



OPEN  
DATA  
CUBE



Digital Earth  
AUSTRALIA



Australian Government  
Geoscience Australia

FRONTIER **S**  
**I** >

# DIGITAL EARTH AUSTRALIA TRAINING WORKSHOP

VIEW, ACCESS AND  
ANALYSE DATA

**Authors:** Caitlin Adams, Jess Keysers, Alex Leith,  
Robbi Bishop-Taylor and Claire Phillips

## Introduction

This workshop will introduce working with Digital Earth Australia (DEA) data in the DEA Sandbox environment for the Open Data Cube (ODC). The workshop is broken into the following sections:

1. Getting started – access the sandbox
2. Learning Jupyter – explore what a Jupyter Notebook is
3. Using Apps – run some simple apps demonstrating case studies
4. Do it yourself – run and modify Python code to load, analyse and visualise data

At the end of the workshop you will know how to use a Jupyter Notebook in conjunction with the ODC to access and analyse Earth observation data. The workshop should take around an hour to complete.

## Getting started

### Sign up for a DEA Sandbox Account

The DEA Sandbox uses requires you to create an account to log in. Please visit <https://app.sandbox.dea.ga.gov.au/> to sign up for a new account, or log in if you already have one.



Sign in with or signup

You'll need to be able to access the email you register with in order to proceed.

A screenshot of a web interface for account confirmation. It has a white background with a grey header and footer. The main content area contains the text: 'We have sent a code by email to a\*\*\*@g\*\*\*.com. Enter it below to confirm your account.' Below this is a label 'Verification Code' followed by a text input field containing six dots. At the bottom is a blue button labeled 'Confirm Account' and a link 'Resend it' in blue text.

We have sent a code by email to a\*\*\*@g\*\*\*.com.  
Enter it below to confirm your account.

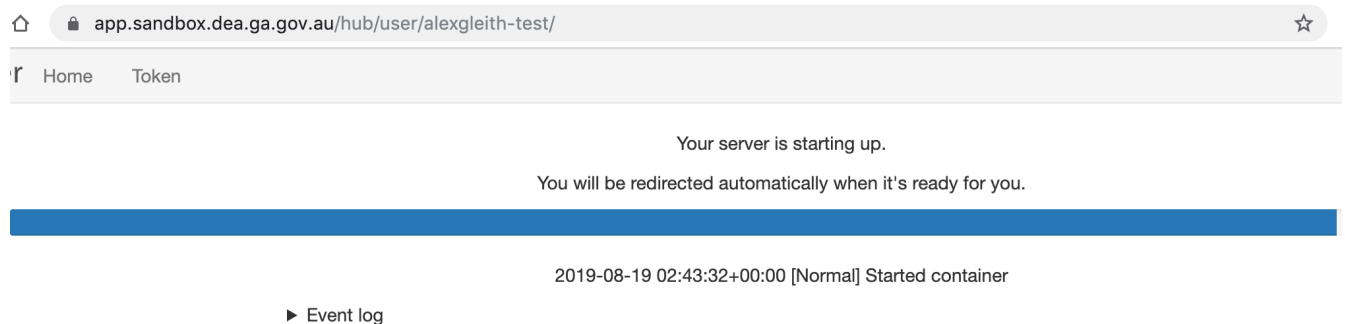
Verification Code

**Confirm Account**

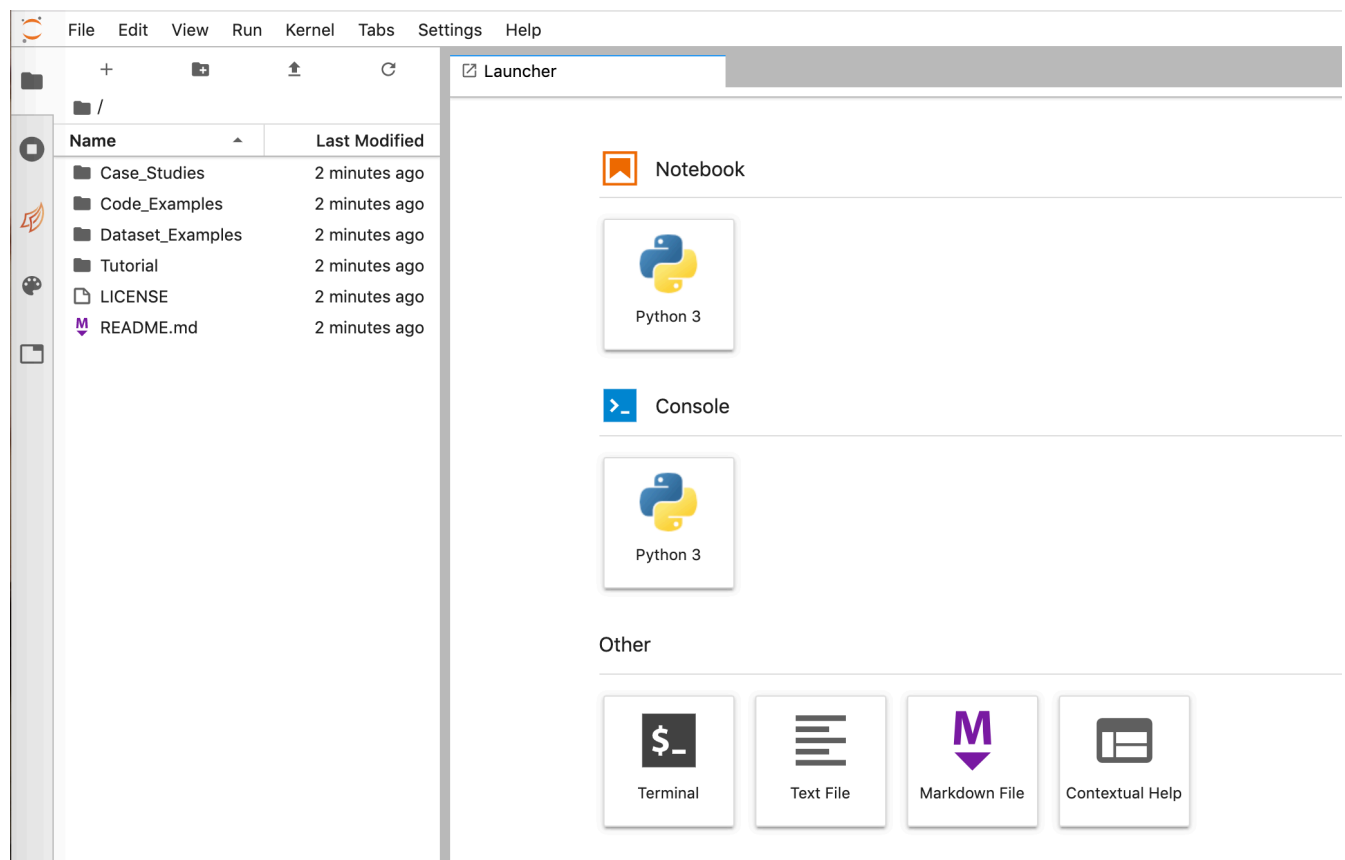
Didn't receive a code? [Resend it](#)

## Accessing the DEA Sandbox

After signing into the DEA Sandbox, your Jupyter environment will be created and you should see a loading screen (also shown below) while the system is working to prepare the environment.



Once signed in, the JupyterLab homepage should appear (see below).



## Learning Jupyter

### Overview

Jupyter is an interactive coding environment. The name 'Jupyter' comes from Julia, Python and R, which are all programming languages that are used in scientific computing. Jupyter started as a purely Python-based environment, called iPython, but there has been rapid progress over the last few years, and now many large organisations like Netflix<sup>1</sup> are using the system to analyse data.

Since the ODC is a Python library, the workshop will cover working with Earth observation data in Python-based notebooks.

### Explore a basic notebook

The first exercise is to explore a very basic notebook. The goal is to understand the key features of notebooks.

*If you've used Jupyter before, you may want to skip this step.*

Open the 'Tutorial' folder, and then open the file named '01\_Jupyter\_notebooks.ipynb'.

**Introduction to Jupyter notebooks**

- Prerequisites:**
  - There is no prerequisite learning required to use this notebook
  - It is designed for a novice user of the Jupyter environment

**Background**

Access to implementations of the [Open Data Cube](#) such as [Digital Earth Australia](#) and [Digital Earth Africa](#) is achieved through the use of Python code and [Jupyter Notebooks](#). The Jupyter Notebook (also termed notebook from here onwards) is an interactive web application that allows for the viewing, creation and documentation of live code. Notebook applications include data transformation, visualisation, modelling and machine learning. The default web interface to access notebooks when using the National Computational Infrastructure (NCI) and the DEA Sandbox is [JupyterLab](#).

To learn about the notebook, read through the text in the notebook and run each code cell step (using 'shift + enter'). Feel free to change the code sections to explore how it works.

*Note that when the section to the side of a cell is showing an asterisk, that means it is running. This is most important when running a data load that may take more than a few seconds*

The next image displays an executed cell.

Run the cell below:

```
[2]: print("I ran a cell!")
```

I ran a cell!

### Cell status

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.

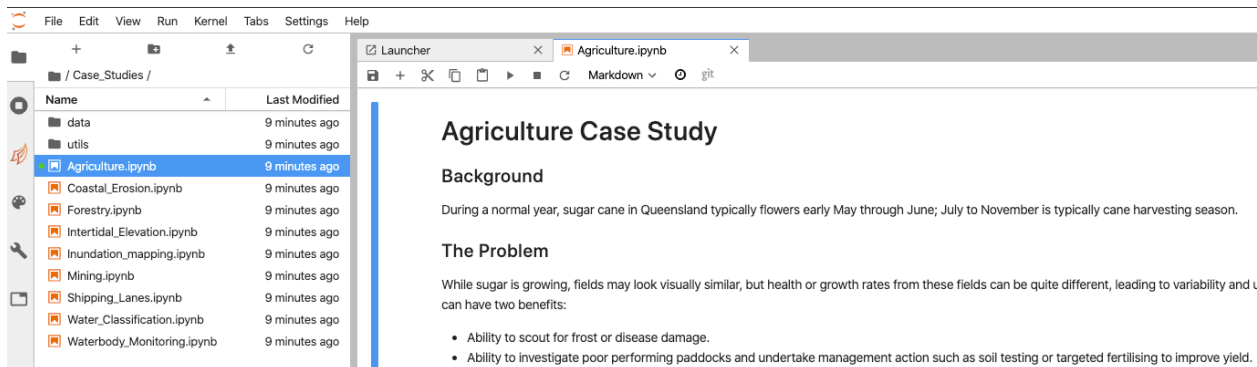
<sup>1</sup> <https://medium.com/netflix-techblog/notebook-innovation-591ee3221233>

## Using Apps

This example makes use of functions, which is where more complex code has been wrapped up into simple expressions. In this document, these functions are referred to as 'apps'. The 'apps' make it simple to change an analysis, as they provide ways to interact with the analysis that don't require changes to the underlying code.

## Agriculture app

From the 'Case\_Studies' folder, load the notebook named 'Agriculture.ipynb'.

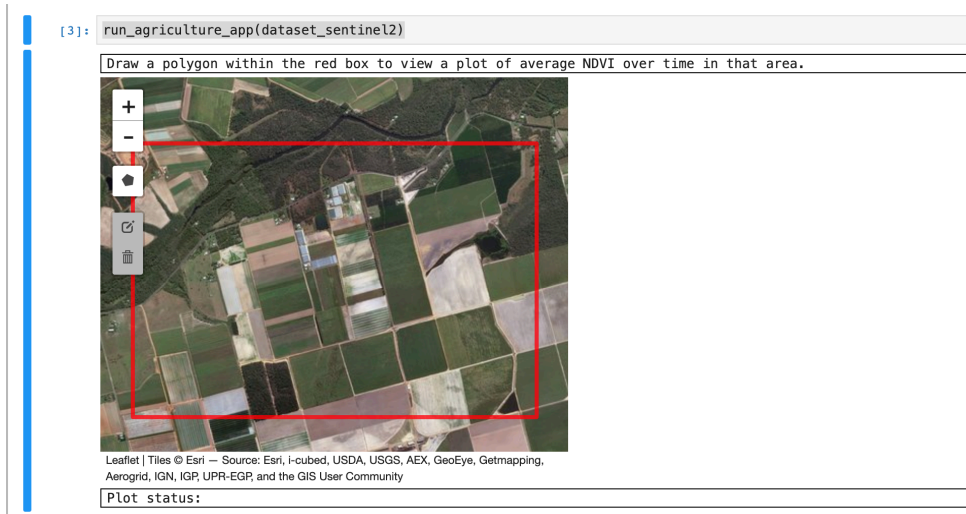


Read through the notes at the top of the notebook, and then run the first two cells of code. The first cell imports the required Python functions, which you can examine if you'd like to know how they work. The second cell runs one of the functions, which will load some data using the ODC. You should see some information logged by this function as it runs.

```
[2]: dataset_sentinel2 = load_agriculture_data()

Loading s2a pixel quality
Loading 8 filtered s2a timesteps
Loading s2b pixel quality
Loading 9 filtered s2b timesteps
Combining and sorting s2a, s2b data
Replacing invalid -999 values with NaN (data will be coerced to float64)
```

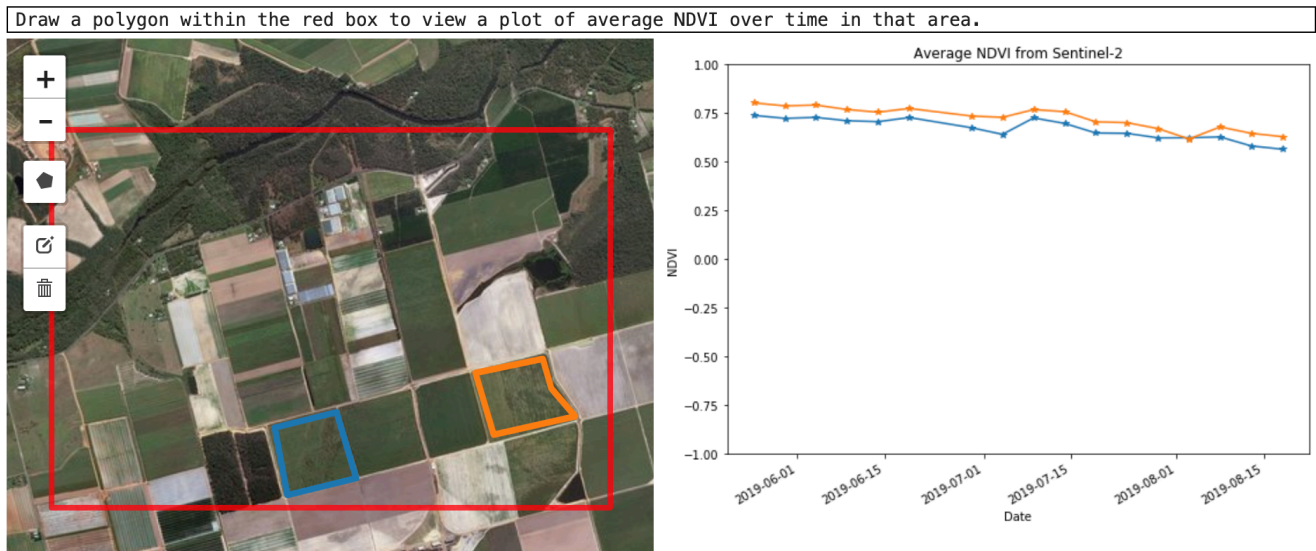
Run the next cell, with the 'run\_agriculture\_app(dataset\_sentinel2)' function in it. This will display an interactive map with a red box showing the area we loaded data for.



In this interactive map, draw polygons to delineate two distinct fields by using the 'Draw a polygon' tool .

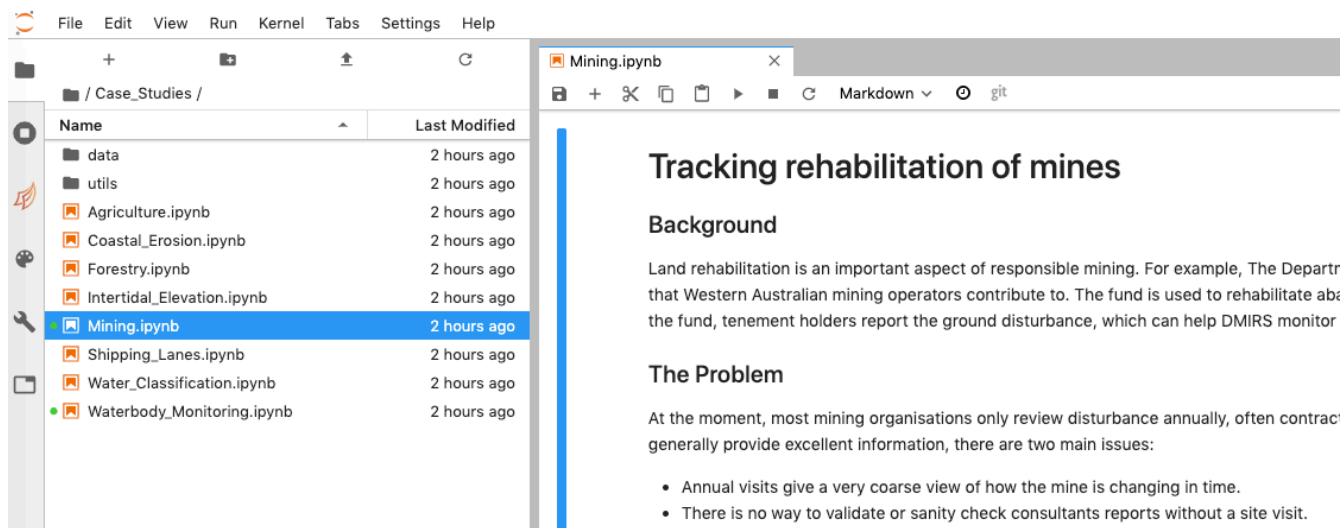
After drawing both polygons, look at the graph produced. Identify the differences between paddocks. The graph displays the normalised difference vegetation index<sup>2</sup>, which indicates the presence of vegetation. High values (approaching one) indicate healthy vegetation, while low values (approaching negative one) are often cloud or snow. Continue delineating a few more fields and see if there are any differences between them.

*Note that the graph will look different to the one below, as the Sentinel near real-time data being used is continually updated.*



## Mining app

To try out another 'app' based notebook, load the notebook named 'Mining.ipynb' from the 'Case\_Studies' folder. This is an optional exercise.



**Tracking rehabilitation of mines**

**Background**

Land rehabilitation is an important aspect of responsible mining. For example, The Department of Water and Environmental Regulation (DWER) that Western Australian mining operators contribute to. The fund is used to rehabilitate abandoned mines, tenement holders report the ground disturbance, which can help DMIRS monitor

**The Problem**

At the moment, most mining organisations only review disturbance annually, often contractors generally provide excellent information, there are two main issues:

- Annual visits give a very coarse view of how the mine is changing in time.
- There is no way to validate or sanity check consultants reports without a site visit.

<sup>2</sup> [https://en.wikipedia.org/wiki/Normalized\\_difference\\_vegetation\\_index](https://en.wikipedia.org/wiki/Normalized_difference_vegetation_index)

## Do it yourself

### Overview

This activity uses a code-based (rather than app-based) notebook, to demonstrate how the ODC Python API works. This will be a simple example of picking a study site in Australia, loading data for that area and plotting bands into the red, green and blue channels of an image.

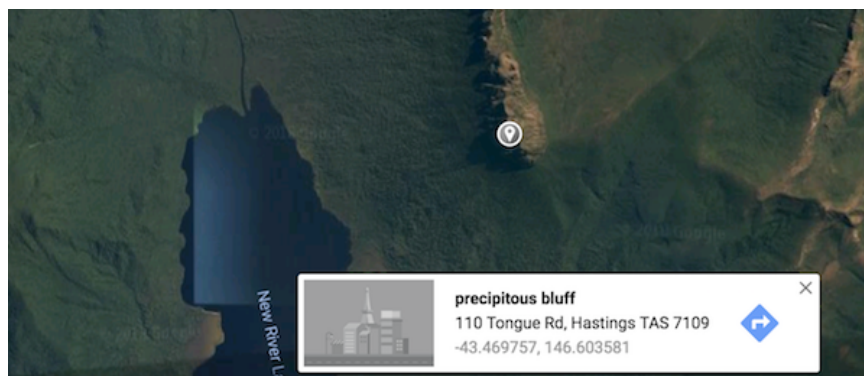
Load the notebook named '02\_Do\_it\_yourself.ipynb' from the 'Tutorial' folder.

*To keep any changes, save this file under a new name in this folder. This is because the example folders get reset at each log in. After opening the notebook, go to File > Save Notebook As..., then enter the name for the copy of the notebook.*

Next, read through the notebook and follow the steps. Feel free to run the cells, as this will work for the example study sites.

### Pick a study site

To set a study site, provide the latitude and longitude for the centre of an area, and the notebook will turn this into a bounding box, which is passed to the datacube load command. To get the latitude and longitude for any area in Australia, visit Google maps (<https://maps.google.com>), find somewhere that looks interesting, and then left-click on the map. A small info panel will pop up with some information including coordinates.



Paste these coordinates into square brackets in the first code cell. These coordinates will be used to create a bounding box to load data for the study site.

```
[ ]: # Supply the latitude and longitude coordinates for your study site here
      # Format them in the same manner as the examples above
      coordinates =
```

## Load data

To perform the datacube load, run the cells below the coordinates cell, including the one with the datacube load command. You will need to set the box size in the second code cell otherwise you'll get the following error.

```
File "<ipython-input-5-38370c876488>", line 3
    box_size =
            ^
SyntaxError: invalid syntax
```

*Note that the load command will take around one minute to fetch data from the AWS S3 cloud storage.*

```
[ ]: # Import necessary python packages
%matplotlib inline
import datacube
import warnings
warnings.filterwarnings('ignore') # suppress warnings

# Set up the datacube object
dc = datacube.Datacube(app='do-it-yourself')

# This command here does the loading of data
# Please be patient, it can take some time to load, depending on the size of your study area
dataset = dc.load(
    product='s2a_nrt_granule',
    x=bounding_box_x,
    y=bounding_box_y,
    resolution = (-10, 10),
    output_crs='epsg:3577',
    measurements=(
        'nbar_red',
        'nbar_green',
        'nbar_blue',
        'nbar_nir_1'
    )
)
```

After the data is loaded, run the cell below. This cell calls the 'dataset' object created in the previous cell, which will return some information about it.

Specifically, 'dataset' is an 'XArray' object containing the loaded data. It has three dimensions: time, longitude (x), and latitude (y). For the Dead Dog Creek example, with a box size of 0.05 degrees, the 'dataset' should contain something like 8 timesteps, 1194 cells in longitude and 1201 cells in latitude. This information comes from the 'Dimensions' heading.

Each timestep is an occasion that the satellite captured an image. The satellite also captures several different spectral 'bands', which are listed under the 'Data variables' heading. The returned variables correspond to the 'measurements' requested as part of the load command in the previous cell. Here, the 'nbar' word refers to processed data, which means the data has been adjusted to Australian conditions<sup>3</sup> (this is called 'analysis ready data').

<sup>3</sup> <https://docs.dea.ga.gov.au/data/data.html>

```
[6]: print(dataset)

<xarray.Dataset>
Dimensions:      (time: 8, x: 1194, y: 1201)
Coordinates:
  * time          (time) datetime64[ns] 2019-08-05T00:41:12.809861 ... 2019-10-24T00:41:11.766458
  * y             (y) float64 -1.617e+06 -1.617e+06 ... -1.629e+06 -1.629e+06
  * x             (x) float64 1.395e+06 1.395e+06 ... 1.407e+06 1.407e+06
Data variables:
  nbar_red        (time, y, x) int16 896 896 890 915 910 ... 1000 979 979 970 961
  nbar_green      (time, y, x) int16 1136 1107 1090 1137 ... 1185 1185 1167 1186
  nbar_blue       (time, y, x) int16 996 981 989 1004 985 ... 1167 1167 1137 1156
  nbar_nir_1      (time, y, x) int16 623 624 634 642 627 ... 1038 1038 1013 982
Attributes:
  crs:            epsg:3577
```

## Visualise the site

After loading data, it is important to visualise it. Fill in the missing values and then run the cell under the heading 'Plotting data'. To get more experience with interacting with Python code, change the timestep being plotted (should be a number between 0 and one fewer than the number of timesteps), and the band combination (should be a list of three data variables that belong to the dataset: ['var\_1', 'var\_2', 'var\_3']). As a stretch goal, include more bands in the datacube load command, and plot other measurements<sup>4</sup>, such as Coastal Aerosol.

```
[ ]: import matplotlib.pyplot as plt
      from utils.dea_plotting import rgb

      # Set the time step to view
      time_step =

      # Set the band combination to plot
      bands =

      # Generate the image by running the rgb function
      rgb(dataset, bands=bands, index=time_step, size=10)

      # Format the time stamp for use as the plot title
      time_string = str(dataset.time.isel(time=time_step).values).split('.')[0]
```

The data is formatted by a function called 'rgb()' which takes the 'dataset' object, as well as the bands and time step chosen above.

Finally, the plotting code constructs the figure and displays the image created.

```
# Set the title and axis labels
ax = plt.gca()
ax.set_title(f"Timestep {time_string}", fontweight='bold', fontsize=16)
ax.set_xlabel('Easting', fontweight='bold')
ax.set_ylabel('Northing', fontweight='bold')

# Display the plot
plt.show()
```

Here, you can change the title of the image, the font size, or axis labels.

<sup>4</sup> <https://en.wikipedia.org/wiki/Sentinel-2#Instruments>

## Save and download data

Once you have a site and timestep worth saving, run the next cell to write it out as a GeoTIFF, which can be loaded into a desktop GIS (e.g. QGIS) for further analysis. To download the rendered image, navigate to the Jupyter folder and find the image, right-click it, and choose 'download'.

```
[ ]: from datacube import helpers

# You can change this, if you like.
filename = "example.tiff"

helpers.write_geotiff(dataset=dataset.isel(time=time_step), filename=filename)
```

After saving the GeoTIFF, return to the cell defining the latitude and longitude coordinates for analysis and choose another set. Then rerun the analysis and visualise the newly selected area.

## Stretch goal (optional)

The final stretch goal, if there is time remaining in the session, is to do a simple band index calculation, manually calculating NDVI. See the notebook for tips on how to do this.

### Stretch goal: Calculate NDVI

If you've come this far and you'd like to do something a bit fancier, you can have a go at calculating the normalised difference vegetation index (NDVI) over your study site. There is a definition of what [NDVI](#) is on Wikipedia.

Basically, you need to use the following formula:

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

Some hints:

- You can access bands of an Xarray (the data format we're using) with their name, like this: `dataset.bandname`
- You can do simple math with bands by simply referring to them, like this: `dataset.bandname_1 + dataset.bandname_2`
- The two band names you're after are `nbar_nir_1`, which is near infra-red, and `nbar_red`, which is red.
- You can pass many arguments to the `.plot()` command to configure the image. One example is `cmap=colormap`, where `colormap` is the name of a [Matplotlib Colour Map](#). See if you can find a colour map that shows high values as green.

## References

- The Open Data Cube website: <https://www.opendatacube.org/>
- Geoscience Australia (including DEA) metadata: <http://cmi.ga.gov.au/>
- DEA data on Amazon Web Services' S3: <http://dea-public-data.s3-ap-southeast-2.amazonaws.com/index.html>
- National Map to view data: <https://nationalmap.gov.au/>