposes.

Use the [Digital Earth Africa Map](#) if you:

- Want to see what the product or dataset looks like

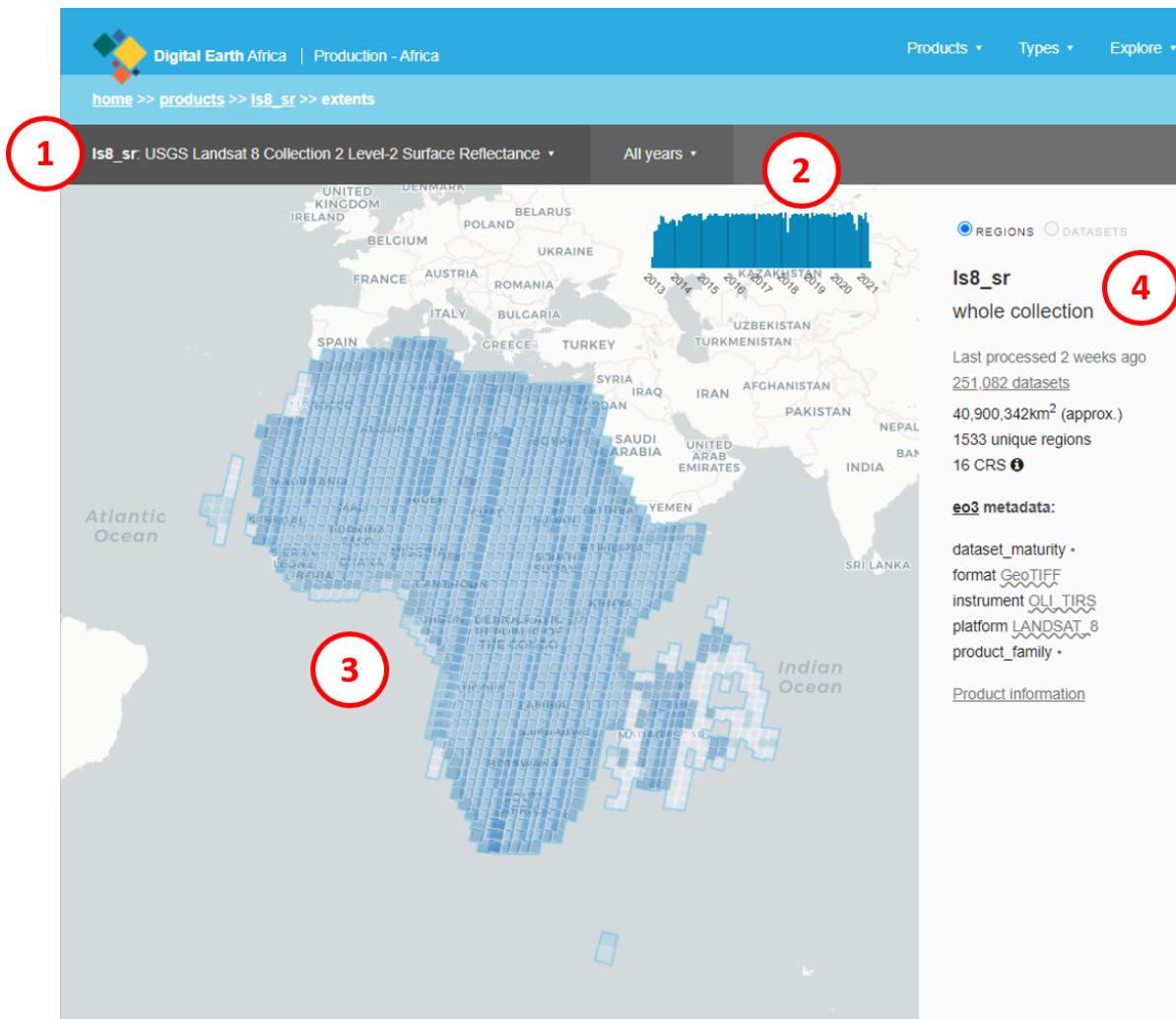
Use the [Digital Earth Africa Metadata Explorer](#) if you:

- Want to know exactly where and when you can find data

2.3.3 Exercise: Data for Tanzania in 2018

Open the [Digital Earth Africa Metadata Explorer](#)

1. Open <https://explorer.digitalearth.africa/>. This will display the Metadata Explorer user interface.

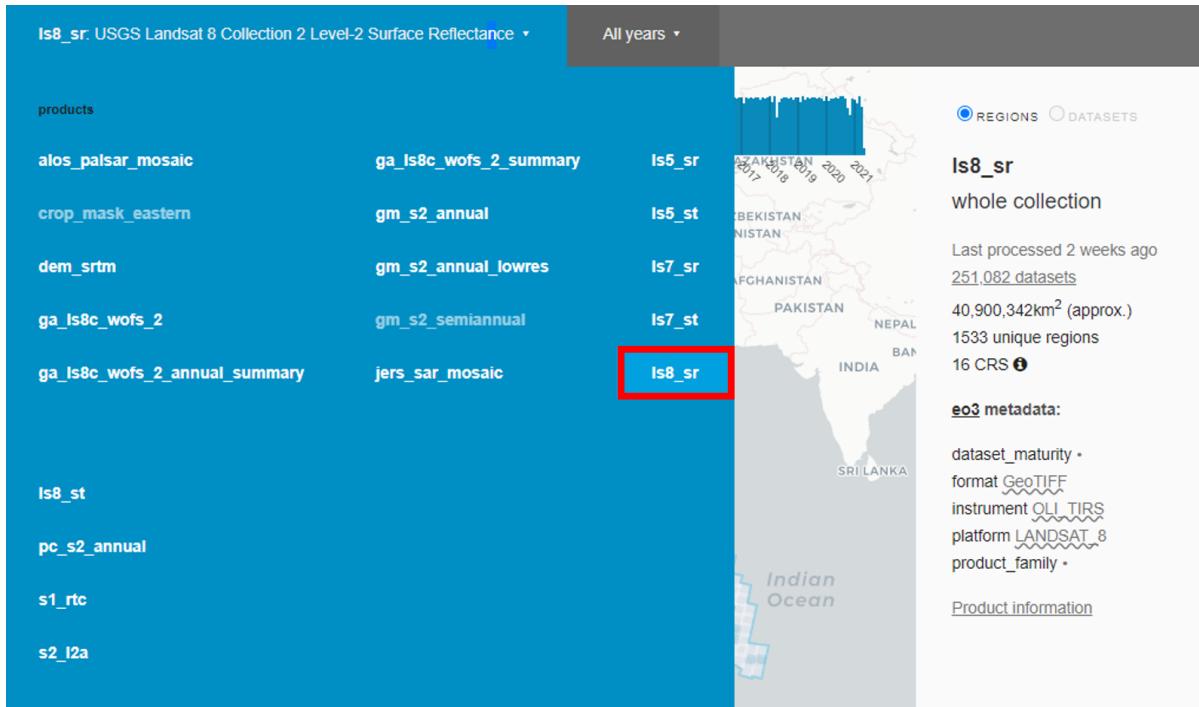


The Digital Earth Africa Metadata Explorer has four main sections.

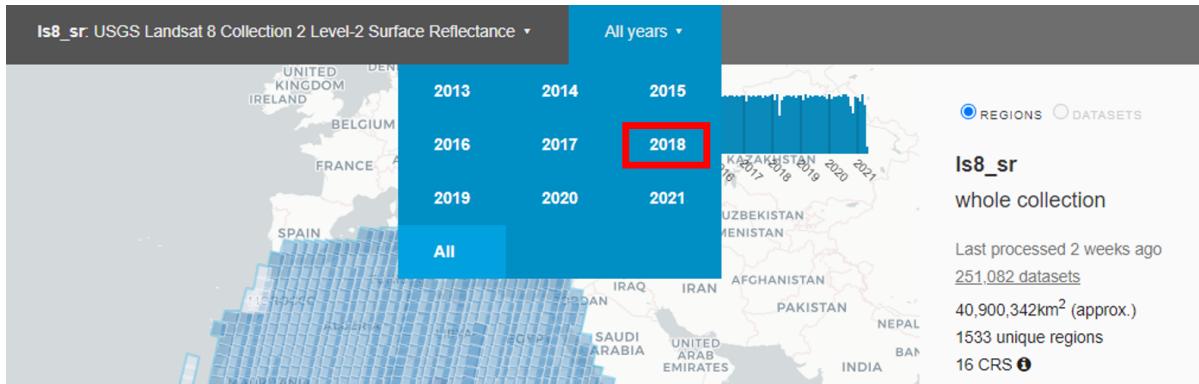
1. **Product selection:** This shows the currently-selected product. Click the selected product name to open the products dropdown menu.
2. **Time period:** This shows the time period for which the selected product is being displayed. Click the selected time period to open the time selection dropdown menu.
3. **Map display:** This shows where data is available, for the selected time and product. Blue shaded tiles indicate the presence of data.
4. **Product information:** The sidebar shows more information about the data for the selected time and product. For example, this includes how many datasets are selected, the name of the datasets in the product, and its coordinate reference system.

Select the Landsat 8 product

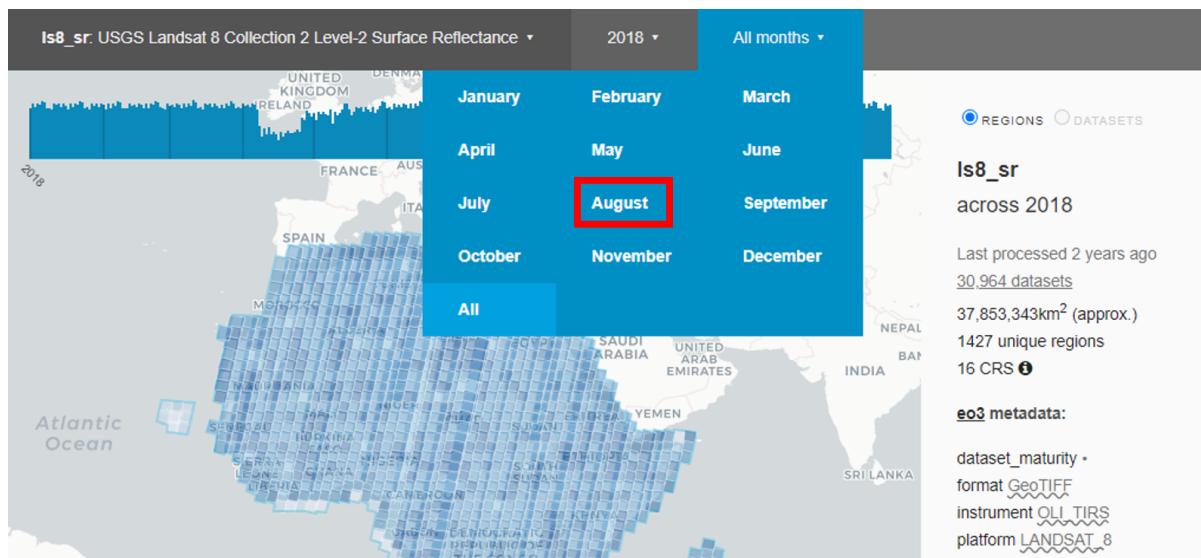
1. Click the **product selection** bar to open the dropdown menu. Select **ls8_sr**. This selects the Landsat 8 product.



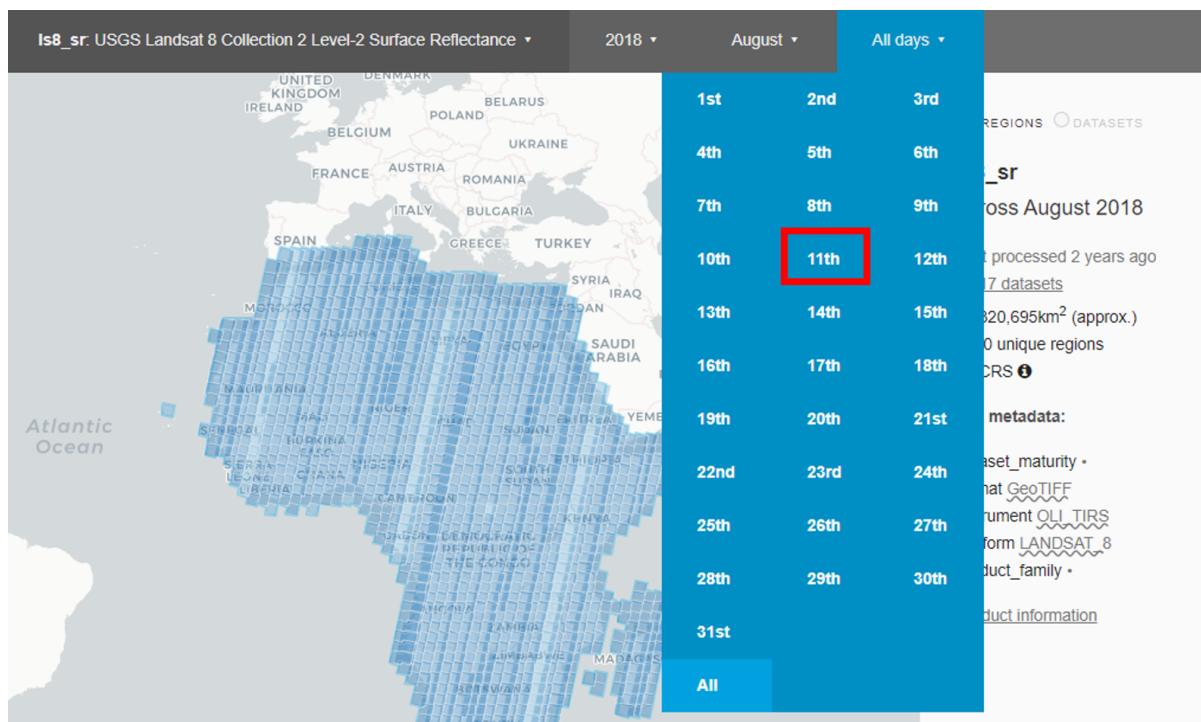
2. Click the **time** bar to open the dropdown menu. Select **2018**. This will show all Landsat 8 datasets for 2018.



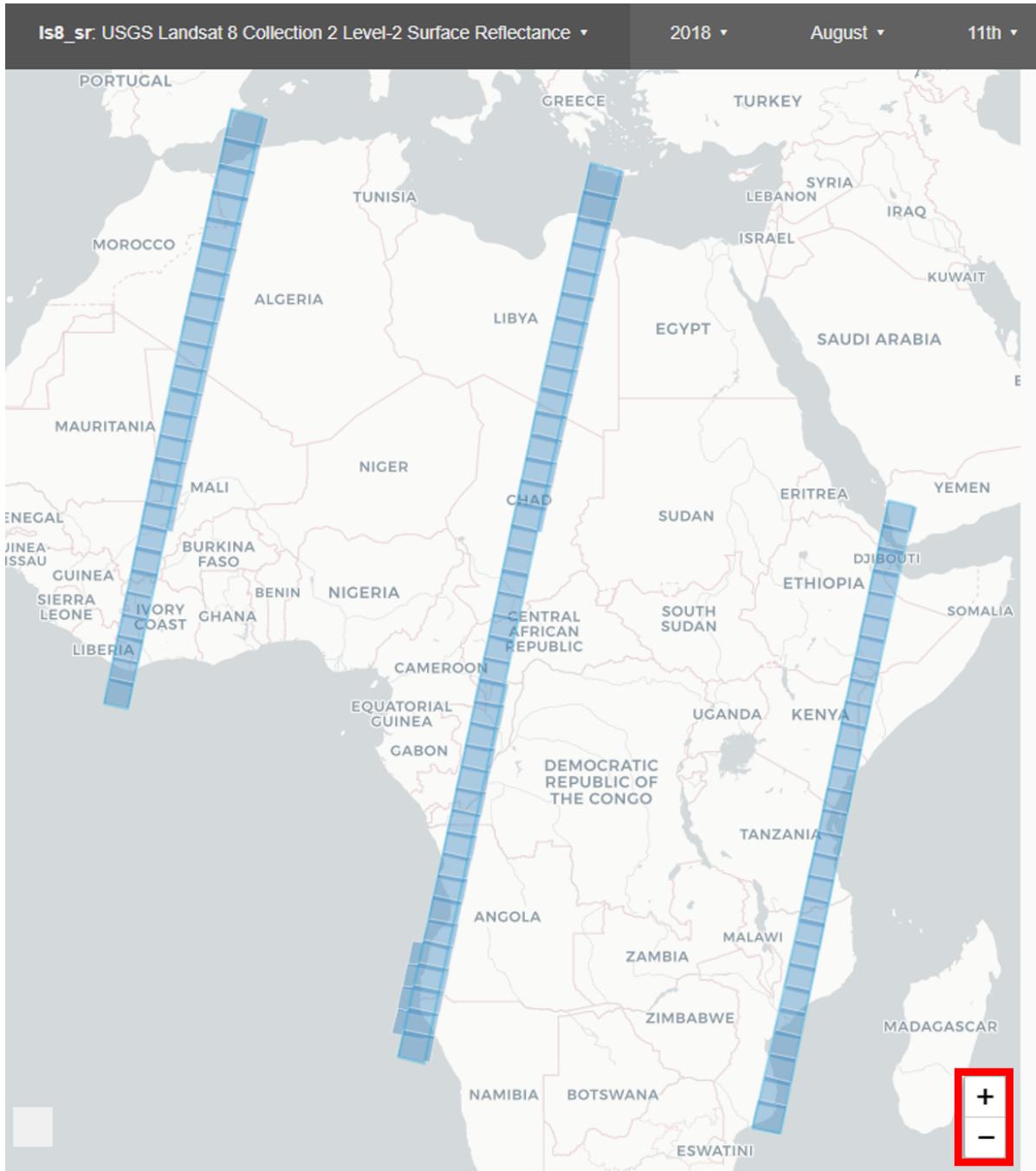
3. Click the **all months** bar to open the dropdown menu. Select **August**. This will show all the Landsat 8 datasets for August 2018.



4. Click the all days bar to open the dropdown menu. Select 11th. This will show all the Landsat 8 datasets for 11 August 2018.



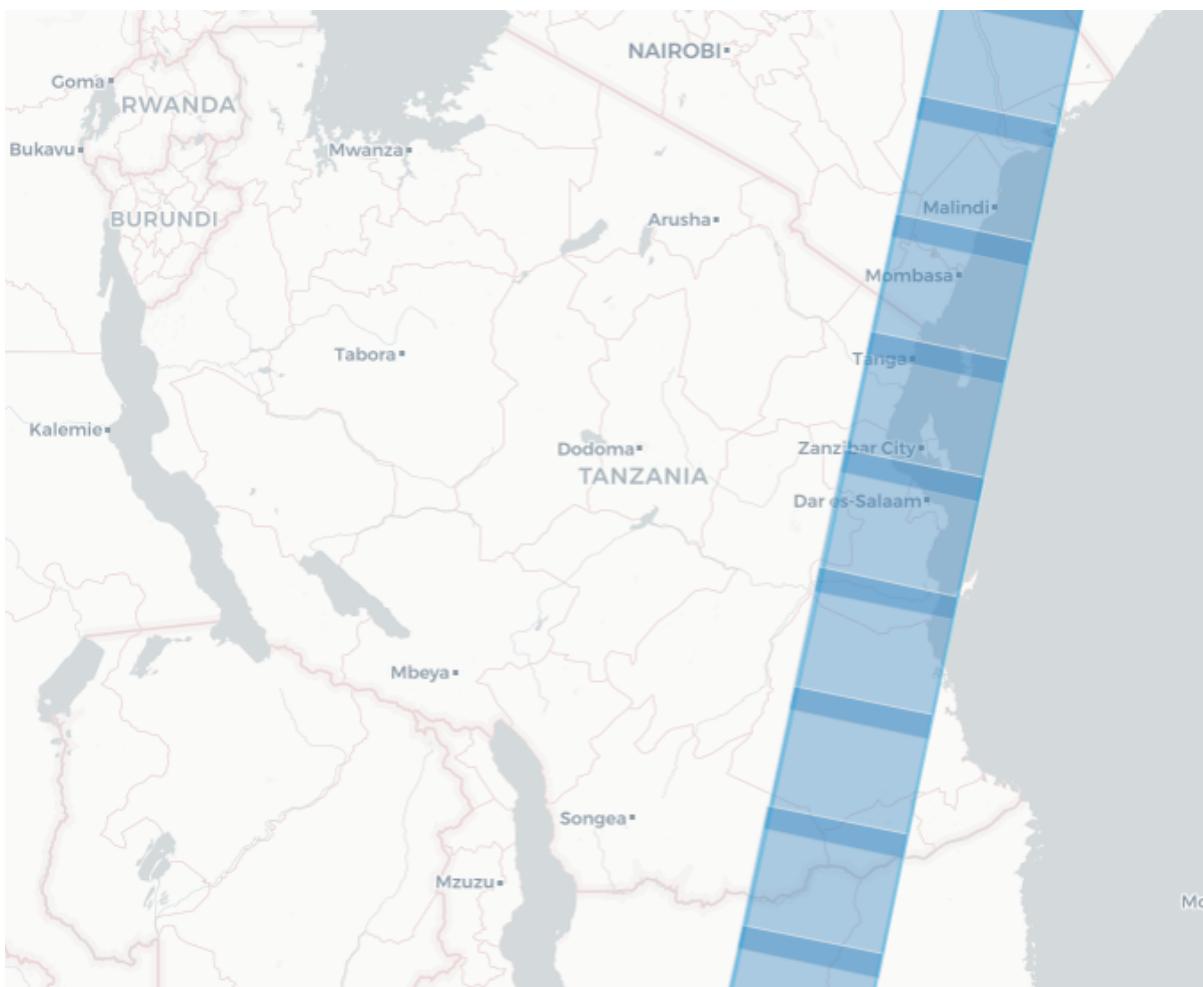
5. The map display will now show all the Landsat 8 datasets for 11 August 2018 as blue shaded boxes. Use the + button on the map to zoom in, and click and drag to pan the map.



We can see that there is only data available for some African countries. Let's take a closer look at Tanzania.

Zoom in on Tanzania

Use the map's + button to zoom in on Africa. Click and drag to see Tanzania.



We can see the data for 11 August 2018 (blue shaded boxes) covers certain parts of Tanzania, which means areas within Tanzania are a suitable location to choose for data analysis for that day. For example, for that day (11 August 2018) we would not be able to do any Landsat 8 analysis over the city of Dodoma in Tanzania, as it does not have any data available for that time.

2.3.4 Conclusion

You now know how to use the Digital Earth Africa Metadata Explorer to check where and when you can perform data analysis. The next step is loading the data.

2.4 Loading data in the Sandbox

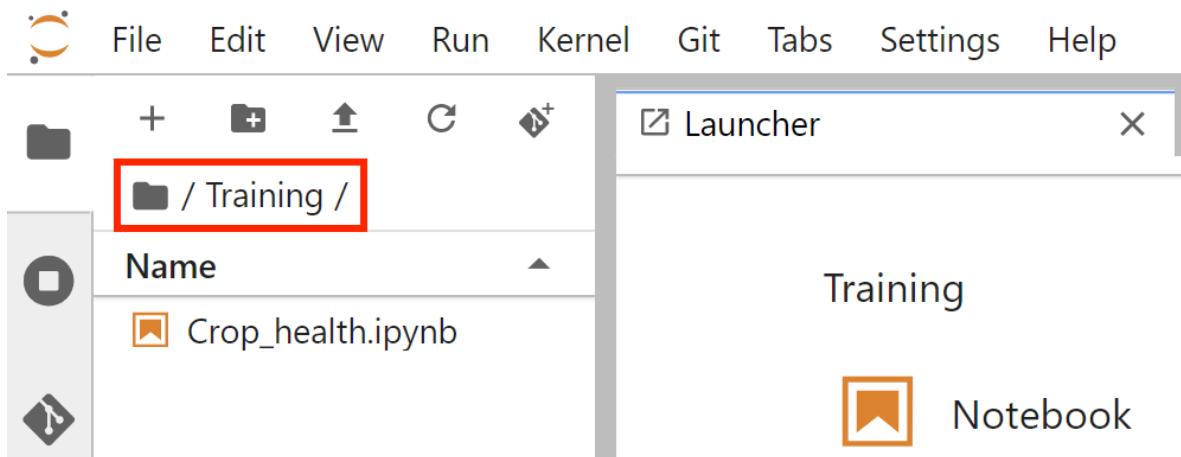
2.4.1 Overview

In this exercise, we will load data from the datacube. First, we will set up a new notebook to work in. Then, we will load Landsat 8 data for a specific time, and use that data to plot a colour image. Finally, we will show you how to modify the load process to load and plot Sentinel-2 data.

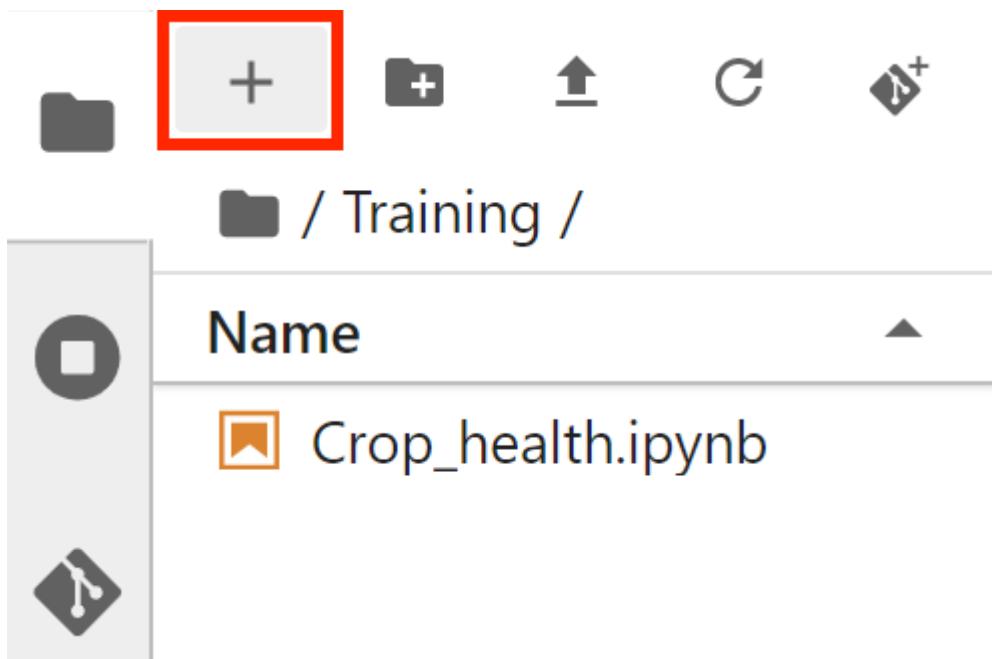
2.4.2 Make a new notebook

Let's create a new, blank Jupyter notebook for this exercise.

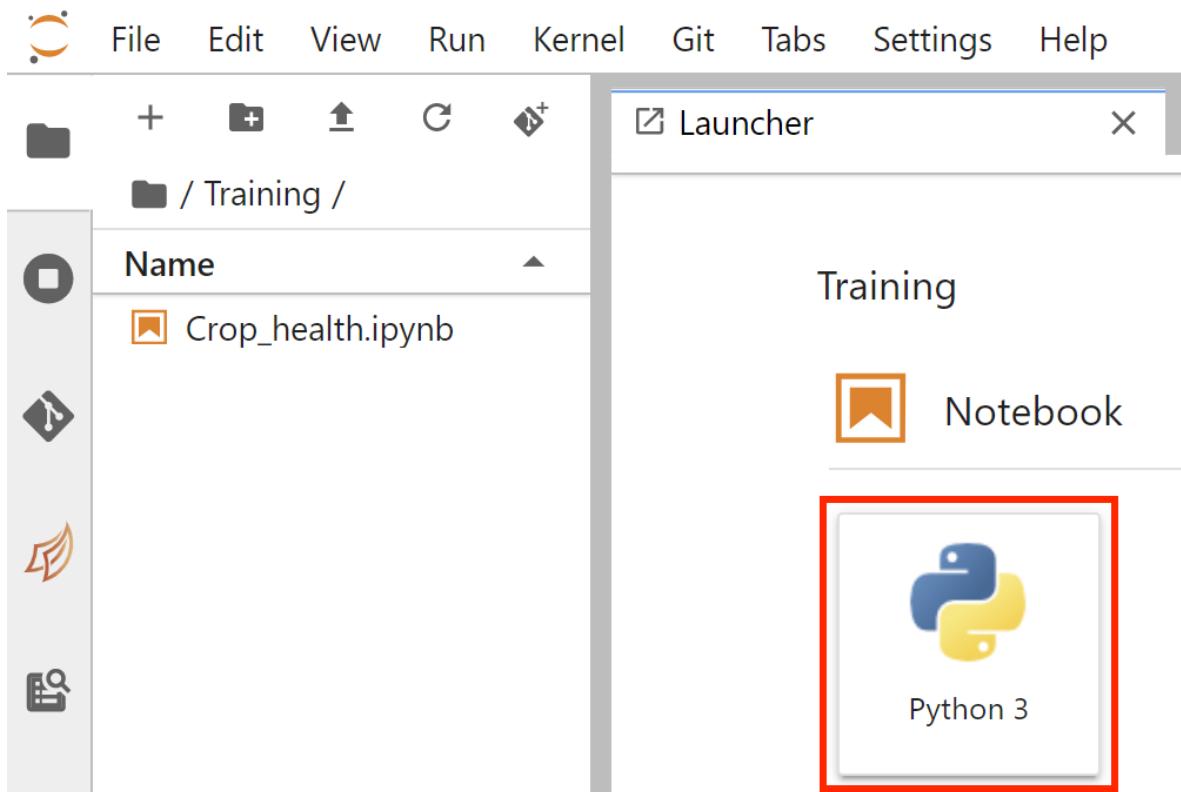
1. Navigate to the **Training** folder. The **Training** folder was created as part of Session 1, for copying and running the Crop Health notebook. If you do not have this folder in the Sandbox, you can create it by following the steps in [Running a Notebook](#).



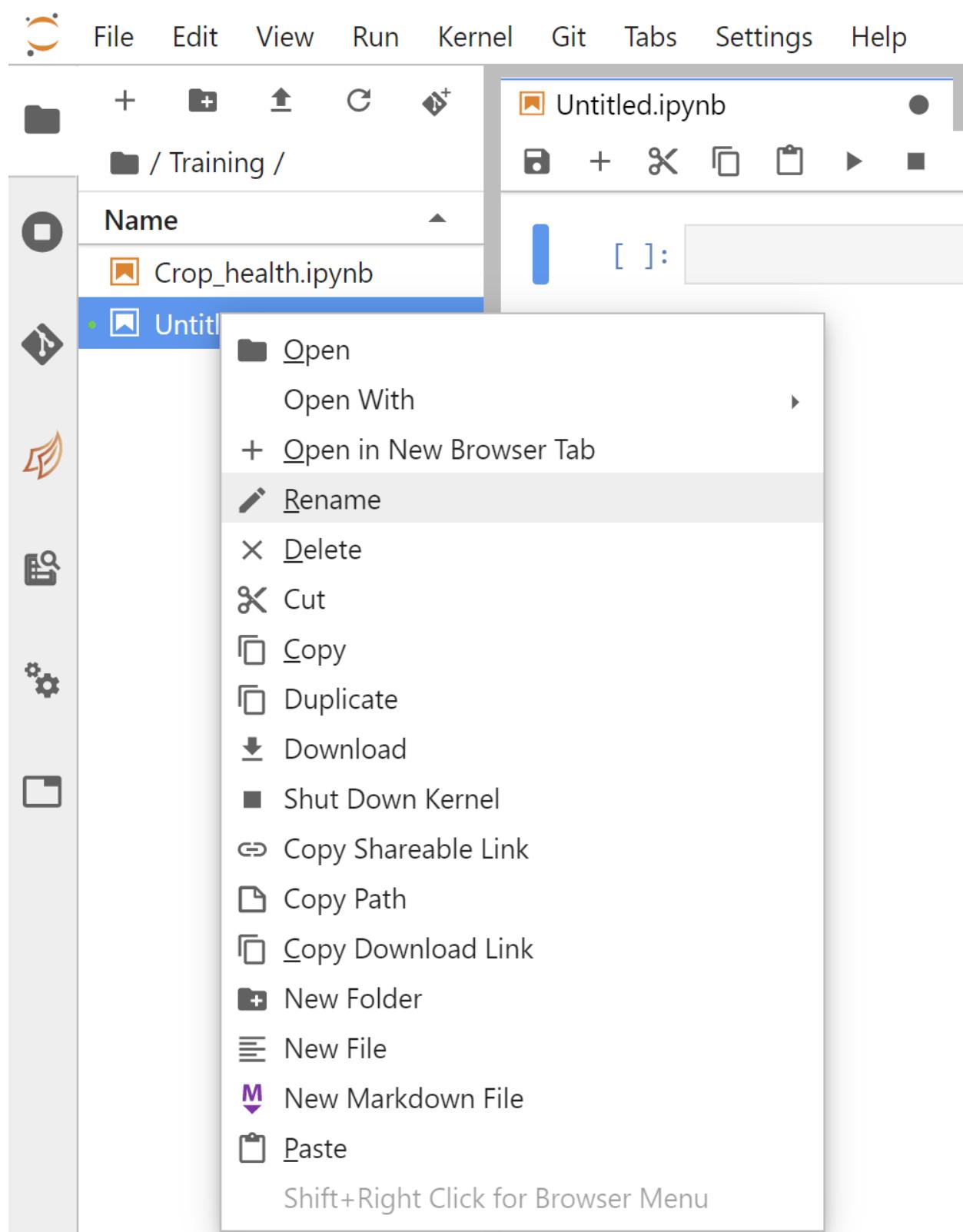
2. If Launcher is not the active tab in the main work area (right pane), click the + button at the top of the left sidebar to open the launcher in the right pane.



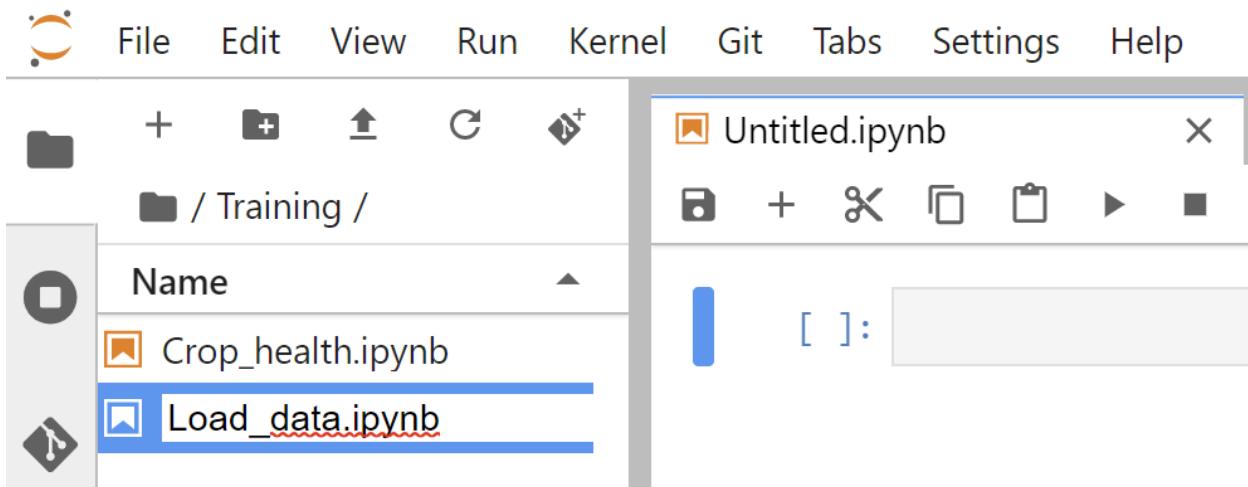
3. In the **Notebook** section of Launcher, select the **Python 3** option to create a new notebook in the current directory.



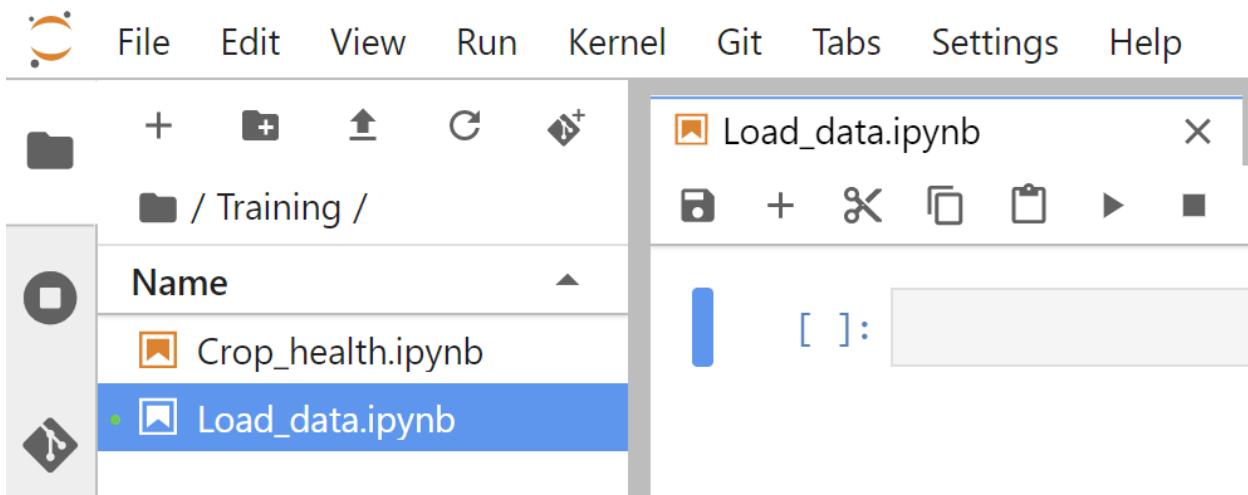
4. The new notebook will be called **Untitled.ipynb**, but you can rename it. Right-click the notebook in the file menu and select **Rename**.



Type in the desired name. For example, we can call it `Load_data.ipynb`.



Press the `Enter` key to finish renaming the notebook.



2.4.3 Set up notebook

Load packages and functions

Packages and functions act as the toolbox of Python programming. We will import the ones which will be useful to us. In the first cell, type the following code and then run the cell.

Note: Run a cell by pressing `Shift + Enter` on your keyboard.

```
%matplotlib inline

import datacube

from deafrica_tools.plotting import rgb
from deafrica_tools.plotting import display_map
```

- `%matplotlib inline` allows us to plot graphs and maps
- The package `datacube` is imported to allow us to create an object that can retrieve data from the datacube, which we will do in the next cell we create
- The package `deafrica_tools` contains several modules which help load, analyse and output data from Digital Earth Africa. Here we call upon the module `deafrica_tools.plotting` to import the `rgb` plot function, which allows us to visualise data as true-colour (red-green-blue, or RGB) images

When the cell has finished running, it will show [1] next to it, and generate a new blank cell below it.

Note: As of June 2021, the `deafrica_tools` package replaces the deprecated `sys.path.append('.. /Scripts')` file import. For more information on `deafrica_tools`, visit the [DE Africa Tools module documentation](#).

Connect to the datacube

The `datacube` package allows us to access the data in the Sandbox. To use it, we must establish a connection with the datacube. Enter the following code and run the cell.

```
dc = datacube.Datacube(app="Load_data")
```

The `datacube.Datacube` class provides access to the datacube. We usually call objects of this class `dc`, as we have done here. The `app` parameter is a unique name for the analysis which is based on the notebook file name.

When the cell has finished running, it will show a [2] next to it, and generate a new blank cell below it.

2.4.4 Load Landsat 8 data

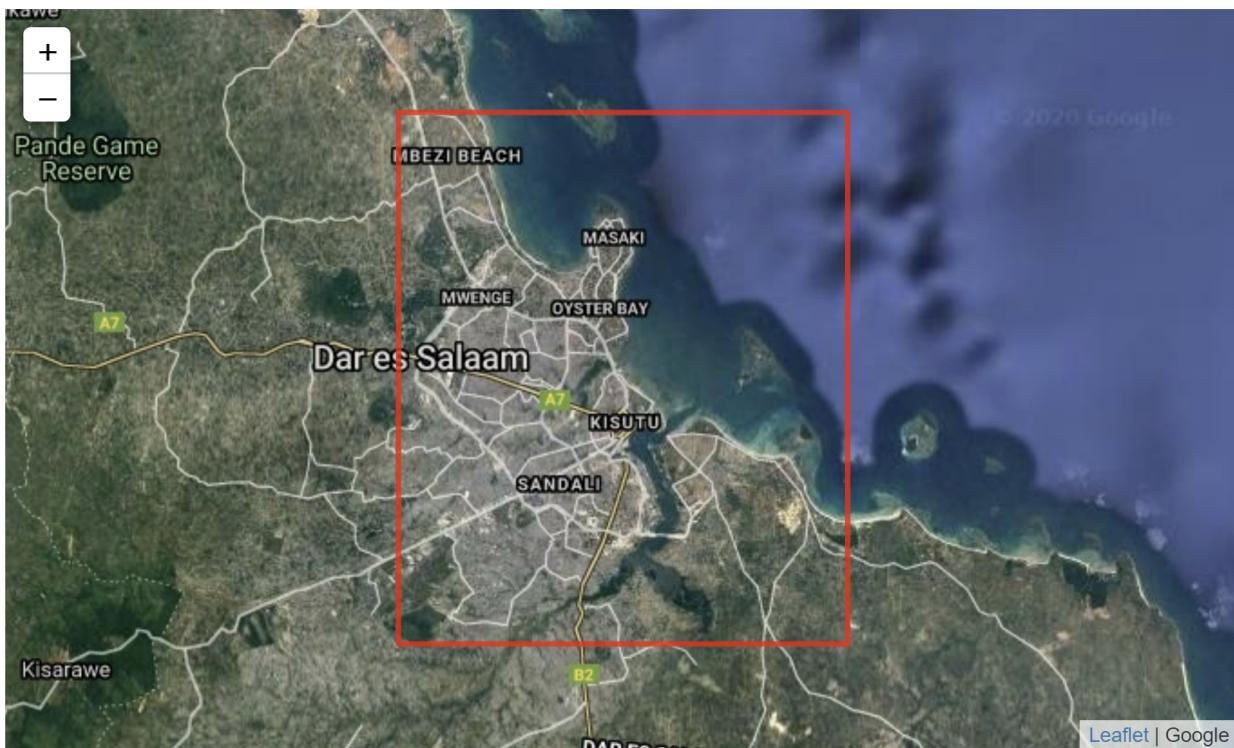
This exercise will load Landsat data for an area of Dar es Salaam, Tanzania. We will use a pair of latitude coordinates (-6.90, -6.70) and a pair of longitude coordinates (39.20, 39.37) to specify the area to load. Data will be loaded for the rectangle defined by these coordinate ranges.

First, we will view this area on a map. This allows us to check we have the correct coordinates. In the new cell below, enter the following code, and then run it to see this area on a map.

```
x=(39.20, 39.37)
y=(-6.90, -6.70)

display_map(x, y)
```

The output of that cell should look like this.



In the new cell below, enter the following code, and then run it to load Landsat 8 data.

```
landsat_ds = dc.load(
    product="ls8_sr",
    x=(39.20, 39.37),
    y=(-6.90, -6.70),
    time=("2018-01-01", "2018-12-31"),
    output_crs="EPSG:6933",
    resolution=(-30, 30),
    group_by="solar_day",
    measurements=["red", "green", "blue"])
```

We load data with the `dc.load()` function. We have chosen to call the loaded dataset `landsat_ds`. The text between the brackets of `dc.load()` are our parameters. We have chosen to put the parameters on separate lines to make the code easier to read (and errors easier to spot). Each parameter must be separated by a comma.

- The `product` argument is the datacube product to load data from. We want to access the Landsat 8 dataset, which is named `ls8_sr`.

`ls8_sr` uses only numbers and lowercase letters. It stands for **Landsat 8 Surface Reflectance**.

- The `x` and `y` arguments specify the area to load data for. In this case, they represent longitude and latitude. This defines a rectangle spanning their ranges of coordinate values as seen in the `display_map` output above.
- The `time` argument specifies the time range of data to load. We have specified all of the year of 2018.
- The `output_crs` argument specifies the Coordinate Reference System (CRS) to load data in. The CRS EPSG: 6933 specifies an equal area projection — each pixel has the same area.
- The `resolution` argument is the `y` and `x` resolutions (in that order) in pixels per degree. The first value is typically negative. In this case, a `resolution` of `(-30, 30)` is a resolution of 30 metres per pixel, which is the maximum resolution of Landsat data.

- The `group_by` argument controls how data that is close in time is combined to provide better images. Specifying a value of '`solar_day`' is recommended.
- The `measurements` argument specifies what bands will be loaded. We will plot a true-colour image of this data later. To do that, we need the red, green, and blue bands.

Note: As of June 2021, DE Africa Landsat data has been upgraded to Collection 2. Datacube names have been updated to `ls5_sr`, `ls7_sr` and `ls8_sr`. Deprecated naming conventions such as `ls8_usgs_sr_scene` will no longer work. For more information on Landsat Collection 2, visit the [DE Africa Landsat documentation](#).

Troubleshooting code

Sometimes, typing mistakes can occur. This will produce an error message when you run the cell.

For example, this error is a `SyntaxError`.

```
File "<ipython-input-5-d826e20eef48>", line 3
    x=(39.20, 39.37),
    ^
SyntaxError: invalid syntax
```

It tells us there might be a mistake just before the section of the code `x=(39.20, 39.39),`. Sure enough, this error message was generated when a comma was missing after the `product` parameter, as shown in the screenshot below.

For illustrative purposes, the point where the comma is missing has been highlighted by a red box, but this will not appear in JupyterLab — you will have to find the error or errors yourself.

```
landsat_ds = dc.load(
    product="ls8_sr" □
    x=(39.20, 39.37),
    y=(-6.90, -6.70),
    time=("2018-01-01", "2018-12-31"),
    output_crs="EPSG:6933",
    resolution=(-30, 30),
    group_by="solar_day",
    measurements=["red", "green", "blue"]))
```

If errors such as `IndentationError` or `SyntaxError` appear, they must be resolved before you can continue. Try checking for some of these common issues:

- Are all brackets and quotation marks in the right place?
- Does every open bracket have a corresponding close bracket?
- Do your bracket types match? (must be closed by) and [with], and they have different meanings in Python, so they are not interchangeable.
- Does every opening quotation mark have a closing quotation mark? You can use either ' or ", but pairs of quotation marks must be the same.
- Are there commas , between items listed in square brackets [] or parentheses ()?
- Is the indentation correct? Press Tab on your keyboard to increase the level of indentation, and press Shift + Tab on your keyboard to decrease the level of indentation.
- Is everything spelt correctly?

Once you have made your changes, try executing the cell again, by pressing Shift + Enter on your keyboard.

If you get a `NameError`, it could be because you have not yet imported the required packages and functions. They must be imported every time you start a new server session. To resolve this, follow the instructions in the section above, **Load packages and functions**.

```
-----
NameError                                 Traceback (most recent call last)
<ipython-input-2-1dbe76b9608a> in <module>
----> 1 dc = datacube.Datacube(app="Load_data")

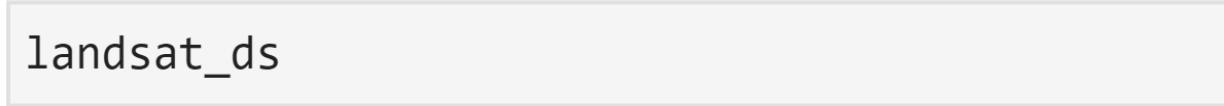
NameError: name 'datacube' is not defined
```

An example of a `NameError` caused by not importing the `datacube` package.

Note: Take your time to type code. If you would like to learn more about Python code syntax, or more chances to practise basic Python skills, take a look at the optional extra session [Python basics](#).

2.4.5 Examine data

When the `dc.load()` cell successfully executes, it will create a new cell below it. In this new cell, we can enter the name of our dataset and run the cell. This will show the dataset we loaded.



landsat_ds

The output of the cell should look similar to this:

xarray.Dataset

► Dimensions:	(time : 21, x : 548, y : 845)	
▼ Coordinates:		
time	(time) datetime64[ns] 2018-01-15T07:32:04.349192 ... 2...	 
y	(y) float64 -8.529e+05 ... -8.782e+05	 
x	(x) float64 3.782e+06 3.782e+06 ... 3.799e+06	 
spatial_ref	() int32 6933	 
▼ Data variables:		
red	(time, y, x) int16 -9999 -9999 -9999 ... 6323 6968	 
green	(time, y, x) int16 -9999 -9999 -9999 ... 6216 6809	 
blue	(time, y, x) int16 -9999 -9999 -9999 ... 6070 6696	 
▼ Attributes:		
crs :	EPSG:6933	
grid_mapping :	spatial_ref	

The output of `dc.load()` is an `xarray.Dataset` object. This type of dataset is a common format for satellite data, and is organised by:

- **Dimensions:** The dimensions of the dataset. For Earth observation data, this is often `x` (longitude), `y` (latitude) and `time`, as seen here. The units for the `x` and `y` dimensions are pixels, while `time` is counted in number of flyovers. In this example we see there were 21 flyovers of our selected location during the year of 2018.
- **Coordinates:** A list of the values of each dimension. `spatial_ref` refers to the CRS we selected in `dc.load()`.
- **Data variables:** The data values for our chosen measurements. We see `red`, `green` and `blue` are loaded as we specified in the `dc.load()` command. This product provides values for surface reflectance, which is unitless.
- **Attributes:** Metadata about this dataset. The CRS is listed again.

2.4.6 Plot a true-colour image

True-colour images are also known as red-green-blue (RGB) images. They are rendered using the image's 'natural' colours and appear how they might be seen by the human eye. As we loaded `red`, `green` and `blue` bands from Landsat 8, we can now plot an RGB image using the data from `landsat_ds`.

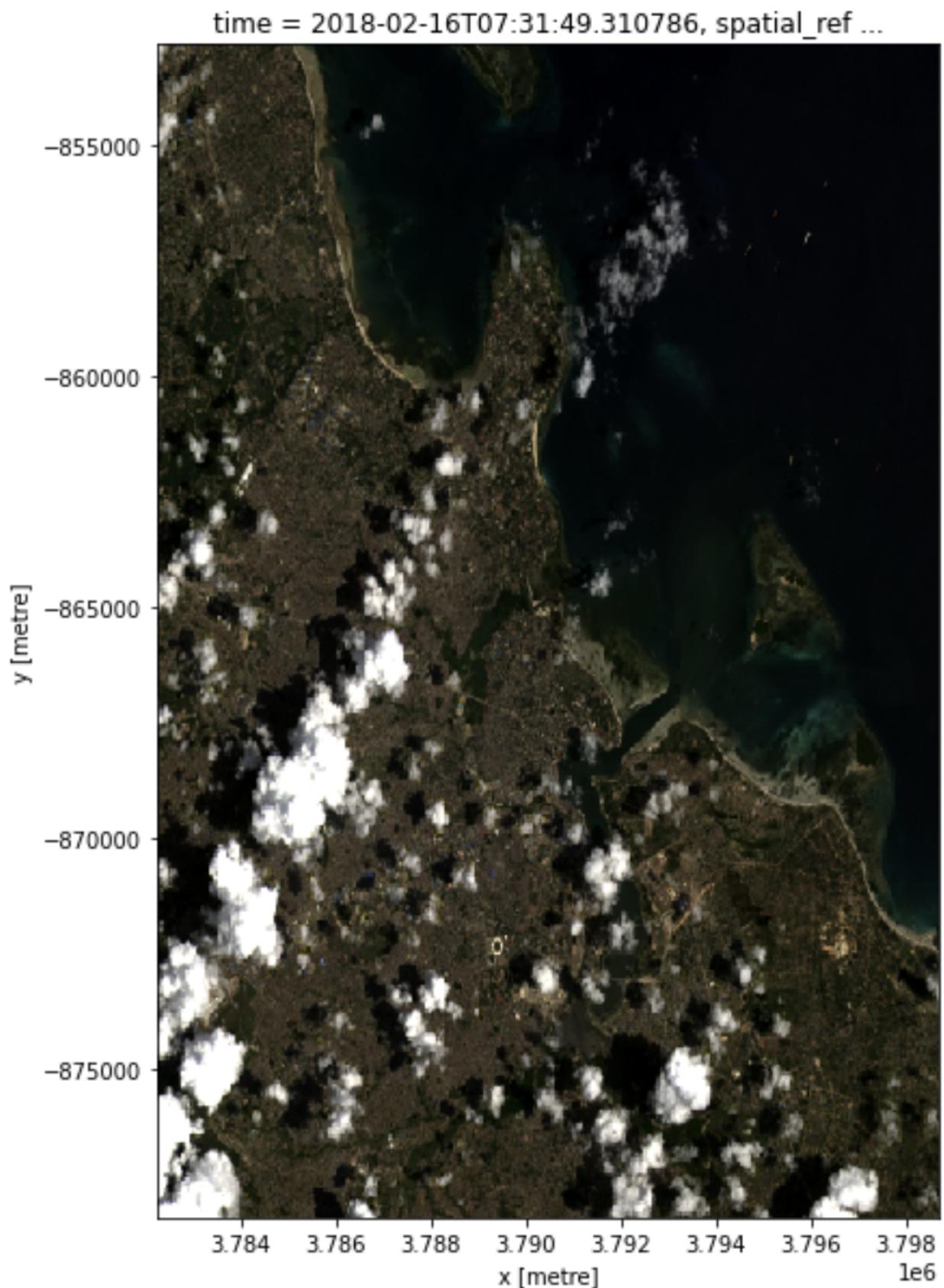
In the next blank cell, enter the following code. Run the cell to generate an RGB image.

```
rgb(landsat_ds, bands=['red', 'green', 'blue'],
    index=2, size=10)
```

The function used here is called `rgb()`.

- The first item inside the `rgb()` brackets is the name of the dataset we are drawing the data from. In this case, we want to pull information from `landsat_ds`.
- `bands` specifies the name of the data variables in the dataset that correspond to red, green and blue. We saw above that in `landsat_ds` they are conveniently named `red`, `green` and `blue`.
- `index` refers to the timestep to view. The default is 0. The Python language counts from 0, so `index=0` shows the first flyover, and `index=1` the second.
- `size` is the height of the image.

The RGB image will look like this:



The title of the image notes that the date for this data is 2018-02-16, or 16 February 2018.

2.4.7 Exercise: Load and plot Sentinel-2 data

Let us repeat the data loading process for Sentinel-2 data. It is a very similar process to loading the Landsat 8 data. We want to load data for the same time and place, so we only have to change the product and resolution.

1. Let us call our Sentinel-2 dataset `sentinel_2_ds`. You must name it something different from the Landsat 8 dataset. In a new cell, type the name of the Sentinel-2 dataset.

```
sentinel_2_ds
```

2. Again, we will use `dc.load()` to import Sentinel-2 data. After `sentinel_2_ds`, type `= dc.load()`. It should look like `sentinel_2_ds = dc.load()`.

How do we fill out the parameters inside the brackets of `dc.load()`? We can do this by copying some of the information from the Landsat 8 `dc.load()` input cell. The first parameter we listed before was `product`. However, we don't want to use the Landsat 8 product, we want to select the Sentinel-2 product, `s2_l2a`.

```
product="s2_l2a",
```

Note: `s2_l2a` stands for **Sentinel-2 Level-2A**. The fourth character is a lowercase alphabet ‘l’. Double-check you have entered the product name correctly to avoid errors.

3. The `resolution` parameter will also be different from the Landsat 8 load. For Sentinel-2, it should be `(-10, 10)`, since our Sentinel-2 data has a resolution of 10 metres per pixel.

```
resolution=(-10,10),
```

4. Now, type the rest of the parameters to be the same as they were for the Landsat 8 load. This includes:

- `x`
- `y`
- `time`
- `output_crs`
- `group_by`
- `measurements`

As before, watch out for commas, quotation marks, and brackets to avoid error messages when running the cell.

5. You should end up with a set of parameters that look like this:

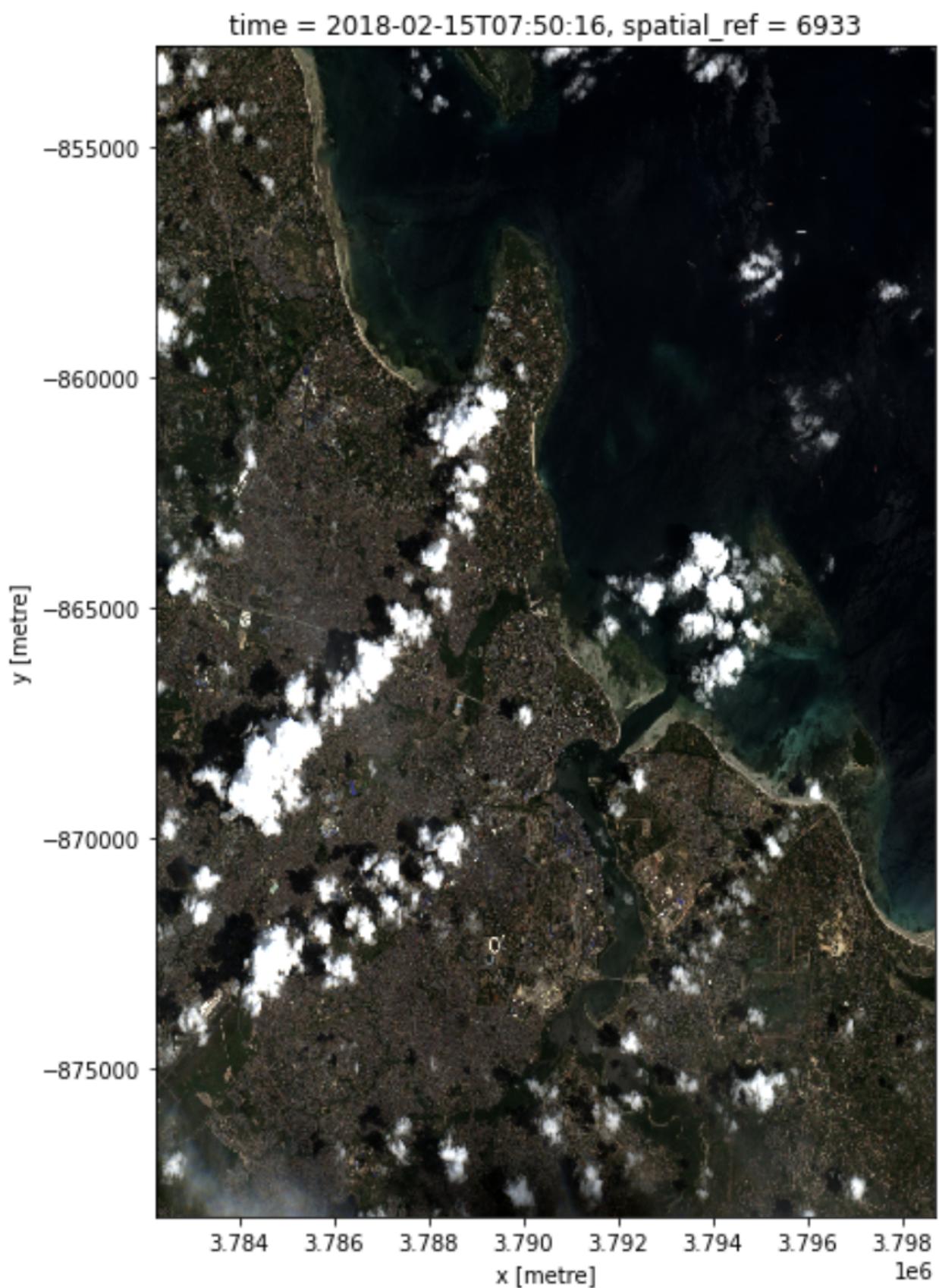
```
sentinel_2_ds = dc.load(  
    product="S2_L2A",  
    x=(39.20, 39.37),  
    y=(-6.90, -6.70),  
    time=("2018-01-01", "2018-12-31"),  
    output_crs="EPSG:6933",  
    resolution=(-10, 10),  
    group_by='solar_day',  
    measurements=['red', 'green', 'blue'])
```

6. Run the cell to load Sentinel-2 data.
7. In the new cell below that, let us plot an RGB image around the same time as the Landsat 8 RGB image, which was from February 16, 2018.

We must specify the dataset first, followed by the bands, index, and size. In this case, we want to use `index=9`. Ensure the cell has the following code and then run it.

```
rgb(sentinel_2_ds, bands=['red', 'green', 'blue'],  
    index=9, size=10)
```

8. An RGB image using Sentinel-2 data will be generated.



As Landsat 8 data and Sentinel-2 data come from different satellites, their flyovers are not always at the same time. In this case, the closest date of Sentinel-2 data to the Landsat data is one day before, on February 15, 2018. It is another cloudy scene, like the Landsat 8 one.

2.4.8 Conclusion

You have successfully loaded and plotted data for Landsat 8 and Sentinel-2.

You have also finished the second session of the Digital Earth Africa training course. In this session, you have learned about:

- Digital Earth Africa products, including Landsat 8 and Sentinel-2
- Visualising data with the [Digital Earth Africa Map](#)
- Exploring data availability with the [Digital Earth Africa Explorer](#)
- Loading data in the Sandbox
- Generating RGB images

Congratulations!

2.5 Session 2 Quiz and Solution

In Session 2, you constructed a notebook to load satellite products and visualise the loaded data as true-colour images. This is often an important step when working with satellite data as it provides a first glimpse of the data, which can help direct your analysis.

2.5.1 Quiz

If you would like to be awarded a certificate of achievement at the end of the course, we ask that you [complete the quiz](#). You will need to supply your email address to progress towards the certificate. After you complete the quiz, you can check if your answers were correct by pressing the **View Accuracy** button.

The quiz will ask you to use the notebook you developed for this session's exercise on loading data. If you would like to confirm that your notebook works as expected, you can check it against the solution notebook provided below.

Note: The solution notebook below does not contain the answer to the quiz. Use it to check that you implemented the exercise correctly, then use your exercise notebook to answer the quiz. Accessing the solution notebook will not affect your progression towards the certificate.

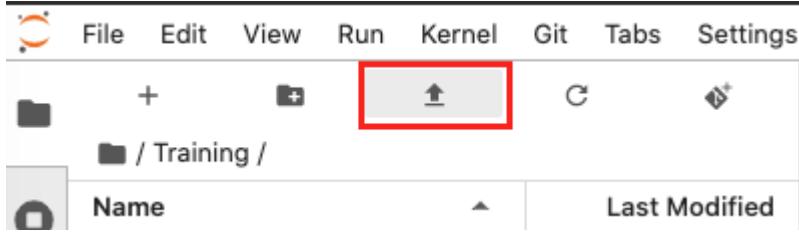
2.5.2 Solution notebook

Note: We strongly encourage you to attempt the exercise on the previous page before downloading the solution below. This will help you learn how to use the Sandbox independently for your own analyses.

[Download the solution notebook for the Session 2 exercise](#)

To view this notebook on the Sandbox, you will need to first download it to your computer, then upload it to the Sandbox. Follow these instructions:

1. Download the notebook by clicking the link above.
2. On the Sandbox, open the **Training** folder.
3. Click the **Upload Files** button as shown below.



4. Select the downloaded notebook using the file browser. Click **OK**.
5. The solution notebook will appear in the **Training** folder. Double-click to open it.

SESSION 3: COMPOSITES

In this session, we will explore the advantages of having satellite data available over a long period of time. We can fill in missing or bad data from one day with data from another day. This creates a complete image known as a **composite**. The most scientifically rigorous way of combining data is to use a **geomedian**.

First, we will introduce composites and geomedians. Then, the exercise will walk through removing cloudy data points from our datasets, and show you how to produce cloud-free geomedian composite images.

3.1 Introduction to cloud-free composites

3.1.1 Overview

This session is about creating representative datasets and images from multiple timesteps. This allows us to remove and replace unwanted data, such as clouds, and also form images that accurately represent the area of interest over a period of time.

This is summarised in this week's video, **Time aggregation of data**. Watch the video to see how to use Earth observation data at different points in time.

3.1.2 Video: Time aggregation of data

Download the video slides as a PDF

In the video, we saw that we can compensate for missing or cloudy data points by using data from different points in time to fill in the gaps. This is a two-step process:

1. Identify and remove cloudy data — this is known as ‘cloud masking’
2. Use data from a different time to fill in missing data — this can be done by calculating geomedians

In this section, we focus on why cloud masking is an important step in preparing your dataset, and introduce the easiest way to do this in the Sandbox. We then briefly explain the significance of geomedians compared to other statistical values.

The two pages following this introduction will involve walkthrough exercises, so you can try performing these steps yourself after reading about them in this section.

3.1.3 Recap: loading and plotting data

In the [last session](#), we plotted RGB images for Dar es Salaam in Tanzania. The image had clouds in both the Landsat 8 and Sentinel-2 versions.



Landsat 8 data from 16 February 2018 (left), and Sentinel-2 data from 15 February 2018 (right). Some cloud cover is visible in both images.

What if we want to know what is underneath those clouds? If you have data at only one point in time, this is not possible. However, if we have data for the same place at a different time, when the clouds are not present, we can use this data to ‘fill’ in areas of cloud.

To do that, we must first identify which pixels are clouds. The process of determining and removing cloud data points is known as ‘cloud masking’.

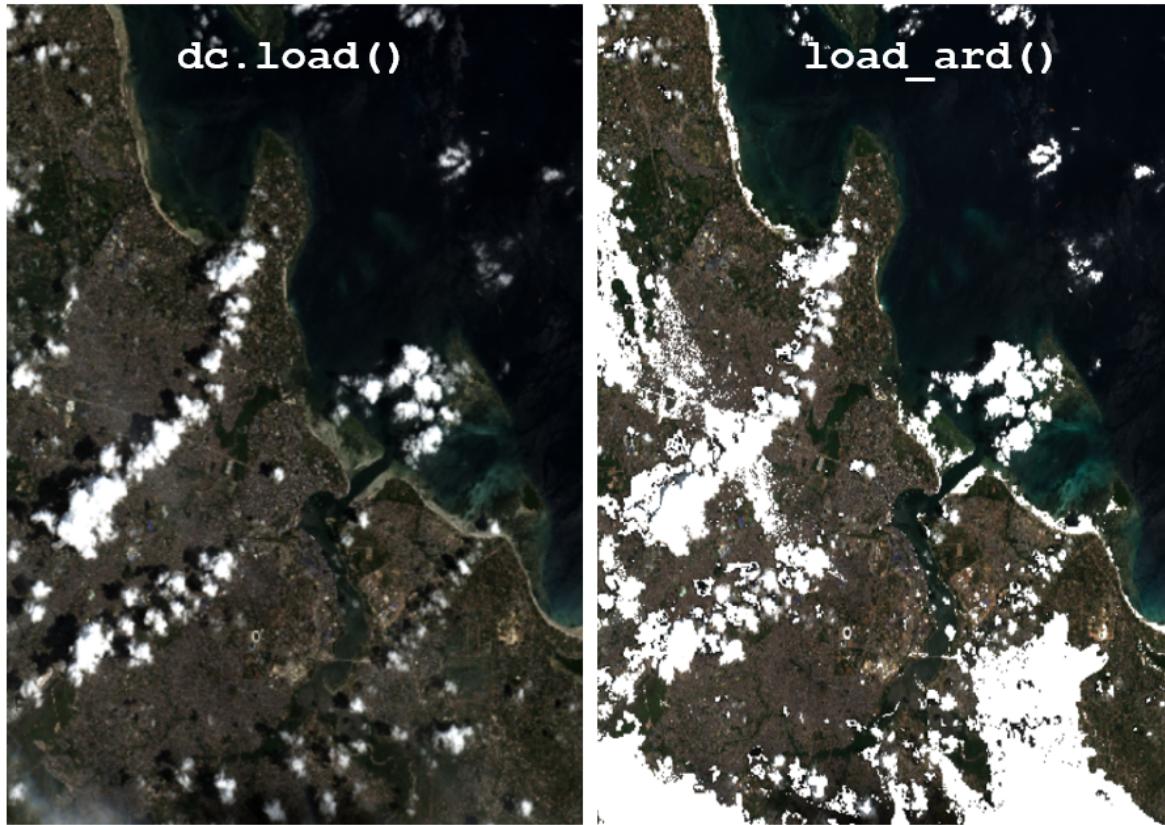
3.1.4 `load_ard()` vs `dc.load()`

The easiest way to apply a cloud mask to your dataset is to load it into the Sandbox using the `load_ard()` function. `ard` stands for ‘Analysis-Ready Data’ and the `load_ard()` function automatically applies a cloud mask.

We previously loaded data in the Sandbox using `dc.load()`. `dc.load()` is a universal function for loading data from the datacube and it is important to know how to use it. However, it does not have a built-in cloud masking capability.

When we plotted the RGB images of Tanzania with data loaded using `dc.load()`, the clouds were part of the image. This makes it difficult to perform cloud masking, as the dataset does not distinguish cloud and not-cloud.

In this exercise, we will instead use the command `load_ard()` to load our data. It demands similar parameters to `dc.load()`, but automatically identifies pixels of cloud, and applies cloud masking to the loaded data.



Sentinel-2 data from 15 February 2018 loaded with `dc.load()` (left) and `load_ard()` (right). The `dc.load` image shows cloud cover, while the white pixels in the `load_ard` image are not clouds, but points where data has been removed by the cloud mask.

The cloud masking algorithm on Sentinel-2 data is more aggressive than its Landsat 8 counterpart. This means it sometimes misinterprets white sand beaches or urban regions as cloud. This can reduce the amount of data available.

Note: `load_ard()` is compatible with both the Sentinel-2 dataset and the Landsat 8 dataset we used in the last session. However, it is not compatible with some other Digital Earth Africa products, such as Water Observations from Space or GeoMAD, with which you will need to use `dc.load()`.

3.1.5 Calculating a composite

Now that we know how to mask out clouds by using `load_ard()`, we can load multiple timesteps of our cloud-masked data. These need to be combined in a meaningful way to produce a single image. We do this by compositing our data. Compositing creates one value for each band for each pixel based on the time series data for that pixel.

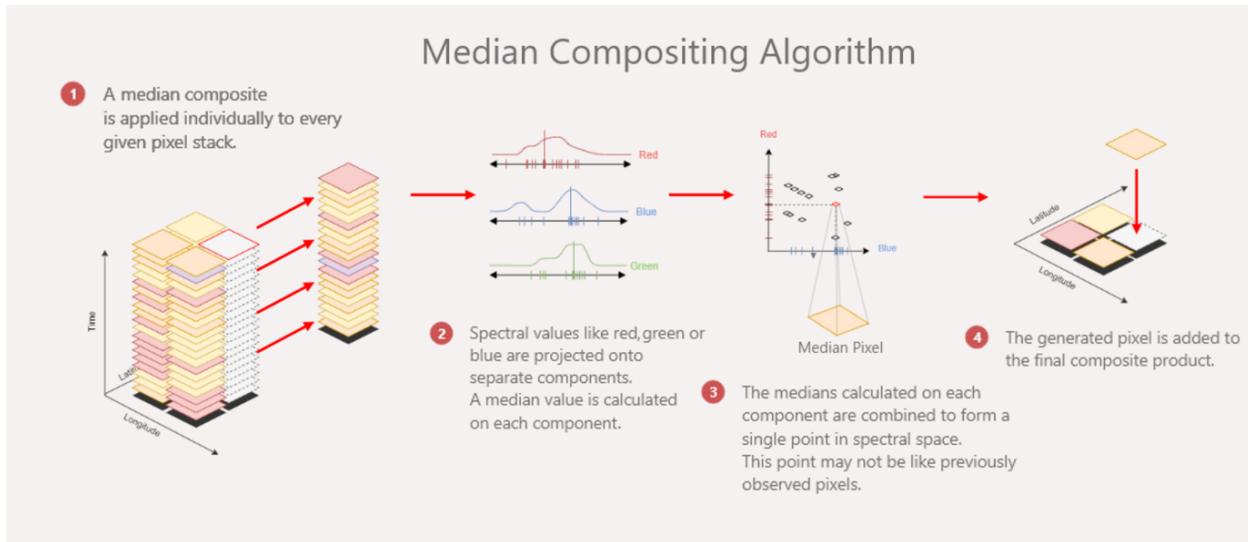
We will briefly compare median and geomedian composites, and explain why it is more reliable to use geomedians.

Median composites

For each band in the image, median composites set the value of each pixel to the median value for that band. For a given pixel, each band's median is independent of the others.

The benefit of a median composite is that it is very fast to compute, so it can be used to quickly create cloud-free images for an area.

However, medians do not account for the fact that every pixel holds information for multiple bands. It is therefore better to use a statistic that is configured for multi-dimensional data, such as a geomedian.

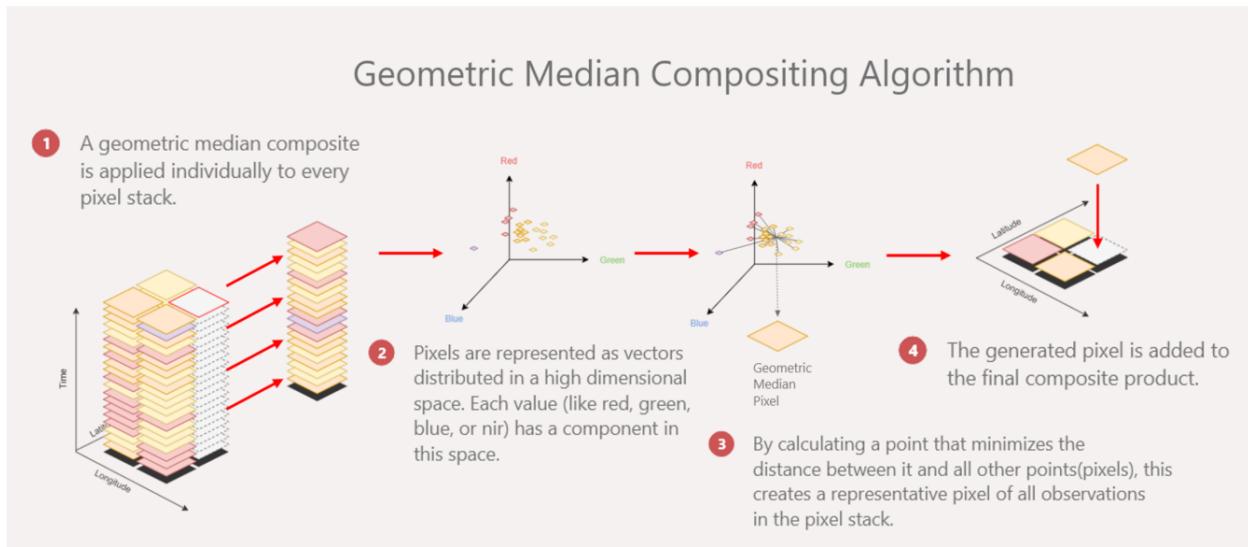


A median considers data from each band independently. This can be seen in Step 2 of the median compositing algorithm shown above.

Geomedian composites

Geomedian — or ‘geometric median’ — composites are multi-band generalisations of median composites. Instead of finding a pixel’s median value for each band **individually**, like a median composite does, a geomedian composite finds the median values of the bands for each pixel when considered **together**.

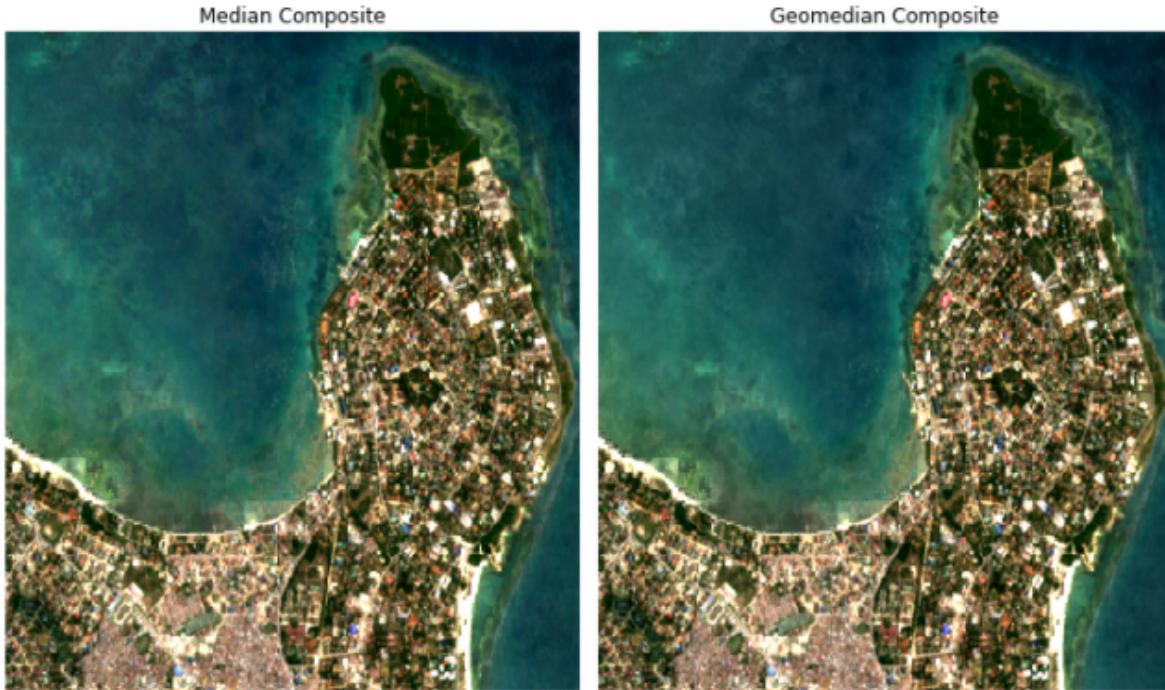
This means they represent the data **better** than median composites.



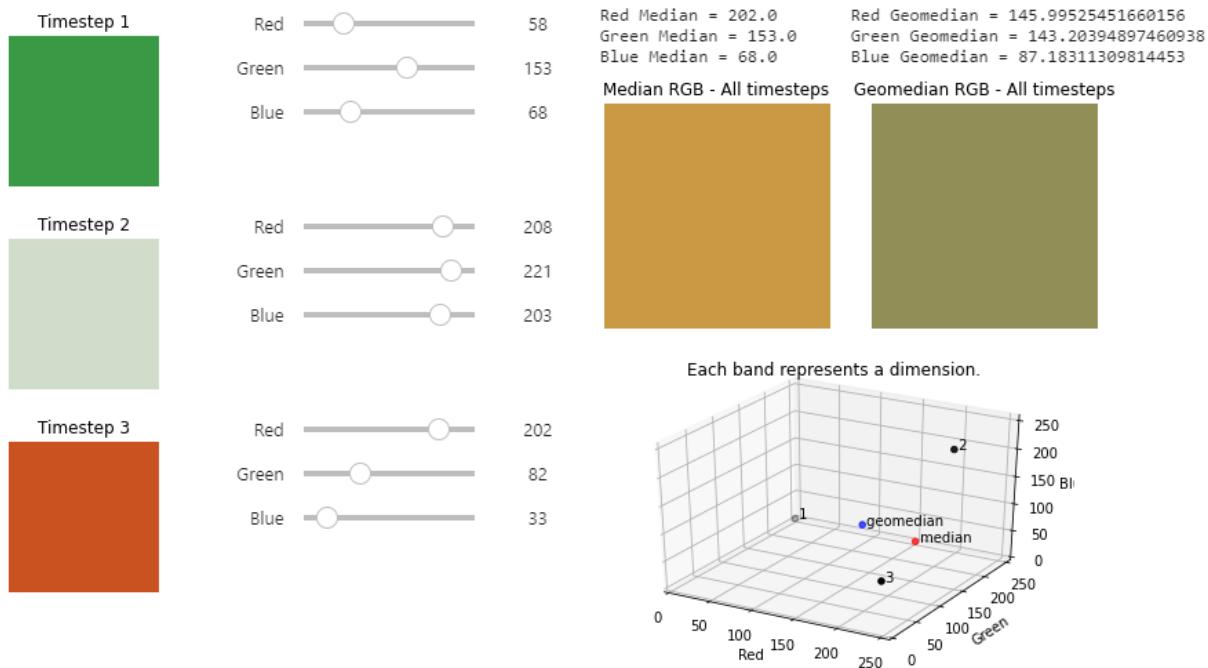
Unlike the median composite, the geometric median (geomedian) considers all bands of data together. Each band adds a dimension to the geomedian calculation. For a three-band dataset, such as the RGB dataset shown in the geomedian compositing algorithm above, each point can be represented on a three-dimensional scatter plot. The geomedian is the minimised ‘sum of distances’ between all the points.

3.1.6 Comparing medians and geomedians

The difference between medians and geomedians can often be subtle, especially if you are looking at the overall composite image. For example, the RGB images for these median and geomedian composites look almost identical.



However, on a pixel-by-pixel basis, it is possible to visualise the difference between median and geomedian. The best way to do this is to use the **geomedian widget** to explore the effects of different datasets on the median and geomedian.



The geomedian widget uses interactive sliders to change the value of the data. This affects the output median and geomedian values. You will soon see that each compositing method can produce significantly different results. On a larger scale (like a whole image, or over several years), this can affect the end composite image.

Download the widget notebook here or visit the [Geomedian widget](#) page for more information, including instructions on running the widget in the Sandbox.

3.1.7 The bottom line: use geomedians

Geomedians take more processing time to calculate than median composites. However, unless you are only doing a quick visualisation, you should use the geomedian method when creating composites. This is because the geomedian value is more scientifically rigorous as it accounts for all the bands in the dataset.

Note: DE Africa produces a pre-computed annual geomedian as part of the GeoMAD service. We will not be using it in this training course, but it produces stunning cloud-free geomedian imagery which can be explored via [DE Africa Maps](#) or in the Sandbox. Take a look at the [GeoMAD documentation](#) to learn more.

3.1.8 Conclusion

You now know we can perform cloud masking using the `load_ard()` function, and that we should combine different timesteps of data using a geomedian calculation.

The exercise for this session is covered in the next two sections.

1. We will walk through the process of using `load_ard()` to load data with a cloud mask.
2. We will then use the loaded data to make and plot geomedians.

This technique will be useful in future sessions for conducting analysis on cloud-free images.

3.2 Cloud masking with load_ard()

In this exercise, we will apply a cloud mask to Sentinel-2 data off the coast of Ghana. Cloud masks are important as they remove bad data points from our dataset, so we can form a reliable composite image.

3.2.1 Make a new notebook

Like in the last exercise, we will begin by making a new, blank Jupyter notebook. If you want more detailed instructions on making a new notebook, see [this section in the exercise on loading data in the Sandbox](#) from the previous session. Otherwise, follow the steps below.

1. Navigate to the **Training** folder (or create this folder following the instructions in [Session 1](#)).
2. Click the + button and click **Python 3** under the **Notebook** section.
3. Rename your file so you know it is from this exercise. We will use this notebook for working with geomedians, so let us call it `Geomedian_composite.ipynb`.
4. Open the notebook.

3.2.2 Set up notebook

Load packages and functions

In the first cell, type the following code and then run the cell. Recall that cells can be run by pressing Shift + Enter on your keyboard.

```
%matplotlib inline

import datacube

from deafrica_tools.datahandling import load_ard
from deafrica_tools.plotting import rgb
from deafrica_tools.plotting import display_map
from odc.algo import xr_geomedian
```

We used most of these packages and functions in the previous exercise on loading data in the Sandbox. `rgb` is for plotting true-colour images. `display_map` is for visualising the area we have selected.

In this session we introduce two new functions: `load_ard` and `xr_geomedian`. We will use `load_ard` to load data so it is cloud masked, and `xr_geomedian` is used in the next section to compute the geomedian.

Note: As of June 2021, the `deafrica_tools` package has replaced the deprecated `sys.path.append('.. / Scripts')` file import. For more information on `deafrica_tools`, visit the [DE Africa Tools module documentation](#).

Connect to the datacube

Enter the following code and run the cell to create our dc object, which provides access to the datacube.

```
dc = datacube.Datacube(app="Geomedian_composite")
```

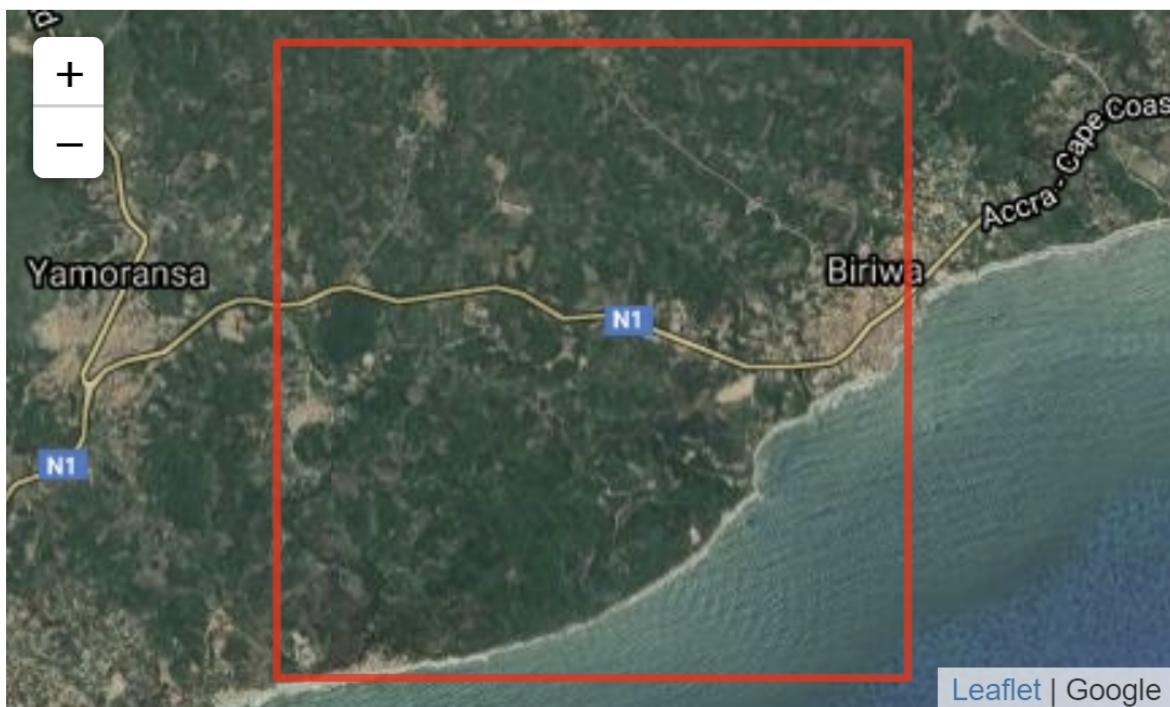
Your notebook is now set up. Next, we will load cloud-masked data using `load_ard()`.

3.2.3 Load data with `load_ard()`

Note: If you experience errors when running cells, check out the [troubleshooting code guide](#) from the previous session.

Let us take a look at a coastal area in Ghana. Enter the following code and run it to display a map of the area. As before, x denotes longitude and y denotes latitude.

```
x=(-1.15, -1.19)  
y=(5.14, 5.18)  
  
display_map(x, y)
```



In the new cell below, enter the following code, and then run it to load Sentinel-2 data. It will generate the output text
Using pixel quality parameters for Sentinel 2 The output text tells us we have loaded 4 timesteps.

```
sentinel_2_ds = load_ard(
    dc=dc,
    products=["s2_l2a"],
    x=x, y=y,
    time=("2018-02-01", "2018-03-15"),
    output_crs="EPSG:6933",
    measurements=['red', 'green', 'blue'],
    resolution=(-10, 10),
    group_by='solar_day',
    min_gooddata=0.7)
```

Using pixel quality parameters for Sentinel 2

Finding datasets

s2_l2a

Counting good quality pixels for each time step

Filtering to 4 out of 9 time steps with at least 70.0% good quality pixels

Applying pixel quality/cloud mask

Loading 4 time steps

Take note of some of the differences between `dc.load()` and `load_ard()`.

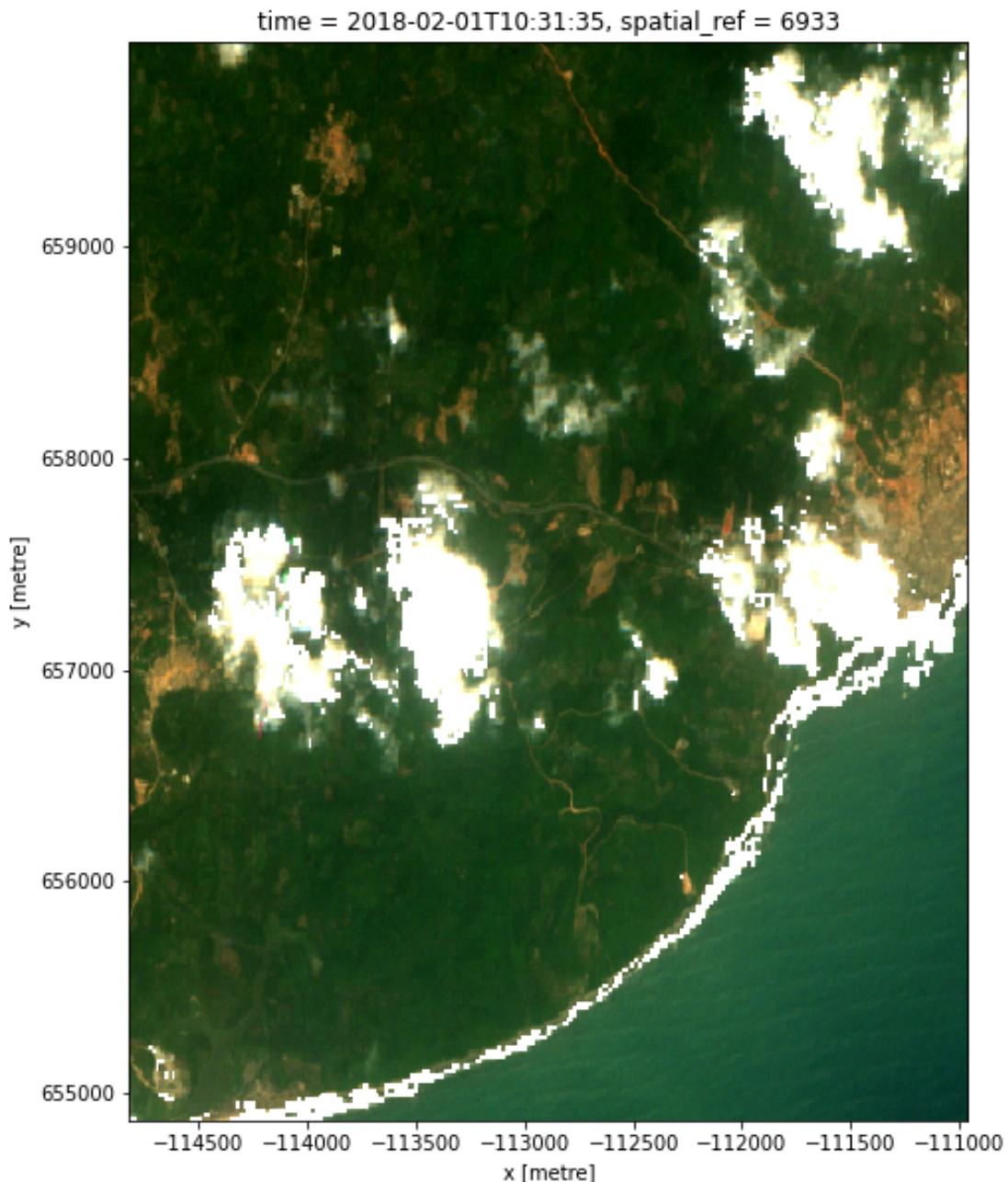
- `dc=dc` is a required parameter for `load_ard()`. This links the data search to the datacube connection, which we defined in the notebook setup as `dc`.
- The parameter for loading products is `products` (plural) not `product` as it is in `dc.load()`.
- Product items must be listed inside square brackets `[]`, which is not required for `dc.load()`.
- `min_gooddata` stands for ‘minimum good data’ and discards observations with less than the fractional requisite of good quality pixels.

Note: `s2_l2a` stands for **Sentinel-2 Level-2A**. The fourth character is a lower-case alphabet ‘l’. Double-check you have entered the product name correctly to avoid errors.

We can use the same `rgb` plotting code as in the last session to show an RGB image of one of the timesteps. Let’s start with the first timestep, which has an `index` of `0`.

```
rgb(sentinel_2_ds, bands=['red', 'green', 'blue'],
    index=0, size=10)
```

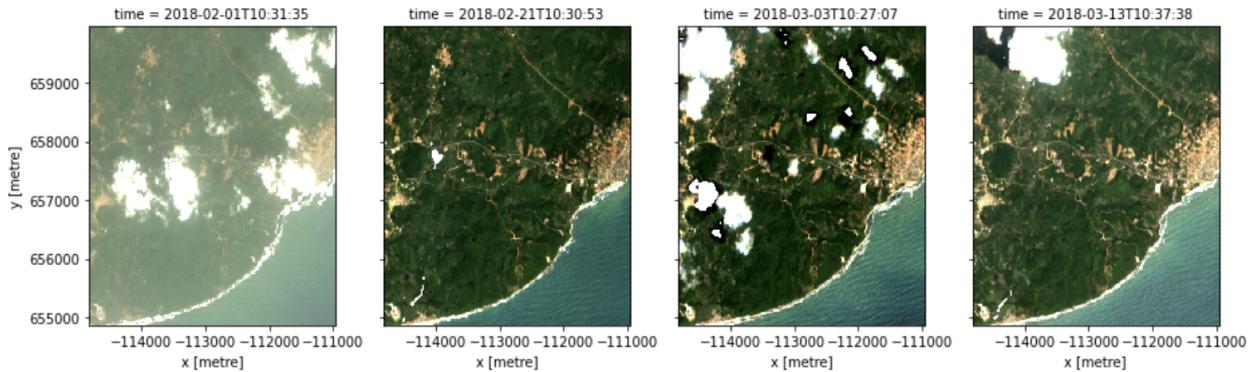
This should produce a single RGB image as shown below. What happens if you try changing the `index` number?



If we want to see RGB images of all the timesteps at once, we can replace the `index` parameter with the `col` parameter. The parameter `col` stands for ‘column’. Specifying `col='time'` creates a row of images for the timesteps.

```
rgb(sentinel_2_ds, bands=['red', 'green', 'blue'],
     col='time', size=4)
```

The output should look like this.



3.2.4 Conclusion

Good work — you have now loaded data using `load_ard()`, which has an automatic cloud mask. We can see that the images at different timesteps have different cloud cover, so they have been masked in different places. This is why having data at different timesteps can allow us to create a composite image without any cloud.

In the next section, we will use this loaded data to create a geomedian composite.

3.3 Create a geomedian composite

This exercise follows on from the previous section. In the previous section, we loaded Sentinel-2 data using `load_ard()`. This applied a cloud mask to the data by identifying and removing cloudy data points. We can now use this data to create a geomedian composite.

In this section, we will re-run the notebook from the previous section, and extend it to create a geomedian composite. First, we will open and run our `Geomedian_composite.ipynb` notebook. This will ensure it loads the Sentinel-2 cloud-masked data. Then, we will use a function called `xr_geomedian` to calculate geomedians. This will allow us to plot a cloud-free geomedian composite image.

Note: We will be using the notebook we created in the [previous section](#), **Cloud masking with `load_ard()`**. If you have not already set up a notebook called `Geomedian_composite.ipynb` with the required packages and functions, follow the instructions in previous section. Ensure you have completed all the steps, including loading the Sentinel-2 dataset for the Ghanaian coast.

3.3.1 Open and run notebook

If you are following directly on from the last section, you can skip this step. If you have closed your Sandbox browser tab or disconnected from the Internet between exercises, follow these steps to ensure correct package imports and connection to the datacube.

1. Navigate to the **Training** folder.
2. Double-click `Geomedian_composite.ipynb`. It will open in the Launcher.
3. Select **Kernel -> Restart Kernel and Run All Cells...**
4. When prompted, select **Restart**.

The notebook may take a little while to run. Check all the cells have run successfully with no error messages.

3.3.2 Check the Sentinel-2 dataset

Let's confirm we have successfully loaded our Sentinel-2 dataset. Recall we named it `sentinel_2_ds`. Scroll to the bottom of the notebook and run the dataset name in the next empty cell.

```
sentinel_2_ds
```

xarray.Dataset

► Dimensions: **(time: 4, x: 386, y: 509)**

▼ Coordinates:

time	(time)	datetime64[ns]	2018-02-01T10:31:35 ... 2...		
y	(y)	float64	6.6e+05 6.599e+05 ... 6.5...		
x	(x)	float64	-1.148e+05 -1.148e+05		
spatial_ref	()	int32	6933		

▼ Data variables:

red	(time, y, x)	float32	1020.0 1036.0 ... 762.0 73...		
green	(time, y, x)	float32	1180.0 1156.0 ... 1050.0 1...		
blue	(time, y, x)	float32	973.0 975.0 984.0 ... 1056....		

▼ Attributes:

crs :	EPSG:6933
grid_mapping :	spatial_ref

The output should show we have 4 timesteps.

Note: If you encounter a `NameError` error message, you might need to reload your packages and functions. Follow the steps above, under **Open and run notebook**.

3.3.3 Create a geomedian composite

Now we will create a geomedian composite using the `xr_geomedian` function. In the next empty cell below, enter the following code, and then run it.

```
geomedian_composite = xr_geomedian(sentinel_2_ds)
```

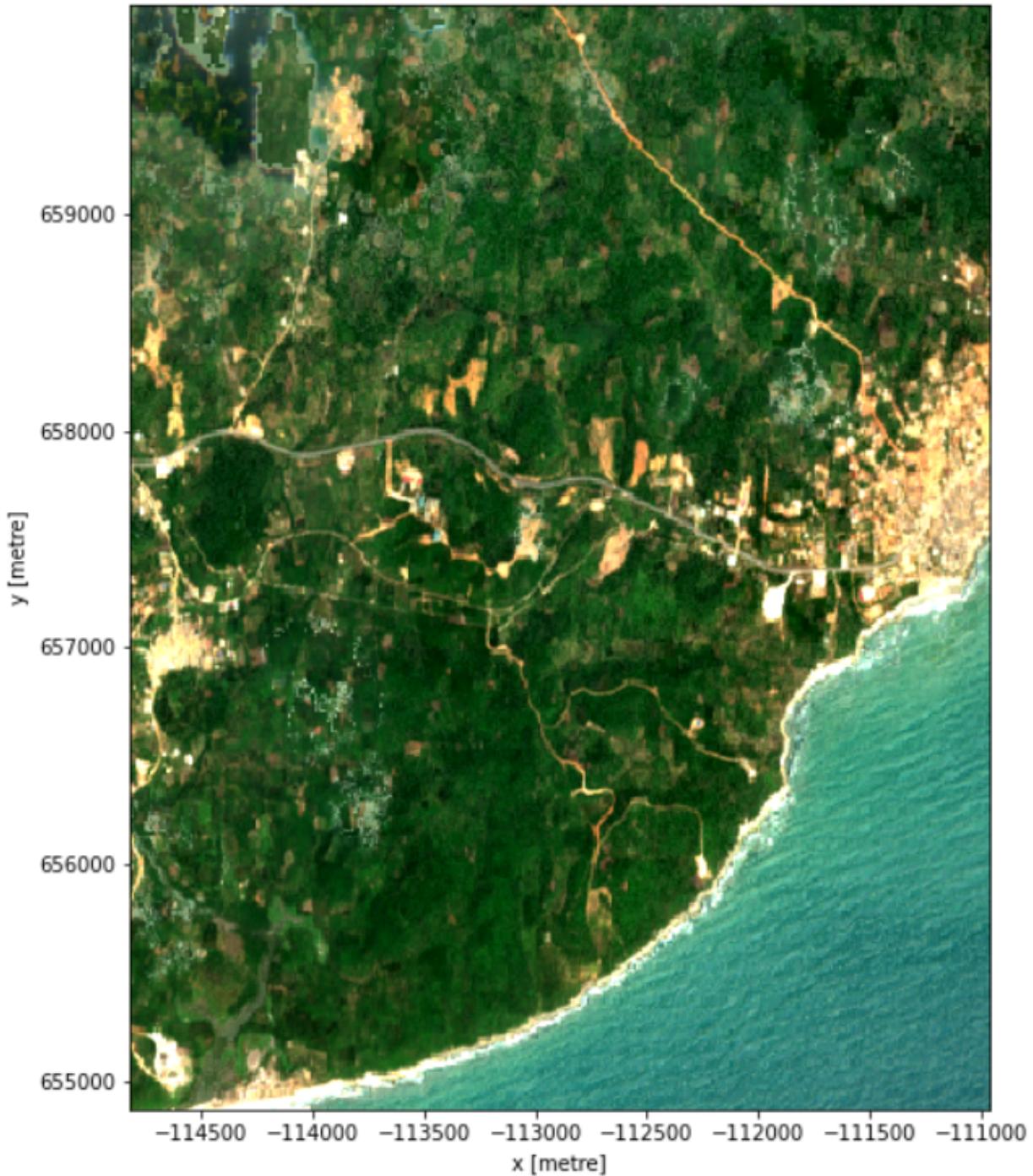
This generates a dataset called `geomedian_composite` which contains one geomedian value for every pixel.

3.3.4 Create an RGB image of the composite

In the new cell below, enter the following code, and then run it to generate an RGB image of the composite.

```
rgb(geomedian_composite,  
    bands=['red', 'green', 'blue'],  
    size=10)
```

The RGB image will look like this:



Excellent! There are no clouds or cloud shadows in the geomedian composite.

There are some visual anomalies, but most of the image accurately represents the area. Note that this image is a representation of what the area looked like over the loaded time range, so from 1 February, 2018, to 15 March, 2018.

Note: Save your work by clicking **File -> Save Notebook** or pressing **Ctrl + S**.

3.3.5 Conclusion

We have created a geomedian composite of Sentinel-2 data and seen that it is an accurate representation of an area. It fills in missing or masked data, such as clouds and cloud shadows.

This is the end of Session 3. In this session, you have been introduced to the concept of using composite datasets and images to fill in data gaps. We have also learned the best statistical method of combining datasets is the geomedian. Good work!

3.4 Session 3 Quiz and Solution

In Session 3, you loaded Sentinel-2 data into the Sandbox using `load_ard()`. The first section of the exercise showed that `load_ard()` applies a cloud mask to each timestep in the dataset. The second part of the exercise used the same dataset to calculate and plot a cloud-free geomedian, using `xr_geomedian()` and `rgb()`.

3.4.1 Quiz

If you would like to be awarded a certificate of achievement at the end of the course, we ask that you [complete the quiz](#). You will need to supply your email address to progress towards the certificate. After you complete the quiz, you can check if your answers were correct by pressing the **View Accuracy** button.

The quiz will ask you to use the notebook you developed for this session's exercise on loading data. If you would like to confirm that your notebook works as expected, you can check it against the solution notebook provided below.

Note: The solution notebook below does not contain the answer to the quiz. Use it to check that you implemented the exercise correctly, then use your exercise notebook to answer the quiz. Accessing the solution notebook will not affect your progression towards the certificate.

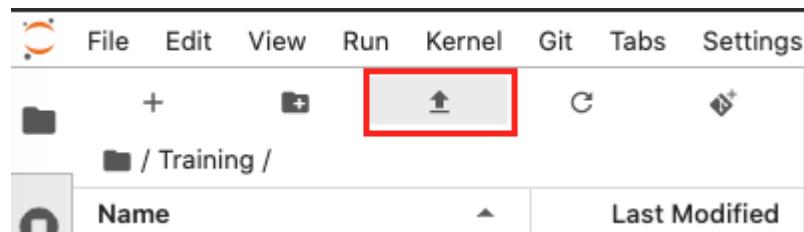
3.4.2 Solution notebook

Note: We strongly encourage you to attempt the exercises on the previous two pages before downloading the solution below. This will help you learn how to use the Sandbox independently for your own analyses.

[Download the solution notebook for the Session 3 exercises](#)

To view this notebook on the Sandbox, you will need to first download it to your computer, then upload it to the Sandbox. Follow these instructions:

1. Download the notebook by clicking the link above.
2. On the Sandbox, open the **Training** folder.
3. Click the **Upload Files** button as shown below.



4. Select the downloaded notebook using the file browser. Click **OK**.
5. The solution notebook will appear in the **Training** folder. Double-click to open it.

CHAPTER FOUR

SESSION 4: INDICES

In this session, we will explore combining multiple bands into various indices to classify and measure terrain features. First, we will introduce several band indices and what they measure. Then we will calculate the geomedian for each season, and show how to measure changes in vegetation over time.

4.1 Band indices

True-colour images, like the RGB ones we created in Session 2 and Session 3, are an intuitive way of displaying spatial data as it appears to our eyes. However, satellites capture many more layers of data: their range extends outside the visible spectrum. We can also make use of these measurements to perform various kinds of data analysis.

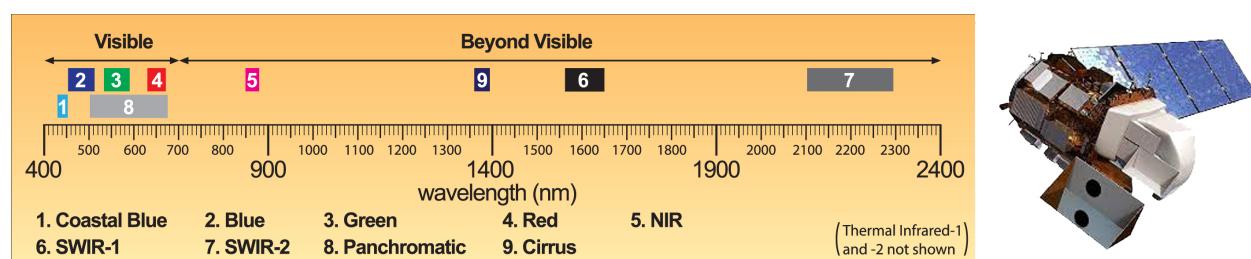
One established analysis technique for identifying specific terrain features, such as water or bare soil, is to calculate a **band index**.

4.1.1 Video: Band indices

Watch the short video below to learn about band indices. Then, read the text below for more detail.

Download the video slides as a PDF

4.1.2 What is a band?



A section of the electromagnetic spectrum showing some of the Landsat 8 bands. [Source]

You may recall from the [introduction to satellite products](#) that the range of data acquired by a satellite is defined by its **bands**. Bands are subdivisions of the electromagnetic spectrum dependent on the sensors on the satellite.

A selection of commonly-used Landsat 8 and Sentinel-2 bands are shown in the table below. We can see the spectral ranges between Landsat 8 and Sentinel-2 are similar, but not the same. Also note the band numbers do not always correspond to the same spectral range.

Band	Landsat 8 wavelength range (nm)	Sentinel-2 wavelength range (nm)
Blue	Band 2 450 – 510	Band 2 458 – 523
Green	Band 3 530 – 590	Band 3 543 – 578
Red	Band 4 640 – 670	Band 4 650 – 680
Near-infrared (NIR)	Band 5 850 – 880	Band 8 785 – 899
Short-wave infrared 1 (SWIR 1)	Band 6 1570 – 1650	Band 11 1565 – 1655

Sources: Landsat 8 bands, Sentinel-2 bands

4.1.3 Bands and terrain features

Different types of ground cover absorb and reflect different amounts of radiation along the electromagnetic spectrum. This is dependent on the physical and chemical properties of the surface.

For example:

- **Water:** Open water bodies generally reflect light in the visible spectrum, and absorb more short-wave infrared than near-infrared radiation. This can change if the water is turbid.
- **Snow:** Ice and snow reflect most visible radiation, but do not reflect much short-wave infrared. Reflectance measurements depend on snow granule size and liquid water content.
- **Green vegetation:** Chlorophyll, a pigment in plants, absorbs a lot of visible light, including red light. When plants are healthy, they strongly reflect near-infrared light.
- **Bare soil:** The mineral composition of soil can be characterised using the visible and near-infrared spectrum. Soil moisture content can greatly influence the results.

Using these spectral differences, we can calculate ratios between bands to isolate and accentuate the specific terrain feature we are interested in. These metrics are known as band ratios or **band indices**.

Note: In practice, variation within terrain feature classes, as well as the presence of multiple features in one area, can make different types of ground cover difficult to distinguish. This is one of the challenges of spectral data analysis.

Example: Normalised Difference Vegetation Index (NDVI)

One of the most widely-used band indices is the Normalised Difference Vegetation Index (NDVI). It is used to show the presence of live green vegetation. Generally, green vegetation has a low red band measurement, as red light is absorbed by chlorophyll. In addition to this, healthy leaf cell structures reflect near-infrared light, giving a high near-infrared (NIR) band measurement.

NDVI is therefore typically calculated using a satellite's NIR band and red band. One value is calculated per pixel.

$$\text{NDVI} = \frac{\text{NIR} - \text{Red}}{\text{NIR} + \text{Red}} \quad (4.1)$$

We see the index is calculated by the difference ($\text{NIR} - \text{Red}$) divided by the sum ($\text{NIR} + \text{Red}$). This normalises the index: all values will fall between -1 and 1 .

Large values of NDVI will occur for pixels where NIR is high and red is low. Conversely, NDVI can be close to 0 or even negative where NIR is low and red is high. This means we interpret NDVI as follows:

$$\text{NDVI} > 0, \text{ or close to } 1 = \text{green vegetation} \quad (4.2)$$

$$\text{NDVI} \leq 0 = \text{not green vegetation; water, soil, etc.} \quad (4.3)$$

But what is green vegetation? The US Geological Survey provides a more specific [guide to interpreting NDVI](#).

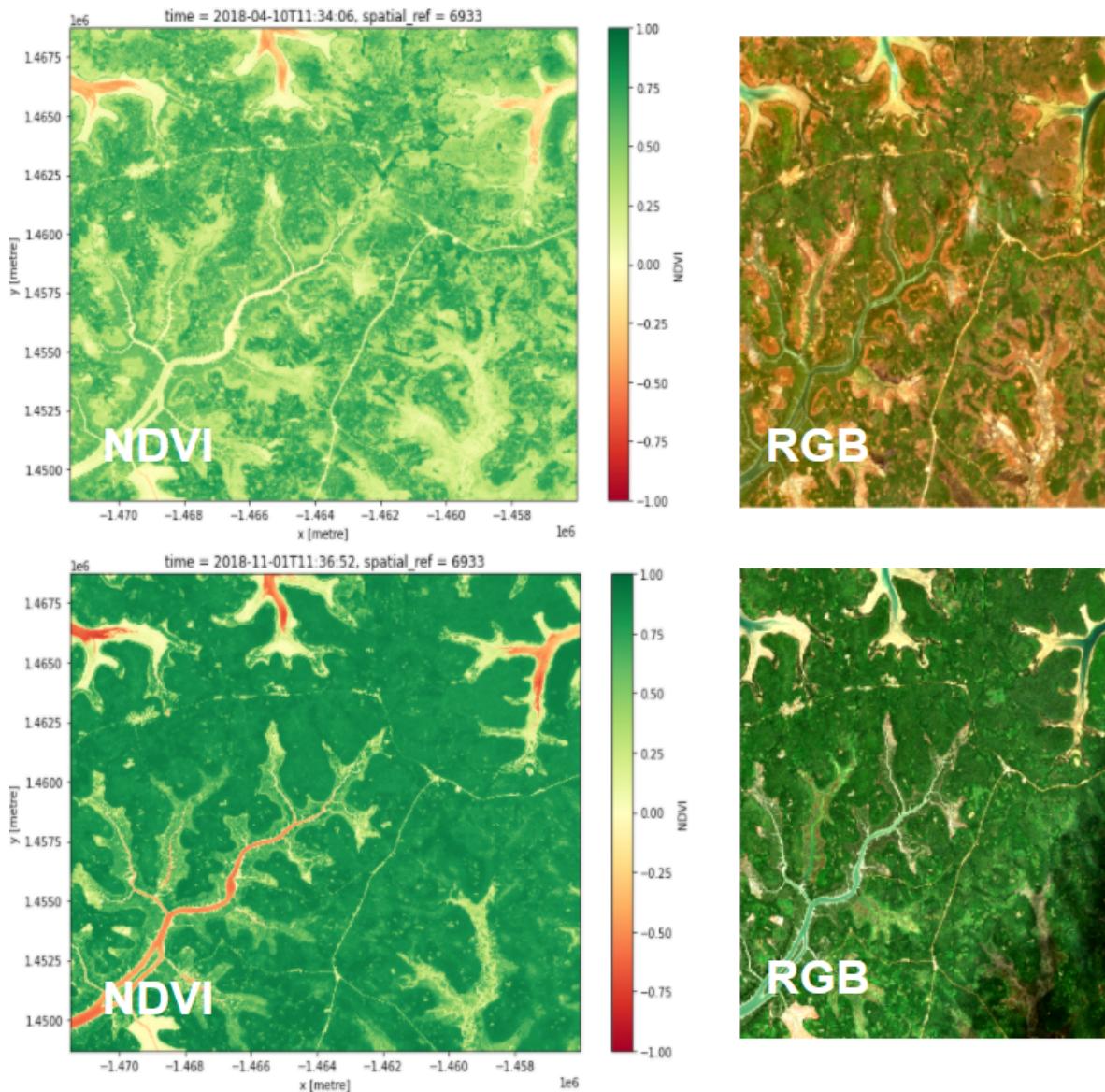
Areas of barren rock, sand, or snow usually show very low NDVI values (for example, 0.1 or less). Sparse vegetation such as shrubs and grasslands or senescent crops may result in moderate NDVI values (approximately 0.2 to 0.5). High NDVI values (approximately 0.6 to 0.9) correspond to dense vegetation such as that found in temperate and tropical forests or crops at their peak growth stage.

You can see even this definition does not exhaustively cover every kind of vegetation. It is important to remember that Earth observation data analysis is sensitive to the dataset location and time. The nature of climate and environment variations across the globe, and even just within the African continent, mean that band indices like NDVI need to be interpreted with knowledge and context of the area.

Note: Normalising band indices so their values fall between -1 and 1 gives a relative scale that allows for easier data analysis. Values of the index can be better compared between different points in the area of analysis, and across time periods.

Below we have two plots of an area with river distributaries in Guinea-Bissau, a country that experiences monsoonal-like seasons. We see the amount of vegetation as detected by NDVI fluctuates over time. The top image shows NDVI in April, at the end of the dry season. NDVI readings are much lower than when compared to the same area in November (bottom image), after several months of rain during the wet season.

Notice the RGB images may show parts of the area to be visibly ‘dry’ or ‘lush’, but in places where this is less obvious, it is easier to analyse NDVI than the multispectral RGB dataset.



NDVI calculated from Sentinel-2 data in Guinea-Bissau in April 2018 (top left) and November 2018 (bottom left). The NDVI values reflect typical seasonal patterns in a quantitative manner.

Note: We see in the table above that satellites have slightly different band ranges for the same band name (e.g. ‘Red’ for Landsat 8 is slightly different than ‘Red’ for Sentinel-2). This means they will produce different band index values even at approximately the same time and place. It is good practice to compare like datasets only.

4.1.4 Band indices in research

NDVI is just one example of a useful band index. There are many other band indices used in Earth observation research to draw out terrain features. Selecting a band index is often dependent on environmental conditions and research purpose.

- **Vegetation:** As described above, NDVI is a good baseline vegetation index. It is simple to calculate, and only requires two bands — red and NIR. However, the Enhanced Vegetation Index (EVI) is often more accurate. EVI is calculated with three bands — red, blue and NIR — and requires some coefficients for scaling.

In arid regions, where vegetative cover is low, consider using the Soil Adjusted Vegetation Index (SAVI), which incorporates a soil brightness correction factor.

- **Urbanisation:** Human settlements can be identified through urbanisation indices, one of which is the Normalised Difference Built-up Index (NDBI). NDBI uses SWIR 1 and NIR bands:

$$\text{NDBI} = \frac{\text{SWIR 1} - \text{NIR}}{\text{SWIR 1} + \text{NIR}}$$

However, NDBI can be confused between built-up areas and bare soil, so in arid and semi-arid regions where this is problematic, it may be better to use the Dry Bare Soil Index (DBSI).

$$\text{DBSI} = \frac{\text{SWIR 1} - \text{Green}}{\text{SWIR 1} + \text{Green}} - \text{NDVI}$$

- **Water bodies:** Delineation between water and land can be defined using the Modified Normalised Difference Water Index (MNDWI). It is calculated using green and SWIR 1 bands:

$$\text{MNDWI} = \frac{\text{Green} - \text{SWIR 1}}{\text{Green} + \text{SWIR 1}}$$

This should not be confused with indices for monitoring water content in vegetation.

It is important to remember band indices are not infallible; their usefulness relies on appropriate index selection and sensible interpretation. However, as the field of remote sensing grows, ongoing research into differentiating terrain types with different band combinations give rise to more nuanced and accurate data analysis. For instance, it is common to use more than one index to help distinguish feature classes with similar spectral characteristics.

4.1.5 Conclusion

Band indices are an integral part of spatial data analysis, and are an efficient method of distinguishing different types of land cover. In the following sections, we will calculate NDVI for a cloud-free composite using the Sandbox.

4.2 Calculating NDVI: Part 1

This exercise follows on from the previous section. In the *final exercise of the previous session*, you constructed a notebook to create geomedian composite.

In this section, we will create a new notebook based on the notebook from the previous section. Most of the code will remain unchanged, but we will change the area of interest and time extent. We will also add steps to resample the new dataset and create a geomedian. In the next section, we will calculate and plot NDVI.

Note: We will be using the notebook we created in the previous section, *Create a geomedian composite*. If you have not already set up a copy of the notebook called `Geomedian_composite.ipynb` with the required packages and functions, follow the instructions in previous section. Ensure you have completed all the steps, including loading the Sentinel-2 dataset.

4.2.1 Set up notebook

Create a copy of the notebook

Before you continue with the next step,

1. Log in to the Sandbox and open the **Training** folder.
2. Make a copy of the `Geomedian_composite.ipynb` notebook.
3. Rename the notebook to `Calculate_ndvi.ipynb`.

See [create a copy of a notebook and rename it](#) for more details.

Clearing the notebook

We will need to remove any output from previous runs of the notebook.

1. Select **Kernel -> Restart Kernel and Clear All Outputs....**
2. When prompted, select **Restart**.

4.2.2 Running the notebook

This notebook is still set up to run the Session 3 exercise, so you will need to follow the instructions below to modify it. Work cell by cell and pay attention to what needs to be changed.

Set-up

1. Run the first cell, which contains the packages and functions for the analysis. No need to change anything here.
2. For the `dc = datacube.Datacube` command, change the app name to "Calculate_ndvi". It should look like:

```
dc = datacube.Datacube(app="Calculate_ndvi")
```

Load the data

1. Change the x and y values to those shown below and run the cell.

```
x=(-6.1495, -6.1380)  
y=(13.9182, 13.9111)
```

2. Change the time in the `load_ard` function to ("2019-01", "2019-12").
3. Remove the option `min_gooddata=0.7`.

If you completed steps 2, 3 and 4, your load cell should look like

```
sentinel_2_ds = load_ard(  
    dc=dc,  
    products=["s2_l2a"],  
    x=x, y=y,  
    time=("2019-01", "2019-12"),  
    output_crs="EPSG:6933",
```

(continues on next page)

(continued from previous page)

```
measurements=['red', 'green', 'blue'],
resolution=(-10, 10),
group_by='solar_day')
```

- Run the cell. The load should return 71 time steps.

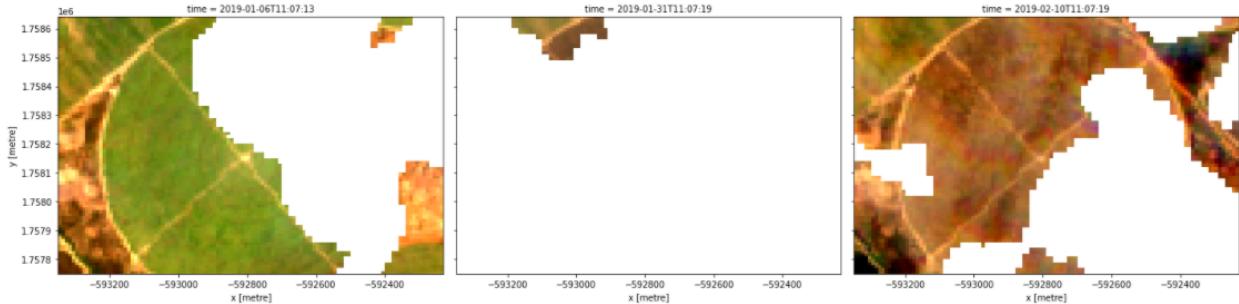
Plot timesteps

The fifth cell of the notebook contains an `rgb` command to plot the loaded data. To match our example below, modify this cell so that it matches the code below:

```
timesteps = [1, 6, 8]

rgb(sentinel_2_ds, bands=['red', 'green', 'blue'], index=timesteps)
```

This will plot images for the 1st, 6th and 8th timestep of the loaded data (remember that Python starts counting at 0). Your image should match the one below.



Note: You may also like to run this cell a few times, experimenting with different values for the `timesteps` parameter. The load command should have returned 71 time steps, meaning the values in your `timesteps` list can be anywhere from `0` to `70`.

Resampling the dataset

Resampling is used to create a new set of times at regular intervals. Using the `resample` method, the data can be arranged in days, months, quarterly (three months) or yearly.

Below gives examples of how the data are grouped.

- 'nD' - number of days (e.g. '7D' for seven days)
- 'nM' - number of months (e.g. '6M' for six months)
- 'nY' - number of years (e.g. '2Y' for two years)

Follow the steps below to resample the dataset time steps to quarterly.

- Delete the code for plotting all RGB images:

```
rgb(sentinel_2_ds, bands=['red', 'green', 'blue'],
col='time', size=4)
```

2. In the cleared cell, write the following code to resample the data and store it in the `resample_sentinel_2_ds` variable:

```
resample_sentinel_2_ds = sentinel_2_ds.resample(time='3MS')
```

`resample_sentinel_2_ds` describes how to group the data into quarterly segments. We can now use this to calculate the geomedian for each quarterly segment.

Note: S at the end of '3MS' is to group the data by start of the month.

Compute the geomedian

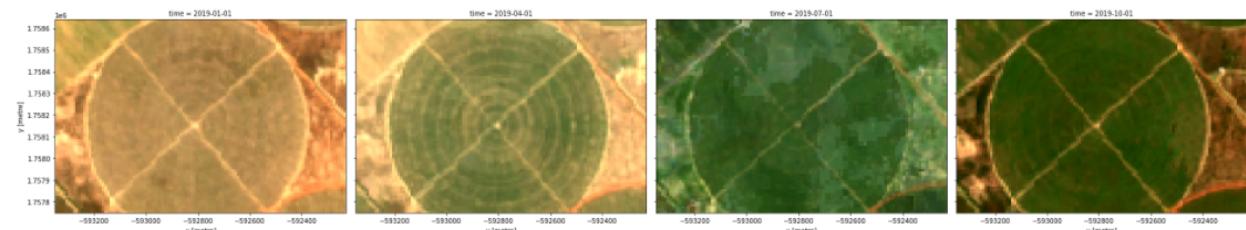
For this session, instead of calling `xr_geomedian(sentinel_2_ds)` on the entire array as we did in the previous exercise, we pass the `xr_geomedian` function to `map` and apply it separately to each resampled group(`resample_sentinel_2_ds`).

Replace the existing `xr_geomedian` code with:

```
geomedian_resample = resample_sentinel_2_ds.map(xr_geomedian)
```

We can plot the output geomedians, and see the change in the landscape over the year. Replace the existing `rgb` code with:

```
rgb(geomedian_resample, bands=['red', 'green', 'blue'], col="time", col_wrap=4)
```



Comparing the two datasets

Comparing the two dataset can you tell the difference from the results shown below?

`sentinel_2_ds`

xarray.Dataset

► Dimensions:	(time: 71, x: 112, y: 89)
▼ Coordinates:	
<code>x</code>	(x) float64 -5.933e+05 ... -5.922e+05
<code>spatial_ref</code>	0 int32 6933
<code>time</code>	(time) datetime64[ns] 2019-01-01T11:07:16 ... 2019-12-...
<code>y</code>	(y) float64 1.759e+06 1.759e+06 ... 1.758e+06
▼ Data variables:	
<code>red</code>	(time, y, x) float32 1192.0 1272.0 ... 939.0 938.0
<code>green</code>	(time, y, x) float32 1036.0 1092.0 ... 985.0 971.0
<code>blue</code>	(time, y, x) float32 766.0 808.0 785.0 ... 550.0 530.0
▼ Attributes:	
<code>crs :</code>	EPSG:6933
<code>grid_mapping :</code>	spatial_ref

`geomedian_resample`

xarray.Dataset

► Dimensions:	(time: 4, x: 112, y: 89)
▼ Coordinates:	
<code>time</code>	(time) datetime64[ns] 2019-01-01 ... 2019-10-01
<code>x</code>	(x) float64 -5.933e+05 ... -5.922e+05
<code>y</code>	(y) float64 1.759e+06 1.759e+06 ... 1.758e+06
▼ Data variables:	
<code>red</code>	(time, y, x) float32 1636.4856 1729.0841 ... 670.9271
<code>green</code>	(time, y, x) float32 1328.8126 1375.9498 ... 716.0723
<code>blue</code>	(time, y, x) float32 837.9998 877.3378 ... 358.5672
► Attributes: (0)	

Take a look at the dimensions. The raw dataset `sentinel_2_ds` has 71 time steps loaded, but after resampling to quarterly as in `geomedian_resample`, the time dimension is now 4. This makes sense because the year has been divided into four quarters.

4.2.3 Conclusion

Congratulations! You have successfully modified a notebook to create a quarterly geomedian composite by resampling Sentinel-2 data.

If you'd like to experiment further, try running the code with different resampling values. Did you learn anything interesting to share with us?

In the next section, we will continue working with this notebook to calculate the NDVI values for each of our quarterly geomedians.

4.3 Calculating NDVI: Part 2

This exercise follows on from the previous section. In the *previous part of this exercise*, you constructed a notebook to resample a year's worth of Sentinel-2 data into quarterly time steps.

In this section, you will continue from where you ended in the previous exercise. Most of the code will remain unchanged, but we will introduce a new measurement to the existing measurements which will enable us to calculate and plot NDVI.

Note: We will be using the notebook we created in the previous section. If you have not already set up a copy of the notebook called `Calculate_ndvi.ipynb` with the required packages and functions, follow the instructions in previous section. Ensure you have completed all the steps, including loading the Sentinel-2 dataset.

4.3.1 Open and run notebook

If you are following directly on from the last section, you can skip this step. If you have closed your Sandbox browser tab or disconnected from the Internet between exercises, follow these steps to ensure correct package imports and connection to the datacube.

1. Navigate to the **Training** folder.
2. Double-click `Calculate_ndvi.ipynb`. It will open in the Launcher.
3. Select **Kernel -> Restart Kernel and Clear All Outputs....**
4. When prompted, select **Restart**.

4.3.2 Making changes to the load cell

Make the following changes below to modify the load cell.

Adding nir measurement

To calculate NDVI, we need to load Sentinel-2's near-infrared band. In the Sandbox, it is called `nir`. To add the band, modify the `load_ard` cell according to the step below:

1. Add `nir` to the measurements array.

```
measurements = ['red', 'green', 'blue', 'nir']
```

If you completed the above step, your `load_ard` cell should look like:

```
sentinel_2_ds = load_ard(
    dc=dc,
    products=["s2_l2a"],
    x=x, y=y,
    time=("2019-01", "2019-12"),
    output_crs="EPSG:6933",
    measurements=['red', 'green', 'blue', 'nir'],
    resolution=(-10, 10),
    group_by='solar_day')
```

Running the notebook

1. Select **Kernel -> Restart Kernel and Run All Cells....**
2. When prompted, select **Restart**.

The notebook may take a little while to run. Check all the cells have run successfully with no error messages.

Did you noticed any additional data variables to the `sentinel_2_ds`?

► Dimensions: (**time**: 4, **x**: 57, **y**: 45)

▼ Coordinates:

time	(time)	datetime64[ns] 2019-01-01 ... 2019-10-01	
x	(x)	float64 -5.934e+05 ... -5.922e+05	
y	(y)	float64 1.759e+06 1.759e+06 ... 1.758e+06	

▼ Data variables:

red	(time, y, x)	float32 1654.7893 1806.2437 ... 712.03784	
green	(time, y, x)	float32 1348.7592 1431.4747 ... 751.21564	
blue	(time, y, x)	float32 837.38544 894.4673 ... 384.61166	
nir	(time, y, x)	float32 3740.0647 3757.8975 ... 2953.526	

► Attributes: (0)

Creating a new cell

After successfully running the notebook, this cell will be the last cell:

```
geomedian_resample
```

```
xarray.Dataset
```

~ Dimensions: (time: 4, x: 112, y: 89)

▼ Coordinates:

time	(time)	datetime64[ns] 2019-01-01 ... 2019-10-01		
y	(y)	float64 1.759e+06 1.759e+06 ... 1.758e+06		
x	(x)	float64 -5.933e+05 ... -5.922e+05		

▼ Data variables:

red	(time, y, x)	float32 1654.7893 1770.8843 ... 694.822		
green	(time, y, x)	float32 1348.7592 1409.6754 ... 739.2761		
blue	(time, y, x)	float32 837.38544 882.7103 ... 380.6102		
nir	(time, y, x)	float32 3740.0647 3712.3347 ... 2931.3264		

~ Attributes: (0)

Notice it now contains the NIR band data, which is the data we just loaded.

Follow the steps below to create a new cell.

1. Make sure the last cell is selected.
2. Press the Esc key, then follow it by pressing the B key. A new cell will be created below the current cell.

Use the method above to create a new cell.

4.3.3 Calculate NDVI

One of the most commonly used remote sensing indices is the Normalised Difference Vegetation Index or NDVI. This index uses the ratio of the red and near-infrared (NIR) bands to identify live green vegetation. The formula for NDVI is:

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

When interpreting this index, high values indicate vegetation, and low values indicate soil or water.

Define NDVI formula

In a new cell, calculate the NDVI for the resampled geomedian dataset. To make it simpler, you can store the red and near-infrared bands in new variables, then calculate the NDVI using those variables, as shown below:

```
nir = geomedian_resample.nir  
red = geomedian_resample.red  
  
NDVI = (nir - red) / (nir + red)
```

Run the cell using Shift + Enter.

Plot NDVI for each geomedian

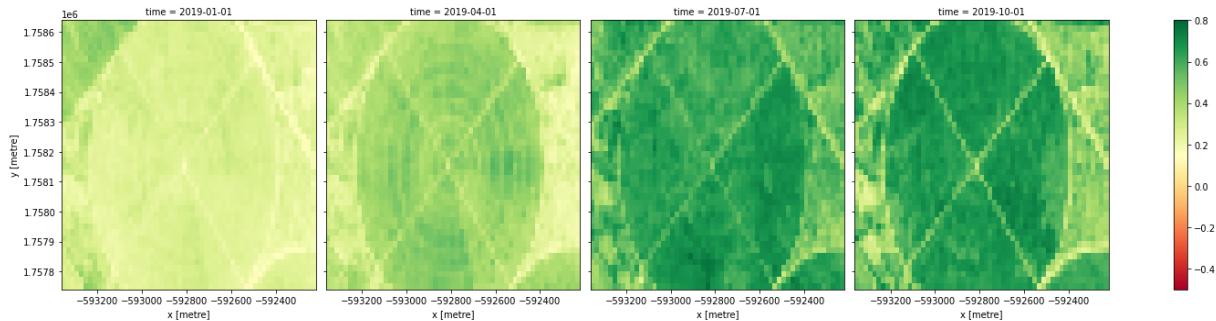
Our calculation is now stored in the NDVI variable. To visualise it, we can attach the `.plot()` method, which will give us an image of the NDVI for each geomedian in our dataset. We can then customise the plot by passing parameters to the `.plot()` method, as shown below:

```
NDVI.plot(col='time', vmin=-0.50, vmax=0.8, cmap='RdYlGn')
```

Run the cell using Shift + Enter

- `col='time'` tells the plot that we want to display one image for each time step in our dataset.
- `vmin=-0.50` tells the plot to display all values below `-0.50` as the same colour. This can help keep contrast in the images (remember that NDVI can take values from -1 to 1).
- `vmax=0.8` tells the plot to display all values above `0.8` as the same colour. This can help keep contrast in the images (remember that NDVI can take values from -1 to 1).
- `cmap='RdYlGn'` tells the plot to display the NDVI values using a colour scale that ranges from red for low values to green for high values. This helps us because healthy vegetation shows up as green, and non-vegetation shows up as red.

If you implement the NDVI plotting code correctly, you should see the image below:



In the image above, vegetation shows up as green ($\text{NDVI} > 0$). Sand shows up as yellow ($\text{NDVI} \sim 0$) and water shows up as red ($\text{NDVI} < 0$).

Note: The `xarray .plot()` function is briefly described in [Python basics 5: Xarray](#).

Plot time series of the NDVI area

While it is useful to see the NDVI values over the whole area in the plots above, it can sometimes be useful to calculate summary statistics, such as the mean NDVI for each geomedian. This can quickly reveal trends in vegetation health across time.

To calculate the mean NDVI, we can apply the `.mean()` method to our NDVI variable. We can also then apply the `.plot()` method to see the result, as shown below:

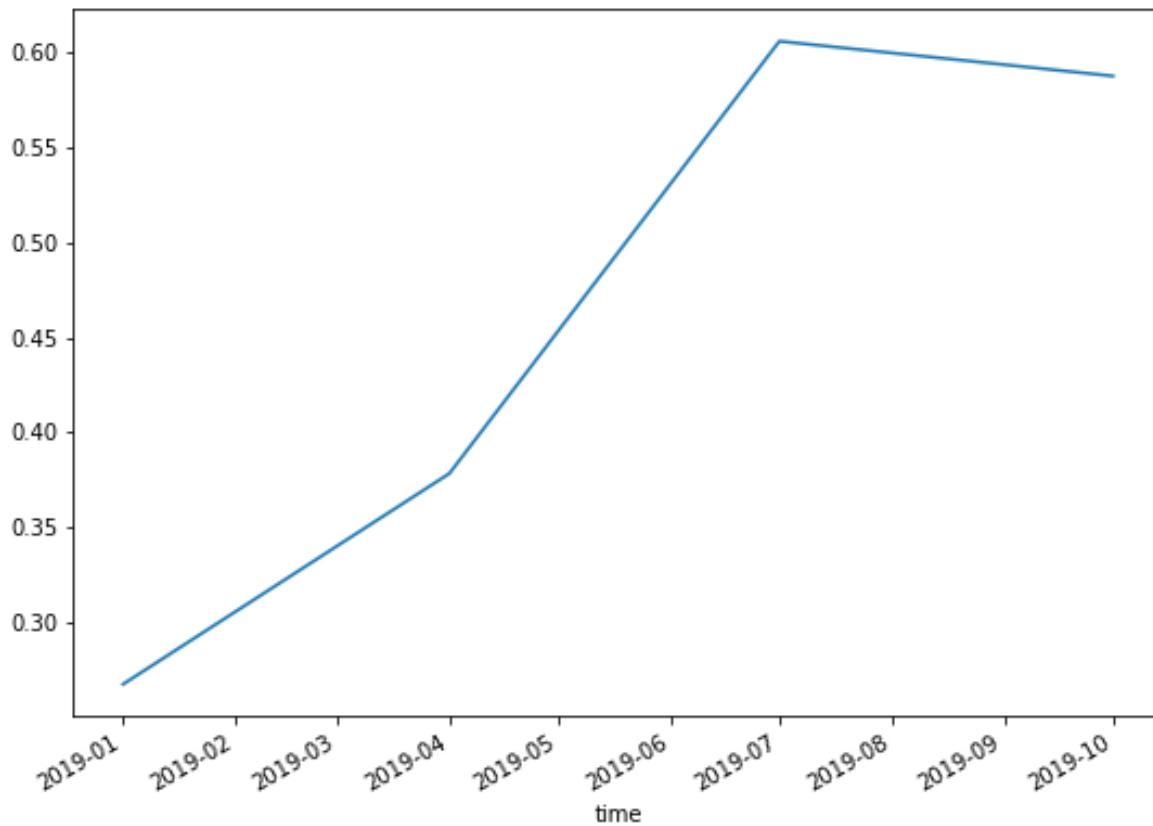
```
NDVI.mean(dim=['x', 'y']).plot(size=6)
```

Run the cell using Shift + Enter

- `NDVI.mean(dim=['x', 'y'])` calculates the mean over all pixels, indicated by `dim=['x', 'y']`. To instead calculate the mean over all times, you would write `dim=['time']`.

- `NDVI.mean(dim=['x', 'y']).plot(size=6)` calculates the mean over all pixels, then plots the result. The `size=6` argument specifies the size of the plot.

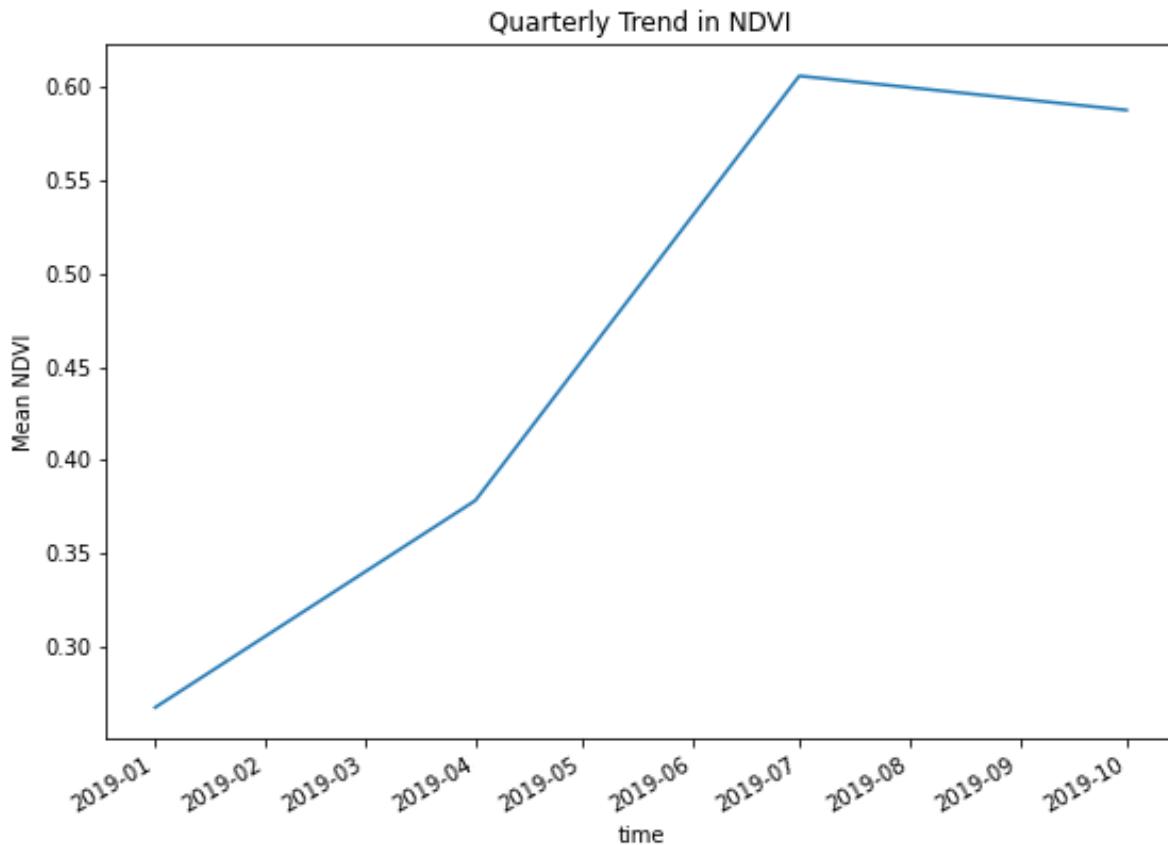
If you implement the calculation and plotting code correctly, you should see the image below:



Rather than a spatial view of NDVI at each time step, we see a single value (the mean NDVI) for each time.

If you would like to add a title and y-axis label to this plot, you can add the following code below the command to calculate and plot the mean:

```
NDVI.mean(dim=['x', 'y']).plot(size=6)
plt.title('Quarterly Trend in NDVI')
plt.ylabel('Mean NDVI')
```



Run the cell using Shift + Enter.

4.3.4 Conclusion

Congratulations! You have successfully calculated and visualised the NDVI for a series of geomedian composite images.

If you'd like to experiment further, try running the code with different areas. Did you learn anything interesting to share with us?

4.4 Session 4 Quiz and Solution

In Session 4, you created a notebook to load satellite products, resample the data into quarterly segments, and then calculate and plot the Normalised Difference Vegetation Index (NDVI). Combining bands into indices such as NDVI can yield important insights into the data which might not be apparent using true-color RGB imagery alone.

4.4.1 Quiz

If you would like to be awarded a certificate of achievement at the end of the course, we ask that you [complete the quiz](#). You will need to supply your email address to progress towards the certificate. After you complete the quiz, you can check if your answers were correct by pressing the **View Accuracy** button.

The quiz will ask you to use the notebook you developed for this session's exercise on calculating NDVI. If you would like to confirm that your notebook works as expected, you can check it against the solution notebook provided below.

Note: The solution notebooks below do not contain the answer to the quiz. Use them to check that you implemented the exercises correctly, then use your exercise notebook to answer the quiz. Accessing the solution notebook will not affect your progression towards the certificate.

4.4.2 Solution notebook

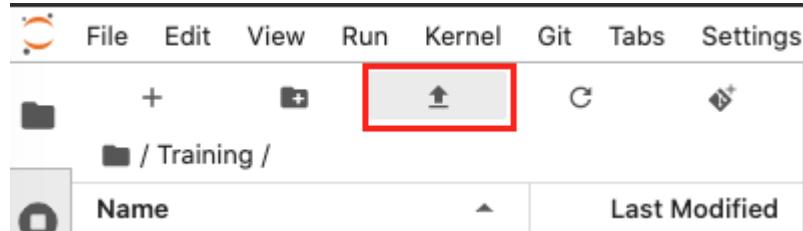
Note: We strongly encourage you to attempt the exercise on the previous page before downloading the solution below. This will help you learn how to use the Sandbox independently for your own analyses.

[Download the solution notebook for the Session 4 Part 1 exercise](#)

[Download the solution notebook for the Session 4 Part 2 exercise](#)

To view a solution notebook on the Sandbox, you will need to first download it to your computer, then upload it to the Sandbox. Follow these instructions:

1. Download the notebook by clicking the link above.
2. On the Sandbox, open the **Training** folder.
3. Click the **Upload Files** button as shown below.



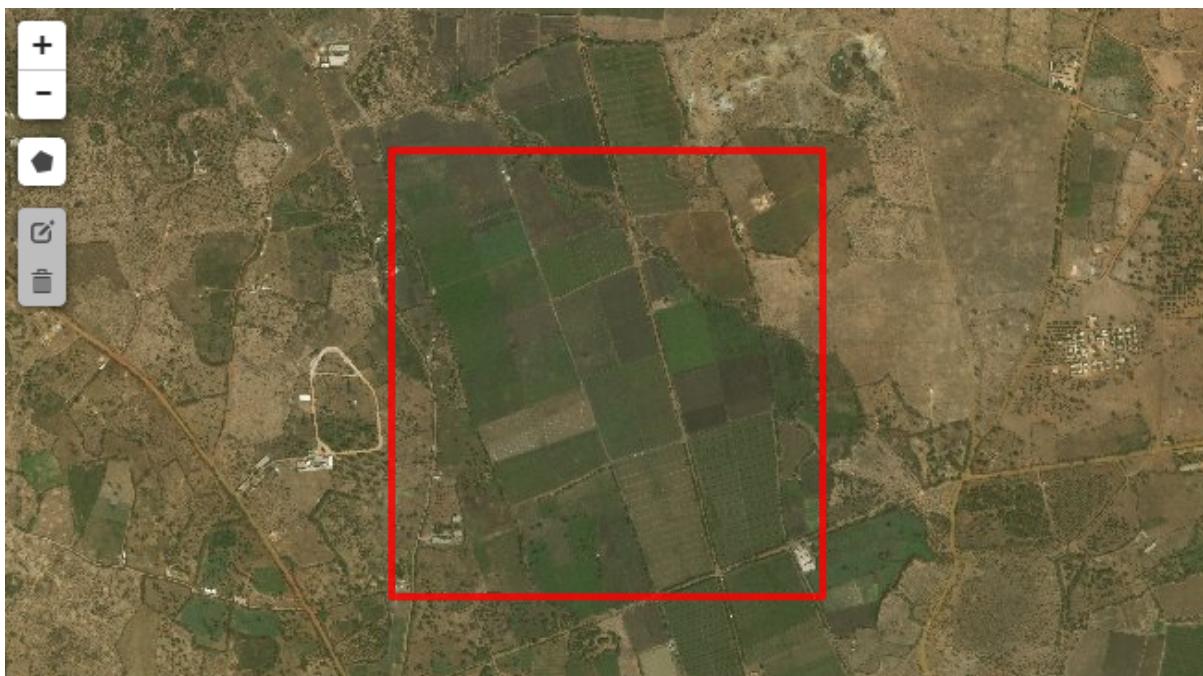
4. Select the downloaded notebook using the file browser. Click **OK**.
5. The solution notebook will appear in the **Training** folder. Double-click to open it.

SESSION 5: VEGETATION ANALYSES

In this session, we will combine many of the techniques you've learned during the training course to build a complete case study. This is where you have a question in mind that you want to answer, load and analyse the appropriate data, then draw conclusions to try and answer your question.

This session will focus on case studies related to vegetation. First, we'll cover the available vegetation case studies on the Sandbox, as well as how to structure your own case study. Then, you'll work through building your own case study to detect changes in vegetation over time.

5.1 Building a case study: vegetation analysis



The Crop Health notebook investigated in Session 1 is an example of a case study. We now have enough knowledge of the Sandbox to build a similar comprehensive vegetation analysis.

5.1.1 Overview

In the [last session](#), we calculated the Normalised Difference Vegetation Index (NDVI) as a method of measuring green vegetation over an area of interest.

The index made the Earth observation data easier to analyse, and allowed comparison across time periods. In our NDVI map of a cropping field, we could see there were seasonal variations over the quarterly geomedians. However, that information by itself provides little context for further use. Why are we looking at this field? Do the results fit what we were expecting to see? What can we use this information for?

How you analyse Digital Earth Africa data will depend on **what you want to know**. Now that we know more about the strengths and limitations of vegetation indices, we can use them to form the basis of a complete case study that informs a user of a real-world outcome.

This session will show you how to extend dataset calculations into a meaningful report. This section outlines some common goals of vegetation analysis, and provides references to examples of case study notebooks. The following sections will walk through a tutorial on how to construct a notebook on vegetation change detection.

5.1.2 The significance of vegetation



Food security is a major focus of the UN's Sustainable Development Goals (SDGs). The SDGs are a key tenet of the Digital Earth Africa program. [Image source]

Growing things are a cornerstone of life on Earth. Monitoring vegetation can give us insight into:

- **Agriculture:** including cropping, seasonal patterns and changes in land use
- **Environment health and industry:** including forestry, logging and mining

- **Climate:** including desertification and drought

5.1.3 Real world example notebooks

Several case studies based on vegetation indices have already been made using the Digital Earth Africa Sandbox. These case study notebooks can be found in the Sandbox folder called **Real_world_examples**.

The **Real_world_examples** folder contains many case studies; the notebooks related to vegetation are summarised below.

- `Crop_health.ipynb`

This notebook draws upon the crop health app function to calculate NDVI for selected fields. By comparing two polygons within the area, it is possible to assess the fields for crop health and productivity over time.

Session 1 used this crop health notebook as an example of Sandbox capability.

- `Vegetation_change_detection.ipynb`

This notebook visualises vegetation change over time. Areas where vegetation has increased compared to a baseline time are marked in blue, while areas where vegetation has decreased are output as red. This change is indexed using either NDVI, or the Enhanced Vegetation Index (EVI). The notebook allows the user to choose which index is being used.

The exercise in this session, Session 5, will contain instructions to create a similar notebook.

Note: It is recommended you open these notebooks in the Sandbox, from the folder called **Real_world_examples**. This will allow you to run the pre-prepared code for yourself. However, they can also be viewed at the [Digital Earth Africa notebook repository](#), which will display a read-only HTML version from GitHub.

5.1.4 Structuring a case study

If you opened up some of the real world example notebooks, you might notice they are more complicated than the exercises in this training course. However, they all have the same structure.

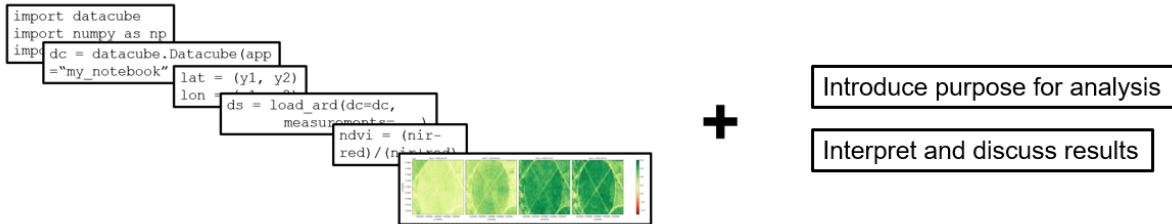
Each notebook you have made from the training course exercises, as well as each real world example notebook, consist of six fundamental components.

1. Load packages and functions
2. Connect to the datacube
3. Define the area of interest using longitude and latitude
4. Load data from the datacube
5. Perform analysis or calculation on the data (such as calculating an index)
6. Plot the results

The **Real_world_examples** notebooks go a few steps further:

- Frame the analysis in the context of a question or purpose — this is normally introduced at the start of the notebook
- Extend results analysis and discussion

Components of a case study



5.1.5 Conclusion

With some understanding of the context behind spatial data analysis, we are now ready to create our own case study notebook. This week's exercise is about incorporating the additional two steps (analysis purpose, and additional results discussion) to make a well-rounded case study.

5.2 Exercise: Vegetation change detection

5.2.1 Overview

In this exercise, we will create a notebook to detect vegetation change. To compare change, we need to have data from two different times: an older dataset, and a newer dataset.

The notebook will include the following steps:

- Load Landsat 8 data
- Calculate a vegetation index for the loaded data
- Split the vegetation index data into half, based on when the data was collected — an older half and a newer half
- Compute the mean composite for each half; and
- Compare the older and newer averages to check for vegetation change.

At the conclusion of this exercise, you will have performed a vegetation analysis which can be used to report on changes in the selected area.

5.2.2 Set up notebook

In your **Training** folder, create a new Python 3 notebook. Name it `Vegetation_exercise.ipynb`. For more instructions on creating a new notebook, see the [instructions from Session 2](#).

Load packages and functions

In the first cell, type the following code and then run the cell to import necessary Python dependencies.

```
import matplotlib.pyplot as plt
%matplotlib inline

import datacube

from deafrica_tools.datahandling import load_ard
from deafrica_tools.plotting import display_map
from deafrica_tools.bandindices import calculate_indices
```

Note: As of June 2021, the `deafrica_tools` package has replaced the deprecated `sys.path.append('.. / Scripts')` file import. For more information on `deafrica_tools`, visit the DE Africa Tools module documentation.

In this exercise, we import one new function, `calculate_indices`.

Instead of calculating band indices such as NDVI by defining a formula, we can call upon preset index calculations in the Sandbox. `calculate_indices` contains definitions for over a dozen different indices, from NDVI to the Bare Soil Index (BSI) to Tasselled Cap Wetness (TCW), and can apply them to your dataset.

Using this function to select the index we want might seem like a lot of effort. However, there are some benefits to using the `calculate_indices` function, as we will see in this exercise.

- Reduce chances of error in typing out the formula, as they are already defined in the `calculate_indices` script — this is great for more complicated formulae
- Compare different index results without manually defining many different formulae
- Reduce the number of definitions you have to type — for instance, there is no need for `red=` or `nir=` as we used in Session 4

We will use `calculate_indices` after we load the dataset.

Connect to the datacube

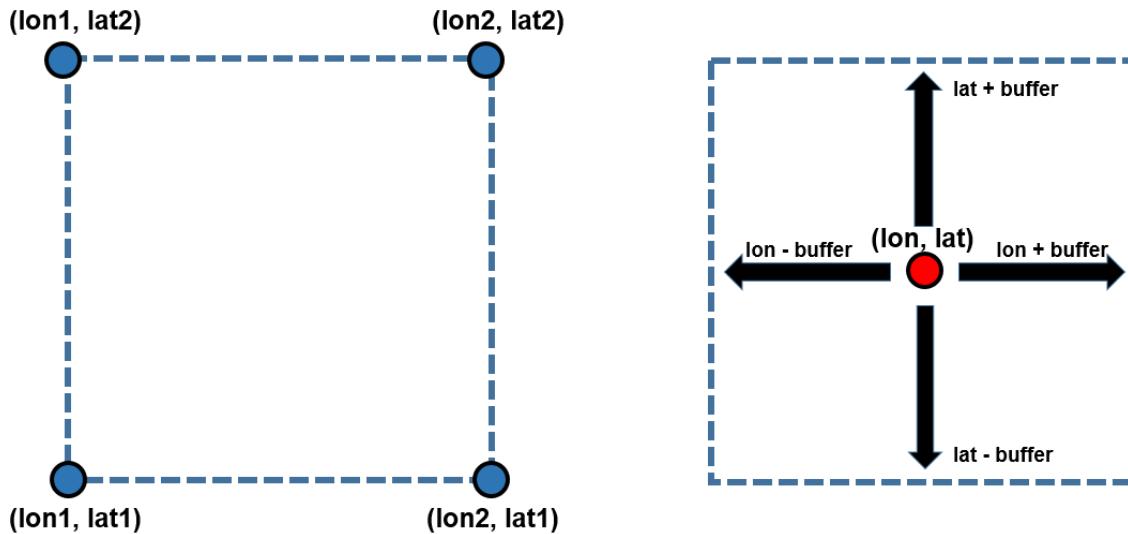
Enter the following code and run the cell to create our `dc` object, which provides access to the datacube.

```
dc = datacube.Datacube(app="Vegetation_exercise")
```

Select area of interest

We need to select an area of interest. Some areas to check for vegetation changes include mining sites, where there might be devegetation, or crops, where seasonal changes in vegetation greenness occur.

Previously, we have provided a longitude range and a latitude range. However, it is more common to define a central point, and provide a buffer around it.



We previously used the `x=(lon1, lon2), y=(lat1, lat2)` method (left) to define our area of interest. In this exercise, we will use the buffer zone method (right). The advantage is that you can define one buffer to use in all four directions, and it is much easier to explore different areas by changing the central point and/or the buffer width.

We will be selecting an area around a centre point. Enter the following code and run the cell to select an area and time range of interest. The parameters are:

- latitude: The latitude at the centre of your area of interest.
- longitude: The longitude at the centre of your area of interest.
- buffer: The number of degrees to load around the central latitude and longitude.
- time: The date range to analyse. For reasonable results, the range should span at least two years to prevent detecting seasonal changes.
- time_baseline: The date at which to split the total dataset into two non-overlapping samples. For this exercise, we choose a date halfway in our time range. Its value here, '2015-12-31', is halfway between '2013-01-01' and '2018-12-31' - the time range in time. So our two time periods will be 2013 to 2015, and 2016 to 2018.

In the next cell, enter the following code, and then run it to select an area.

```
# Define the area of interest
latitude = 0.02
longitude = 35.425
buffer = 0.1

# Combine central lat,lon with buffer to get area of interest
lat_range = (latitude-buffer, latitude+buffer)
lon_range = (longitude-buffer, longitude+buffer)

# Set the range of dates for the complete sample
time = ('2013-01-01', '2018-12-01')

# Set the date to separate the data into two samples for comparison
time_baseline = '2015-12-31'
```

In the next cell, enter the following code, and then run it to show the area on a map. Since we have defined our

area using the variables `lon_range` and `lat_range`, we can use those instead of typing out (`latitude-buffer`, `latitude+buffer`) and (`longitude-buffer`, `longitude+buffer`) again.

```
display_map(x=lon_range, y=lat_range)
```

Load data

Since we want to look at a longer time range, Landsat 8 data is suitable. In the new cell below, enter the following code, and then run it to load Landsat 8 data.

Notice `lat_range`, `lon_range` and `time` were all defined in the previous cell, so we can use them as variables here.

Note: If you are unsure where we defined `lat_range`, `lon_range` and `time`, scroll up to the previous cell and look for the lines starting with `lat_range = ...`, `lon_range = ...` and `time =`

```
landsat_ds = load_ard(
    dc=dc,
    products=["ls8_sr"],
    lat=lat_range,
    lon=lon_range,
    time=time,
    output_crs="EPSG:6933",
    resolution=(-30, 30),
    align=(15, 15),
    group_by='solar_day',
    measurements=['nir', 'red', 'blue'],
    min_gooddata=0.7)
```

The output from running the `load_ard()` function should include a statement that says Loading 46 time steps.

Note: As of June 2021, DE Africa Landsat data has been upgraded to Collection 2. Datacube names have been updated to `ls5_sr`, `ls7_sr` and `ls8_sr`. Deprecated naming conventions such as `ls8_usgs_sr_scene` will no longer work. For more information on Landsat Collection 2, visit the [DE Africa Landsat documentation](#).

5.2.3 Calculate indices

Now we need to calculate a vegetation index.

Until now, we have used NDVI, which uses the ratio of the red and near-infrared (NIR) bands to identify live green vegetation. The formula is:

$$\text{NDVI} = \frac{\text{NIR} - \text{Red}}{\text{NIR} + \text{Red}}$$

This time we will use the Enhanced Vegetation Index (EVI). EVI uses the red, near-infrared (NIR) and blue bands to identify vegetation, and is particularly sensitive to high biomass regions, which is why it can be superior to NDVI. The formula for EVI is more complicated than NDVI as it uses three different bands and some empirical scaling constants.

$$\text{EVI} = \frac{2.5 \times (\text{NIR} - \text{Red})}{\text{NIR} + 6 \times \text{Red} - 7.5 \times \text{Blue} + 1}$$

Instead of typing out that whole formula, we can use the `calculate_indices` function to calculate EVI. `calculate_indices` requires three inputs:

- The dataset name, e.g. landsat_ds
- The name of the index to calculate, e.g. index='EVI'
- The Landsat collection number, e.g. collection='c2'

c2 stands for ‘Collection 2’, which is the currently-available Landsat collection of data as named by its publishers, US Geological Survey.

In the next cell, enter the following code, and then run it to calculate the EVI vegetation index for this data.

```
landsat_ds = calculate_indices(landsat_ds, index='EVI', collection='c2')
```

This adds a variable called EVI to our landsat_ds dataset.

5.2.4 Detect changes

We want to determine what changed in vegetation between the older and newer halves of the data.

First, we will split the vegetation index data into the older half and newer half. Data that was collected in the first half of our time range (2013 to 2015) will go in the older half, and data collected in the second half (2016 to 2018) will be in the newer half.

The split is done using sel() and slice().

- sel() stands for ‘selection’ and tells us we are taking a selection of the dataset. We have to define which coordinate we are selecting by. In this case, we will use the time coordinate.
- slice() specifies which part of the coordinate we are taking. In this case, we want to slice time between 2013 – 2015, and then again 2016 – 2018. Recall we named the halfway point time_baseline.

We use sel() and slice() to create two new datasets:

- baseline_sample: EVI from 2013 to 2015
- postbaseline_sample: EVI from 2016 to 2018

To do this, enter the following code in the next cell.

```
baseline_sample = landsat_ds.EVI.sel(time=slice(time[0], time_baseline))
postbaseline_sample = landsat_ds.EVI.sel(time=slice(time_baseline, time[1]))
```

Here, using time[0] will give us the first date that we stored in the time variable ('2013-01-01'), and time[1] will give us the second date we stored in the time variable ('2018-12-01'). By using the time variable directly, we ensure that the code will work if those dates are changed and the notebook is rerun.

Note: Carefully check all your ., , , () pairs, and ' ' pairs in the above code to avoid generating errors.

Detect per-pixel changes

Now we have our ‘before’ and ‘after’ datasets, we can compare them for change in EVI. To do this, we will form a composite for each half of the dataset. Then we will calculate the differences in the average EVI for each pixel.

The composite method we are using is the **mean** (average).

In the next cell, enter the following code, and then run it to create mean composites for the two time periods.

```
baseline_composite = baseline_sample.mean('time')
postbaseline_composite = postbaseline_sample.mean('time')
```

Now we need to subtract the first time period EVI mean composite, `baseline_composite`, from the second time period EVI mean composite, `postbaseline_composite`. This will determine the change in EVI between the two time periods.

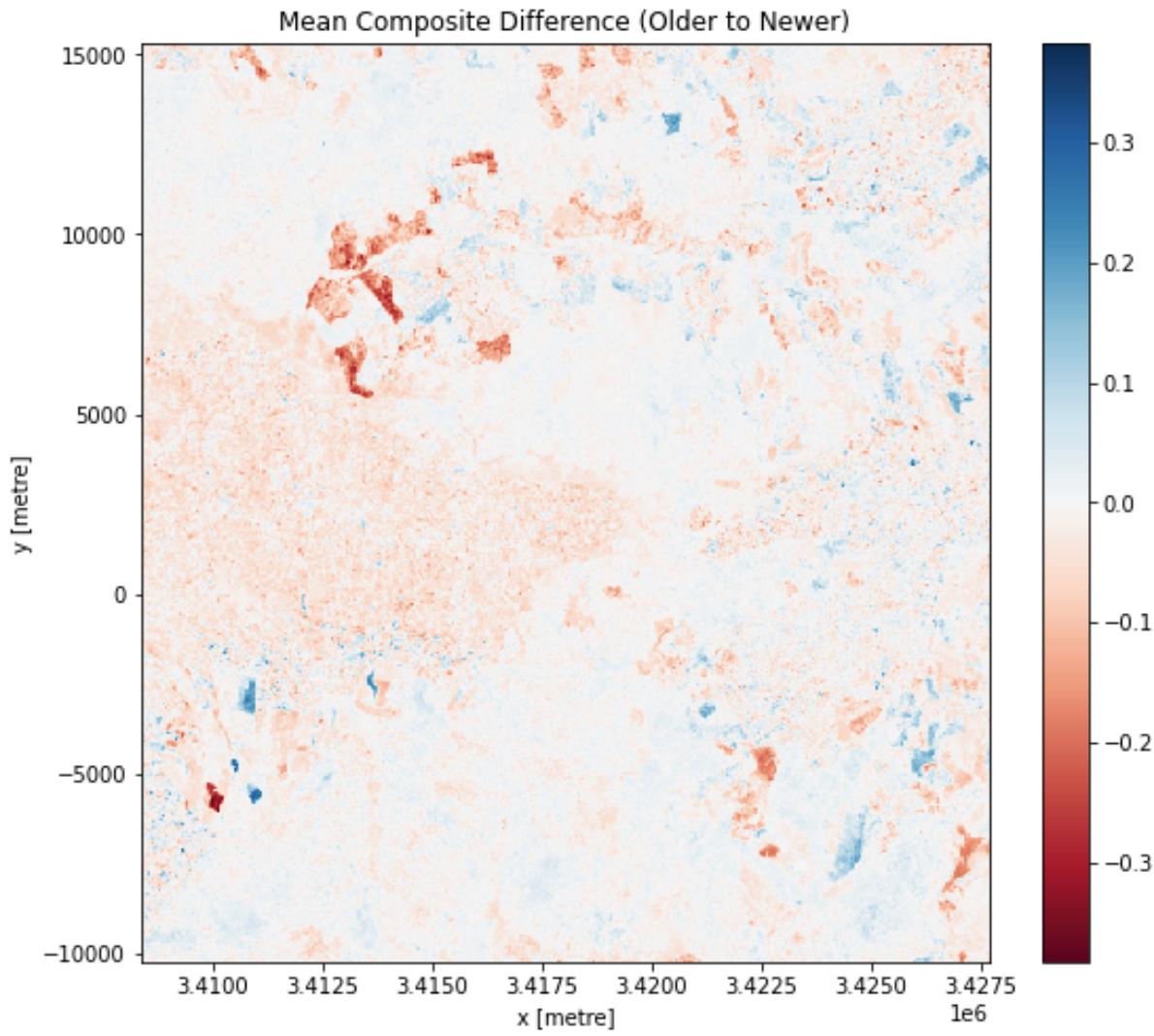
In the next cell, enter the following code, and then run it to determine the change in the vegetation index.

```
diff_mean_composites = postbaseline_composite - baseline_composite
```

In the next cell, enter the following code, and then run it to show the difference between the mean composites for the time periods. This will allow us to see where the vegetation index increased or decreased and by how much.

```
plt.figure(figsize=(9, 8))
diff_mean_composites.plot.imshow(cmap='RdBu')
plt.title("Mean Composite Difference (Older to Newer)")
plt.show()
```

Your plot should look like the image below.



Interpreting the plot

In the code above, the colour for the plot is set using the `cmap='RdBu'` setting in the `diff_mean_composites.plot.imshow()` function call. Here, RdBu corresponds to a Red-Blue colour-map, with lower values appearing as red, and higher values appearing as blue.

The areas in **blue** (positive change) correspond to **vegetation increase** as measured by EVI. There is more vegetation in these areas in the 2016-2018 sample than the 2013-2015 sample.

The areas in **red** (negative change) correspond to **vegetation decrease** as measured by EVI. There is less vegetation in these areas in the 2016-2018 sample than the 2013-2015 sample.

What conclusions can you draw about the changes in the landscape from this plot? What other information might you need to help you assess it?

5.2.5 Conclusion

Congratulations! You have made your own vegetation change detection notebook. It is comparable to [the existing Sandbox vegetation change detection notebook](#). The existing notebook may look daunting, but it includes many of the steps that you have just done! In addition to setting a ‘before’ and ‘after’ scene, the existing notebook:

- Plots some true-colour maps (RGB) to inspect the area of interest
- Quantifies change using statistical tests (Welch’s *t*-test for areas of unequal variance)
- Identifies statistically significant change

A difference plot, like the one we made above, is a good way to start. You can then decide if you need more complicated analysis or not. You now understand how to structure a complete case study — you can calculate a relevant band index and identify meaningful changes in that index over time.

5.2.6 Optional activity

If you’re curious about how the existing case study works, you can open and run it in the Sandbox:

1. From the main Sandbox folder, open the **Real_world_examples** folder
2. Double-click the **Vegetation_change_detection.ipynb** notebook to open it

The notebook has already been run, so you can read through it step by step. However, you may find it valuable to clear the outputs and run each cell step by step to see how it works. You can do this by clicking **Kernel -> Restart Kernel and Clear All Outputs**. When asked whether you want to restart the kernel, click **Restart**.

There are many similarities between the notebook you built in this session, and the existing Sandbox notebook. Make a note of what is similar and what is different, and spend some time inspecting the different code. If you have any questions about how the existing notebook works, please ask the instructors during a [Live Session](#).

5.3 Session 5 Quiz and Solution

In Session 5, you looked at the framework of a case study. Building on elements from previous sessions — such as loading data, calculating an index, and plotting results — you used a new index, the Enhanced Vegetation Index (EVI), to show differences over time. The first half of EVI data was compared to the second half, and mapped to show relative increase or decrease.

5.3.1 Quiz

If you would like to be awarded a certificate of achievement at the end of the course, we ask that you [complete the quiz](#). You will need to supply your email address to progress towards the certificate. After you complete the quiz, you can check if your answers were correct by pressing the **View Accuracy** button.

This quiz does not require a notebook to solve. However, you may find the EVI vegetation change detection notebook useful as a reference. If you would like to confirm that your vegetation change notebook works as expected, you can check it against the solution notebook provided below.

Note: The solution notebook below does not contain the answer to the quiz. Use it to check that you implemented the exercise correctly, then use your exercise notebook to help with the quiz. Accessing the solution notebook will not affect your progression towards the certificate.

5.3.2 Solution notebook

Note: We strongly encourage you to attempt the exercise on the previous page before downloading the solution below. This will help you learn how to use the Sandbox independently for your own analyses.

Download the solution notebook for the Session 5 exercise

To view a solution notebook on the Sandbox, you will need to first download it to your computer, then upload it to the Sandbox. Follow these instructions:

1. Download the notebook by clicking the link above.
2. On the Sandbox, open the **Training** folder.
3. Click the **Upload Files** button as shown below.



4. Select the downloaded notebook using the file browser. Click **OK**.
5. The solution notebook will appear in the **Training** folder. Double-click to open it.

SESSION 6: WATER ANALYSES

In this session, we will continue to combine many of the techniques you've learned during the training course to build a complete case study. This is where you have a question in mind that you want to answer, load and analyse the appropriate data, then draw conclusions to try and answer your question.

This session will focus on case studies related to water. First, we'll cover the available water and coastal case studies on the Sandbox, as well as covering the **Water Observations from Space (WOfs)** dataset. Then, you'll work through building your own case study to monitor water extent over time.

6.1 Introduction to water analyses

6.1.1 Overview

In the [*last session*](#), we learned how to construct a complete case study, focusing on vegetation. In this session, we will look at another vital resource — water. Using the same case study techniques, we will construct a new case study focusing on water. This includes learning to use the Water Observations from Space (WOfs) dataset.

6.1.2 The significance of water



Water is a key concern across the African continent. Many countries are experiencing extreme conditions, such as drought or floods. Water case study analyses can pave the way towards data-driven policy and mitigation strategies.

[[Image source](#)]

There are many useful analyses that involve water, such as:

- Detection of drought
- Flood modelling and prediction
- Mangrove analysis: mangroves can lessen damage from water surges such as those from storms and tsunamis
- Detection of coastal erosion
- Reporting for indicators in the United Nations (UN) Sustainable Development Goals (SDGs) that involve water. For example, [Goal 6](#) is about water and sanitation. Some example indicators that the Open Data Cube can facilitate reporting for include:
 1. Indicator 6.3.2: Proportion of bodies of water with good ambient water quality
 2. Indicator 6.6.1: Change in the extent of water-related ecosystems over time

6.1.3 Real world example notebooks

There are several notebooks in the Digital Earth Africa Sandbox that analyse water bodies. These case study notebooks can be found in the Sandbox folder called **Real_world_examples**.

The **Real_world_examples** folder contains many case studies; the notebooks related to water are summarised below.

- `Chlorophyll_monitoring.ipynb`

This notebook monitors chlorophyll-a in water bodies using Sentinel-2 data. Chlorophyll-a can indicate the presence of algal blooms, and monitoring this with satellite imagery can provide insights into water quality. The notebook plots the value of the Normalised Difference Chlorophyll Index (NDCI) along with water body area to understand how the presence of chlorophyll-a changes with time.

- `Coastal_erosion.ipynb`

This notebook monitors coastal erosion with Landsat 8 data. Using satellites to monitor coastal change over large areas can be a cost-effective method of tracking change and preempting infrastructure needs. This notebook extracts shorelines from composite measures of the Modified Normalised Difference Water Index (MNDWI) and plots how they change from year to year.

- `Intertidal_elevation.ipynb`

This notebook models tides with Landsat 5, 7, and 8 data. By examining the presence of water between low and high tide, the notebook builds up a 3D elevation map of inter-tidal zones such as sandy beaches and rocky shores.

- `Radar_water_detection.ipynb`

This notebook detects water using Sentinel-1 data. Sentinel-1 is a radar satellite, which is unaffected by cloud cover. This allows the detection of water in cloudy conditions, such as during heavy rain events.

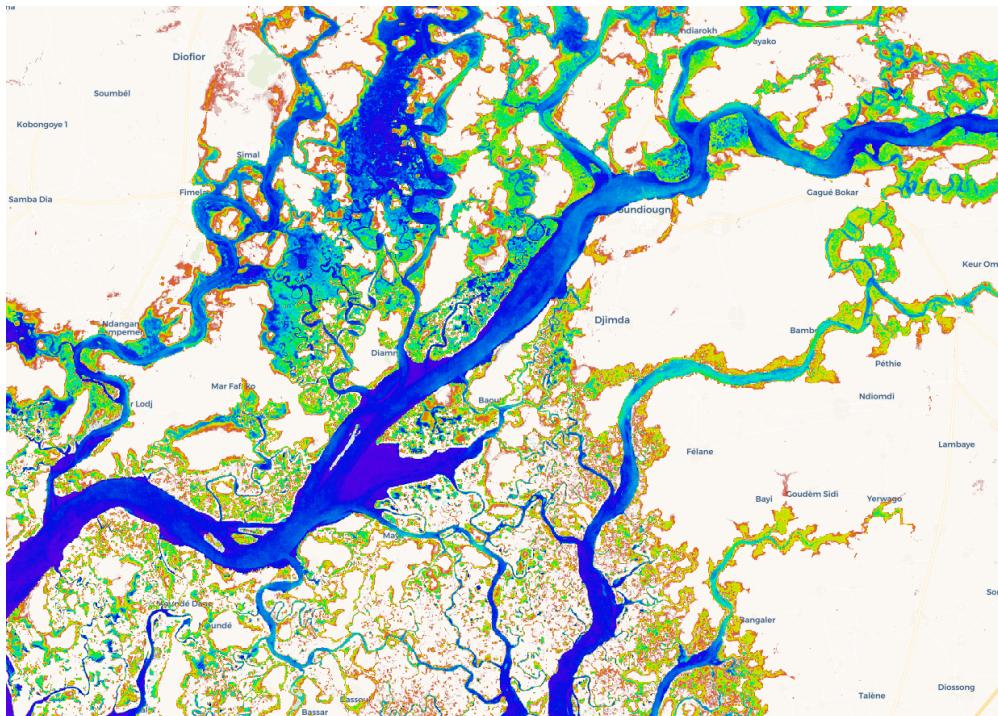
- `Water_extent.ipynb`

This notebook determines the extent of water bodies using the WOfS dataset, which is derived from Landsat data. This notebook relates to Sustainable Development Goals around the spatial extent of water systems, such as UN SDG Indicator 6.6.1. This is an important first step in monitoring the change of water systems over time.

The exercise in this session, Session 6, will contain instructions to create a similar notebook.

Note: It is recommended you open these notebooks in the Sandbox, from the folder called **Real_world_examples**. This will allow you to run the pre-prepared code for yourself. However, they can also be viewed at the [Digital Earth Africa notebook repository](#), which will display a read-only HTML version from GitHub.

6.1.4 Water detection with WOfS



An example of the Water Observations from Space (WOfS) dataset over Senegal.

While we can detect water using band indices, Digital Earth Africa provides the Water Observations from Space (WOfS) dataset. The algorithm that generates this dataset has been trained to detect water in Landsat 8 imagery. You can learn more about WOfS and its algorithm by accessing the **Datasets** folder on the Sandbox, and double-clicking the `Water_Observations_from_Space.ipynb` notebook to open it.

There are several WOfS products available through the Digital Earth Africa Sandbox. They can be used for analysis in the same way we have been loading and using Landsat and Sentinel satellite products. We summarise two WOfS products below — **feature layers** and **annual summary**.

Feature layers

The base product for WOfS is the feature layers product, `ga_ls8c_wofs_2`. It has a single measurement called `water`, which indicates how the algorithm classifies each pixel in the image. `water=0` means a dry pixel, and `water=128` is a water pixel.

There are a few different classifications for other types of pixels, such as cloud. These are also mapped to unique values.

- `water=128` - Water
- `water=64` - Cloud
- `water=0` - Dry

Note: Don't worry if this seems confusing, or you're not sure how to use this to find water in an image. We'll cover this during the exercise on the next page. If you want to read more about feature layers, open the **Frequently_used_code** folder on the Sandbox, then double-click the **Applying_WOfS_bitmasking.ipynb** notebook to open it.

Annual summary

Digital Earth Africa also has an annual summary WOfS product called [ga_ls8c_wofs_2_annual_summary](#). This product records how often a pixel was classified as water in a given year. This is useful for comparing the presence of water over different years. The WOfS annual summary dataset contains three measurements:

- **count_wet**: The number of times a given pixel was wet during the year.
- **count_clear**: The number times a given pixel was clear (not cloudy) during the year. A clear pixel could be either wet or dry.
- **frequency**: The ratio of **count_wet** to **count_clear**, which describes the percentage of time a pixel was wet out of all the times it could be seen during the year.

6.1.5 Conclusion

With a rapidly changing climate, it is incredibly important to be able to understand and analyse water resources. This can easily be done in the Digital Earth Africa Sandbox, which allows access to satellite datasets and products such as WOfS.

In the next section, we will see how to construct a case study investigating water extent using WOfS.

6.2 Exercise: Determining the extent of water bodies

6.2.1 Overview

In this exercise, we will create a new notebook to determine the extent of water bodies using the [Water Observation from Space \(WOfS\)](#) product. The WOfS product uses an automated water mapping algorithm to identify water in Landsat 8 images.

The notebook will include the following steps:

- Load the WOfS feature layer product and Landsat 8 data
- Identify water pixels from WOfS
- Plot a true-colour image using Landsat data
- Plot the water body area for the same area using WOfS data
- Customise the plots

At the conclusion of this exercise, you will be able to determine the extent of water bodies using the WOfS product.

6.2.2 Set up notebook

In your **Training folder**, create a new Python 3 notebook. Name it `Water_extent_exercise.ipynb`. For more instructions on creating a new notebook, see the [instructions from Session 2](#).

Load packages and functions

In the first cell, type the following code and then run the cell to import necessary Python dependencies.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import datacube

from deafrica_tools.datahandling import load_ard, wofs_fuser
from deafrica_tools.plotting import display_map, rgb
```

In this exercise, we import one new function, `wofs_fuser`. `wofs_fuser` ensures WOfS data between scenes is combined correctly.

Note: As of June 2021, the `deafrica_tools` package has replaced the deprecated `sys.path.append('.. / Scripts')` file import. For more information on `deafrica_tools`, visit the [DE Africa Tools module documentation](#).

Connect to the datacube

Enter the following code and run the cell to create our `dc` object, which provides access to the datacube.

```
dc = datacube.Datacube(app="water_extent_exercise")
```

Select area of interest

In the next cell, enter the following code, and then run it to select an area and time. In this exercise, we use a central point and buffer to define our area of interest, similar to what we did in the [Session 5 exercise](#).

The only difference here is that we provide a latitude buffer and a longitude buffer. In this example, they have the same value. However, choosing different buffer values allows you to select rectangular areas of interest, rather than squares.

```
# Define the area of interest
lat = -6.0873
lon = 35.1817

lat_buffer = 0.2
lon_buffer = 0.2

# Combine central lat, lon with buffer to get area of interest
lat_range = (lat - lat_buffer, lat + lat_buffer)
lon_range = (lon - lon_buffer, lon + lon_buffer)

# Define the year
time = '2018'
```

Note: Recall code lines beginning with # are comments. They help explain the code, and can be removed or added without impacting the actual Python scripts.

In the next cell, enter the following code, and then run it to show the area on a map. Since we have defined our area using the variables `lon_range` and `lat_range`, we can use those instead of typing out `(lat - lat_buffer, lat + lat_buffer)` and `(lon - lon_buffer, lon + lon_buffer)` again.

```
display_map(x=lon_range, y=lat_range)
```

Create query object

Notice `lat_range`, `lon_range` and `time` were all defined in the previous cell, so we can use them as variables here. We will use them to create a **query**.

The `query` variable below is a Python dictionary. It can be used to store parameters. Creating an object variable such as `query` makes it possible to reuse parameters in various functions that accept the same input parameters.

This is useful to us because we can use it to load the Landsat 8 data, and then use it again to load the WOfS data.

In the next cell, enter the following code, and then run it.

```
query = {  
    'x': lon_range,  
    'y': lat_range,  
    'resolution': (-30, 30),  
    'output_crs': 'EPSG:6933',  
    'group_by': 'solar_day',  
    'time': (time),  
}
```

Note: Notice the structure of the `query` dictionary is slightly different from `dc.load` or `load_ard`. Each parameter name is in quotes '' and is followed by a colon :.

6.2.3 Load data

In the next cell, we load the Landsat and WOfS datasets, naming them `ds_landsat` and `ds_wofs` respectively.

In the functions below, we can directly pass the `query` object using `**query` — this will give all the settings defined in `query` to the function.

The main benefit is that we can use the same `query` for both Landsat 8 and WOfS, which saves us typing it again and prevents us from making mistakes.

Load Landsat 8

For Landsat 8, we can use the `load_ard` function.

```
ds_landsat = load_ard(dc=dc,
                      products=['ls8_sr'],
                      measurements=['red', 'green', 'blue'],
                      **query)

ds_landsat
```

► Dimensions:	(time: 44, x: 1287, y: 1693)
▼ Coordinates:	
y	(y) float64 -7.498e+05 ... -8.005e+05
time	(time) datetime64[ns] 2018-01-04T07:51:05.647190 ... 2...
spatial_ref	0 int32 6933
x	(x) float64 3.375e+06 3.375e+06 ... 3.414e+06
▼ Data variables:	
red	(time, y, x) float32 nan nan nan nan ... nan nan nan nan
green	(time, y, x) float32 nan nan nan nan ... nan nan nan nan
blue	(time, y, x) float32 nan nan nan nan ... nan nan nan nan
▼ Attributes:	
crs :	EPSG:6933
grid_mapping :	spatial_ref

Load WOfS

For WOfS, we need to use the `dc.load` function.

```
ds_wofs = dc.load(product=["ga_ls8c_wofs_2"],
                   fuse_func=wofs_fuser,
                   **query
                  )

ds_wofs
```

► Dimensions:	time : 46, x : 1287, y : 1693
▼ Coordinates:	
time	(time) datetime64[ns] 2018-01-04T07:50:41.718027 ... 2...
y	(y) float64 -7.498e+05 ... -8.005e+05
x	(x) float64 3.375e+06 3.375e+06 ... 3.414e+06
spatial_ref	0 int32 6933
▼ Data variables:	
water	(time, y, x) uint8 64 64 64 64 64 ... 64 64 64 64 64
▼ Attributes:	
crs :	EPSG:6933
grid_mapping :	spatial_ref

6.2.4 Calculating water extent

Understanding the WOfS feature layers

WOfS feature layers are stored as a binary number, where each digit of the number is independently set or not based on the presence (1) or absence (0) of a particular feature. Below is a breakdown of which decimal value represent which features.

Attribute	Decimal value
No data	1
Non contiguous	2
Sea	4
Terrain or low solar angle	8
High slope	16
Cloud shadow	32
Cloud	64
Water	128

For example, a value of 128 indicates that water were observed for the pixel, whereas a value of 32 would indicate cloud shadow.

In the next cell we will extract only the water features from the WOfS data. This is done by finding values where the `water` measurement equals 128. In Python, we can find which pixels have a value of 128 by using the `==` expression:

Extract the water pixels

```
ds_valid_water = ds_wofs.water == 128
```

The `ds_valid_water` array does not contain the decimal values of the WOfS feature layers. Instead, it has a value of `False` if the pixel was not water, and `True` if it was water. You can check this by viewing the `ds_valid_water` DataArray.

```
ds_valid_water
```

Calculate the area per pixel

The number of pixels can be used for the area of the waterbody if the pixel area is known. We can extract the size of a pixel from the `resolution` setting in our query, then divide the area of a single pixel (in square metres) by the number of square meters in a square kilometre.

In Python, `number**2` returns the squared value of `number`.

```
pixel_length = query["resolution"][1] # in metres
m_per_km = 1000 # conversion from metres to kilometres
area_per_pixel = pixel_length**2 / m_per_km**2
```

Calculate area of water pixels

Now that we know how much area is covered by one pixel, we can count up the number of water pixels, and multiply it by this value to get the total area covered by water.

As we saw above, the `ds_valid_water` array contains `True` values for water pixels, and `False` otherwise. When we use the `.sum` function, it counts `True` values as 1, and `False` as 0. Therefore, the sum will be equal to the total number of water pixels for that timestep.

Below, we set the dimensions as `x` and `y` to make sure we sum up all the pixels over the spatial dimensions. This means we get one pixel sum for each timestep. This will let us track how the water area changes over time.

```
ds_valid_water_pixel_sum = ds_valid_water.sum(dim=['x', 'y'])
ds_valid_water_area = ds_valid_water_pixel_sum * area_per_pixel
```

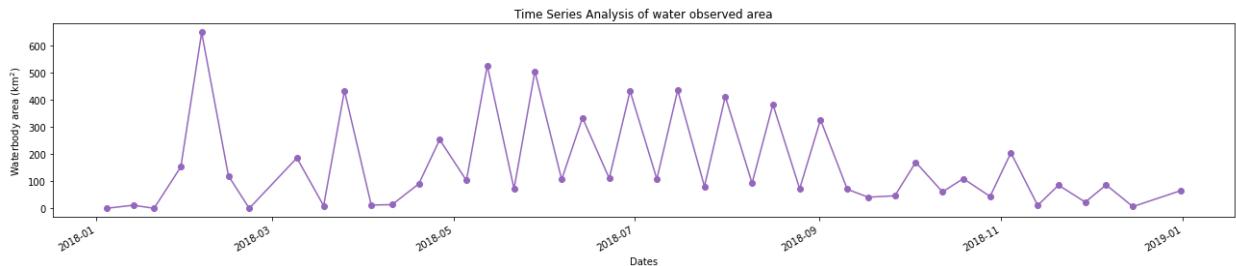
6.2.5 Plot time series

Now that we have the area of water in each observation, we can plot a time series to help us identify dates where there was more or less water within the area of interest.

Below, there is code to set-up, display and customise the plot. The settings are as follows:

- `plt.figure(figsize=(18, 4))`: set up a figure object to contain the plot, and make it 18 inches long and 4 inches high
- `ds_valid_water_area.plot(marker='o', color='#9467bd')`: plot the water area data with circular markers in purple (HEX colour code #9467bd)
- `plt.title('Time Series Analysis of water observed area')`: Give the plot a title
- `plt.xlabel('Dates')`: Label the x-axis
- `plt.ylabel('Waterbody area (km2)')`: Label the y-axis. The \$ symbols allow the use of LaTeX, a mathematical typesetting language
- `plt.tight_layout()`: Formats the image so that all axes can be clearly seen

```
plt.figure(figsize=(18, 4))
ds_valid_water_area.plot(marker='o', color='#9467bd')
plt.title('Time Series Analysis of water observed area')
plt.xlabel('Dates')
plt.ylabel('Waterbody area (km$^2$)')
plt.tight_layout()
```



6.2.6 Display of water coverage for a selected timestep

From the graph above you can choose any timestep (between 0 and 45) to display the result on the for both WOfS and Landsat 8.

For example, let us look at the fifth timestep, `timestep = 4`.

```
timestep = 4

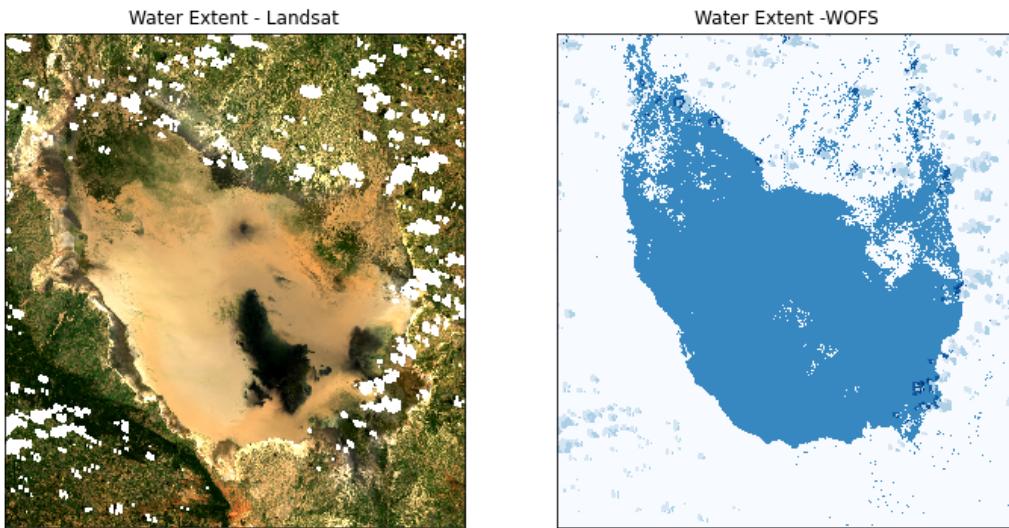
# Plot water extent
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

#plot the true colour image
ds_nearest_landsat = ds_landsat.sel(time=ds_wofs.time.isel(time=timestep), method=
    ↪'nearest')
rgb(ds_nearest_landsat, ax=ax[0])

# plot the water extent from WOfS
ds_wofs.isel(time=timestep).water.plot.imshow(ax=ax[1], cmap="Blues", add_colorbar=False)

# Titles
ax[0].set_title("Water Extent - Landsat"), ax[0].xaxis.set_visible(False), ax[0].yaxis.
    ↪set_visible(False)
ax[1].set_title("Water Extent - WOFS"), ax[1].xaxis.set_visible(False), ax[1].yaxis.set_.
    ↪visible(False)

plt.show()
```



This code uses some additional settings to customise the plot, including allowing to have two plots together. If you want to know more about making this kind of plot, please ask the instructors during a live session.

Try different `timestep` values — can you find an image where the lake is dried out?

6.2.7 Conclusion

Congratulations! You have made your own water extent notebook. It is comparable to [the existing Sandbox water extent notebook](#).

You've now built your second case study! You make like to reflect on what was similar and different between the two. Are there any pieces of code you could reuse for a new analysis? How might you modify your case studies to do more complex analysis?

If you'd like to experiment futher, try running the code with different areas. Did you learn anything interesting to share with us?

6.2.8 Optional activity

If you're curious about how the existing case study works, you can open and run it in the Sandbox:

1. From the main Sandbox folder, open the **Real_world_examples** folder
2. Double-click the **Water_extent.ipynb** notebook to open it

The notebook has already been run, so you can read through it step by step. However, you may find it valuable to clear the outputs and run each cell step by step to see how it works. You can do this by clicking **Kernel -> Restart Kernel and Clear All Outputs**. When asked whether you want to restart the kernel, click **Restart**.

Note: If you want to significantly modify it, we recommend you make a copy, like you did in [Session 1](#).

There are many similarities between the notebook you built in this session, and the existing Sandbox notebook. Maybe make a note of what is similar and what is different. If you have any questions about how the existing notebook works, please ask the instructors during a live session.

6.3 Session 6 Quiz and Solution

In Session 6, you used the case study framework to construct an analysis on water extent. This involved using the Water Observations from Space (WOoS) dataset. The WOoS dataset identifies water pixels, which you used to calculate the area of water. The water extent was plotted next to an RGB map of the same area.

6.3.1 Quiz

If you would like to be awarded a certificate of achievement at the end of the course, we ask that you [complete the quiz](#). You will need to supply your email address to progress towards the certificate. After you complete the quiz, you can check if your answers were correct by pressing the **View Accuracy** button.

If you have successfully passed all six assessment quizzes, you will be awarded a Certificate of Completion. If you have any questions about the Certificate of Completion, please see the [FAQ](#).

The quiz will ask you to use the notebook you developed for this session's exercise on plotting water extent using WOoS data. If you would like to confirm that your notebook works as expected, you can check it against the solution notebook provided below.

Note: The solution notebook below does not contain the answer to the quiz. Use it to check that you implemented the exercise correctly, then use your exercise notebook to help with the quiz. Accessing the solution notebook will not affect your progression towards the certificate.

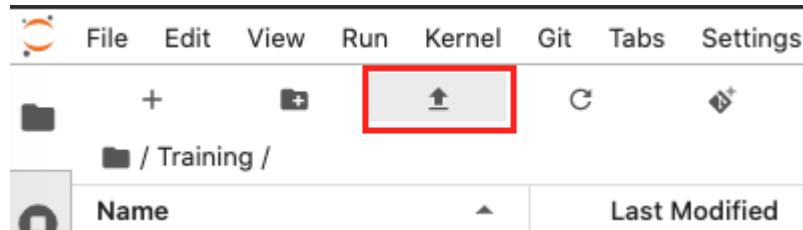
6.3.2 Solution notebook

Note: We strongly encourage you to attempt the exercise on the previous page before downloading the solution below. This will help you learn how to use the Sandbox independently for your own analyses.

[Download the solution notebook for the Session 6 exercise](#)

To view a solution notebook on the Sandbox, you will need to first download it to your computer, then upload it to the Sandbox. Follow these instructions:

1. Download the notebook by clicking the link above.
2. On the Sandbox, open the **Training** folder.
3. Click the **Upload Files** button as shown below.



4. Select the downloaded notebook using the file browser. Click **OK**.
5. The solution notebook will appear in the **Training** folder. Double-click to open it.

COURSE CONCLUSION & WRAP-UP

7.1 Congratulations! You have reached the end of the Digital Earth Africa training course.

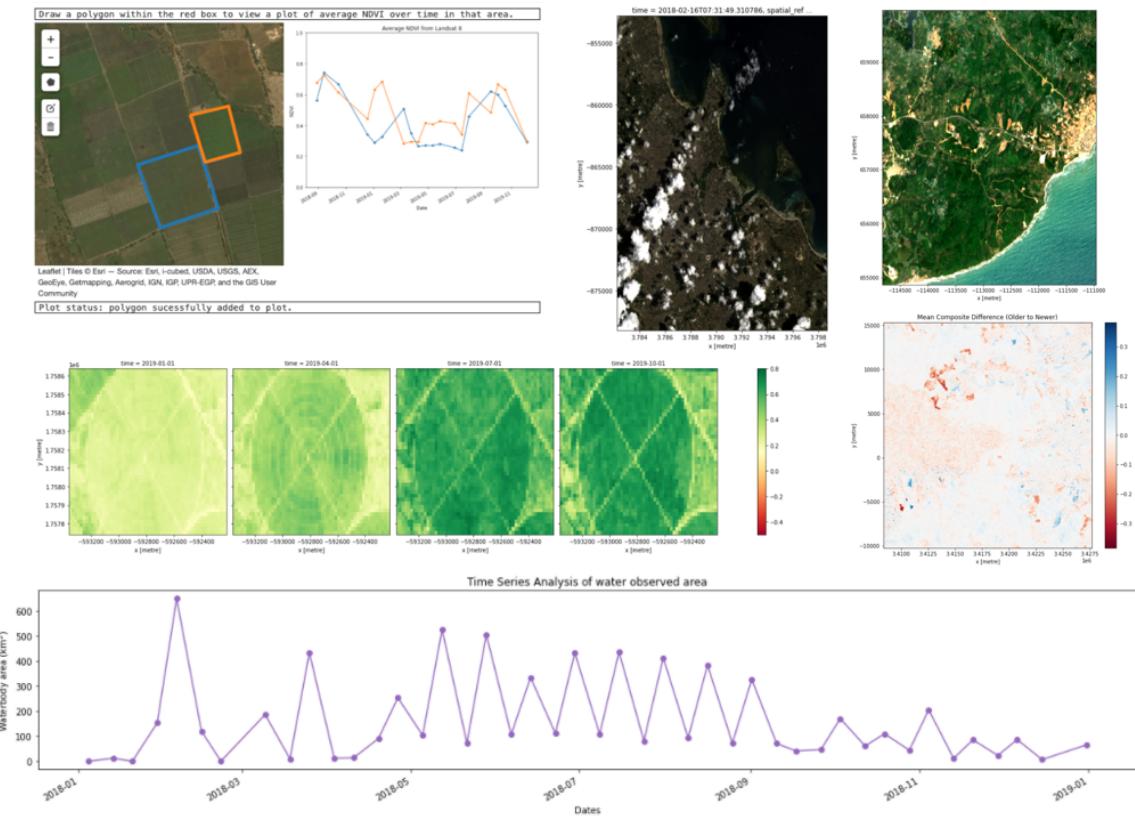
Over the six sessions, you have learnt how to

- Run data analysis notebooks in the Digital Earth Africa Sandbox.
- Check data availability for an area of interest using the Digital Earth Africa Maps portal and Metadata Explorer.
- Load Earth observation data from a variety of products, including Landsat 8 and Sentinel-2.
- Recognise the importance of composite images in providing cloud-free scenes.
- Understand band indices and their role in multi-spectral data analysis.
- Select a suitable band index for analysis purposes.
- Visualise data by plotting maps.

In the Sandbox, you are able to

- Set up a notebook with common packages and functions.
- Connect to the Open Data Cube to retrieve data.
- Load data both with and without cloud masking.
- Create geomedian composites.
- Calculate band indices such as NDVI.
- Construct complete case studies based on vegetation or water indices.

This is a significant achievement. Well done!



A snapshot of some of the different plots you have created over the course of the six training sessions.

7.2 What next?

You now have insight to Digital Earth Africa, and experience with the Sandbox. There are many ways you can take this further. To continue expanding your skills, here are some suggestions on exploring the platform capabilities and functions.

Venturing outside of training material content means there is not always a solutions notebook available. Not to worry — if you run into any trouble or have any questions, you can continue to attend [live sessions](#) or contact the course convenors. There are always people ready to help!

7.2.1 1. Complete the training course participant survey

Your feedback, suggestions and ideas are very important to us. Filling in this [short survey](#) will provide us with insight to continue refining and developing training material tailored for Digital Earth Africa users.

7.2.2 2. Modify case studies for your area of interest

In Sessions 5 and 6, we built case studies looking at vegetation change with EVI, and water extent over time with Water Observations from Space (WOFS).

Make a copy of those notebooks and change the area of interest. You could investigate:

- NDVI or EVI over an area of cropland in your country
- Water extent fluctuations over dams or water catchments

You can also try selecting a different index from the `calculate_indices` function. The full list of available indices can be viewed by accessing the **Scripts** folder in the Sandbox, and opening `deafrica_bandindices.py`.

7.2.3 3. Consolidate your Python knowledge using the Python basics and Beginner's Guide tutorials

During the six sessions, we gradually introduced Python coding concepts with hands-on examples. If you would like to study the theory behind some of the functions and commands we used, have a look at the [*Extra session: Python basics*](#) module, and the **Beginners_guide** folder in the Sandbox.

The **Beginners_guide** series of notebooks provide background information, code examples, and a range of applications. Some of this material will be familiar to you, but many of the notebooks contain extra details that can help solidify your understanding of the Sandbox mechanics.

7.2.4 4. Explore real world examples

The case studies we created in Sessions 5 and 6 were based off more complex analyses documented in the **Real_world_examples** folder of the Sandbox.

These case studies often have a few more data processing steps than the exercises in the training course. After reviewing the Beginner's Guide contents, these steps should be more accessible and easy to understand.

Use the real world example notebooks to inspire your analysis and improve your data processing technique.

7.2.5 5. Advance your analysis with code snippets

Real world examples are great for generating ideas and providing examples of case study structure. However, when you are writing your own case studies, you may find code snippets in the **Frequently_used_code** folder even more helpful.

The **Frequently_used_code** folder contains a series of notebooks demonstrating code doing useful and common functions. This includes plotting boundary or contour lines, exporting files, incorporating external datasets and masking data. They can be easily adapted into your own notebooks.

7.3 Keep in contact

Thank you for joining us to explore Digital Earth Africa. We hope you found the training course helpful and informative. If you have any questions or feedback on the Digital Earth Africa training course, don't hesitate to *contact the course convenors or access the community knowledge bank on Slack*.

We are always keen to see what our users have created — feel free to share your results by tagging us on Twitter at #DigitalEarthAfrica.

**CHAPTER
EIGHT**

EXTRA SESSION: PYTHON BASICS

Completion of this section is optional. All six DE Africa Training Course sessions can be successfully completed without this module.

This section gives an introduction to Python, a popular programming language. Python is used in the Digital Earth Africa Sandbox platform to extract data, perform analysis, and display results.

This section is intended to provide training course participants with more context around the common commands we use in the Sandbox. In particular, it targets functions specific to geospatial data analysis. The training course can be completed without reviewing this section, but users may find it helpful to better understand some of the code in the later sections.

The Python basics module consists of five separate lessons. You can come back to the lessons at any time. They are not intended as a comprehensive Python training course. We do not endorse any particular generic Python training course and urge interested users to find a course which aligns with their needs.

Users who are short on time or already confident using Python are welcome to skip this section, although we suggest looking over the content to make sure it is familiar to you.

Structure of each Python basics lesson page

- Brief overview of tutorial contents
- Tutorial file download: user to download and open the file in the Sandbox as instructed. The tutorial notebook is interactive
- Tutorial walkthrough (not interactive), to be followed in tandem with the downloaded file
- Exercises to check understanding
- Conclusion

An internet connection will be required to use the interactive tutorial notebooks and exercises on the Sandbox platform.

Click on **Python basics 1: Jupyter** below or select **Next** to get started.

8.1 Python basics 1: Jupyter

The first lesson of the Python basics session is on performing basic Python commands in a Jupyter Notebook. We will be using a prepared tutorial notebook to guide you through the steps.

8.1.1 Set-up prerequisites

The Python basics interactive tutorials use the Digital Earth Africa Sandbox platform. This requires a Digital Earth Africa Sandbox account.

To set up the learning environment, please complete all sections in *Session 1: Introduction* before starting this Python basics module.

Then, follow the instructions below to download the tutorial and open it in the Sandbox.

8.1.2 Download the tutorial notebook

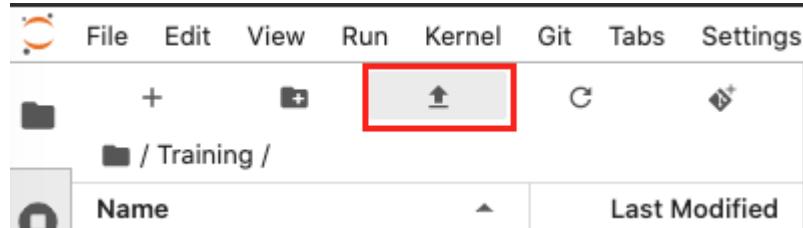
Download the Python basics 1 tutorial notebook

To view this notebook on the Sandbox, you will need to first download it to your computer, then upload it to the Sandbox.

Note: Ensure you have created a **Training** folder in the Sandbox as instructed in *Session 1: Running a Notebook*.

Once you have created a **Training** folder, follow these instructions:

1. Download the notebook by clicking the link above.
2. On the Sandbox, open the **Training** folder.
3. Click the **Upload Files** button as shown below.



4. Select the downloaded notebook using the file browser. Click **OK**.
5. The solution notebook will appear in the **Training** folder. Double-click to open it.

You can now use the tutorial notebook as an interactive version of this webpage.

Note: The tutorial notebook should look like the text and code below. However, the tutorial notebook outputs are blank (i.e. no results showing after code cells). Follow the instructions in the notebook to run the cells in the tutorial notebook. Refer to this page to check your outputs look similar.

8.1.3 Recap: Jupyter Notebooks

First, we will review the Python commands introduced in Session 1. This lesson provides more examples and additional detail to the explanations.

A Jupyter notebook is an interactive environment where you can combine text with Python code that you can modify and execute. The gray cell below contains Python code that you can run. Place the cursor on the **Run** icon || in the menu at the top of the notebook to execute the code inside.

Tip: Use the shortcut Shift+Enter for running the active cell (where your cursor is). It is a convenient trick that saves you from having to click on the **Run** icon each time.

Select and run the grey code cell below.

```
[10]: print("Welcome to the Digital Earth Africa Sandbox!")
```

You'll see that once the code has executed, any output generated by your code will show up just below the cell.

```
[11]: print("3,2,1")
```

```
# # Use the hash symbol to comment lines of your code. This does not get executed.  
# # Use comments to make notes about what your code is doing.
```

```
print("this is fun!")
```

We can do mathematical computation using Python. Below, we have assigned the sum to a variable. We have chosen to name this variable `result`.

```
[12]: # Set this sum as a variable called result  
result = 999 + 1
```

```
# The output of result is displayed once the computation is completed  
result
```

The []: symbol to the left of each code cell describes the state of the cell:

- []: means that the cell has not been run yet.
- [*]: means that the cell is currently running.
- [1]: means that the cell has finished running and was the first cell run.

Sometimes, the code that you run in a cell takes a while to compute because it is loading a large dataset or performing complex computations. You'll notice the number that appears next to the cell. Before the cell is run you'll see the symbol [] meaning that cell has not been executed yet. While a cell is running it shows the [*] symbol and once it has completed running you'll see a number representing the number of cells being run, for example [4]. This allows you to keep track of the cells that have been run and their relative order.

Note: To check whether a cell is currently executing in a Jupyter notebook, inspect the small circle in the top-right of the window. The circle will turn black ("Kernel busy") when the cell is running, and return to white ("Kernel idle") when the process is complete.

Consider this Python program. Here, we have decided to call our variable `a`.

```
a = 1  
print(a)  
a = a + 1  
print(a)
```

We can break down this program and execute each line using separate cells.

Note: Python is case sensitive. The variable `a` is not the same as `A`.

```
[4]: a = 1
```

```
[13]: print(a)
```

```
[6]: a = a + 1
```

```
[14]: print(a)
```

If you run again the first a cell you'll see that it returns 2, the updated value. This is called 'global state', and means that once a variable is declared it is accessible anywhere in the notebook, even in cells above where it has been declared. This is different to traditional programs which execute sequentially line by line from the top to the bottom. This can be confusing in the beginning, but keep an eye on the number in the brackets [] to see cell execution order.

This also means you can modify all the code in the cells and run them as many times as you want in any order. You can jump back and forth to update variables or re-run analysis.

8.1.4 Exercises

1.1 Fill the asterisk line with your name and run the cell.

```
[15]: # Fill the ***** space with your name and run the cell.  
  
message = "My name is *****"  
  
message
```

1.2 You can add new cells to insert new code at any point in a notebook. Click on the + icon in the top menu to add a new cell below the current one. Add a new cell below the next cell, and use it to print the value of variable a.

```
[9]: a = 365*24  
  
# Add a new cell just below this one. Use it to print the value of variable `a`
```

Question: Now what happens if you scroll back up the notebook and execute a different cell containing `print(a)`?

8.1.5 Conclusion

Well done on finishing the recap lesson on Jupyter Notebooks. You are now ready to learn about Numpy! Click **Next** to continue.

8.2 Python basics 2: Numpy

This tutorial introduces numpy (pronounced *num-pye*, rhymes with *eye*), a Python library for performing numerical computations in Python. We will learn how to create and manipulate numpy arrays, which are useful matrix-like structures for holding large amounts of data.

Follow the instructions below to download the tutorial and open it in the Sandbox.

8.2.1 Download the tutorial notebook

Download the Python basics 2 tutorial notebook

To view this notebook on the Sandbox, you will need to first download it to your computer, then upload it to the Sandbox. Ensure you have followed the set-up prerequisites listed in [Python basics 1: Jupyter](#), and then follow these instructions:

1. Download the notebook by clicking the link above.
2. On the Sandbox, open the **Training** folder.
3. Click the **Upload Files** button as shown below.



4. Select the downloaded notebook using the file browser. Click **OK**.
5. The solution notebook will appear in the **Training** folder. Double-click to open it.

You can now use the tutorial notebook as an interactive version of this webpage.

Note: The tutorial notebook should look like the text and code below. However, the tutorial notebook outputs are blank (i.e. no results showing after code cells). Follow the instructions in the notebook to run the cells in the tutorial notebook. Refer to this page to check your outputs look similar.

8.2.2 Introduction to numpy

In order to be able to use numpy we need to import the numpy library using the special word `import`. To avoid typing `numpy` every time we want to use one of its functions, we can provide an alias using the special word `as`. We will nickname numpy as `np`:

[1]: `import numpy as np`

Note: If we do not `import numpy`, we cannot use any of the numpy functions. If you forget to import packages, you may get an error that says `name is not defined`.

Now, we have access to all the functions available in numpy by typing `np.name_of_function`. For example, the equivalent of `1 + 1` in Python can be done in numpy:

[2]: `np.add(1, 1)`

[2]: 2

By default the result of a function or operation is shown underneath the cell containing the code. If we want to reuse this result for a later operation we can assign it to a variable. For instance, let us call the variable `a`:

[3]: `a = np.add(2, 3)`

We have just declared a variable `a` that holds the result of the function. We can now use of display this variable, at any point of this notebook. For example we can show its contents by typing the variable name in a new cell:

[4]: a

[4]: 5

One of numpy's core concepts is the array. They can hold multi-dimensional data. To declare a numpy array explicitly we do:

[5]: np.array([1,2,3,4,5,6,7,8,9])

[5]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])

Note: The array defined above has only 1 dimension.

Most of the functions and operations defined in numpy can be applied to arrays. For example, with the previous add operation:

[6]: arr1 = np.array([1,2,3,4])
arr2 = np.array([3,4,5,6])

np.add(arr1, arr2)

[6]: array([4, 6, 8, 10])

We can also add arrays using the following convenient notation:

[7]: arr1 + arr2

[7]: array([4, 6, 8, 10])

Arrays can be sliced and diced. We can get subsets of the arrays using the indexing notation which is [start : end : stride]. Let's see what this means:

[8]: arr = np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15])

```
print(arr[5])  
print(arr[5:])  
print(arr[:5])  
print(arr[::-2])
```

```
5  
[ 5  6  7  8  9 10 11 12 13 14 15]  
[0 1 2 3 4]  
[ 0  2  4  6  8 10 12 14]
```

Experiment playing with the indexes to understand the meaning of `start`, `end` and `stride`. What happens if you don't specify a `start`? What value does numpy use instead?

Note: Numpy indexes start on 0, the same convention used in Python lists.

Indexes can also be negative, meaning that you start counting by the end. For example, to select the last 2 elements in an array we can do:

[9]: arr = np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15])

arr[-2:]

[9]: array([14, 15])

Numpy arrays can have multiple dimensions. Dimensions are indicated using nested square brackets []. The convention in numpy is that the outer [] represent the first dimension and the innermost [] contains the last dimension.

3D array

1D array

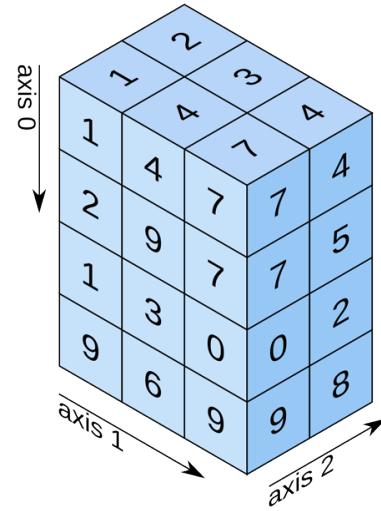
7	2	9	10
---	---	---	----

axis 0 →

2D array

axis 0 ↓	axis 1 →
5.2	3.0
9.1	0.1

shape: (2, 3)



shape: (4, 3, 2)

The following cell declares a 2-dimensional array with shape (1, 9).

Tip: Notice the nested (double) square brackets [[]]. As there are two brackets, this indicates the array is 2-dimensional.

```
[10]: np.array([[1,2,3,4,5,6,7,8,9]])
```

```
[10]: array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

To visualise the shape (dimensions) of a numpy array we can add the suffix `.shape` to an array expression or variable containing a numpy array.

```
[11]: arr1 = np.array([1,2,3,4,5,6,7,8,9])
arr2 = np.array([[1,2,3,4,5,6,7,8,9]])
arr3 = np.array([[1],[2],[3],[4],[5],[6],[7],[8],[9]])

arr1.shape, arr2.shape, arr3.shape, np.array([1,2,3]).shape
```

```
[11]: ((9,), (1, 9), (9, 1), (3,))
```

Numpy arrays can contain numerical values of different types. These types can be divided in these groups:

- Integers
 - Unsigned
 - * 8 bits: `uint8`
 - * 16 bits: `uint16`
 - * 32 bits: `uint32`
 - * 64 bits: `uint64`
 - Signed

- * 8 bits: int8
- * 16 bits: int16
- * 32 bits: int32
- * 64 bits: int64
- Floats
 - 32 bits: float32
 - 64 bits: float64

We can look up the type of an array by using the `.dtype` suffix.

```
[12]: arr = np.ones((10, 10, 10))

arr.dtype
```



```
[12]: dtype('float64')
```

Numpy arrays normally store numeric values but they can also contain boolean values, 'bool'. Boolean is a data type that can have two possible values: True, or False. For example:

```
[13]: arr = np.array([True, False, True])

arr, arr.shape, arr.dtype
```



```
[13]: (array([ True, False,  True]), (3,), dtype('bool'))
```

We can operate with boolean arrays using the numpy functions for performing logical operations such as and and or.

```
[14]: arr1 = np.array([True, True, False, False])
arr2 = np.array([True, False, True, False])

print(np.logical_and(arr1, arr2))
print(np.logical_or(arr1, arr2))

[ True False False False]
[ True  True  True False]
```

These operations are conveniently offered by numpy with the symbols * (and), and + (or).

Note: Here the * and + symbols are not performing multiplication and addition as with numerical arrays. Numpy detects the type of the arrays involved in the operation and changes the behaviour of these operators.

```
[15]: print(arr1 * arr2)
print(arr1 + arr2)

[ True False False False]
[ True  True  True False]
```

Boolean arrays are often the result of comparing a numerical arrays with certain values. This is sometimes useful to detect values that are equal, below or above a number in a numpy array. For example, if we want to know which values in an array are equal to 1, and the values that are greater than 2 we can do:

```
[16]: arr = np.array([1, 3, 5, 1, 6, 3, 1, 5, 7, 1])

print(arr == 1)
print(arr > 2)
```

```
[ True False False  True False False  True False False  True]
[False  True  True False  True  True False  True  True False]
```

You can use a boolean array to mask out False values from a numeric array. The returned array only contains the numeric values which are at the same index as True values in the mask array.

```
[17]: arr = np.array([1,2,3,4,5,6,7,8,9])
mask = np.array([True,False,True,False,True,False,True,False])

arr[mask]

[17]: array([1, 3, 5, 7, 9])
```

8.2.3 Exercises

2.1 Use the numpy add function to add the values 34 and 29 in the cell below.

```
[ ]: # Use numpy add function to add 34 and 29
```

2.2 Declare a new array with contents [5,4,3,2,1] and slice it to select the last 3 items.

```
[ ]: # Substitute the ? symbols by the correct expressions and values

# Declare the array

arr = ?

# Slice array for the last 3 items only

arr[?:?]
```

2.3 Select all the elements in the array below excluding the last one, [15].

```
[ ]: arr = np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15])

# Substitute the ? symbols by the correct expressions and values

arr[?]
```

2.4 Use arr as defined in 2.3. Exclude the last element from the list, but now only select every 3rd element. Remember the third index indicates stride, if used.

Hint: The result should be [0, 3, 6, 9, 12].

```
[ ]: # Substitute the ? symbols by the correct expressions and values  
arr[?:?:?]
```

2.5 You'll need to combine array comparisons and logical operators to solve this one. Find out the values in the following array that are greater than 3 AND less than 7. The output should be a boolean array.

Hint: If you are stuck, reread the section on boolean arrays.

```
[ ]: arr = np.array([1, 3, 5, 1, 6, 3, 1, 5, 7, 1])  
  
# Use array comparisons (<, >, etc.) and logical operators (*, +) to find where  
# the values are greater than 3 and less than 7.  
  
boolean_array = ?
```

2.6 Use your boolean array from 2.5 to mask the False values from arr.

Hint: The result should be [5, 6, 5].

```
[ ]: # Use your resulting boolean_array array from 2.5  
# to mask arr as defined in 2.5
```

8.2.4 Conclusion

Numpy is a fundamental numerical computing library in Python programming and it is useful to understand how it works. Next, we explore plotting geospatial data using matplotlib.

8.3 Python basics 3: Matplotlib

This tutorial introduces matplotlib, a Python library for plotting numpy arrays as images. We will learn how to:

Follow the instructions below to download the tutorial and open it in the Sandbox.

8.3.1 Download the tutorial notebook

Download the Python basics 3 tutorial notebook

Download the exercise image file

To view this notebook on the Sandbox, you will need to first download the notebook and the image to your computer, then upload both of them to the Sandbox. Ensure you have followed the set-up prerequisites listed in [Python basics 1: Jupyter](#), and then follow these instructions:

1. Download the notebook by clicking the first link above. Download the image by clicking the second link above.
2. On the Sandbox, open the **Training** folder.
3. Click the **Upload Files** button as shown below.



4. Select the downloaded notebook using the file browser. Click **OK**.
5. Repeat to upload the image file to the **Training** folder. It may take a while for the upload to complete.
6. Both files will appear in the **Training** folder. Double-click the tutorial notebook to open it and begin the tutorial.

You can now use the tutorial notebook as an interactive version of this webpage.

Note: The tutorial notebook should look like the text and code below. However, the tutorial notebook outputs are blank (i.e. no results showing after code cells). Follow the instructions in the notebook to run the cells in the tutorial notebook. Refer to this page to check your outputs look similar.

8.3.2 Introduction to matplotlib's pyplot

We are going to use part of matplotlib called pyplot. We can import pyplot by specifying it comes from matplotlib. We will abbreviate pyplot to plt.

```
[1]: %matplotlib inline
# Generates plots in the same page instead of opening a new window

import numpy as np
from matplotlib import pyplot as plt
```

Images are 2-dimensional arrays containing pixels. Therefore, we can use 2-dimensional arrays to represent image data and visualise with matplotlib.

In the example below, we will use the numpy arange function to generate a 1-dimensional array filled with elements from 0 to 99, and then reshape it into a 2-dimensional array using reshape.

```
[2]: arr = np.arange(100).reshape(10,10)

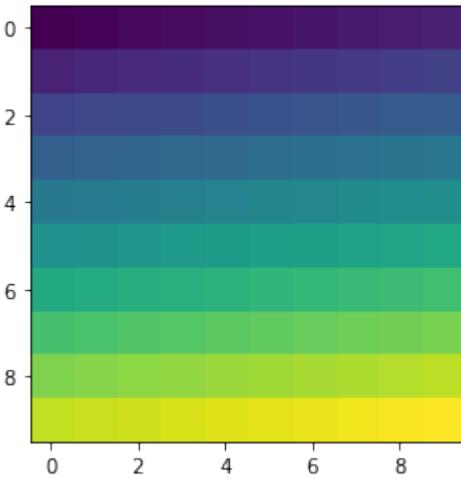
print(arr)
```

(continues on next page)

(continued from previous page)

```
plt.imshow(arr)  
[[ 0  1  2  3  4  5  6  7  8  9]  
 [10 11 12 13 14 15 16 17 18 19]  
 [20 21 22 23 24 25 26 27 28 29]  
 [30 31 32 33 34 35 36 37 38 39]  
 [40 41 42 43 44 45 46 47 48 49]  
 [50 51 52 53 54 55 56 57 58 59]  
 [60 61 62 63 64 65 66 67 68 69]  
 [70 71 72 73 74 75 76 77 78 79]  
 [80 81 82 83 84 85 86 87 88 89]  
 [90 91 92 93 94 95 96 97 98 99]]
```

[2]: <matplotlib.image.AxesImage at 0x7f33279840f0>



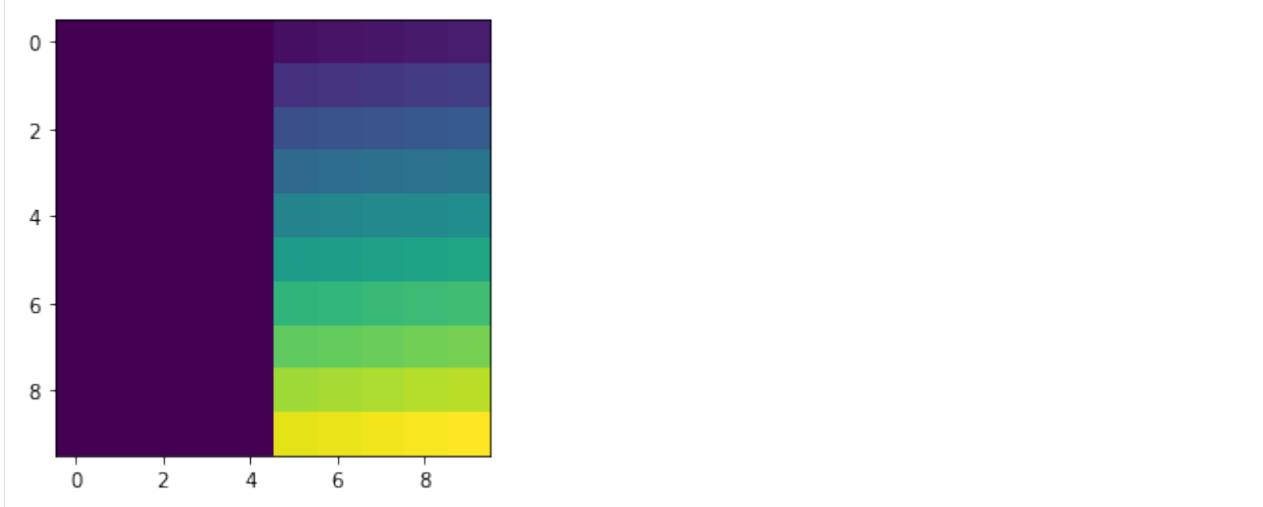
If you remember from the [last tutorial](#), we were able to address regions of a numpy array using the square bracket [] index notation. For multi-dimensional arrays we can use a comma , to distinguish between axes.

```
[ first dimension, second dimension, third dimension, etc. ]
```

As before, we use colons : to denote [start : end : stride]. We can do this for each dimension.

For example, we can update the values on the left part of this array to be equal to 1.

```
[3]: arr = np.arange(100).reshape(10,10)  
arr[:, :5] = 1  
  
plt.imshow(arr)  
[3]: <matplotlib.image.AxesImage at 0x7f33274d7198>
```



The indexes in the square brackets of `arr[:, :5]` can be broken down like this:

```
[ 1st dimension start : 1st dimension end, 2nd dimension start : 2nd dimension end ]
```

Dimensions are separated by the comma ,. Our first dimension is the vertical axis, and the second dimension is the horizontal axis. Their spans are marked by the colon :. Therefore:

```
[ Vertical start : Vertical end, Horizontal start : Horizontal end ]
```

If there are no indexes entered, then the array will take all values. This means `[:, :5]` gives:

```
[ Vertical start : Vertical end, Horizontal start : Horizontal start + 5 ]
```

Therefore the array index selected the first 5 pixels along the width, at all vertical values.

Now let's see what that looks like on an actual image.

Tip: Ensure you uploaded the file `Guinea_Bissau.JPG` to your **Training** folder along with the tutorial notebook. We will be using this file in the next few steps and exercises.

We can use the pyplot library to load an image using the matplotlib function `imread`. `imread` reads in an image file as a 3-dimensional numpy array. This makes it easy to manipulate the array.

By convention, the first dimension corresponds to the vertical axis, the second to the horizontal axis and the third are the Red, Green and Blue channels of the image. Red-green-blue channels conventionally take on values from 0 to 255.

```
[4]: im = np.copy(plt.imread('Guinea_Bissau.JPG'))

# This file path (red text) indicates 'Guinea_Bissau.JPG' is in the
# same folder as the tutorial notebook. If you have moved or
# renamed the file, the file path must be edited to match.

im.shape
```

[4]: (590, 602, 3)

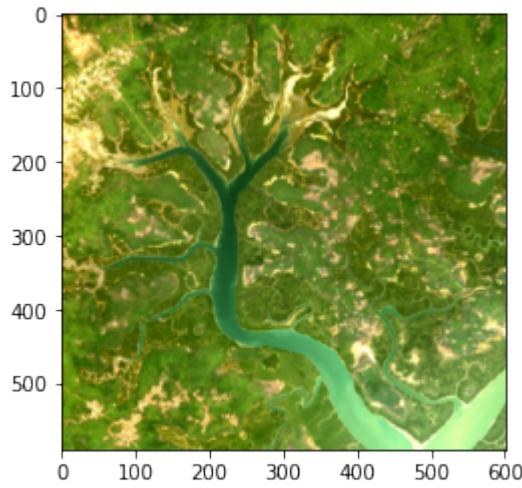
`Guinea_Bissau.JPG` is an image of Rio Baboque in Guinea-Bissau in 2018. It has been generated from Landsat 8 satellite data.

The results of the above cell show that the image is 590 pixels tall, 602 pixels wide, and has 3 channels. The three channels are red, green, and blue (in that order).

Let's display this image using the pyplot `imshow` function.

```
[5]: plt.imshow(im)
```

```
[5]: <matplotlib.image.AxesImage at 0x7f33273bb400>
```



8.3.3 Exercises

3.1 Let's use the indexing functionality of numpy to select a portion of this image. Select the top-right corner of this image with shape (200, 200).

Hint: Remember there are three dimensions in this image. Colons separate spans, and commas separate dimensions.

```
[ ]: # We already defined im above, but if you have not,
# you can un-comment and run the next line

# im = np.copy(plt.imread('Guinea_Bissau.JPG'))

# Fill in the question marks with the correct indexes

topright = im[?, ?, ?]

# Plot your result using imshow

plt.imshow(topright)
```

If you have selected the correct corner, there should be not much water in it!

3.2 Let's have a look at one of the pixels in this image. We choose the top-left corner with position $(0,0)$ and show the values of its RGB channels.

```
[ ]: # Run this cell to see the colour channel values
im[0,0]
```

The first value corresponds to the red component, the second to the green and the third to the blue. `uint8` can contain values in the range [0-255] so the pixel has a lot of red, some green, and not much blue. This pixel is a orange-yellow sandy colour.

Now let's modify the image.

What happens if we set all the values representing the blue channel to the maximum value?

```
[ ]: # Run this cell to set all blue channel values to 255
# We first make a copy to avoid modifying the original image

im2 = np.copy(im)

im2[:, :, 2] = 255

plt.imshow(im2)
```

The index notation `[:, :, 2]` is selecting pixels at all heights and all widths, but only the 3rd colour channel.

Can you modify the above code cell to set all red values to the maximum value of 255?

8.3.4 Conclusion

We have successfully practised indexing numpy arrays and plotting those arrays using matplotlib. We can now also read a file into Python using `pyplot.imread`. The next lesson covers data cleaning and masking.

8.4 Python basics 4: Cleaning data

This tutorial explores further concepts in Numpy such as categorical data, advanced indexing and dealing with Not-a-Number (NaN) data.

Follow the instructions below to download the tutorial and open it in the Sandbox.

8.4.1 Download the tutorial notebook

Download the Python basics 4 tutorial notebook

To view this notebook on the Sandbox, you will need to first download it to your computer, then upload it to the Sandbox. Ensure you have followed the set-up prerequisites listed in [Python basics 1: Jupyter](#), and then follow these instructions:

1. Download the notebook by clicking the link above.
2. On the Sandbox, open the **Training** folder.

3. Click the **Upload Files** button as shown below.



4. Select the downloaded notebook using the file browser. Click **OK**.
5. The solution notebook will appear in the **Training** folder. Double-click to open it.

You can now use the tutorial notebook as an interactive version of this webpage.

Note: The tutorial notebook should look like the text and code below. However, the tutorial notebook outputs are blank (i.e. no results showing after code cells). Follow the instructions in the notebook to run the cells in the tutorial notebook. Refer to this page to check your outputs look similar.

8.4.2 Numpy dictionaries and categorical data

We will introduce a numpy structure called a **dictionary**. This will be useful for the next lesson on **xarray**.

A dictionary represents a mapping between **keys** and **values**. The keys and values are Python objects of any type. We declare a dictionary using curly braces {}. Inside we specify the key then its associated value, with the keys and values separated by a colon :. Commas , are used to separate elements in the dictionary.

```
dictionary_name = {key1: value1, key2: value2, key3: value3}
```

For example:

```
[ ]: d = {1: 'one',
          2: 'two',
          3: 'apple'}
```

In the above dictionary d, we have three **keys** 1, 2, 3, and their respective **values** 'one', 'two' and 'apple'.

We can look up elements in a dictionary using the [key_name] to address the value stored under a key. The syntax looks like:

```
dictionary_name[key_name]
```

In our example dictionary d above, we can call upon the value associated with the key name 1 like so:

```
d[1]
```

```
[ ]: print(d[1], " + ", d[2], " = ", d[3])
```

Elements in a dictionary can be modified or new elements added by using the `dictionary_name[key_name] = value` syntax.

```
[ ]: d[3] = 'three'
      d[4] = 'four'
```

(continues on next page)

(continued from previous page)

```
print(d[1], " + ", d[2], " = ", d[3])
```

Again, the dictionary name, key name, and value must be specified.

Dictionaries are useful for data analysis (including satellite data analysis) because they make it easy to assign **categorical values** to our dataset. Remote sensing can be used to create classification products that use categorical values. These products do not contain continuous values. They use discrete values to represent different classes individual pixels can belong to.

As an example, the following cells simulate a very simple image containing three different land cover types. Value 1 represents area covered with grass, 2 croplands and 3 city.

First, we import the libraries we want to use.

```
[4]: %matplotlib inline

import numpy as np
from matplotlib import pyplot as plt
from matplotlib import colors
```

We will now create a 2-dimensional 100 pixel x 100 pixel numpy array where every value is 1. This is done using the `numpy.ones` function. Then, we use array indexing to assign part of the area to have the value 2, and another part to have the value 3.

```
[5]: # grass = 1
area = np.ones((100,100))

# crops = 2
area[10:60,20:50] = 2

# city = 3
area[70:90,60:80] = 3

area.shape, area.dtype
```

```
[5]: ((100, 100), dtype('float64'))
```

```
[6]: area
```

```
[6]: array([[1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       ...,
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.]])
```

We now have a matrix filled with 1s, 2s and 3s. At this point, there is no association between the numbers and the different types of ground cover.

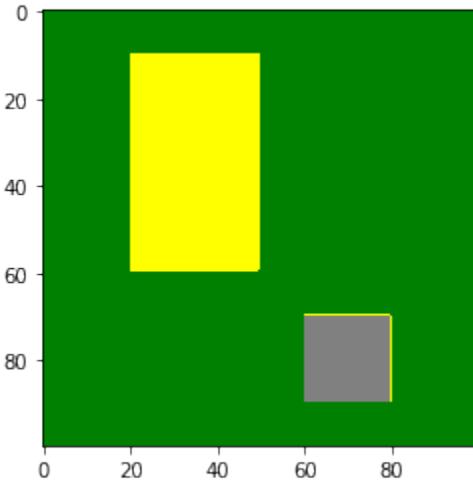
If we want to show what the area looks like according to the grass/crops/city designation, we might want to give each of the classifications a colour.

```
[7]: # We map the values to colours
index = {1: 'green', 2: 'yellow', 3: 'grey'}

# Create a discrete colour map
cmap = colors.ListedColormap(index.values())

# Plot
plt.imshow(area, cmap=cmap)
```

[7]: <matplotlib.image.AxesImage at 0x7fe7b5487f98>



In the case above, every pixel had a value of either 1, 2 or 3. What happens if our dataset is incomplete and there is no data in some places?

This is a common problem in real-life datasets. Real datasets can be incomplete and may be missing data at certain times or places. To deal with this, we use the special value known as `NaN`, which stands for **Not a Number**.

`Nan`s are designated by the numpy `np.nan` function.

```
[8]: arr = np.array([1, 2, 3, 4, 5, np.nan, 7, 8, 9], dtype=np.float32)

arr

[8]: array([ 1.,  2.,  3.,  4.,  5., nan,  7.,  8.,  9.], dtype=float32)
```

To compute statistics on arrays containing `NaN` values, numpy has special versions of common functions such as `mean`, standard deviation `std`, and `sum` that ignore the `NaN` values. For example, the next cell shows the difference between using the usual `mean` function and the `nanmean` function.

The `mean` function cannot handle `NaN` values so it will return `nan`. The `nanmean` function does not include `NaN` values in the calculation, and therefore returns a number value.

```
[9]: print(np.mean(arr))

print(np.nanmean(arr))

nan
4.875
```

Note that `NaN` is generally not used as a key in dictionary key-value entries because there are different ways of expressing `NaN` in Python and they are not always equivalent. However, it is still possible to visualise data with `NaN`s; there will

be gaps in the image where there is no data.

8.4.3 Exercises

4.1 The harvesting season has arrived and our cropping lands have changed colour to brown. Can you:

4.1.1 Modify the yellow area to contain the new value 4?

4.1.2 Add a new entry to the index dictionary mapping number 4 to the value brown.

4.1.3 Plot the area.

```
[ ]: # 4.1.1 Modify the yellow area to hold the value 4
```

```
[ ]: # 4.1.2 Add a new key-value pair to index that maps 4 to 'brown'
```

```
[ ]: # 4.1.3 Copy the cmap definition and re-run it to add the new colour
```

```
# Plot the area
```

Hint: If you want to plot the new area, you have to redefine `cmap` so the new value is assigned a colour in the colour map. Copy and paste the `cmap = ...` line from the original plot.

4.2 Set `area[20:40, 80:95] = np.nan`. Plot the area now.

```
[ ]: # Set the nan area
```

```
[ ]: # Plot the entire area
```

4.3 Find the median of the area array from 4.2 using `np.nanmedian`. Does this match your visual interpretation? How does this compare to using `np.median`?

```
[ ]: # Use np.nanmedian to find the median of the area
```

8.4.4 Conclusion

Two key Python capabilities have been introduced in this section. We can organise our data using the dictionary syntax, and understand incomplete datasets that may use NaN values to show blanks. The next lesson provides a guide to xarray, a Python package that builds on these concepts to make multi-dimensional data easier to load and use.

8.5 Python basics 5: Xarray

This tutorial introduces xarray (pronounced *ex-array*), a Python library for working with labeled multi-dimensional arrays. It is widely used to handle Earth observation data, which often involves multiple dimensions — for instance, longitude, latitude, time, and channels/bands. It can also display metadata such as the dataset Coordinate Reference System (CRS).

Accessing data held in an `xarray.DataArray` or `xarray.Dataset` is very similar to calling values from dictionaries.

Note: This tutorial is about using xarray. An introduction to remote sensing datasets and more information on Earth observation products used on the Digital Earth Africa platform can be found in [Session 2: Digital Earth Africa products](#).

Follow the instructions below to download the tutorial and open it in the Sandbox.

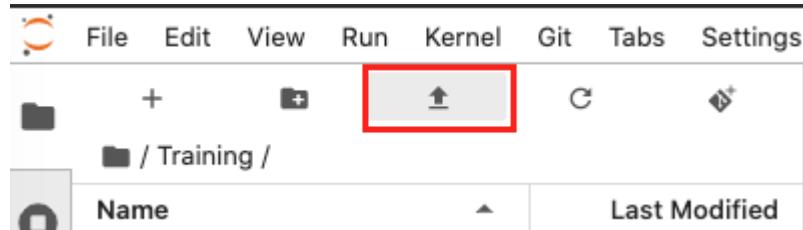
8.5.1 Download the tutorial notebook

[Download the Python basics 5 tutorial notebook](#)

[Download the exercise NetCDF file](#)

To view this notebook on the Sandbox, you will need to first download the notebook and the NetCDF file to your computer, then upload both of them to the Sandbox. Ensure you have followed the set-up prerequisites listed in [Python basics 1: Jupyter](#), and then follow these instructions:

1. Download the notebook by clicking the first link above. Download the image by clicking the second link above.
2. On the Sandbox, open the **Training** folder.
3. Click the **Upload Files** button as shown below.



4. Select the downloaded notebook using the file browser. Click **OK**.
5. Repeat to upload the NetCDF (.nc) file to the **Training** folder. It may take a while for the upload to complete.
6. Both files will appear in the **Training** folder. Double-click the tutorial notebook to open it and begin the tutorial.

You can now use the tutorial notebook as an interactive version of this webpage.

Note: The tutorial notebook should look like the text and code below. However, the tutorial notebook outputs are blank (i.e. no results showing after code cells). Follow the instructions in the notebook to run the cells in the tutorial notebook. Refer to this page to check your outputs look similar.

8.5.2 Introduction to xarray

Xarray makes datasets easier to interpret when performing data analysis. To demonstrate this, we can start by opening a NetCDF file (file suffix .nc) of the area of Guinea-Bissau we used in the previous [matplotlib lesson](#).

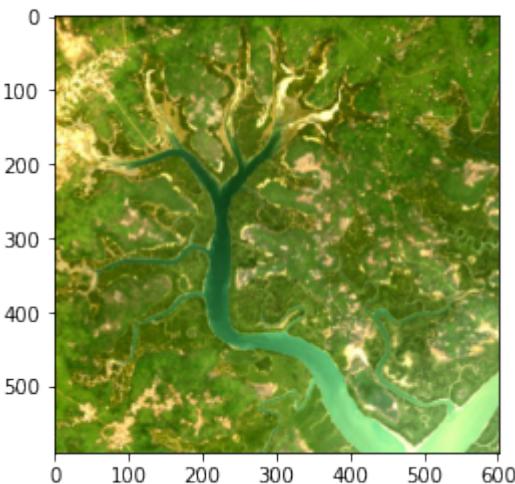
Recap: In that lesson, we plotted the .JPG version of the image using `imread` and `imshow`.

```
[1]: # Review of Python basics lesson 3: matplotlib
%matplotlib inline

import numpy as np
from matplotlib import pyplot as plt

im = np.copy(plt.imread('Guinea_Bissau.JPG'))
plt.imshow(im)

[1]: <matplotlib.image.AxesImage at 0x7f6fc79feba8>
```



Using xarray, we will now read in a .nc file of the area. .nc NetCDF files are a filetype used for storing scientific array-oriented data. Remembering that image data is an array, this makes both formats almost equivalent.

We could also load the .JPG as a dataset using xarray, but the NetCDF file contains more metadata so it will be easier to interpret.

As usual, we will first import the xarray package.

```
[2]: # Import the xarray package as xr
import xarray as xr

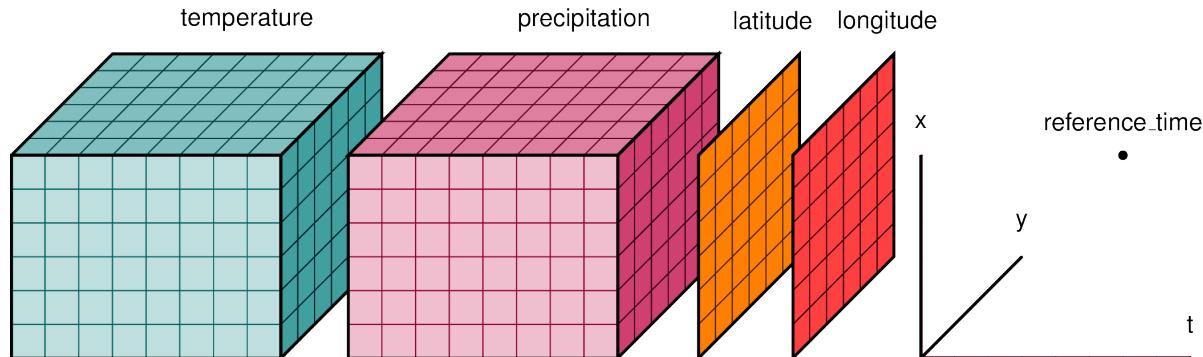
[3]: # Use xarray to open the data file
guinea_bissau = xr.open_dataset('guinea_bissau.nc')

[4]: # Inspect the xarray.DataSet by
# typing its name
guinea_bissau
```

```
[4]: <xarray.Dataset>
Dimensions:      (x: 500, y: 501)
Coordinates:
  time            datetime64[ns] ...
* y              float64 1.338e+06 1.338e+06 ... 1.323e+06 1.323e+06
* x              float64 3.88e+05 3.880e+05 3.881e+05 ... 4.03e+05 4.03e+05
  spatial_ref    int32 ...
Data variables:
  red             (y, x) float64 ...
  green           (y, x) float64 ...
  blue            (y, x) float64 ...
Attributes:
  crs:           EPSG:32628
  grid_mapping:  spatial_ref
```

Let's inspect the data. This image (our dataset) has a height `y` of 501 pixels and a width `x` of 500 pixels. It has 3 **variables**, which correspond to the red, green, and blue bands needed to plot a colour image. Each of these bands has a value at each of the pixels, so we have a total of 751500 values (the result of $501 \times 500 \times 3$). Each of the `x` and `y` values are associated with a longitude and latitude and their values are stored under the **Coordinates** section.

The `xarray.Dataset` uses numpy arrays in the backend. What we see above is functionally similar to a bunch of numpy arrays holding the image band measurements and the longitude/latitude coordinates, combined with a dictionary of key-value pairs that defines the **Attributes** such as CRS and resolution (`res`). That sounds more complicated, doesn't it? Accessing the data through `xarray` avoids the complications of managing uncorrelated numpy arrays by themselves.



A *visualisation of an xarray.Dataset*. The coordinates are each a single dimension; latitude, longitude and time. Each variable (in this case temperature and precipitation) holds one value at each of the three coordinate dimensions.

An `xarray.Dataset` can be seen as a dictionary structure for packing up the data, dimensions and attributes all linked together.

As in the NetCDF we loaded above, Digital Earth Africa follows the convention of storing spectral bands as separate variables, with each one as 3-dimensional cubes containing the temporal dimension.

To access a single variable we can use the same format as if it were a Python dictionary.

```
var = dataset_name['variable_name']
```

Alternatively, we can use the `.` notation.

```
var = dataset_name.variable_name
```

A single variable pulled from an `xarray.Dataset` is known as an `xarray.DataArray`. In the visualisation image above, the example `xarray.Dataset` is formed out of two `xarray.DataArrays` — one each for temperature and precipitation.

Note: Variable names are often strings (enclosed in quotation marks). This is true of most Digital Earth Africa datasets, where variables such as satellite bands have names such as '`red`', '`green`', '`nir`'. Do not forget the quotation marks in first call method above, or the data will not load.

The dataset we loaded only has 1 timestep. We will now pull just the '`red`' variable data from our `guinea_bissau` `xarray.Dataset`, and then plot it.

```
[5]: guinea_bissau['red']
```

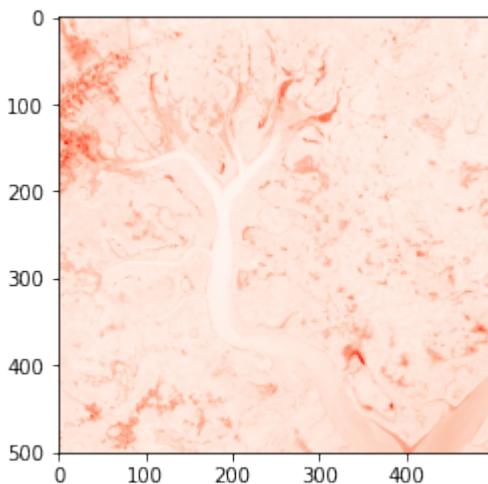
```
# This is equivalent to
# guinea_bissau.red
# Try it!
```

```
[5]: <xarray.DataArray 'red' (y: 501, x: 500)>
[250500 values with dtype=float64]
Coordinates:
  time      datetime64[ns] 2018-01-09T11:22:11.054208
  * y          float64 1.338e+06 1.338e+06 ... 1.323e+06 1.323e+06
  * x          float64 3.88e+05 3.880e+05 3.881e+05 ... 4.03e+05 4.03e+05
    spatial_ref int32 32628
Attributes:
  units:        reflectance
  nodata:       -9999
  crs:          EPSG:32628
  grid_mapping: spatial_ref
```

```
[6]: # Plot using imshow
```

```
plt.imshow(guinea_bissau['red'], cmap='Reds')
```

```
[6]: <matplotlib.image.AxesImage at 0x7f6fb0b77400>
```



Note: `cmap` stands for ‘colour map’. What happens if you remove this argument? `plt.imshow` will then plot all the `red` data values in the default colour scheme, which is a purple-to-yellow sequential colour map known as `viridis`. This is because matplotlib does not know the variable name `red` is associated

with the colour red — it is just a name. You can find out more about matplotlib colormaps through this [matplotlib tutorial](#).

We stated above that *xarray uses numpy arrays in the backend*. You can access these numpy arrays by adding `.values` after a `DataArray` name. In the example below, `guinea_bissau.red` is the `DataArray` name.

```
[7]: guinea_bissau.red.values
```

```
[7]: array([[1007., 1007., 1069., ..., 489., 476., 422.],
       [ 988., 991., 1048., ..., 456., 459., 444.],
       [ 976., 1023., 1062., ..., 453., 454., 448.],
       ...,
       [ 518., 518., 546., ..., 551., 577., 553.],
       [ 517., 527., 523., ..., 530., 572., 570.],
       [ 502., 490., 473., ..., 545., 576., 580.]])
```

It is possible to index `xarray.DataArrays` by position using the numpy `[:,:]` syntax we introduced in previous lessons, without converting to numpy arrays. However, we can also use the `xarray` syntaxes, which explicitly label coordinates instead of relying on knowing the order of dimensions.

Each method is demonstrated below.

- `[:,:]`: numpy syntax — requires knowing order of dimensions and positional indexes
- `isel(coordinate_name = coordinate_index)`: index selection — `xarray` syntax for selecting data based on its positional index (similar to numpy)
- `sel(coordinate_name = coordinate_value)`: value selection — `xarray` syntax for selecting data based on its value in that dimension

For example, we can call out the value of green in the top leftmost pixel of the image dataset.

```
[8]: # Using numpy syntax
```

```
guinea_bissau.green[0,0]
```

```
[8]: <xarray.DataArray 'green' ()>
array(730.)
Coordinates:
  time            datetime64[ns] 2018-01-09T11:22:11.054208
  y              float64 1.338e+06
  x              float64 3.88e+05
  spatial_ref    int32 32628
Attributes:
  units:           reflectance
  nodata:          -9999
  crs:             EPSG:32628
  grid_mapping:   spatial_ref
```

In this case our positional indexes for `x` and `y` are the same, but it is important to note `y` is the first dimension, and `x` is the second. This is not immediately obvious and can cause confusion. For this reason, it is recommended to use one of the `xarray` syntaxes shown below. They explicitly call on the dimension names.

```
[9]: # Using index selection, isel()
```

```
guinea_bissau.green.isel(y=0, x=0)
```

```
[9]: <xarray.DataArray 'green' ()>
array(730.)
Coordinates:
  time      datetime64[ns] 2018-01-09T11:22:11.054208
  y        float64 1.338e+06
  x        float64 3.88e+05
  spatial_ref int32 32628
Attributes:
  units:       reflectance
  nodata:     -9999
  crs:        EPSG:32628
  grid_mapping: spatial_ref
```

As before, the measurement for green in the top leftmost pixel is 730. In this example, the *values* of x and y are latitude and longitude values, but their *positional indexes* are 0.

We can call out the value by using the index, as shown below.

Note: The CRS we are using has units of metres. This means the x and y values are measured in metres.

```
[10]: # The value of x at the pixel located at index (0,0)

guinea_bissau.green.x[0].values
array(388020.)
```

```
[11]: # The value of y at the pixel located at index (0,0)

guinea_bissau.green.y[0].values
array(1338000.)
```

The interpretation of the cell above is:

Output the **value** of y in the **1st position** of the **green** variable from the dataset called **guineau_bissau**.

We see the first element (index of 0) in x has a value of 388020 metres, and the first value of y is 1338000 metres.

How about the second method, `sel()`? This method was not available in the numpy arrays we used in previous lessons, and is one of the strengths of xarray as you can use the value of the dimension without knowing its positional index.

If we are given (y, x) = (1338000, 388020), we can find the green measurement at that point using `sel()`.

```
[12]: guinea_bissau.green.sel(y=1338000, x=388020)
<xarray.DataArray 'green' ()>
array(730.)
Coordinates:
  time      datetime64[ns] 2018-01-09T11:22:11.054208
  y        float64 1.338e+06
  x        float64 3.88e+05
  spatial_ref int32 32628
Attributes:
  units:       reflectance
  nodata:     -9999
  crs:        EPSG:32628
  grid_mapping: spatial_ref
```

Finally, we can select ranges of DataArrays with the xarray syntaxes using `slice`. The three cells below all extract the same extent from the `guinea_bissau.green` DataArray.

The xarray syntaxes for ranges using `slice` are:

```
dataarray_name.isel(dimension_name = slice(index_start, index_end))
```

```
dataarray_name.sel(dimension_name = slice(value_start, value_end))
```

As before, it is easier to call upon the dimensions by name rather than using the implicit numpy square bracket syntax, although it still works.

```
[13]: # Numpy syntax - dimensions are not named
# Valid but not recommended

guinea_bissau.green[:250, :250]
```

[13]: <xarray.DataArray 'green' (y: 250, x: 250)>
array([[730., 737., 811., ..., 493., 514., 539.],
 [729., 717., 773., ..., 512., 532., 534.],
 [714., 738., 764., ..., 528., 541., 521.],
 ...,
 [662., 665., 676., ..., 506., 510., 480.],
 [667., 669., 664., ..., 553., 502., 501.],
 [662., 661., 676., ..., 600., 575., 560.]])

Coordinates:

```
    time      datetime64[ns] 2018-01-09T11:22:11.054208
  * y          float64 1.338e+06 1.338e+06 ... 1.331e+06 1.331e+06
  * x          float64 3.88e+05 3.880e+05 ... 3.955e+05 3.955e+05
    spatial_ref int32 32628
```

Attributes:

```
    units:        reflectance
    nodata:      -9999
    crs:         EPSG:32628
    grid_mapping: spatial_ref
```

```
[14]: # Index selection slice

guinea_bissau.green.isel(x=slice(0,250), y=slice(0,250))
```

[14]: <xarray.DataArray 'green' (y: 250, x: 250)>
array([[730., 737., 811., ..., 493., 514., 539.],
 [729., 717., 773., ..., 512., 532., 534.],
 [714., 738., 764., ..., 528., 541., 521.],
 ...,
 [662., 665., 676., ..., 506., 510., 480.],
 [667., 669., 664., ..., 553., 502., 501.],
 [662., 661., 676., ..., 600., 575., 560.]])

Coordinates:

```
    time      datetime64[ns] 2018-01-09T11:22:11.054208
  * y          float64 1.338e+06 1.338e+06 ... 1.331e+06 1.331e+06
  * x          float64 3.88e+05 3.880e+05 ... 3.955e+05 3.955e+05
    spatial_ref int32 32628
```

Attributes:

```
    units:        reflectance
```

(continues on next page)

(continued from previous page)

```

nodata:      -9999
crs:         EPSG:32628
grid_mapping: spatial_ref

```

```
[15]: # Value selection slice

guinea_bissau.green.sel(x=slice(388020,395500), y=slice(1338000,1330530))

[15]: <xarray.DataArray 'green' (y: 250, x: 250)>
array([[730., 737., 811., ..., 493., 514., 539.],
       [729., 717., 773., ..., 512., 532., 534.],
       [714., 738., 764., ..., 528., 541., 521.],
       ...,
       [662., 665., 676., ..., 506., 510., 480.],
       [667., 669., 664., ..., 553., 502., 501.],
       [662., 661., 676., ..., 600., 575., 560.]])
Coordinates:
  time          datetime64[ns] 2018-01-09T11:22:11.054208
  * y            (y) float64 1.338e+06 1.338e+06 ... 1.331e+06 1.331e+06
  * x            (x) float64 3.88e+05 3.880e+05 ... 3.955e+05 3.955e+05
    spatial_ref  int32 32628
Attributes:
  units:        reflectance
  nodata:      -9999
  crs:         EPSG:32628
  grid_mapping: spatial_ref

```

We can plot the selected area using matplotlib `plt.imshow`. We can also use the `xarray .plot()` function. Plotting is a good way to check the selection is showing the top left section of the image as expected.

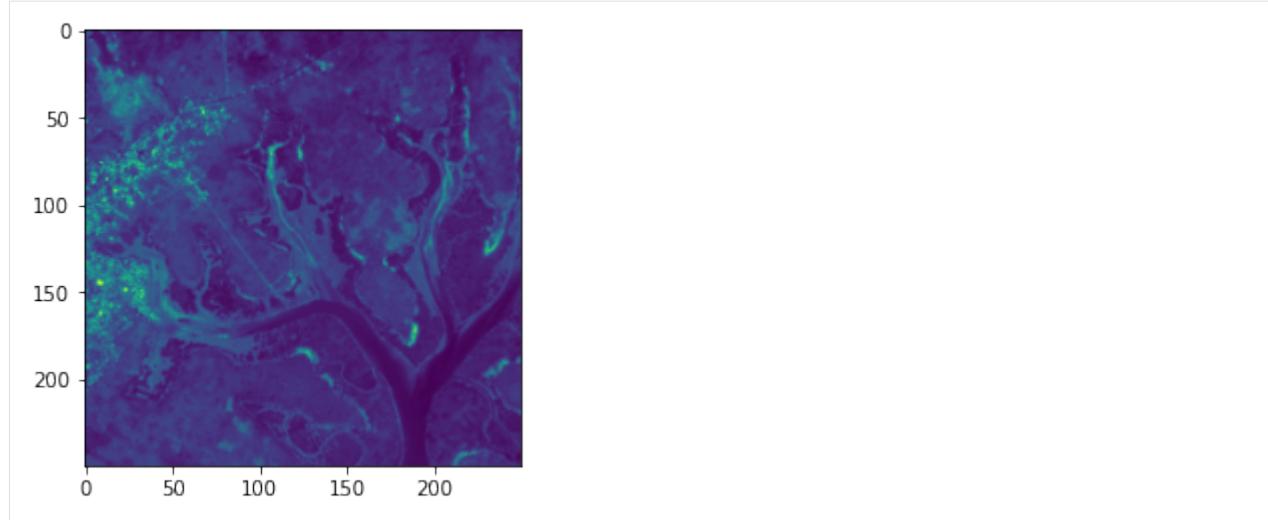
The syntax of `plt.imshow()` is:

```
plt.imshow(data)
```

Therefore we copy the code into the brackets of `plt.imshow()`.

```
[16]: plt.imshow(guinea_bissau.green.sel(x=slice(388020,395500), y=slice(1338000,1330530)))

[16]: <matplotlib.image.AxesImage at 0x7f6fb0af2470>
```



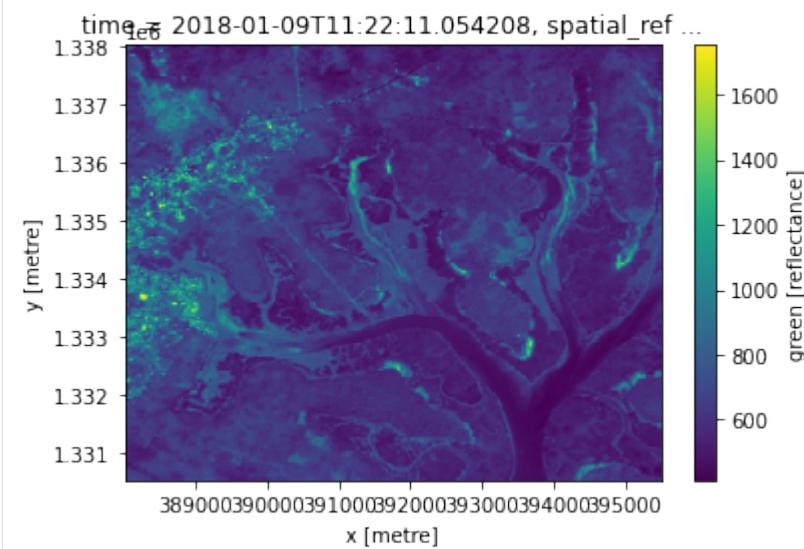
Note: We did not specify a colourmap `cmap` so you can see it takes on the default colour scheme. Plotting true-colour (RGB) images is first introduced in [Session 2: Loading data in the Sandbox](#).

The syntax of the `xarray` plot function is:

```
data.plot()
```

Therefore we copy the code before we type `.plot()`.

```
[17]: guinea_bissau.green.sel(x=slice(388020,395500), y=slice(1338000,1330530)).plot()  
[17]: <matplotlib.collections.QuadMesh at 0x7f6fb0a0cba8>
```



The advantage of using `xarray`'s `.plot()` is that it automatically shows the `x` and `y` axes using coordinate values, not their positional index.

8.5.3 Exercises

5.1 Can you access to the crs value in the attributes of the guinea_bissau xarray.Dataset?

Hint: You can call upon attributes in the same way you would select a variable or coordinate.

```
[ ]: # Replace the ? with the attribute name  
guinea_bissau.?
```

5.2 Select the region of the blue variable delimited by these coordinates:

- latitude of range [1335000, 1329030]
- longitude of range [389520, 395490]

Hint: Do we want to use `sel()` or `isel()`? Which coordinate is `x` and which is `y`?

5.3 Plot the selected region using `imshow`, then plot the region using `.plot()`.

```
[ ]: # Plot using plt.imshow
```

```
[ ]: # Plot using .plot()
```

Can you change the colour map to 'Blues'?

8.5.4 Conclusion

Xarray is an important tool when dealing with Earth observation data in Python. Its flexible and intuitive interface allows multiple methods of selecting data and performing analysis.

8.6 Exercise solutions

This section contains possible solutions to the exercises posed in the Python basics module. There is more than one correct solution for most of the exercises so these answers are for reference only.

[Download the exercise solutions notebook](#)

To view the solutions as an interactive notebook, download the file to your machine and upload it to the Sandbox as in the Python basics lessons.

8.6.1 Python basics 1

1.1 Fill the asterisk line with your name and run the cell.

```
[1]: # Fill the ***** space with your name and run the cell.  
  
message = "My name is Python"  
  
message  
[1]: 'My name is Python'
```

1.2 You can add new cells to insert new code at any point in a notebook. Click on the + icon in the top menu to add a new cell below the current one. Add a new cell below the next cell, and use it to print the value of variable a.

```
[2]: a = 365*24  
  
# Add a new cell just below this one. Use it to print the value of variable `a`  
  
[3]: a  
[3]: 8760
```

Question: Now what happens if you scroll back up the notebook and execute a different cell containing `print(a)`?

Answer: It should now print 8760 as ‘global state’ means the value of a has been changed.

8.6.2 Python basics 2

```
[4]: import numpy as np
```

2.1 Use the numpy add function to add the values 34 and 29 in the cell below.

```
[5]: # Use numpy add to add 34 and 29  
  
np.add(34, 29)  
[5]: 63
```

2.2 Declare a new array with contents [5,4,3,2,1] and slice it to select the last 3 items.

```
[6]: # Substitute the ? symbols by the correct expressions and values
# Declare the array
arr = np.array([5, 4, 3, 2, 1])
# Slice array for the last 3 items only
arr[-3:]
[6]: array([3, 2, 1])
```

2.3: Select all the elements in the array below excluding the last one, [15].

```
[7]: arr = np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15])
# Substitute the ? symbols by the correct expressions and values
arr[:-1]
[7]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

2.4 Use arr as defined in 2.3. Exclude the last element from the list, but now only select every 3rd element. Remember the third index indicates stride, if used.

Hint: The result should be [0,3,6,9,12].

```
[8]: # Substitute the ? symbols by the correct expressions and values
arr[:-1:3]
[8]: array([ 0,  3,  6,  9, 12])
```

2.5 You'll need to combine array comparisons and logical operators to solve this one. Find out the values in the following array that are greater than 3 AND less than 7. The output should be a boolean array.

Hint: If you are stuck, reread the section on boolean arrays.

```
[9]: arr = np.array([1, 3, 5, 1, 6, 3, 1, 5, 7, 1])
# Use array comparisons (<, >, etc.) and logical operators (*, +) to find where
# the values are greater than 3 and less than 7.

boolean_array = (arr > 3)*(arr < 7)
[10]: boolean_array
[10]: array([False, False,  True, False,  True, False, False,  True, False,
```

2.6 Use your boolean array from 2.5 to mask the False values from arr.

Hint: The result should be [5, 6, 5].

```
[11]: # Use your resulting boolean_array array from 2.5
      # to mask arr as defined in 2.5

      arr[boolean_array]

[11]: array([5, 6, 5])
```

8.6.3 Python basics 3

```
[12]: %matplotlib inline

import numpy as np
from matplotlib import pyplot as plt

im = np.copy(plt.imread('Guinea_Bissau.JPG'))
```

3.1 Let's use the indexing functionality of numpy to select a portion of this image. Select the top-right corner of this image with shape (200,200).

Hint: Remember there are three dimensions in this image. Colons separate spans, and commas separate dimensions.

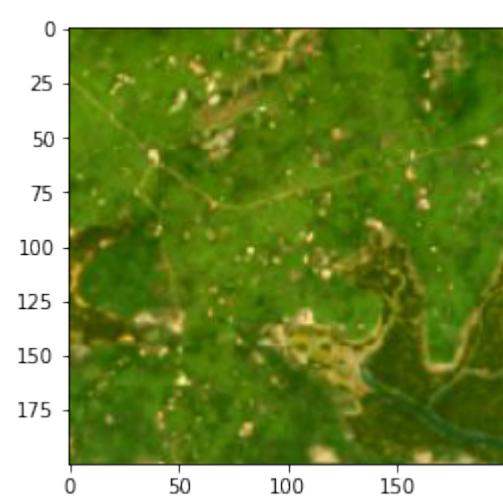
```
[13]: # Both options below are correct

topright = im[:200, -200:, ]
topright = im[:200, 400:600, ]

# Plot your result using imshow

plt.imshow(topright)

[13]: <matplotlib.image.AxesImage at 0x7f08e28f7940>
```



3.2 Let's have a look at one of the pixels in this image. We choose the top-left corner with position $(0,0)$ and show the values of its RGB channels.

```
[14]: # Run this cell to see the colour channel values
```

```
im[0,0]
```

```
[14]: array([249, 196, 104], dtype=uint8)
```

The first value corresponds to the red component, the second to the green and the third to the blue. `uint8` can contain values in the range [0-255] so the pixel has a lot of red, some green, and not much blue. This pixel is a orange-yellow sandy colour.

Now let's modify the image.

What happens if we set all the values representing the blue channel to the maximum value?

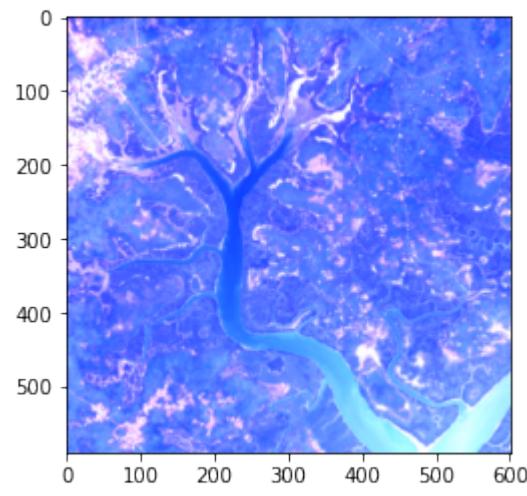
```
[15]: # Run this cell to set all blue channel values to 255
# We first make a copy to avoid modifying the original image
```

```
im2 = np.copy(im)
```

```
im2[:, :, 2] = 255
```

```
plt.imshow(im2)
```

```
[15]: <matplotlib.image.AxesImage at 0x7f08e23d0ac8>
```



The index notation `[:,:,2]` is selecting pixels at all heights and all widths, but only the 3rd colour channel.

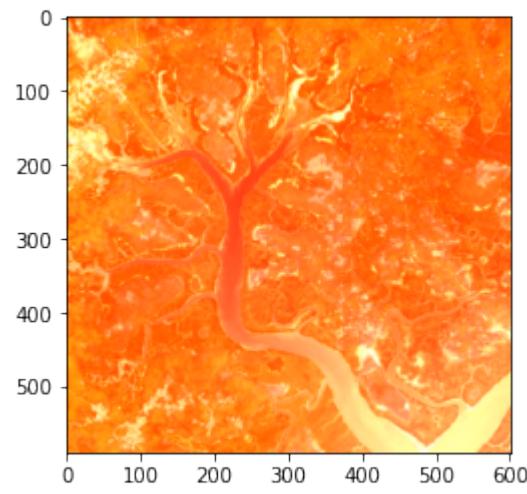
Can you modify the above code cell to set all red values to the maximum value of 255?

```
[16]: im2 = np.copy(im)

im2[:, :, 0] = 255

plt.imshow(im2)
```

[16]: <matplotlib.image.AxesImage at 0x7f08e0b34ef0>



8.6.4 Python basics 4

```
[17]: %matplotlib inline

import numpy as np
from matplotlib import pyplot as plt
from matplotlib import colors

# grass = 1
area = np.ones((100,100))
# crops = 2
area[10:60,20:50] = 2
# city = 3
area[70:90,60:80] = 3
index = {1: 'green', 2: 'yellow', 3: 'grey'}
cmap = colors.ListedColormap(index.values())
```

4.1 The harvesting season has arrived and our cropping lands have changed colour to brown. Can you:

4.1.1 Modify the yellow area to contain the new value 4?

4.1.2 Add a new entry to the index dictionary mapping number 4 to the value brown.

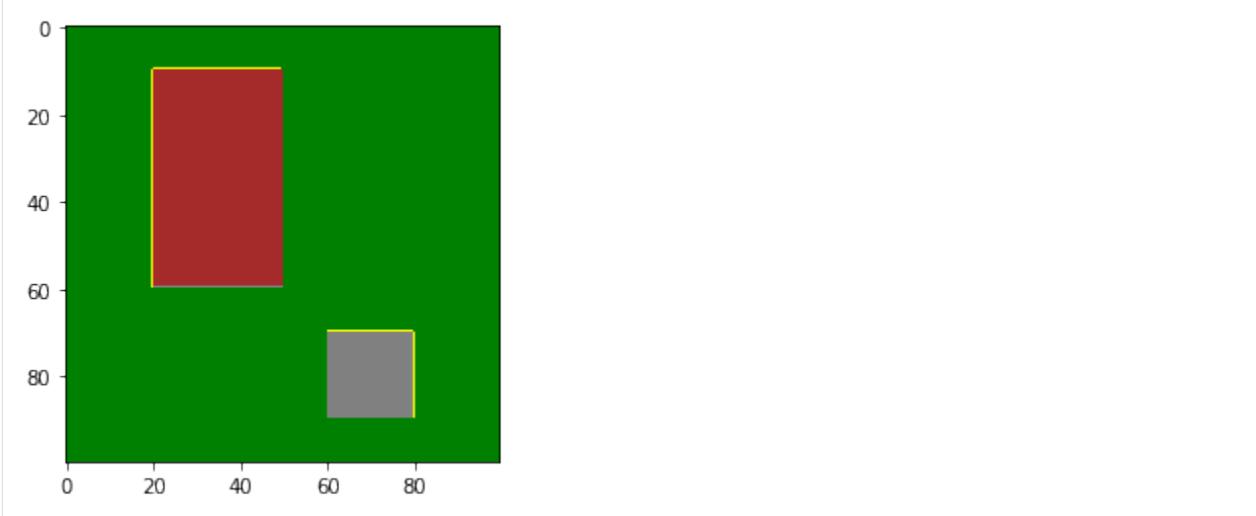
4.1.3 Plot the area.

```
[18]: # 4.1.1 Modify the yellow area to hold the value 4
area[10:60,20:50] = 4
```

```
[19]: # 4.1.2 Add a new key-value pair to index that maps 4 to 'brown'
index[4] = 'brown'
```

```
[20]: # 4.1.3 Copy the cmap definition and re-run it to add the new colour
cmap = colors.ListedColormap(index.values())
# Plot the area
plt.imshow(area, cmap=cmap)
```

```
[20]: <matplotlib.image.AxesImage at 0x7f08e0b26198>
```



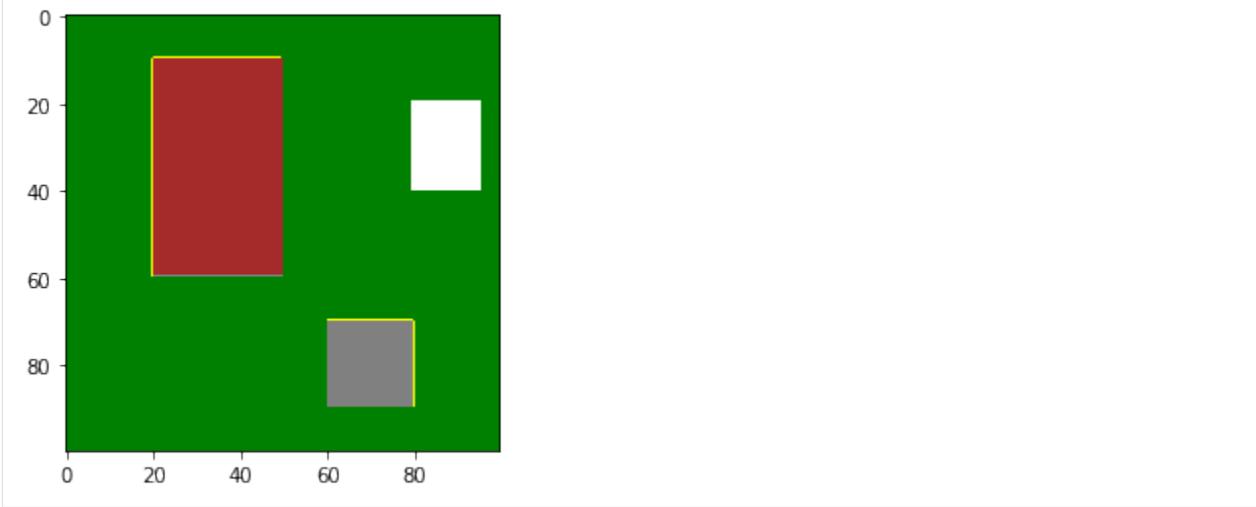
Hint: If you want to plot the new area, you have to redefine `cmap` so the new value is assigned a colour in the colour map. Copy and paste the `cmap = ...` line from the original plot.

4.2 Set `area[20:40, 80:95] = np.nan`. Plot the area now.

```
[21]: # Set the nan area  
area[20:40, 80:95] = np.nan
```

```
[22]: # Plot the entire area  
plt.imshow(area, cmap=cmap)
```

```
[22]: <matplotlib.image.AxesImage at 0x7f08e0a78e48>
```



4.3 Find the median of the area array from 4.2 using `np.nanmedian`. Does this match your visual interpretation? How does this compare to using `np.median`?

```
[23]: # Use np.nanmedian to find the median of the area
np.nanmedian(area)
```

```
[23]: 1.0
```

```
[24]: np.median(area)
```

```
[24]: nan
```

`np.median` returns a value of `nan` because it cannot interpret no-data pixels. `np.nanmedian` excludes NaN values, so it returns a value of 1 which indicates grass. This matches the plot of `area`.

8.6.5 Python basics 5

```
[25]: %matplotlib inline

import numpy as np
from matplotlib import pyplot as plt
import xarray as xr
guinea_bissau = xr.open_dataset('guinea_bissau.nc')
```

5.1 Can you access to the `crs` value in the attributes of the `guinea_bissau xarray.Dataset`?

Hint: You can call upon attributes in the same way you would select a variable or coordinate.

```
[26]: # Replace the ? with the attribute name
```

```
guinea_bissau.crs
```

```
[26]: 'EPSG:32628'
```

5.2 Select the region of the blue variable delimited by these coordinates:

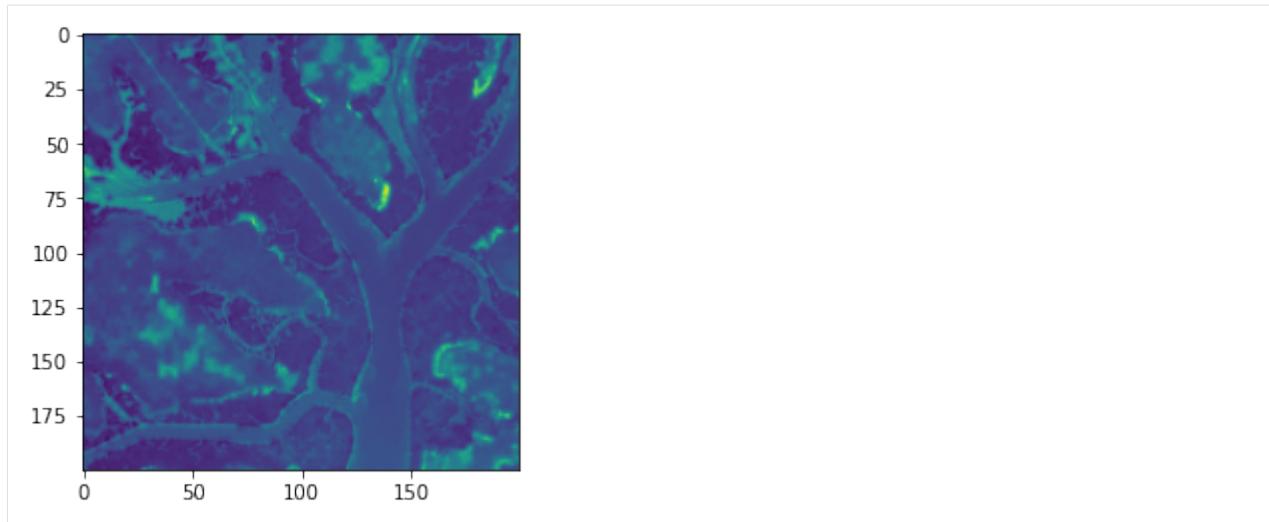
- latitude of range [1335000, 1329030]
- longitude of range [389520, 395490]

Hint: Do we want to use `sel()` or `isel()`? Which coordinate is x and which is y?

5.3 Plot the selected region using `imshow`, then plot the region using `.plot()`.

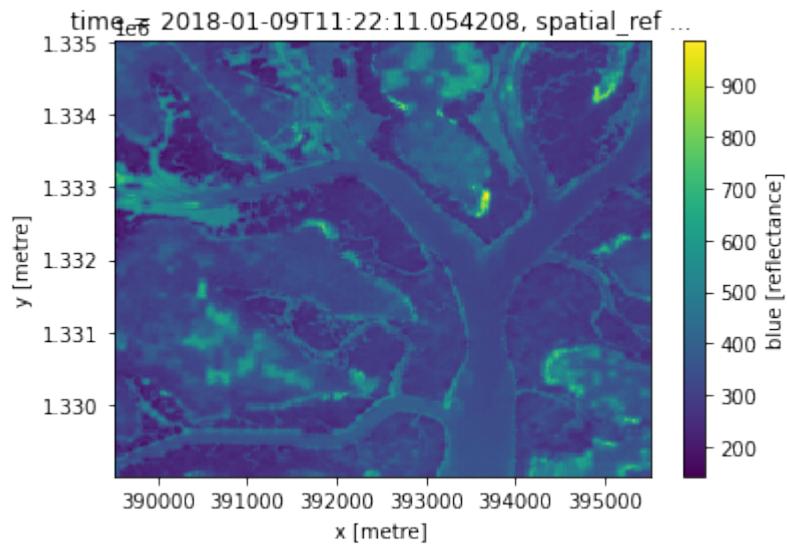
```
[27]: # Plot using plt.imshow
plt.imshow(guinea_bissau.blue.sel(x=slice(389520, 395490), y=slice(1335000, 1329030)))
```

```
[27]: <matplotlib.image.AxesImage at 0x7f08cb8f0da0>
```



```
[28]: # Plot using .plot()  
guinea_bissau.blue.sel(x=slice(389520, 395490), y=slice(1335000, 1329030)).plot()
```

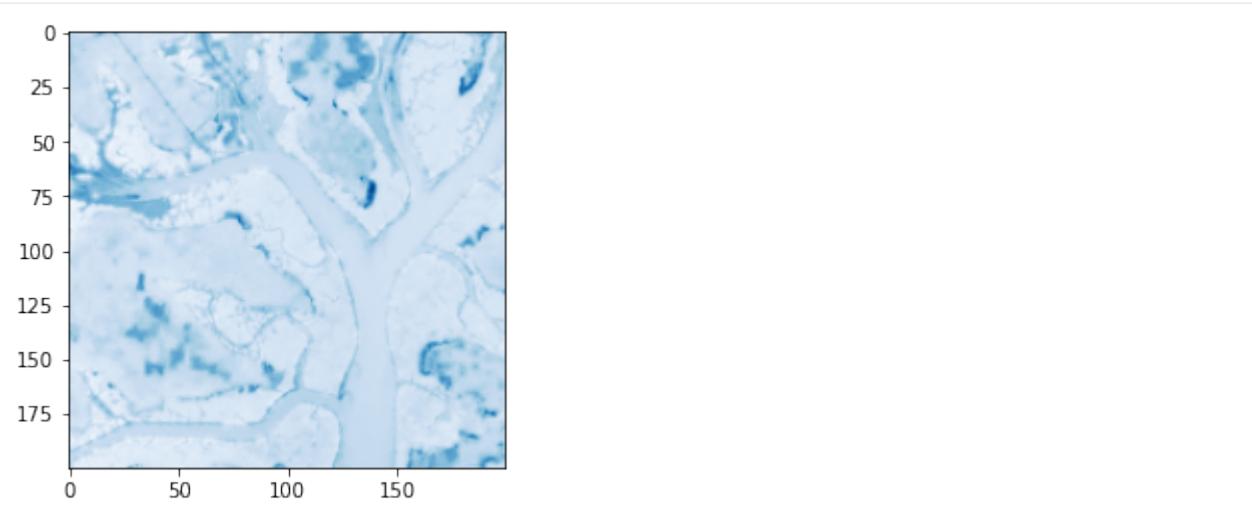
```
[28]: <matplotlib.collections.QuadMesh at 0x7f08cb824898>
```



Can you change the colour map to 'Blues'?

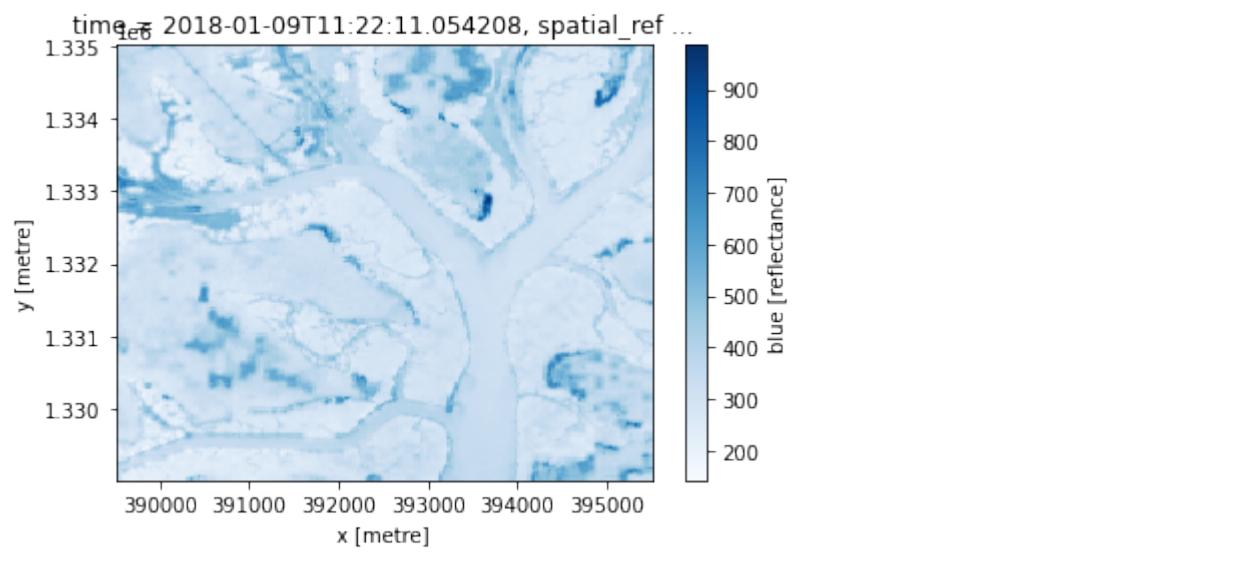
```
[29]: plt.imshow(guinea_bissau.blue.sel(x=slice(389520, 395490), y=slice(1335000, 1329030)),  
             cmap='Blues')
```

```
[29]: <matplotlib.image.AxesImage at 0x7f08cb763eb8>
```



```
[30]: guinea_bissau.blue.sel(x=slice(389520, 395490), y=slice(1335000, 1329030)).plot(cmap='Blues')
```

```
[30]: <matplotlib.collections.QuadMesh at 0x7f08cb6d1710>
```



LATEST UPDATES

- **01/06/2021:** Digital Earth Africa Landsat datasets have been upgraded to USGS Collection 2. Datacube names have been updated to `ls5_sr`, `ls7_sr` and `ls8_sr`. Deprecated naming conventions such as `ls8_usgs_sr_scene` will no longer work. Additionally, the `deafrica_tools` package has replaced the deprecated `sys.path.append('../Scripts')` file import.
- **21/01/2021:** *Virtual live sessions* run by our course convenors have resumed for 2021. We encourage all current and past course participants to join in.
- **15/12/2020:** If you have completed any or all sections of the Digital Earth Africa training course, complete our [participant survey](#).
- **14/12/2020:** An extra session on *Python basics for geospatial analysis* has been added. The session is optional, standalone, and provides a more detailed introduction to the Python programming language.
- **09/12/2020:** All six assessment quizzes can be accessed from the [Quiz index](#). page. Score full marks on each quiz to receive a Certificate of Completion. Questions about the Certificate of Completion? See the [FAQ](#).

9.1 Contact us

9.1.1 Course conveners

Edward Boamah — Technical Manager

Kenneth Mubea, PhD — User Engagement Manager

The course conveners can be contacted at training@digitalearthafrica.org.

9.1.2 Getting help

The **Digital Earth Africa FAQ** can be found at [Frequently asked questions](#).

An overview of the **Digital Earth Africa help documentation** can be found at [Help documentation](#).

Sandbox Notebooks: The [Sandbox Notebooks](#) is a repository of help documentation and notebooks that cover common data analysis functions and questions. The files can be accessed without signing up.

Slack: Slack is a platform for team communication and information sharing. Sign up to the [Open Data Cube Slack](#). You can then join the `#de-africa-training` channel by clicking the `+` next to the **Channels** list. It is a friendly environment where anyone can ask questions about course content.

9.1.3 Provide feedback

Whether you have completed the entire Digital Earth Africa training course or are just getting started, we want to hear from you! Email our course convenors at training@digitalearthafrica.org or fill in this [short survey](#).

9.2 Frequently asked questions

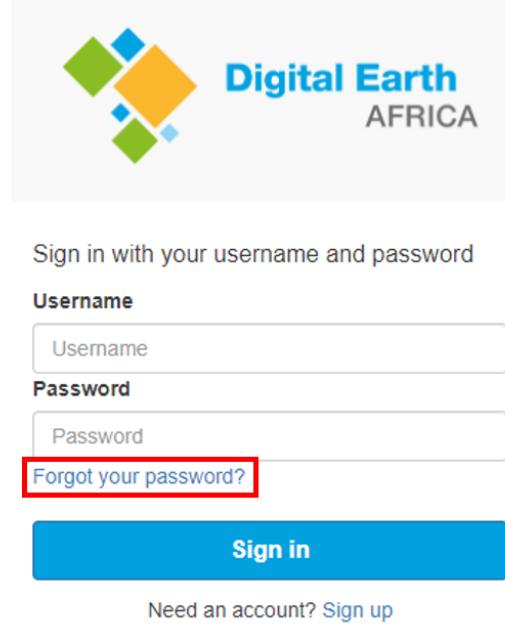
- *I didn't receive a Sandbox verification code email.*
- *I forgot my DE Africa Sandbox password. How do I reset it?*
- *I accidentally deleted a pre-loaded notebook in the Sandbox. How do I get it back?*
- *I have fallen behind on course content. Can I still complete the course?*
- *How do I download an offline copy of course material?*
- *My kernel has crashed. What do I do?*
- *I am having Sandbox registration problems. What do I do?*
- *How do I join the Slack channel?*
- *Am I eligible for a Certificate of Completion?*
- *How do I connect to GIS web services?*
- *How do I import shapefiles or geojson files?*
- *How do I provide feedback?*

9.2.1 I didn't receive a Sandbox verification code email.

Please check the spam and junk folders in your email inbox.

9.2.2 I forgot my DE Africa Sandbox password. How do I reset it?

Visit the Sandbox homepage at <https://sandbox.digitalearth.africa/hub/login> and click **Login or Sign up**. Click **Forgot your password?** and follow the instructions to reset your password.



9.2.3 I accidentally deleted a pre-loaded notebook in the Sandbox. How do I get it back?

The DE Africa Sandbox session automatically loads several folders of default notebooks. If you have accidentally deleted any of the folders or notebooks, they can be restored by restarting the server.

1. Save any open working files.
2. Select **File -> Hub Control Panel**. This will open a new tab.
3. On the new tab, select **Stop My Server**. The server will take a few minutes to stop.
4. When the server has stopped, the buttons will refresh. Select **Start My Server**.
5. Select **Launch Server** and **Start**.

This should reload all default folders and notebooks for the DE Africa Sandbox. Note that this will **not** overwrite files that have been added, renamed, or modified. Additionally, this only applies to default pre-loaded files and will not restore personal notebooks that have been deleted.

9.2.4 I have fallen behind on course content. Can I still complete the course?

Yes. It is recommended to spend 2 hours a week on new content. This will allow you to build on the previous week's content and finish the training course in around 6 weeks. However, this training course is self-paced, so there is no cut-off date for access. You are encouraged and welcome to spend more time on the course as necessary.

Certification will be evaluated by submissions to the course conveners, so it is possible to obtain certification even if the course takes you longer than 6 weeks to complete.

If you need help with the content or have course feedback, see the *Contact us* page for support channels and convener contact information.

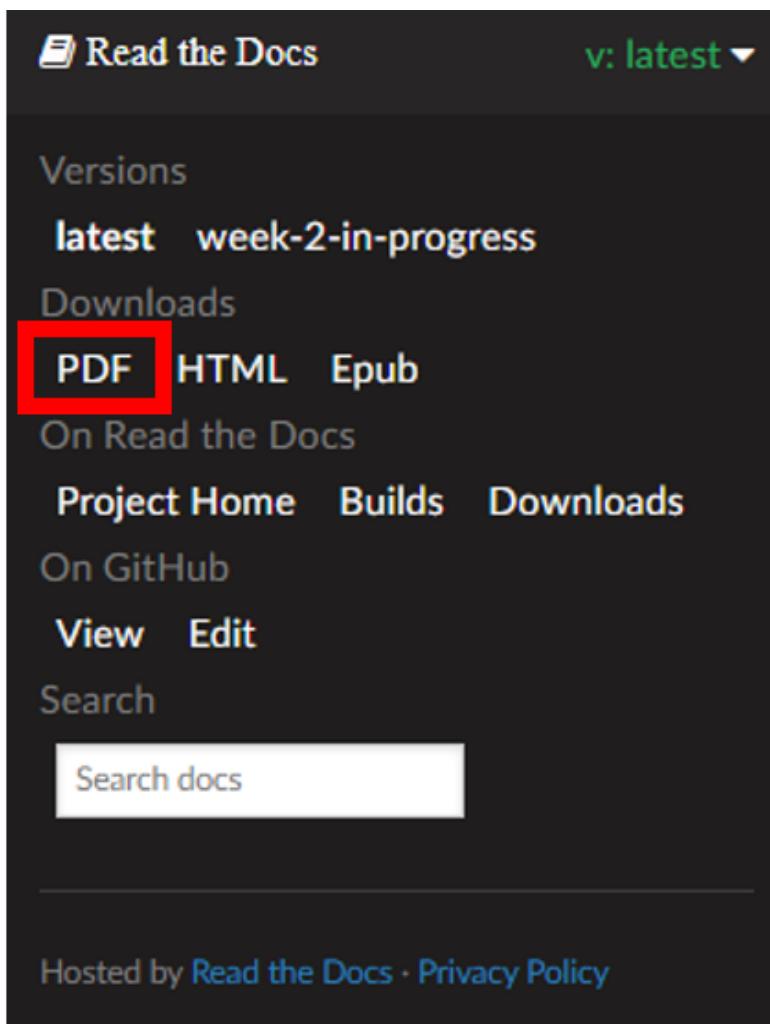
9.2.5 How do I download an offline copy of course material?

The Digital Earth Africa training course material can be downloaded as a PDF for offline reading or printing. The PDF contains all text and images, but does not embed videos. As content may be updated, it is recommended you download a new copy every 1 – 2 weeks.

1. Go to the Digital Earth Africa training homepage at <https://training.digitalearthafrica.org/en/latest/index.html>.
2. Scroll to the bottom of the page and click **v: latest** to expand the menu.



3. Select **PDF**. This will generate a PDF of the training website. Click **Save** or **Download** to save a copy to your computer.



Note Sandbox access requires an internet connection. You will not be able to complete Sandbox exercises offline.

PDFs can be viewed using [Adobe Acrobat Reader DC](#), which can be downloaded for free.

9.2.6 My kernel has crashed. What do I do?

The Digital Earth Africa Sandbox has a finite amount of processing power allocated to each user. This means if you attempt to load or calculate a lot of data, the kernel may crash. You will see a message like this:

Kernel Restarting

The kernel for `example_file.ipynb` appears to have died. It will restart automatically.

OK

You will not lose any files, but will have to rerun your notebook to continue working. However, if the notebook is still demanding too many resources, it will crash again.

You can prevent kernel crashes by:

- Decreasing the area selected for data processing

- Shortening the time extent
- Shutting down kernels of notebooks you have finished using: open the notebook and select **Kernel -> Shut Down Kernel**

These actions reduce the memory load of your computation on the Sandbox.

9.2.7 I am having Sandbox registration problems. What do I do?

If you have specific Sandbox issues, please send an email to training@digitalearthafrica.org. These may include:

- Cannot register with the Sandbox
- Entered the wrong credentials during registration
- Need to change your email address

If you have forgotten your Sandbox password, please see [I forgot my DE Africa Sandbox password. How do I reset it?](#)

If you have experienced a Sandbox kernel crash or kernel restart error message, please see [My kernel has crashed. What do I do?](#)

9.2.8 How do I join the Slack channel?

Digital Earth Africa uses Slack as a communications platform for asking questions and general discussion. It is free to join.

1. Go to <http://slack.opendatacube.org/>. If you do not have a Slack account, enter your email address to receive an invitation to sign up and log in. If you are already a Slack member, follow the prompts to log in.

OpenDataCube Slack

Submit your email address below and we'll send you an invitation!

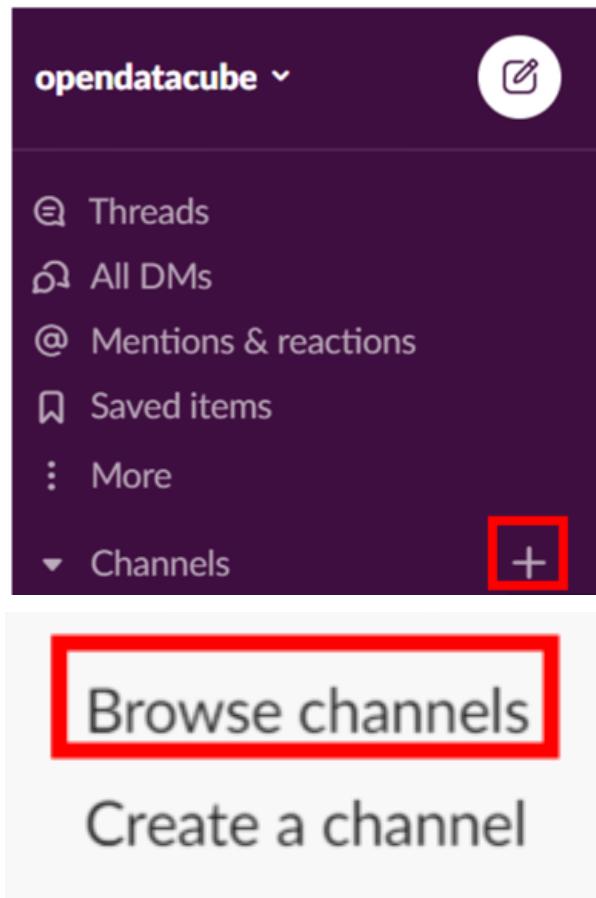
Email address

Join Slack

If you're already a member, log in now

Go to Slack

2. Open Slack. Select the + button next to **Channels** and click **Browse channels**.



3. Hover over the channel called #de-africa-training and click **Join**.

Channel browser

Create Channel

Search by channel name or description

66 channels

#de-africa-training
8 members - A forum for people who are training on the Digital Earth Africa platform to discuss and interact between each others and the instructors

Sort: A to Z Filter

Join

9.2.9 Am I eligible for a Certificate of Completion?

Training course participants who have passed all six assessment quizzes will receive a personalised Certificate of Completion. Quizzes can be retaken as many times as you like.

Submissions are identified by email address, so **please use the same email address for all quizzes**. Contact the course convenors at training@digitalearthafrica.org if you have any of the following issues:

- Used a different email address for different quizzes
- Unsure if you have successfully completed all the quizzes
- Did not receive a Certificate of Completion via email

9.2.10 How do I connect to GIS web services?

See [this tutorial](#) on using the Digital Earth Africa Web Map Services or Web Coverage Services.

9.2.11 How do I import shapefiles or geojson files?

The Sandbox is compatible with external datasets such as shapefiles or geojson files. See [this tutorial](#).

9.2.12 How do I provide feedback?

Complete the Digital Earth Africa [participant survey](#). All responses are anonymous and you do not need to have finished the entire course to take the survey. You are also welcome to [contact our course convenors](#).

9.3 Download course content

The DE Africa Training Course can be downloaded as a PDF. This can be used as an offline resource, although we encourage following along with exercises online on the [DE Africa Sandbox](#) when possible.

[Click here to download the PDF of course content \(29 MB\)](#)

Note: The PDF only contains text and images. Videos and tutorials will not be accessible or interactive through the PDF. Please check this page regularly to download the most up-to-date version of the training material.

You can also generate the latest PDF version of the training material by following the [FAQ on downloading course content](#).

9.4 Events

The DE Africa training course convenors host regularly scheduled **virtual live sessions**. They are intended to be a friendly, open environment where anyone can ask questions and meet people from the DE Africa community.

The live sessions are conducted on Zoom, an online meeting platform. They are open to anyone who is currently completing the 6-week training course, or has recently completed the 6-week training course.

To receive details on how to join the live sessions, please send an email to training@digitalearthafrica.org.

9.5 Sandbox help documentation

The Digital Earth Africa Sandbox, by default, will load several folders containing helpful notebooks and source code. They are essential to the Sandbox's operation, so do not delete or move any of the files.

The Beginner's Guide notebooks cover similar material to the explanations and exercises in this workshop series. However, they also provides more in-depth detail on how to perform data extraction and analysis. The Beginner's Guide notebooks, along with the 'frequently used code' notebooks, are a great reference tool.

This section provides a summary of these folders and their contents. It also contains some recommended but **not compulsory** activities that will help increase your familiarity with Digital Earth Africa and the Sandbox.

For tips on how to access files and folders in the Sandbox, check out the section on [navigating the Sandbox](#).

For a review on how to execute code in a notebook, see this section on [running a notebook](#).

9.5.1 Beginner's Guide - start here

/ Beginners_guide /		
Name		Last Modified
01_Jupyter_notebooks.ipynb		a minute ago
02_Products_and_measurements.ipynb		a minute ago
03>Loading_data.ipynb		a minute ago
04_Plotting.ipynb		a minute ago
05_Basic_analysis.ipynb		a minute ago
06_Intro_to_numpy.ipynb		a minute ago
07_Intro_to_xarray.ipynb		a minute ago
08_Parallel_processing_with_dask.ipynb		a minute ago
README.md		a minute ago

The **Beginners_guide** folder contains step-by-step tutorials to help you become more familiar with the Digital Earth Africa Sandbox. The tutorials contain background information and context, as well as code and data examples and explanations. The guide covers a range of basic Sandbox functions, including how to load data and plot it.

If you are still unclear on the scope and focus of Digital Earth Africa and the Sandbox, this is also a great place to read more about it and see some hands-on demonstrations of Earth observation data at work.

The notebooks are numbered in progress order; start at `01_Jupyter_notebooks.ipynb`. There are no prerequisites to be able to complete this notebook.

Suggested activity: Read and follow examples in the Beginner's Guide notebooks, starting with `01_Jupyter_notebooks.ipynb`. This can be done interactively (recommended), as detailed in [running a notebook](#). Open the file and from the horizontal menu bar select **Kernel -> Restart Kernel and Clear All Outputs**, then follow instructions in the notebook.

Take your time — it is not necessary to review all of the notebooks in one sitting. If you are new to Python coding, there will be a fair amount of content. Some of it will be covered in subsequent sessions of this workshop series.

9.5.2 Frequently used code

Name	Last Modified
Analyse_multiple_polygons.ipynb	3 hours ago
Animated_timeseries.ipynb	3 hours ago
Applying_WOfS_bitmasking.ipynb	3 hours ago
Calculating_band_indices.ipynb	3 hours ago
Contour_extraction.ipynb	3 hours ago
Exporting_GeoTIFFs.ipynb	3 hours ago
Generating_composites.ipynb	3 hours ago
Generating_geomedian_composites.ipynb	3 hours ago
Image_segmentation.ipynb	3 hours ago
Imagery_on_web_map.ipynb	3 hours ago
Integrating_external_data.ipynb	3 hours ago
Masking_data.ipynb	3 hours ago
Monitoring_water_quality.ipynb	3 hours ago
Principal_component_analysis.ipynb	3 hours ago
Rasterize_vectorize.ipynb	3 hours ago
README.md	3 hours ago
Tidal_modelling.ipynb	3 hours ago
Using_load_ard.ipynb	3 hours ago
Working_with_time.ipynb	3 hours ago

The **Frequently_used_code** folder contains a series of notebooks that demonstrate code snippets doing useful and common functions. For example, want to calculate the Normalised Difference Vegetation Index (NDVI)? You could set up your own formula — but it would be faster and easier to use the NDVI function as shown in `Calculating_band_indices.ipynb`. How about drawing a contour line between water and land? Check out `Contour_extraction.ipynb`. There are many useful tips and tricks in the **Frequently_used_code** folder.

It is helpful to know what is in this folder, so you can draw upon the code if you need to do something similar when writing your own notebooks.

Suggested activity: Read all the titles of the notebooks in the folder. Open `Masking_data.ipynb` and run through its contents by selecting **Kernel -> Restart Kernel and Clear All Outputs** and then executing cells as you go.

9.5.3 Real world examples

/ Real_world_examples /		
Name		Last Modified
dask-worker-space		a month ago
Burnt_area_mapping.ipynb		3 hours ago
Change_filmstrips.ipynb		3 hours ago
Chlorophyll_monitoring.ipynb		3 hours ago
Coastal_erosion.ipynb		3 hours ago
Crop_health.ipynb		3 hours ago
Intertidal_elevation.ipynb		3 hours ago
Machine_learning_with_ODC.ipynb		3 hours ago
Radar_water_detection.ipynb		3 hours ago
README.md		3 hours ago
Vegetation_change_detection.ipynb		3 hours ago
Water_extent.ipynb		3 hours ago
Wetlands_insight_tool.ipynb		3 hours ago

The analytical power of the Sandbox is shown in the folder **Real_world_examples**. This folder contains multiple notebooks which cater to ‘real world’ use cases, from monitoring crop health to detecting water.

The code in these notebooks is more complex, but it is exciting to see what Digital Earth Africa can do.

Suggested activity: Pick a notebook that interests you and run through it.

9.5.4 Datasets

/ Datasets /		
Name		Last Modified
Climate_Data_ERA5_AWS.ipynb		2 minutes ago
Fractional_cover.ipynb		2 minutes ago
Landsat_collections.ipynb		2 minutes ago
README.md		2 minutes ago
Sentinel_1.ipynb		2 minutes ago
Sentinel_2.ipynb		2 minutes ago
Soil_Moisture.ipynb		2 minutes ago
Water_Observations_from_Space.ipynb		2 minutes ago

The notebooks in the Sandbox are all based on Earth observation data available to Digital Earth Africa. These datasets are explained in individual notebooks in the **Datasets** folder. Each notebook contains background information about the dataset and examples of use.

Some of the datasets are based directly on satellite measurements. For example, information on the Landsat and Sentinel satellite datasets can be found in their respective notebooks:

- `Landsat_collections.ipynb`
- `Sentinel_1.ipynb`
- `Sentinel_2.ipynb`

Other datasets refer to derived products which have been created to show a particular environmental feature. These include:

- `Soil_Moisture.ipynb`
- `Fractional_cover.ipynb`
- `Water_Observations_from_Space.ipynb`

More dataset notebooks are added to this folder as the Digital Earth Africa collection expands. Use the notebooks in this folder to familiarise yourself with the available data and products.

Suggested activity: Run through the notebook on Sentinel 2 data, `Sentinel_2.ipynb`.

9.5.5 Other folders

There are a few other folders in the home folder, such as **Scripts** and **DCAL**. Feel free to have a browse, but they are not important to know about when you are starting out.

Suggested activity: Ignore these folders for now.

9.5.6 Help! I deleted a file...

See the *Frequently asked questions* section on restoring pre-loaded notebooks.

9.6 Quiz index

Note: The Digital Earth Africa Training website will experience a scheduled downtime from 31 May 2021 to 2 June 2021 to upgrade data hosting services. During this time, the Sandbox will be cleared of personal user data. The Sandbox may be inaccessible and you may experience some code errors with notebooks and quizzes. Sandbox users will have received an email regarding these changes. This Training website will be updated to reflect changes to data loading processes following the upgrade.

To be eligible for a DE Africa Training Course Certificate of Completion, you must successfully complete all six assessment quizzes. The links to the quiz page for each week can be found below.

On each quiz page, read the summary of the week's contents and note any suggestions or requirements for the quiz. Click the link labelled **complete the quiz** to access the questions.

You may submit more than one response to each quiz. Your highest score for each quiz will be used in assessing Certificate of Completion eligibility. Full marks must be scored to successfully pass each quiz.

If you have any questions about the assessment quizzes, or cannot remember which quizzes you have completed, see the *FAQ* or *contact the course convenors*.

Note: Please use the same email address with every quiz submission. This allows the course convenors to track your progress and award certificates in a timely manner.

- Session 1: Copy, open and run a notebook
- Session 2: Load data and plot an image
- Session 3: Plot a cloud-free geomedian
- Session 4: Calculate NDVI
- Session 5: Show vegetation change over time

- Session 6: Show water extent change over time

9.7 Geomedian widget

Geomedian composites are introduced in [Session 3](#). Geomedians are a method of combining multiple timesteps of data into one image.

Geomedians were defined as follows:

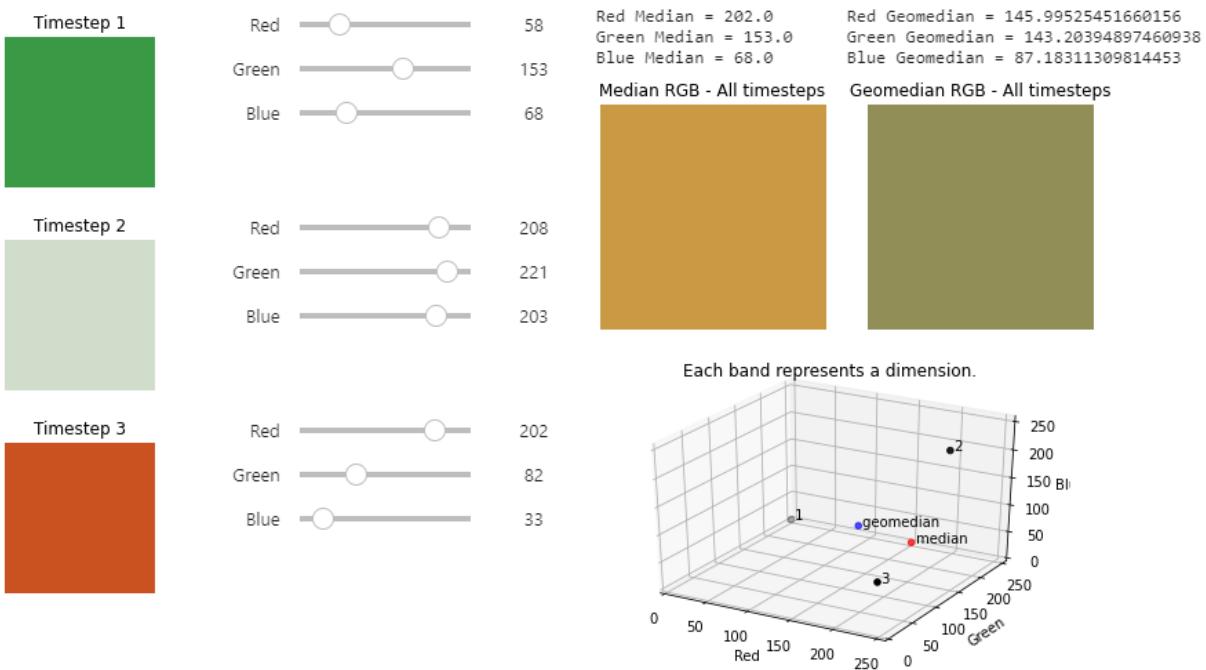
A geomedian composite finds the median values of the bands for each pixel when considered **together**. This means they represent the data **better** than median composites.

This gives rise to a few common questions.

- What does it mean to **consider the bands together**?
- What might that look like?
- How does it affect the colour of each pixel in my final composite image?

To explore the answers to these questions, we can use an interactive module known as a **widget**. Instead of looking at one whole image, the widget focuses on a single pixel.

The geomedian widget gives three timesteps of data. You can click on the sliders to change the data for each timestep, which affects the pixel colour. In turn, this will impact the computation of the median and geomedian. As you will see, they are not always the same!



An example of what the geomedian widget looks like. To interact with the widget, download and run the geomedian widget notebook.

Follow the instructions below to download the widget notebook. It can help with understanding the following concepts:

- In which cases are the median and geomedian very similar or the same?
- In which cases are the median and geomedian very different?

- Why are geomedians more representative of the whole dataset, compared to medians?

9.7.1 Download geomedian widget notebook

Download the geomedian widget notebook

To view this notebook on the Sandbox, you will need to first download it to your computer, then upload it to the Sandbox. Follow these instructions:

1. Download the notebook by clicking the link above.
2. On the Sandbox, open the **Training** folder.
3. Click the **Upload Files** button as shown below.



4. Select the downloaded notebook using the file browser. Click **OK**.
5. The solution notebook will appear in the **Training** folder. Double-click to open it.

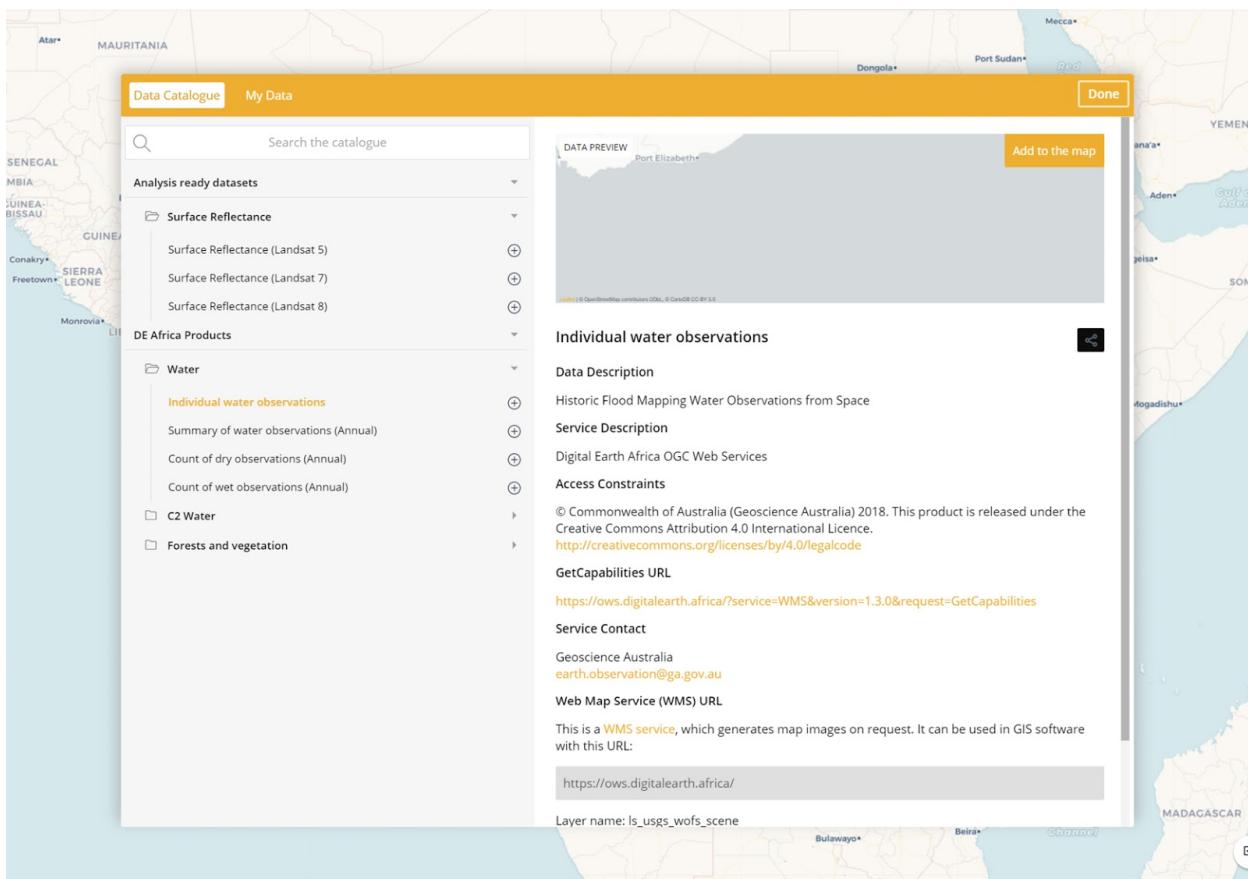
9.8 Map portal user guide

This guide provides detailed instructions on how to use the different features of the Digital Earth Africa Map portal at <https://maps.digitalearth.africa/>.

9.8.1 Get Started

To launch the DE Africa Map and display some basic data follow these steps.

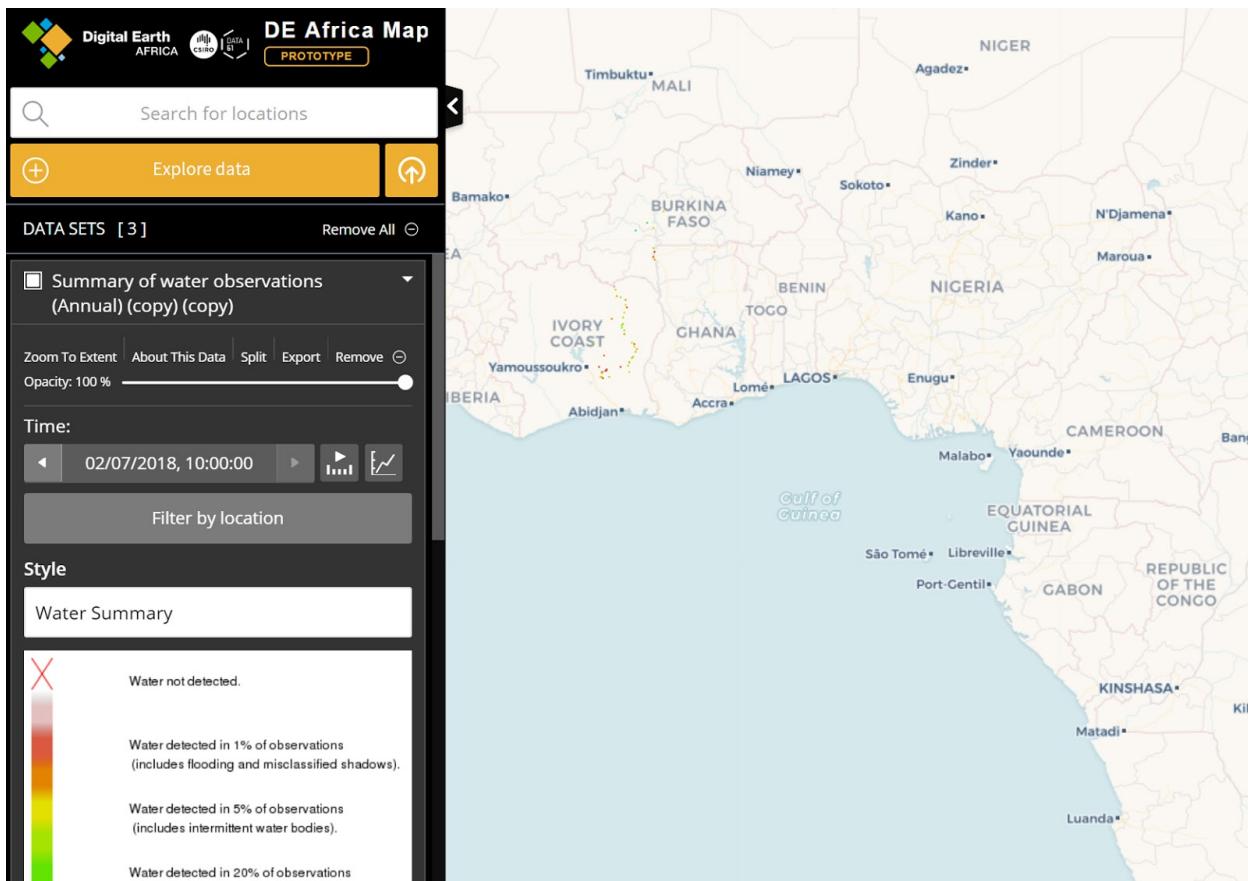
- Launch the DE Africa Map at maps.digitalearth.africa.
- In the left panel click the **Add Data** button to open the data catalogue (see image below).
- Find a dataset of interest such as water.
- Select the dataset to see a preview of that data and description.
- To view your selected dataset on the map, click the **Add to the map** button.



Visualise and analyse your dataset:

- The spatial data will be displayed in the map view, and a visual legend will appear in the Data workbench, on the left side of the page.
- It may not be immediately obvious where your selected spatial data has loaded on the map if it does not cover a large part of Africa. To locate loaded data on the map, go to the workbench, and select **Zoom to extent**.
- The dataset should now be visible in the map view.
- To add additional datasets to the map, repeat the above steps.
- Zoom manually by moving your mouse pointer over the map and using your mouse wheel to zoom in or out further.
- Click and drag the map to further show the region in which you are interested.

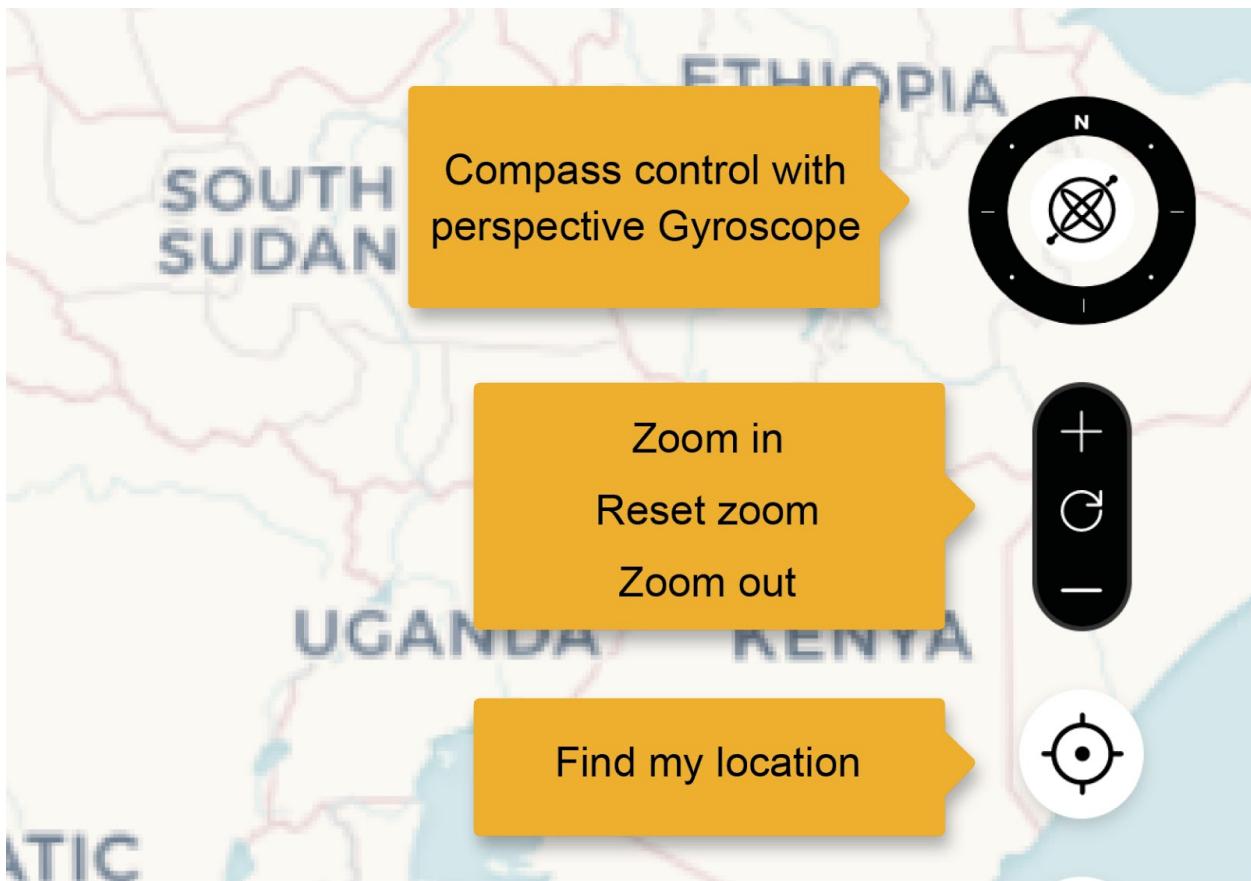
9.8.2 Explore the workbench



When a dataset is added to the map via the data catalogue, a legend for that dataset will appear in the workbench. From the workbench you can:

- Set the order data is shown on the map. To do this, click the title of a dataset and drag it to a new position in your workbench.
- Toggle the visibility of added datasets by selecting the checkbox opposite your preferred dataset title
- Zoom to the geographical extent of an added data set
- Set the opacity of individual data sets
- Remove data sets from the map. Note: they can be re-added via the data catalogue.

9.8.3 Navigate the Map



There are multiple ways to navigate DE Africa Map's map view:

Zooming

Move your mouse pointer over the map and use the scroll wheel to zoom in or out. The location at the centre of the map display is the centre of the zooming. Right-click and drag upwards or downwards over the map to zoom about the centre point. Select the zoom control to zoom in or out quickly by a set amount.

Panning

Click anywhere on the map and drag it to the required location.

Rotate the map

Use the compass control to rotate the map so North is no longer at the top. Select the “gyroscope” in the centre of the compass control and drag slowly to the left or right to rotate the map clockwise or anti-clockwise respectively. The further you drag, the faster it rotates. Release the mouse button when you reach the desired rotation.

Select the North Point or outer ring of the Compass Control and drag it around to set the desired rotation directly.

Control + left-click and drag left or right over the map to rotate the view about the centre.

Perspective view

Select the “gyroscope” in the centre of the compass control and drag slowly upwards to tilt the view into a perspective view. Drag downwards to tilt the view back to vertical. The further you drag, the faster it tilts. Release the mouse button when you reach the desired view.

Control + left-click and drag upwards or downwards over the map to enter or adjust the perspective view.

When you are in perspective view, control + left-click and drag left or right over the map to “orbit” around the centre of the view. You can also click and drag left or right over the “gyroscope” at the centre of the Compass Control to orbit about the centre of the view.

Double-click the “gyroscope” in the centre of the Compass Control to return the view quickly to a vertical view with North to the top at the current location and scale.

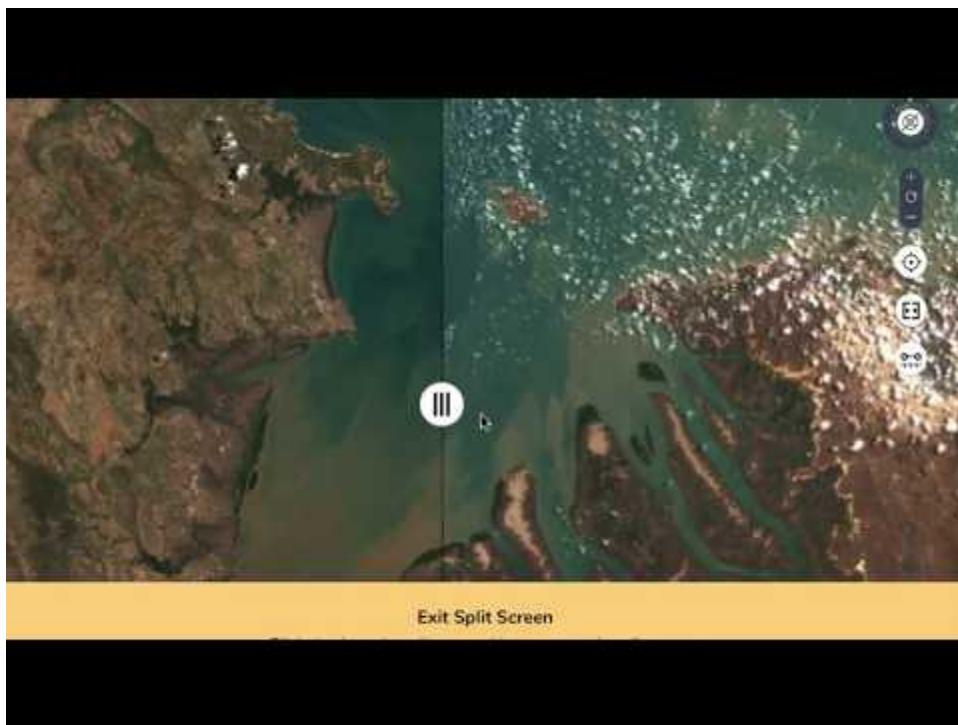
Dragging to pan and using the mouse wheel to zoom still work while showing a perspective view.

Use the splitter functionality

The splitter functionality allows you to compare a dataset for different time periods. To use it:

- Select an area/location
- Select a dataset from the catalogue to add it to the workbench
- Select the **Split** link in the workbench to create a copy of the data already selected. Note that you can use the splitter with two different datasets, not just with copies of the same data
- Select different times (using the date picker) for the “left” and “right” sections of the screen
- Use the back and forward arrows to explore imagery and select cloud-free views
- Drag the splitter on the screen to observe the differences

This video shows an example of the splitter used over a map of Australia, but the functionality is the same.



9.8.4 Upload Data to the Map

There are two ways to load your data:

1. Drag your data file onto the DE Africa Map map view. The format of the data file will be auto-detected.
2. Select **Add Data** in the left panel. This will launch the data catalogue. Select the **My Data** tab at the top of the modal window and follow the provided instructions.

As for DE Africa Map data sets, you can select regions or points to see the data available for that location. If the file is a CSV file, the data from all columns will be shown in the feature information dialogue.

You can also use all of the features of the workbench on the data you have loaded as well.

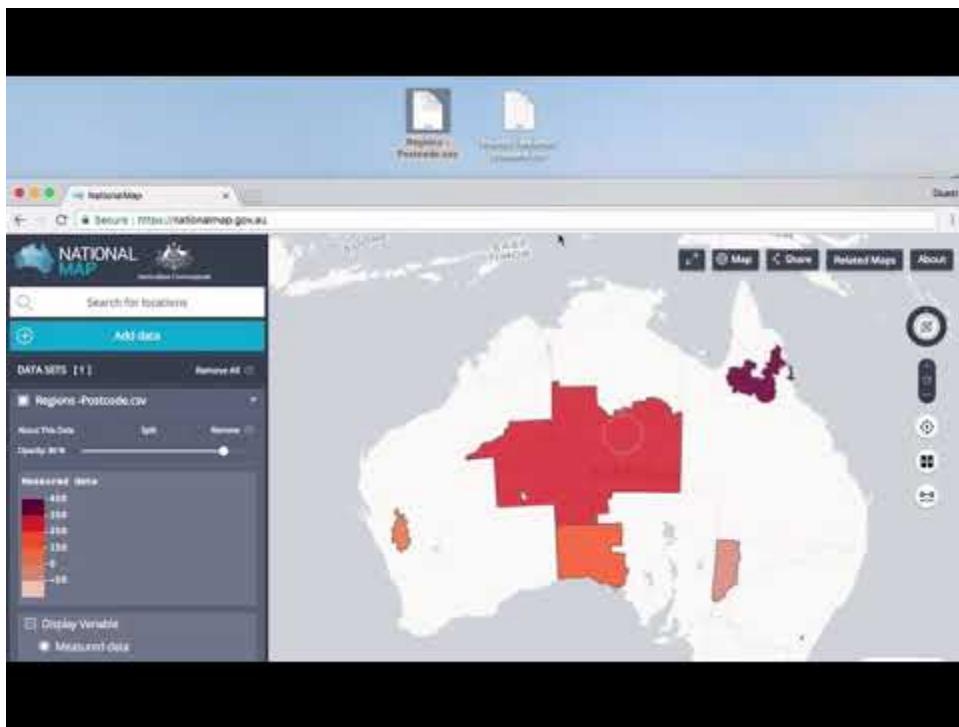
To share a view of your data with others, you must first publish it to the web somewhere with a URL, and then load it from there.

DE Africa Map can display two kinds of spreadsheets:

1. Spreadsheets with a point location (latitude and longitude) for each row, expressed as two columns: lat and lon. These will be displayed as points (circles).
2. Spreadsheets where each row refers to a region such as a local government area (council), state, postcode, or ABS statistical unit such as an SA2 or CED (Commonwealth Electoral Division). Columns must be named according to the CSV-geo-au standard. These will be displayed as regions, highlighting the actual shape of each area.

Spreadsheets must be saved as CSV (comma-separated values).

Other standard spatial data types such as GeoJSON and KML are also supported.



9.9 GIS web services

9.9.1 Web Map Service / Web Coverage Service

A Web Map Service (WMS) is an Open Geospatial Consortium (OGC) standard that allows users to remotely access georeferenced map images via secure hypertext transfer protocol (HTTPS) requests.

DE Africa provides two types of maps services:

- Web Map Service (WMS) – A standard protocol for serving georeferenced map images over the internet that are generated from a map server using data from a GIS database. It is important to note that with a WMS, you are essentially getting an image of geospatial data (i.e. JPG, GIF, PNG file). While this has its uses, it is an image only, and therefore does not contain any of the underlying geospatial data that was used to create the image.
- Web Coverage Service (WCS) – A standard protocol for serving coverage data which returns data with its original semantics (instead of just pictures) which may be interpreted, extrapolated, etc., and not just portrayed. Essentially, a WCS can be thought of as the raw geospatial raster data behind an image. Using a WCS, you can pull the raw raster information you need to perform further analysis.

So, to give a quick summarisation, a WMS is simply an image of a map. You can almost think of this like taking a screenshot of Google Maps. A WCS is the raw raster data, so for example, if you are working with a WCS containing Landsat imagery, you can effectively chunk off the piece you are interested in and download the full multispectral image at the spatial resolution of the original image. The beauty of these services is that you can grab only the information you need. So, rather than retrieving a file that contains the data you are seeking and possibly much more, you can confine your download to only your area of interest, allowing you to get what you need and no more.

For more information, see this article on the [difference between GIS web services](#).

The tutorials below cover setting up WMS and connecting to WCS.

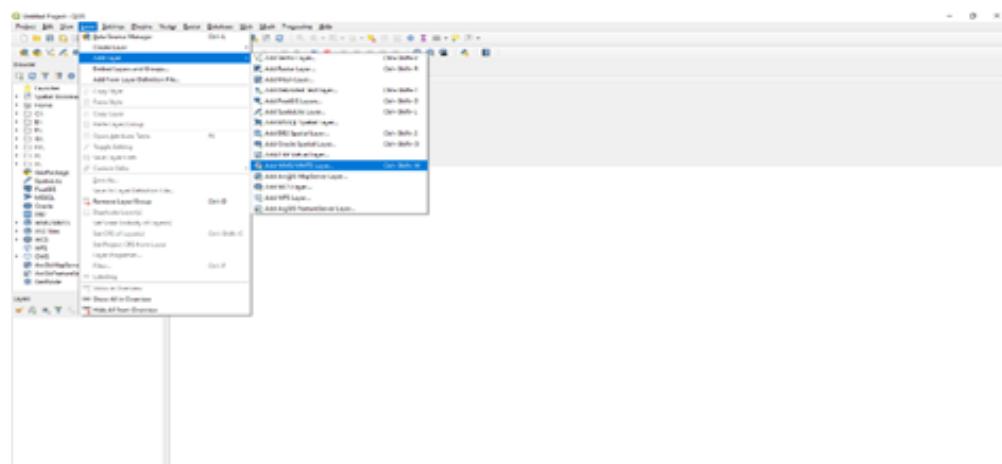
9.9.2 Tutorial: Setting up WMS

This tutorial shows how to set up the Web Map Services in QGIS, and use it with other data on your computer such as drone imagery, vector or raster data. This may be useful for you if you cannot upload the data to the DE Africa Map or the DE Africa Sandbox due to uploading due to size or internet bandwidth. It may also be useful if you feel more comfortable doing analysis in a GIS application.

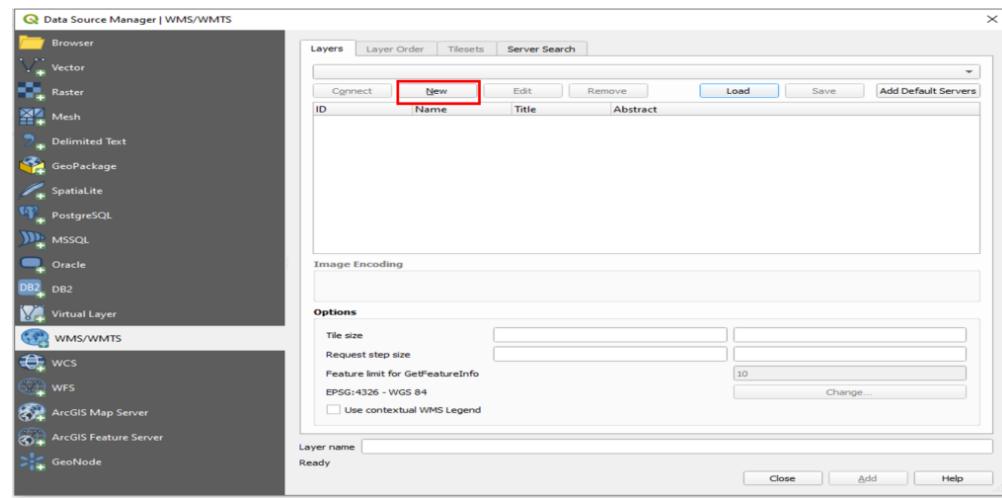
Although this tutorial focuses on QGIS, the same process can be used to connect other Desktop GIS applications. [QGIS](https://qgis.org/en/site/) is a free and open-source desktop GIS application. You can download it from <https://qgis.org/en/site/>.

How to connect to WMS using QGIS

1. Launch QGIS.
2. On the Menu Bar click on **Layer**.
3. A sub-menu tab will show below Layer; click on **Add Layer**, choose **Add WMS/WMTS Layer**.



4. A dialogue will open as shown below. Click on the **New** button.



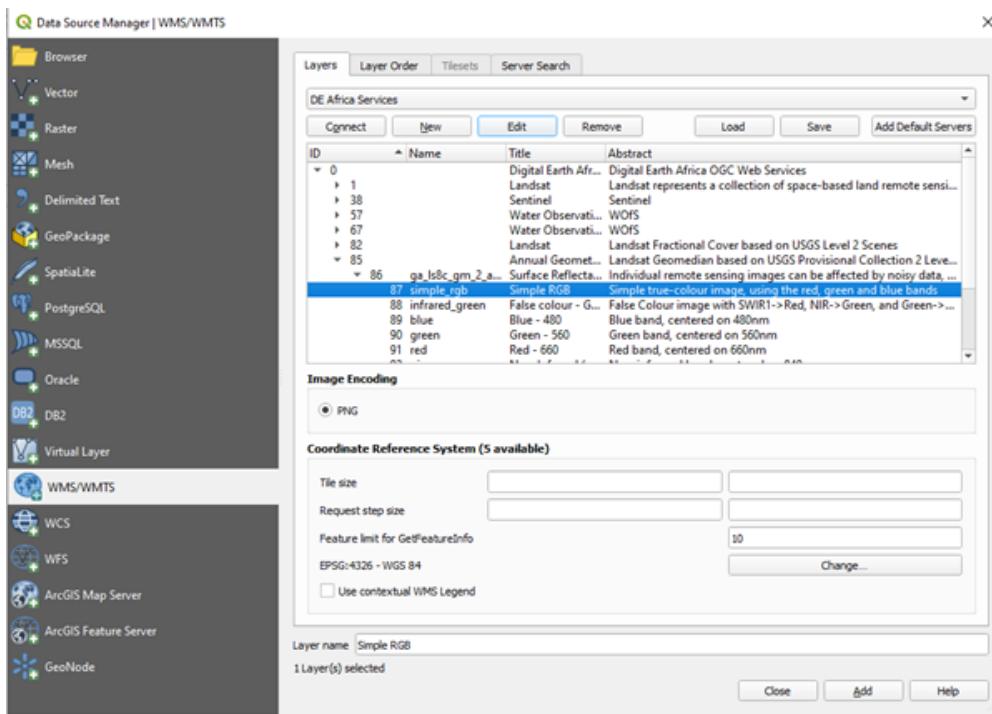
5. A dialogue will open, as shown below: Provide the following details, these can be found at the URL <https://ows.digitalearth.africa/>.

Name: DE Africa Services

URL: <https://ows.digitalearth.africa/wms?version=1.3.0>



6. After providing the details above, click on **OK**.
7. The previous dialogue will show up, in the dropdown above the **New** button, you will see DE Africa Services. If it is not there click the dropdown button below and select it.
8. The **Connect** button will be activated, click on it to load the layers. Anytime this page is open, because the connection has already been established, click on **Connect** to load the data.



9. The layer will be loaded as shown below in the dialogue.
10. Navigate through layers and choose the layer you will need to display on the Map Page.
11. After selecting the layer, click on **Add** button at the bottom of the dialogue.
12. Close the dialogue, the selected layer will be loaded onto the Map Page.

For web developers

The sites below provide instructions on how to load these map services onto your platform.

<https://leafletjs.com/examples/wms/wms.html>

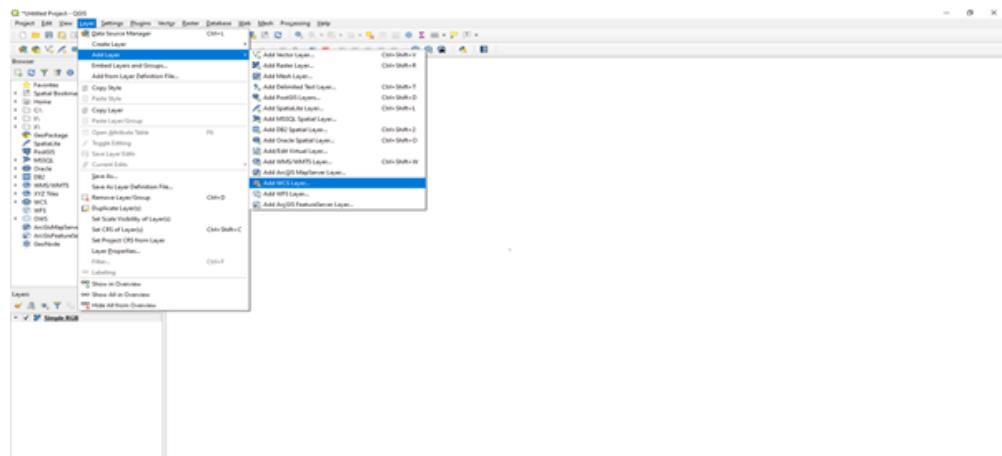
<https://openlayers.org/en/latest/examples/wms-tiled.html>

<https://docs.microsoft.com/en-us/bingmaps/v8-web-control/map-control-concepts/layers/wms-tile-layer-example>

9.9.3 Tutorial: How to connect WCS

This tutorial shows how to create a Web Coverage Service connection using QGIS.

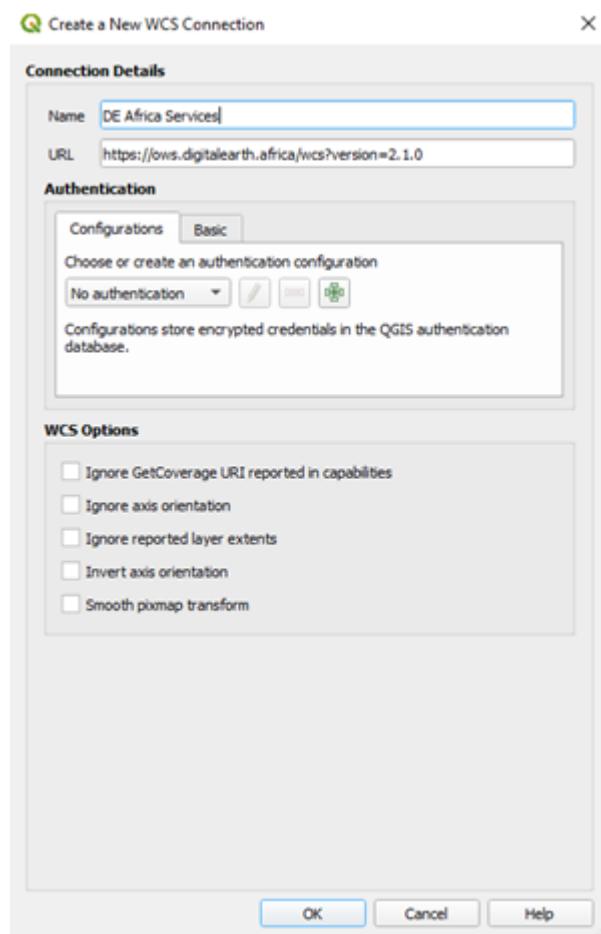
1. Launch QGIS.
2. On the Menu Bar click on **Layer**.
3. A sub-menu tab will show below Layer; click on **Add Layer**, choose **Add WCS Layer**.



4. Click on the **New** button.
5. A dialogue will open, as shown below: Provide the following details, these can be found at the URL <https://ows.digitalearth.africa/>

Name: DE Africa Services

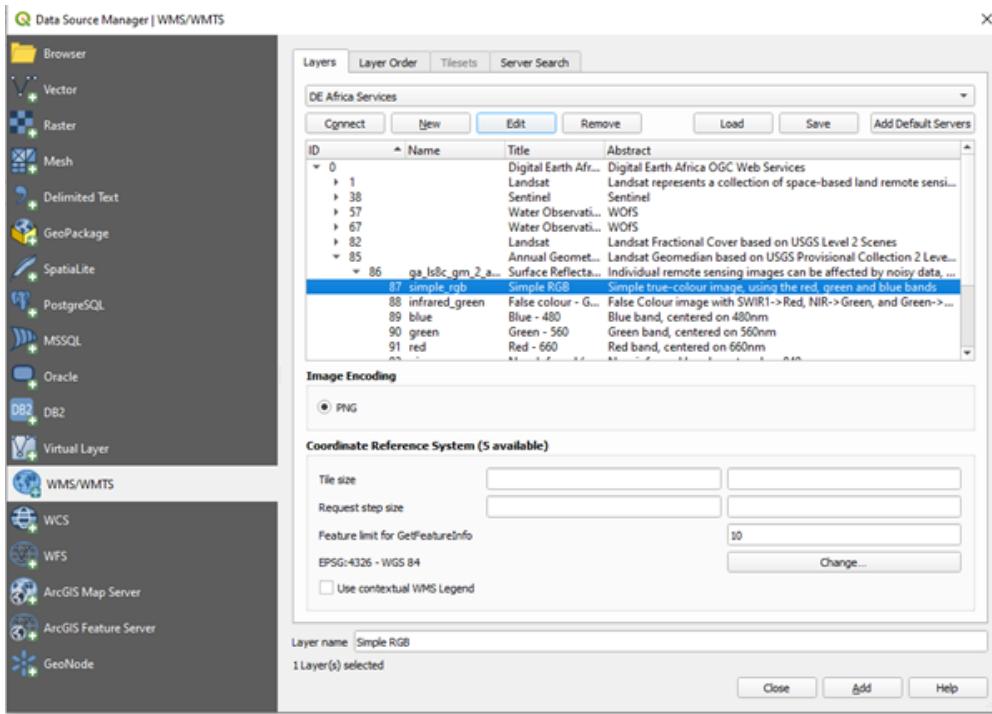
URL: <https://ows.digitalearth.africa/wcs?version=2.1.0>



6. After providing the details above, click on **OK**.
7. The previous dialogue will show up, in the dropdown above the New button, you will see DE Africa Services, if

it is not there click the dropdown button below and select it.

8. The **Connect** button will be activated, click on it to load the layers. Anytime this page is open, because the connection has already been established, click on the **Connect** button to load the data.
9. The layer will be loaded as shown below in the dialogue.



10. Navigate through layers and choose the layer you will need to display on the Map Page. With WCS you can select Time and Format of Image.
11. After selecting the layer click on the **Add** button at the bottom of the dialogue.

9.10 Import external datasets

The Digital Earth Africa Sandbox allows users to add external data such as shapefiles and .geojson files to their algorithms.

This tutorial will take you through:

1. The packages to import
2. Setting the path for the vector file
3. Loading the external dataset
4. Displaying the dataset on a basemap
5. Loading the satellite imagery by using the extent of the external dataset
6. Mask the area of interest from the satellite imagery using the extenal dataset

For this tutorial, the example external dataset is in a shapefile format.

Note: Before you proceed, ensure you have completed all lessons in the *DE Africa Six-Week Training Course*.

9.10.1 Set up notebook

In your **Training** folder, create a new Python 3 notebook. Name it `external_dataset.ipynb`. For more instructions on creating a new notebook, see the [instructions from Session 2](#).

Load packages and functions

In the first cell, type the following code and then run the cell to import necessary Python dependencies.

```
import sys
import datacube
import numpy as np
import pandas as pd
import geopandas as gpd

from datacube.utils import geometry

sys.path.append('../Scripts')
from deafrica_datahandling import load_ard, mostcommon_crs
from deafrica_plotting import map_shapefile, rgb
from deafrica_spatialtools import xr_rasterize
```

Take note of the packages below on how they were imported with other packages above.

These packages are the packages you will need when you want to use external dataset.

```
import geopandas as gpd
from datacube.utils import geometry
from deafrica_plotting import map_shapefile
from deafrica_spatialtools import xr_rasterize
```

Connect to the datacube

Enter the following code and run the cell to create our `dc` object, which provides access to the datacube.

```
dc = datacube.Datacube(app='import_dataset')
```

Create a folder called **data** in the **Training** directory. Download this [zip file](#) and extract on your local machine. Upload the **reserve** shapefile (cpg, dbf, shp, shx) into the **data** folder.

Create a variable called `shapefile_path`, to store the path of the shapefile as shown below.

```
shapefile_path = "data/reserve.shp"
```

Read the shapefile into a GeoDataFrame using the `gpd.read_file` function.

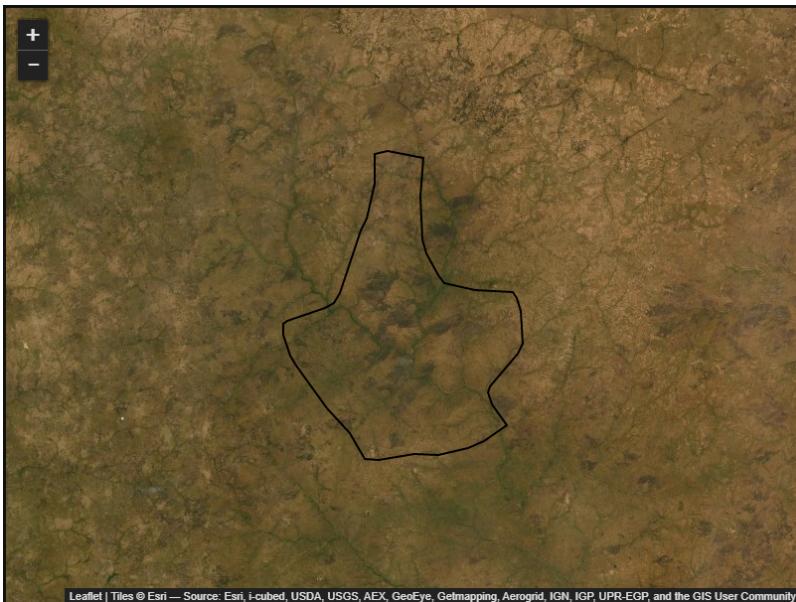
```
gdf = gpd.read_file(shapefile_path)
```

Convert all of the shapes into a datacube geometry using `geometry.Geometry`

```
geom = geometry.Geometry(gdf.unary_union, gdf.crs)
```

Use the `map_shapefile` function to display the shapefile on a basemap.

```
map_shapefile(gdf, attribute=gdf.columns[0], fillOpacity=0, weight=2)
```



9.10.2 Create a query object

We will replace `x` and `y` with `geopolygon`, as shown below. We remove the `x`, `y` arguments and replace it with `geopolygon`.

```
query = {
    'x' : x,
    'y' : y,
    'group_by': 'solar_day',
    'time' : ('2019-01-15'),
    'resolution': (-10, 10),
}
```

Remove `x`, `y` from `query` and update with `geopolygon`:

```
query = {
    'geopolygon' : geom,
    'group_by': 'solar_day',
    'time' : ('2019-01-15'),
    'resolution': (-10, 10),
}
```

We then identify the most common projection system in the input query, and load the dataset `ds`.

```
output_crs = mostcommon_crs(dc=dc, product='s2_l2a', query=query)

ds = load_ard(dc=dc,
              products=['s2_l2a'],
              output_crs=output_crs,
              measurements=["red", "green", "blue"],
```

(continues on next page)

(continued from previous page)

```
**query  
)
```

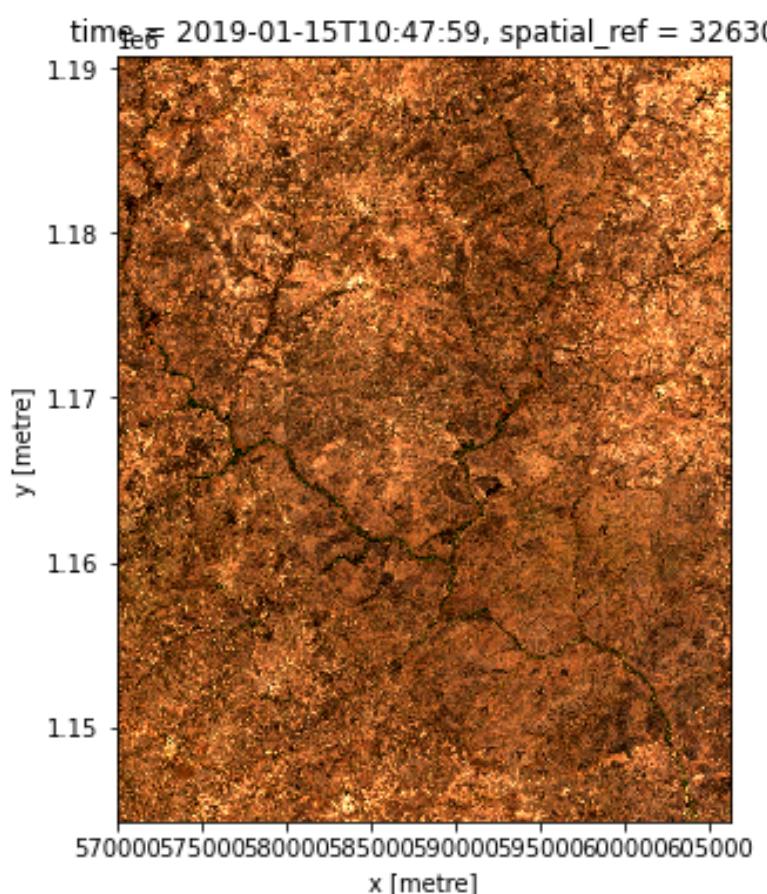
Print the ds result.

```
ds
```

9.10.3 Ploting of the result

We will display the returned dataset using the `rgb` functions.

```
rgb(ds)
```



9.10.4 Rasterise the shapefile

Before we can apply the shapefile data as a mask, we need to convert the shapefile to a raster using the `xr_rasterize` function.

```
mask = xr_rasterize(gdf, ds)
```

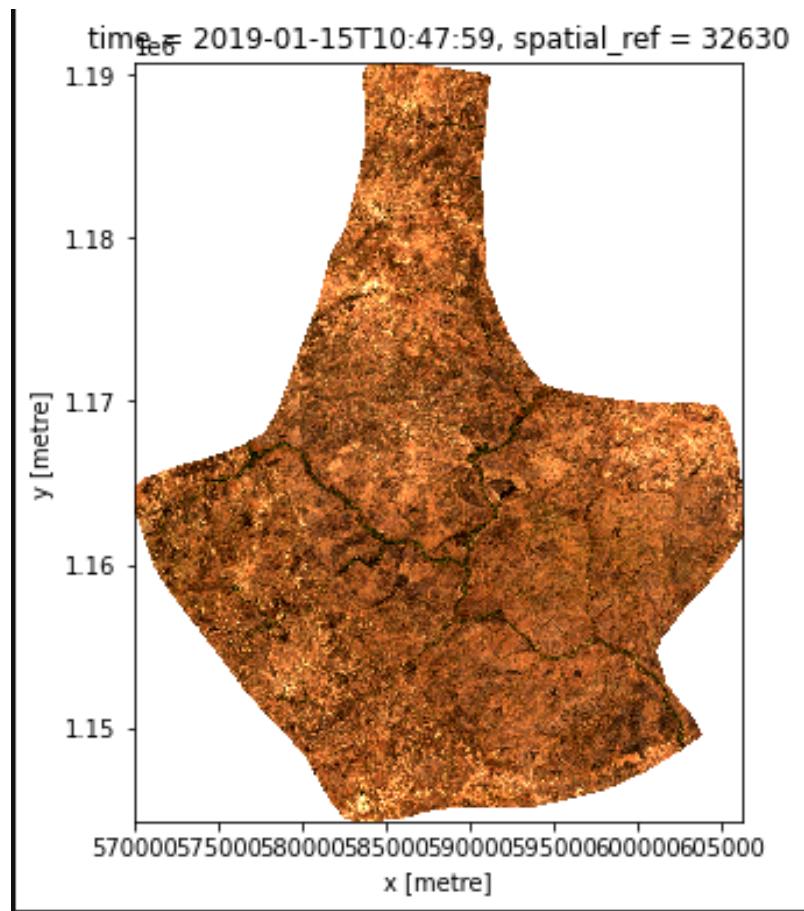
9.10.5 Mask the dataset

Mask the dataset using the `ds.where` and `mask` to set pixels outside the polygon to NaN.

```
ds = ds.where(mask)
```

Plot the masked result of the dataset

```
rgb(ds)
```



9.10.6 Conclusion

You can apply this method to already existing notebooks you are working with. It is useful for selecting specific areas of interest, and for transferring information between the Sandbox and GIS platforms.

9.11 License information

Digital Earth Africa data and training material is licensed under the [Creative Commons by Attribution 4.0](#) license.

The code featured in this site is licensed under the [Apache License, Version 2.0](#).