



Africa Regional Data Cube (ARDC)

Open Data Cube (ODC) Training Workshop

May 9th – 11th, Nairobi, Kenya



The Africa Regional Data Cube was developed by the Committee on Earth Observation Satellites (CEOS) in partnership with the Group on Earth Observations, Amazon Web Services, Office of the Deputy President – Kenya, Strathmore University in Kenya, Global Partnership for Sustainable Development Data, and Analytical Mechanics Associates.

Table of Contents

➤ Workshop Overview.....	4
➤ Workshop Agenda.....	6
➤ Module 1: AWS User Interface.....	9
➤ Module 2: Jupyter Notebooks.....	19
➤ Special Topic: Data Cube Management.....	35
➤ Data Cube User's Guide.....	41
➤ Open Data Cube Resources.....	70
➤ Survey.....	72
➤ Appendix: Jupyter Notebook Python Code	74

On behalf of the entire ARDC workshop team,

Thank you for your participation!

Africa Regional Data Cube (ARDC)

Workshop Overview

May 9-11, 2018

Strathmore University

Nairobi, Kenya

Objectives: The Africa Regional Data Cube (ARDC) Workshop is intended to introduce the Open Data Cube (ODC) to local users and provide hands-on training on the management of the data and approaches for applying application algorithms to produce relevant products for decision-making. The workshop is intended to be "invitation only" to maintain a focus on the ARDC and to achieve an operational condition at the end of the 3 days of training.

Attendees: Invitations will be given to representatives (e.g. National Statistics, Ministries of Environment or Agriculture, geospatial researchers or scientists) from the five ARDC countries (Senegal, Sierra Leone, Ghana, Kenya, Tanzania), Strathmore University (meeting host and ARDC data manager), GPSDD (ARDC initiator), Amazon (data storage and processing donor), U.K. Rhea (leading a similar effort for Uganda), U.K. Catapult (ODC Partner), SERVIR (manages a hub at RCMRD), Radiant Earth (interest in capacity building for agriculture), and the CEOS Systems Engineering Office - SEO (implementation and training lead).

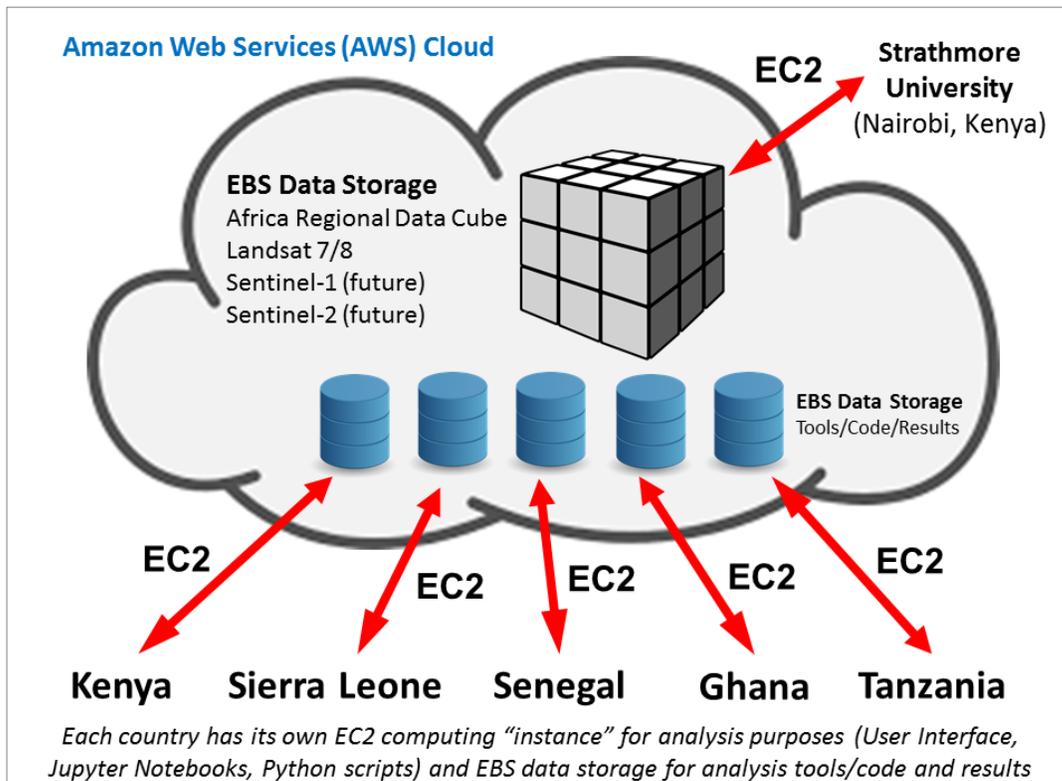
Non-technical attendees should have control of resources that might be applied to ARDC countries for capacity building or data management. These attendees will observe the training and be part of strategic discussions. Technical attendees should have a good knowledge of geospatial satellite data with an interest in learning how to use the data cube for the generation of application products or the management of such data. Python code experience is not required but is desired. Technical attendees should bring a laptop computer (or use one from Strathmore) with internet access and an installed GIS tool (e.g. QGIS or ArcGIS).

Overall Plan: Day 1 will be focused on stakeholder introductions, goals and objectives of the ARDC, and presentations by country representatives on their desired application needs and products. Day 2 and Day 3 will be focused on "hands on" training where the users are split into country-level groups and given tasks. Each group would be using the data from their country cube to develop application products. They will use the online user interface and also use Jupyter notebooks (Python scripting).

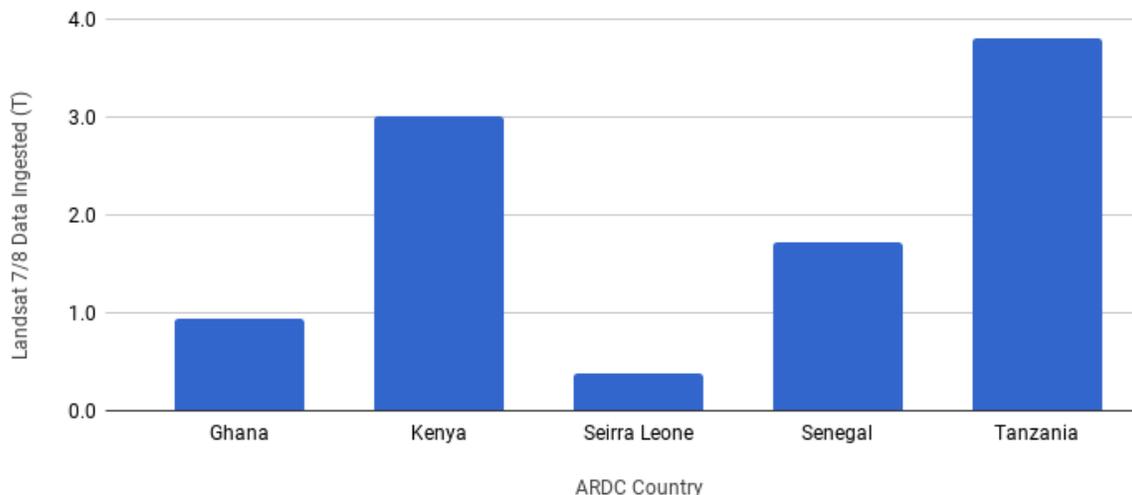


Africa Regional Data Cube Configuration

The ARDC has been deployed and configured on the Amazon AWS Cloud. We have deployed six EC2 instances to serve the initial needs of the ARDC. Each EC2 instance, combined with Elastic Block Storage (EBS), can be leveraged for analysis purposes (User interface, Jupyter Notebooks, Python scripts). *The initial configuration of the ARDC is 10 Terabytes of Landsat 7/8 Data.*



ARDC Ingested LS7/8 Data (Terabytes), Total ARDC Cube = 10T



ARDC Workshop Agenda - Wednesday, May 9th

8:30 AM	Registration	
9:00 AM	Opening Remarks / Logistics / Goals	Strathmore University
10:00 AM	ARDC Vision	GPSDD
10:30 AM	Open Data Cube: Background	CEOS SEO (NASA, AMA)
11:00 AM	<i>Break</i>	
11:30 AM	Open Data Cube: ARDC Plans	CEOS SEO
12:00 PM	Uganda IPP Project*	U.K. Rhea
12:30-2:00	<i>Lunch</i>	
Country Reports: <i>Each country will have 30 minutes to present their goals, objectives, and challenges regarding the use of the Data Cube. This might include a summary of desired application products and connections with national stakeholder agencies (e.g. statistical, environment). The outcome of this session will improve the "hands on" training sessions (Days 2 and 3) and will set the path for the future.</i>		
2:00 PM	Kenya Report	
2:30 PM	Ghana Report	
3:00 PM	Senegal Report	
3:30 PM	<i>Break</i>	
4:00 PM	Tanzania Report	
4:30 PM	Sierra Leone Report	
5:00 PM	<i>Adjourn</i>	
6:30 PM	<i>Dinner / Social</i>	

* An International Partnership Program (IPP) project supporting drought and flood mitigation using the Open Data Cube

ARDC Workshop Agenda - Thursday, May 10th**7:30-9:00 AM** **ARDC Launch Event** at the Radisson Blu Hotel**ARDC Training at Strathmore University**

9:30 AM	Data Cube Applications (summary)	CEOS SEO
10:00 AM	Fundamentals of the ODC Framework	AMA
10:45 AM	DEMO - Online User Interface	CEOS SEO
11:30 AM	<i>Break</i>	
12:00 PM	Jupyter Python Notebook Tutorial	AMA
1:00-2:00	<i>Lunch</i>	

Hands-On Training Module #1 (AWS Online User Interface)

The CEOS Systems Engineering Office (SEO) team will be available during the training period to answer questions and provide guidance

2:00 to 3:30 PM **Cloud-free Mosaics**

Each country group will generate a Landsat cloud-free mosaic in a small area of interest in their country using several methods (median, most recent pixel, geomedian) and compare the results in the user interface (UI) and in a GIS tool (e.g. QGIS). Mosaics should be long-term (annual) and short-term (seasonal). Users can also explore false-color RGB images, time-series animations and evaluate cloud issues in resulting mosaic products.

3:30 to 5:00 PM **Water Extent and Water Quality**

Each country group will develop a Landsat water extent product (Australian WOFS algorithm) for 2000 to 2017 and evaluate areas of water change due to seasonal flooding or single flood events. Users will then create separate annual water extent products to determine when and where the extreme events occurred and try to correlate those events with known changes in rainfall (droughts or heavy rains using precipitation data). Users will then create a water quality product (Total Suspended Matter - TSM) over a known reservoir to determine water quality variability and to assess changes in water quality over time.

2:00 to 5:00 PM **Special Topic: Data Cube Ingestion** (*Parallel Session)

Training on managing AWS instances, obtaining new data, and ingesting new data into the cube. This targeted training on data ingestion may also continue into Friday morning.

5:00 PM **Adjourn**

ARDC Workshop Agenda - Friday, May 11th

9:00 AM Recap from Training Day #2 *CEOS SEO + Country Feedback*
 Question and Answer + Feedback

Hands-On Training Module #2 (Jupyter Python Notebooks)

Each group will use a set of Jupyter Python Notebooks to create application products and evaluate those results. These will include:

- A. **Cloud-free Mosaics and K-means Clustering:** land classes, agriculture or forest mapping
- B. **WOFS and TSM** - water extent and quality variations in space and time
- C. **Fractional Cover and Spectral Indices (NDBI/NDVI)** - urbanization and vegetation
- D. **PyCCD or NDVI Trend** - land change, compare with Google Earth or GFW (forests)
- E. **Transect Analysis** - 2D and 3D plotting of bands and products
- F. **Data Export** - interfacing externally with QGIS and EXCEL

10:00 - 11:15 AM Task (A)

11:15 - 12:30 PM Task (B)

** informal break can be taken at any time*

12:30 to 2:00 *Lunch*

2:00 - 3:15 PM Task (C)

3:15 - 4:30 PM Task (D)

4:30 - 5:00 PM Task (E)

5:00 - 5:30 PM Task (F)

** informal break can be taken at any time*

5:30 PM Q&A / Wrap-up

6:00 PM Adjourn

Module 1

AWS Online User Interface

ARDC Training Workshop

Module #1 (AWS Online User Interface) - Exercise #1 (Cloud-free Mosaics)

Each country team will be given a separate website address and login information for their user interface.

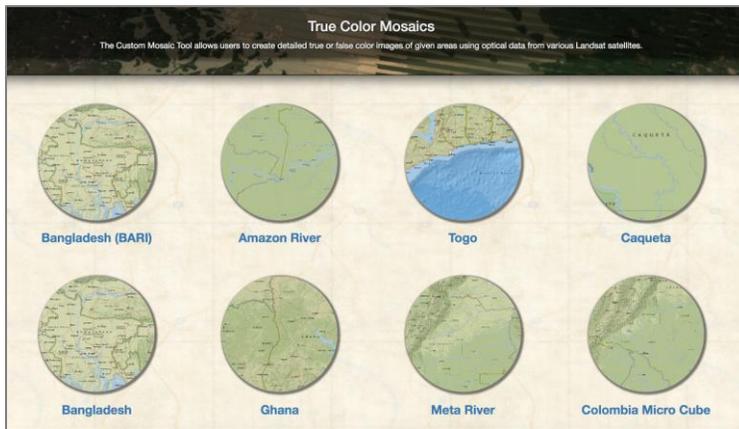
Objective: Create a Landsat cloud-free mosaic in a small region using several methods (median, most recent pixel) and compare the results in the user interface (UI) and in a GIS tool (e.g. QGIS). Users can also explore false-color RGB images, time-series animations and evaluate cloud issues in resulting mosaic products.

Complete the following steps:

- (1) Visit the web-based User Interface (UI) and LOGIN, if required.
- (2) Explore the Data Cube Manager > Data Cube Visualization menu to view the location and size of available data cubes. Click on any region to view the datasets and the click on the datasets to view more details.



- (3) Select Tools > General > Custom Mosaic. This will give you a list of data cubes. Pick a cube of interest.



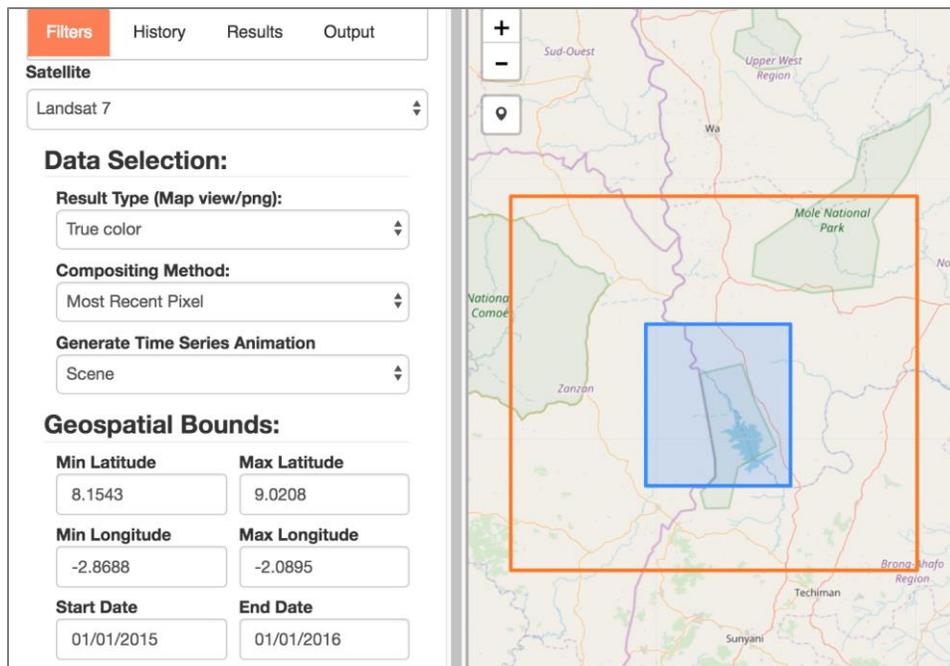
(4) Pick a region for your mosaic by clicking on the top-left corner and then dragging your mouse to the bottom-right corner of the region. You should see a shaded box on your screen. Check the LAT and LON boundaries of your cube and try to keep the region to less than 1deg x 1deg. This will make the execution faster for everyone. As you move around the map, you can also see the LAT-LON position in the upper-right corner of your screen.

(5) Select a Start and End data for your mosaic. Pick ONE year within the range of dates for your cube. The system will default to the range of dates that are available. You can also pick shorter time periods for seasonal mosaics.

(6) Pick a "Result Type". You might start with a "true color" image first, which is the typical Red-Green-Blue (RGB) output. There are also several other false color outputs that can be interesting to identify certain land features. Try a few of them to see the differences in these RGB combinations.

- NIR-SWIR1-SWIR2 or NIR-SWIR1-RED = Land appears in shades of orange and green, ice stands out as a vibrant magenta color, and water appears in blue.
- SWIR1-NIR-RED = Good for vegetation analyses
- SWIR2-NIR-GREEN = This band combination was used for the global Landsat mosaic created by NASA.
- SWIR1-NIR-BLUE = Useful for the monitoring of agricultural crops, which appear as a vibrant green. Bare earth appears as magenta color and non-crop vegetation appears as more subdued shades of green.
- NIR-RED-GREEN = Vegetation is red, with healthier vegetation being more vibrant. This is commonly used when looking at vegetation, crops and wetlands.

(7) Pick a "Compositing Method" which is the basis for building the mosaic. You will see Least/Most Recent Pixel, Min/Max NDVI, Median and Geomedian. For your first mosaic, you should try the Most Recent Pixel, as it will select the most recent cloud-free pixel in the time series. For your second case, you should try the "Median Pixel" as it will select the "middle" value of each band in the time series. We suggest you DO NOT run the Geomedian mosaic, as it takes some time to execute and is more complex than the other choices.

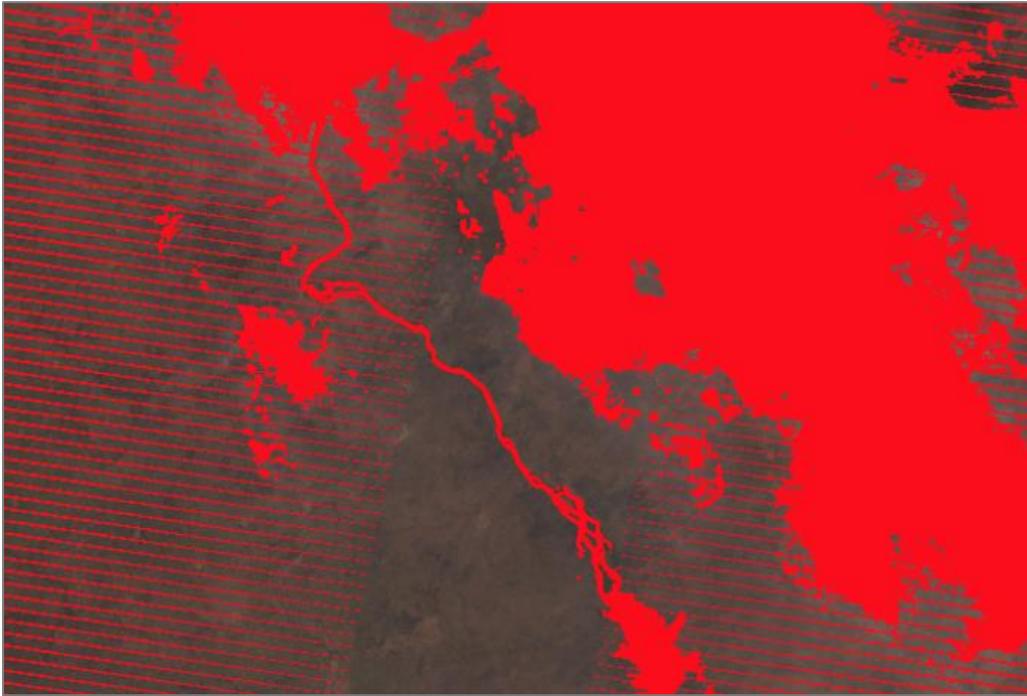


(8) Pick a "Generate Time Series Animation" choice. This will produce an animation file that will allow you to see each of the images in the time series stack which are the basis for the mosaic product. Scene = this will give you a scene by scene look. Cumulative = this will give you a cumulative look at the mosaic as it filters for clouds over the time series. It is suggested to use the SCENE selection as this is a nice way to view the available data for each scene in the time series. NOTE: SCENE is not possible with median mosaics, but should be used for least/most recent pixel mosaics.

(9) Select the "SUBMIT" button when are ready to run your analysis. The "Running Tasks" window will show you the progress of your analysis. It make take a few minutes, so be patient. Once complete, you will see the final mosaic (cloud-filtered) on the main screen.

(10) Select the HISTORY tab on the top-left menu to see a history of your analysis cases. By clicking on any of the listed tasks you can see the details of analysis and load past results into the screen.

(11) Select the RESULTS tab on the top-left menu to see the details of the underlying scenes that were used for the mosaic. You will see an "Acquisition List" that shows details for each of the scenes in the mosaic. For example, you can see the "clean pixel percentage", which is the same as 1-Cloud%. In addition, any one of these individual scenes can be loaded for viewing. Try one! When a scene or mosaic is selected the user can also check the "Highlight No Data" box to color the no-data or clouds as RED in the image. An example is shown on the following page.



(11) Select the OUTPUT tab on the top-left menu to choose a variety of output products. Once a result is selected there will be a drop-down list of possible outputs, such as PNG, animations and GeoTIFF. Try downloading a PNG file (flat image), a GeoTIFF (to view in QGIS or ArcGIS) and a time series animation (GIF file). For each one, you will need to press the "Download Selected" button to download the final result. Be sure to download and view a GIF animation file as it is interesting to see the variations in cloud cover, Landsat-7 banding and scene edges as you go thru the time slices in the data cube.

In addition, when a product is selected you will see a list of details for the mosaic. This includes the run time, the number of scenes in the image, the pixel count and the clean pixel percentage (% cloud free), Lat-Lon range, time range and compositing method. This is valuable information.

(12) Repeat the steps above with variations in the selections. For example, try a different region, time period, false color mosaic output, or compositing method.

=====

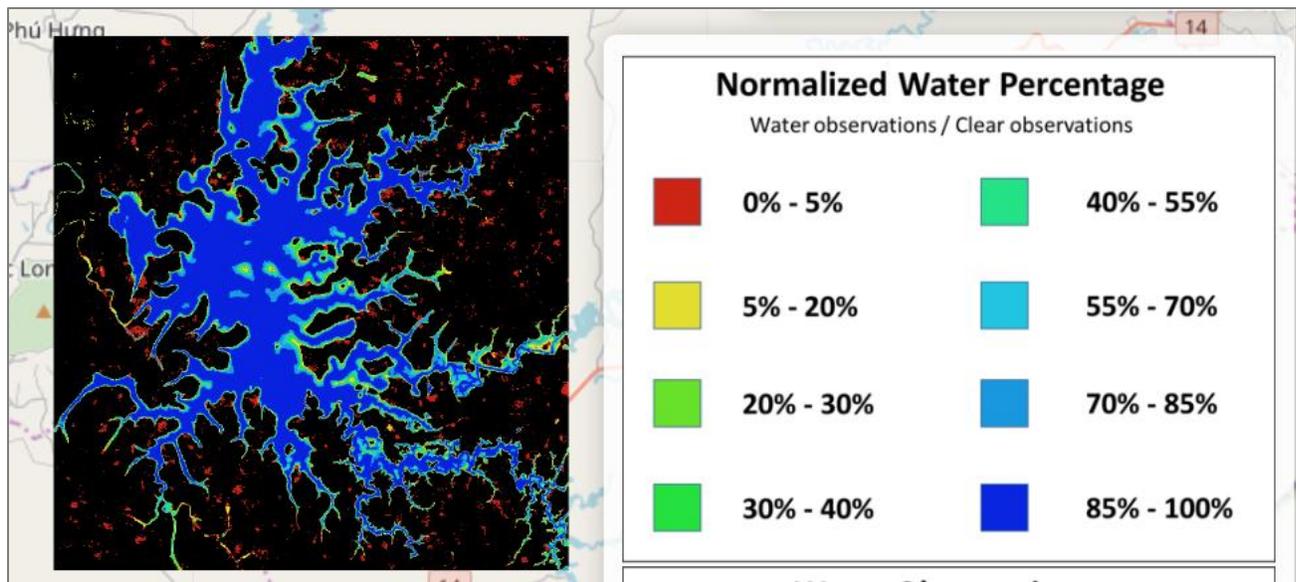
Module #1 (AWS Online User Interface) - Exercise #2 (Water Analyses)

Objective: Create a Landsat water extent product (Australian WOFS algorithm) and evaluate areas of water change due to seasonal flooding or single flood events. Users will then create separate annual water extent products to determine when and where the extreme events occurred and try to correlate those events with known changes in rainfall (droughts or heavy rains using precipitation data). Users will then create a water quality product (Total Suspended Matter - TSM) over an inland water body to determine water quality spatial variability and to assess changes in water quality over time.

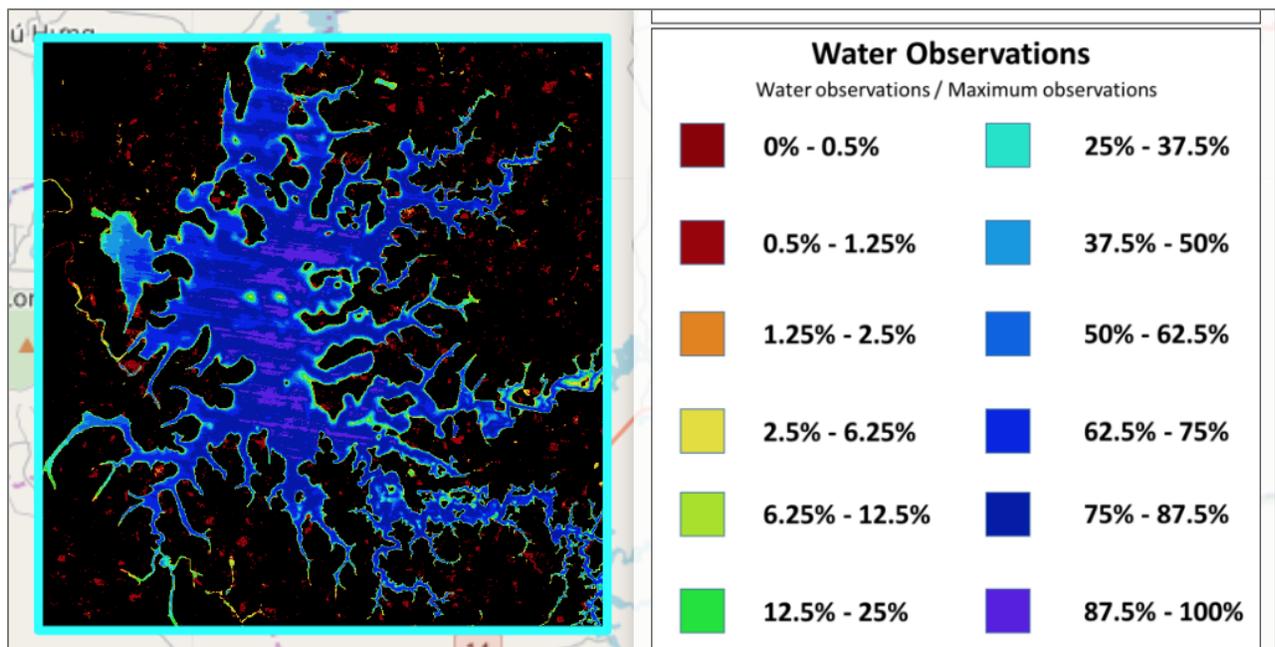
Complete the following steps:

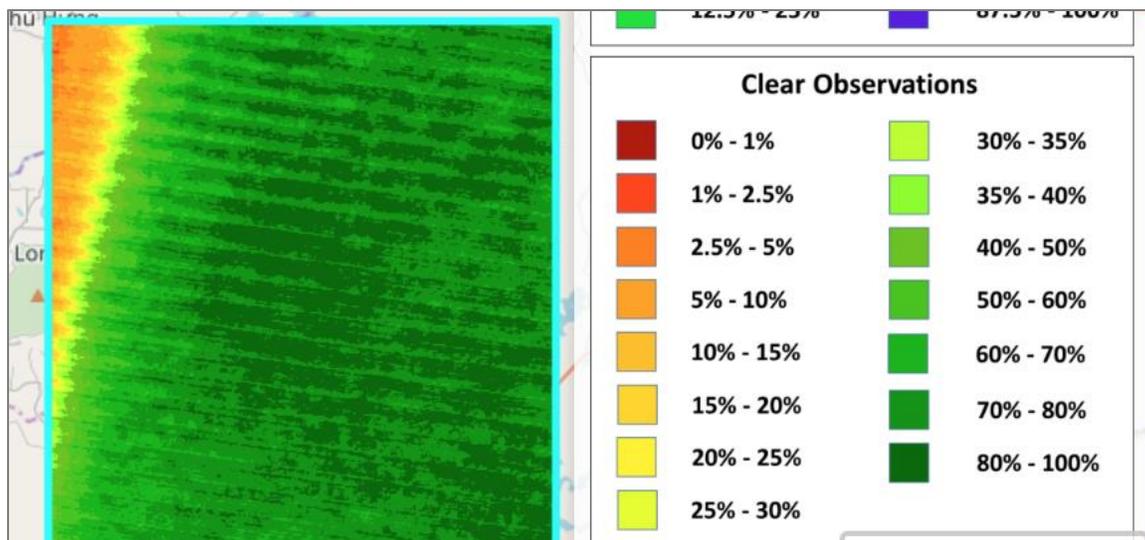
Water Extent (part 1)

- (1) Visit the web-based ODC User Interface and LOGIN, if required.
- (2) Select Tools > Water > Water Detection. This will give you a list of data cubes. Pick a cube of interest.
- (3) Pick a region over an inland water body by clicking on the top-left corner and then dragging your mouse to the bottom-right corner of the region. You should see a shaded box on your screen. Check the LAT and LON boundaries of your cube and try to keep the region to less than 1deg x 1deg. This will make the execution faster for everyone. As you move around the map, you can also see the LAT-LON position in the upper-right corner of your screen.
- (4) Select a Start and End data for your mosaic. Pick 10+ years for your analysis or use the entire time range of the data cube.
- (5) In the "Data Selection" area, select the image background color (BLACK is best for viewing the data). DO NOT select a time series animation at this point. We will do this later in Step #9.
- (6) Select the "SUBMIT" button when are ready to run your analysis. The "Running Tasks" window will show you the progress of your analysis. It make take a few minutes, so be patient. Once complete, you will see the final product (normalized water percentage) on the main screen.
- (7) Review the output product and ZOOM in for more details. In the top-right of the screen is a small button that looks like a water droplet. Click this for the legend. The first legend is the one used for the default screen output. An example is below. The product shows the percent water observations versus the number of clear observations for each pixel. RED regions are very infrequent water observations where water may have existed for a very short time in the time series. DARK BLUE regions are water for the majority (85-100%) of the time series. It is interesting to see how water changes along the edges of inland lakes or coastlines, how water exists in low lying areas during rainy seasons or storm events and the variability of water over time.

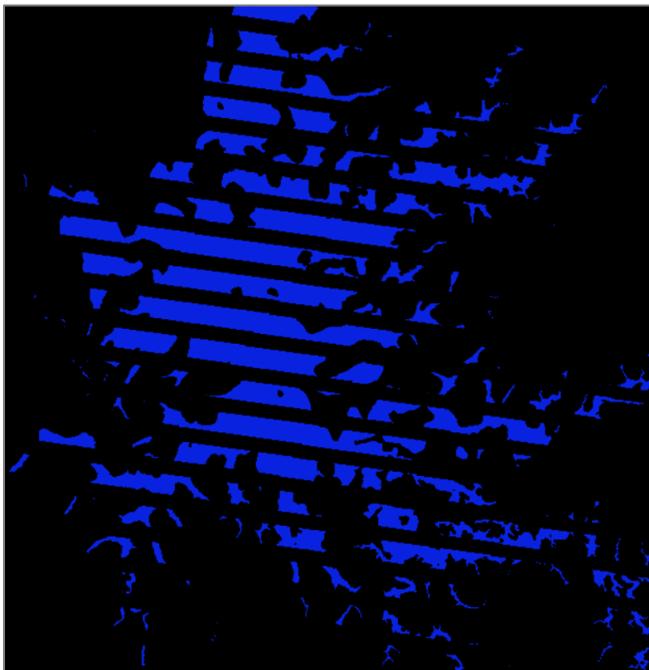


(8) Select the RESULTS tab and the analysis product to view other types of output. You will be able to select 2 other products ... Water Observations / Maximum Observations and the % Clear Observations. For the example above, here are those products with the legend on the right. You will see that some results are skewed by the number of clear observations due to scene positioning or available data.



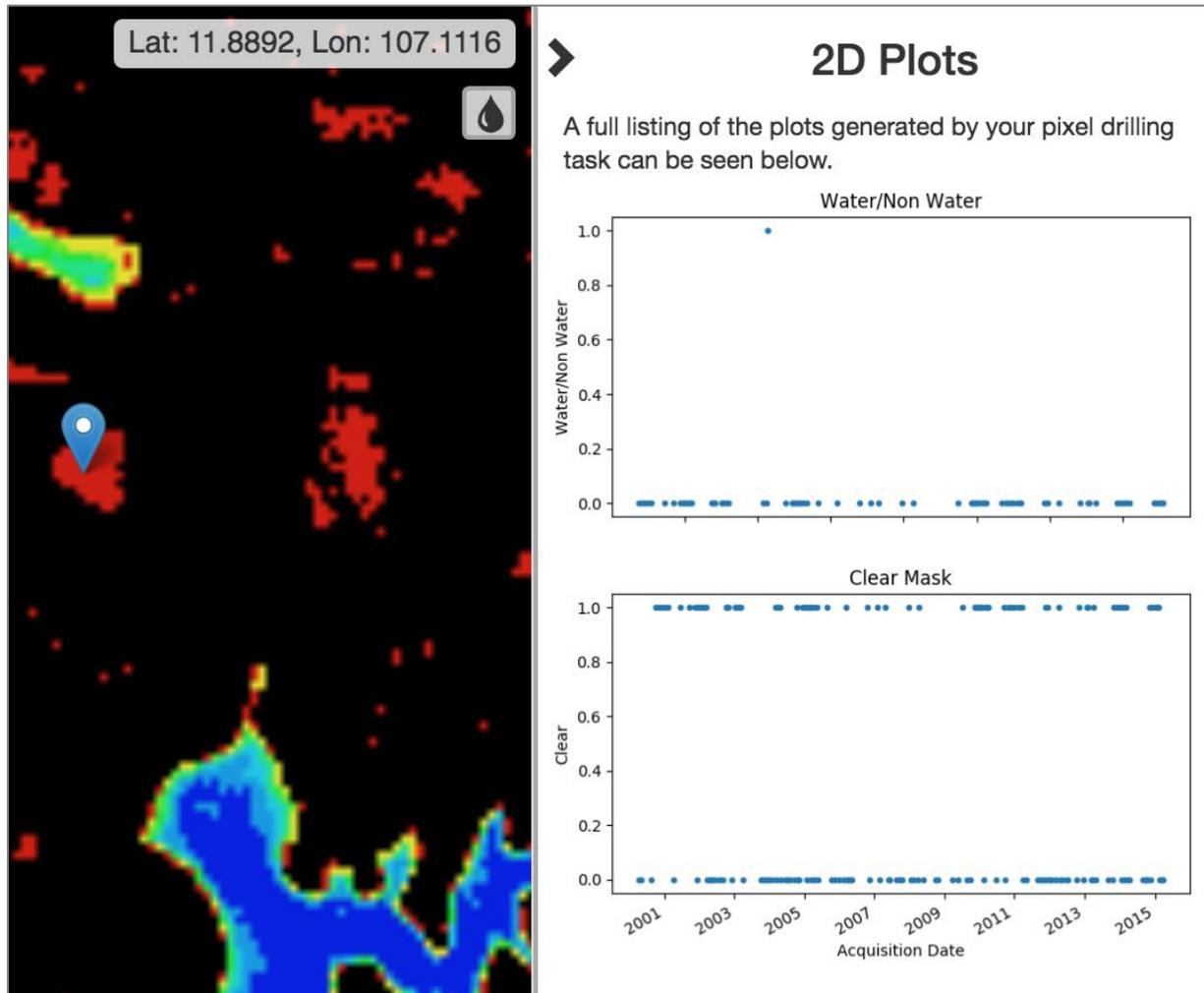


Using the same RESULTS tab, it is also possible to see individual scenes to evaluate the extent of water at any specific time. Be sure to view the clean pixel % as many scenes could be covered by clouds. Below is a 57% clean pixel image from Feb-2015. Water extent is shown in one color ... BLUE. In this example you can also see the Landsat-7 "banding" that causes missing data across the scene.



(9) Run another case that produces a time series animation with "Scene" selected. You may want to choose another region, but make it small, since the animation file takes more time to generate than the prior cases. There are several time series animation products. Scene = water extent for each time slice, Cumulative Percentages = cumulative water extent over time, Cumulative Observations = cumulative number of clear observations for a each pixel. The most common selection is the "Scene" animation. The animation product must be downloaded by selecting the OUTPUT tab, selecting the output case, selecting "Water Animation" and then selecting the "Download Selected" button. The product will be an animated GIF file. These animations are quite interesting for viewing water extent, cloud cover variation, and Landsat "banding" issues.

(10) Use the "Marker Tool" to look at any single pixel location. This tool is located in the top-left of the map and looks like a small "marker". After the result has been generated in Step #6, you can click the marker tool and then move the marker to any point in the figure. A 2-D plot will be generated to show the time series water / non-water results for the selected pixel. This will allow you to view the impact of clouds (clear mask graph) and the time series variation of water existence. Try this for a RED area on your output image, as this area would experience times of water and non-water. The 2-D result will allow you to determine the exact times when the water existed. The example below has placed the marker on a RED area in the image. The top plot shows that water existed ONE time in the 15 year time series and it appears to be in 2004. The bottom plot shows the variations cloud vs clear over this same location.



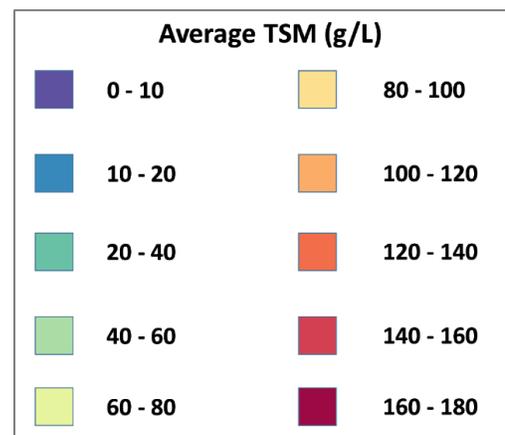
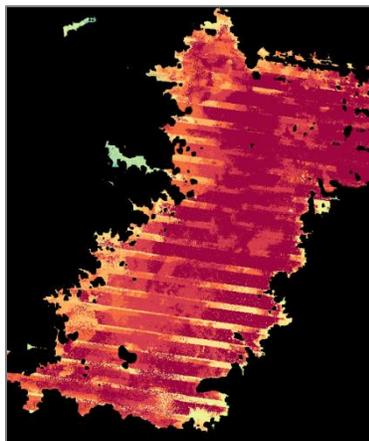
(11) Use these products and analysis tools to review several areas and try to find areas where there has been single flood events, areas that have experienced seasonal flooding due to a rainy season, or water boundaries that have experienced change over time.

Water Quality (part 2)

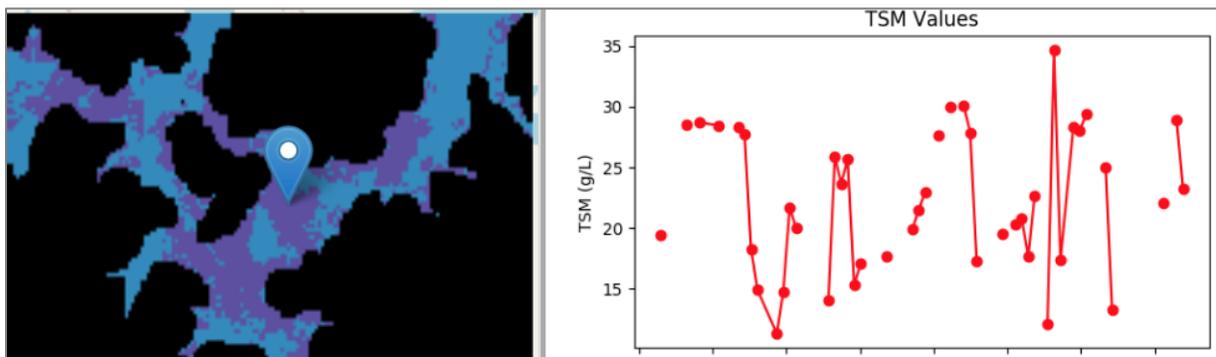
(12) Select Tools > Water > Water Quality TSM. Select a cube and then select an inland water body where you might think the water is used for drinking water supplies. Similar to above, select a region and time period.

This analysis will use a simple single-band algorithm to predict Total Suspended Matter (TSM) in the water. It is believed that TSM is a proxy for water quality and represents the "cleanliness" or "turbidity" of the water. Though many scientists will agree that Landsat cannot accurately measure TSM, there is value in these results when viewing spatial differences in TSM and time series variations in TSM. In general, Landsat-8 is better than Landsat-7 for water quality, as the signal-to-noise is better for L8.

(13) Select a "Result Type" such as "Maximum TSM" or "TSM Variability" and run an analysis for a long time period. A sample output is shown below (Maximum TSM) for a lake in Vietnam.



(14) Similar to Step #10, one can view a single pixel to see the variability in TSM values over time. Below is an example for a water reservoir in Uruguay using Landsat-8 from 2013 thru 2016 (4 years). The variability can be easily seen with the 2-D pixel level plot. In this case it appears there is seasonal variability in water quality as well as overall changes in water quality (year to year). Try this same analysis for a water body in your region and use this information to determine how you might take water quality samples or monitor the impact of a water management change.



Module 2

Jupyter Python Notebooks

ARDC Training Workshop

Module #2 (Jupyter Notebooks)

Each country team will be given a separate website address and password for their EC2 instance. This instance will contain a folder of Jupyter Notebooks for this training module.

Objective: Demonstrate the use of Jupyter Python Notebooks to create application products and evaluate those results. This session will be broken into 6 separate tasks and will test the following application algorithms:

- (A) Cloud-free Mosaics and K-means Clustering ... land classification
- (B) WOFS and TSM ... water extent and water quality variations in space and time
- (C) Fractional Cover and NDBI/NDVI/NDWI ... urbanization, vegetation and water extent
- (D) PyCCD or NDVI Trend ... land change, compare with Google Earth or GFW
- (E) Transect Analysis ... 2D and 3D plotting of bands and products
- (F) Data Export ... data export for interfacing externally with QGIS and EXCEL

DEMO NOTEBOOKS

- (A) Training_TaskA_Mosaics
- (B) Training_TaskB_Water
- (C) Training_TaskC_Indices
- (D) Training_TaskD_LandChange
- (E) Training_TaskE_Transect
- (F) Training_TaskF_DataExport

=====

General: How to use Jupyter Python Notebooks

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live Python code, equations, visualizations and narrative text. These notebooks contain the core application algorithms of the Open Data Cube and allow customization of any application.

- When you arrive at the website, you will find a list of Python Notebooks under the "Files" tab. To open any notebook, just click on the filename.
- You can make a copy of any notebook by using the "File > Make a Copy" menu selection. The new file will contain "-Copy1" at the end of the filename. You can rename the file by clicking on the title at the top of the screen and then editing the filename.
- You will find two types of "cells" in the Data Cube notebooks. These can be found under the "Cell > Cell Type" menu. A cell used for text or comment is called "Markdown" format. A cell used for Python code is called "Code" format. When you want to add a new cell in the notebook, be sure you use the correct cell type. To add new cells you will use the INSERT menu item. To Cut/Copy/Delete cells, you will use the EDIT menu.
- There are several ways to "run" the notebook code. To run the entire script (starting from the top), you can select "Kernel > Restart & Run All". Once the code has been executed (top to

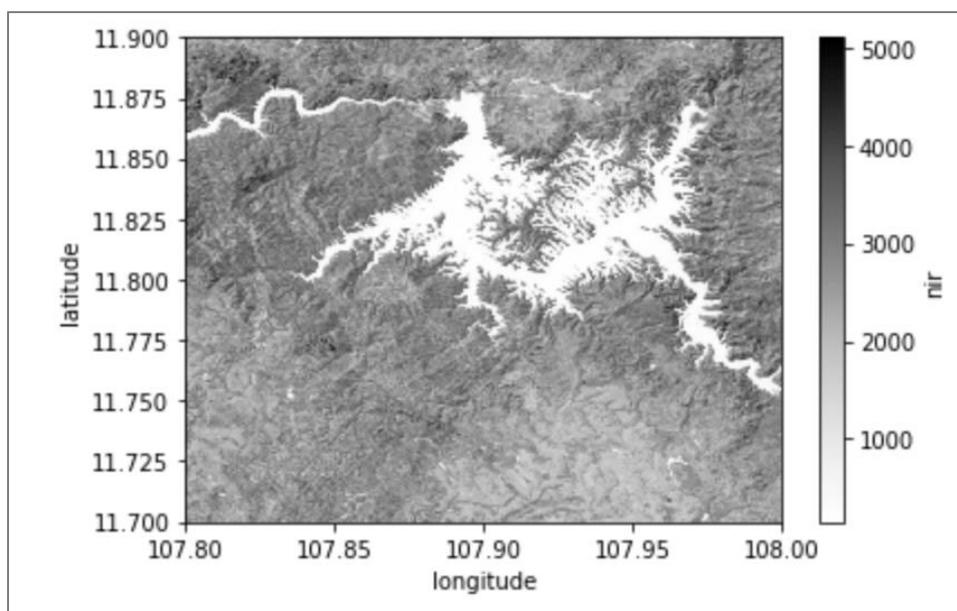
bottom) you can change individual cell content and rerun portions of the code by going to any cell and hitting "Shift - Enter". You will notice this approach will renumber the code blocks starting with the last number that was executed. So, it may be confusing. To reset the numbering (1 to xxx), just run the entire script, as suggested above.

- When the code is "running" you will notice the cell blocks will look like "In [*]". The "star" means the code is executing. When the cell is done executing the "star" will turn into a sequential number, starting with the last executed block number. You will see that some blocks run very fast, and others take some time. If you run the entire stack, you can scroll to the top and see the code execute along the way as it creates output as it moves along the blocks.
- Most of the code blocks have comment blocks directly above the code blocks. By clicking on any cell, it will allow you to edit the cell. To rerun the code changes, just click "Shift - Enter". You will notice that the "#" symbol is used to make any line a comment and it is not executed.
- As the code is executed, you will occasionally see some "pink" warnings. In most cases these are only warnings and do not stop the execution. If you want to stop the execution at any time, just select "Kernel > Interrupt".
- If your code gets "hung up" and does not appear to be executing you can go back to the main Jupyter Notebook page and select the "Running" tab to view which notebooks are being "executed". In some cases these notebooks are actually running, but in other cases they are just "open" and sitting in the memory and ready for editing or running. You can "Shutdown" any notebook from this screen.
- It should be noted that most of the notebook algorithms are integrated into the online user interface tool. The advantage of using notebooks is that you can view the code and have more flexibility in creating your own products.

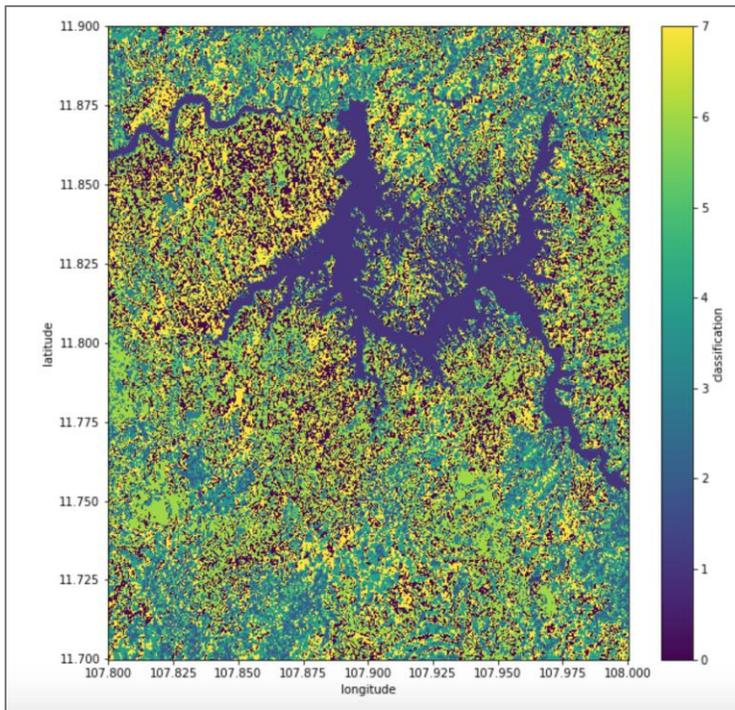
Task-A: Cloud-free Mosaics and K-means Clustering

Objective: This notebook will demonstrate the creation of a cloud-free mosaic and then use that mosaic to create a clustering product for land classification. Clustering is an approach that groups pixels according to their spectral similarity. Land classes (e.g. water, bare soil, vegetation, trees) can be identified using these cluster groups if one assigns the clusters to land types. We call this "supervised" classification.

- When you arrive at the website, you will find a list of Python Notebooks under the "Files" tab with the prefix "Training". There are meant to be backup files. You will also find notebooks with the prefix of your last name. These are the notebooks you should run and modify for the training session. If you do not find notebooks with your last name as the prefix, please ask the training team for help.
- Open the notebook with the following name " *LastName_TaskA_Mosaics*". Review the entire notebook and scan the content. You will see the sample case is set up for a Vietnam data cube. Modify the cell that connects to data cubes and choose a cube of your choice. You will see that many of the selections are "commented out", so you just need to keep one line "active".
- Define a "region of interest" for your analysis. Keep this region small (less than 0.2-deg x 0.2-deg) so that it runs fast.
- Once ready, run the entire script by selecting "Kernel > Restart & Run All". You will be able to scroll to the top of the code and watch the execution.
- Once complete, review your analyses. If you have questions, ask the training team for help.
- Continue running new cases where you change the location of the analysis, the single scene (acquisition number), and number of clusters. You may want to compare your clustering results with known land types (e.g. water, trees, agriculture fields) to see if the results look accurate.



Example Output: Cloud-free Median Mosaic - Grey-scale NIR



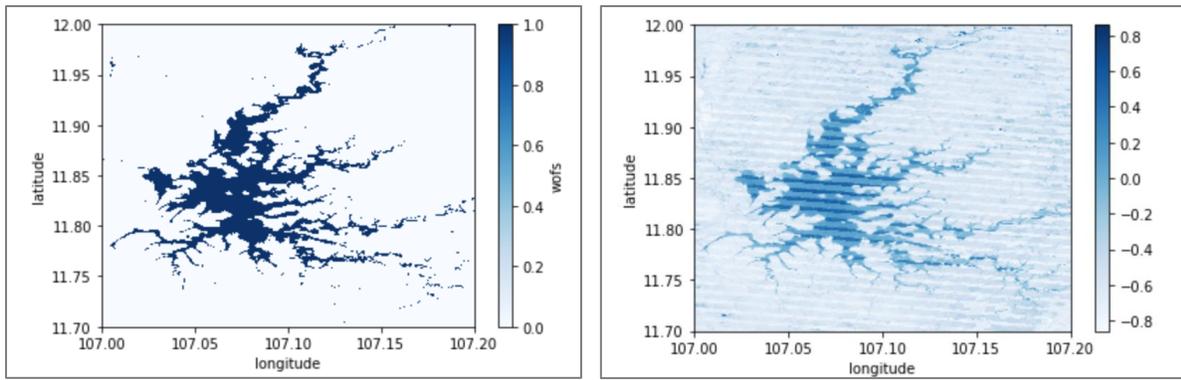
Example Output: 8-class clustering

=====

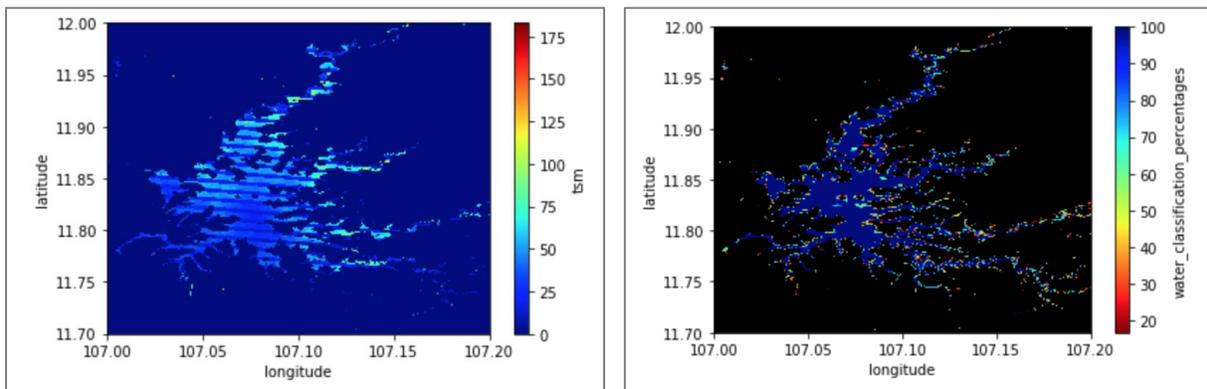
Task-B: Water Extent (WOFS) and Water Quality (TSM)

Objective: This notebook will demonstrate the creation of a time-series water extent product based on the Australian Water Observation from Space (WOFS) algorithm and a water quality product based on the Australian Total Suspended Matter (TSM) algorithm. The time series product displays a percentage of the number of times water was detected divided by the number of clear (non cloudy) views. The water quality product (TSM) displays the amount of sediment in the water which is a proxy for water quality. This algorithm is merely an approximation and not very precise but it can help detect spatial and temporal variations in a water body. We will also look at another water extent product called NDWI (Normalized Difference Water Index) to view the water extent in individual data cube layers.

- When you arrive at the website, you will find a list of Python Notebooks under the "Files" tab with the prefix "Training". There are meant to be backup files. You will also find notebooks with the prefix of your last name. These are the notebooks you should run and modify for the training session. If you do not find notebooks with your last name as the prefix, please ask the training team for help.
- Open the notebook with the following name " *LastName_TaskB_Water*". Review the entire notebook and scan the content. You will see the sample case is set up for a Vietnam data cube. Modify the cell that connects to data cubes and choose a cube of your choice. You will see that many of the selections are "commented out", so you just need to keep one line "active".
- Define a "region of interest" for your analysis. Keep this region small (less than 0.2-deg x 0.2-deg) so that it runs fast. Be sure that your region is over an inland water body (lake).
- Define a short time period (a few months or up to a year). We will use this to create a cloud-free most-recent pixel mosaic and then look for water.
- You will see results for water detection using 2 algorithms (WOFS and NDWI) and you will a result for the water quality (TSM). Review the outputs and see if the water is evident. How do the WOFS and NDWI results compare? Can you see "banding" in the NDWI results? Is there spatial variation in your TSM output? Do there appear to be clouds in your result?
- Rerun the analysis for new water bodies. Also consider shorter time periods (monthly) to look at variations of the water for each month in a year. Remember that you can change a cell and then rerun any cell by clicking Shift + Return. It is NOT always necessary to run the case from the beginning if you are only changing a small portion of the code. The cube will remain in "memory".
- The WOFS time series results will be shown at the end. Remember that this represents the percent of time that any pixel has been water over the full time period. If the time period is short (as we used above) then there may not be many spatial and temporal variations in the water extent. Try changing the time extents to 5 years and review the results. It is always interesting to see the areas where there has been water at very infrequent times (red).
- Below the WOFS result is an X-Y plot of a single pixel showing the WOFS result over the time series. You can see when the water existed (value=1) and when there was no water (value=0). Change the Lat-Lon position within your region and then review the results. Can you locate an area of RED (infrequent water) in your region? When was the water present? You may want to change the region to a much smaller area and then rerun the entire analysis. Can you find another location with seasonal water variations? Look for yellow or orange colors, find the location and then use the X-Y plot to view the results.



Example Output: Water extent (WOFs-left) and (NDWI-right)



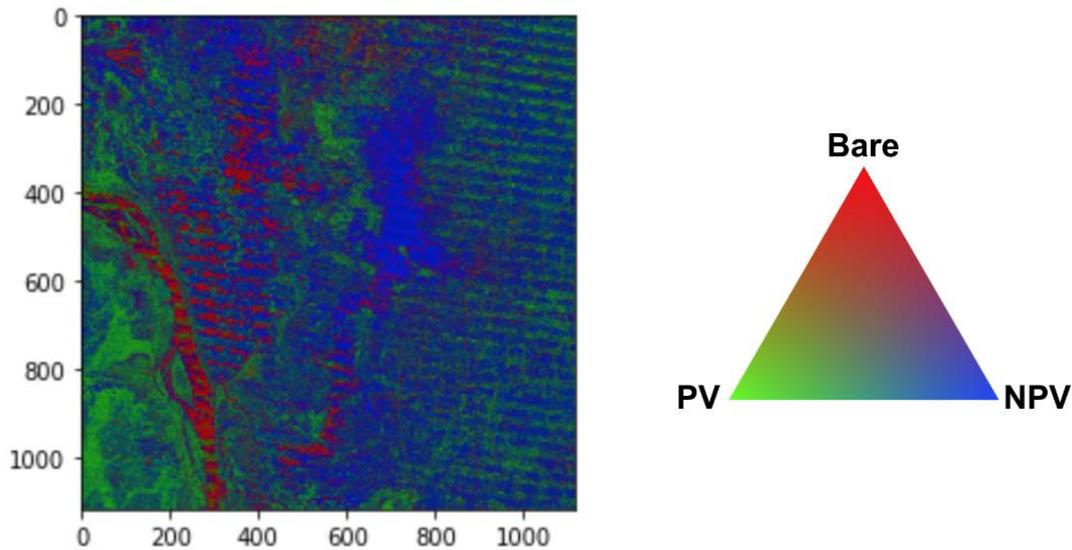
Example Output: TSM Water Quality (left) and WOFs Time Series Water Extent (right)

=====

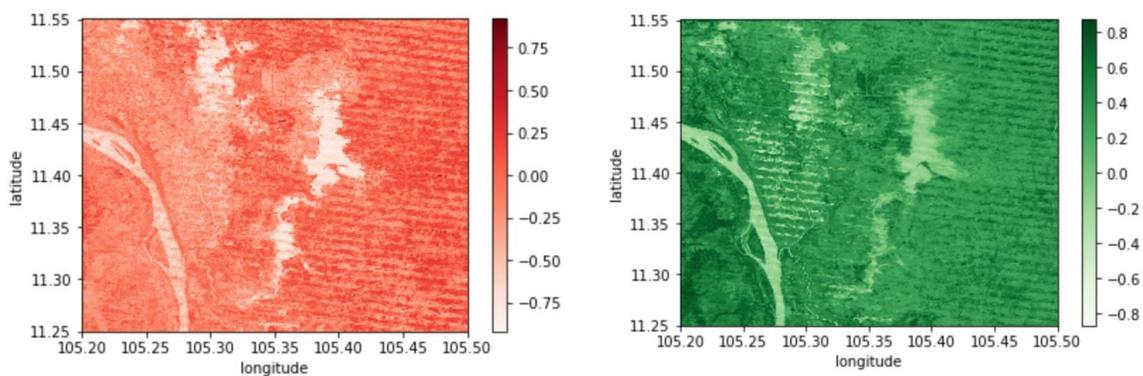
Task-C: Fractional Cover (FC) and Spectral Indices (NDBI and NDVI)

Objective: This notebook will demonstrate the creation of products for vegetation and urbanization. The Fractional Cover (FC) algorithm was developed by Juan Gerschmann (CSIRO) and is used for land cover type estimation (vegetation, non-green vegetation, bare soil) of each pixel. The algorithm iterates between these 3 land types using a median mosaic (see figure). In addition, there are 2 other simple spectral indices that show urbanization and vegetation. These are the Normalized Difference Buildup Index (NDBI) and the Normalized Difference Vegetation Index (NDVI).

- When you arrive at the website, you will find a list of Python Notebooks under the "Files" tab with the prefix "Training". There are meant to be backup files. You will also find notebooks with the prefix of your last name. These are the notebooks you should run and modify for the training session. If you do not find notebooks with your last name as the prefix, please ask the training team for help.
- Open the notebook with the following name" "*LastName_TaskC_Indices*". Review the entire notebook and scan the content. You will see the sample case is set up for a Vietnam data cube. Modify the cell that connects to data cubes and choose a cube of your choice. You will see that many of the selections are "commented out", so you just need to keep one line "active".
- Define a "region of interest" for your analysis. Keep this region small (less than 0.2-deg x 0.2-deg) so that it runs fast. You may want to select a region that is over an urban area or a vegetation area. It is even better if you can find an area with diversity ... urban, vegetation and water.
- Define a short time period (a few months or up to a year). We will use this to create a cloud-free median mosaic and then produce the various products.
- Review the results from the FC product. Can you see water in your result and what color is the water? Can you see difference between water and bare soil? Can you see areas of active green vegetation and other areas of non-green vegetation?
- Review the Urbanization (NDBI) product. Can you see areas of high urbanization where there would be buildings and roads? How are they shown in the product? Are these areas low values or high values? You will typically find that "build up" or barren areas have NDBI values > 0.
- Review the Vegetation (NDVI) product. Can you see areas of high vegetation? If you review the online literature you will find that NDVI of 0.6 to 0.9 is typically dense vegetation (forest) and NDVI of 0.2 to 0.5 is shrubs or agriculture. Areas of negative NDVI are typically bare soil, water, NPV or urbanized land.

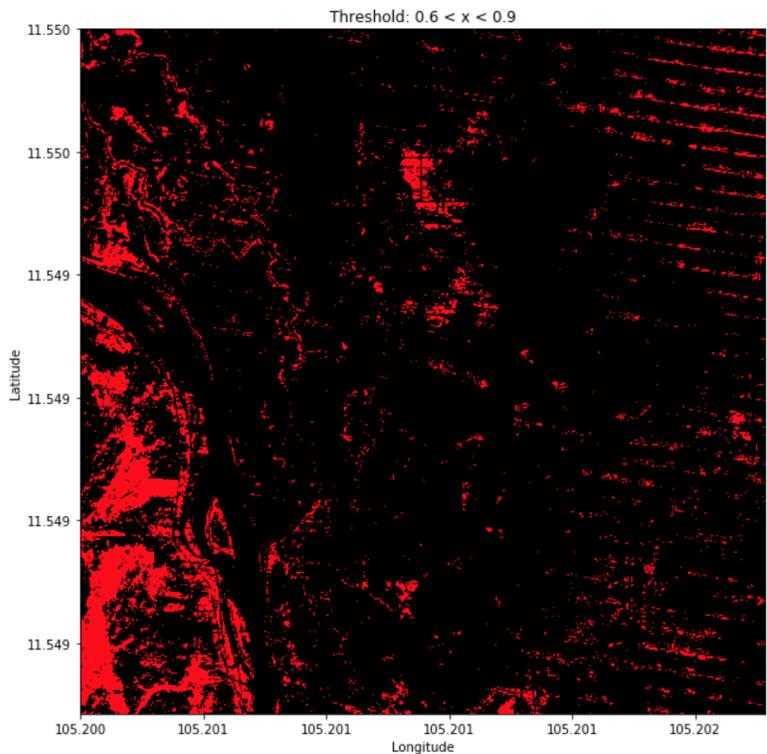


Example Output: Fractional Cover (FC)

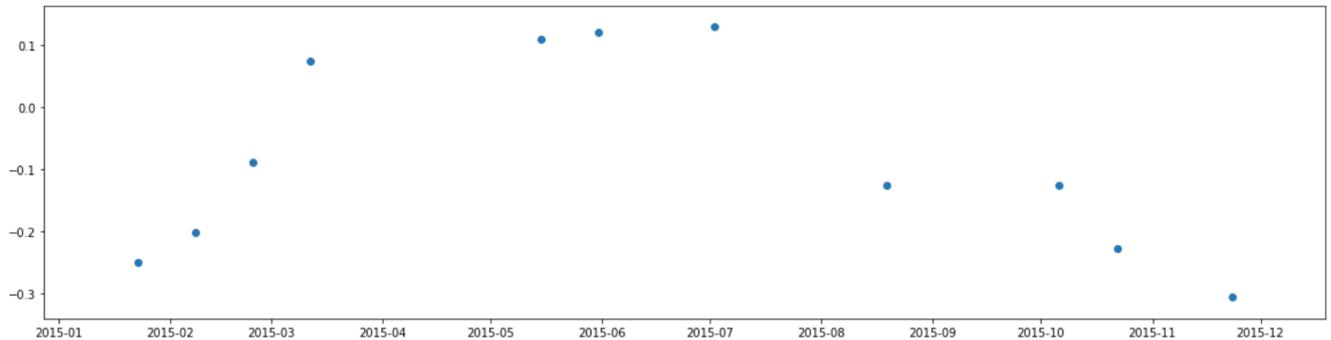


Example Output: NDBI (urbanization-left) and NDVI (vegetation-right)

- Now you will create a "threshold" plot that will define a region (minimum and maximum) and then highlight the pixels that are within this region. Choose a range of 0.6 to 0.9 (typical of forests) and then view the threshold plot. An example output is shown below. What did you find? Can you see forests (RED data within the threshold)? Can you see other features, such as the Landsat "banding"? Try changing the threshold range and also changing the index type. Try changing the analysis to view a single time slice. This is done by setting the time slice ($t=0$ for the first time layer). Can you see changes between time slices? Don't forget that clouds can have a big impact on these single slice images. Can you see where the clouds are located? Now you know why cloud-filtered mosaics are so important.



- Below the NDVI result is an X-Y plot of a single pixel showing the NDVI result over the time series. Try to select a location that coincident with grassland vegetation (NDVI = 0.2 to 0.5). When you view the results can you see the seasonal variations in NDVI? Why is the graph not consistent and smooth? Why are there missing data points? An example one-year plot is shown below.

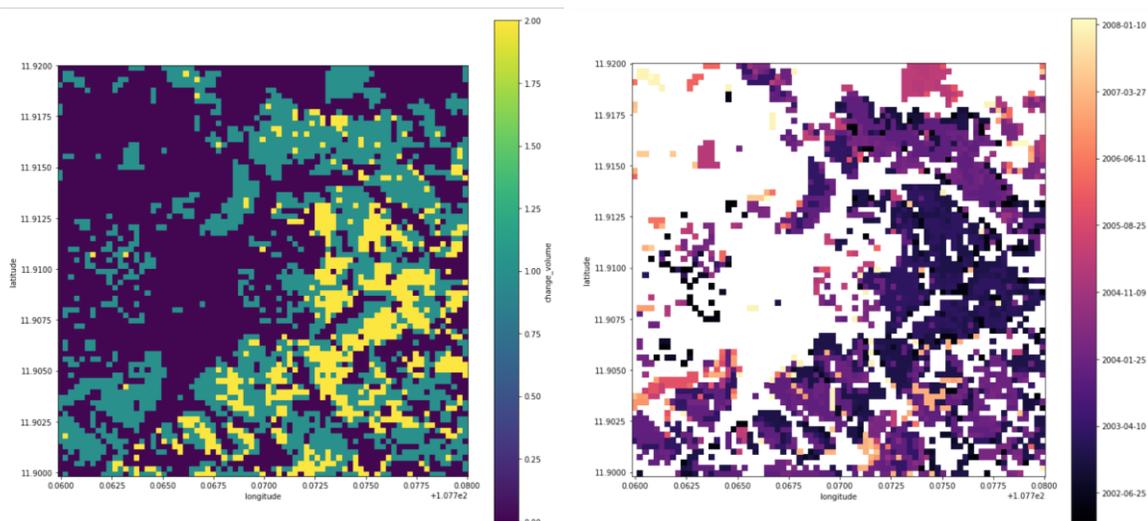


- Add a new index to the workbook. You will insert new cells (using Insert > Insert Cell Above/Below) to add new lines to the notebook. If you desire to add text to any cell, it is best to change the "cell type" using Cell > Cell Type > Markdown. You will create a new index called EVI = Enhanced Vegetation Index, which is an "optimized" vegetation index designed to enhance the vegetation signal with improved sensitivity in high biomass regions. The formula is: $EVI = 2.5 * (NIR - Red) / (NIR + 6 * Red - 7.5 * Blue + 1)$. Plot the results using red-shading and compare with NDVI. Do you see any differences?

Task-D: Land Change

Objective: This notebook will demonstrate the detection of land change using several approaches. This analysis will compare the results of the PyCCD (Python Continuous Change Detection) algorithm and an NDVI Trend algorithm (by Vogelmann). The user will also analyze the details of individual scenes to validate (visually) the time and location of these changes.

- When you arrive at the website, you will find a list of Python Notebooks under the "Files" tab with the prefix "Training". There are meant to be backup files. You will also find notebooks with the prefix of your last name. These are the notebooks you should run and modify for the training session. If you do not find notebooks with your last name as the prefix, please ask the training team for help.
- Open the notebook with the following name "LastName_TaskD_LandChange". Review the entire notebook and scan the content. You will see the sample case is set up for a Vietnam data cube. Modify the cell that connects to data cubes and choose a cube of your choice. You will see that many of the selections are "commented out", so you just need to keep one line "active".
- Define a "region of interest" for your analysis. Keep this region VERY small (less than 0.02-deg x 0.02-deg) so that it runs fast. You may want to select a region that has experienced known land change, such as deforestation or urbanization. Your region should be less than 100 x 100 pixels.
- Define a long time period (10+ years). We will first use the PyCCD algorithm to compute the number of land changes for each pixel over the time series. This is accomplished by creating a "curve fit" of the spectral bands and then detecting when the spectral response changes significantly from this seasonal variation.
- You will see 2 products (samples below). The Change Volume is the number of times the land has changed types for each pixel over the time series. The Change Time is the date of the first land change in the time series. Reviewing the output of these two products is very helpful to determine the extent of land change and when and where it occurred. Take a close look at your results ... can you identify where changes occurred, the extent of this change, and when it happened?

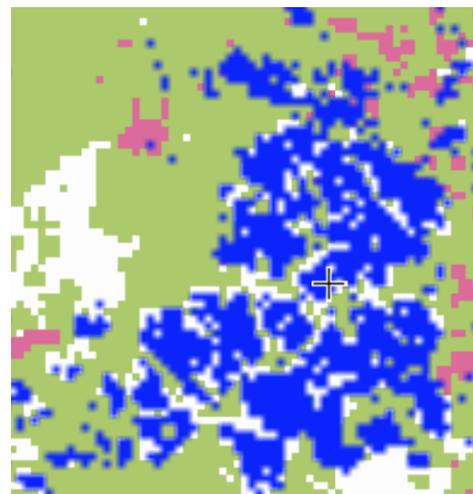


Example Output: Change Volume (left) and Change Time (right)

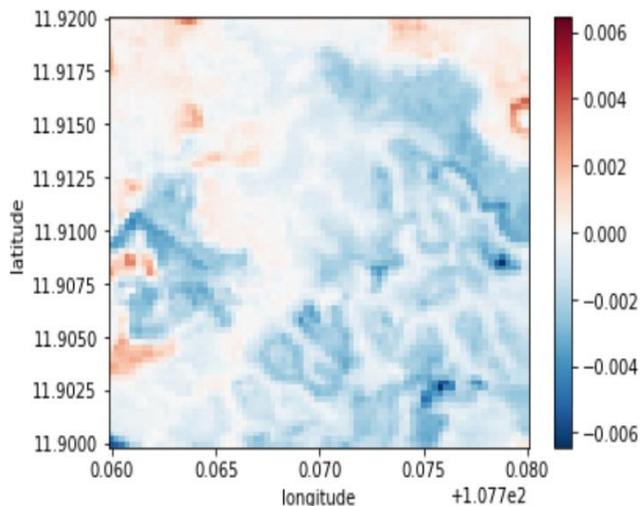
- If you selected an area with expected forest change, you can compare your results against the Global Forest Watch (GFW) website.
- Visit: <https://www.globalforestwatch.org/map/> and use the time slider to match your analysis period.

When you zoom into your image area, do you see the same results? Why is there a difference? See the example on the right and compare it to the sample images above.

Example Output (right): Global Forest Watch analysis (forest loss=red, forest gain=blue)



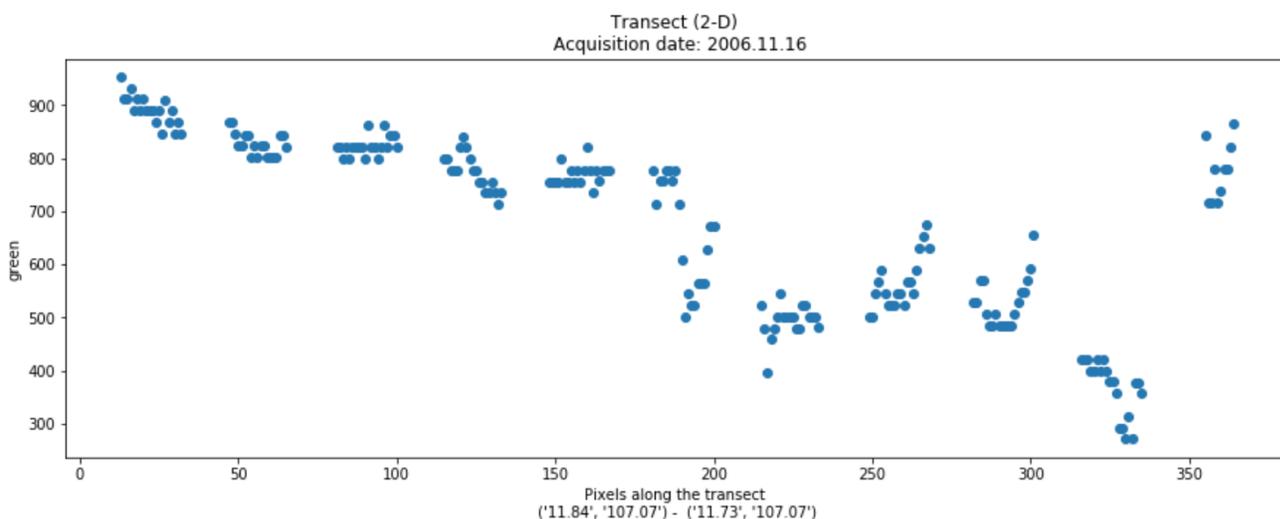
- Next, you will look at individual scenes and view the land changes, visually, using false color mosaics. Under the "Validating Change" section you will see code that will allow you to select 2 specific scenes, generate a custom RGB output, and then compare those results. Do you see the land change between an early and late image in the time series? Does there appear to be more or less vegetation in the change areas?
- You can change the RGB images to anything you desire to view different kinds of output. For example, here are some common combinations of RGB bands that yield interesting results. Try them out and see which ones you like.
 - RGB = red, green, blue = true color image (easy to see clouds, but a bit dark in color)
 - RGB = swir1, nir, red = good for vegetation
 - RGB = swir2, nir, green = good for vegetation. NASA uses this for its global mosaic.
 - RGB = nir, red, green = vegetation is red, with healthier vegetation being more vibrant.
- Finally, look at the output results from the NDVI Trend algorithm by Jim Vogelmann (Forests, 2017). This change detection algorithm is based on a regression analysis of simple NDVI to identify increased or decreased NDVI trend (slope) in the time series. An example output is shown below. You will see that the results show increased vegetation in RED and decreased vegetation (e.g. deforestation) in BLUE. The patterns should be quite similar to the PyCCD results.



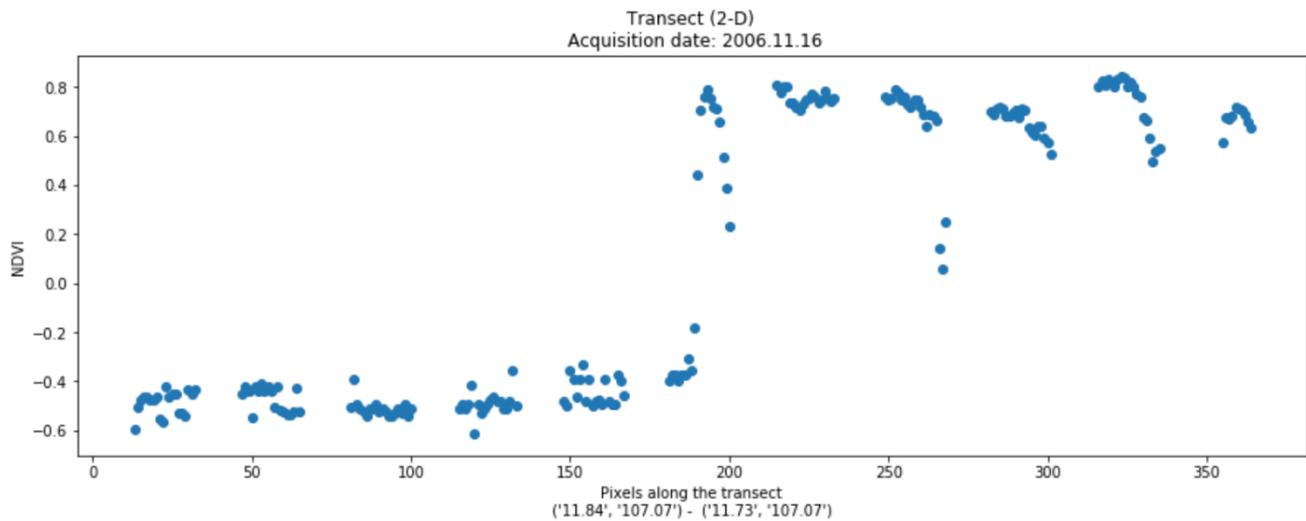
Task-E: Transect Analyses

Objective: This notebook will demonstrate 2D transect plots and 3D Hovmoller plots. We will run these for NDVI (land) and Water extent to show the spatial and temporal variation of data along a line (transect) for a given time slice and for the entire time series. These analyses show the power of data cubes and time series variations.

- When you arrive at the website, you will find a list of Python Notebooks under the "Files" tab with the prefix "Training". There are meant to be backup files. You will also find notebooks with the prefix of your last name. These are the notebooks you should run and modify for the training session. If you do not find notebooks with your last name as the prefix, please ask the training team for help.
- Open the notebook with the following name " *LastName_TaskE_Transect*". Review the entire notebook and scan the content. You will see the sample case is set up for a Vietnam data cube. Modify the cell that connects to data cubes and choose a cube of your choice. You will see that many of the selections are "commented out", so you just need to keep one line "active".
- Define a "region of interest" for your analysis. Keep this region small (less than 0.2-deg x 0.2-deg) so that it runs fast. Define a time period of ~10 years.
- Define a transect line that will run across our region of interest. Use the display map above to find the end points of your desired line. If you click on the map it will give you precise Lat-Lon positions for a point. You will want to find a line across water and vegetation so that you can see the difference between vegetation (NDVI) and water in the products.
- Select an acquisition (time slice) number in your time series. Remember that the time series will go from t=0 to t=time, where the maximum value is found in Block-11 of the XARRAY output.
- Select an XARRAY parameter for plotting. This can be one of the Landsat bands (e.g. red, green, nir, swir1). Review the plot results for your selected band and the NDVI product. Can you tell the difference between the land and water pixels on your transect? Can you see differences between the band products and the NDVI product? Can you tell where there are clouds in the time series? See the examples below.

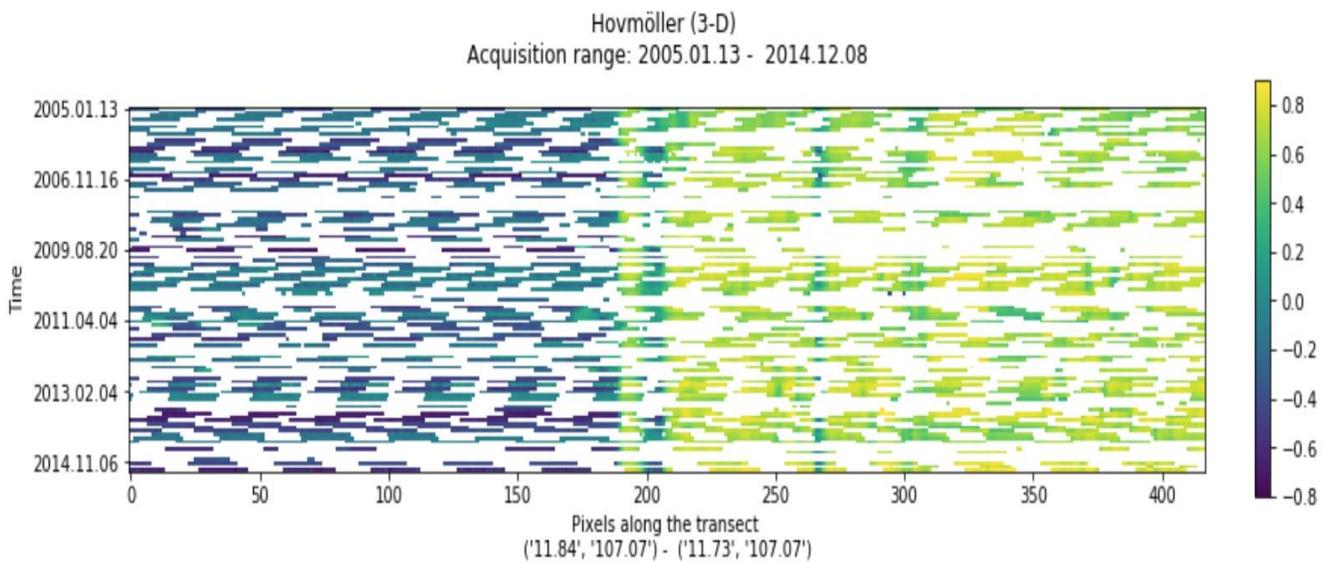


Example Output: 2D Transect Plot of the Green band for a single acquisition date



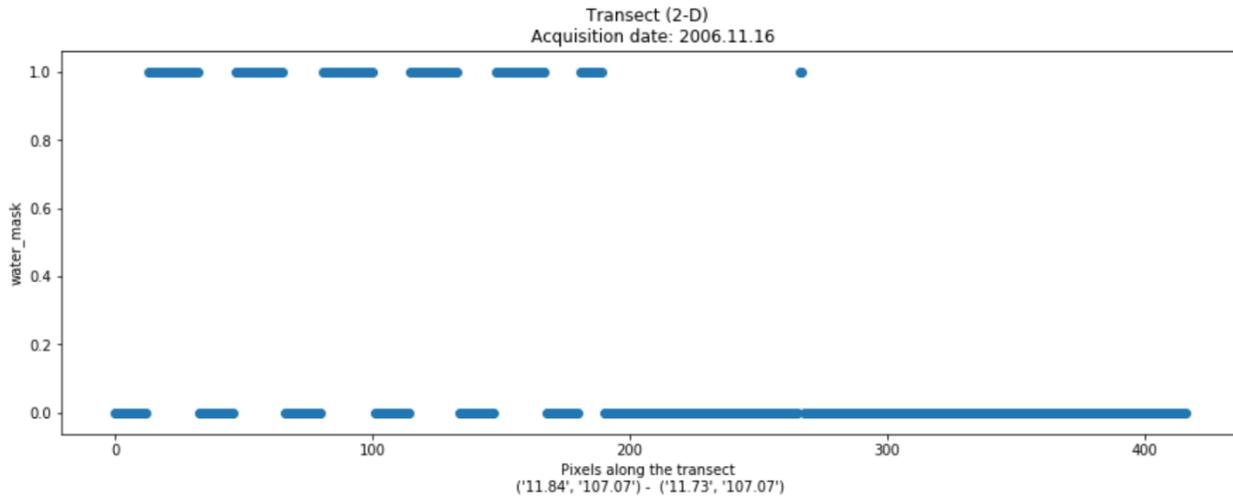
Example Output: 2D Transect Plot of NDVI for a single acquisition date

- Review the 3D Hovmöller plot of NDVI. What can you see in this product? Can you see the impact of clouds? Can you see any changes in the land or water over time? Can you see spatial differences along the transect? See the example below.

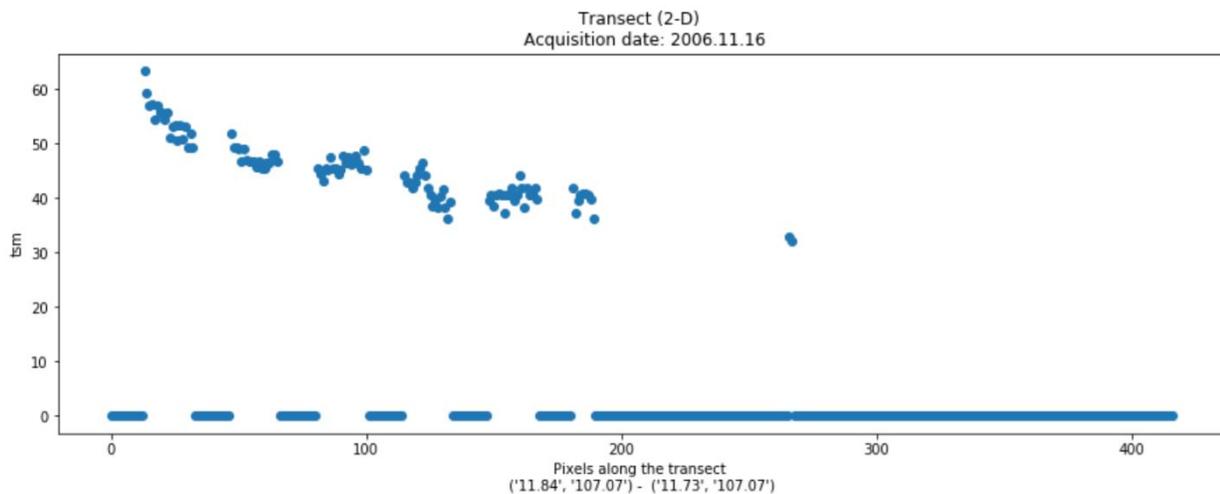


Example Output: 3D Hovmöller Plot of NDVI for the entire time series

- Review the 2D and 3D plots of water extent (water_xarray) and water quality (tsm_xarray) at the end. You can change the acquisition number to see variations in output throughout the time series. Do the values change for various time slices? The water mask is 1 for water and 0 for non-water. What is happening when there is water and non-water close together in the transect? What is happening when the TSM=0 for portions of the transect? Do you see any trends in water quality over the transect length or over time?



Example Output: 2D Transect Plot of Water for a single acquisition date



Example Output: 2D Transect Plot of TSM (water quality) for a single acquisition date

- Run this notebook several times and change the location of your data window, transect lines and the parameters you are plotting. Were you able to find any interesting results?

=====

Task-F: Data Export

Objective: This notebook will demonstrate the creation of GeoTIFF and CSV output products that can be used in other software programs (e.g. QGIS, ArcGIS, EXCEL) for more specific analyses.

- When you arrive at the website, you will find a list of Python Notebooks under the "Files" tab with the prefix "Training". There are meant to be backup files. You will also find notebooks with the prefix of your last name. These are the notebooks you should run and modify for the training session. If you do not find notebooks with your last name as the prefix, please ask the training team for help.
- Open the notebook with the following name "LastName_TaskF_DataExport". Review the entire notebook and scan the content. You will see the sample case is set up for a Vietnam data cube. Modify the cell that connects to data cubes and choose a cube of your choice. You will see that many of the selections are "commented out", so you just need to keep one line "active".
- Define a "region of interest" for your analysis. Keep this region small (less than 0.2-deg x 0.2-deg) so that it runs fast. Define a time period of 1 year. This will give you many time slices in your output file and possibly up to 23 slices, if there are images at every possible Landsat acquisition.
- Before running the code for the first time, check the list of parameters that are going into your "combined_dataset". Are these what you desire? If not, this is the time to edit the code.
- Next you will create a sample CSV export file for a given pixel. You will identify the pixel by selecting a specific Lat-Lon position and then the code will find the closest pixel to that point (nearest neighbor). Use the map at the top of the notebook to view your region and pick a Lat-Lon location. You can find an exact location by clicking on the map. The CSV file will contain the time series data for each XARRAY parameter.
- When you are ready to run the code to create the GeoTIFF export, you will remove the comment tags (#) in the last 2 lines and execute those lines in series. These lines are commented so that you can run the entire notebook without executing these lines of code, since they will generate large files. You will notice that several TIF files will be created in your directory when this part of the code is executed.
- Go to your Jupyter platform which shows your list of notebook files. You will see a folder named "geotiffs". If you click on that folder, you will find all of the TIF files that you created. If you click on any one of those files it will start to download that file to your local computer. Select one of those files and download to your computer.
- Assuming you have ArcGIS or QGIS installed on your computer, you will now open that application and view your GeoTIFF. It is important to understand what is in the new GeoTIFF file. You will find many "layers" in your file that correspond to the XARRAY layers from our data cube. Review the list of data variables in your final XARRAY as these will be the data layers in your GeoTIFF.
- After you have loaded the GeoTIFF in your GIS tool, you might want to review the RGB bands, water mask, cloud mask, and perhaps one of the indices. GIS tools have much more capability than the Data Cube User Interface or any of the Jupyter Notebooks. This is the place that most people will perform detailed GIS analyses.

Special Topic:

Data Cube Ingestion and Management

Data Cube Ingestion & Management

This session discusses managing AWS instances, obtaining new data, and ingesting new data into the cube. Though the focus is data ingestion, it will touch a broad range of topics. No materials are needed. We will cover six sections:

1. Overview of the ingestion process
2. Creating dataset types
3. Preparation
4. Indexing
5. Ingestion
6. Results Overview

1. Overview of ingestion processes

- a. What is ingestion?
 - i. Ingestion is what we call the entire process of adding new data to the Data Cube
 1. You start with any raster dataset on disk, and end with a database entry, optionally reformatted data on disk, and the ability to query for data simply using the Python API
 - ii. This process includes:
 1. Describing your source dataset in a well-defined schema - .yaml dataset type
 2. Creating a script that creates a .yaml metadata file for each dataset including required metadata
 - a. Extents, time, paths to each 'band' or measurement on disk relative to the dataset's root directory, etc.
 - b. This metadata file relates the data on disk (GeoTiff, XML, etc.) to the dataset type. For example, the dataset type defines a band sr_band1 and the metadata file defines a path to a raster dataset for 'sr_band1'
 3. Creating an ingestion configuration file that defines the input dataset type (step 1) and the output characteristics
 - a. Output characteristics refers to projection, resolution, file type, and tiling details as well as various metadata elements like the name, product type, processing level, etc.
 - b. The ingestion process will take all applicable source datasets and process them into the desired output characteristics, taking care of all the database connections and data on disk
- b. Why is it necessary?
 - i. Performance – NetCDF vs GeoTiff
 - ii. Not required if datasets are in a performant data type already

- iii. Allows for reprojection/resampling if desired, allowing for datasets to exist on a common grid/projection

2. Creating dataset types

- a. Materials needed: GPM product guide, GPM dataset type
- b. Creating a dataset type is essentially just describing dataset attributes and metadata in .yaml format
 - i. Various metadata elements – all optional
 - ii. Enumerate all ‘bands’ or measurements
 - 1. Band names
 - 2. Units (mm/h, reflectance, etc.)
 - 3. Dtype – numpy/rasterio datatypes
 - 4. Nodata value – what value in the data signifies ‘bad/missing’ data, you can enter any number here so if your data has no ‘no data’ concept then simply enter a value that won’t occur in your dataset or any value that can be considered ‘null’ or zero.
- c. A dataset type allows the Data Cube interface to identify products by their names and open them as the correct dtype. Additionally the nodata value is significant as it allows output data to correctly mask missing data.
- d. Dataset types are the base level schema of the Data Cube. The next step involves creating metadata .yaml files for each source dataset that correspond with the dataset type.
- e. Demonstration: Open both the product guide and the dataset type definition.
 - i. Identify the band names from the product guide and relate them to the dataset type
 - ii. Find a table with the dtypes, units, etc. and relate them to the dataset type
 - iii. You can pull any amount of metadata for the description, metadata fields

3. Preparation

- a. Materials needed: Preparation script, example dataset for file structure reference, output metadata .yaml file, link to default metadata type
- b. The preparation process is simply generating a .yaml file that contains all the metadata that can be generated from a raster dataset and its accompanying auxiliary data
 - i. At a minimum, this is:
 - 1. Acquisition date
 - 2. Geospatial extents in the format of lower left, upper left, lower right, upper right.
 - 3. A list of the measurement names from the dataset type and a path/layer number to the dataset on disk
 - ii. The default metadata set can be found <https://github.com/ceos-seo/agdc-v2/blob/master/datacube/index/default-metadata-types.yaml>
 - 1. Briefly show this document

- c. Generally, there are three different ways that metadata is captured.
 - i. Metadata from file names – most raster datasets are named in a way that they can be easily identified. For example, GPM data has the start/end sample times, the product type, and processing level in the file name. All of this is parsed out and saved for the final metadata set.
 - 1. Show example in script
 - ii. Metadata from the raster dataset itself – datasets are opened using rasterio and the extent and projection data are read from there.
 - 1. Show example in script
 - iii. Metadata pulled from auxiliary data, xml or otherwise
- d. The simplest way to create a new script is to use an existing script as a base, replacing only the elements that are different
- e. Show the final product of the process, show the relationship between data on disk (paths) to the band mapping

4. Indexing

- a. Materials needed: Terminal with ability to add datasets (GPM)
- b. Indexing is a very simple process after the metadata has been generated and the dataset type has been added.
- c. The indexing process creates entries in the database that contain the metadata in json format
 - i. The names and paths are saved as actual fields, but the rest of the metadata is stored directly as json
- d. Show console output of adding a dataset – this should be very quick/minor

5. Ingestion

- a. Materials needed: Ingestion configuration file, terminal with ingestion configuration ready to run
- b. The main task in ingesting a new dataset is defining a transformation between the source dataset and the output dataset
 - i. Say you have a global, UTM projection dataset that you want to line up over a specific area over Lake Chad
 - 1. You could do your analysis in the source projection and restrict the bounds manually, then reproject and align everything afterwards
 - ii. You can use the ingestion process to define a bounding box, a resolution, projection, and tile size, that will ensure that your dataset is perfectly aligned with other dataset
- c. Open the ingestion configuration file
 - i. A lot of the fields are completely arbitrary – this may be intimidating, so emphasize that only some metadata fields are required
 - ii. There are a few essential parts to this file:
 - 1. Input/output dataset types define what datasets should be considered and what the new dataset should be named

2. File naming locations/templates – these describe where your datasets will be stored and what they will be named. Since files are named using string formatting, you can use the variables `tile_index` and `start_time` to create descriptive names
3. General metadata – the only fields that are ‘required’ are those that are present in the dataset type of the source dataset. Products, platforms, product level, etc.
4. Ingestion bounds: Top/Bottom, Left/Right bounding box for your ingestion routine. This allows users to clip out data that they aren’t using to save storage space
5. Storage attributes – this is where things get a little complex. In this section, we’ll define how our data should be stored
 - a. Driver – this is always going to be NetCDF, at least for the time being. NetCDF files are much more performant than GeoTiffs/IMG files so it is the preferred format.
 - b. CRS – Almost any EPSG or WKT formatted CRS will work here. If your CRS isn’t supported then it is fairly trivial to add support for it.
 - c. Tile size: this should be the size of each storage unit in the units of your CRS.
 - i. If your CRS deals with meters, then this figure will be in terms of meters (e.g. 100000m).
 - ii. As a general rule of thumb, we like our storage units to be roughly 3000 unit squares.
 - iii. It’s worth noting that you’ll have some fairly strange looking numbers if your resolution/CRS is in fractional degrees.
 - d. Resolution: this should be the pixel spacing in terms of your CRS units – e.g. degrees, meters.
 - i. For example, an Albers equal area projection would be 25m for Landsat, while the resolution would be 0.000269494585236 for a WGS84 based projection.
 - ii. It’s extremely important that the tile size is evenly divisible by the resolution – this is less of an issue with meter based datasets and projections and more of an issue with fractional degrees.
 - iii. You’ll notice that the latitude resolution is negative – this is due to the fact that the reference point is the upper left corner.
 - e. Chunking refers to the internal chunking of the NetCDF files. Generally smaller numbers refers to better random access performance while larger number refers to better throughput.
 - f. Dimension order – standard time, lat, lon.

- g. Measurements list – this is one of the more involved parts of creating an ingestion configuration.
 - i. Each measurement that you want included in the ingested dataset should be listed
 - 1. Datasets can be excluded from the source dataset type, but measurements not in the source dataset type cannot be entered here.
 - 2. The name is what you want the name to be in the ingested dataset type – you can rename measurements here, e.g. sr_band1 to blue
 - 3. The dtype, nodata values, etc. are what you want the destination dataset type to be – if you want to convert uint16s to int32s here, there is no problem there. Nodata values will be cast to the destination dataset type, so if the source nodata is -9999 and the destination nodata is 0, all instances of -9999 will be replaced with 0.
 - 4. The src_varname will map the new measurement to the dataset type measurement. You are able to rename the name attribute as long as the src_varname corresponds to a measurement in the dataset type.
 - 5. Other attributes are all optional and can be included or excluded as you see fit.
 - h. Once everything has been entered into the configuration file, you are able to run ingestion
- iii. You can now run ingestion
 - 1. Run ingestion live on the screen – show output, take some questions
- iv. Tie things back to what was previously shown – this is how you get to the steps previously seen in the Notebooks.

6. Results overview

- a. Tie everything together back to the API demonstration – this is a ‘simple’ prerequisite step to simple, managed, large scale data access and analysis.
- b. Q&A. I’m assuming there will be quite a few ‘can this be ingested’ etc. questions.



CEOS Data Cube User Interface

User Guide



Welcome to the Open Data Cube

CEOS is using the power of the Open Data Cube to help address the needs of satellite data users, giving them a better picture of their land resources and land change.

- Ease of use and access to satellite-based data
- Multiple dataset interoperability and spatial consistency
- Use of 'Analysis Ready' Data Products
- A Shift in Paradigm from Scenes to Pixels

[Log In](#)

The Open Data Cube

Open Data Cube (ODC), opendatacube.org, is an open source project was born out of the need to *better manage Satellite Data*. It has evolved to support interactive data science and scientific computing. ODC will always be 100% open source software, free for all to use and released under the liberal terms of the Apache 2.0 The Open Data Cube provides a software framework used in remote sensing and Earth observation sciences. It is composed of data structures, tools, and mechanisms which facilitate the storage, organization, and analysis of large data grids. For the latest copy of the code, questions, help with installation, or general suggestions you can visit our GitHub: <https://github.com/opendatacube>



CEOS Data Cube User Interface (UI) Overview

The Committee on Earth Observation Satellites (CEOS), www.ceos.org, has developed a web-based interface to run high level analyses on case study areas using the Open Data Cube. The UI provides a typical GIS interface to the Open Data Cube.

This guide is designed to get you started with the CEOS Data Cube UI and provides short instructions for the common tasks performed and overviews built-in analysis capabilities.

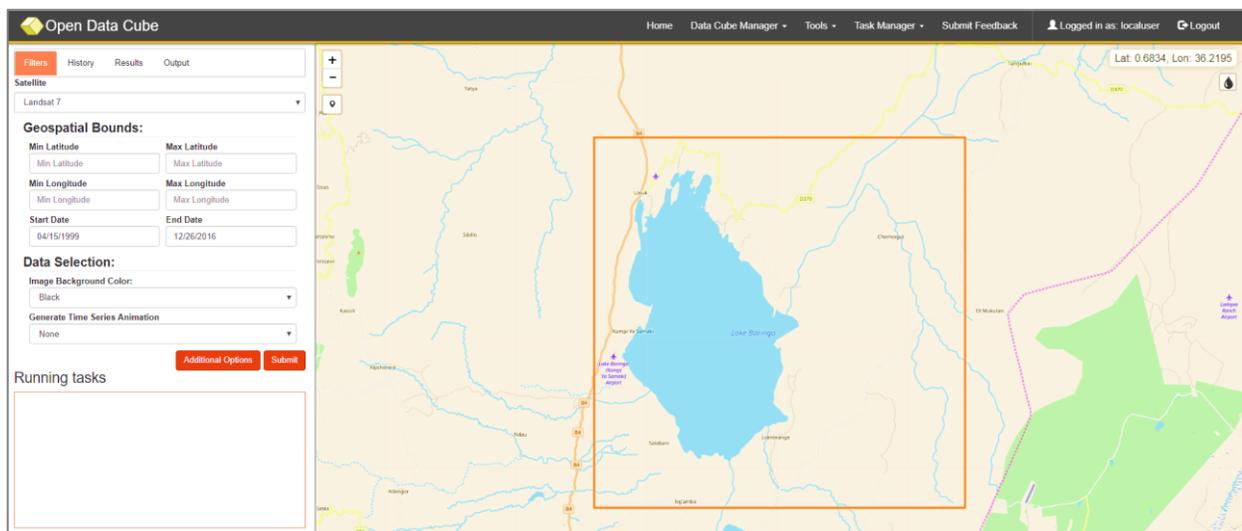
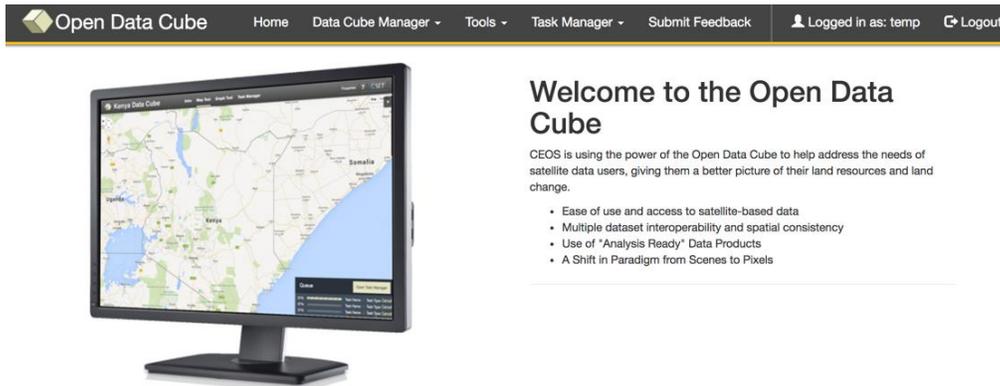


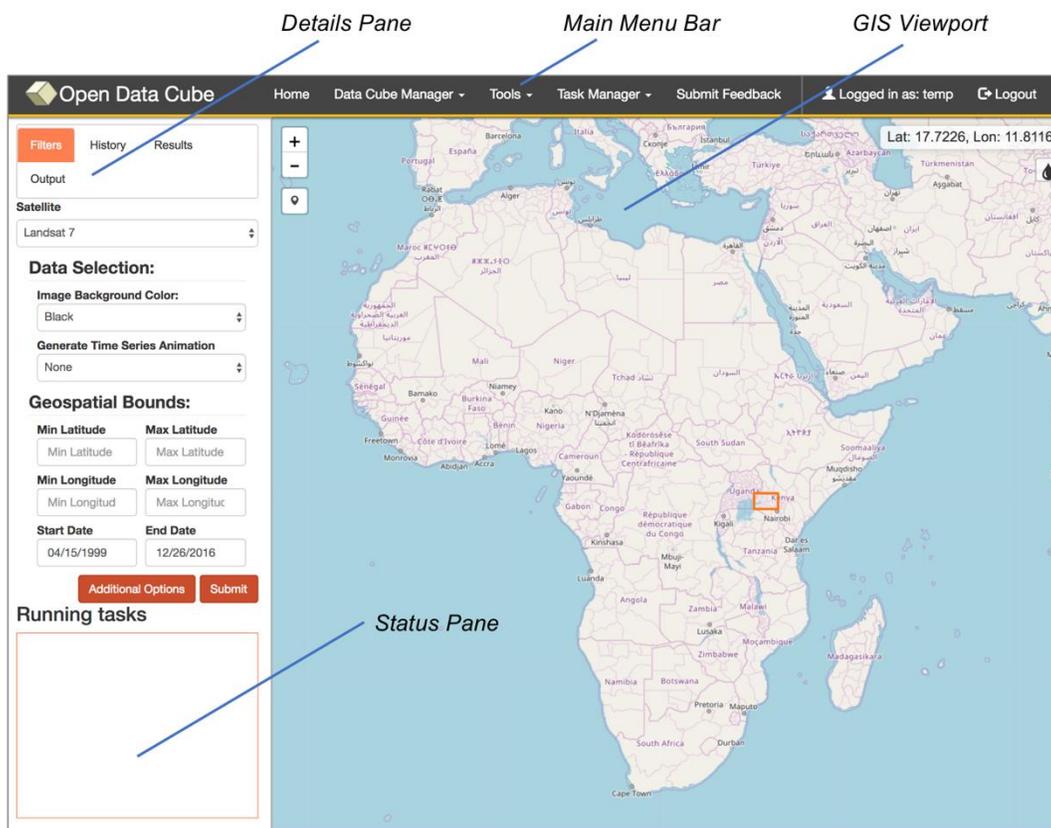
Figure 1: Overview of CEOS Data Cube UI

CEOS User Interface Features

The CEOS user interface (UI) was developed using the Open Data Cube to showcase high level analysis on case study areas. This guide is intended to list the features and options of the various parts of the CEOS UI. The UI home page is shown below. At the home page, select **Login** if you have an account or click **Register here** to create a new one. An account is required in order to run an analysis.



The primary layout of the user interface follows that of a standard GIS tool. A typical interface that you would see during an analysis is shown below. In general, there are four areas you will see: (1) Main Menu Bar, (2) GIS Viewport, (3) Details Pane, and (4) Status Pane.



Run Analysis

Run specialized case studies ranging from flood extent monitoring, and coastal erosion, to experimenting with image compositing techniques.

The screenshot shows the Open Data Cube web application interface. The top navigation bar includes 'Home', 'Data Cube Manager', 'Tools', 'Task Manager', 'Submit Feedback', and 'Logged in as: localuser'. The left sidebar contains a 'Filters' tab, 'History', 'Results', and 'Output' sections. Under 'Filters', the 'Satellite' is set to 'Landsat 7'. The 'Data Selection' section includes 'Image Background Color' set to 'Black' and 'Generate Time Series Animation' set to 'None'. The 'Geospatial Bounds' section shows 'Min Latitude' (0.5000), 'Max Latitude' (0.7462), 'Min Longitude' (36.0009), and 'Max Longitude' (36.2471). The 'Start Date' is 04/15/1999 and the 'End Date' is 12/26/2016. A 'Submit' button is visible. The main map area shows a satellite image of a region with a color-coded overlay (blue, green, yellow, red) indicating analysis results. The map includes a search bar, zoom controls, and a coordinate display showing 'Lat: 0.7953, Lon: 36.1321'. The bottom right corner of the map area includes the text 'Leaflet | Map data © OpenStreetMap contributors'.

Visualize Data Cube Holdings

Visualize data-cube data-holdings, extents, and cloud cover meta-data.

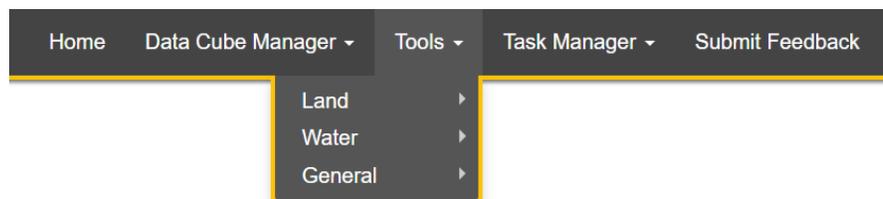
The screenshot shows the Open Data Cube web application interface for visualizing data cube holdings. The top navigation bar is identical to the previous screenshot. The left sidebar features the 'Ingested Data Cubes' section, which includes a description: 'The regions visualized on the map represent the various Data Cubes that are present and ingested on the system.' Below this, there are instructions on how to use the filtering options. The 'Start Date' is set to 01/01/2010 and the 'End Date' is 01/02/2010. The 'Source Dataset Type' is set to 'All'. A 'Select a platform to filter Data Cubes' dropdown menu is also present. The main map area shows a world map with blue squares indicating data cube holdings across various regions, including South America, Africa, and Asia. The map includes a search bar, zoom controls, and a coordinate display showing 'Lat: 0.7953, Lon: 36.1321'. The bottom right corner of the map area includes the text 'Leaflet | Map data © OpenStreetMap contributors'.

Running an Analysis – Water Detection, Lake Baringo, Kenya

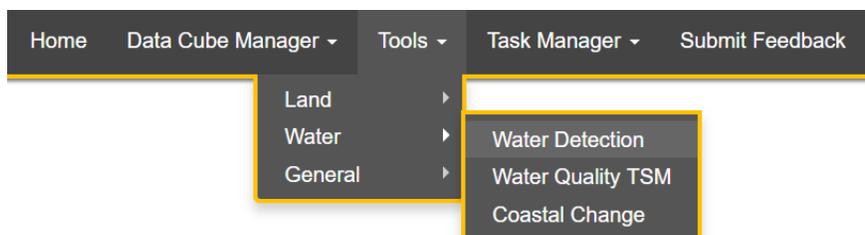


For our example we will be concentrate on Lake Baringo in Kenya. Note, an account is required in order to run an analysis, if don't have an account please register from the home page.

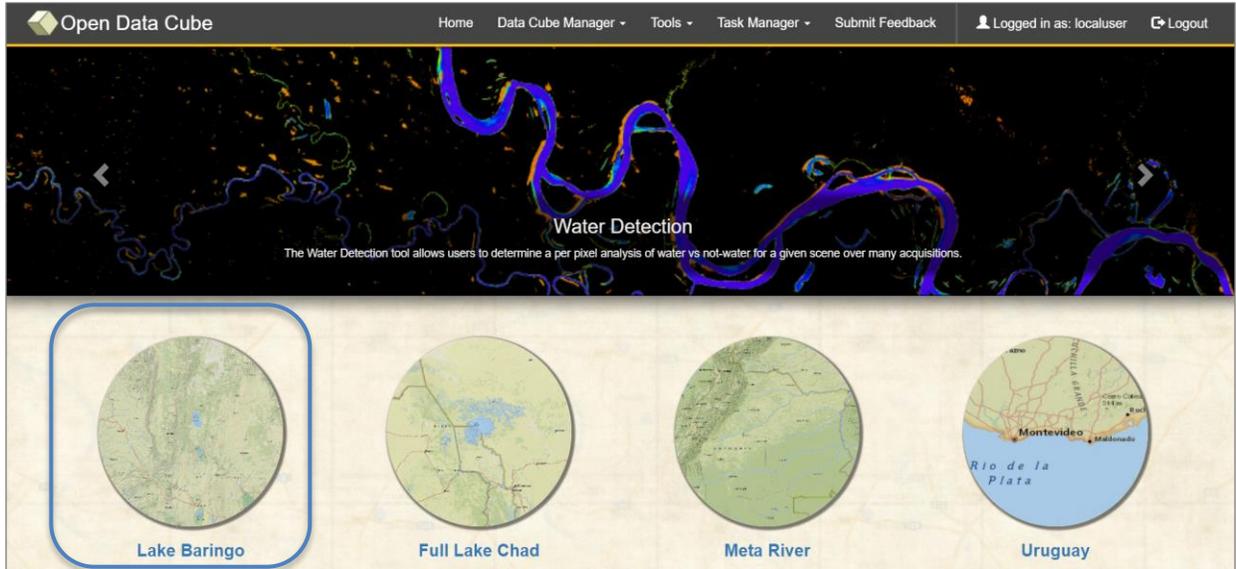
The Tools tab in the menu at the top of the screen allows you to select from several different classes of analyses.



For this example you will be running a water classifier to monitor flooding extents. Under the Tools menu, first select Water, and then select Water Detection.



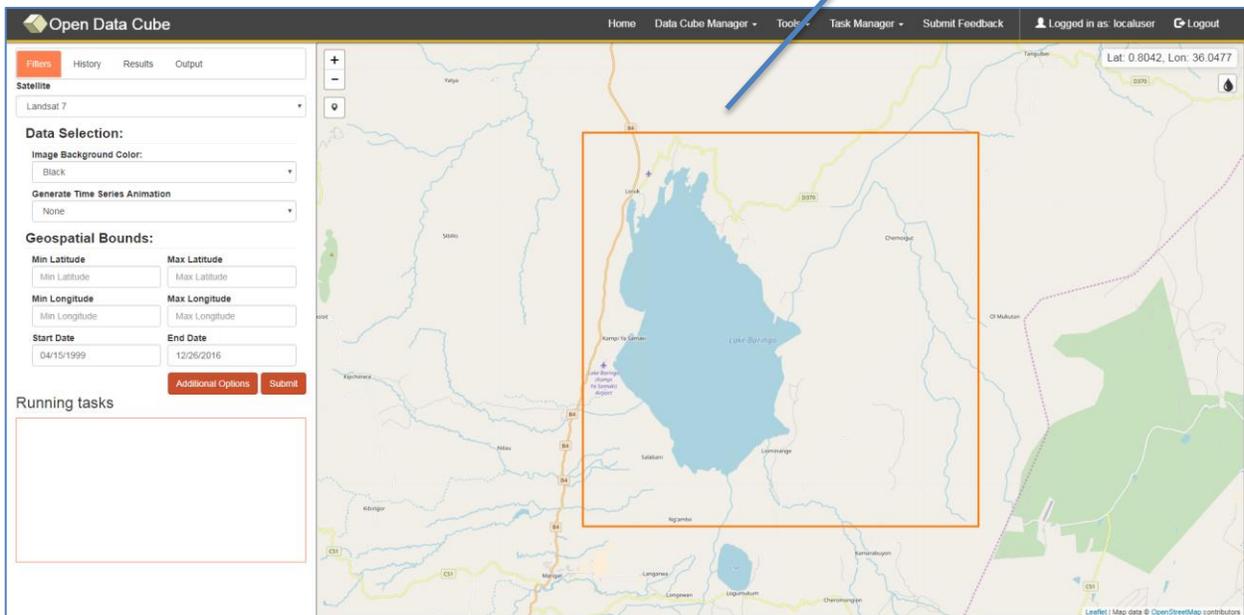
You will be greeted with a page listing all case study areas on which the chosen tool can be run:



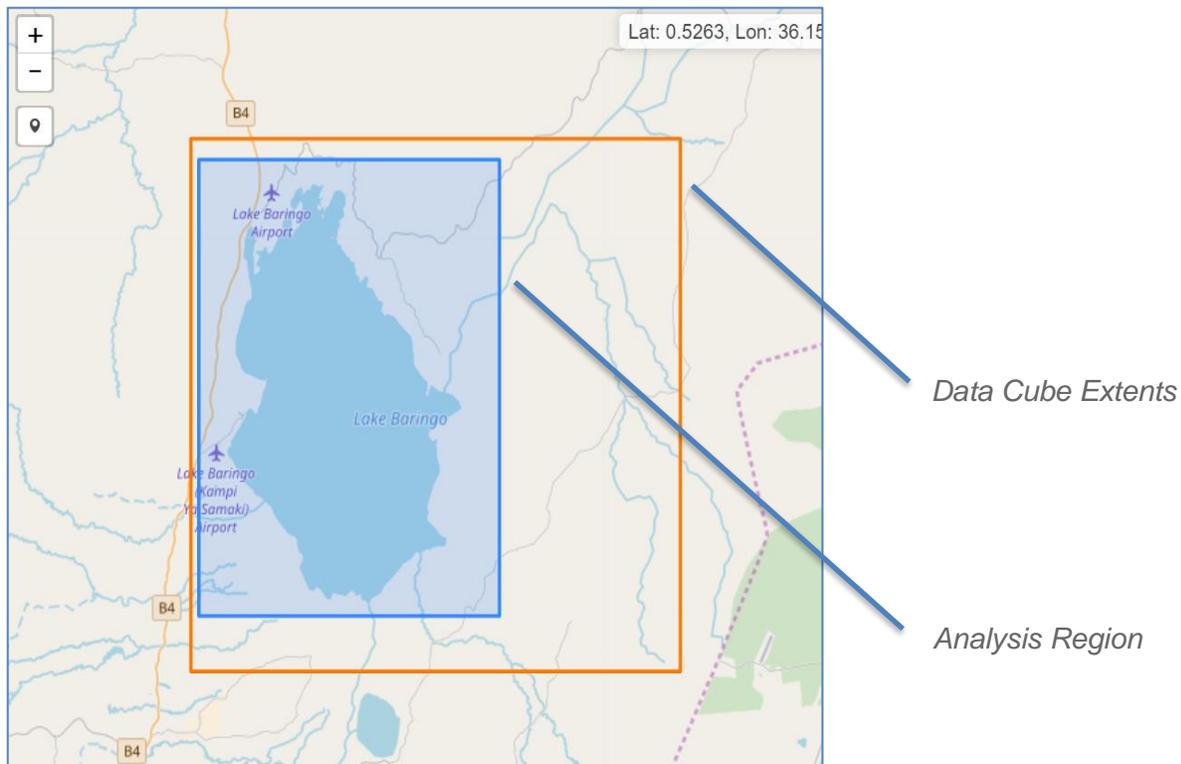
Select Lake Baringo from your list of case study areas. Note, there are multiple data cubes that are accessible from this UI.

Data Cube Extents

You should see the following interface:



To run your first analysis, begin by selecting an area of the map to analyze. The orange box drawn on the map represents the maximum extents of the data contained in the Data Cube. Click and drag a rectangle anywhere within that box to define a region for your analysis.



The panel on the left side of the user interface contains the processing options for the tool:

Note: There are several options which are common among all of the tools in the UI.

- The geospatial bounds (latitude and longitude) are dynamically updated to reflect the region drawn on the map but can also be entered manually.
- The temporal bounds (start and end dates) similarly provide a way to choose a subset of the data in the Data Cube.

For now, simply click the **Submit** button to start running the water detection algorithm.

Filters
History
Results
Output

Satellite

Landsat 7 ▼

Data Selection:

Image Background Color:
Black ▼

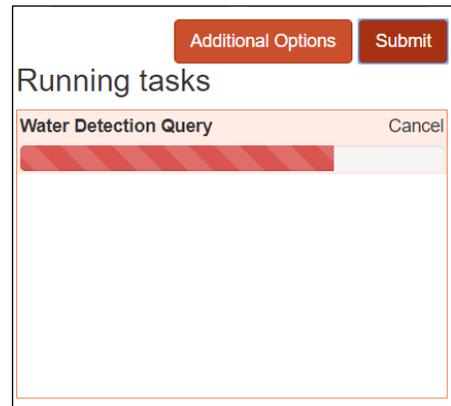
Generate Time Series Animation
None ▼

Geospatial Bounds:

Min Latitude	Max Latitude
<input type="text" value="0.5221"/>	<input type="text" value="0.7415"/>
Min Longitude	Max Longitude
<input type="text" value="36.0046"/>	<input type="text" value="36.1469"/>
Start Date	End Date
<input type="text" value="04/15/1999"/>	<input type="text" value="12/26/2016"/>

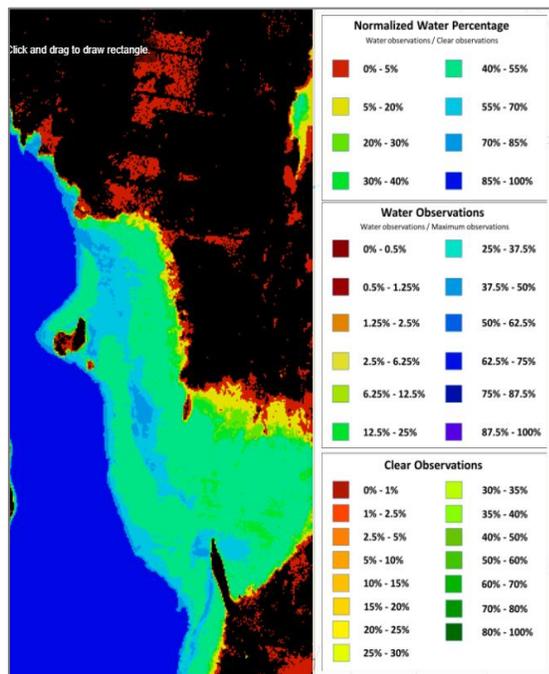
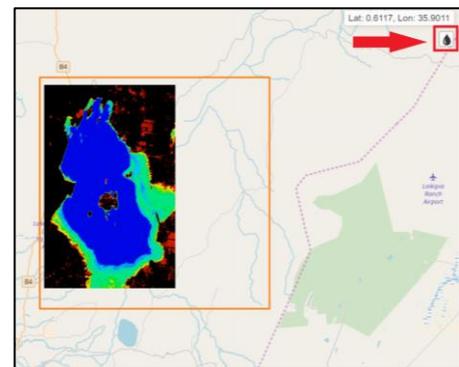
Additional Options
Submit

A new task labeled **Water Detection Query** will appear in the list of **Running Tasks** below the processing options panel. The red progress bar will show the progress of your algorithm. The amount of time required to complete the analysis varies depending on the type of analysis and the spatial and temporal extents you have chosen.



When the processing is finished, your result will appear on the map (see figure to the right).

To view information about the color scale used in the result, select the teardrop-shaped icon [🔹] on the top right corner of the map panel.



Legend Description

What's revealed is a legend detailing the coloring of the water product.

Black - no water observations

Red - on rare occasion, water was observed here

Yellow - Up to 25% of observations contain water

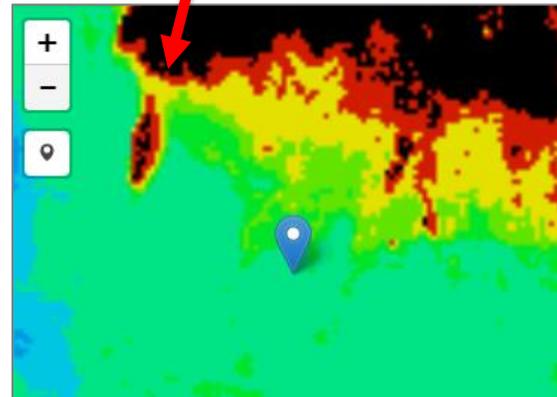
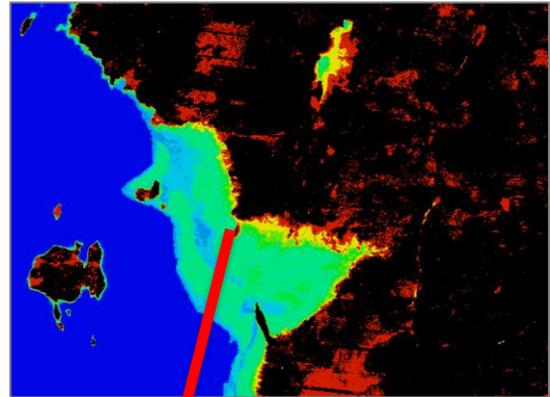
Teal - Up to 50% of observations contain water

Dark Blue - majority of observations are water

Pixel Drill Analysis

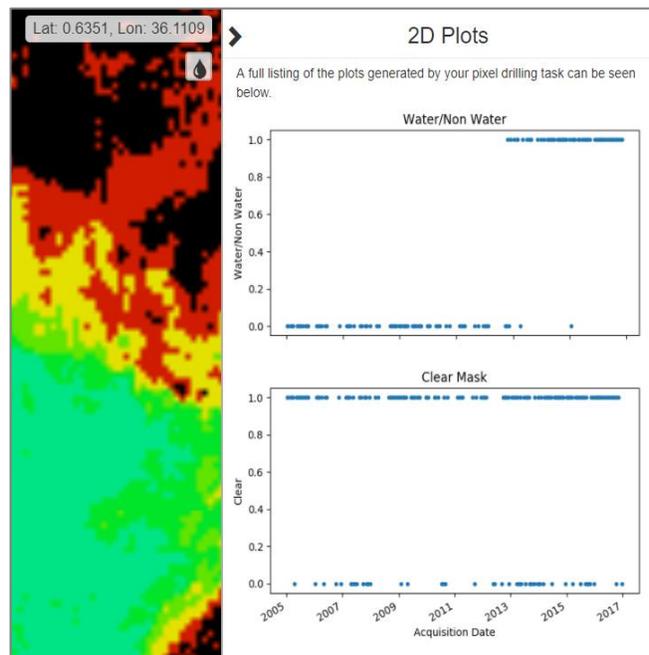
To further the analysis, you may wish to “drill down” through the entire time series of a single pixel. Since green pixels represent locations that had a mix of wet and dry observations, let’s do a pixel drill on one of the green areas of the result.

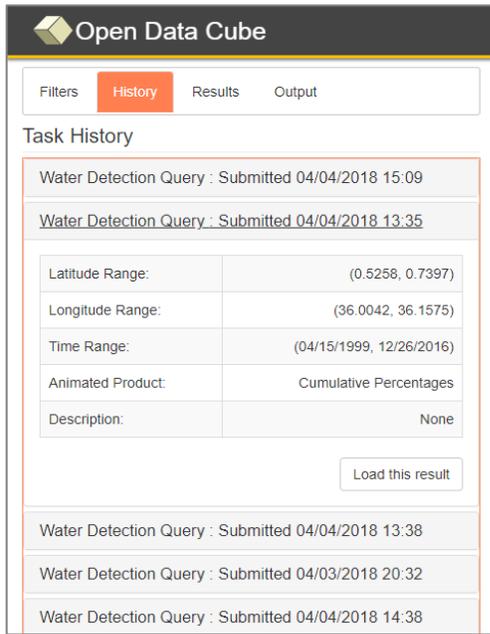
Select the marker icon [📍] in the top left corner of the map and click on the part of the map you wish to drill.



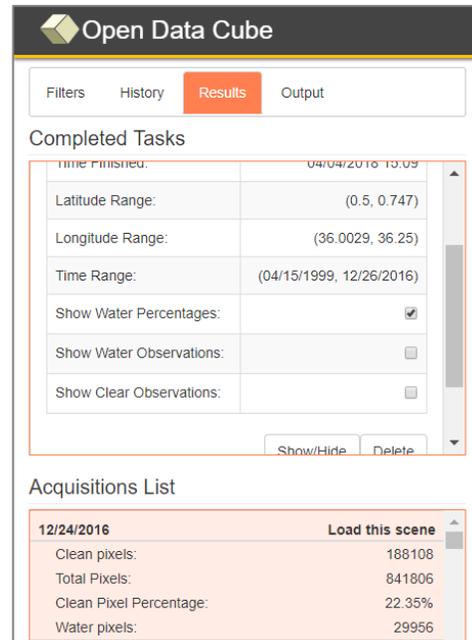
Results of the pixel drill will be displayed as a pair of 2D plots on the right side of the page. These plots mark observations reporting water or clear as 1, and not water or not clear as 0.

In this case, it appears that this area became permanently inundated after 2013.





History Tab



Results Tab

History Tab

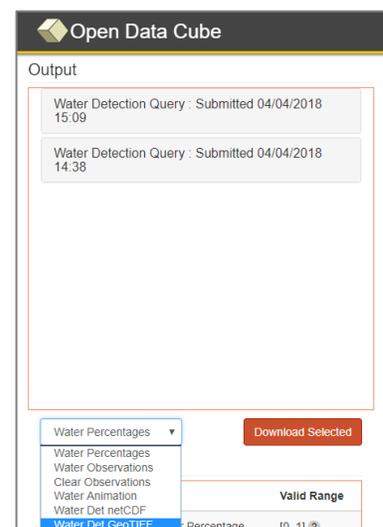
The history tab allows you to load in past analyses, as well as view metadata about their geospatial and temporal extents.

Results Tab The results tab provides metadata about the data used in the analysis, such as:

- How many acquisitions are considered in the analysis?
- How many of the pixels are cloud, how many of them are water?
- What are cloud coverage or water coverage stats for the area?

Output Tab

This tab is used to export the results of your analysis for download. It supports common formats such as PNG, NetCDF, and GeoTIFF.



Output Tab

Algorithm Library

The UI offers access to numerous other analyses from urbanization and coastal erosion to cloud-free mosaicking and landslide detection. The following section is a complete listing of algorithms implemented in the User Interface as well as a glossary of processing options unique to each algorithm. We cover three general classes in the following sections:

1. Water-based Algorithms:

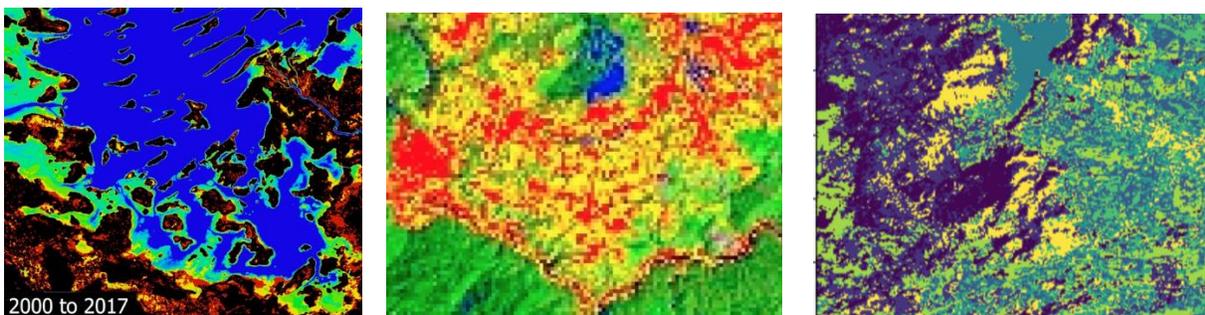
- a. Water Detection – Water Observations Form Space (WOFS)
- b. Water Quality – Total Suspended Matter (TSM)
- c. Costal Change – Change in coastal region

2. Land-based Algorithms:

- a. Urbanization – Normalized Difference Built-up Index (NDBI)
- b. Spectral Indices – Normalized Difference Vegetation Index (NDVI), Normalized Difference Water Index (NDWI), Normalized Difference Built Index (NDBI), Enhanced Vegetation Index (EVI), Soil Adjusted Vegetation Index (SAVI), Normalized Burn Ratio (NBR), Normalized Burn Ration 2 (NBR 2)
- c. Landslides – Sudden Landslide Identification Product (SLIP)
- d. NDVI Anomaly – Deviation in NDVI index from a specific baseline period
- e. Fractional Coverage – Vegetational fractional cover

3. General Algorithms

- a. Custom Mosaic – Creation of cloud-free composites
- b. Cloud Coverage – Mosaic displaying cloud coverage



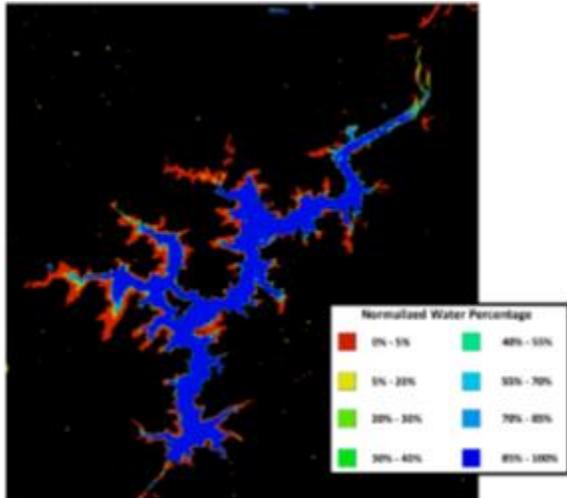
As the library of algorithms grows, the benefit to the community will be significant. All algorithms that have been implemented in the UI are also available in our GitHub repository, please see opendatacube.org.

Water-based Algorithms

The following section describes three water-based Algorithms that have been implemented in the UI: Water Detection, Water Quality, and Coast change.

Water Detection

The water detection analysis tool allows users to run the WOFS (Water Observations from Space) algorithm on a selected area. The output is a time series of water classifications and observations. This can indicate water cycle dynamics, historical water extent, and the risk of floods and droughts.



WOFS Analysis over an area in Uruguay

Generate Time Series Animation

None: Generate static image

Scene: A general time series animation

Cumulative Percentages: The number of water observations over the number of clear observations

Cumulative Observations: The numbers of water observations over the maximum observation

Image Background Color

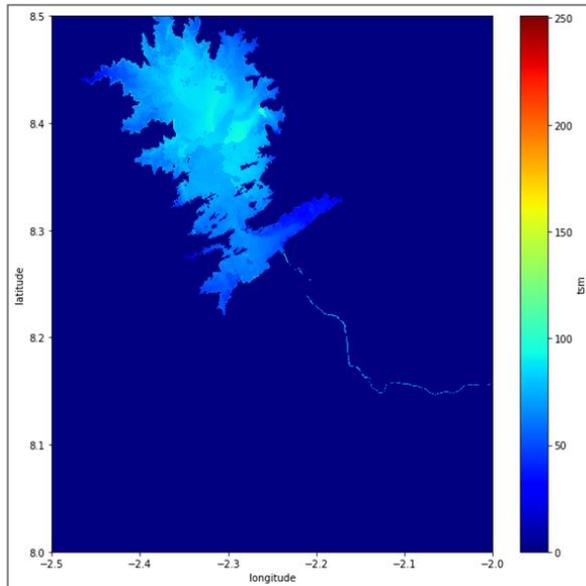
Black: Sets the background to solid black

White: Sets the background to solid white

Transparent: Displays the selected area as the background

Water Quality TSM

TSM (Total Suspended Matter) is a measure of the particulate matter in water and is often a proxy for water quality.



Total Suspended Matter: An Area in the Bui National Park (Ghana)

Result Type (Map view/PNG)

Average TSM: Display the average TSM of every observation for the given pixel

Minimum TSM: Display the lowest TSM from every observation for the given pixel

Maximum TSM: Display the highest TSM from every observation for the given pixel

TSM Variability: Variability of Total Suspended Matter

Generate Time Series Animation

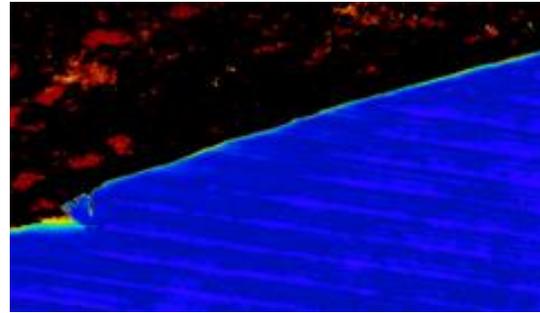
None: Does not have any time series animation

Scene: A general times series animation

Cumulative Average: Total Suspended Matter in grams per liter (g/L)

Coastal Change

This parameter is a measurement of change in a coastal region based on two selected time periods. It can display both the overall change in the coastal area as well as highlight the gain or loss in coastlines.



Water detection map of the Area east of Lomé,
Spanning over a decade (2000, 2016)

Generate Time Series Animation

None: Generate static image

Coastline Change: Time series of the coastline change over a period of time

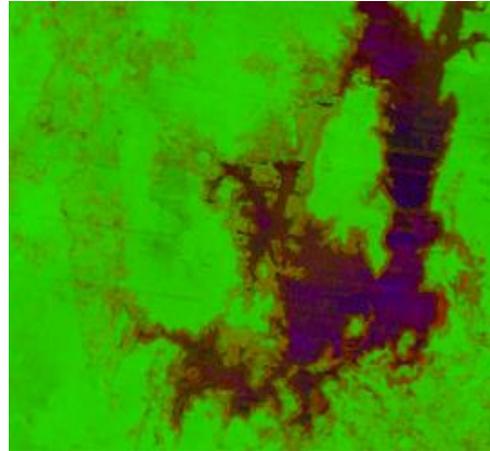
Coastal Change: Times series of the overall coastal change over a period of time

Land-based Algorithms

The following section describes five land-based algorithms that have been implemented in the UI: Urbanization, Spectral Indices, SLIP, NDVI Anomaly, Fractional Coverage

Urbanization

A measurement in the growth or loss of urbanized land in a selected area. It uses NDBI (Normalized Difference Built-up Index) as a proxy that correlates with urbanization.



Compositing Method

Least Recent Pixel: Creates a composite of the least recent pixels with no cloud coverage

Max NDVI Pixel: Creates a composite using pixels with the max NDVI values

Median Pixel: Creates a composite of the median recent pixels with no cloud coverage

Min NDVI Pixel: Creates a composite using pixels with the min NDVI values

Most Recent Pixel: Creates a composite of the most recent pixels with no cloud coverage

GeoMed: Computes geometric median of the observations for this pixel

Spectral Indices

These are used to highlight a particular feature of interest. The selected spectral index can be used to view changes over a designated time period, highlighting a particular spectral index using various compositing methods.

Spectral Index

NDVI: Normalized Difference Vegetation Index

NDWI: Normalized Difference Water Index

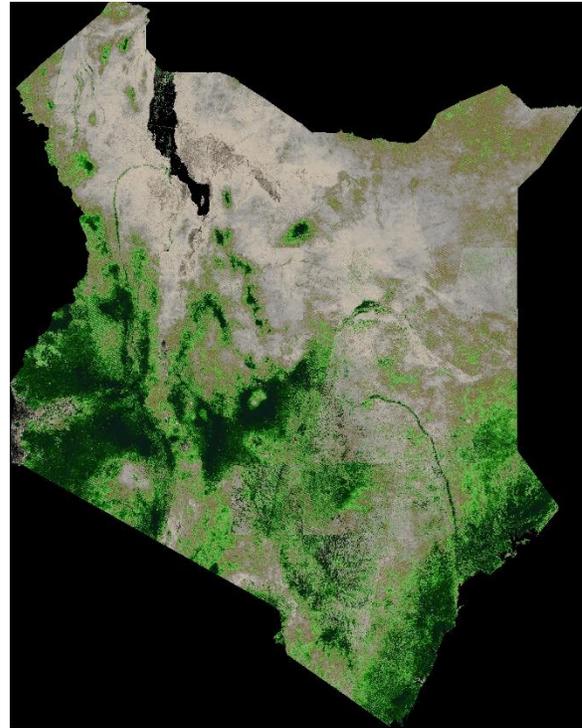
NDBI: Normalized Difference Built Index

EVI: Enhanced Vegetation Index

SAVI: Soil Adjusted Vegetation Index

NBR: Normalized Burn Ratio

NBR2: Normalized Burn Ratio 2



NDVI mosaic for Kenya (LS7 2000 to 2017)

SLIP

SLIP (Sudden Landslide Identification Product) is used to detect locations of potential landslides by comparing an acquisition to a historic baseline before it. The differences in several vegetative spectral indices identify sudden clearing of vegetation, and filtering those through a DEM slope mask highlights regions where the clearing may be due to landslide.



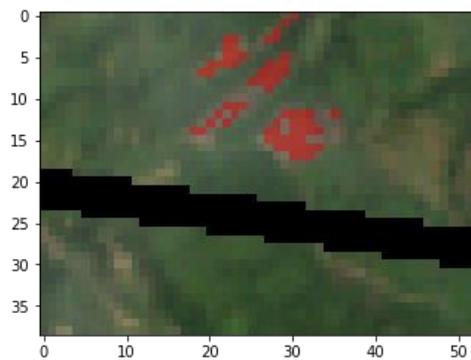
Baseline Length

Number of acquisitions used to create baseline composite.

Baseline Method

Average: An average of all available acquisitions

Composite: The most recent acquisitions



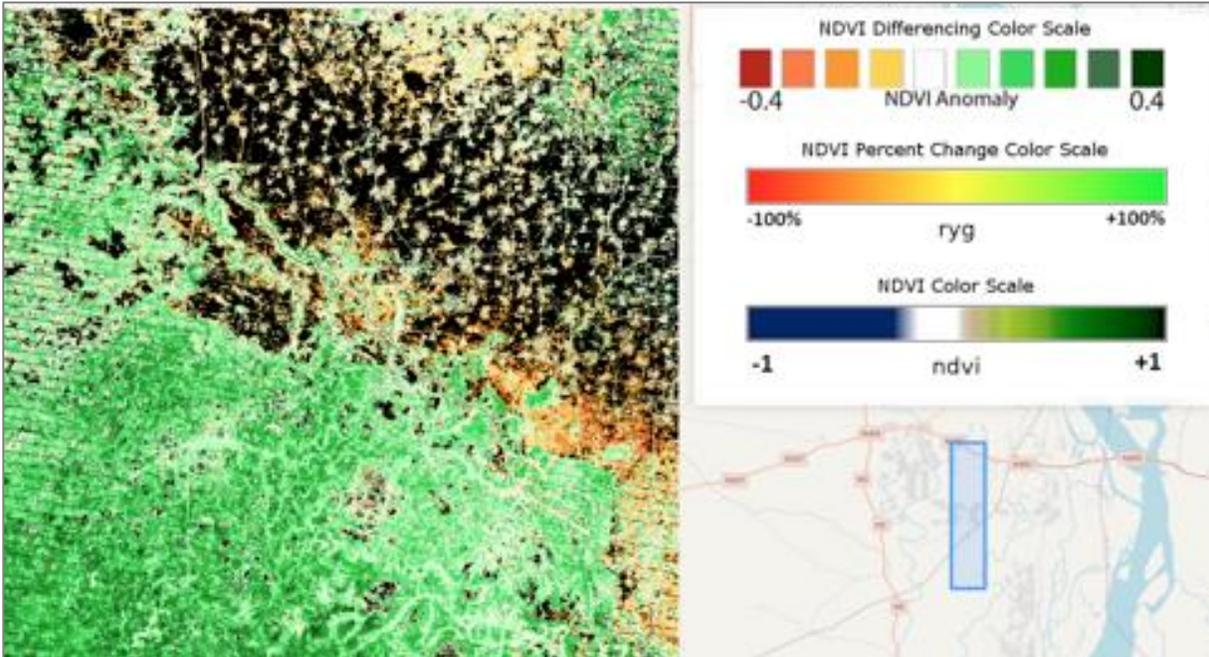
SLIP detects landslides in *Salgar, Colombia* by making use of both ASTER DEM and Landsat7 data in shared setting. The output is a binary Land-Slide Classification.

NDVI Anomaly

This analysis identifies deviations in NDVI index from a specified baseline period. Depending on the threshold the output will highlight areas with noticeable increases and decreases in vegetation.

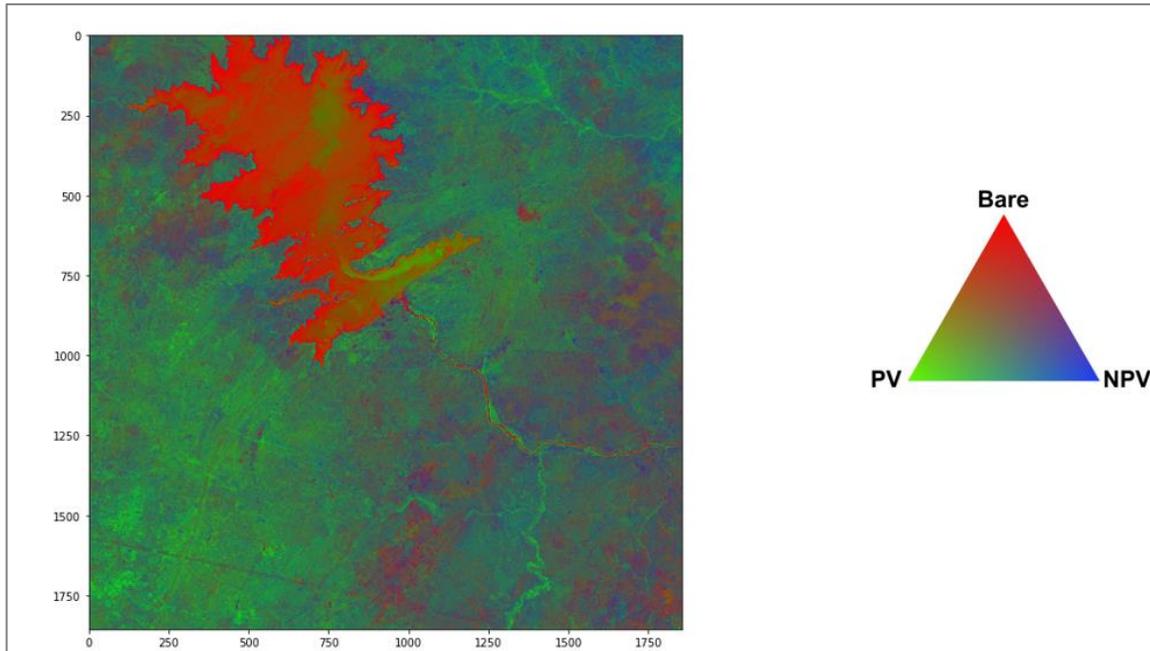
Baseline Period

Select the month(s) to focus the analysis on with the given time period.



Fractional Coverage

Vegetation fractional cover represents the exposed proportion of Photosynthetic Vegetation (PV), Non-Photosynthetic Vegetation (NPV), and Bare Soil (BS) within each pixel. This product is useful for natural resource management, modeling carbon dynamics and assessing land cover change in time series.



Fractional Cover – An Area in the Bui National Park (Ghana)

General Algorithms

Custom Mosaic

This tool allows users to create detailed cloud free composites of imagery. The result lets you specify any ordering of NIR, SWIR1, SWIR2, RED, GREEN, and BLUE composited bands

Users can also select their desired compositing method ranging from Least Recent Pixel to Most Recent Pixel and also MAX and MIN NDVI Pixel. Data is represented on an RGB scale.

Result Type

Let's you map any ordering of bands to red green blue channels in an image composite.



True color mosaic for Kenya (LS7 2000 to 2017)
with most recent pixel method

Compositing Method

Least Recent Pixel: Composite by choosing oldest pixels.

Most Recent Pixel: Composite by choosing most recent pixels.

Max NDVI Pixel: Composite is built using pixels with the highest NDVI values.

Min NDVI Pixel: Composite is built using pixels with the lowest NDVI values.

Median Pixel: Chooses median valued pixel on respective bands.

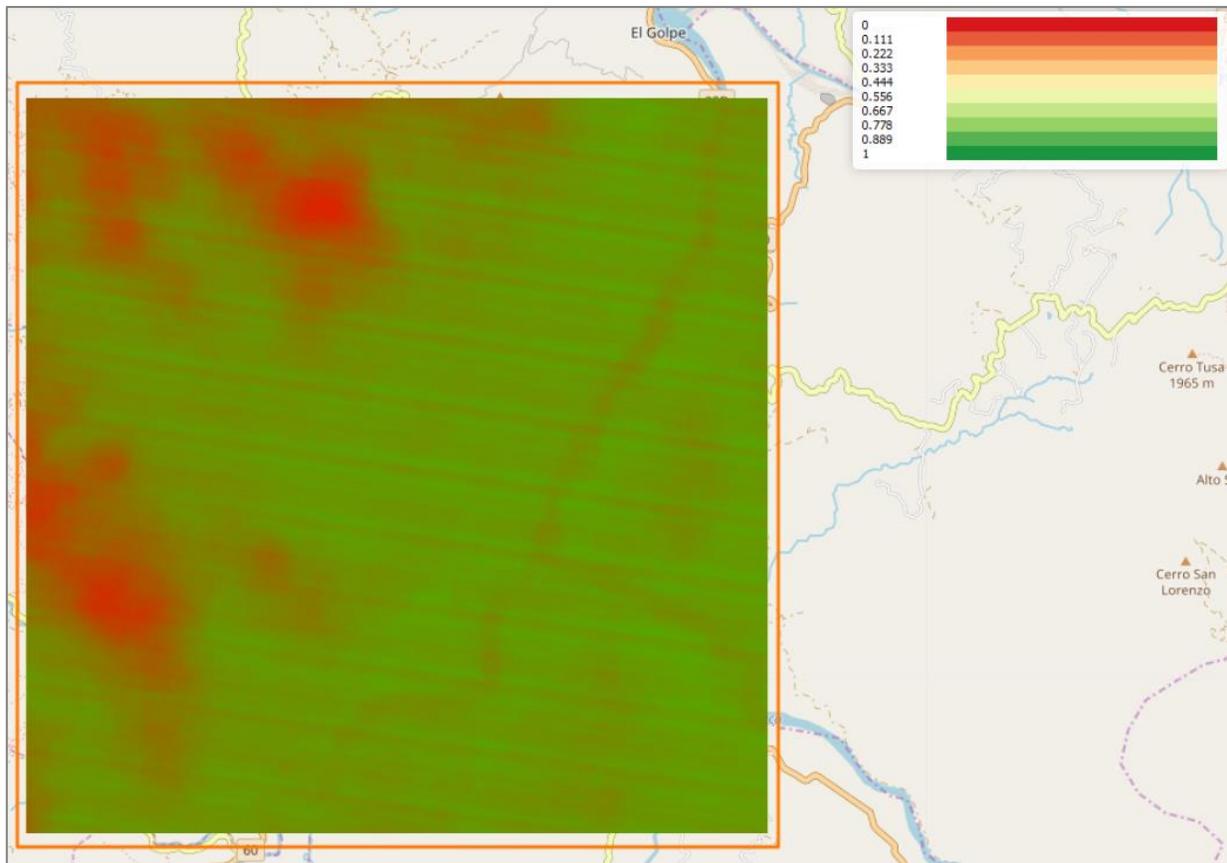
GeoMed: Computes geometric median of the observations for this pixel

Cloud Coverage

This tool creates a mosaic displaying cloud coverage by computing the percentage of pixels that are identified as a cloud for all acquisitions that are selected. Users can then specify the coordinates they desire under Geospatial Bounds.

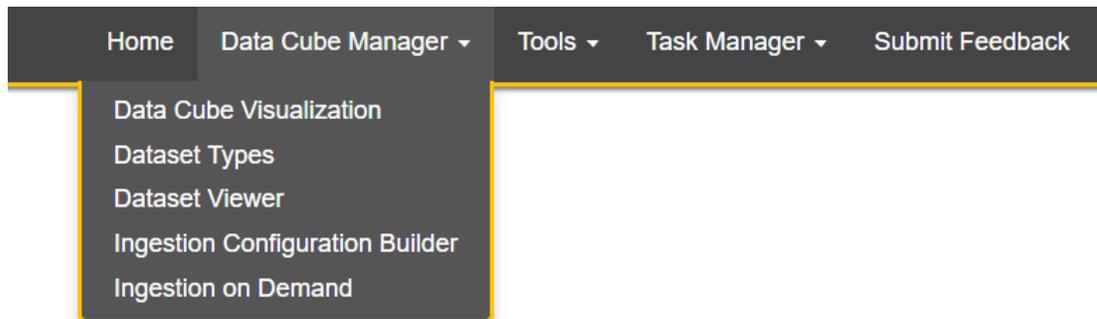
Processing Options

Beyond specifying spatial and temporal extents, there are no additional processing options.



Data Cube Manager

The Data Cube Manager contains several options for exploring, exporting, and viewing configuration details of your data cube data holdings.



Data Cube Visualization: Visualize and browse available data on an interactive map.

Dataset Types: Displays a list of all dataset types defined in the cube.

Dataset Viewer: A list of every acquisition in the cube organized by dataset.

Ingestion Configuration Builder: A form used to build data cube ingestion configurations.

Ingestion on Demand: create and download a subset your data-holdings.



Visualize Data Holdings

Ingested Data Cubes page presents all available platforms for viewing ingested data cubes.

The screenshot displays the 'Open Data Cube' interface. The main heading is 'Ingested Data Cubes'. Below the heading, there is a brief explanation of the map and filtering options. The 'Start Date' is set to 01/01/2010 and the 'End Date' is 01/02/2010. The 'Source Dataset Type' is set to 'All'. A list of datasets is shown with checkboxes, including 's1_gamma0_vietnam', 'ls8_lasrc_vietnam', 'alos2_palsar_vietnam', 'alos_palsar_vietnam', 'ls7_ledaps_vietnam', 'gpm_imerg_gis_daily_global', and 'gpm_imerg_gis_monthly_global'. The map on the right shows a region in Southeast Asia with several blue boxes highlighting specific areas.

- Displayed dataset can be filtered by **date range**, or by **dataset type**
- Selecting a region with multiple datasets available will create a list below **Source Dataset Type**. You can either remove a dataset source from the list or click it to view additional information
- You can click on any of the existing regions to view additional metadata.

Additional Information ✕

Satellite	LANDSAT 8
Product	ls8_lasrc_vietnam
Min Latitude	10.5137922538121
Max Latitude	12.6112686107039
Min Longitude	106.789924345617
Max Longitude	108.919201063567
Start Date	2014-01-14
End Date	2016-12-21
Scene Count	60
Pixel Count	61493483

The Additional Information dialog box will display a brief overview of the dataset, including the satellite platform, product, extents, and display details.

- **View Dataset Type** will load selected data into *Dataset Types* tool.
- **View Datasets** will load selected data *Dataset Viewer* tool.
- **Create a Sample Cube** will redirect you to the *Ingestion On Demand* tool.

Dataset Types

Datasets in the Data Cube are organized using a data-type system. This page lists all available dataset types as well as options to view the full configuration of the types and a link to the Dataset Viewer too.

Open Data Cube Home Data Cube Manager Tools Task Manager Submit Feedback Logged in as: localuser Logout

Dataset Types

Dataset type definitions are used to describe datasets contained in the Data Cube. Individual datasets are associated with a single dataset type. Each dataset type includes a variety of data and metadata including dataset measurement data, product types and platforms, and creation dates and users.

Show 10 entries Search:

id	Name	Platform	Instrument	Product Type	Measurements	Description	View Datasets	View Full Definition
3	terra_aster_gdm_v2_scene	TERRA	ASTER	GDEM_V2	dem, num.	Global Elevation Data from TERRA satellite's ASTER sensor.	View datasets	View definition
4	gpm_imerg_gis_monthly	GPM	GPM	monthly	total_precipitation, liquid_precipitation, ice_precipitation, pe...	GIS translation of the imerg data product. Multi satellite. gi...	View datasets	View definition
5	gpm_imerg_gis_daily	GPM	GPM	daily	total_precipitation, liquid_precipitation, ice_precipitation, pe...	GIS translation of the imerg data product. Multi satellite. gi...	View datasets	View definition
6	gpm_imerg_gis_hourly	GPM	GPM	hourly	total_precipitation, liquid_precipitation, ice_precipitation, pe...	GIS translation of the imerg data product. Multi satellite. gi...	View datasets	View definition
7	ls5_collections_sr_scene	LANDSAT_5	TM	LEDAPS	sr_band1, sr_band2, sr_band3, sr_band4, sr_band5, sr_ba...	Landsat 5 USGS Collection 1 Higher Level SR scene proe...	View datasets	View definition
8	ls7_collections_sr_scene	LANDSAT_7	ETM	LEDAPS	sr_band1, sr_band2, sr_band3, sr_band4, sr_band5, sr_ba...	Landsat 7 USGS Collection 1 Higher Level SR scene proc...	View datasets	View definition
9	ls8_collections_sr_scene	LANDSAT_8	OLI_TIRS	LaSRC	sr_band1, sr_band2, sr_band3, sr_band4, sr_band5, sr_ba...	Landsat 8 USGS Collection 1 Higher Level SR scene proe...	View datasets	View definition
10	s1_gamma0_scene	SENTINEL_1	SAR	gamma0	vh, vv.	Sentinel-1A/B SAR Gamma0 scenes, processed to the CE...	View datasets	View definition
13	terra_aster_gdm_saigar_colombia	TERRA	ASTER	GDEM_V2	dem, num.	Terra Aster -- metre, UTM 36 projection	View datasets	View definition
14	gpm_imerg_gis_daily_global	GPM	GPM	daily	total_precipitation, liquid_precipitation, ice_precipitation, pe...	Global NetCDF GPM IMERG GIS data	View datasets	View definition

Showing 1 to 10 of 63 entries Previous 1 2 3 4 5 6 7 Next

Dataset Viewer

The data set viewer panel organizes all individual products stored in the Data Cube. Options exist to sort by **product name**, by **spatial extent**, or by **time range**.

Open Data Cube Home Data Cube Manager Tools Task Manager Submit Feedback Logged in as: localuser Logout

Datasets

Datasets are individual chunks of data - either full acquisitions (scenes) or chunks of acquisitions created during ingestion. Information found in the dataset is the id, added date, and acquisition/chunk data such as extents, date acquired, and path to dataset. The table found on this page contains the datasets that match the criteria specified by the datasets form.

Products

- terra_aster_gdm_v2_scene - Source only
- gpm_imerg_gis_monthly - Source only
- gpm_imerg_gis_daily - Source only
- gpm_imerg_gis_hourly - Source only

Show 15 entries

id	Platform	Instrument	Product Type	Upper Left	Lower Right	Dataset Acq. Date	Format
6cc518be-33df-4951-a716-9722f0dc8380	TERRA	ASTER	GDEM_V2	-173.00, -13.00	-172.00, -14.00	2018-03-16 01:59:58	GeoTiff
f3e8c26e-961a-4b1a-949d-7a0526509b0a	TERRA	ASTER	GDEM_V2	-73.00, 2.00	-72.00, 1.00	2017-08-03 13:15:20	GeoTiff
4227a25d-6b65-450c-9d9a-95159446761b	TERRA	ASTER	GDEM_V2	-74.00, 2.00	-73.00, 1.00	2017-08-03 13:02:53	GeoTiff
c91b2e9e-48b6-4567-a8bc-6ad593f5d30a	TERRA	ASTER	GDEM_V2	-75.00, 2.00	-74.00, 1.00	2017-08-03 13:51:59	GeoTiff
eb446042-0b00-452b-a664-46865f1f0f3	TERRA	ASTER	GDEM_V2	-76.00, 2.00	-75.00, 1.00	2017-08-03 14:00:11	GeoTiff
621f9fe9-4a5e-4c7d-8248-9cea43e024d9	TERRA	ASTER	GDEM_V2	-77.00, 2.00	-76.00, 1.00	2017-08-03 13:27:23	GeoTiff
0d85a62c-19a7-41ea-889e-e1ce931739a	TERRA	ASTER	GDEM_V2	-78.00, 2.00	-77.00, 1.00	2017-08-03 12:17:31	GeoTiff
722584f3-801e-4105-bccc-24c7682e290b	TERRA	ASTER	GDEM_V2	-73.00, 3.00	-72.00, 2.00	2017-08-03 12:12:40	GeoTiff
185a3753-1501-4b92-b91c-a0aa12245d9f	TERRA	ASTER	GDEM_V2	-74.00, 3.00	-73.00, 2.00	2017-08-03 12:21:23	GeoTiff
fc0e6992-3bba-480e-9e07-c1b311a568c4	TERRA	ASTER	GDEM_V2	-75.00, 3.00	-74.00, 2.00	2017-08-03 13:05:41	GeoTiff
6ef27432-cdad-42fc-9d81-d0ba705ee3f5	TERRA	ASTER	GDEM_V2	-76.00, 3.00	-75.00, 2.00	2017-08-03 13:24:51	GeoTiff
d9511aef-8228-4cf0-a0ae-6b72ad032ed7	TERRA	ASTER	GDEM_V2	-77.00, 3.00	-76.00, 2.00	2017-08-03 14:00:04	GeoTiff
76baa595-ebc6-4b6f-8ac3-d0cb970e9d33	TERRA	ASTER	GDEM_V2	-78.00, 3.00	-77.00, 2.00	2017-08-03 12:36:04	GeoTiff
f092d47d-8128-4db4-b50e-ee953acece539	TERRA	ASTER	GDEM_V2	-73.00, 4.00	-72.00, 3.00	2017-08-03 12:46:06	GeoTiff
ae545210-b254-4daf-b211-ba3e567b2005	TERRA	ASTER	GDEM_V2	-74.00, 4.00	-73.00, 3.00	2017-08-03 13:09:50	GeoTiff

Min Latitude:
 Max Latitude:
 Min Longitude:
 Max Longitude:
 Start Date:
 End Date:

[Delete filtered datasets](#) [Create Data Cube request](#)

Ingestion Configuration Builder

Some workflows call for performing time intensive preprocessing of imagery ahead of time and store results for reuse in the future. Ingestion configurations are a data-cube configuration used to specify what sort of preprocessing you're interested in. This tool lets you build a configuration that specifies spatial preprocessing preferences like:

- **Target projection:** requires you specify a coordinate references
- **Resolution:** refers the intended resolution of your pixels. (Expressed in latitude, longitude)
- **Sampling method:** refers to resampling methods like, nearest-neighbors, bi-cubic interpolation.

 Open Data Cube
Home Data Cube Manager Tools Task Manager

Create a Ingestion Configuration

Create a new ingestion configuration using the forms below. Select an existing dataset type to use as the source data and the measurements will be populated with source measurements. Fill in all metadata fields and edit or change measurement attributes and download the resulting .yaml file. Measurements can be renamed using the 'name' field as long as the 'src_varname' stays the same.

Dataset Metadata:

Source Dataset Type

Select an existing source dataset type for this ingestion configuration.

Output Type Name

Enter a description for this new ingested dataset

Description

Enter a description for this new ingested dataset

Data Storage Location

Enter the absolute base path for your storage units

Storage Unit Naming Template

Measurements:

Please enter the name of the measurement. Spaces and special characters are not allowed.

Data Type

Choose your dataset's data type from the provided options.

Nodata Value

Please enter the number that represents nodata in your dataset.

Resampling Method

Choose your dataset's resampling method.

Source Variable Name

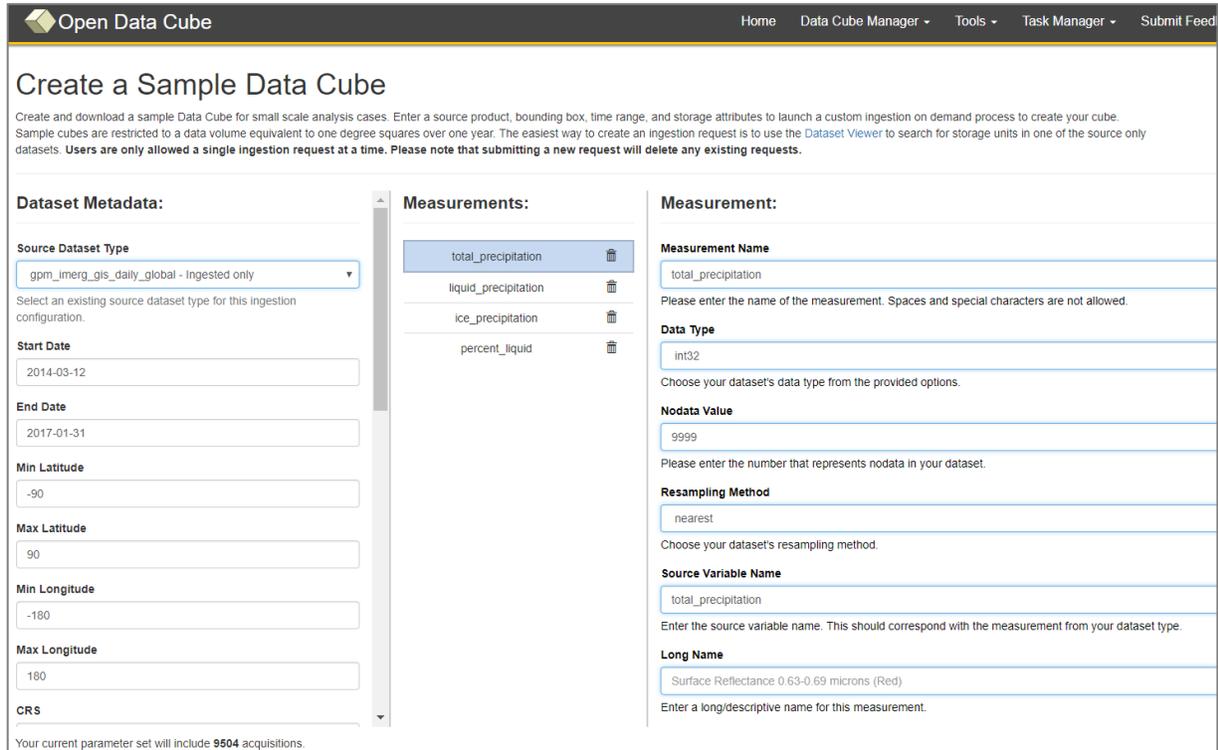
Enter the source variable name. This should correspond with the measurement from your dataset

Long Name

Enter a long/descriptive name for this measurement.

Ingestion on Demand

The ingestion on demand tool offers the option of sub-setting data-cube data holdings and sharing them with other users. After selecting what projection, resolution, and area you want to subset, the system will prepare data for download and provide you with a python download script.



Open Data Cube Home Data Cube Manager Tools Task Manager Submit Feed

Create a Sample Data Cube

Create and download a sample Data Cube for small scale analysis cases. Enter a source product, bounding box, time range, and storage attributes to launch a custom ingestion on demand process to create your cube. Sample cubes are restricted to a data volume equivalent to one degree squares over one year. The easiest way to create an ingestion request is to use the [Dataset Viewer](#) to search for storage units in one of the source only datasets. **Users are only allowed a single ingestion request at a time. Please note that submitting a new request will delete any existing requests.**

Dataset Metadata:

Source Dataset Type

Select an existing source dataset type for this ingestion configuration.

Start Date

End Date

Min Latitude

Max Latitude

Min Longitude

Max Longitude

CRS

Your current parameter set will include 9504 acquisitions.

Measurements:

- total_precipitation
- liquid_precipitation
- ice_precipitation
- percent_liquid

Measurement:

Measurement Name

Please enter the name of the measurement. Spaces and special characters are not allowed.

Data Type

Choose your dataset's data type from the provided options.

Nodata Value

Please enter the number that represents nodata in your dataset.

Resampling Method

Choose your dataset's resampling method.

Source Variable Name

Enter the source variable name. This should correspond with the measurement from your dataset type.

Long Name

Enter a long/descriptive name for this measurement.

Task Manager

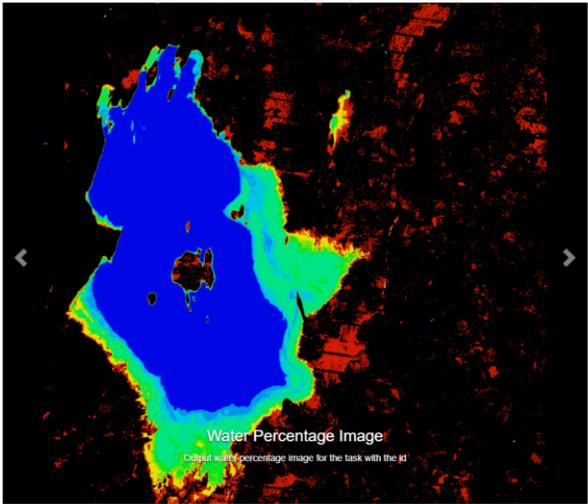
The task manager tool displays a history of completed jobs for an algorithm

Open Data Cube													Home	Data Cube Manager	Tools	Task Manager	Submit Feedback	Logged in as: localuser	Logout
Satellite	Area Id	Time Start	Time End	Latitude Max	Latitude Min	Longitude Max	Longitude Min	Title	Description	Query Type	Animated Product	More Info							
LANDSAT_7	lake_baringo	April 15, 1999	Dec. 26, 2016	0.7485	0.5	36.25	36.0	Water Detection Query	None	Black	None	Details							
LANDSAT_8	uruguay	Feb. 11, 2013	Dec. 26, 2016	-34.4015	-35.0939	-55.7125	-56.823	Water Detection Query	None	Black	None	Details							
LANDSAT_8	uruguay	Feb. 11, 2013	Dec. 26, 2016	-33.915	-34.7028	-55.2095	-56.5617	Water Detection Query	None	Black	None	Details							

Showing 1 to 3 of 3 entries

Previous 1 Next

Clicking a **Details** button will load a page displaying statistics and outputs for a given algorithm.

Open Data Cube													Home	Data Cube Manager	Tools	Task Manager	Submit Feedback	Logged in as: localuser	Logout
Task Details				Task Metadata				 <p>Water Percentage Image Output: water percentage image for the task with the id</p>											
Title	Water Detection Query			Platform	LANDSAT_7														
Description	None			Scene Count	174														
Status	Complete			Pixel Count	856544														
Start Time	04/08/2018 22:43			Task Parameters				Task Outputs											
End Time	04/08/2018 22:43			(Lat, Lon) Min	(0.500000, 36.000000)			Water Percentage Path	View image										
				(Lat, Lon) Max	(0.748500, 36.250000)			Water Observations P...	View image										
				Task Type	Black			Clear Observation Path	View image										
				Animation Type	None			NetCDF Path	Download nc										
								GeoTIFF Path	Download tif										
Scene Metadata																			
12/24/2016																			
Clean pixels											194168								
Water Pixels											29963								
Total Pixels											856544								
Clean Pixel Perc...											22.67%								
11/06/2016																			
Clean pixels											637477								
Water Pixels											163331								
Total Pixels											856544								
Clean Pixel Perc...											74.42%								
10/21/2016																			
Clean pixels											772021								
Water Pixels											197053								
Total Pixels											856544								
Clean Pixel Perc...											90.13%								
10/05/2016																			
Clean pixels											327109								
Water Pixels											71715								

Open Data Cube Resources & Support

ODC Websites:

www.opendatacube.org
www.opendatacube.org/ceos
<https://www.opendatacube.org/brand>

*Primary ODC website
CEOS ODC Project
ODC Brand Assets*

Slack channels – Technical Support

Open Data Cube Slack: An ODC slack frequented by GA, CSIRO, Catapult, Swiss partners
<https://opendatacube.slack.com>

Mexican Data Cube Slack:
<https://mexcube.slack.com/messages>

User Interface Links:

Public Facing User interface:
tinyurl.com/datacubeui

Code Repositories

Open Data Cube Github:
A Github organization for Open Data Cube
<https://github.com/opendatacube>

Open Data Cube Core Github:
A repository that houses code we refer to as the open data-cube. Responsible for managing ingestion, retrieval, and management of data.
<https://github.com/opendatacube/datacube-core>

CEOS Github:
A user account created for CEOS SEO. Contains all AMA/CEOS code approved for release.
<https://github.com/ceos-seo/>

CEOS Data Cube User Interface:
User interface to the cube.
https://github.com/ceos-seo/data_cube_ui

CEOS Data Cube Notebooks:

Inventory of all jupyter notebooks

https://github.com/ceos-seo/data_cube_notebooks

CEOS Data Cube Utilities:

Assorted scripts used in Jupyter notebooks or CEOS UI:

https://github.com/ceos-seo/data_cube_utilities

S1Prepro

Pre-processing repository for sentinel 1.

<https://github.com/ceos-seo/s1prepro>

Data Cube WCS:

A django plugin that implements a subset of of OGC WCS standards, used in conjunction with the open data-cube QGIS plugin:

<https://github.com/ceos-seo/django-datacube-wcs>

Data Cube QGIS Plugin:

A data cube plugin that interfaces with QGIS

<https://github.com/ceos-seo/qgis-datacube-plugin>

AGDC-V2

A clone of agdc-v2. When we refer to installation instructions, we refer to this clone.

<https://github.com/ceos-seo/agdc-v2>



ARDC Workshop Survey

African Regional Data Cube (ARDC) Workshop Survey

A new way to manage satellite data utilizing an open source platform

The African Regional Data Cube was developed by the Committee on Earth Observation Satellites (CEOS) in partnership with the Group on Earth Observations, Amazon Web Services, Office of the Deputy President – Kenya, Strathmore University in Kenya, Global Partnership for Sustainable Development Data, and Analytical Mechanics Associates.

Please provide us the following information to better serve you and ODC community.

1. What is your country of nationality?

2. Identify the category/categories that most closely identify your current role?

- | | | | | |
|---------------------------------------------|--------------------------------------------------|--------------------------------------------------------------------------------------|-------------------------------------------------|----------------------------------------------|
| <input type="checkbox"/> Professional - GIS | <input type="checkbox"/> Professional - Engineer | <input type="checkbox"/> Professional – Software | <input type="checkbox"/> Professional – Analyst | <input type="checkbox"/> Government Official |
| <input type="checkbox"/> Faculty (research) | <input type="checkbox"/> Faculty (teaching) | <input type="checkbox"/> Other educator (e.g. lecturer, high school science teacher) | <input type="checkbox"/> Graduate student | <input type="checkbox"/> Undergraduate |

3. Please indicate your level of experience with remote sensing systems.

- | | | | |
|--------------------------------------|---------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| <input type="radio"/> Little or none | <input type="radio"/> I know something about the field, but don't work/study/teach directly in it | <input type="radio"/> I've recently begun working/studying /teaching in the field | <input type="radio"/> I have extensive experience in the field (>3 years) |
|--------------------------------------|---------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|---------------------------------------------------------------------------|

4. Which of the following applications are you most interested in?

- | | | | | |
|------------------------------------------|-------------------------------------------|----------------------------------------------------|----------------------------------------------|----------------------------------------------------------|
| <input type="checkbox"/> Custom Mosaics | <input type="checkbox"/> Change Detection | <input type="checkbox"/> Land Cover Classification | <input type="checkbox"/> Disaster Monitoring | <input type="checkbox"/> Agriculture and Crop Monitoring |
| <input type="checkbox"/> Water Detection | | | | |

5. Which of the following satellite imagery/data are you most interested in?

- | | | | | |
|----------------------------------|-----------------------------------|--------------------------------------|------------------------------|--------------------------------|
| <input type="checkbox"/> Landsat | <input type="checkbox"/> Sentinel | <input type="checkbox"/> ALOS-PALSAR | <input type="checkbox"/> GPM | <input type="checkbox"/> MODIS |
| <input type="checkbox"/> DEM | | | | |

Others, please specify



ARDC Workshop Survey

African Regional Data Cube (ARDC) Workshop Survey

A new way to manage satellite data utilizing an open source platform

6. How did you learn about the Open Data Cube (ODC) workshop?

7. Which of the following statements describes your intention(s) for participating in the Open Data Cube (ODC) workshop?

- I want to expand my general knowledge. I have a strong personal interest in ODC. I am curious to know what ODC is all about. I want to use ODC as a tool in my work.

8. How will participating in the Open Data Cube (ODC) workshop help your mission? Please provide any other feedback here. Thank you!

Please submit this sheet to us directly -or- provide your feedback online at:

<https://www.surveymonkey.com/r/QZBL77J>

Thank you for your participation and feedback!

Appendix:

Jupyter Python Notebooks

Task A:

Cloud-free Mosaics and K-means Clustering

ARDC Training: Python Notebooks

Task-A: Cloud-free Mosaics and K-means Clustering

Import the Datacube Configuration

```
import datacube
dc = datacube.Datacube(app = 'my_app', config = '/home/localuser/.datacube.conf')
```

Browse the available Data Cubes

```
list_of_products = dc.list_products()
netCDF_products = list_of_products[list_of_products['format'] == 'NetCDF']
# netCDF_products
```

Pick a product

Use the platform and product names from the previous block to select a Data Cube.

```
import utils.data_cube_utilities.data_access_api as dc_api
api = dc_api.DataAccessApi(config = '/home/localuser/.datacube.conf')

# Change the data platform and data cube here

platform = "LANDSAT_7"
# platform = "LANDSAT_8"

# product = "ls7_ledaps_ghana"
# product = "ls7_ledaps_kenya"
# product = "ls7_ledaps_senegal"
# product = "ls7_ledaps_sierra_leone"
# product = "ls7_ledaps_tanzania"
product = "ls7_ledaps_vietnam"

# Get Coordinates
coordinates = api.get_full_dataset_extent(platform = platform, product = product)
```

Display Latitude-Longitude and Time Bounds of the Data Cube

```
latitude_extents = (min(coordinates['latitude'].values),max(coordinates['latitude'].values))
print( latitude_extents )
```

```
(9.1764253745784181, 13.964805165051667)
```

```
longitude_extents = (min(coordinates['longitude'].values),max(coordinates['longitude'].values))
print( longitude_extents )
```

```
(102.40430421277932, 108.93092407802477)
```

```
time_extents = (min(coordinates['time'].values),max(coordinates['time'].values))
print( time_extents )
```

```
(numpy.datetime64('1999-09-08T03:13:19.000000000'), numpy.datetime64('2016-12-29T03:10:00.000000000'))
```

Visualize Data Cube Region

```
## The code below renders a map that can be used to orient yourself with the region.
from utils.data_cube_utilities.dc_display_map import display_map
display_map(latitude = latitude_extents, longitude = longitude_extents)
```



Pick a smaller analysis region and display that region

Try to keep your region to less than 0.2-deg x 0.2-deg for rapid processing Pick the time extents for your mosaic product (keep to 1 year or less)

```
## Vietnam - Central Lam Dong Province ##
longitude_extents = (107.80, 108.00)
latitude_extents = (11.70, 11.90)

time_extents = ('2015-01-01', '2016-01-01')
print ( time_extents )

('2015-01-01', '2016-01-01')
```

```
display_map(latitude = latitude_extents, longitude = longitude_extents)
```



Load the dataset and the required spectral bands or other parameters

After loading, you will view the Xarray dataset. Notice the dimensions represent the number of pixels in your latitude and longitude dimension as well as the number of time slices (time) in your time series.

```
landsat_dataset = dc.load(latitude = latitude_extents,
                          longitude = longitude_extents,
                          platform = platform,
                          time = time_extents,
                          product = product,
                          measurements = ['red', 'green', 'blue', 'nir', 'swir1', 'swir2',
                                         'pixel_qa'])
```

```
landsat_dataset
#view the dimensions and sample content from the cube
```

```
<xarray.Dataset>
Dimensions:   (latitude: 743, longitude: 743, time: 19)
Coordinates:
  * time      (time) datetime64[ns] 2015-01-09T03:06:13 2015-01-25T03:06:16
  ...
  * latitude  (latitude) float64 11.9 11.9 11.9 11.9 11.9 11.9 11.9 11.9 ...
  * longitude (longitude) float64 107.8 107.8 107.8 107.8 107.8 107.8 107.8
  ...
Data variables:
  red        (time, latitude, longitude) int16 514 457 382 419 456 437 587
  ...
  green      (time, latitude, longitude) int16 482 421 381 401 400 360 481
  ...
  blue       (time, latitude, longitude) int16 296 257 238 276 276 256 294
  ...
  nir        (time, latitude, longitude) int16 2457 2331 2288 2288 1865 ...
  swir1      (time, latitude, longitude) int16 1774 1826 1568 1697 1671 ...
  swir2      (time, latitude, longitude) int16 1003 1057 867 894 894 676
  ...
  pixel_qa   (time, latitude, longitude) int32 66 66 66 66 66 66 66 72 72
  ...
Attributes:
  crs:       EPSG:4326
```

Display Example Images

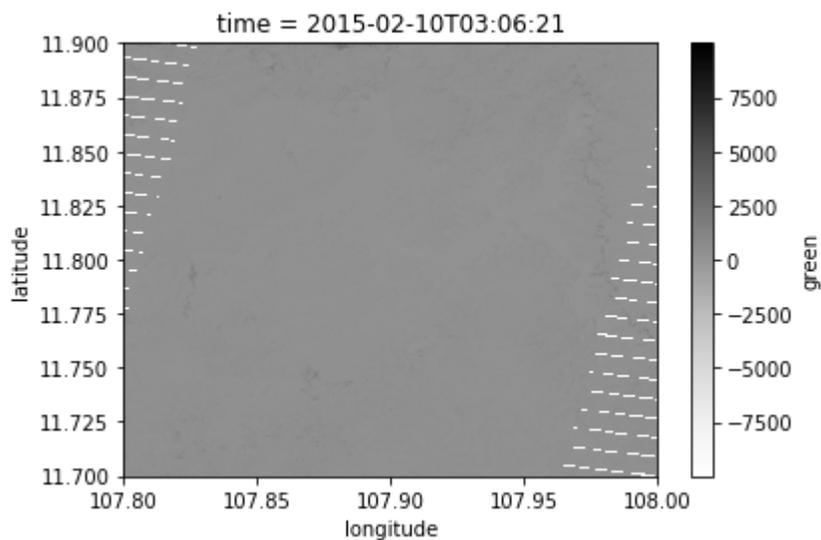
Single band visualization

For a quick inspection, let's look at two images. The first image will allow the selection of any band (red, blue, green, nir, swir1, swir2) to produce a grey-scale image of any band. The second image will mask clouds with bright red on an RGB image. Select the desired acquisition (time slice) in the block below. You can select from 1 to #, where the max value is the number of time slices noted in the block above. Change the comment statements below to select the bands for the first image.

```
acquisition_number = 2
# select an acquisition number from 1 to "time" using the array limits above
```

```
%matplotlib inline
#Landsat_dataset.red.isel(time = acquisition_number).plot(cmap = "Greys")
landsat_dataset.green.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.blue.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.nir.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.swir1.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.swir2.isel(time = acquisition_number).plot(cmap = "Greys")
```

<matplotlib.collections.QuadMesh at 0x7f2b8e786fd0>



Define Cloud Masking Function

Removes clouds and cloud shadows based on the Landsat pixel QA information This is only for reference ... nothing to modify here

```
import numpy as np

def generate_cloud_mask(dataset, include_shadows = False):
    #Create boolean Masks for clear and water pixels
    clear_pixels = dataset.pixel_qa.values == 2 + 64
    water_pixels = dataset.pixel_qa.values == 4 + 64
    shadow_pixels= dataset.pixel_qa.values == 8 + 64

    a_clean_mask = np.logical_or(clear_pixels, water_pixels)

    if include_shadows:
        a_clean_mask = np.logical_or(a_clean_mask, shadow_pixels)

    return np.invert(a_clean_mask)

def remove_clouds(dataset, include_shadows = False):
    #Create boolean Masks for clear and water pixels
    clear_pixels = dataset.pixel_qa.values == 2 + 64
    water_pixels = dataset.pixel_qa.values == 4 + 64
    shadow_pixels= dataset.pixel_qa.values == 8 + 64

    a_clean_mask = np.logical_or(clear_pixels, water_pixels)

    if include_shadows:
        a_clean_mask = np.logical_or(a_clean_mask, shadow_pixels)

    return dataset.where(a_clean_mask)
```

Mask clouds from your selected acquisition and visualize the scene and mask

Now we will look at two RGB images where the second image includes the cloud, cloud shadow and no data mask in RED. Also, the scene is the same as the acquisition selected above.

```
cloud_mask = generate_cloud_mask(landsat_dataset)
cloudless = remove_clouds(landsat_dataset) #landsat_dataset.where(image_is_clean)
```

```

import matplotlib.pyplot as plt
import matplotlib.cm as color_maps
from time import time
import numpy as np

# Change the bands (RGB) here if you want other false color combinations
def rgb(dataset, at_index = 0, bands = ['red', 'green', 'blue'], paint_on_mask = []):

    rgb = np.stack([dataset[bands[0]], dataset[bands[1]], dataset[bands[2]]], axis = -1)
    max_possible = 3500
    rgb = rgb.astype(np.float32)
    rgb[rgb<0] = 0
    rgb[rgb > max_possible] = max_possible
    rgb *= 255.0/rgb.max()

    rgb = rgb.astype(int)
    rgb = rgb.astype(np.float32)
    rgb = 255-rgb

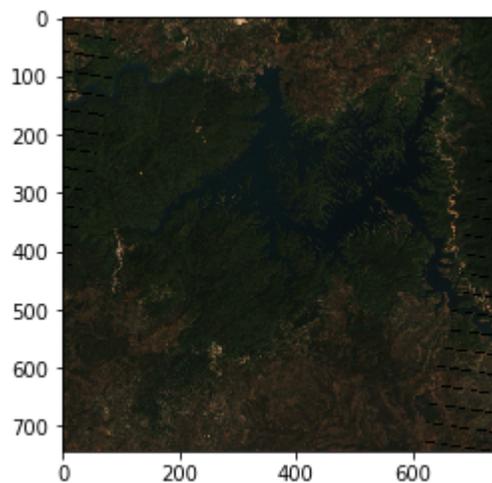
    rgb[rgb > 254] = 254
    rgb[rgb < 1] = 1

    for mask, color in paint_on_mask:
        rgb[mask] = np.array([256,256,256]) - np.array(color).astype(np.int16)

    if 'time' in dataset:
        plt.imshow((rgb[at_index]))
    else:
        plt.imshow(rgb)

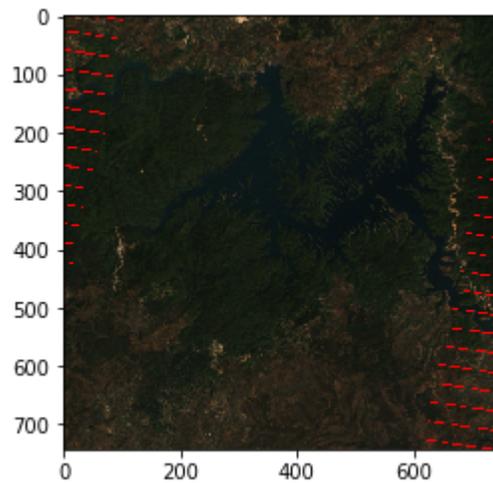
```

```
rgb(landsat_dataset,at_index = acquisition_number)
```



```
red = [255,0,0]
```

```
rgb(landsat_dataset,at_index = acquisition_number,paint_on_mask = [(cloud_mask, red)])
```



Cleaning up the clouds and creating a cloud-free mosaic

Remember that this process will filter clouds from the entire time series stack

Most Recent Pixel Mosaic

Masks clouds from imagery and uses the most recent cloud-free pixels.

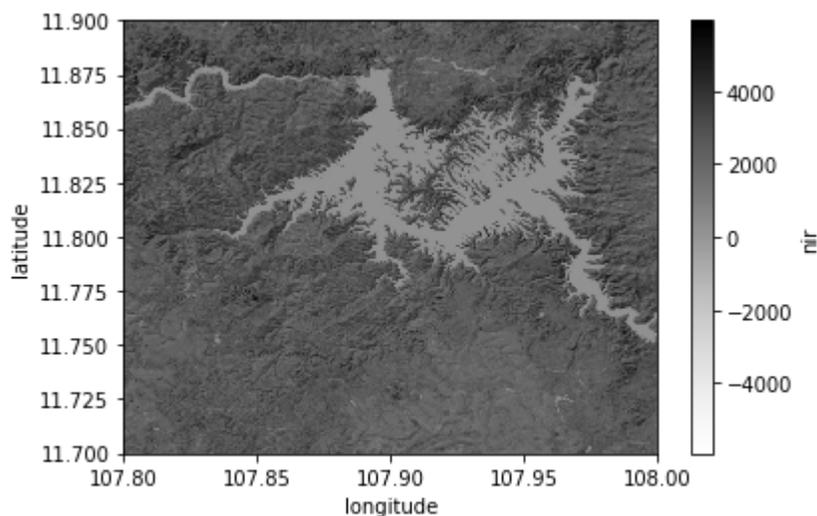
```
from utils.data_cube_utilities.dc_mosaic import create_mosaic

def mrf_mosaic(dataset):
    # The mask here is based on pixel_qa products. It comes bundled in with most Landsat P
    # roducts.
    cloud_free_boolean_mask = np.invert(generate_cloud_mask(dataset))
    return create_mosaic(dataset, clean_mask = cloud_free_boolean_mask)
```

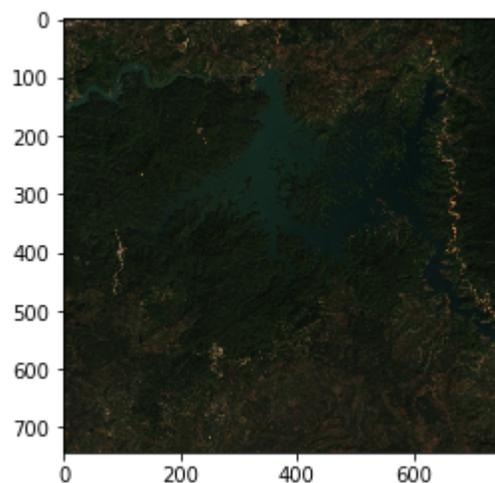
```
recent_composite = mrf_mosaic(landsat_dataset)
```

```
recent_composite.nir.plot(cmap = "Greys")
```

```
<matplotlib.collections.QuadMesh at 0x7f2b8efe82e8>
```



```
rgb(recent_composite)
```



Median Mosaic

Masks clouds from imagery using the median valued cloud-free pixels in the time series

```
from utils.data_cube_utilities.dc_mosaic import create_median_mosaic
```

```
def median_mosaic(dataset):
```

```
    # The mask here is based on pixel_qa products. It comes bundled in with most Landsat P
    # products.
```

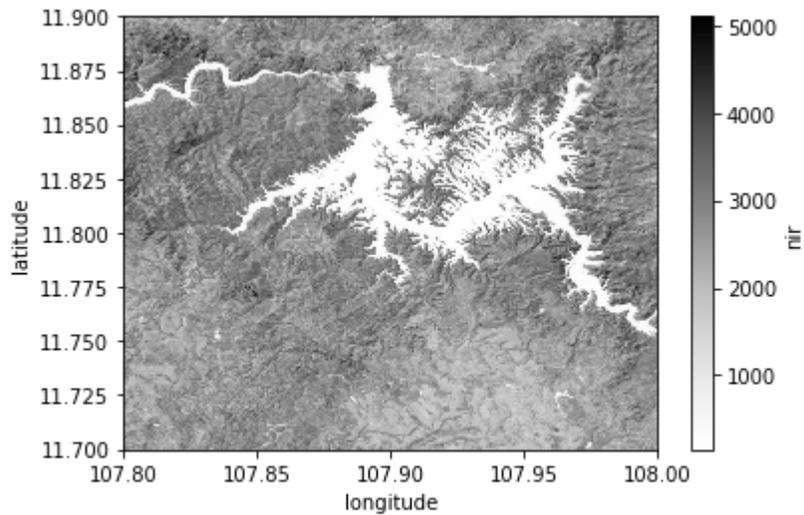
```
    cloud_free_boolean_mask = np.invert(generate_cloud_mask(dataset))
```

```
    return create_median_mosaic(dataset, clean_mask = cloud_free_boolean_mask)
```

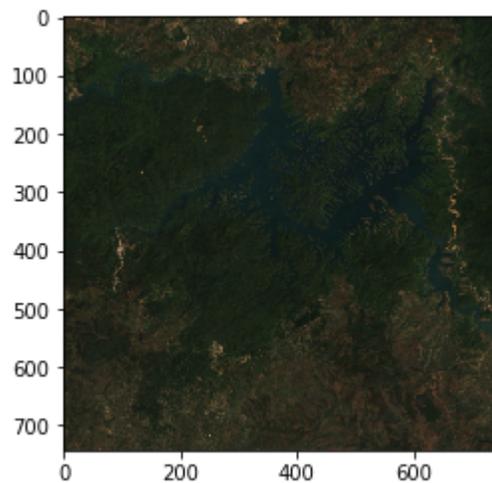
```
median_composite = median_mosaic(landsat_dataset)
```

```
median_composite.nir.plot(cmap = "Greys")
```

```
<matplotlib.collections.QuadMesh at 0x7f2b901b3d30>
```



```
rgb(median_composite)
```



Select bands used for clustering

```
#cluster_bands = ['red', 'green', 'blue', 'swir1', 'swir2', 'nir']
cluster_bands = ['red', 'green', 'blue', 'swir1']
```

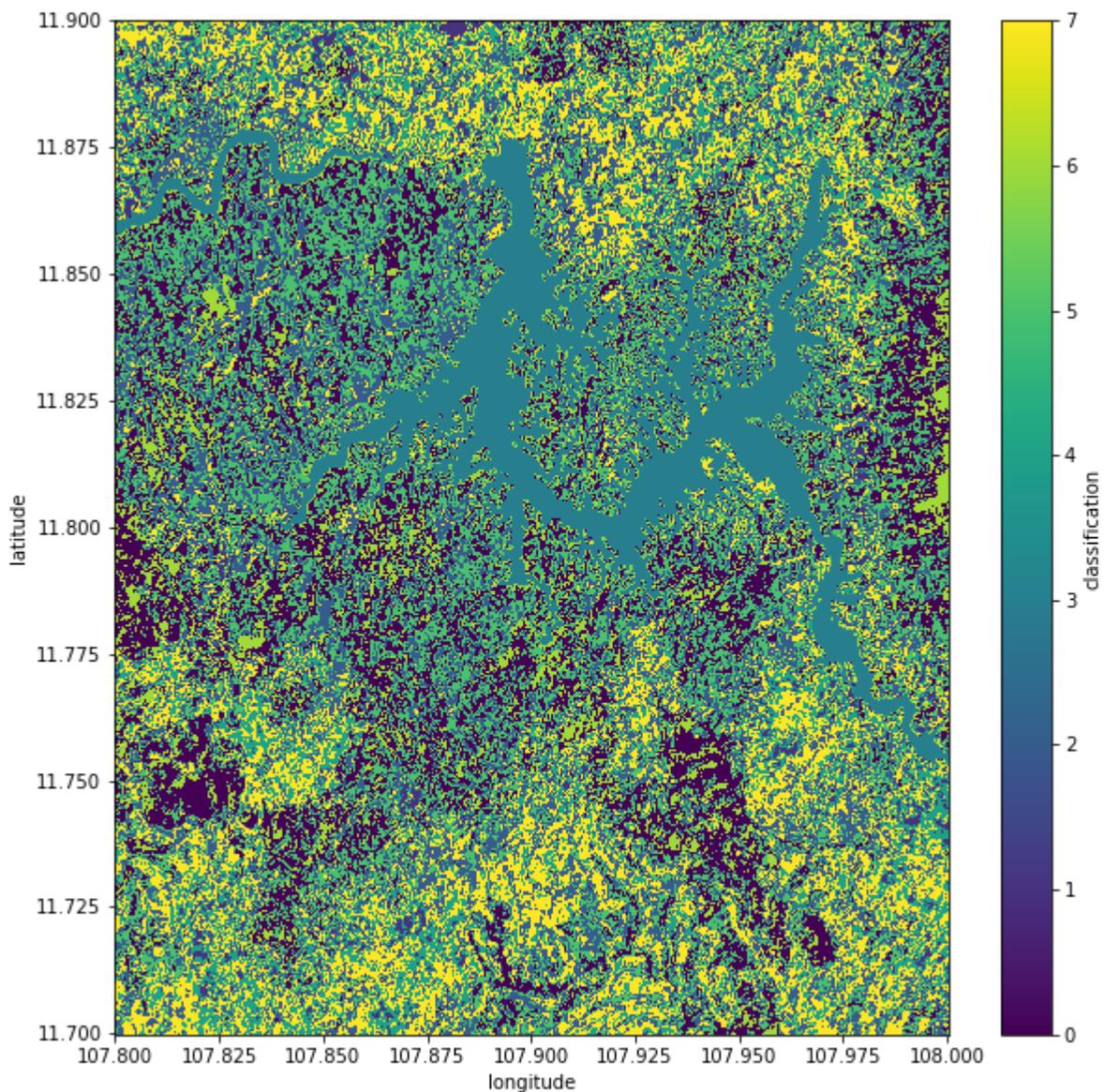
Perform K-Means clustering and view the output

```
def figure_ratio(ds, fixed_width = 10):
    width = fixed_width
    height = len(ds.latitude) * (fixed_width / len(ds.longitude))
    return (width, height)
```

```
from utils.dc_clustering import kmeans_cluster_dataset

# change the number of clusters in the line below, as desired
# this example uses the "median composite" image from above
classification_x = kmeans_cluster_dataset(median_composite, cluster_bands, n_clusters=8)
# plot the k-mean classification result
classification_x.classification.plot(figsize = figure_ratio(classification_x))
```

```
<matplotlib.collections.QuadMesh at 0x7f2b4b03c6d8>
```



Task B:

Water Extent (WOFS) and Water Quality (TSM)

ARDC Training: Python Notebooks

Task-B: Water Extent (WOFS) and Water Quality (TSM)

Import the Datacube Configuration

```
import datacube
dc = datacube.Datacube(app = 'my_app', config = '/home/localuser/.datacube.conf')
```

Browse the available Data Cubes

```
list_of_products = dc.list_products()
netCDF_products = list_of_products[list_of_products['format'] == 'NetCDF']
netCDF_products
```

Pick a product

Use the platform and product names from the previous block to select a Data Cube.

```
import utils.data_cube_utilities.data_access_api as dc_api
api = dc_api.DataAccessApi(config = '/home/localuser/.datacube.conf')

# Change the data platform and data cube here

platform = "LANDSAT_7"
# platform = "LANDSAT_8"

# product = "ls7_ledaps_ghana"
# product = "ls7_ledaps_kenya"
# product = "ls7_ledaps_senegal"
# product = "ls7_ledaps_sierra_leone"
# product = "ls7_ledaps_tanzania"
product = "ls7_ledaps_vietnam"

# Get Coordinates
coordinates = api.get_full_dataset_extent(platform = platform, product = product)
```

Display Latitude-Longitude and Time Bounds of the Data Cube

```
latitude_extents = (min(coordinates['latitude'].values),max(coordinates['latitude'].values))
print( latitude_extents )

(9.1764253745784181, 13.964805165051667)
```

```
longitude_extents = (min(coordinates['longitude'].values),max(coordinates['longitude'].values))
print( longitude_extents )

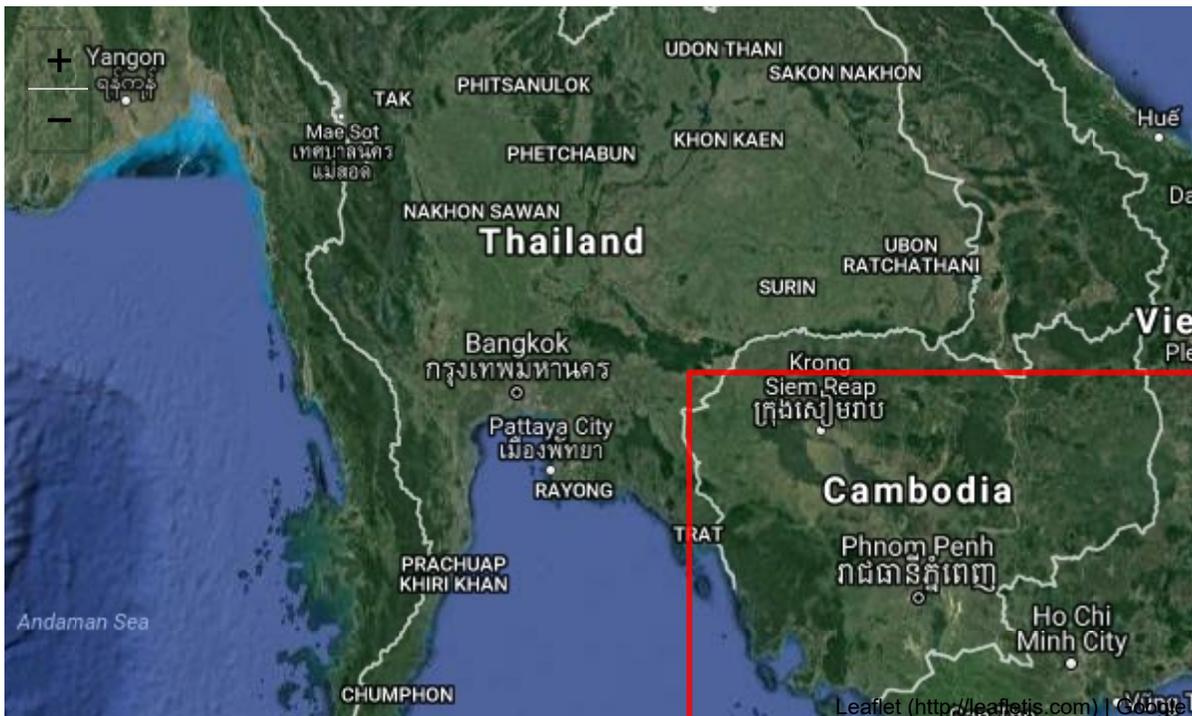
(102.40430421277932, 108.93092407802477)
```

```
time_extents = (min(coordinates['time'].values),max(coordinates['time'].values))
print( time_extents )

(numpy.datetime64('1999-09-08T03:13:19.000000000'), numpy.datetime64('2016-12-29T03:10:00.000000000'))
```

Visualize Data Cube Region

```
## The code below renders a map that can be used to orient yourself with the region.
from utils.data_cube_utilities.dc_display_map import display_map
display_map(latitude = latitude_extents, longitude = longitude_extents)
```

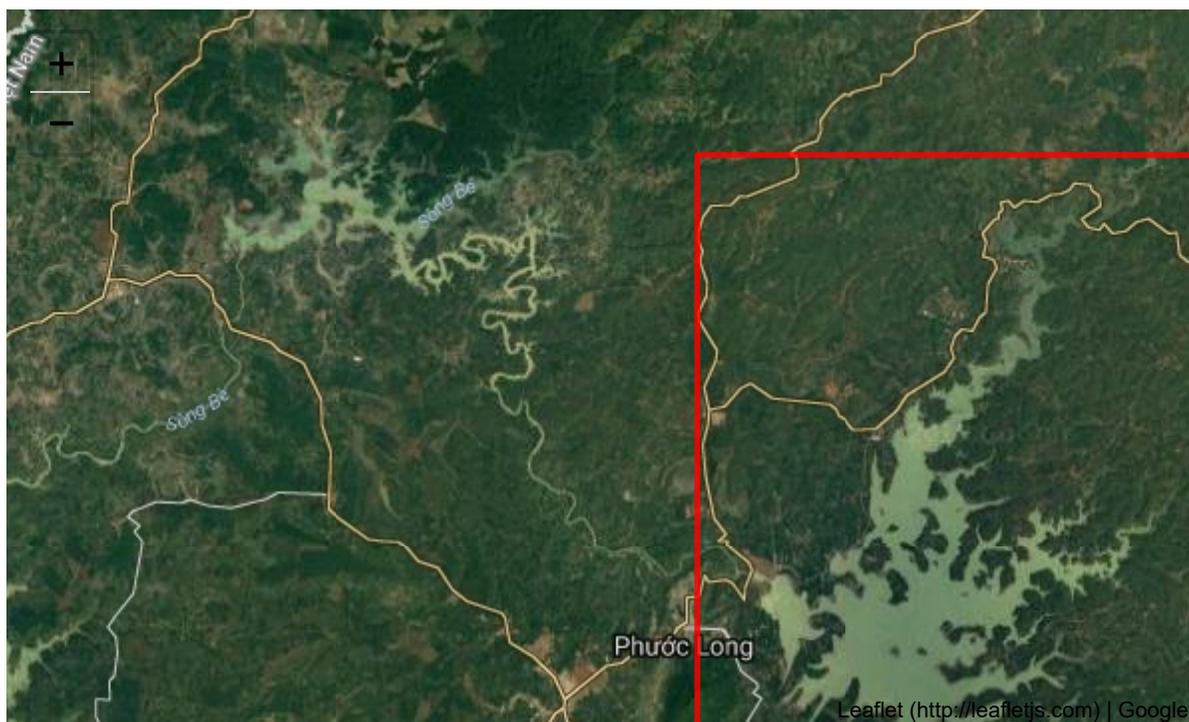


Pick a smaller analysis region and display that region

Try to keep your region to less than 0.2-deg x 0.2-deg for rapid processing. You can click on the map above to find the Lat-Lon coordinates of any location. You will want to identify a region with an inland water body. Pick a time window of a few months so we can pick out some clear pixels and plot the water.

```
## Vietnam - Central Lam Dong Province ##  
longitude_extents = (107.0, 107.2)  
latitude_extents = (11.7, 12.0)  
  
time_extents = ('2010-01-01', '2015-01-01')  
print ( time_extents )  
  
('2010-01-01', '2015-01-01')
```

```
display_map(latitude = latitude_extents, longitude = longitude_extents)
```



Load the dataset and the required spectral bands or other parameters

After loading, you will view the Xarray dataset. Notice the dimensions represent the number of pixels in your latitude and longitude dimension as well as the number of time slices (time) in your time series.

```
landsat_dataset = dc.load(latitude = latitude_extents,
                          longitude = longitude_extents,
                          platform = platform,
                          time = time_extents,
                          product = product,
                          measurements = ['red', 'green', 'blue', 'nir', 'swir1', 'swir2',
                                          'pixel_qa'])
```

```
landsat_dataset
#view the dimensions and sample content from the cube
```

```
<xarray.Dataset>
Dimensions:    (latitude: 1114, longitude: 743, time: 57)
Coordinates:
  * time       (time) datetime64[ns] 2010-01-11T02:59:07 2010-01-27T02:59:20
  ...
  * latitude   (latitude) float64 12.0 12.0 12.0 12.0 12.0 12.0 12.0 12.0 ...
  * longitude  (longitude) float64 107.0 107.0 107.0 107.0 107.0 107.0 107.0
  ...
Data variables:
  red          (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  green        (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  blue         (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  nir          (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  swir1        (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  swir2        (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  pixel_qa     (time, latitude, longitude) int32 1 1 1 1 1 1 1 1 1 1 1 1 1
  ...
Attributes:
  crs:         EPSG:4326
```

Display Example Images

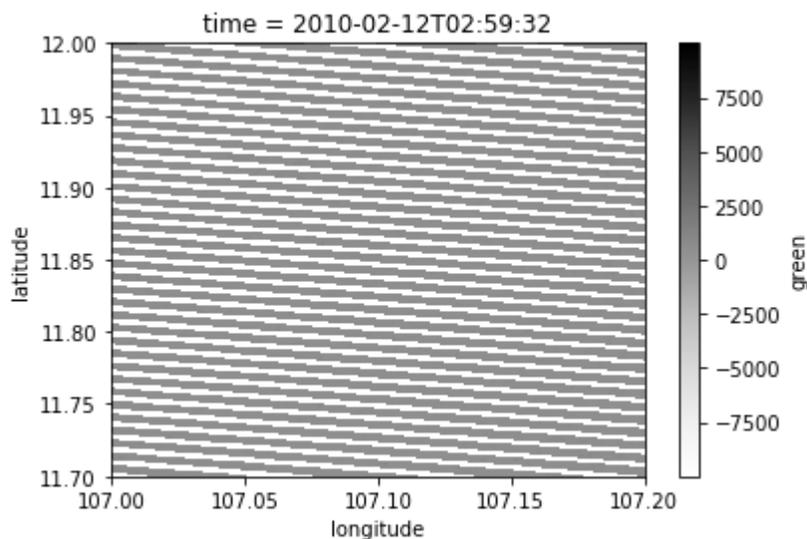
Single band visualization

For a quick inspection, let's look at one image. The code will allow the selection of any band (red, blue, green, nir, swir1, swir2) to produce a grey-scale image. Select the desired acquisition (time slice) in the block below. You can select from 1 to #, where the max value is the number of time slices noted in the block above. Change the comment statements below to select the bands for the first image.

```
acquisition_number = 2
# select an acquisition number from 1 to "time" using the array limits above
```

```
%matplotlib inline
#Landsat_dataset.red.isel(time = acquisition_number).plot(cmap = "Greys")
landsat_dataset.green.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.blue.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.nir.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.swir1.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.swir2.isel(time = acquisition_number).plot(cmap = "Greys")
```

<matplotlib.collections.QuadMesh at 0x7fb86f1b90f0>



Define Cloud Masking Function

Removes clouds and cloud shadows based on the Landsat pixel QA information This is only for reference ... nothing to modify here

```
import numpy as np

def generate_cloud_mask(dataset, include_shadows = False):
    #Create boolean Masks for clear and water pixels
    clear_pixels = dataset.pixel_qa.values == 2 + 64
    water_pixels = dataset.pixel_qa.values == 4 + 64
    shadow_pixels= dataset.pixel_qa.values == 8 + 64

    a_clean_mask = np.logical_or(clear_pixels, water_pixels)

    if include_shadows:
        a_clean_mask = np.logical_or(a_clean_mask, shadow_pixels)

    return np.invert(a_clean_mask)

def remove_clouds(dataset, include_shadows = False):
    #Create boolean Masks for clear and water pixels
    clear_pixels = dataset.pixel_qa.values == 2 + 64
    water_pixels = dataset.pixel_qa.values == 4 + 64
    shadow_pixels= dataset.pixel_qa.values == 8 + 64

    a_clean_mask = np.logical_or(clear_pixels, water_pixels)

    if include_shadows:
        a_clean_mask = np.logical_or(a_clean_mask, shadow_pixels)

    return dataset.where(a_clean_mask)
```

```
cloud_mask = generate_cloud_mask(landsat_dataset)
cloudless = remove_clouds(landsat_dataset) #Landsat_dataset.where(image_is_clean)
```

Set up plotting function (to be used later) Nothing to modify here

```

import matplotlib.pyplot as plt
import matplotlib.cm as color_maps
from time import time
import numpy as np

# Change the bands (RGB) here if you want other false color combinations
def rgb(dataset, at_index = 0, bands = ['red', 'green', 'blue'], paint_on_mask = []):

    rgb = np.stack([dataset[bands[0]], dataset[bands[1]], dataset[bands[2]]], axis = -1)
    max_possible = 3500
    rgb = rgb.astype(np.float32)
    rgb[rgb<0] = 0
    rgb[rgb > max_possible] = max_possible
    rgb *= 255.0/rgb.max()

    rgb = rgb.astype(int)
    rgb = rgb.astype(np.float32)
    rgb = 255-rgb

    rgb[rgb > 254] = 254
    rgb[rgb < 1] = 1

    for mask, color in paint_on_mask:
        rgb[mask] = np.array([256,256,256]) - np.array(color).astype(np.int16)

    if 'time' in dataset:
        plt.imshow((rgb[at_index]))
    else:
        plt.imshow(rgb)

```

Most Recent Pixel Mosaic

Masks clouds from imagery and uses the most recent cloud-free pixels.

```

from utils.data_cube_utilities.dc_mosaic import create_mosaic

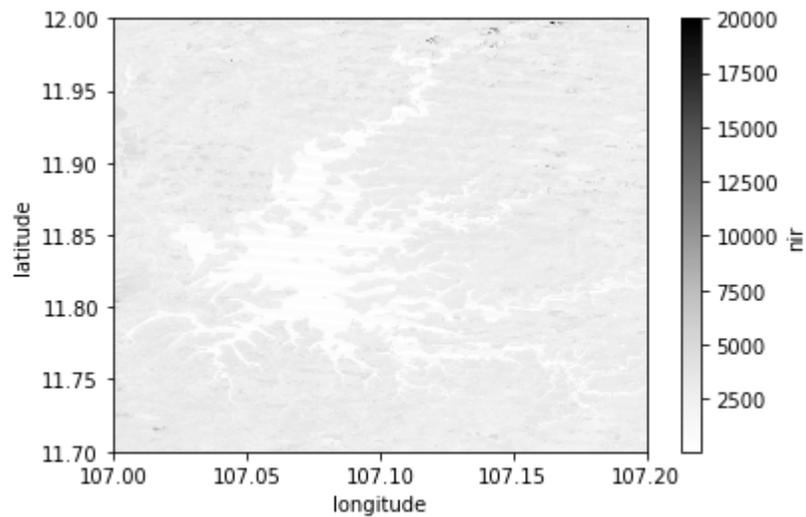
def mrf_mosaic(dataset):
    # The mask here is based on pixel_qa products. It comes bundled in with most Landsat P
    # roducts.
    cloud_free_boolean_mask = np.invert(generate_cloud_mask(dataset))
    return create_mosaic(dataset, clean_mask = cloud_free_boolean_mask)

```

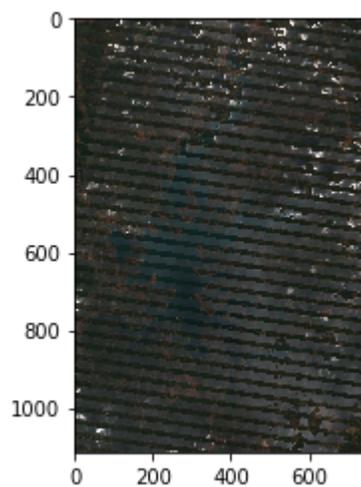
```
recent_composite = mrf_mosaic(landsat_dataset)
```

```
recent_composite.nir.plot(cmap = "Greys")
```

```
<matplotlib.collections.QuadMesh at 0x7fb86f6e4198>
```



```
rgb(recent_composite)
```



Plot WOFs water detection results

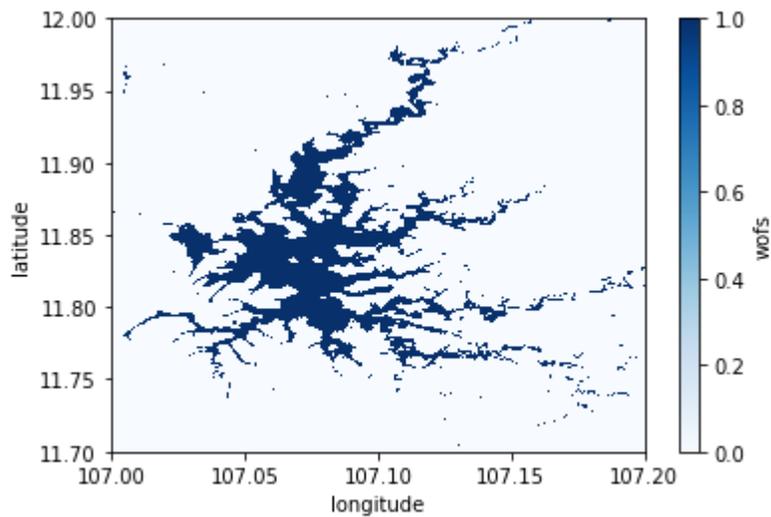
This example uses the Australian Water Detection from Space (WOFs) algorithm for water detection. The base image will use a most-recent pixel composite (from above). When reviewing the results, 1=water, 0=no water.

```
from utils.data_cube_utilities.dc_water_classifier import wofs_classify
```

```
water_classification = wofs_classify(recent_composite, clean_mask = np.ones(recent_composite.pixel_qa.shape).astype(np.bool), mosaic = True)
```

```
water_classification.wofs.plot(cmap='Blues')
```

```
<matplotlib.collections.QuadMesh at 0x7fb8724cf8d0>
```



Plot NDWI water detection results

This example uses the Normalized Difference Water Index (NDWI) which is a spectral "index" that correlates well with the existence of water.

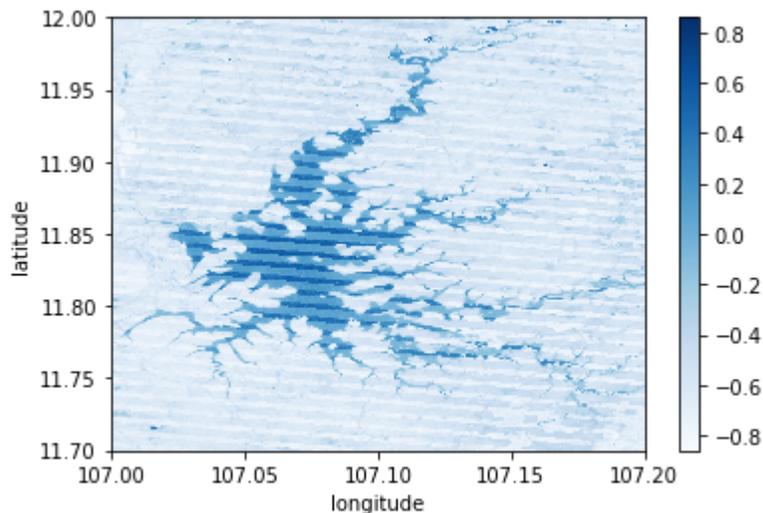
$$NDWI = \frac{GREEN - NIR}{GREEN + NIR}$$

```
def NDWI(dataset):
    return (dataset.green - dataset.nir)/(dataset.green + dataset.nir)
```

```
ndwi = NDWI(recent_composite) # High Concentrations of Water - Blues
```

```
(ndwi).plot(cmap = "Blues")
```

```
<matplotlib.collections.QuadMesh at 0x7fb8700afe10>
```



Plot TSM water quality results

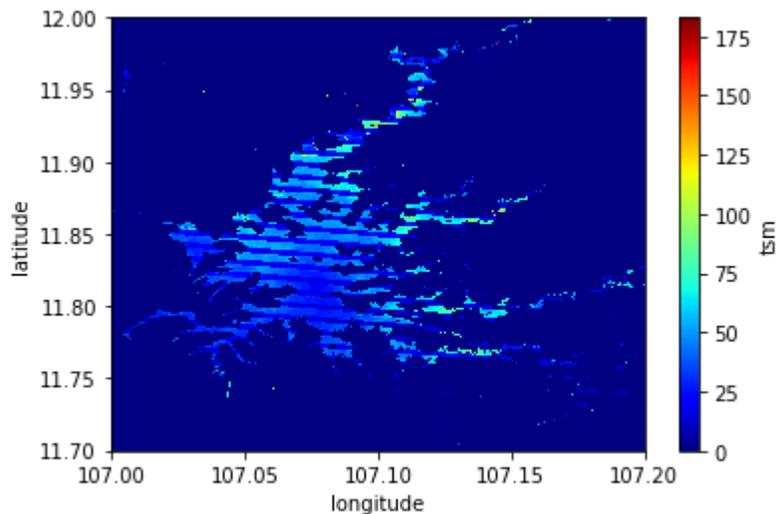
This example uses the Australian Total Suspended Matter (TSM) algorithm. The TSM value is the mean over the entire time range. This parameter is a measure of the particulate matter in water and is often a proxy for water quality.

```
from utils.data_cube_utilities.dc_water_quality import tsm
```

```
mask_that_only_includes_water_pixels = water_classification.wofs == 1
tsm_dataset = tsm(recent_composite, clean_mask = mask_that_only_includes_water_pixels )
```

```
tsm_dataset.tsm.plot(cmap = "jet")
```

```
<matplotlib.collections.QuadMesh at 0x7fb86f75f550>
```



Time Series Water Detection Analysis

Time series output of the Australian Water Detection from Space (WOFs) results. The results show the percent of time that a pixel is classified as water over the entire time series. BLUE = frequent water, RED = infrequent water.

```
ts_water_classification = wofs_classify(landsat_dataset,clean_mask = np.invert(cloud_mask
))
```

```
# Apply nan to no_data values
ts_water_classification = ts_water_classification.where(ts_water_classification != -9999)

##Time series aggregation that ignores nan values.
water_classification_percentages = (ts_water_classification.mean(dim = ['time']) * 100).wo
fs.rename('water_classification_percentages')
```

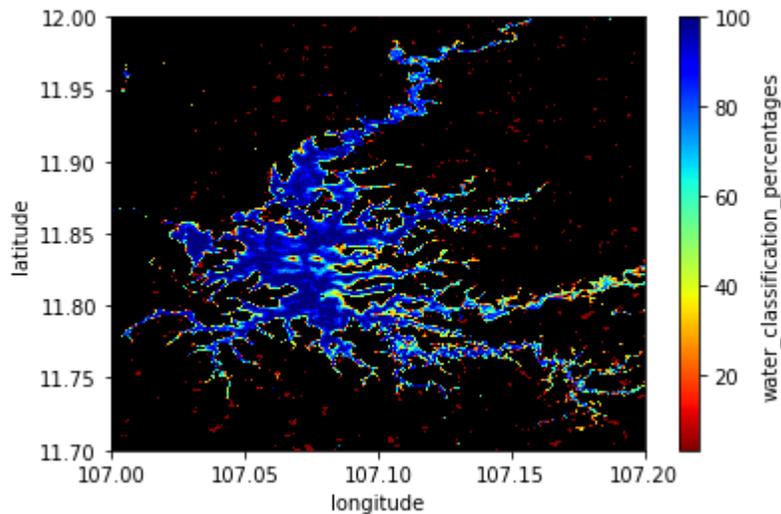
```

## import color-scheme and set nans to black
from matplotlib.cm import jet_r as jet_r
jet_r.set_bad('black',1)

## apply nan to percentage values that aren't greater than 0, then plot
water_classification_percentages\
    .where(water_classification_percentages > 0)\
    .plot(cmap = jet_r)

```

<matplotlib.collections.QuadMesh at 0x7fb86f65be80>



Create a WOFS plot for a single pixel

First select the Lat-Lon position. Then the code will find the closest pixel in the dataset using a "nearest neighbor" selection.

```

pixel_lat = 11.84
pixel_lon = 107.09

```

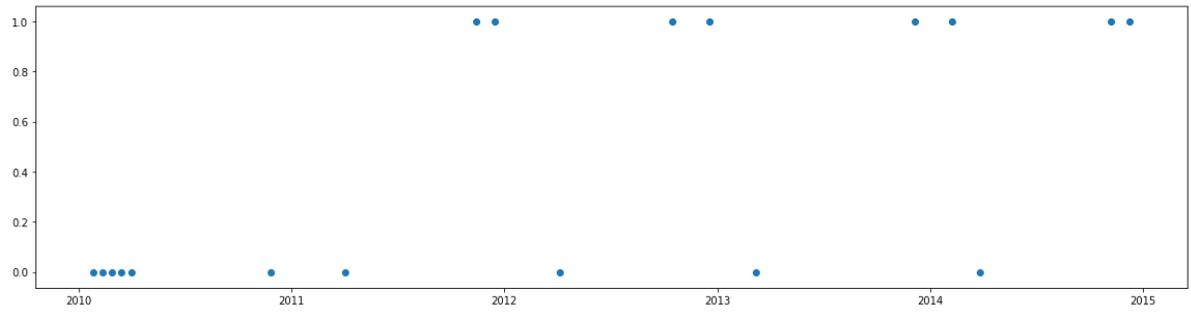
```

pixel = ts_water_classification.sel( latitude = pixel_lat,
                                   longitude = pixel_lon,
                                   method = 'nearest') # nearest neighbor selection

```

```
plt.figure(figsize = (20,5))  
plt.scatter(pixel.time.values, pixel.wofs.values)
```

<matplotlib.collections.PathCollection at 0x7fb86f524240>



Task C:

Fractional Cover (FC)

and Spectral Indices (NDBI, NDVI)

ARDC Training: Python Notebooks

Task-C: Fractional Cover (FC) and Spectral Indices (NDBI and NDVI)

Import the Datacube Configuration

```
import datacube
dc = datacube.Datacube(app = 'my_app', config = '/home/localuser/.datacube.conf')
```

```
# Suppress Warning
import warnings
warnings.filterwarnings('ignore')
```

Browse the available Data Cubes

```
list_of_products = dc.list_products()
netCDF_products = list_of_products[list_of_products['format'] == 'NetCDF']
netCDF_products
```

Pick a product

Use the platform and product names from the previous block to select a Data Cube.

```
import utils.data_cube_utilities.data_access_api as dc_api
api = dc_api.DataAccessApi(config = '/home/localuser/.datacube.conf')

# Change the data platform and data cube here

platform = "LANDSAT_7"
# platform = "LANDSAT_8"

# product = "ls7_ledaps_ghana"
# product = "ls7_ledaps_kenya"
# product = "ls7_ledaps_senegal"
# product = "ls7_ledaps_sierra_leone"
# product = "ls7_ledaps_tanzania"
product = "ls7_ledaps_vietnam"

# Get Coordinates
coordinates = api.get_full_dataset_extent(platform = platform, product = product)
```

Display Latitude-Longitude and Time Bounds of the Data Cube

```
latitude_extents = (min(coordinates['latitude'].values),max(coordinates['latitude'].values))
print( latitude_extents )
```

```
(9.1764253745784181, 13.964805165051667)
```

```
longitude_extents = (min(coordinates['longitude'].values),max(coordinates['longitude'].values))
print( longitude_extents )
```

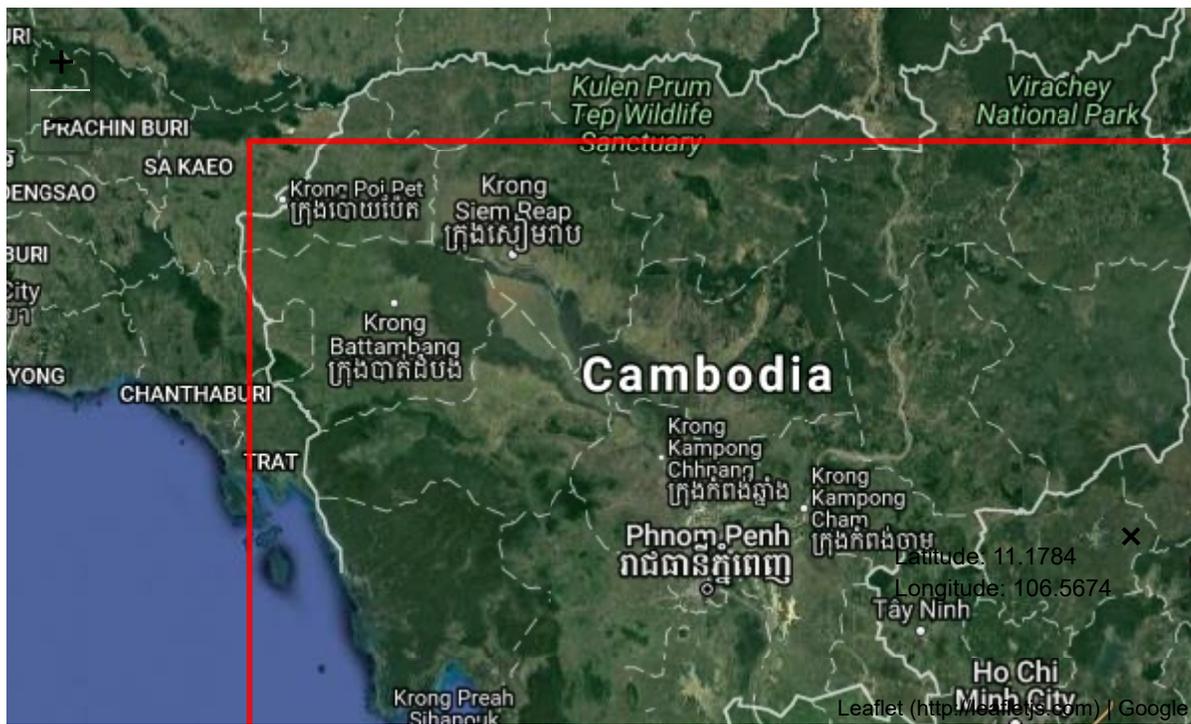
```
(102.40430421277932, 108.93092407802477)
```

```
time_extents = (min(coordinates['time'].values),max(coordinates['time'].values))
print( time_extents )
```

```
(numpy.datetime64('1999-09-08T03:13:19.000000000'), numpy.datetime64('2016-12-29T03:10:00.000000000'))
```

Visualize Data Cube Region

```
## The code below renders a map that can be used to orient yourself with the region.
from utils.data_cube_utilities.dc_display_map import display_map
display_map(latitude = latitude_extents, longitude = longitude_extents)
```



Pick a smaller analysis region and display that region

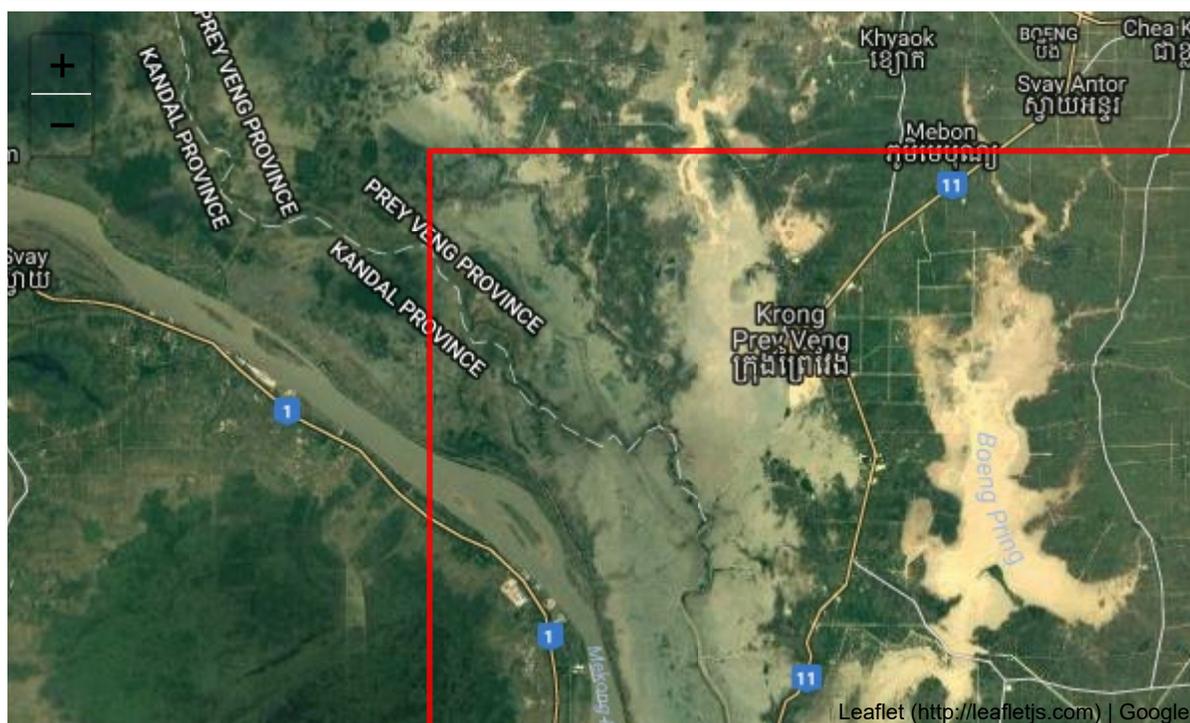
Try to keep your region to less than 0.2-deg x 0.2-deg for rapid processing. You can click on the map above to find the Lat-Lon coordinates of any location. You will want to identify a region with an urban area or some known vegetation. Pick a time window of a few months to a year so we can pick out some clear pixels.

```
## Vietnam - Central Lam Dong Province ##
longitude_extents = (105.2, 105.5)
latitude_extents = (11.25, 11.55)

time_extents = ('2015-01-01', '2016-01-01')
print ( time_extents )

('2015-01-01', '2016-01-01')
```

```
display_map(latitude = latitude_extents, longitude = longitude_extents)
```



Load the dataset and the required spectral bands or other parameters

After loading, you will view the Xarray dataset. Notice the dimensions represent the number of pixels in your latitude and longitude dimension as well as the number of time slices (time) in your time series.

```
landsat_dataset = dc.load(latitude = latitude_extents,
                          longitude = longitude_extents,
                          platform = platform,
                          time = time_extents,
                          product = product,
                          measurements = ['red', 'green', 'blue', 'nir', 'swir1', 'swir
2', 'pixel_qa'])
```

```
landsat_dataset
#view the dimensions and sample content from the cube
```

```
<xarray.Dataset>
Dimensions:    (latitude: 1115, longitude: 1114, time: 16)
Coordinates:
  * time       (time) datetime64[ns] 2015-01-07T03:18:35 2015-01-23T03:18:36
  ...
  * latitude  (latitude) float64 11.55 11.55 11.55 11.55 11.55 11.55 11.55
  ...
  * longitude (longitude) float64 105.2 105.2 105.2 105.2 105.2 105.2 105.2
  ...
Data variables:
  red          (time, latitude, longitude) int16 1392 1251 1190 1169 1290 ...
  green        (time, latitude, longitude) int16 1367 1257 1257 1257 1301 ...
  blue         (time, latitude, longitude) int16 1242 1158 1094 1115 1137 ...
  nir          (time, latitude, longitude) int16 2654 2700 2972 3017 2791 ...
  swir1        (time, latitude, longitude) int16 1550 1577 1818 1791 1845 ...
  swir2        (time, latitude, longitude) int16 923 952 980 1008 1038 923
  ...
  pixel_qa     (time, latitude, longitude) int32 224 224 224 224 224 224 224
  ...
Attributes:
  crs:         EPSG:4326
```

Display Example Images

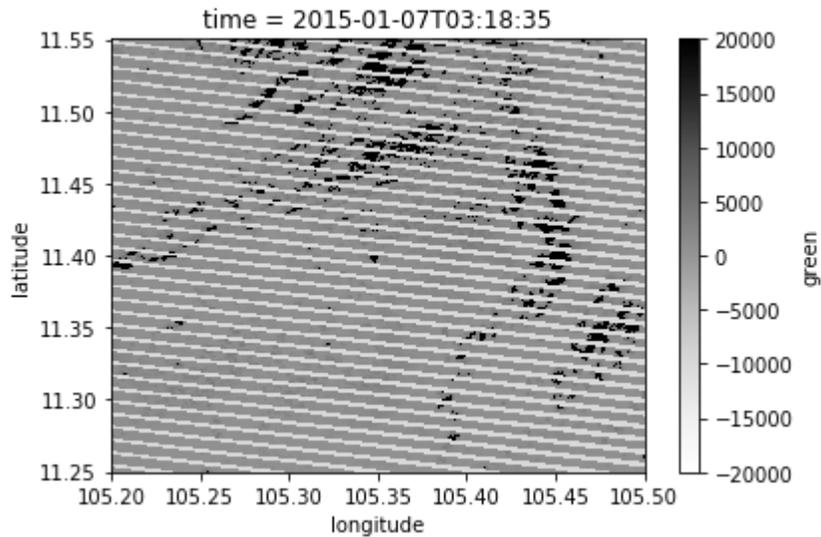
Single band visualization

For a quick inspection, let's look at one image. The code will allow the selection of any band (red, blue, green, nir, swir1, swir2) to produce a grey-scale image. Select the desired acquisition (time slice) in the block below. You can select from 1 to #, where the max value is the number of time slices noted in the block above. Change the comment statements below to select the bands for the first image.

```
acquisition_number = 0
# select an acquisition number from 0 (first time layer) to "time" using the array limits
above
```

```
%matplotlib inline
#Landsat_dataset.red.isel(time = acquisition_number).plot(cmap = "Greys")
Landsat_dataset.green.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.blue.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.nir.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.swir1.isel(time = acquisition_number).plot(cmap = "Greys")
#Landsat_dataset.swir2.isel(time = acquisition_number).plot(cmap = "Greys")
```

<matplotlib.collections.QuadMesh at 0x7f0932357048>



Define Cloud Masking Function

Removes clouds and cloud shadows based on the Landsat pixel QA information This is only for reference ...
nothing to modify here

```
import numpy as np

def generate_cloud_mask(dataset, include_shadows = False):
    #Create boolean Masks for clear and water pixels
    clear_pixels = dataset.pixel_qa.values == 2 + 64
    water_pixels = dataset.pixel_qa.values == 4 + 64
    shadow_pixels= dataset.pixel_qa.values == 8 + 64

    a_clean_mask = np.logical_or(clear_pixels, water_pixels)

    if include_shadows:
        a_clean_mask = np.logical_or(a_clean_mask, shadow_pixels)

    return np.invert(a_clean_mask)

def remove_clouds(dataset, include_shadows = False):
    #Create boolean Masks for clear and water pixels
    clear_pixels = dataset.pixel_qa.values == 2 + 64
    water_pixels = dataset.pixel_qa.values == 4 + 64
    shadow_pixels= dataset.pixel_qa.values == 8 + 64

    a_clean_mask = np.logical_or(clear_pixels, water_pixels)

    if include_shadows:
        a_clean_mask = np.logical_or(a_clean_mask, shadow_pixels)

    return dataset.where(a_clean_mask)
```

```
cloud_mask = generate_cloud_mask(landsat_dataset)
cloudless = remove_clouds(landsat_dataset) #Landsat_dataset.where(image_is_clean)
```

Set up plotting function (to be used later) Nothing to modify here

```

import matplotlib.pyplot as plt
import matplotlib.cm as color_maps
from time import time
import numpy as np

# Change the bands (RGB) here if you want other false color combinations
def rgb(dataset, at_index = 0, bands = ['red', 'green', 'blue'], paint_on_mask = []):

    rgb = np.stack([dataset[bands[0]], dataset[bands[1]], dataset[bands[2]]], axis = -1)
    max_possible = 3500
    rgb = rgb.astype(np.float32)
    rgb[rgb<0] = 0
    rgb[rgb > max_possible] = max_possible
    rgb *= 255.0/rgb.max()

    rgb = rgb.astype(int)
    rgb = rgb.astype(np.float32)
    rgb = 255-rgb

    rgb[rgb > 254] = 254
    rgb[rgb < 1] = 1

    for mask, color in paint_on_mask:
        rgb[mask] = np.array([256,256,256]) - np.array(color).astype(np.int16)

    if 'time' in dataset:
        plt.imshow((rgb[at_index]))
    else:
        plt.imshow(rgb)

```

Median Mosaic

Masks clouds from imagery using the median valued cloud-free pixels in the time series

```

from utils.data_cube_utilities.dc_mosaic import create_median_mosaic

def median_mosaic(dataset):
    # The mask here is based on pixel_qa products. It comes bundled in with most Landsat P
    # products.
    cloud_free_boolean_mask = np.invert(generate_cloud_mask(dataset))
    return create_median_mosaic(dataset, clean_mask = cloud_free_boolean_mask)

```

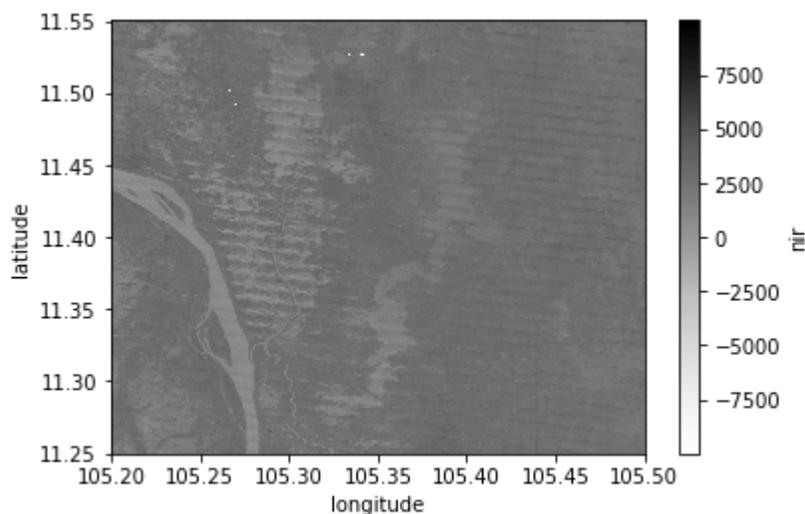
```

median_composite = median_mosaic(landsat_dataset)

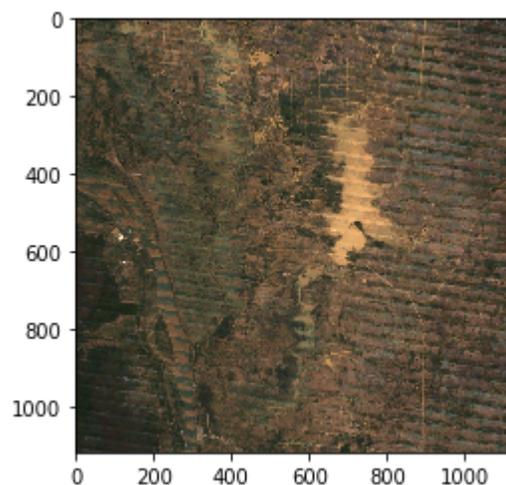
```

```
median_composite.nir.plot(cmap = "Greys")
```

```
<matplotlib.collections.QuadMesh at 0x7f0932552940>
```



```
rgb(median_composite)
```



Fractional Cover

Fractional Cover (FC) is used for landcover type estimation (vegetation, non-green vegetation, bare soil) of each pixel. We use a model from CSIRO (Juan Gerschmann) and apply it to a median mosaic.

```
from utils.data_cube_utilities.dc_fractional_coverage_classifier import frac_coverage_classify
frac_classes = frac_coverage_classify(median_composite, clean_mask = np.ones(median_composite.pixel_qa.shape).astype(np.bool))
```

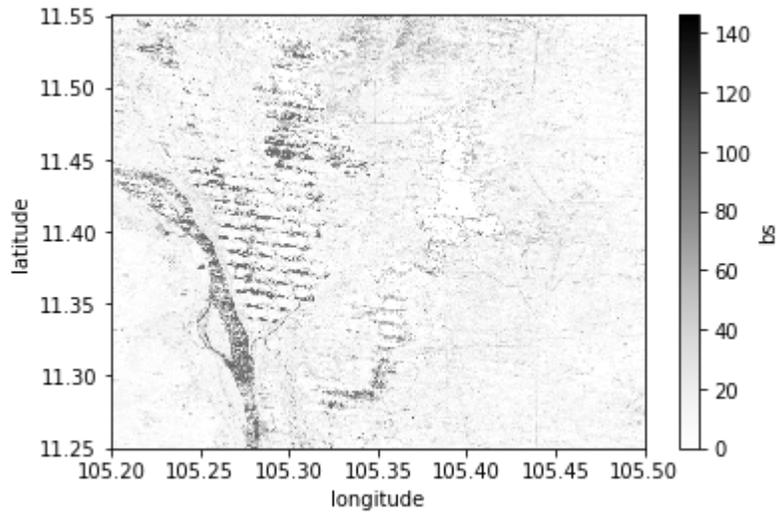
Plotting Fractional Cover Results

Plot Bare Soil (bs), Photosynthetic Vegetation (pv) or Non Photosynthetic Vegetation (npv)

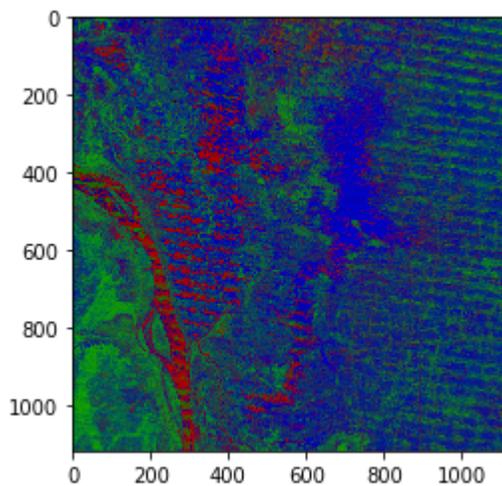
Plot a False Color RGB result where RGB = bs/pv/npv.

```
frac_classes.bs.plot(cmap = "Greys")
#frac_classes.pv.plot(cmap = "Greys")
#frac_classes.npv.plot(cmap = "Greys")
```

```
<matplotlib.collections.QuadMesh at 0x7f0932ba7898>
```



```
rgb(frac_classes, bands = ['bs', 'pv', 'npv'])
```



Spectral Indices

NDVI (vegetation) and NDBI (urbanization)

NDVI = Normalized Difference Vegetation Index

A derived index that correlates well with the existence of vegetation.

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

```
def NDVI(dataset):
    return (dataset.nir - dataset.red)/(dataset.nir + dataset.red)
```

NDBI = Normalized Difference Build-Up Index

A derived index that correlates well with the existence of urbanization.

$$NDBI = \frac{(SWIR1 - NIR)}{(SWIR1 + NIR)}$$

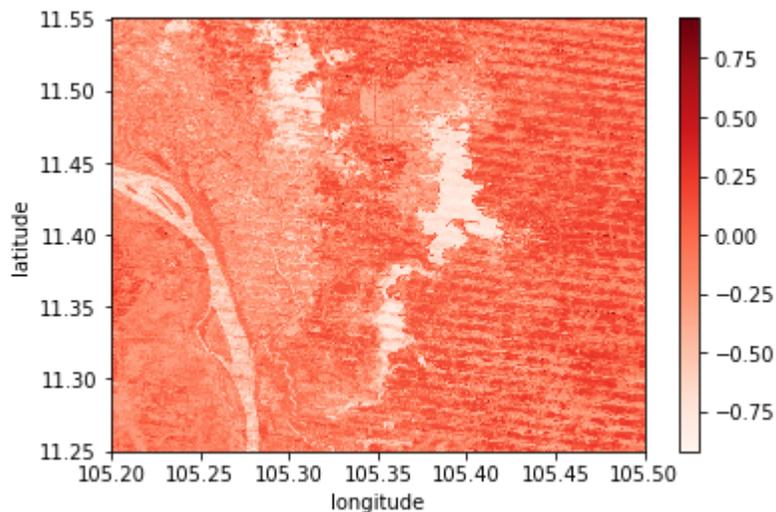
```
def NDBI(dataset):
    return (dataset.swir1 - dataset.nir)/(dataset.swir1 + dataset.nir)
```

```
landsat_mosaic = median_mosaic(landsat_dataset)
```

```
ndbi = NDBI(landsat_mosaic) # Urbanization - Reds
ndvi_mosaic = NDVI(landsat_mosaic) # Dense Vegetation - Greens
```

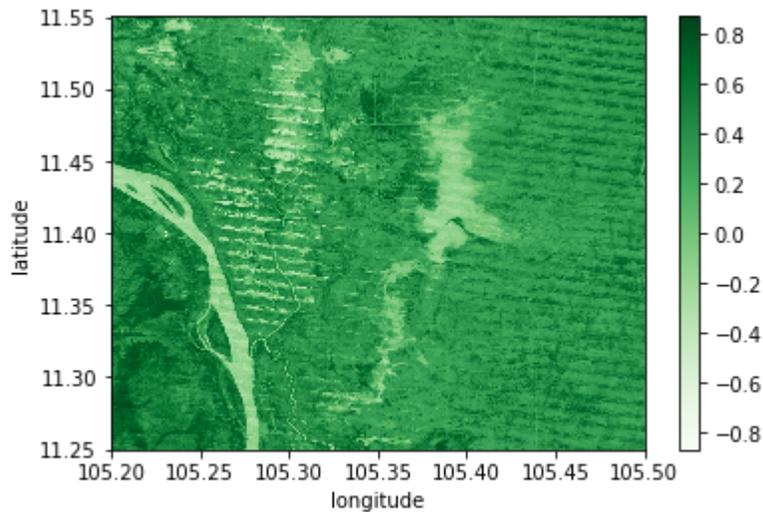
```
(ndbi).plot(cmap = "Reds")
```

```
<matplotlib.collections.QuadMesh at 0x7f09181ab320>
```



```
(ndvi_mosaic).plot(cmap = "Greens")
```

```
<matplotlib.collections.QuadMesh at 0x7f09180ee0b8>
```



Create a threshold plot

First we will define a minimum threshold and a maximum threshold. Then you will create a plot that colors the region between the threshold a single color (e.g. red) and the region outside the threshold will be BLACK or WHITE. Also, we will calculate the % of pixels and the number of pixels in the threshold range.

```
# Select the time slice for the NVDI output (first slice=0)  
t = 0  
ndvi_dataset_at_time_t = NDVI(landsat_dataset).isel(time = t)  
mask_at_time_t = generate_cloud_mask(landsat_dataset.isel(time = t))
```

```
# Define the threshold region bounds  
minimum_threshold = 0.6  
maximum_threshold = 0.9
```

```
from matplotlib.ticker import FuncFormatter

def threshold_plot(da, min_threshold, max_threshold, mask = None, width = 10, *args, **kwargs):
    color_in = np.array([255,0,0])
    color_out = np.array([0,0,0])
    color_cloud = np.array([255,255,255])

    array = np.zeros((*da.values.shape, 3)).astype(np.int16)

    inside = np.logical_and(da.values > min_threshold, da.values < max_threshold)
    outside = np.invert(inside)
    masked = np.zeros(da.values.shape).astype(bool) if mask is None else mask

    array[inside] = np.array([256,256,256]) - color_in
    array[outside] = np.array([256,256,256]) - color_out
    array[masked] = np.array([256,256,256]) - color_cloud

    def figure_ratio(ds, fixed_width = 10):
        width = fixed_width
        height = len(ds.latitude) * (fixed_width / len(ds.longitude))
        return (width, height)

    fig, ax = plt.subplots(figsize = figure_ratio(da, fixed_width = width))

    lat_formatter = FuncFormatter(lambda y_val, tick_pos: "{0:.3f}".format(da.latitude.values[tick_pos] ))
    lon_formatter = FuncFormatter(lambda x_val, tick_pos: "{0:.3f}".format(da.longitude.values[tick_pos]))

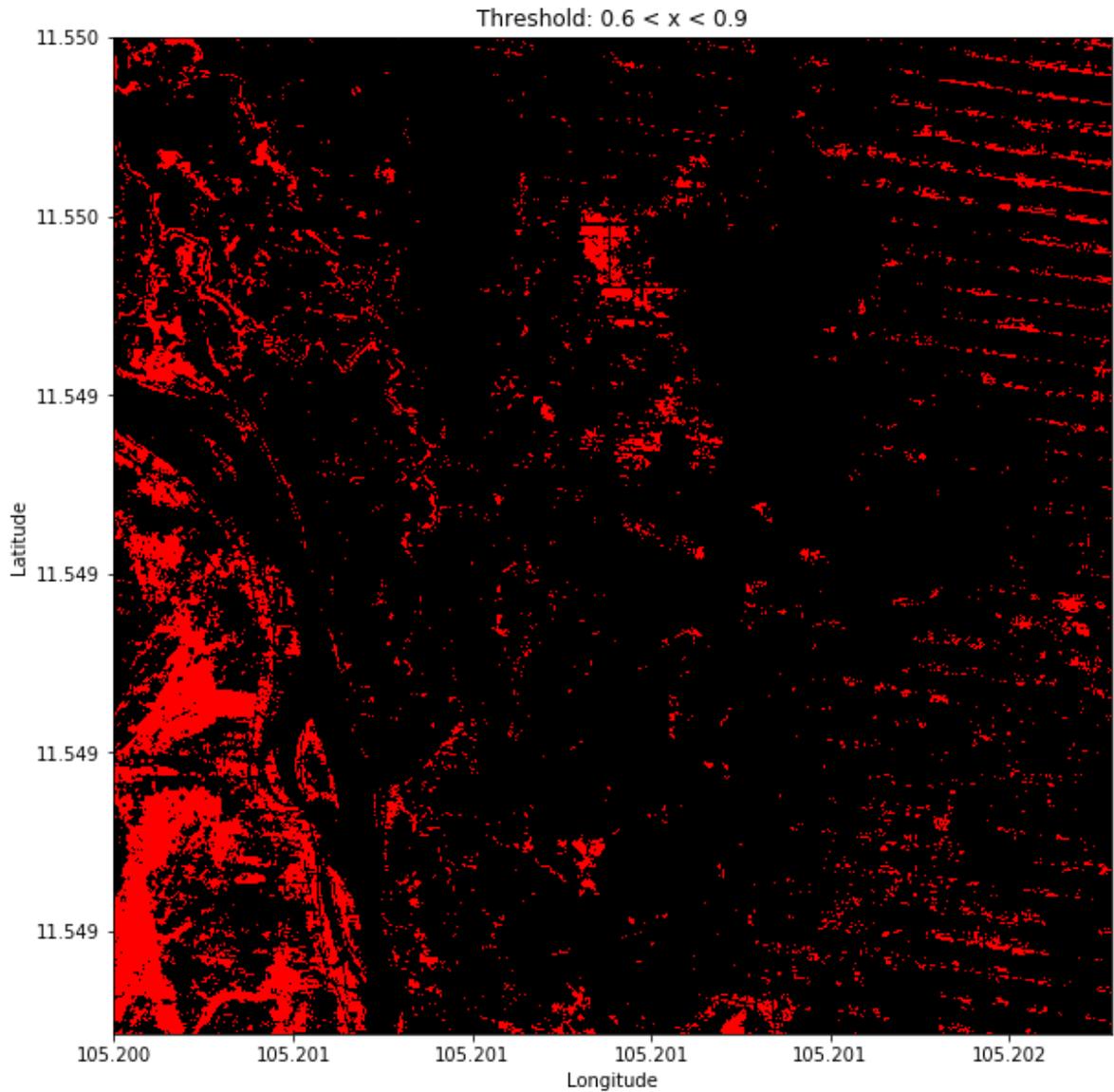
    ax.xaxis.set_major_formatter(lon_formatter)
    ax.yaxis.set_major_formatter(lat_formatter)

    plt.title("Threshold: {} < x < {}".format(min_threshold, max_threshold))
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')

    plt.imshow(array, *args, **kwargs)
    plt.show()
```

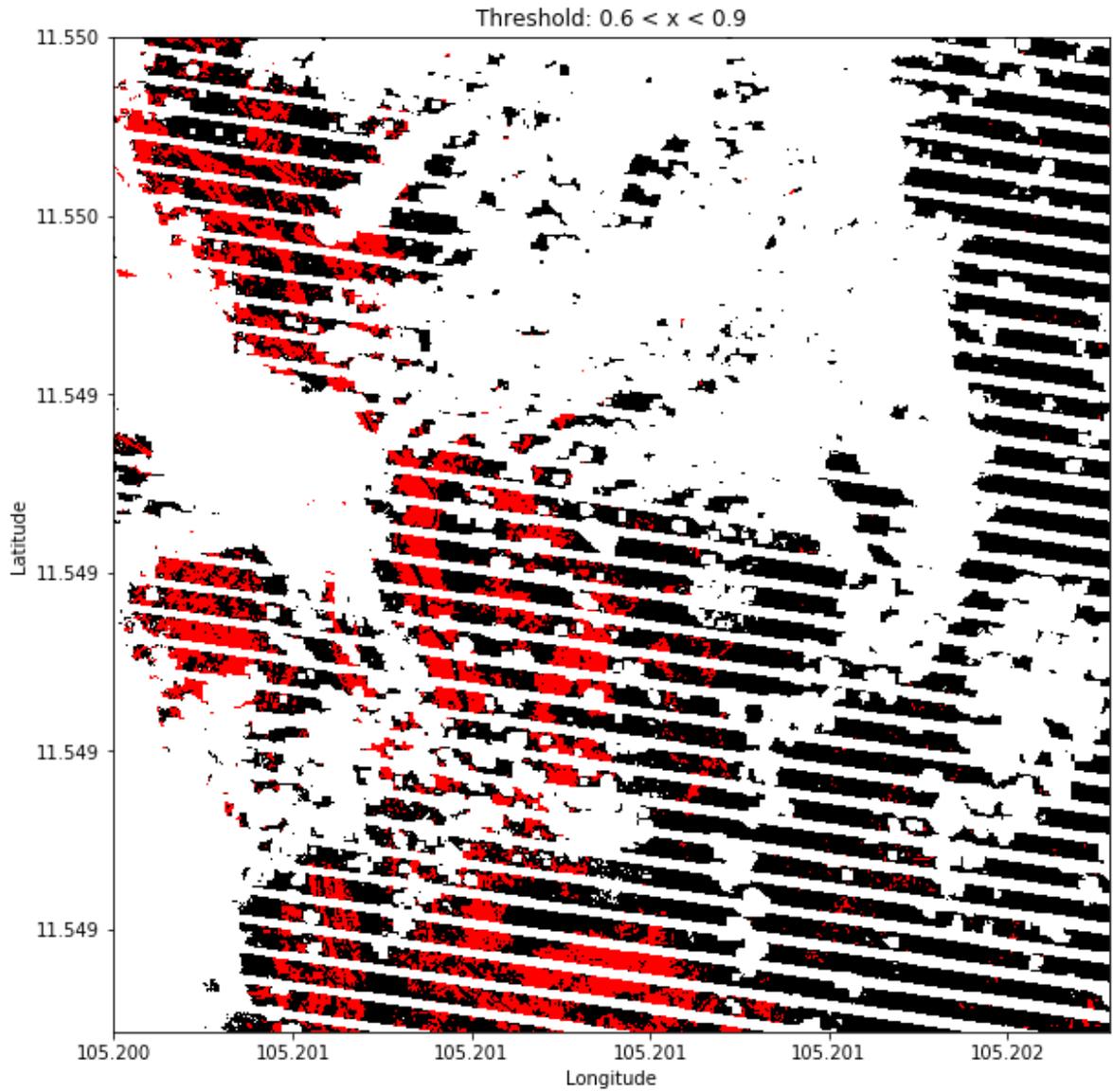
Plot NDVI Mosaic

```
# Plot the NDVI threshold product using a cloud-filtered mosaic  
threshold_plot(ndvi_mosaic, minimum_threshold, maximum_threshold, width = 10)
```



Plot NDVI at time t

```
# Plot the NDVI threshold product using a single time slice (one scene)  
threshold_plot(ndvi_dataset_at_time_t, minimum_threshold, maximum_threshold, width = 10, m  
ask = mask_at_time_t,)
```



```

def threshold_count(da, min_threshold, max_threshold, mask = None):
    def count_not_nans(arr):
        return np.count_nonzero(~np.isnan(arr))

    in_threshold = np.logical_and( da.values > min_threshold, da.values < max_threshold)

    total_non_cloudy = count_not_nans(da.values) if mask is None else np.sum(mask)

    return dict(total = np.size(da.values),
                total_non_cloudy = total_non_cloudy,
                inside = np.nansum(in_threshold),
                outside = total_non_cloudy - np.nansum(in_threshold)
                )

def threshold_percentage(da, min_threshold, max_threshold, mask = None):
    counts = threshold_count(da, min_threshold, max_threshold, mask = mask)
    return dict(percent_inside_threshold = (counts["inside"] / counts["total"]) * 100.0,
                percent_outside_threshold = (counts["outside"] / counts["total"]) * 100.0,
                percent_clouds = ( 100.0-counts["total_non_cloudy"] / counts["total"] * 10
0.0))

```

```

threshold_count(ndvi_mosaic,
                minimum_threshold,
                maximum_threshold)

```

```

{'inside': 112330,
 'outside': 1129780,
 'total': 1242110,
 'total_non_cloudy': 1242110}

```

```

threshold_percentage(ndvi_mosaic,
                    minimum_threshold,
                    maximum_threshold)

```

```

{'percent_clouds': 0.0,
 'percent_inside_threshold': 9.0434824612956994,
 'percent_outside_threshold': 90.956517538704304}

```

```

threshold_count(ndvi_dataset_at_time_t,
                minimum_threshold,
                maximum_threshold,
                mask = mask_at_time_t)

```

```

{'inside': 97721,
 'outside': 730582,
 'total': 1242110,
 'total_non_cloudy': 828303}

```

```
threshold_percentage(ndvi_dataset_at_time_t,  
                    minimum_threshold,  
                    maximum_threshold,  
                    mask = mask_at_time_t )
```

```
{'percent_clouds': 33.314843290851854,  
'percent_inside_threshold': 7.8673386415051807,  
'percent_outside_threshold': 58.817818067642961}
```

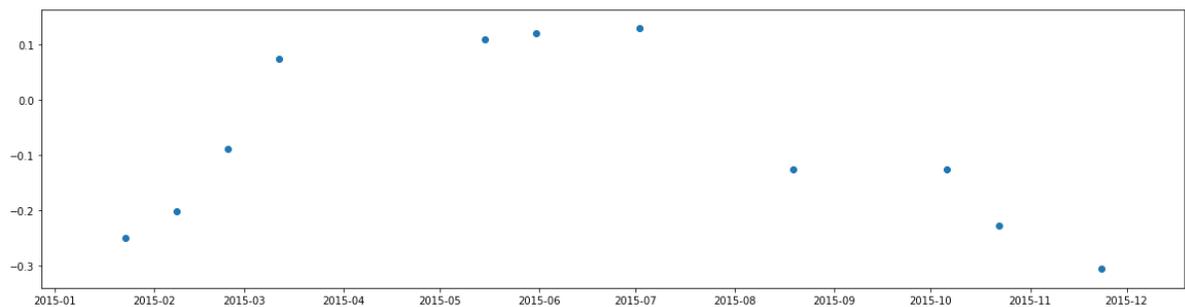
Select a single pixel and plot an index value over time

```
pixel_lat = 11.45  
pixel_lon = 105.40
```

```
pixel = NDVI(remove_clouds(landsat_dataset)).sel(latitude = pixel_lat,  
                                                  longitude = pixel_lon,  
                                                  method = 'nearest') # nearest neighbor selection
```

```
plt.figure(figsize = (20,5))  
plt.scatter(pixel.time.values, pixel.values)
```

```
<matplotlib.collections.PathCollection at 0x7f0932a92550>
```



Task D:

Land Change

ARDC Training: Python Notebooks

Task-D: Land Change

Import the Datacube Configuration

```
import datacube
dc = datacube.Datacube(app = 'my_app', config = '/home/localuser/.datacube.conf')
```

```
# Enable plotting
%matplotlib inline

# Suppress Warning
import warnings
warnings.filterwarnings('ignore')
```

Browse the available Data Cubes

```
list_of_products = dc.list_products()
netCDF_products = list_of_products[list_of_products['format'] == 'NetCDF']
netCDF_products
```

Pick a product

Use the platform and product names from the previous block to select a Data Cube.

```
import utils.data_cube_utilities.data_access_api as dc_api
api = dc_api.DataAccessApi(config = '/home/localuser/.datacube.conf')

# Change the data platform and data cube here

platform = "LANDSAT_7"
# platform = "LANDSAT_8"

# product = "ls7_ledaps_ghana"
# product = "ls7_ledaps_kenya"
# product = "ls7_ledaps_senegal"
# product = "ls7_ledaps_sierra_leone"
# product = "ls7_ledaps_tanzania"
product = "ls7_ledaps_vietnam"

# Get Coordinates
coordinates = api.get_full_dataset_extent(platform = platform, product = product)
```

Display Latitude-Longitude and Time Bounds of the Data Cube

```
latitude_extents = (min(coordinates['latitude'].values),max(coordinates['latitude'].values
))
print( latitude_extents )
```

```
(9.1764253745784181, 13.964805165051667)
```

```
longitude_extents = (min(coordinates['longitude'].values),max(coordinates['longitude'].values))
print( longitude_extents )
```

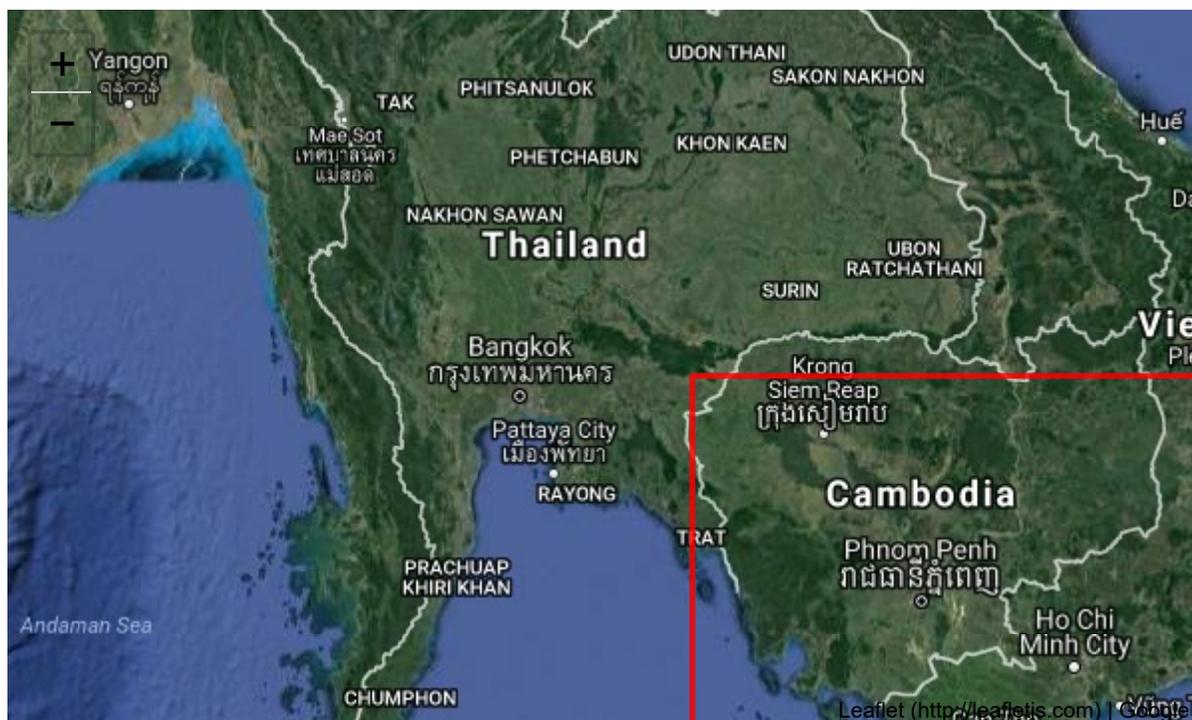
```
(102.40430421277932, 108.93092407802477)
```

```
time_extents = (min(coordinates['time'].values),max(coordinates['time'].values))
print( time_extents )
```

```
(numpy.datetime64('1999-09-08T03:13:19.000000000'), numpy.datetime64('2016-12-29T03:10:00.000000000'))
```

Visualize Data Cube Region

```
## The code below renders a map that can be used to orient yourself with the region.
from utils.data_cube_utilities.dc_display_map import display_map
display_map(latitude = latitude_extents, longitude = longitude_extents)
```



Pick a smaller analysis region and display that region

Try to keep your region to less than 0.02-deg x 0.02-deg for rapid processing. This will give you a region of about 75x75 pixels. You can click on the map above to find the Lat-Lon coordinates of any location. You will want to identify a region with an urban area or some known vegetation. Pick a time window of 10+ years, as the curvefit algorithm requires a long time series to detect change.

Here is what to expect ... 0.02-deg x 0.02-deg (75x75 pixels) over 10 years will take 10 to 15 minutes to execute. Be patient ...

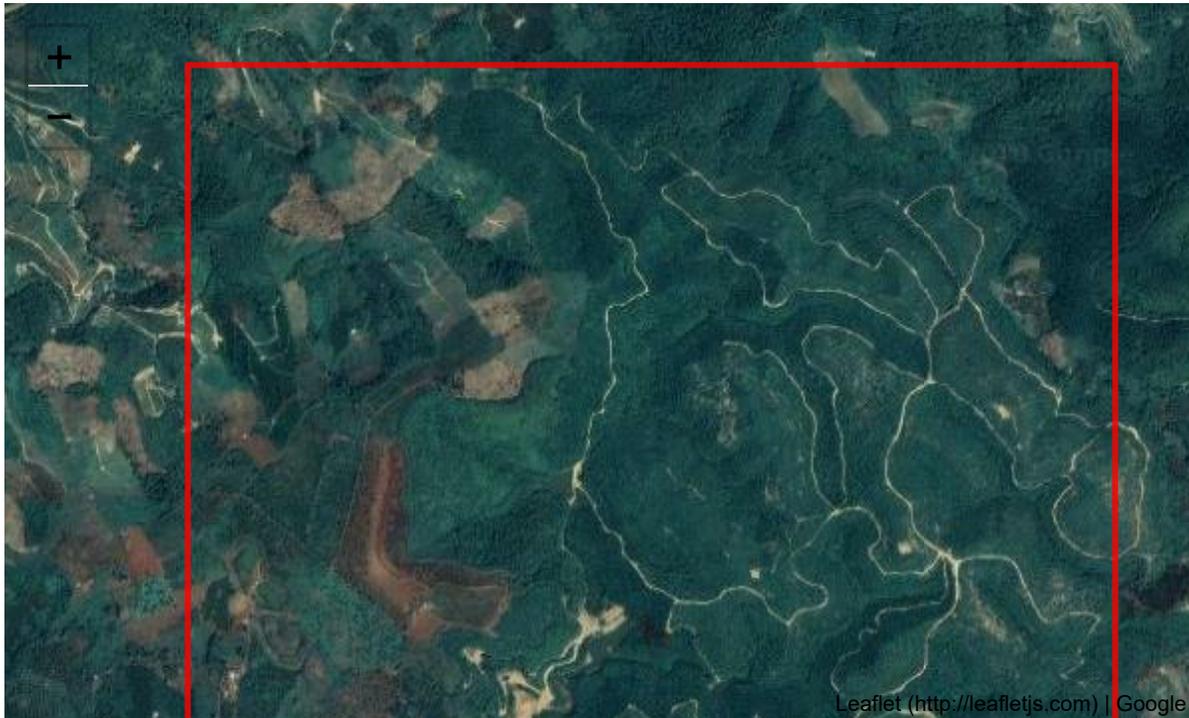
```
# Lam Dong Province near reservior
# latitude_extents = (11.843, 11.922)
# longitude_extents = (107.723, 107.821)

latitude_extents = (11.90, 11.92)
longitude_extents = (107.76, 107.78)

time_extents = ('2001-01-01', '2010-01-01')
print ( time_extents )

('2001-01-01', '2010-01-01')
```

```
display_map(latitude = latitude_extents, longitude = longitude_extents)
```



Load the dataset and the required spectral bands or other parameters

After loading, you will view the Xarray dataset. Notice the dimensions represent the number of pixels in your latitude and longitude dimension as well as the number of time slices (time) in your time series.

```
landsat_dataset = dc.load(latitude = latitude_extents,  
                           longitude = longitude_extents,  
                           platform = platform,  
                           time = time_extents,  
                           product = product,  
                           measurements = ['red', 'green', 'blue', 'nir', 'swir1', 'swir2',  
                                           'pixel_qa'])
```

```
landsat_dataset
#view the dimensions and sample content from the cube
```

```
<xarray.Dataset>
Dimensions:   (latitude: 75, longitude: 75, time: 93)
Coordinates:
  * time      (time) datetime64[ns] 2001-01-02T02:58:02 2001-02-03T02:58:07
  ...
  * latitude  (latitude) float64 11.92 11.92 11.92 11.92 11.92 11.92 11.92
  ...
  * longitude (longitude) float64 107.8 107.8 107.8 107.8 107.8 107.8 107.8
  ...
Data variables:
  red        (time, latitude, longitude) int16 476 412 434 434 412 412 369
  ...
  green      (time, latitude, longitude) int16 733 662 638 615 615 520 473
  ...
  blue       (time, latitude, longitude) int16 361 383 361 383 383 361 361
  ...
  nir        (time, latitude, longitude) int16 4688 4379 4159 4159 4159 ...
  swir1      (time, latitude, longitude) int16 2160 2053 1944 1972 1918 ...
  swir2      (time, latitude, longitude) int16 889 889 805 861 833 578 608
  ...
  pixel_qa   (time, latitude, longitude) int32 66 66 66 66 66 66 66 66
  ...
Attributes:
  crs:        EPSG:4326
```

PyCCD

The compute time of PyCCD scales sharply with extent sizes. Seemingly small areas can take anywhere from 10-15 minutes to process.

```
import utils.data_cube_utilities.dc_ccd as ccd
```

```
%time
change_matrix = ccd.process_xarray(landsat_dataset, distributed = True, process = "matrix"
) #Run process xarray on large dataset
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.48 µs
```

```
%time
change_volume = (change_matrix.sum(dim='time') - 1).rename('change_volume')
```

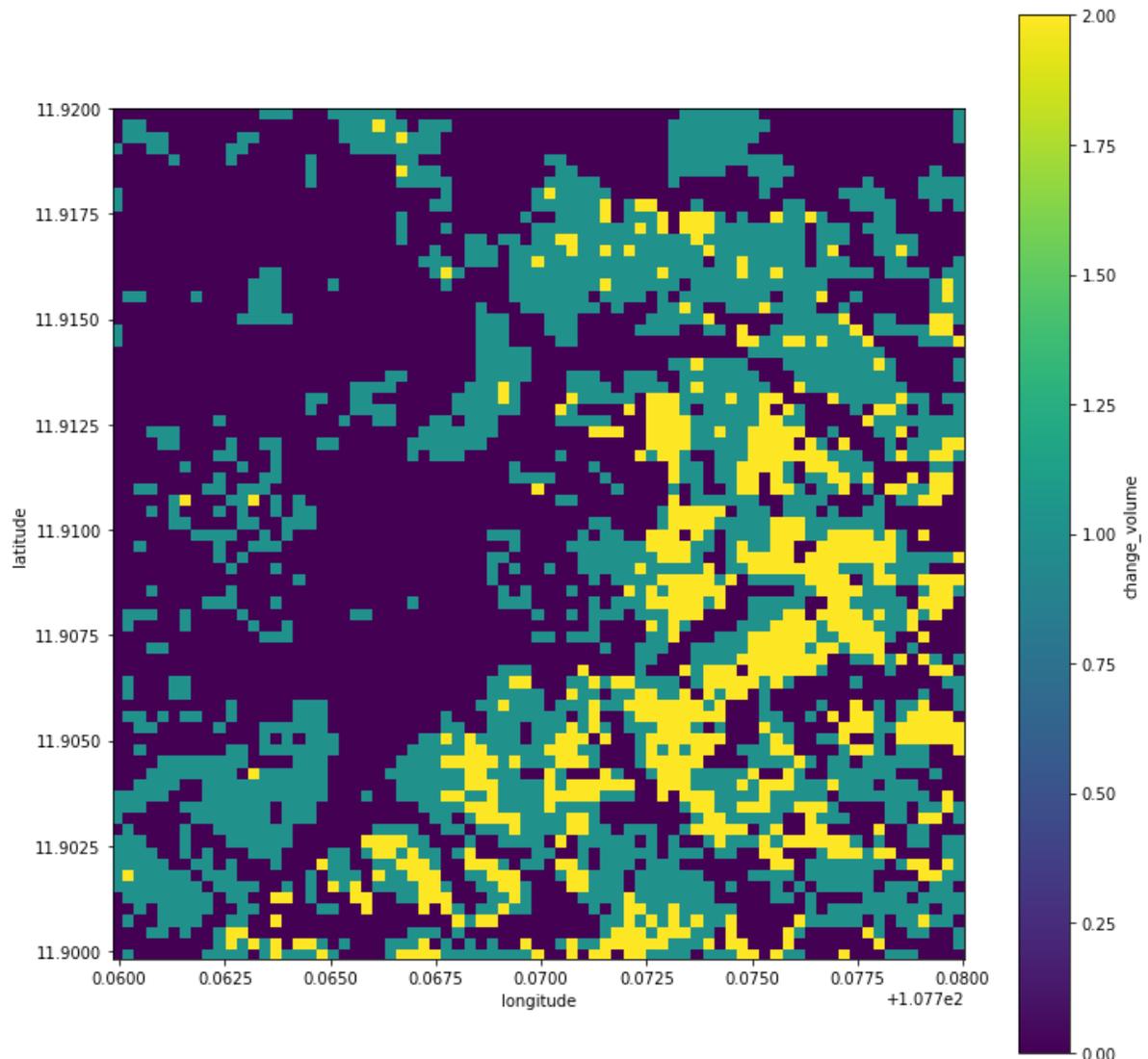
```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 4.53 µs
```

Plotting change volume

Plot change volume to identify regions/areas of change. The change volume represents the number of land changes for a pixel over the time series.

```
def figure_ratio(ds, fixed_width = 12):  
    width = fixed_width  
    height = len(ds.latitude) * (fixed_width / len(ds.longitude))  
    return (width, height)
```

```
import matplotlib.pyplot as plt  
  
plt.figure(figsize = figure_ratio(change_volume))  
change_volume.plot()  
plt.axes().set_aspect("equal")
```



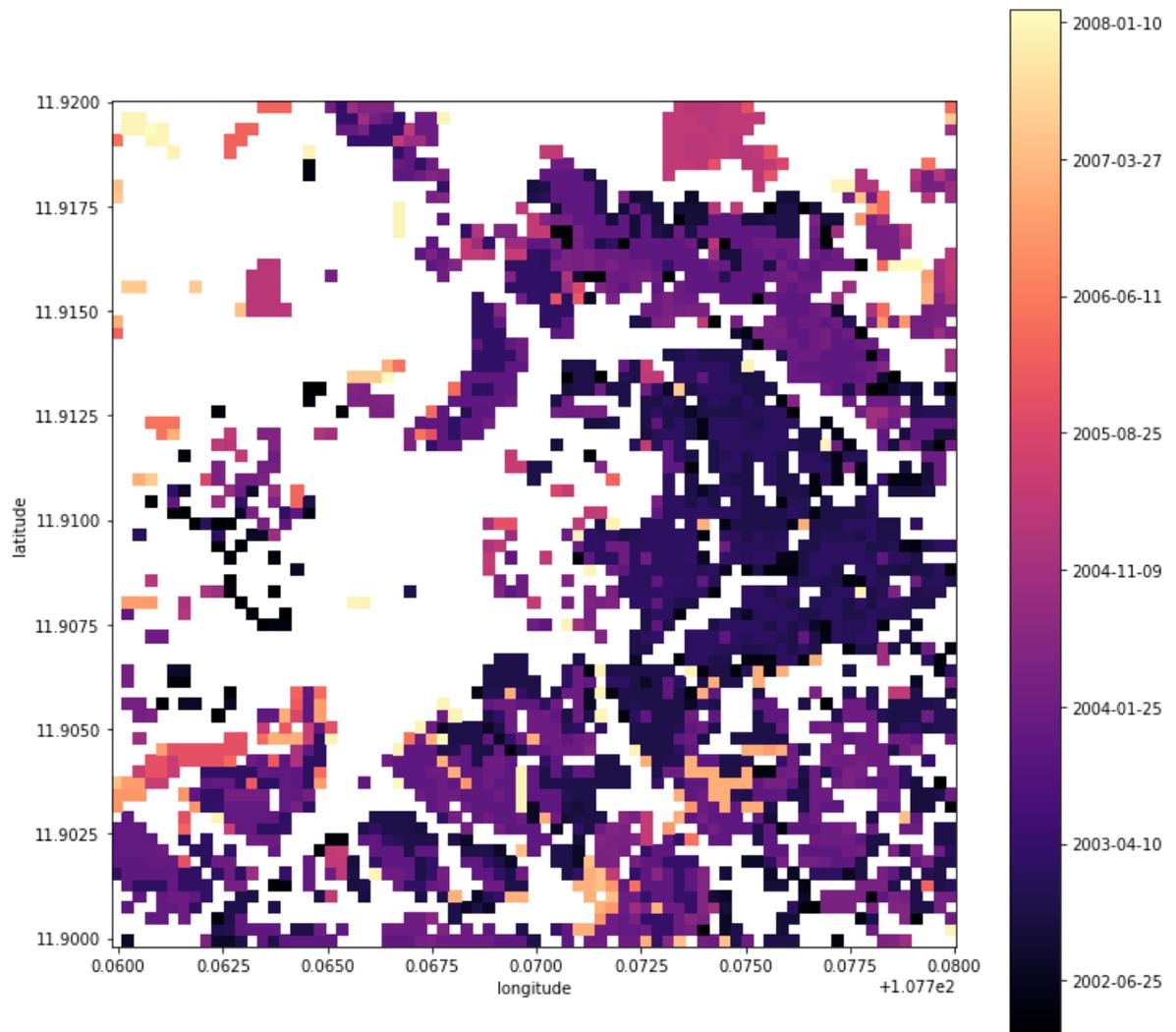
Plot the time of first changes

```
%time
time_map_ccd_product = ccd._nth_occurrence_in_ccd_matrix(change_matrix, 1, f = ccd._n64_dat
etime_to_scalar)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.48 µs
```

```
import datetime
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter

plt.figure(figsize = figure_ratio(time_map_ccd_product))
epochFormatter = FuncFormatter(lambda x, pos: datetime.datetime.utcfromtimestamp(x).strfti
me('%Y-%m-%d'))
time_map_ccd_product.plot(cmap = "magma", cbar_kwargs=({'format': epochFormatter}))
plt.axes().set_aspect("equal")
```



Validating Change

Use the two images below to review scenes at the beginning of the time series and the end of the time series.

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

## Used to plot rgb composites of xarray. Simply pass in a dataset, adjust the 'at_index' parameter
def rgb(dataset, at_index = 0, bands = ['red', 'green', 'blue'], paint_on_mask = [], figsize = (7,7)):

    rgb = np.stack([dataset[bands[0]], dataset[bands[1]], dataset[bands[2]]], axis = -1)
    max_possible = 3500
    rgb = rgb.astype(np.float32)
    rgb[rgb<0] = 0
    rgb[rgb > max_possible] = max_possible
    rgb *= 255.0/rgb.max()

    rgb = rgb.astype(int)
    rgb = rgb.astype(np.float32)
    rgb = 255-rgb

    rgb[rgb > 254] = 254
    rgb[rgb < 1] = 1

    for mask, color in paint_on_mask:
        rgb[mask] = np.array([256,256,256]) - np.array(color).astype(np.int16)

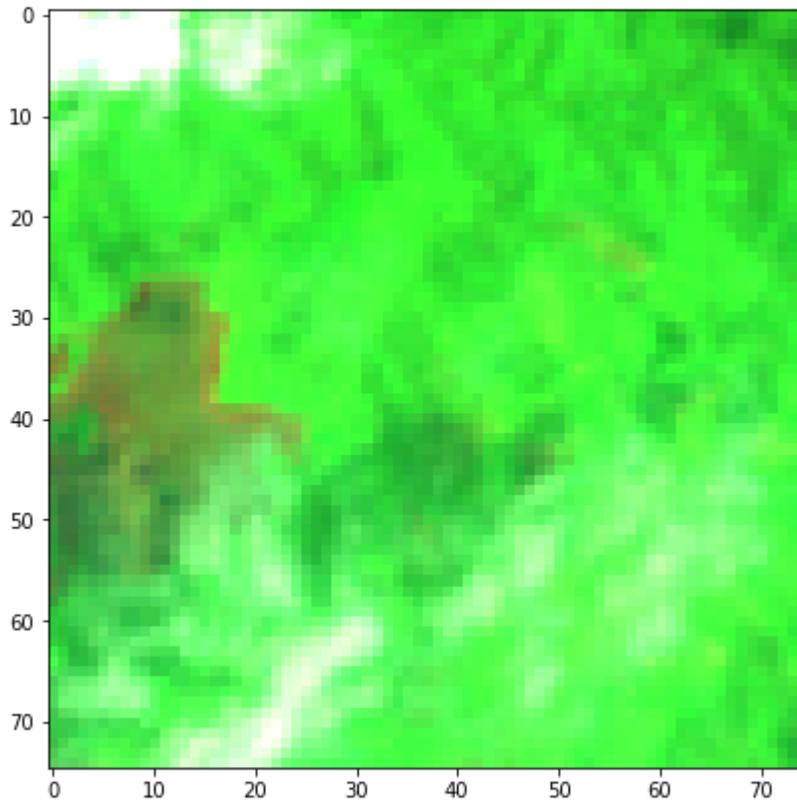
    plt.figure(figsize = figsize)
    if 'time' in dataset:
        plt.imshow((rgb[at_index]))
    else:
        plt.imshow(rgb)
```

Review of RGB images

Choose an image from the start and end of the time series. Remember that 0 is the first acquisition and the last acquisition if the number of time steps. You can find that number in the XARRAY report above. Try various combinations of the start and end images to identify the land changes using visual interpretation. You will see some images will have clouds and others will have Landsat-7 "bands". You can also change the RGB image bands to give you an combination you desire.

```
first = 2 # Acquisition Number ... change here  
  
print( landsat_dataset.time.values[first] )  
  
rgb( landsat_dataset,  
      at_index = first,  
      bands = ['swir2','nir','green'])
```

2001-04-08T02:57:43.000000000

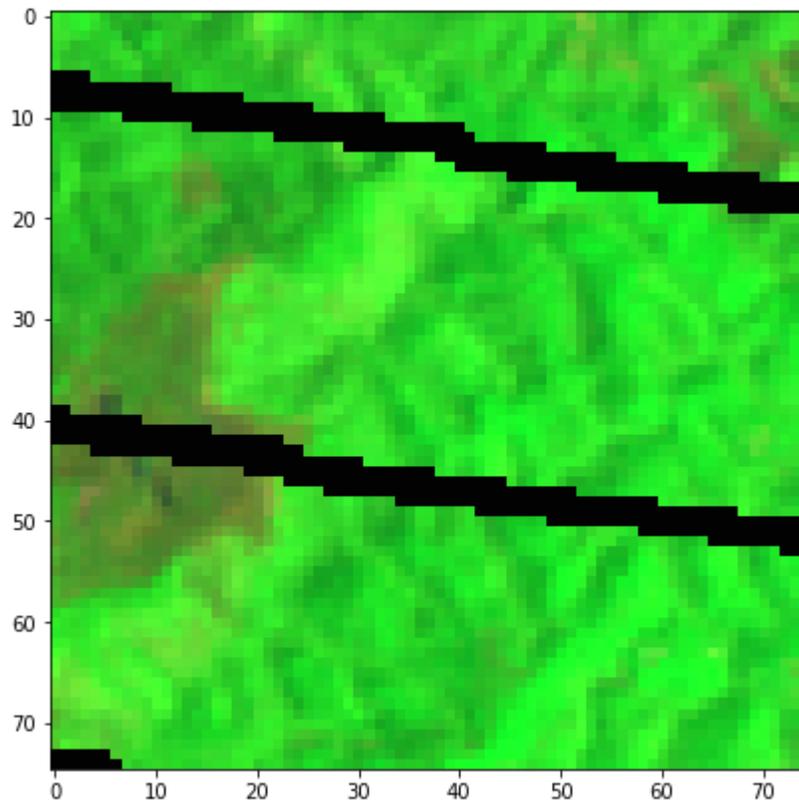


```
last = 50 # Acquisition Number ... change here

print( landsat_dataset.time.values[last] )

rgb(landsat_dataset,
     at_index = last,
     bands = ['swir2','nir','green'])
```

2005-03-02T02:57:20.000000000



Vogelmann NDVI Trend

```
import utils.data_cube_utilities.trend as trend
```

```
import numpy as np
```

```
def land_and_water_masking_l7(dataset):
    #Create boolean Masks for clear and water pixels
    clear_pixels = dataset.pixel_qa.values == 2 + 64
    water_pixels = dataset.pixel_qa.values == 4 + 64

    a_clean_mask = np.logical_or(clear_pixels, water_pixels)
    return a_clean_mask
```

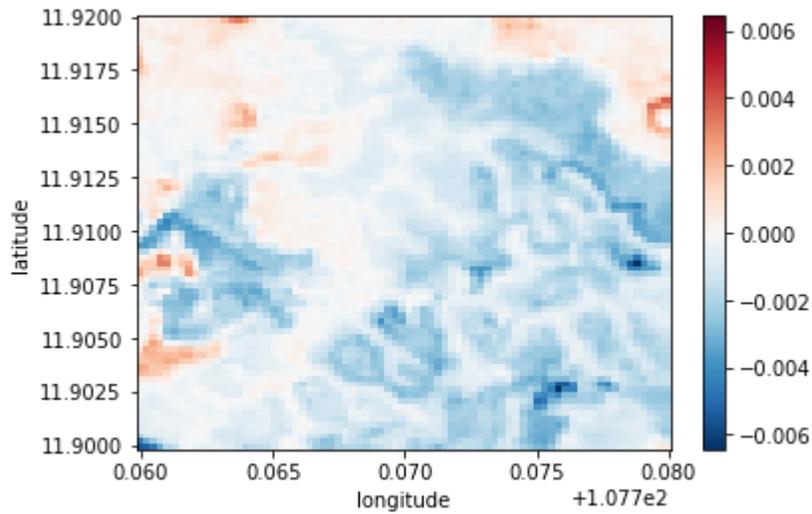
```
def remove_clouds(dataset):
    return dataset.where(land_and_water_masking_ls7(dataset)).drop('pixel_qa')
```

```
def NDVI(dataset):  
    return (dataset.nir - dataset.red)/(dataset.nir + dataset.red)
```

```
ndvi_trend_product = trend.linear(NDVI(remove_clouds(landsat_dataset)))
```

```
(-ndvi_trend_product).plot()
```

```
<matplotlib.collections.QuadMesh at 0x7f6839563be0>
```



Task E:

2D Transect Analyses and 3D Hovmoller Plots

ARDC Training: Python Notebooks

Task-E: This notebook will demonstrate 2D transect analyses and 3D Hovmoller plots. We will run these for NDVI (land) and TSM (water quality) to show the spatial and temporal variation of data along a line (transect) for a given time slice and for the entire time series.

Import the Datacube Configuration

```
import datacube
dc = datacube.Datacube(app = 'my_app', config = '/home/localuser/.datacube.conf')
```

```
# Suppress Warning
import warnings
warnings.filterwarnings('ignore')
```

Browse the available Data Cubes

```
list_of_products = dc.list_products()
netCDF_products = list_of_products[list_of_products['format'] == 'NetCDF']
netCDF_products
```

Pick a product

Use the platform and product names from the previous block to select a Data Cube.

```
import utils.data_cube_utilities.data_access_api as dc_api
api = dc_api.DataAccessApi(config = '/home/localuser/.datacube.conf')

# Change the data platform and data cube here

platform = "LANDSAT_7"
# platform = "LANDSAT_8"

# product = "ls7_ledaps_ghana"
# product = "ls7_ledaps_kenya"
# product = "ls7_ledaps_senegal"
# product = "ls7_ledaps_sierra_leone"
# product = "ls7_ledaps_tanzania"
product = "ls7_ledaps_vietnam"

# Get Coordinates
coordinates = api.get_full_dataset_extent(platform = platform, product = product)
```

Display Latitude-Longitude and Time Bounds of the Data Cube

```
latitude_extents = (min(coordinates['latitude'].values),max(coordinates['latitude'].values))
print( latitude_extents )
```

```
(9.1764253745784181, 13.964805165051667)
```

```
longitude_extents = (min(coordinates['longitude'].values),max(coordinates['longitude'].values))
print( longitude_extents )
```

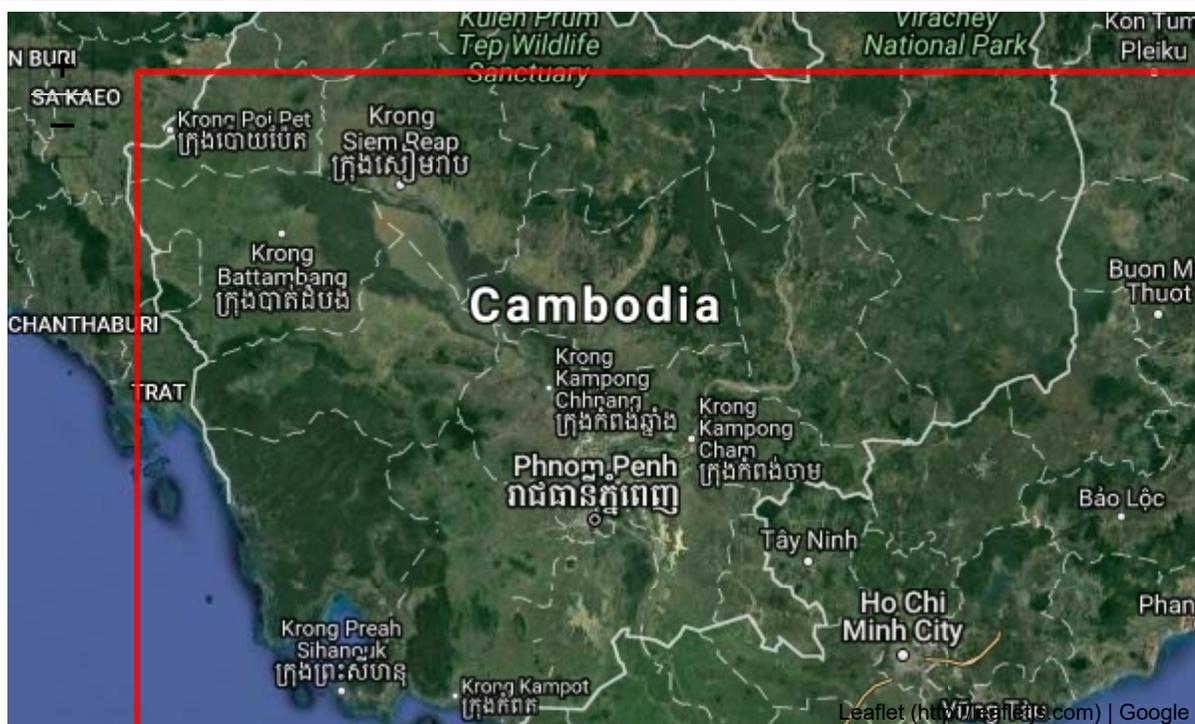
```
(102.40430421277932, 108.93092407802477)
```

```
time_extents = (min(coordinates['time'].values),max(coordinates['time'].values))
print( time_extents )
```

```
(numpy.datetime64('1999-09-08T03:13:19.000000000'), numpy.datetime64('2016-12-29T03:10:00.000000000'))
```

Visualize Data Cube Region

```
## The code below renders a map that can be used to orient yourself with the region.
from utils.data_cube_utilities.dc_display_map import display_map
display_map(latitude = latitude_extents, longitude = longitude_extents)
```



Pick a smaller analysis region and display that region

Try to keep your region to less than 0.2-deg x 0.2-deg for rapid processing. You can click on the map above to find the Lat-Lon coordinates of any location. You will want to identify a region with an inland water body and some vegetation. Pick a time window of several years.

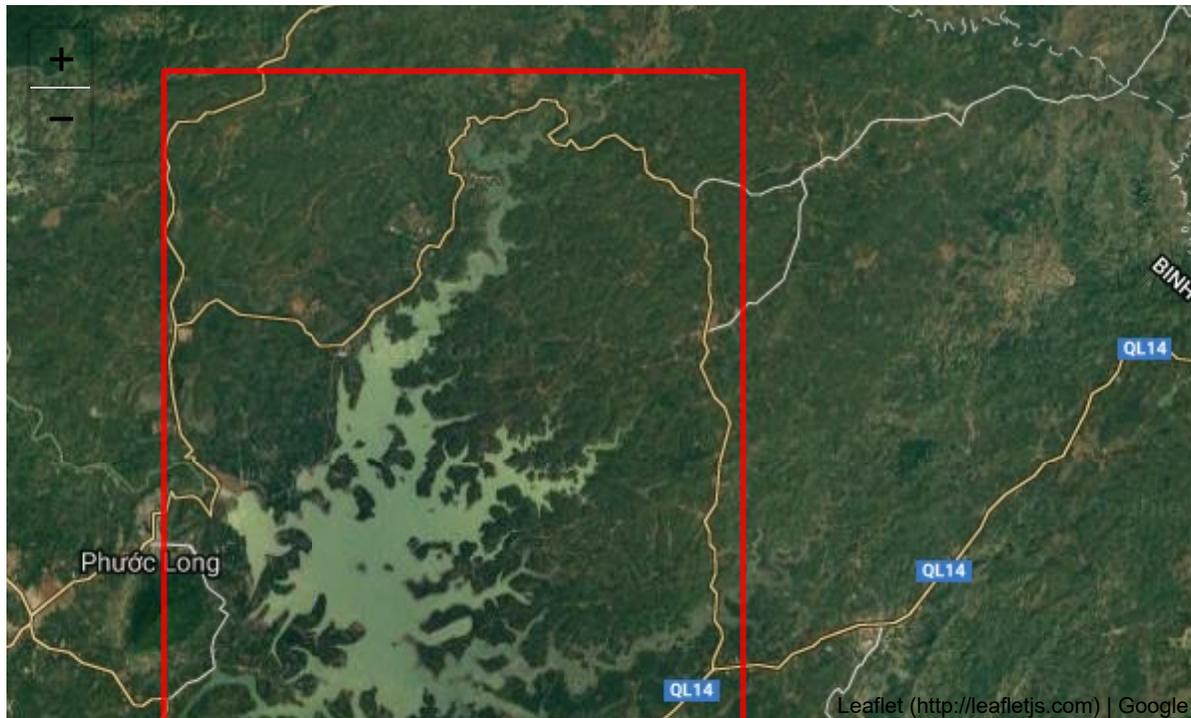
```
## Vietnam - Central Lam Dong Province ##
longitude_extents = (107.0, 107.2)
latitude_extents = (11.7, 12.0)

## Vietnam Ho Tri An Lake
# longitude_extents = (107.0, 107.2)
# latitude_extents = (11.1, 11.3)

time_extents = ('2005-01-01', '2015-01-01')
print ( time_extents )

('2005-01-01', '2015-01-01')
```

```
display_map(latitude = latitude_extents, longitude = longitude_extents)
```



Load the dataset and the required spectral bands or other parameters

After loading, you will view the Xarray dataset. Notice the dimensions represent the number of pixels in your latitude and longitude dimension as well as the number of time slices (time) in your time series.

```
landsat_dataset = dc.load(latitude = latitude_extents,  
                           longitude = longitude_extents,  
                           platform = platform,  
                           time = time_extents,  
                           product = product,  
                           measurements = ['red', 'green', 'blue', 'nir', 'swir1', 'swir2',  
                                           'pixel_qa'])
```

```
landsat_dataset
#view the dimensions and sample content from the cube
```

```
<xarray.Dataset>
Dimensions:   (latitude: 1114, longitude: 743, time: 103)
Coordinates:
  * time      (time) datetime64[ns] 2005-01-13T02:57:03 2005-01-29T02:57:05
  ...
  * latitude  (latitude) float64 12.0 12.0 12.0 12.0 12.0 12.0 12.0 12.0 ...
  * longitude (longitude) float64 107.0 107.0 107.0 107.0 107.0 107.0 107.0
  ...
Data variables:
  red        (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  green      (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  blue       (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  nir        (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  swir1      (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  swir2      (time, latitude, longitude) int16 -9999 -9999 -9999 -9999 ...
  pixel_qa   (time, latitude, longitude) int32 1 1 1 1 1 1 1 1 1 1 1 1 1
  ...
Attributes:
  crs:       EPSG:4326
```

Preparing the data

We will filter out the clouds and the water using the Landsat pixel_qa information. Next, we will calculate the values of NDVI (vegetation index) and TSM (water quality).

```
import xarray as xr
import numpy as np
def ls7_unpack_qa( data_array , cover_type):

    land_cover_encoding = dict( fill      = [1],
                               clear     = [66, 130],
                               water     = [68, 132],
                               shadow    = [72, 136],
                               snow      = [80, 112, 144, 176],
                               cloud     = [96, 112, 160, 176, 224],
                               low_conf  = [66, 68, 72, 80, 96, 112],
                               med_conf  = [130, 132, 136, 144, 160, 176],
                               high_conf= [224]
                               )

    boolean_mask = np.isin(data_array.values, land_cover_encoding[cover_type])
    return xr.DataArray(boolean_mask.astype(int),
                        coords = data_array.coords,
                        dims = data_array.dims,
                        name = cover_type + "_mask",
                        attrs = data_array.attrs)
```

```
clear_xarray = ls7_unpack_qa(landsat_dataset.pixel_qa, "clear")
water_xarray = ls7_unpack_qa(landsat_dataset.pixel_qa, "water")
shadow_xarray = ls7_unpack_qa(landsat_dataset.pixel_qa, "shadow")
cloud_xarray = ls7_unpack_qa(landsat_dataset.pixel_qa, "cloud")
```

```
clean_xarray = xr.ufuncs.logical_or(clear_xarray , water_xarray).astype(np.int8).rename("clean_mask")

clean_mask = np.logical_or(clear_xarray.values.astype(bool),
                          water_xarray.values.astype(bool))
```

```
def NDVI(dataset):
    return ((dataset.nir - dataset.red)/(dataset.nir + dataset.red)).rename("NDVI")
```

```
ndvi_xarray = NDVI(landsat_dataset) # Vegetation Index
```

```
from utils.data_cube_utilities.dc_water_quality import tsm

tsm_xarray = tsm(landsat_dataset, clean_mask = water_xarray.values.astype(bool) ).tsm
```

Combine everything into one XARRAY for further analysis

```
combined_dataset = xr.merge([landsat_dataset,
                              clean_xarray,
                              clear_xarray,
                              water_xarray,
                              shadow_xarray,
                              cloud_xarray,
                              ndvi_xarray,
                              tsm_xarray])

# Copy original crs to merged dataset
combined_dataset = combined_dataset.assign_attrs(landsat_dataset.attrs)
```

Define a path for a transect

A transect is just a line that will run across our region of interest. Use the display map above to find the end points of your desired line. If you click on the map it will give you precise Lat-Lon positions for a point.

Start with a line across a mix of water and land

```
# Water and Land Mixed Examples

# North-South Path
start = ( 11.8428, 107.0734 )
end   = ( 11.7306, 107.0734 )

# East-West Path
# start = ( 11.9167, 107.0554 )
# end   = ( 11.9167, 107.1719 )

# East-West Path for Lake Ho Tri An
# start = ( 11.25, 107.02 )
# end   = ( 11.25, 107.18 )
```

Plot the transect line

```

import folium
import numpy as np
from folium.features import CustomIcon

def plot_a_path(points , zoom = 15):
    xs,ys = zip(*points)

    map_center_point = (np.mean(xs), np.mean(ys))
    the_map = folium.Map(location=[map_center_point[0], map_center_point[1]], zoom_start =
zoom, tiles='http://mt1.google.com/vt/lyrs=y&z={z}&x={x}&y={y}', attr = "Google Attribution")
    path = folium.PolyLine(locations=points, weight=5, color = 'orange')
    the_map.add_child(path)

    start = ( xs[0] ,ys[0] )
    end   = ( xs[-1],ys[-1])

    return the_map

plot_a_path([start,end])

```



Find the nearest pixels along the transect path

```
from utils.data_cube_utilities.transect import line_scan

import numpy as np

def get_index_at(coords, ds):
    '''Returns an integer index pair.'''
    lat = coords[0]
    lon = coords[1]

    nearest_lat = ds.sel(latitude = lat, method = 'nearest').latitude.values
    nearest_lon = ds.sel(longitude = lon, method = 'nearest').longitude.values

    lat_index = np.where(ds.latitude.values == nearest_lat)[0]
    lon_index = np.where(ds.longitude.values == nearest_lon)[0]

    return (int(lat_index), int(lon_index))

def create_pixel_trail(start, end, ds):
    a = get_index_at(start, ds)
    b = get_index_at(end, ds)

    indices = line_scan.line_scan(a, b)

    pixels = [ ds.isel(latitude = x, longitude = y) for x, y in indices]
    return pixels
```

```
list_of_pixels_along_segment = create_pixel_trail(start, end, landsat_dataset)
```

Groundwork for Transect (2-D) and Hovmöller (3-D) Plots

```

import xarray
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
from datetime import datetime
import time

def plot_list_of_pixels(list_of_pixels, band_name, y = None):
    start = (
        "{0:.2f}".format(float(list_of_pixels[0].latitude.values)),
        "{0:.2f}".format(float(list_of_pixels[0].longitude.values))
    )
    end = (
        "{0:.2f}".format(float(list_of_pixels[-1].latitude.values)),
        "{0:.2f}".format(float(list_of_pixels[-1].longitude.values))
    )

    def reformat_n64(t):
        return time.strftime("%Y.%m.%d", time.gmtime(t.astype(int)/1000000000))

    def pixel_to_array(pixel):
        return(pixel.values)

    def figure_ratio(x,y, fixed_width = 10):
        width = fixed_width
        height = y * (fixed_width / x)
        return (width, height)

    pixel_array = np.transpose([pixel_to_array(pix) for pix in list_of_pixels])

    #If the data has one acquisition, then plot transect (2-D), else Hovmöller (3-D)
    if y.size == 1:
        plt.figure(figsize = (15,5))
        plt.scatter(np.arange(pixel_array.size), pixel_array)
        plt.title("Transect (2-D) \n Acquisition date: {}".format(reformat_n64(y)))
        plt.xlabel("Pixels along the transect \n {} - {} \n ".format(start,end))
        plt.ylabel(band_name)

    else:
        m = FuncFormatter(lambda x :x )
        figure = plt.figure(figsize= figure_ratio(len(list_of_pixels),
                                                len(list_of_pixels[0].values),
                                                fixed_width = 15))

        number_of_y_ticks = 5

        ax = plt.gca()
        cax = ax.imshow(pixel_array, interpolation='none')
        figure.colorbar(cax,fraction=0.110, pad=0.04)

        ax.set_title("Hovmöller (3-D) \n Acquisition range: {} - {} \n ".format(reforma
t_n64(y[0]),reformat_n64(y[-1])))
        plt.xlabel("Pixels along the transect \n {} - {} \n ".format(start,end))
        ax.get_yaxis().set_major_formatter( FuncFormatter(lambda x, p: reformat_n64(list
_of_pixels[0].time.values[int(x)]) if int(x) < len(list_of_pixels[0].time) else ""))
        plt.ylabel("Time")
    plt.show()

```

```
def transect_plot(start,
                  end,
                  da):
    if type(da) is not xarray.DataArray and (type(da) is xarray.Dataset) :
        raise Exception('You should be passing in a data-array, not a Dataset')

    pixels = create_pixel_trail(start, end, da)
    dates = da.time.values

    lats = [x.latitude.values for x in pixels]
    lons = [x.longitude.values for x in pixels]

    plot_list_of_pixels(pixels, da.name, y = dates)
```

```
pixels = create_pixel_trail(start, end, landsat_dataset)
```

```
t = 2
subset = list( map(lambda x: x.isel(time = t), pixels))
```

Mask Clouds

```
def land_and_water_masking_ls7(dataset):
    #Create boolean Masks for clear and water pixels
    clear_pixels = landsat_dataset.pixel_qa.values == 2 + 64
    water_pixels = landsat_dataset.pixel_qa.values == 4 + 64

    a_clean_mask = np.logical_or(clear_pixels, water_pixels)
    return a_clean_mask
```

```
cloudless_dataset = landsat_dataset.where(land_and_water_masking_ls7(landsat_dataset))
```

Select an acquisition date and then plot a 2D transect without clouds

```
# select an acquisition number from the start (t=0) to "time" using the array limits above
acquisition_number = 20
```

```
#If plotted will create the 2-D transect
cloudless_dataset_for_acq_no = cloudless_dataset.isel(time = acquisition_number)
```

```
#If Plotted will create the 3-D Hovmoller plot for a portion of the time series (min to max)
min_acq = 1
max_acq = 4

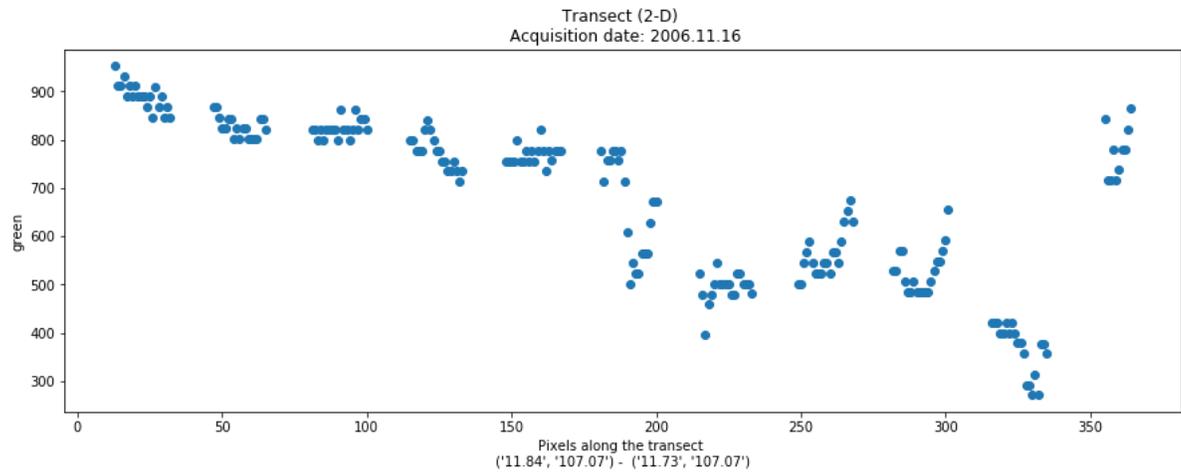
cloudless_dataset_from_1_to_acq_no = cloudless_dataset.isel(time = slice(min_acq, max_acq
))
```

Select one of the XARRAY parameters for analysis

```
band = 'green'
```

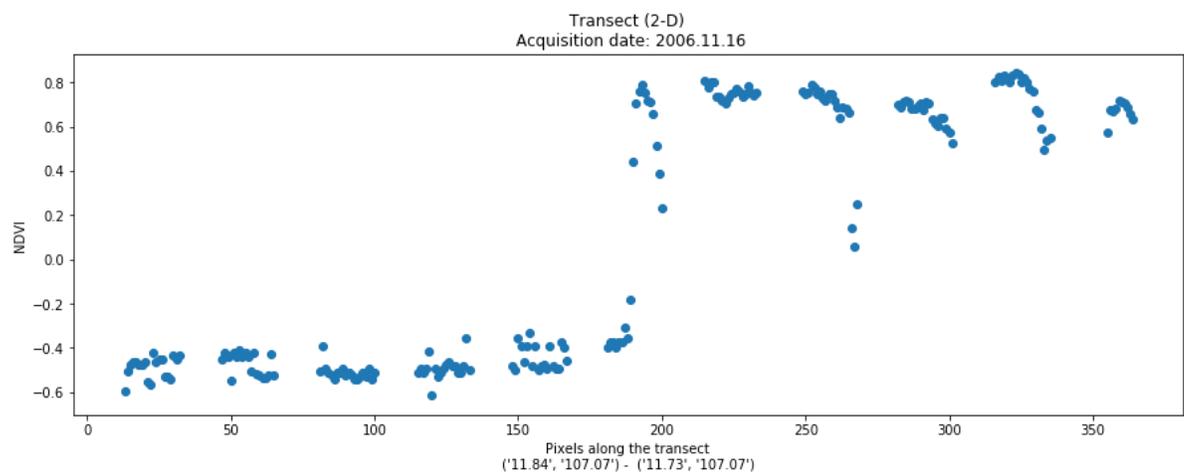
Create a 2D Transect plot of the "band" for one date

```
transect_plot(start, end, cloudless_dataset_for_acq_no[band])
```



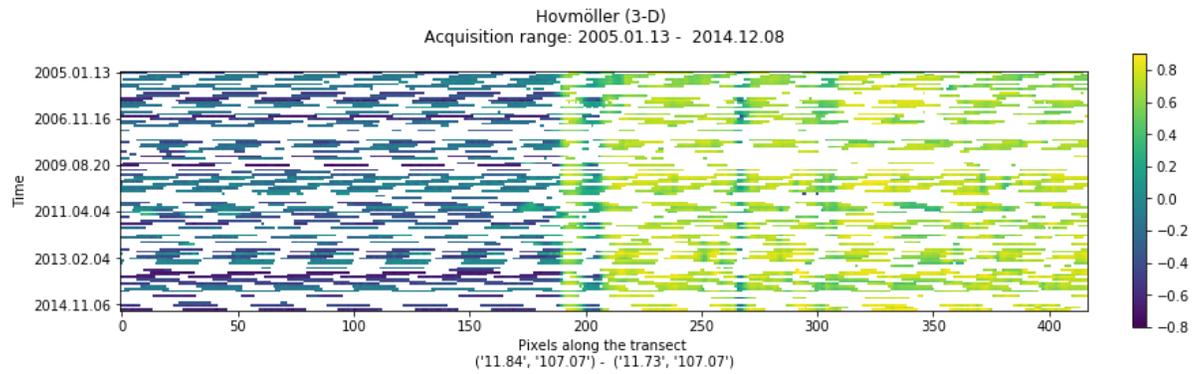
Create a 2D Transect plot of NDVI for one date

```
transect_plot(start, end, NDVI(cloudless_dataset_for_acq_no))
```



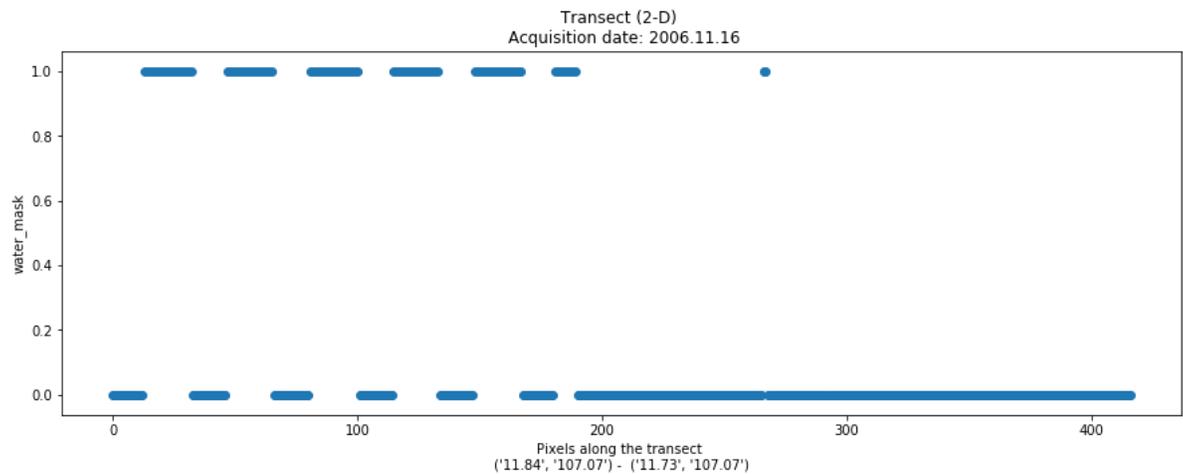
Create a 3D Hovmoller plot of NDVI for the entire time series

```
transect_plot(start, end, NDVI(cloudless_dataset))
```



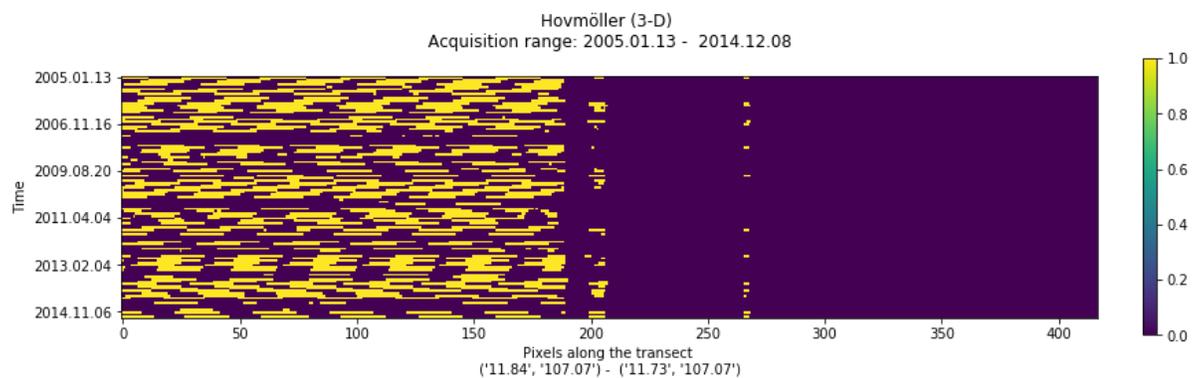
Create a 2D Transect plot of water existence for one date

```
transect_plot(start, end, water_xarray.isel(time = acquisition_number))
```



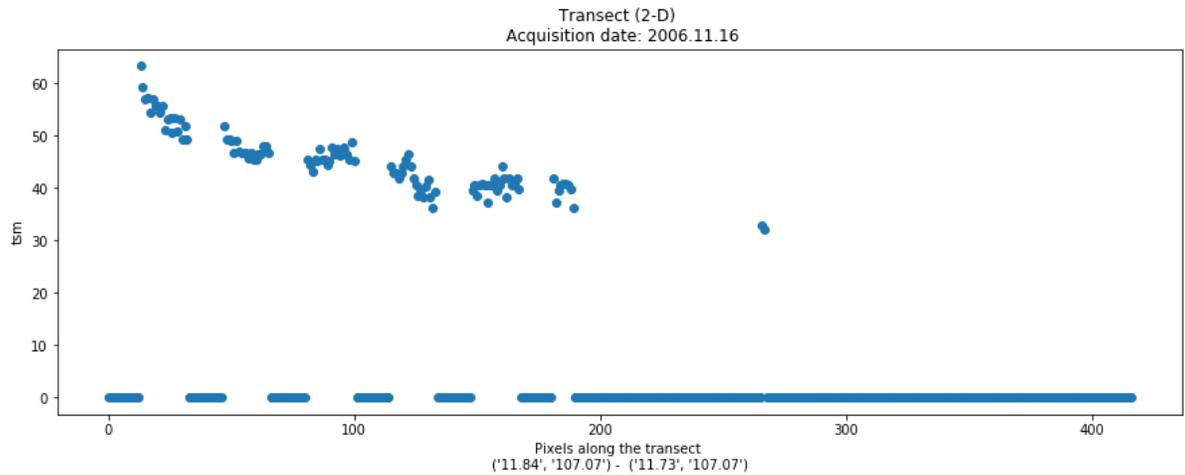
Create a 3D Hovmoller plot of water extent for the entire time series

```
transect_plot(start, end, water_xarray)
```



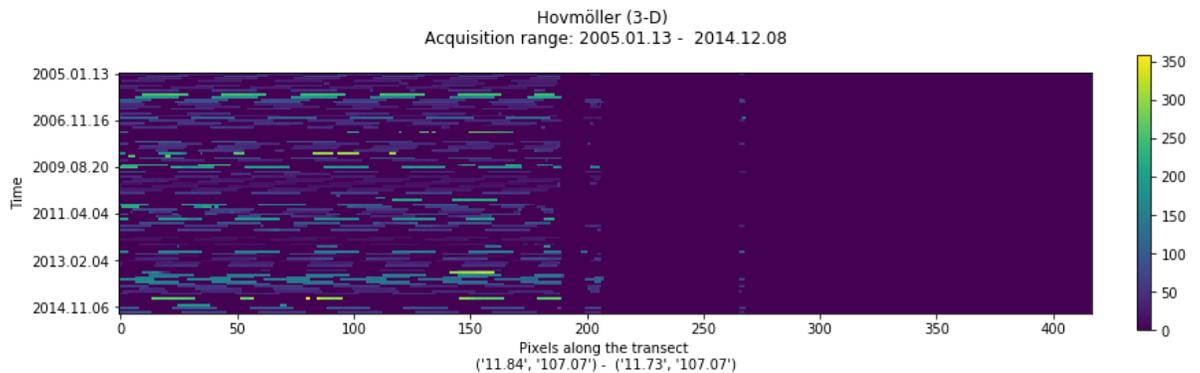
Create a 2D Transect plot of water quality (TSM) for one date

```
transect_plot(start, end, tsm_xarray.isel(time = acquisition_number))
```



Create a 3D Hovmoller plot of water quality (TSM) for one date

```
transect_plot(start, end, tsm_xarray)
```



Task F:

Data Export

ARDC Training: Python Notebooks

Task-F: Data Export

The code in this notebook subsets a data cube, selects a specific set of variables, creates a new XARRAY, and then outputs that data into a GeoTIFF file and CSV file. This output file can be used in other software programs (e.g. QGIS, ArcGIS, EXCEL) for more specific analyses. We will keep the region small so that we can control file sizes.

Import the Datacube Configuration

```
import datacube
dc = datacube.Datacube(app = 'my_app', config = '/home/localuser/.datacube.conf')

# Suppress Warning
import warnings
warnings.filterwarnings('ignore')
```

Browse the available Data Cubes on the storage platform

You might want to learn more about what data is stored and how it is stored.

```
list_of_products = dc.list_products()
netCDF_products = list_of_products[list_of_products['format'] == 'NetCDF']
netCDF_products
```

Pick a product

Use the platform and product names from the previous block to select a Data Cube.

```
import utils.data_cube_utilities.data_access_api as dc_api
api = dc_api.DataAccessApi(config = '/home/localuser/.datacube.conf')

# Change the data platform and data cube here
# This data export notebook will only work for Landsat-7 datasets

platform = "LANDSAT_7"

# product = "ls7_ledaps_ghana"
# product = "ls7_ledaps_kenya"
# product = "ls7_ledaps_senegal"
# product = "ls7_ledaps_sierra_leone"
# product = "ls7_ledaps_tanzania"
product = "ls7_ledaps_vietnam"

# Get Coordinates
coordinates = api.get_full_dataset_extent(platform = platform, product = product)

# Print out Latitude, Longitude and Time Extents
latitude_extents = (min(coordinates['latitude'].values),max(coordinates['latitude'].values))
print( latitude_extents )
longitude_extents = (min(coordinates['longitude'].values),max(coordinates['longitude'].values))
print( longitude_extents )
time_extents = (min(coordinates['time'].values),max(coordinates['time'].values))
print( time_extents )
```

```
(9.1764253745784181, 13.964805165051667)
(102.40430421277932, 108.93092407802477)
(numpy.datetime64('1999-09-08T03:13:19.000000000'), numpy.datetime64('2016-12-29T03:10:00.000000000'))
```

Visualize Data Cube Region

```
## The code below renders a map that can be used to orient yourself with the region.
from utils.data_cube_utilities.dc_display_map import display_map
display_map(latitude = latitude_extents, longitude = longitude_extents)
```



Pick a smaller analysis region and display that region

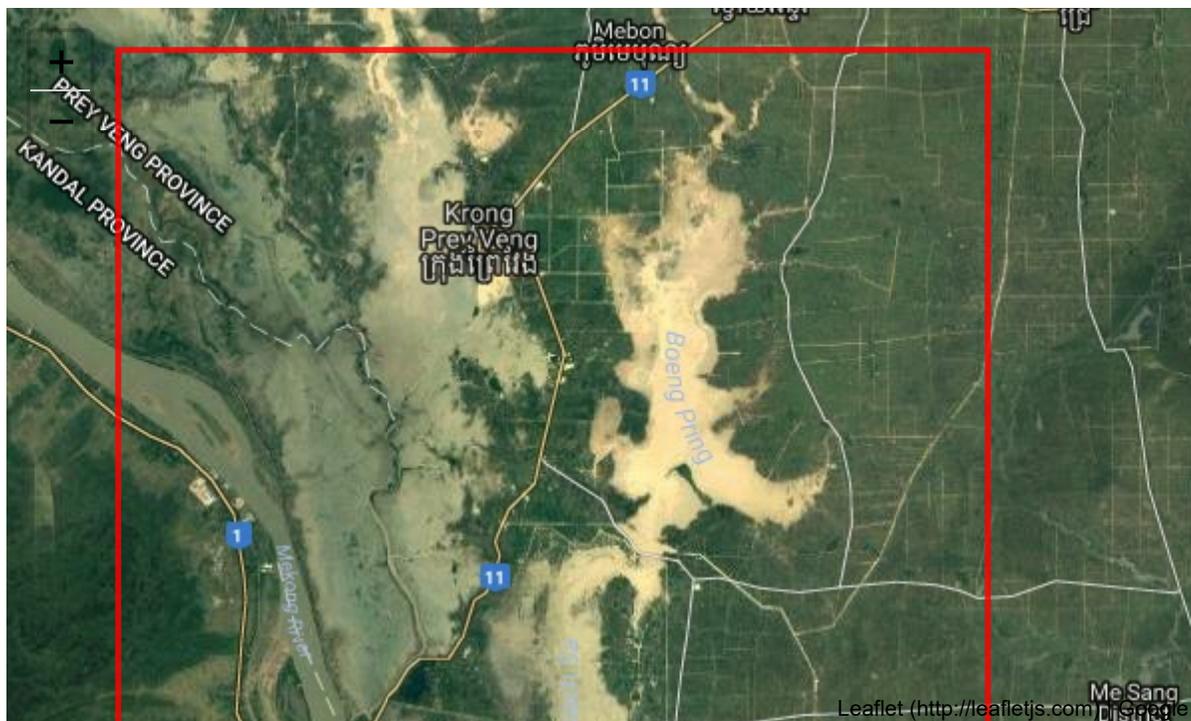
Try to keep your region to less than 0.2-deg x 0.2-deg for rapid processing. You can click on the map above to find the Lat-Lon coordinates of any location. Pick a time window of 1 year to keep the file small.

```
## Vietnam - Central Lam Dong Province ##
longitude_extents = (105.2, 105.5)
latitude_extents = (11.25, 11.55)

time_extents = ('2010-01-01', '2011-01-01')
print ( time_extents )

('2010-01-01', '2011-01-01')
```

```
from utils.data_cube_utilities.dc_display_map import display_map
display_map(latitude = latitude_extents, longitude = longitude_extents)
```



```
landsat_dataset = dc.load(latitude = latitude_extents,
                           longitude = longitude_extents,
                           platform = platform,
                           time = time_extents,
                           product = product,
                           measurements = ['red', 'green', 'blue', 'nir', 'swir1', 'swir2',
                                           'pixel_qa'])
```

```
landsat_dataset # this is a printout of the first few values of each parameter in the XARR
AY
```

```
<xarray.Dataset>
Dimensions:    (latitude: 1115, longitude: 1114, time: 10)
Coordinates:
  * time       (time) datetime64[ns] 2010-01-09T03:11:26 2010-02-26T03:12:02
  ...
  * latitude   (latitude) float64 11.55 11.55 11.55 11.55 11.55 11.55 11.55
  ...
  * longitude  (longitude) float64 105.2 105.2 105.2 105.2 105.2 105.2 105.2
  ...
Data variables:
  red         (time, latitude, longitude) int16 757 881 922 922 921 1045 ...
  green       (time, latitude, longitude) int16 806 828 874 851 941 986 ...
  blue        (time, latitude, longitude) int16 601 600 621 642 597 706 860
  ...
  nir         (time, latitude, longitude) int16 2038 1852 1620 1574 1714 ...
  swir1       (time, latitude, longitude) int16 1136 1001 565 483 729 1219
  ...
  swir2       (time, latitude, longitude) int16 529 500 327 240 442 616 414
  ...
  pixel_qa    (time, latitude, longitude) int32 66 66 66 66 66 66 66 66
  ...
Attributes:
  crs:         EPSG:4326
```

Create Several Common Application Products

Unpack pixel_qa

This is the Landsat-7 pixel quality data that is used to screen for clouds, shadows, snow, etc. These values can be quite valuable when doing an analysis in a GIS tool, but you will not need all of them.

```

import xarray as xr
import numpy as np

def ls7_unpack_qa( data_array , cover_type):

    land_cover_endcoding = dict( fill      = [1] ,
                                clear     = [322, 386, 834, 898, 1346],
                                water     = [324, 388, 836, 900, 1348],
                                shadow    = [328, 392, 840, 904, 1350],
                                snow      = [336, 368, 400, 432, 848, 880, 812, 944, 13
52],
                                cloud     = [352, 368, 416, 432, 848, 880, 912, 944, 13
52],
                                low_conf_cl = [322, 324, 328, 336, 352, 368, 834, 836, 84
0, 848, 864, 880],
                                med_conf_cl = [386, 388, 392, 400, 416, 432, 898, 900, 90
4, 928, 944],
                                high_conf_cl = [480, 992],
                                low_conf_cir = [322, 324, 328, 336, 352, 368, 386, 388, 39
2, 400, 416, 432, 480],
                                high_conf_cir = [834, 836, 840, 848, 864, 880, 898, 900, 90
4, 912, 928, 944],
                                terrain_occ = [1346, 1348, 1350, 1352]
                                )
    boolean_mask = np.isin(data_array, land_cover_endcoding[cover_type])
    return xr.DataArray(boolean_mask.astype(np.int8),
                        coords = data_array.coords,
                        dims = data_array.dims,
                        name = cover_type + "_mask",
                        attrs = data_array.attrs)

```

```

clear_xarray = ls7_unpack_qa(landsat_dataset.pixel_qa, "clear")
water_xarray = ls7_unpack_qa(landsat_dataset.pixel_qa, "water")
shadow_xarray = ls7_unpack_qa(landsat_dataset.pixel_qa, "shadow")
cloud_xarray = ls7_unpack_qa(landsat_dataset.pixel_qa, "cloud")

```

```

clean_xarray = xr.ufuncs.logical_or(clear_xarray , water_xarray).astype(np.int8).rename("c
lean_mask")

```

```

clean_mask = np.logical_or(clear_xarray.values.astype(bool),
                           water_xarray.values.astype(bool))

```

Spectral Indices

Below are a number of common spectral indices.

```

def NDVI(dataset):
    return ((dataset.nir - dataset.red)/(dataset.nir + dataset.red)).rename("NDVI")

```

```
def NDWI(dataset):  
    return ((dataset.green - dataset.nir)/(dataset.green + dataset.nir)).rename("NDWI")
```

```
def NDBI(dataset):  
    return ((dataset.swir2 - dataset.nir)/(dataset.swir2 + dataset.nir)).rename("NDBI")  
)
```

```
def EVI(dataset, c1 = None, c2 = None, L = None):  
    return ((dataset.nir - dataset.red)/((dataset.nir + (c1 * dataset.red) - (c2 * dataset.blue) + L))).rename("EVI")
```

```
ndbi_xarray = NDBI(landsat_dataset) # Urbanization - Reds  
ndvi_xarray = NDVI(landsat_dataset) # Dense Vegetation - Greens  
ndwi_xarray = NDWI(landsat_dataset) # High Concentrations of Water - Blues  
evi_xarray = EVI(landsat_dataset, c1 = 6, c2 = 7.5, L = 1 ) # Enhanced Vegetation Index
```

TSM

This is the Total Suspended Matter (TSM) index which measures the quality of water using a simple equation with one of Landsat bands. For the analysis below we will use the water pixels from the Landsat "pixel_qa" so that the code run faster than using the WOFS water analysis.

```
from utils.data_cube_utilities.dc_water_quality import tsm  
  
tsm_xarray = tsm(landsat_dataset, clean_mask = water_xarray.values.astype(bool) ).tsm
```

Combine Everything

```
combined_dataset = xr.merge([landsat_dataset,
                              clean_xarray,
                              clear_xarray,
                              water_xarray,
                              shadow_xarray,
                              cloud_xarray,
                              evi_xarray,
                              ndbi_xarray,
                              ndvi_xarray,
                              ndwi_xarray,
                              tsm_xarray])

# Copy original crs to merged dataset
combined_dataset = combined_dataset.assign_attrs(landsat_dataset.attrs)

combined_dataset # this is a printout of the first few values of each parameter in the XARRAY
```

```

<xarray.Dataset>
Dimensions:      (latitude: 1115, longitude: 1114, time: 10)
Coordinates:
  * time         (time) datetime64[ns] 2010-01-09T03:11:26 ...
  * latitude     (latitude) float64 11.55 11.55 11.55 11.55 11.55 11.55 ...
  * longitude    (longitude) float64 105.2 105.2 105.2 105.2 105.2 105.2 ...
Data variables:
  red           (time, latitude, longitude) int16 757 881 922 922 921 1045
  ...
  green        (time, latitude, longitude) int16 806 828 874 851 941 986
  ...
  blue         (time, latitude, longitude) int16 601 600 621 642 597 706
  ...
  nir          (time, latitude, longitude) int16 2038 1852 1620 1574 1714
  ...
  swir1        (time, latitude, longitude) int16 1136 1001 565 483 729 ...
  swir2        (time, latitude, longitude) int16 529 500 327 240 442 616
  ...
  pixel_qa     (time, latitude, longitude) int32 66 66 66 66 66 66 66 66
  ...
  clean_mask   (time, latitude, longitude) int8 0 0 0 0 0 0 0 0 0 0 0 0
  ...
  clear_mask   (time, latitude, longitude) int8 0 0 0 0 0 0 0 0 0 0 0 0
  ...
  water_mask   (time, latitude, longitude) int8 0 0 0 0 0 0 0 0 0 0 0 0
  ...
  shadow_mask  (time, latitude, longitude) int8 0 0 0 0 0 0 0 0 0 0 0 0
  ...
  cloud_mask   (time, latitude, longitude) int8 0 0 0 0 0 0 0 0 0 0 0 0
  ...
  EVI          (time, latitude, longitude) float64 0.6178 0.3679 0.2797 ...
  NDBI         (time, latitude, longitude) float64 -0.5878 -0.5748 -0.6641
  ...
  NDVI         (time, latitude, longitude) float64 0.4583 0.3553 0.2746 ...
  NDWI         (time, latitude, longitude) float64 -0.4332 -0.3821 -0.2991
  ...
  tsm          (time, latitude, longitude) float64 0.0 0.0 0.0 0.0 0.0 0.0
  ...
Attributes:
  crs:         EPSG:4326

```

Export CSV

This section will be used to create a CSV export file for a given pixel. You will identify the pixel by selecting a specific Lat-Lon position and then the code will find the closest pixel to that point (nearest neighbor). Use the map at the top of this notebook to view your region and pick a Lat-Lon location. You can find an exact location by clicking on the map. The CSV file will contain the time series data for each XARRAY parameter.

```

# Lat and Lon coordinates extracted from the map above
pixel_lat = 11.3972
pixel_lon = 105.3528

```

```
pixel = combined_dataset.sel(latitude = pixel_lat,
                             longitude = pixel_lon,
                             method = 'nearest') # nearest neighbor selection
```

```
import xarray as xr
import csv

def ts_pixel_to_csv(pixel: xr.Dataset,
                   csv_file_name: str):
    def __yield_from_time_axis(px):
        for i in range(len(px.time)):
            yield px.isel(time = i)
    def __format_time(t):
        return t

    with open(csv_file_name, 'w') as out:
        csv_out = csv.writer(out)
        column_names = ['time'] + list(pixel.data_vars)
        csv_out.writerow(column_names)

        for row in __yield_from_time_axis(pixel):
            csv_out.writerow([__format_time(row.time.values)] + [row[key].values for key in list(pixel.data_vars)])
```

```
csv_name = 'test.csv'
```

```
ts_pixel_to_csv(pixel, csv_name)
```

Export GeoTIFF

This section will be used to create a GeoTIFF export.

File formatting

```
import time
def time_to_string(t):
    return time.strftime("%Y_%m_%d_%H_%M_%S", time.gmtime(t.astype(int)/1000000000))
```

This function can be used to write a single time slice from an xarray to geotiff

```
from utils.data_cube_utilities import dc_utilities
def export_slice_to_geotiff(ds, path):
    dc_utilities.write_geotiff_from_xr(path,
                                      ds.astype(np.float32),
                                      list(combined_dataset.data_vars.keys()),
                                      crs="EPSG:4326")
```

For each time slice in a dataset we call `export_slice_to_geotif`

```
def export_xarray_to_geotiff(ds, path):
    for t in ds.time:
        time_slice_xarray = ds.sel(time = t)
        export_slice_to_geotiff(time_slice_xarray,
                                path + "_" + time_to_string(t) + ".tif")
```

Start Export

This is where we will start the GeoTIFF export and review the final product. The lines after this text box have been "commented out" so that they can be run at the end, after you have completed the creation of the XARRAY above and reviewed the data.

```
# export_xarray_to_geotiff(combined_dataset, "geotiffs/Landsat7")
```

Check to see what files exist in `geotiffs`

```
# !ls -lah geotiffs/*.tif
```

