SAR Doppler User Guide

Contents

This jupyter-book is to serve as a user manual on how to access, open, interact, visualize and plot SAR data. This refers to the calibrated geophysical range Doppler frequency shift retrievals from the ENVISAT ASAR wide-swath acqusitions obtained between 2002 and 2012.

Table of Contents

Below is a table of contents. Here you find topics on how to find, access and use the SAR Doppler data.

SAR Dataset Accessibility

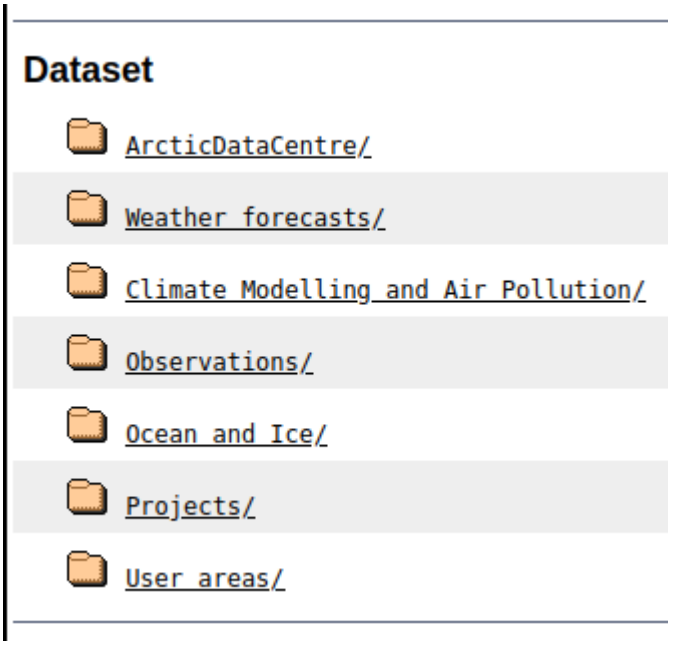
Contents


```
%%capture
!python3 -m venv .venv      # sets up a virtual environment
!source .venv/bin/activate  # Activates said environment

# Installing the required packages to current environment
!pip install -r ../requirements.txt

# The "%%capture" command ensures that the output
# (in this case directories) are not displayed.
```

There are several ways to find and access the SAR datasets. These are the datasets which contain the calibrated geophysical range Doppler frequency shift retrievals from the ENVISAT ASAR wide-swath acqusitions obtained between 2002 and 2012. In the following a description of some selected methods for finding and extracting these datasets are presented.

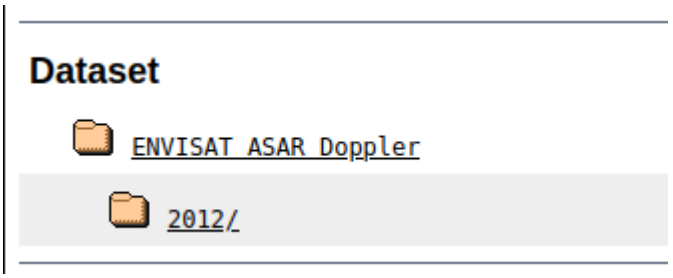
Find Data Through Web Search

All data is set to be freely available and some of it can be found in the MET Norway thredds catalog: https://thredds.met.no/thredds/catalog.html.



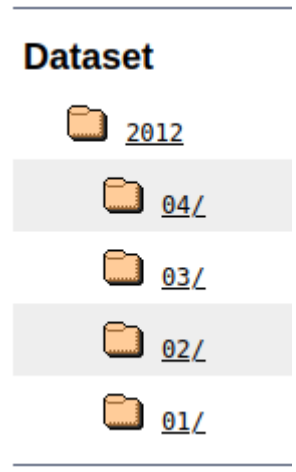
The ENVISAT ASAR datasets are located at: https://thredds.met.no/thredds/catalog/remotesensingenvisat/asar-doppler/catalog.html

Or just following this folder structure: Observations/Remotesensing_archive/ENVISAT_ASAR_Doppler:



Entering the subfolder, each individual netCDF-file is found under separate pathways depending on their respective dates. Wanting to access the files for a specific date, the datasets are listed with the following structure: YEAR/MONTH/DAY

Underneath the path to 2012/01/27 is shown:



Dataset

- 01
  - 31/
  - 30/
  - 29/
  - 28/
  - 27/

Dataset

- 27
  - ASA_WSDV2PRNMI20120127_215005_000614583111_00101_51839_0000.nc
  - ASA_WSDV2PRNMI20120127_215005_000614473111_00101_51839_0000.nc
  - ASA_WSDV2PRNMI20120127_214952_000597273111_00101_51839_0000.nc
  - ASA_WSDV2PRNMI20120127_214952_000597163111_00101_51839_0000.nc
  - ASA_WSDV2PRNMI20120127_205740_000624723111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_205740_000624603111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_205639_000624723111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_205639_000624603111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_205539_000624723111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_205539_000624603111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_205457_000598263111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_205457_000598163111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_200143_000616353111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_200143_000616233111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_200132_000624723111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_200132_000624603111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_200032_000624723111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_200032_000624603111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_195957_000598263111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_195957_000598163111_00100_51838_0000.nc
  - ASA_WSDV2PRNMI20120127_182343_000611603111_00099_51837_0000.nc
  - ASA_WSDV2PRNMI20120127_182343_000611483111_00099_51837_0000.nc

The entire list of files from the specified date are then accessible (the list goes on).

Access Data

Upon accessing a specific netCDF-file four different “Access”-options are available. These are “OPENDAP”, “HTTPServer”, “WCS” and “WMS”.

In the following the use of “OPENDAP” is explained closer. This is an easy and efficient way of accessing data. In the examples below the netCDF file “ASA_WSDV2PRNMI20120127_215005_000614583111_00101_51839_0000.nc” (the uppermost file under 2012/01/27) is used as an example.

OPENDAP - Using xarray:

The data is easily accessed through OPENDAP by the use of the xarray python package. Below is a an example on how to use xarray to open and investigate a desired dataset. This procedure makes it easy to inspect the Dimensions, Coordinates, Data Variables, Indexes and Attributes of the dataset in question.

```
# Import the required package: xarray
import xarray as xr

''' The backslashes serves as line shifts '''

# Providing the OPENDAP-url
OPENDAP_url = '''https://thredds.met.no/thredds/dodsC\
/remotesensingenvisat/asar-doppler/2012/01/27/\
ASA_WSDV2PRNMI20120127_215005_000612433111_00101\
_51839_0000.nc'''

# Using xarray to open the dataset using the OPENDAP-url
ds = xr.open_dataset(OPENDAP_url)

# Investigating the metadata as an xarray.Dataset
ds
```

xarray.Dataset

▸ Dimensions: (y: 602, x: 851, **zero_doppler_time**: 602)

▾ Coordinates:

zero_dopple… (zero_doppler_time) datetime64[ns] 2012-01-27T21:50:0…

▸ Data variables: (23)

▸ Indexes: (1)

▸ Attributes: (60)

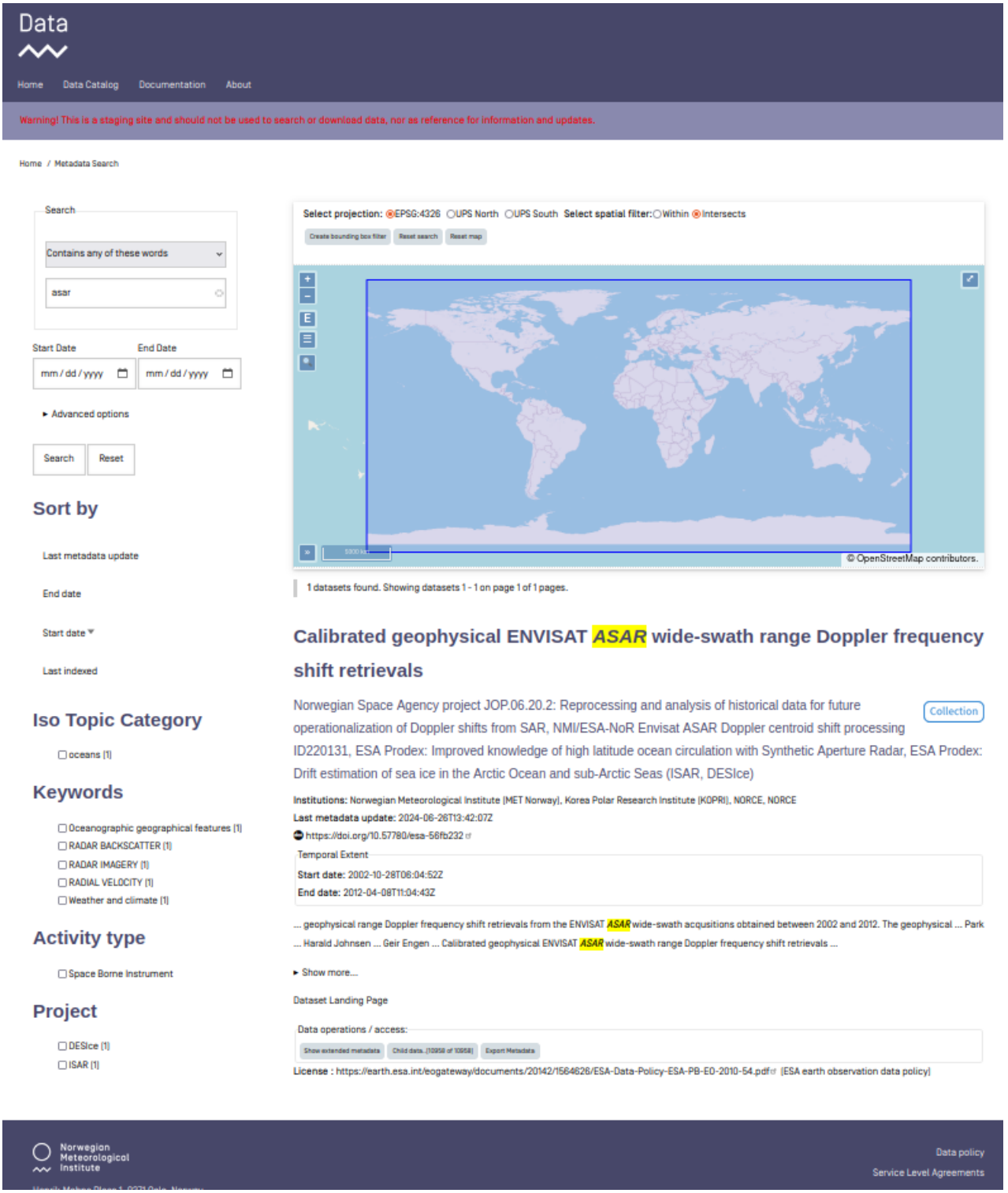
How to Find and Visualize Data with WMS (Web Map Service)

By the Use of data.met.no

By using data.met.no it is possible to both find and visualise datasets. The web search interface can be accessed from the “Data Catalog” menu item, or directly at https://data.met.no/metsis/search. As seen below the search interface consists of a map and a series of filters.

NB! The fact that the image below showcase a staging site can be ignored. The Data

Catalog of https://data.met.no/metsis/search have the same functionalities.



The map provides a pagination of available datasets in the metadata catalog [max/min longitude/latitude rectangle], sorted to showcase the latest additions first. One can also interact with the map to better diplay the results, and to perform data search.

- “Select Projection” located just above the map can be altered to change the map projection. “Spatial filter” can be set to both “Within” and “Intersects”.

- The “Create bounding box”-button enables to set a bounding box directly on the map and works as a filter on the results.
- The “Reset Search”-button clears the filters and starts a new search.
- The “Reset Map”-button resets the map.

Map widgets allows direct interaction with the map:

- +/-: Zoom in/out.
- E: Zooms to the extent of the displayed datasets.
- Menu tag: Opens side panel where WMS Layers, Features and Base Layers can be altered.
- Magnifying glass: Enables searching for location names.
- ‘>>’: Showing the location in an overview world map.
- Upper right hand widget: Full screen mode

Search filters can also be used to find the desired datasets. The results are updated dynamically when filters are selected. These allows:

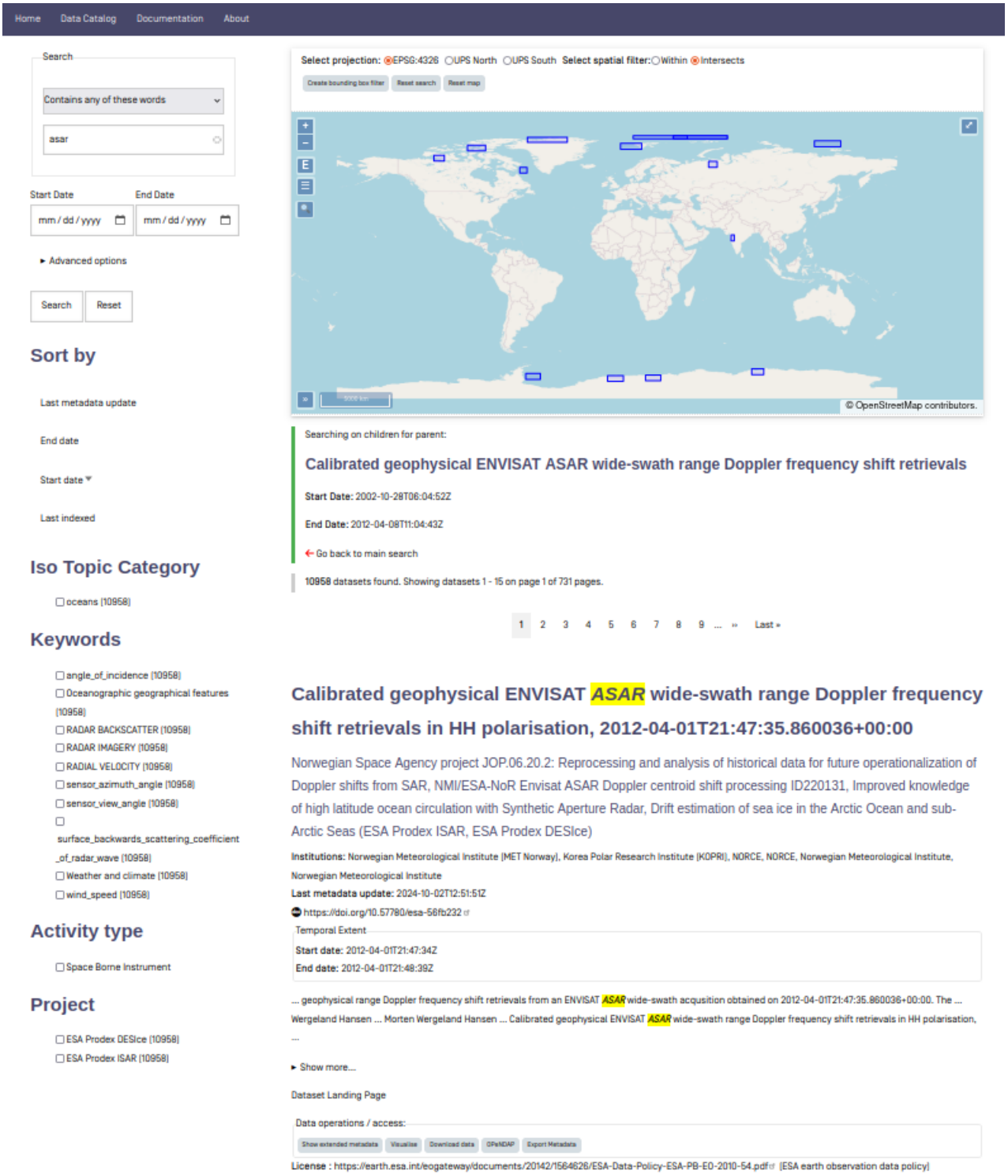
- A full text search block where the options “Contains all of these words” and “Contains any of these words” are eligible.
- Start and end date of the desired datasets.
- An option named “Has children” which can be ticked to determine whether datasets are parents with children (i.e. records of the same type).
- The desired sorting mechanism (Last metadata update, End date, Start date, Last indexed).
- Isotopic categories: The general subjects for which the geospatial data may be relevant, as defined by the ISO standard.
- Keywords: Keywords from a controlled vocabulary.
- Activity type: The nature of the dataset(s) generation process (Numerical Simulation, Climate Indicator, In Situ Land-based station, Space Borne Instrument).
- Project: Datasets related to a certain project.

By clicking the “Reset”-button all filters are removed and a new search can be initiated.

To further visualize the data, simply click on “Child data..” under “Data operations / access:” under one of the parent datasets that correspond to the search criteria (as seen

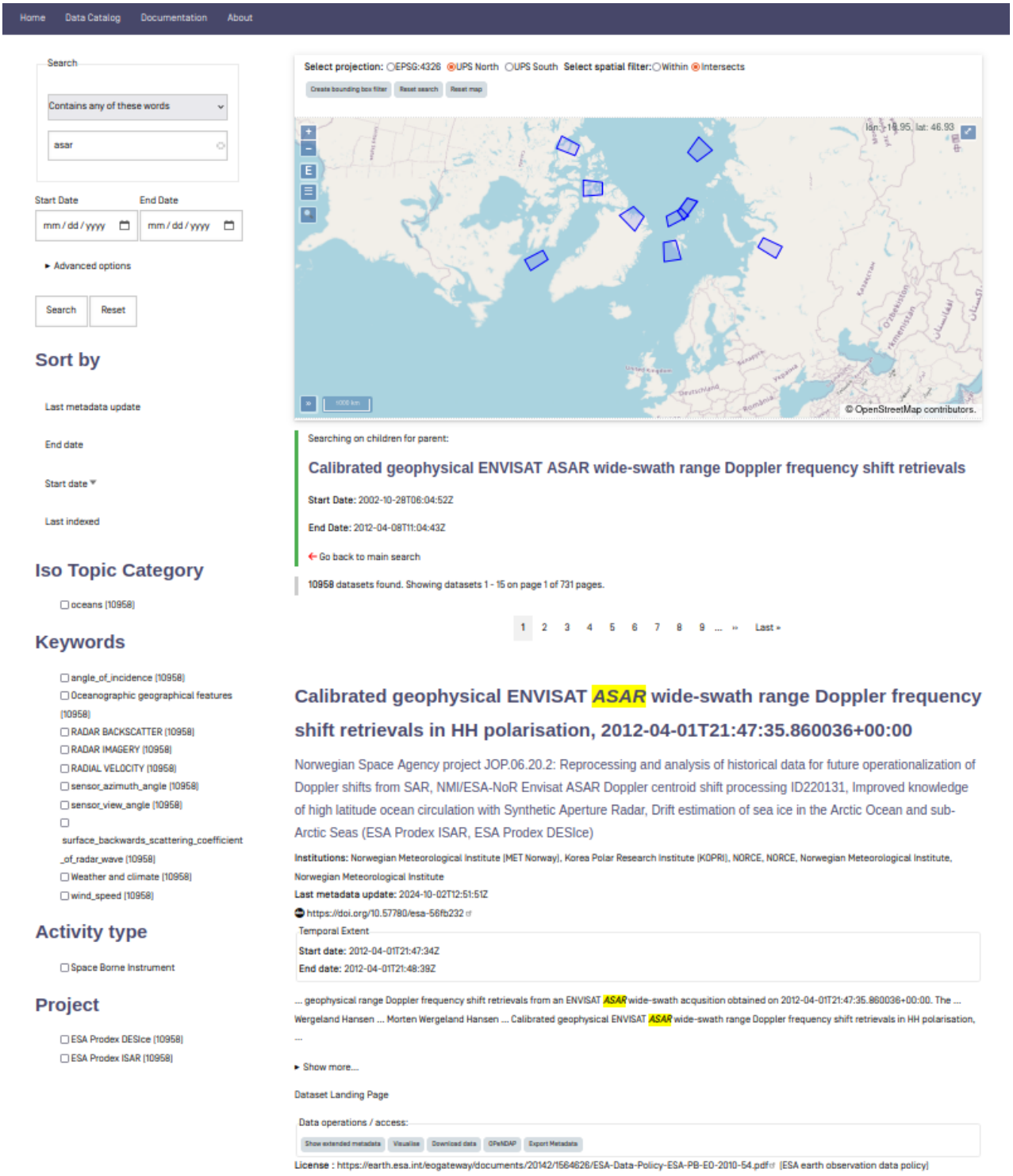
towards the bottom of the image above). In this example there is just this one parent dataset that correspond to the search criteria.

The list of children within this parent dataset are then listed. The bounding boxes for the different datasets also show up on the map. By clicking on a bouding box, the corresponding dataset are also singled out on top of the list below.

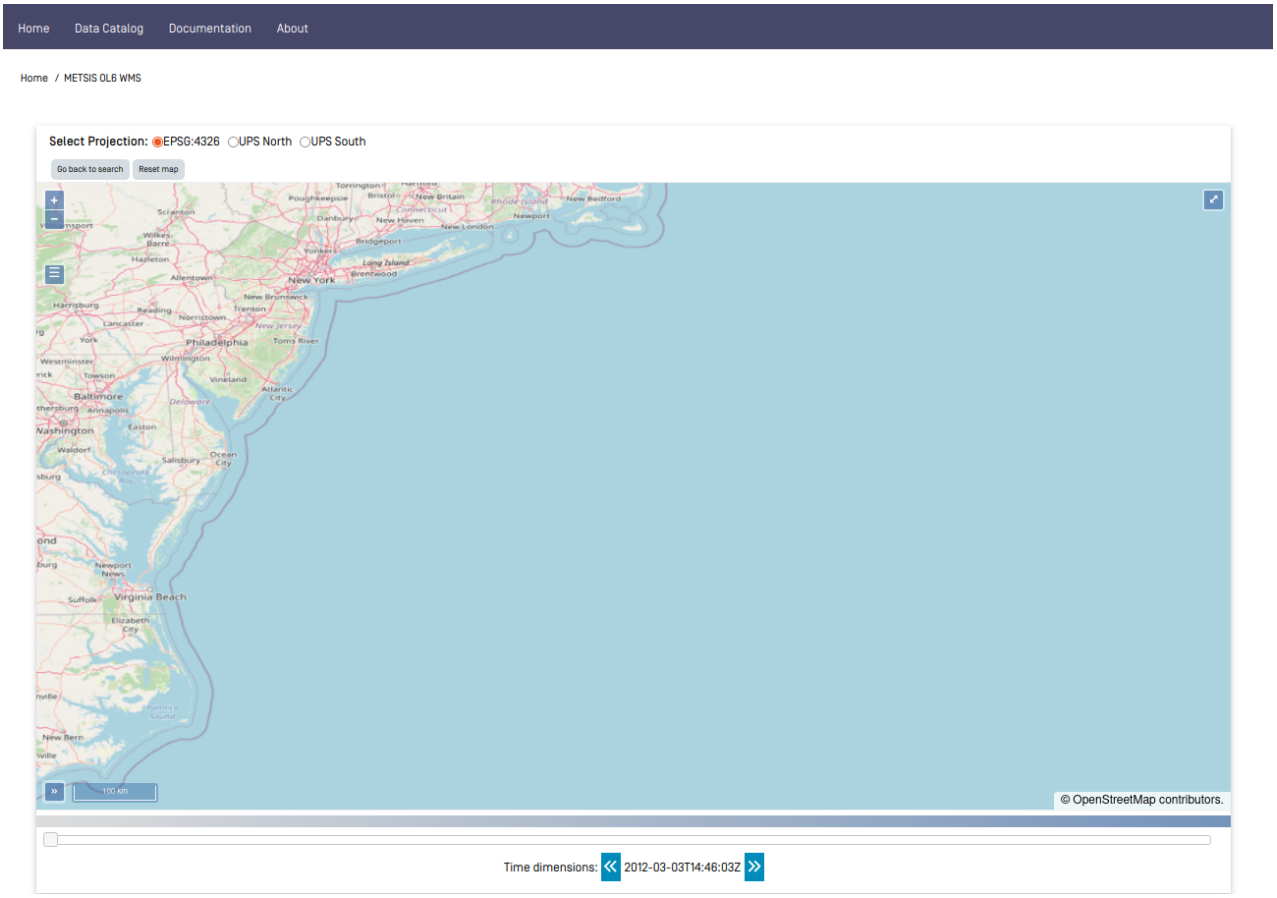


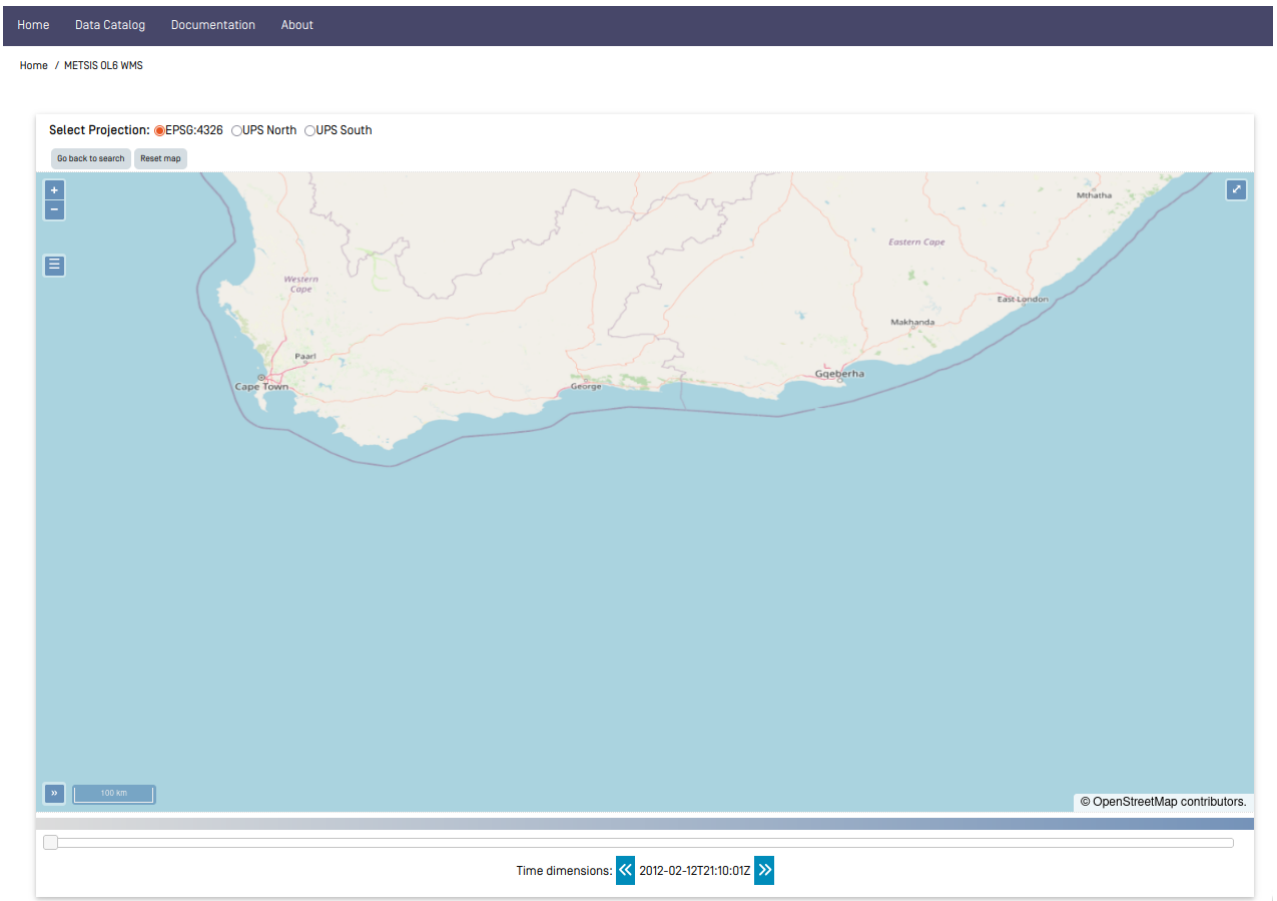
Changing the projection of the map might also make it easier to visualise the extent and

location of the different boundary boxes:



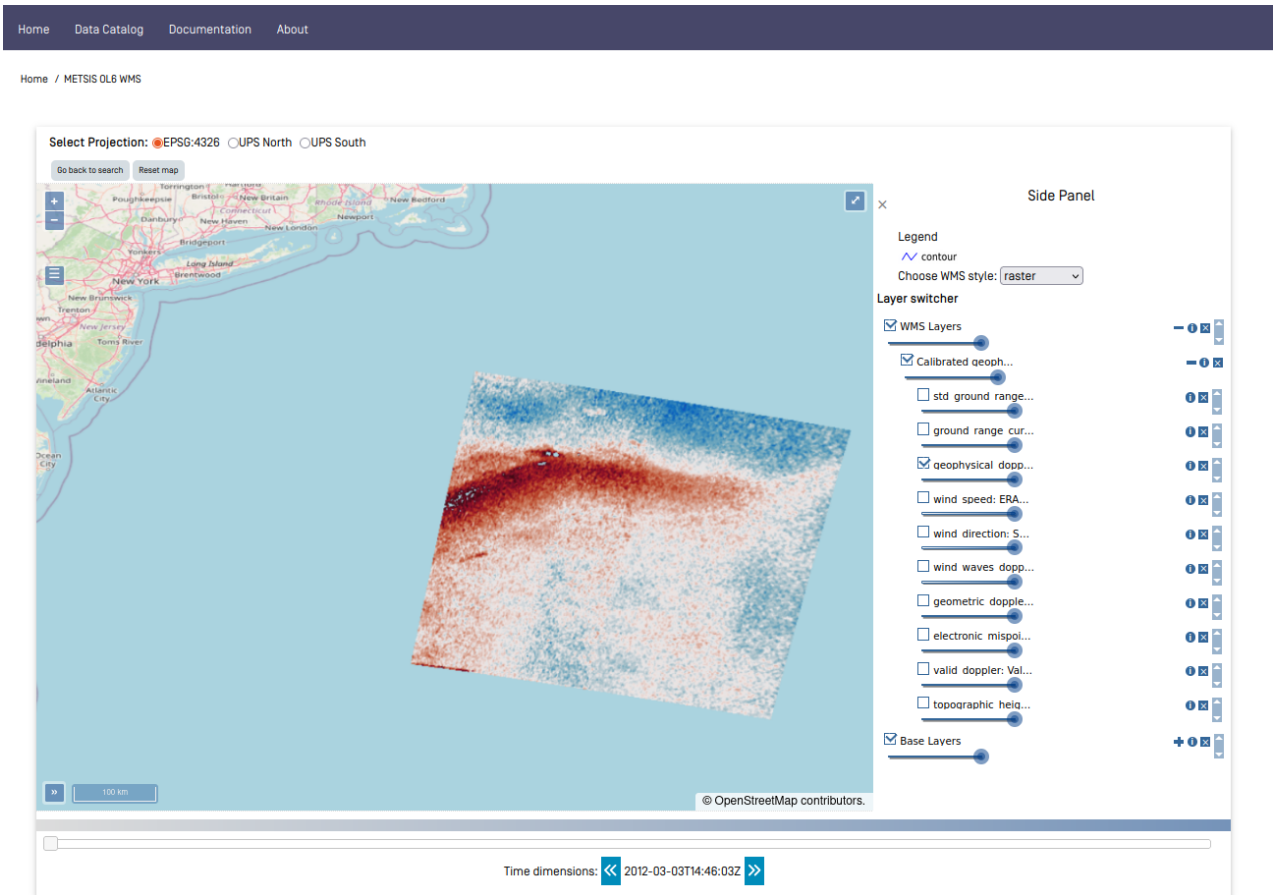
To visualise the data simply click on the “Visualise” option under “Data operations / access:” on a desired dataset. A close up area in question will then show up. Two clickable image examples are shown below:

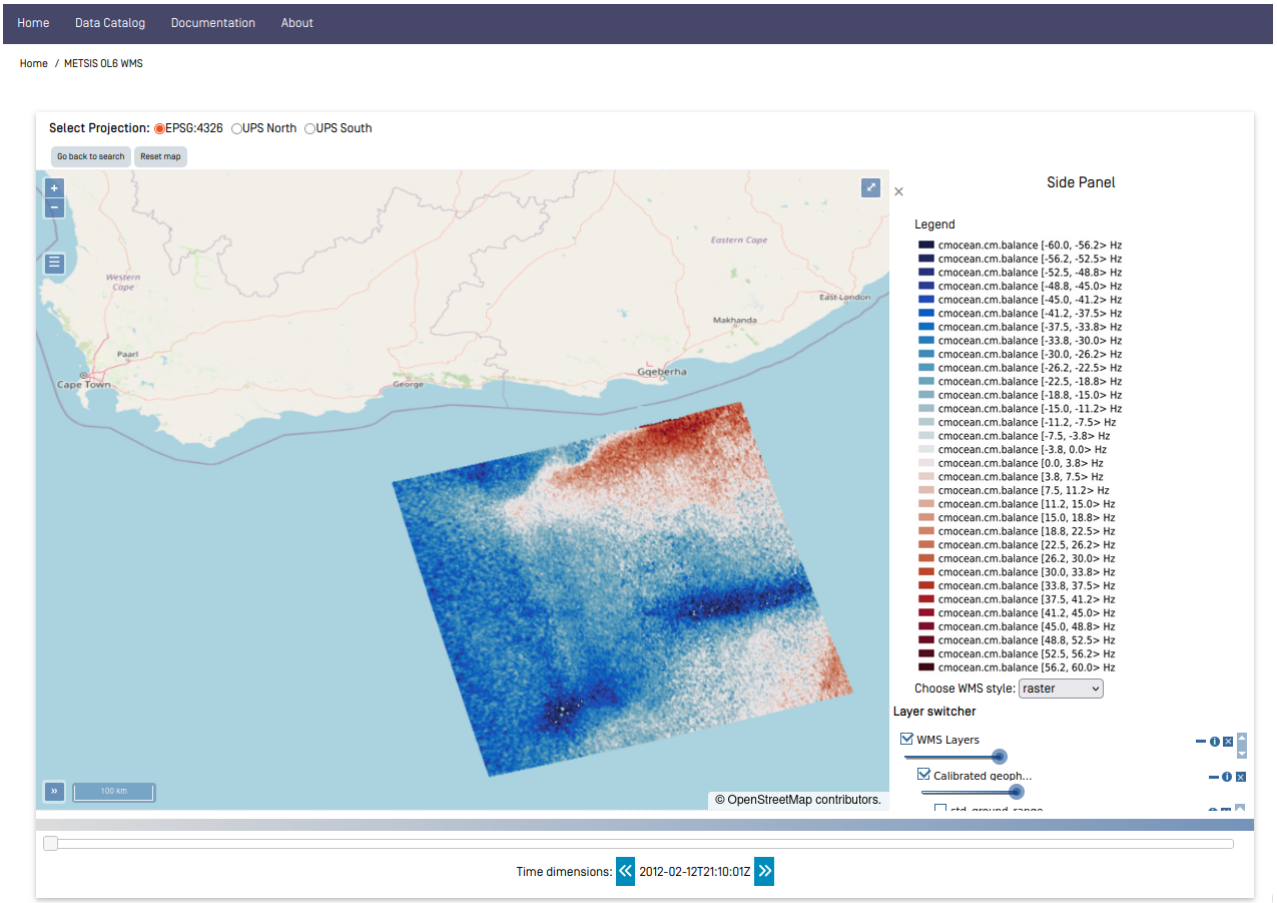




To visualise one specific variable of the dataset, e.g. the geophysical doppler, click on the

menu tab up towards the top left hand corner. Choose “raster” as the WMS style, and select your desired variable. Below the gephysical doppler is shown for the two examples above. The upper one shows an image from the Gulf Stream, and the lower one shows a part of the Agulhas Current close to the Cape of Good Hope off the coast of South Africa:





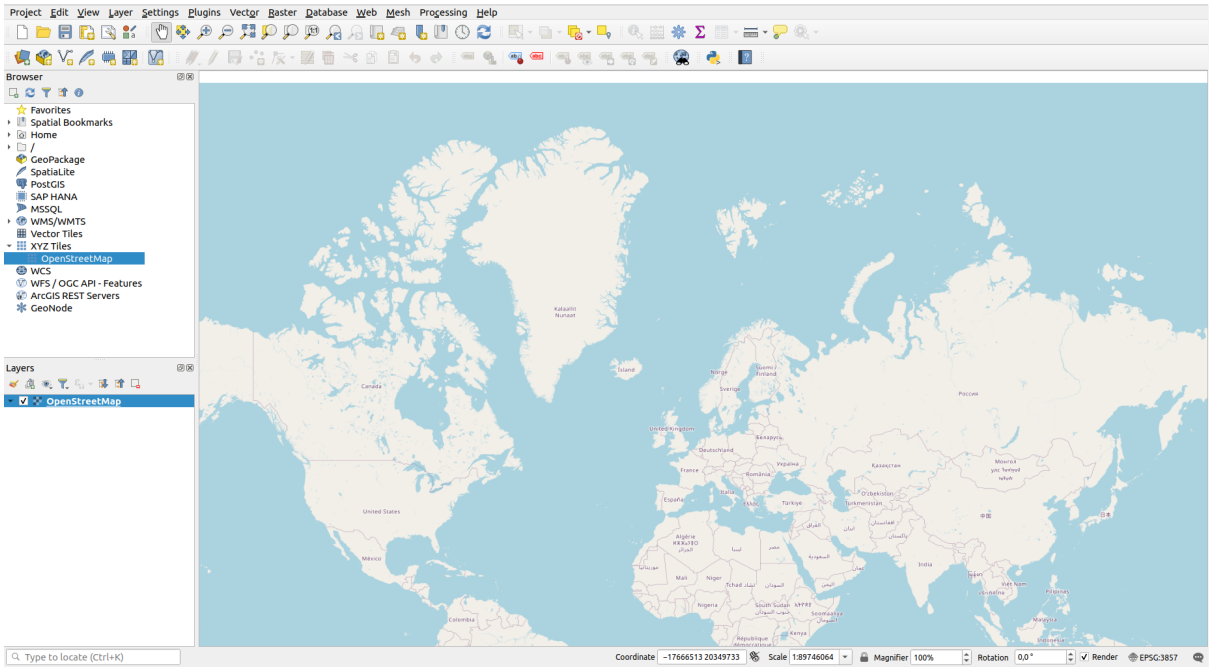
NB! Bare in mind that it might tak a bit of time for the visualisation tool to finish the visualization…

By the Use of QGIS

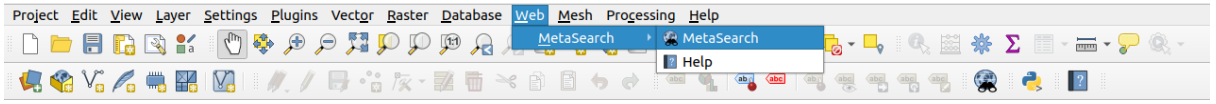
NB! In the following guide the endpoint “https://csw.s-enda-staging.k8s.met.no” must be exchanged with “https://data.csw.met.no” to be able to find any data.

The MET Norway’s S-ENDA CSW catalog service is also available through QGIS. Desired series/datasets can therefore also be found and inspected as follows:

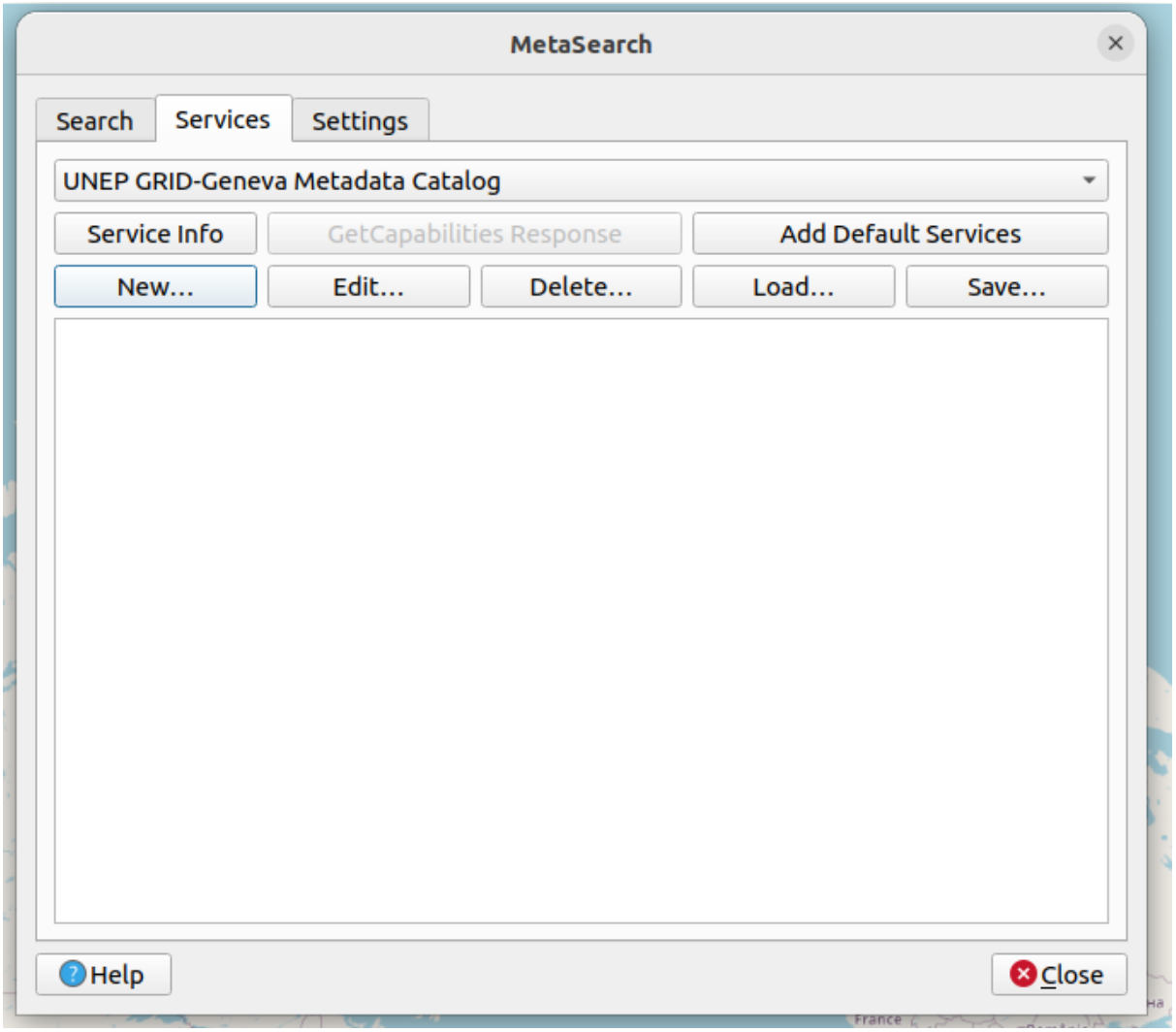
1. First open Qgis and select a map, e.g. the OpenStreetMap:



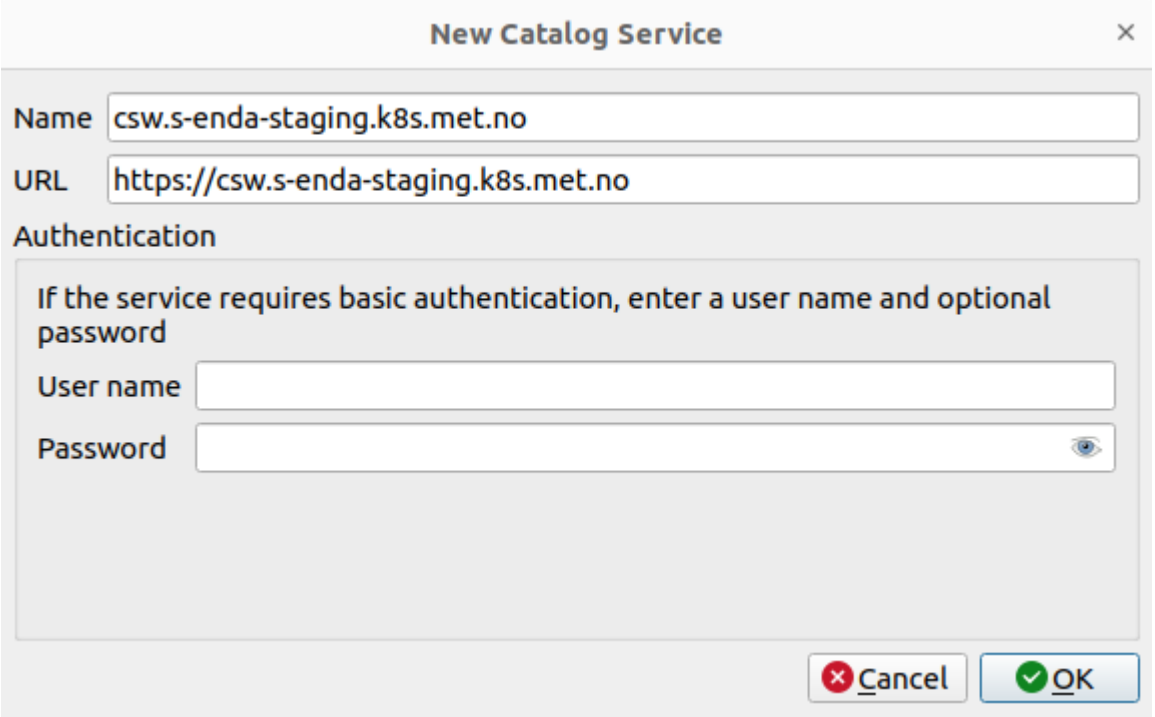
2. From the menu select “Web > MetaSearch > MetaSearch”.



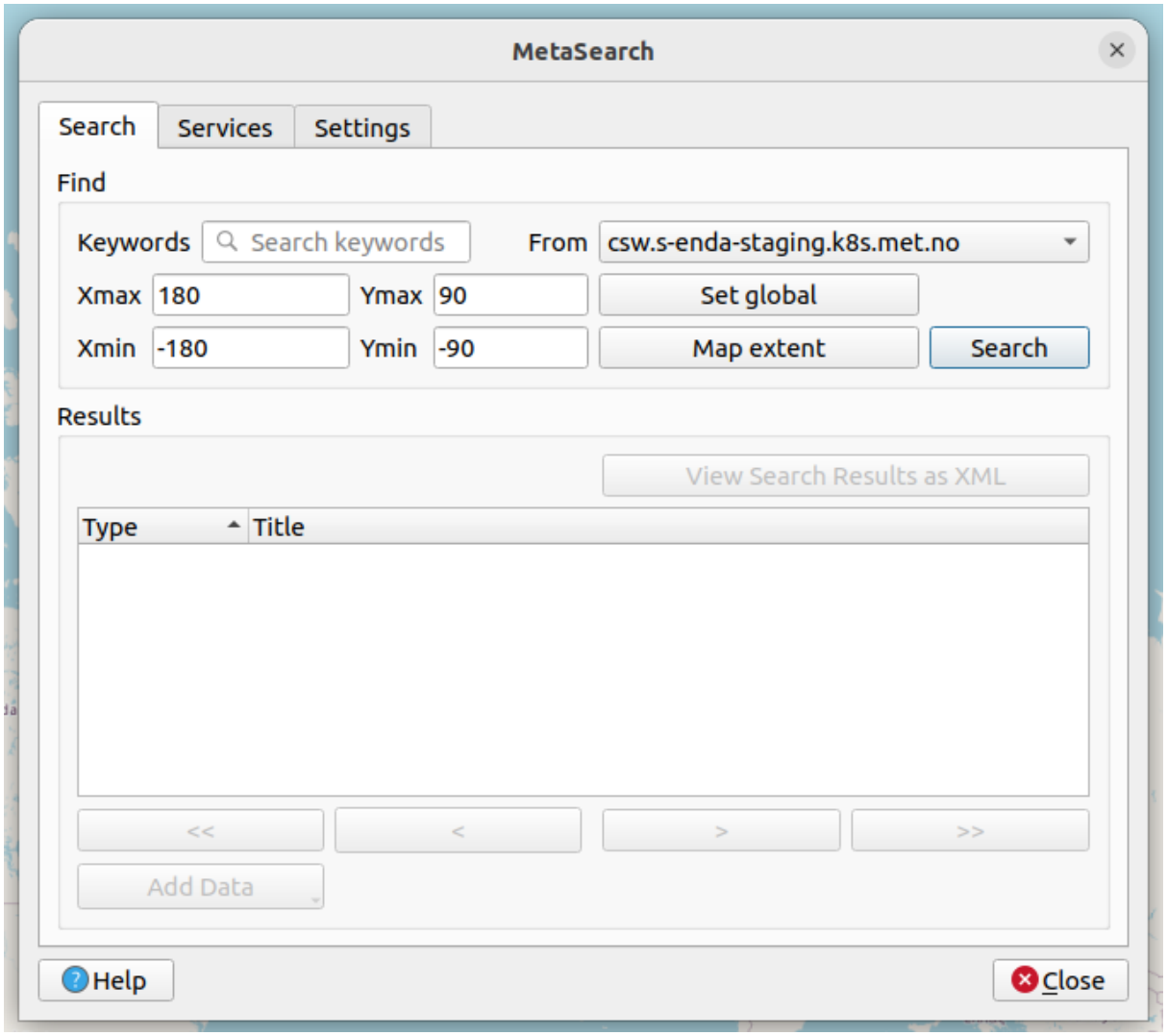
3. Select “Services > New” to open the “New Catalog Service”.



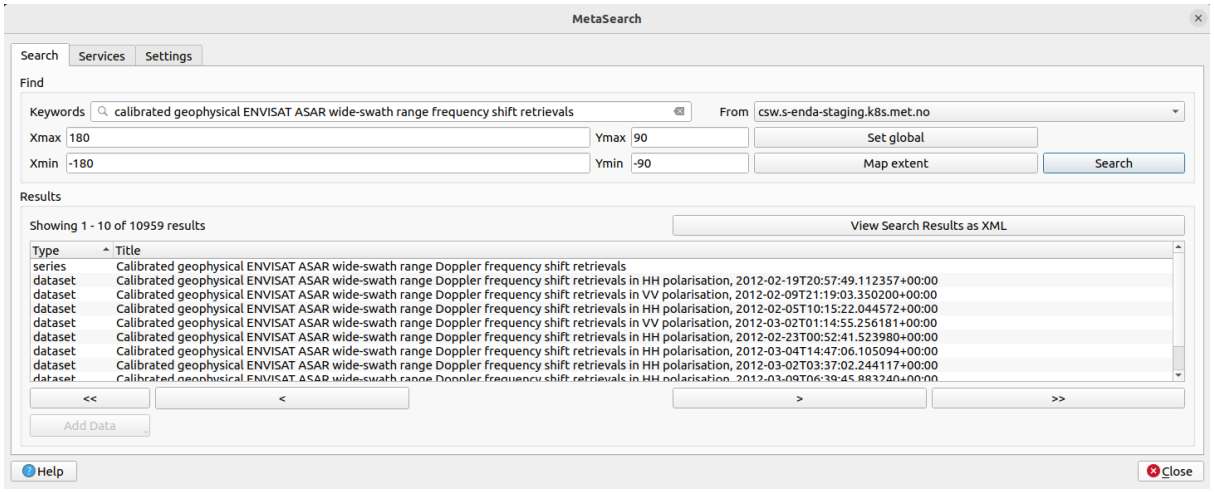
4. For the “Name” type “data.csw.met.no” (*not* “csw.s-enda-staging.k8s.met.no” as the image suggests). As for the “URI”, type “https://data.csw.met.no” (*not* “https://csw.s-enda-staging.k8s.met.no” as the image suggests). By then clicking “Ok” the required server is added.



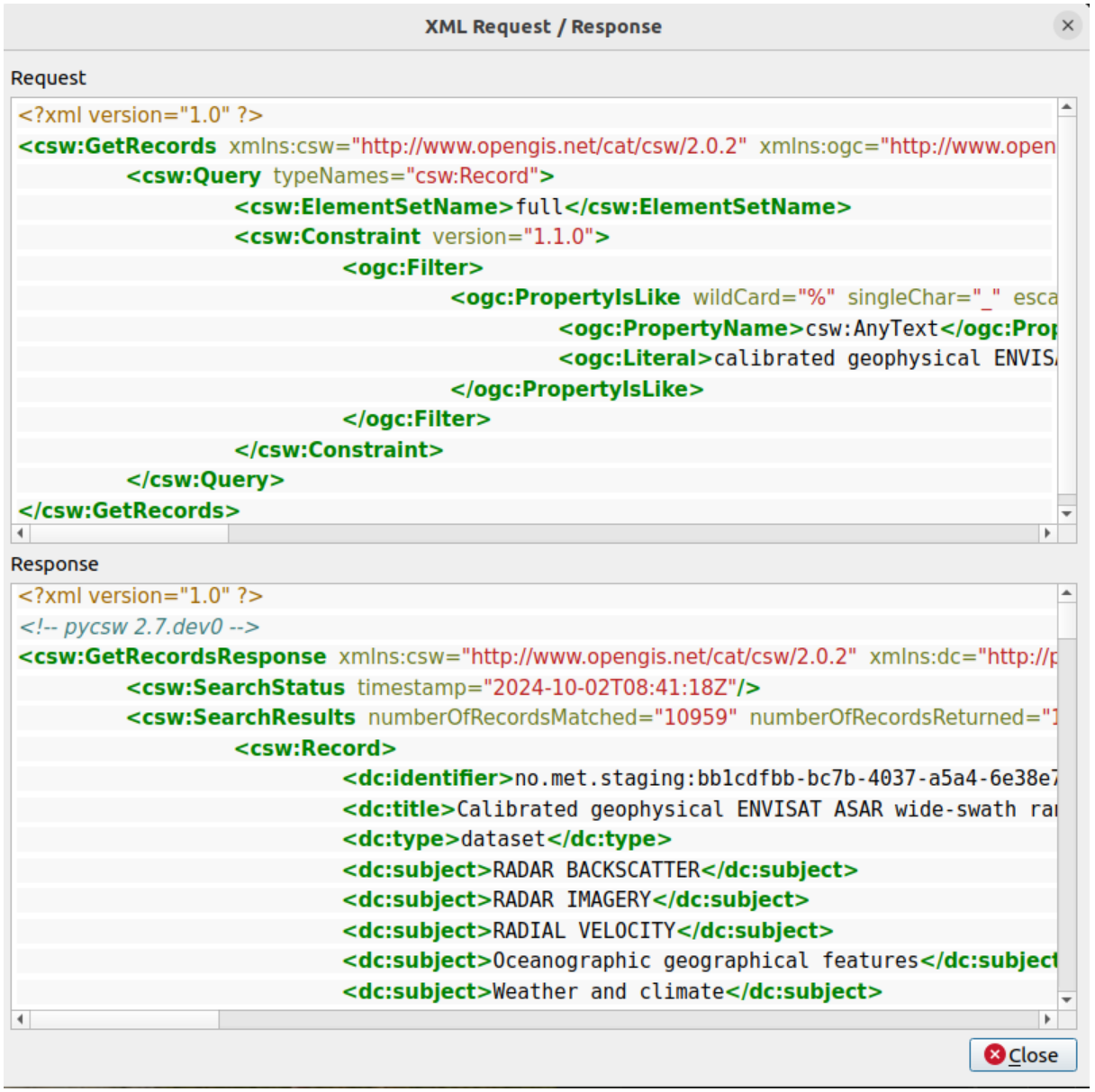
5. Without exiting “MetaSearch”, move back to the “Search” tab. Now the server that was just added is selected in the “From”-menu (this should now rather be “data.csw.met.no”).



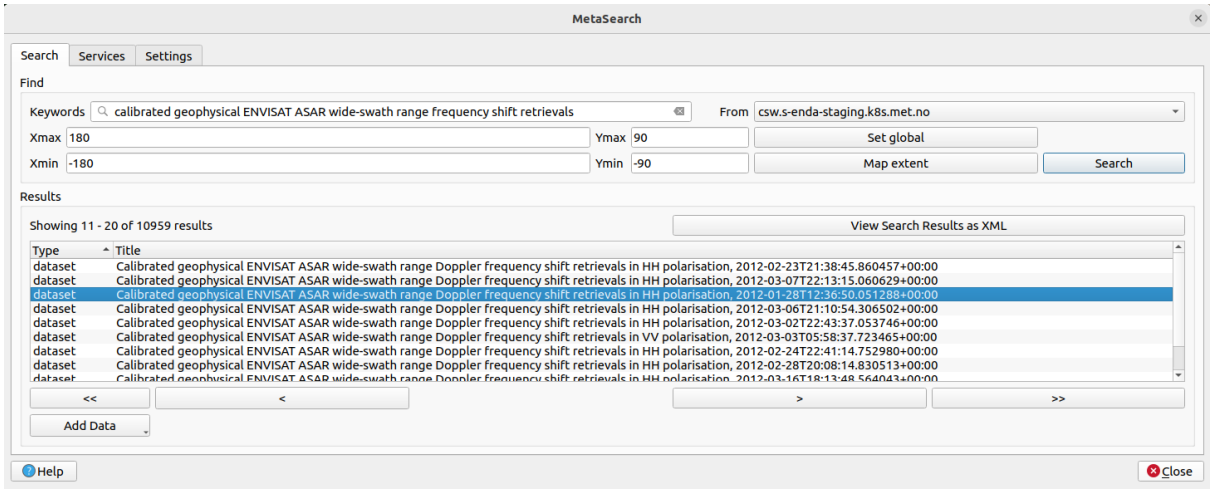
6. To get a list of the available series/datasets there is the option to add different search parameters under the “Search” tab. Adding keywords will single out the series and datasets with these as part of their “Title”. To find the “*calibrated geophysical ENVISAT ASAR wide-swath range frequency shift retrievals*” series/datasets the sequence in italics can be provided into the “Keywords” search tab, but “ENVISAT ASAR” or “Doppler” will also suffice. To actually search for datasets klick the “Search” option. The series/datasets will then show up in the “Results” section.



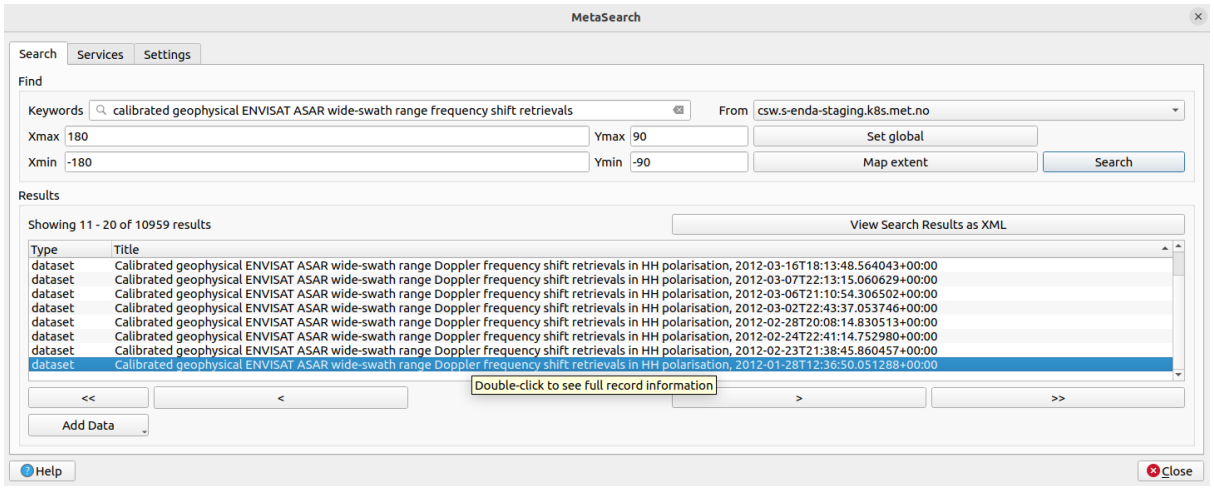
7. When a search is made, the results can alternatively be displayed as a scrollable list of XMLs. This is easily done by clicking “View Search Results as XML” in the “MetaSearch” window. This will open a new window, namely “XML Request / Response”. Here the resulting series/datasets from the search are displayed as XML.

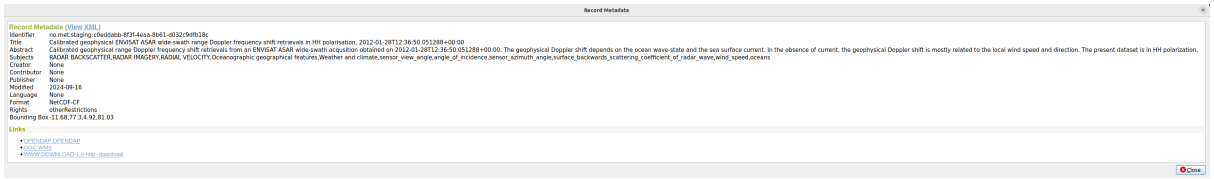


8. Moving back to the MetaSearch window the possibility to quickly display the geographical extent of selected series/dataset is available. By klicking one of series/ datasets a red bounding box will pop up on the map highligting the geographical extent of said dataset.

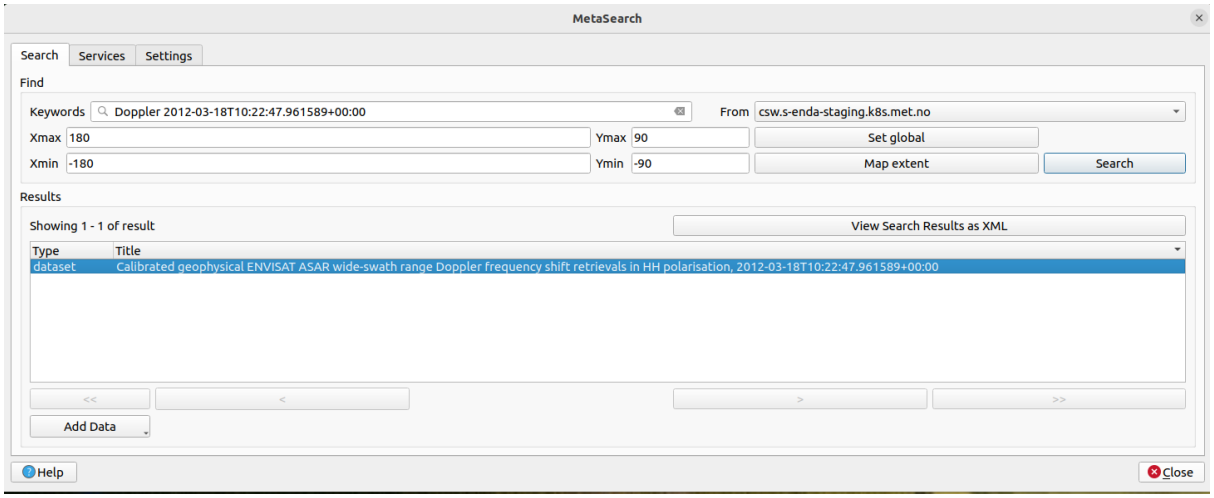


9. To further display the full record information alongside adherent links, double klick the selected series/dataset. A new window named “Record Metadata” will then be opened.

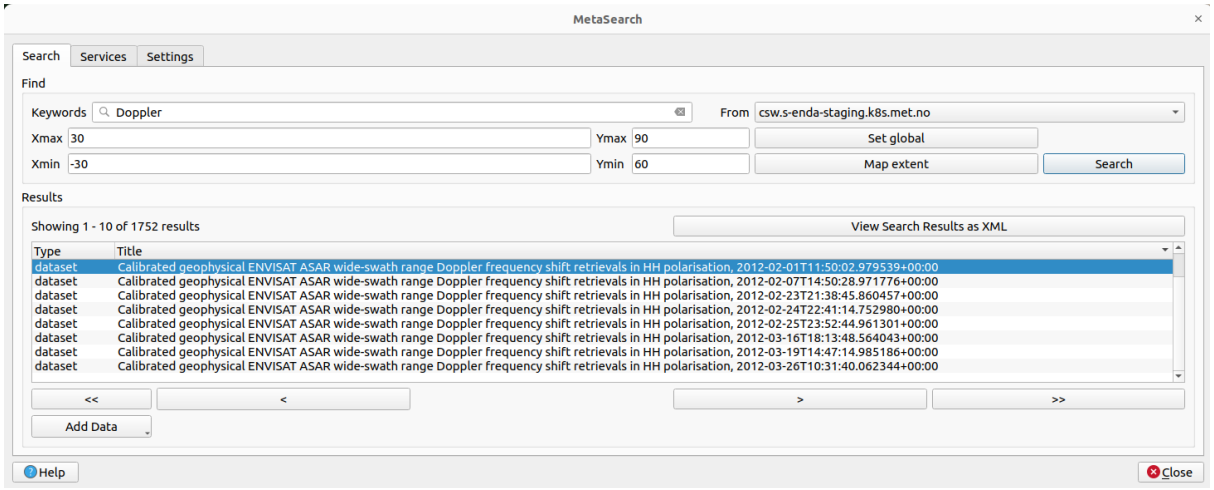




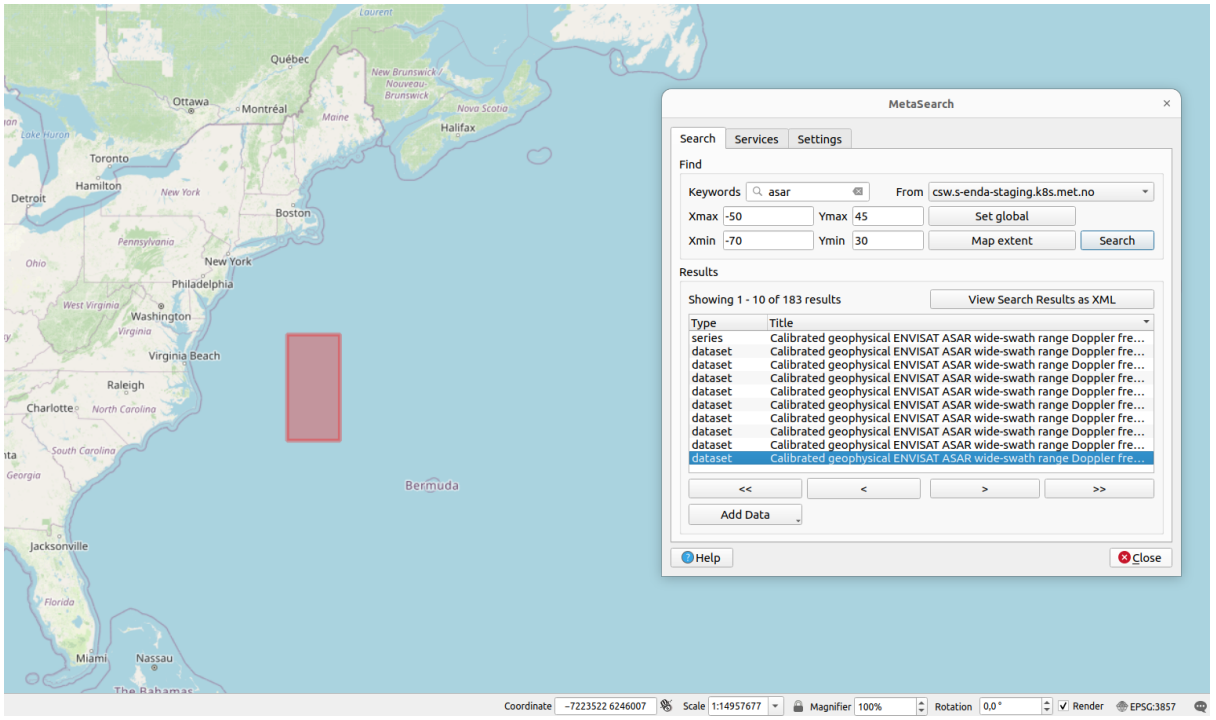
10. If the exact date and time of the desired dataset is known, this can be also added alongside keywords as “ENVISAT ASAR” or “Doppler” in the MetaSearch. This will single out this specific dataset.



11. There is also the possibility to alter the bounding box of the desired datasets. This box is altered by altering the latitude and longitude values found within the “Ymax/min” and “Xmax/min” search tabs, respectively. To reset these quickly to global default settings click “Set Global”. Clicking “Map Extent” will limit the bounding box to the extent of the map.

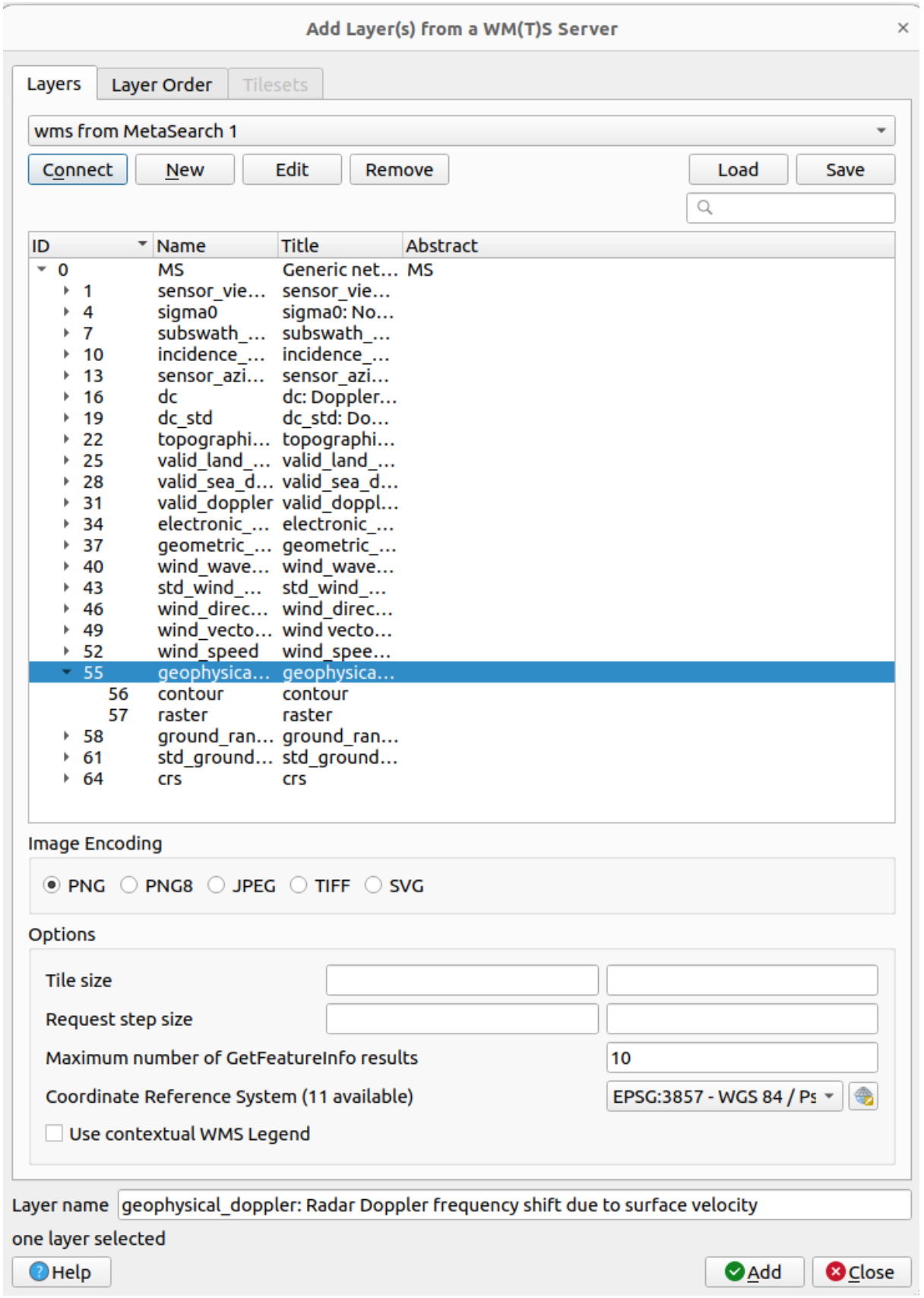


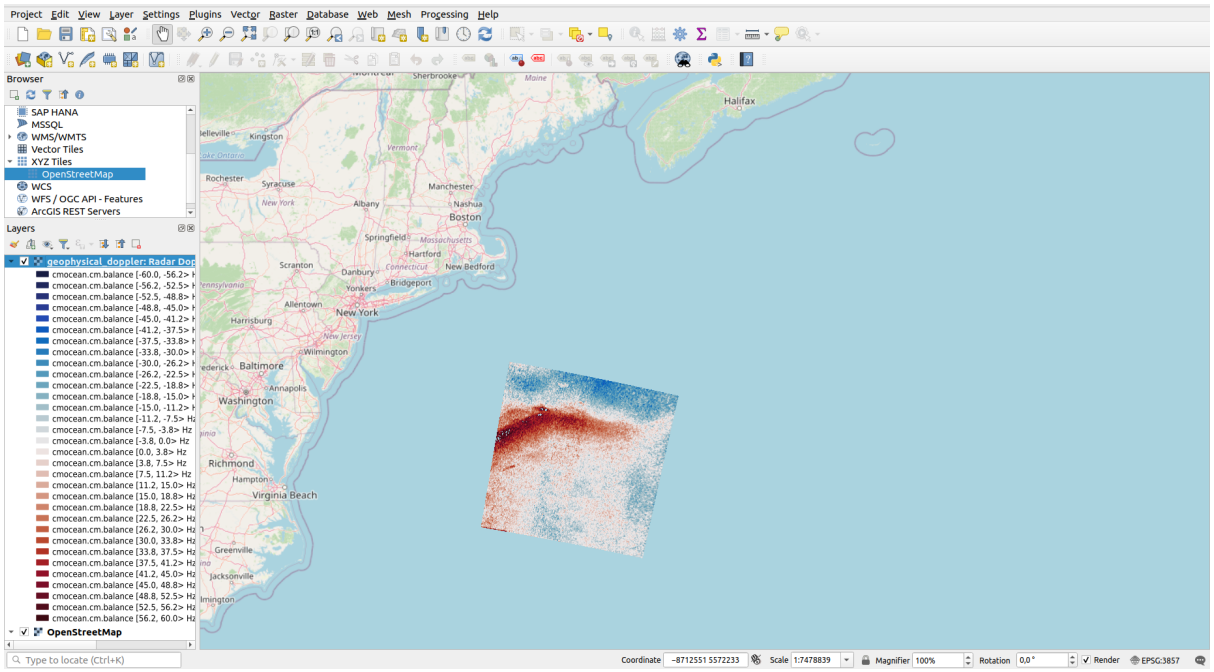
12. Finally, to visualise one specific variable of the dataset, e.g. the geophysical doppler, start by clicking the “Add Data” option when the desired dataset is selected. Select “Add WMS/WMTS”:



13. As shown below, choose the desired variable from the “Add Layer(s) from a WM(T)S Server” window that pops up. Click on “Add” in the lower right hand corner to add the layer. The variable will then show up as a layer on the left hand side menu along with

the selected map. A visualization of the selected data variable should now be visible on the map.




Project Edit View Layer Settings Plugins Vector Raster Database Web Mesh Processing Help
Browser
SAP HANA
MSSQL
WMS/WMTS
Vector Tiles
XYZ Tiles
OpenStreetMap
WCS
WFS / OGC API - Features
ArcGIS REST Servers
Layers
geophysical_doppler: Radar Dop
OpenStreetMap
Halifax
Boston
New York
Baltimore
Washington
Richmond
Virginia Beach
Type to locate (Ctrl+K)
Coordinate
Scale
Magnifier
Rotation
Render
EPSG:3857

Find Data Through CSW (Catalog Service on the Web)

Data can also be found through CSW (Catalog Service on the Web). An efficient and practiacl function to extract data which satisfes certain conditions can be found here metno/esa-coscaw-data-search. An example on how to import the required function from its folder, and how to use it is included below. The SearchCSW funtion takes the following arguments:

- time - This is a specific datetime.datetime to set as a starting point. Default is “now” (time = datetime.datetime.now(timezone(“utc”))), i.e. the time at each individual execution of the function.

- dt - The time intervall to search within. Depending on what “time” is selected, the search will higlight datasets which spans from (time - dt/2) and up to (time + dt/2). Default is dt = 24.

- text - A certain part of the dataset title to be served as a string. Default is text = None.

- boundary_box - Just what it sounds like; a geographically bounded box for which the desired datasets only need to intersect. It is structured as follows: [Westernmost Longitude, Southernmost Latitude, Easternmost Longitude, Northernmost Latitude]. Values are in degrees east and degrees north. Default spans the entire globe [-180, -90, 180, 90].

- endpoint - The endpoint for which to search through.

NB! There are provided two endpoints below. Before the SAR data is made publicly available at https://data.csw.met.no, the staging site https://csw.s-enda-staging.k8s.met.no is used. The latter is however only accessible to MET Norway employees. Others will have to switch to the publicly available endpoint (https://data.csw.met.no).

```
from fadg.find_and_collocate import SearchCSW
from datetime import datetime, timedelta

############ Time and dt ############

time_str = '2012-02-15 00:00:00'
''' Valid datetime string for the SearchCSW function.
    Default is the time right now; now = datetime.now() '''

time = datetime.strptime(time_str, '%Y-%m-%d %H:%M:%S')

dt = 24         # dt : float (default 24)
                # Total time interval in hours before and after the given t
                # (dt is centered around the selected time).

print(f'''Finding data within the timespan of:
      {time - timedelta(hours=dt/2)} and
      {time + timedelta(hours=dt/2)}.''')
print('\n')


############ Text ############

Text = "Doppler"
''' This text string needs to be part of
    the title of the files to be found.'''

print(f'Finding data with titles containing "{Text}".')
print('\n')

############ bbox ############

Boundary_Box = [34.9, 80.9, 35.1, 81]
''' This boundary box only have to be intersected by
    the geographical extent of the desired datasets.
    Default : [-180, -90, 180, 90] '''

print(f'''Finding data intersected by this specified boundary box:
                {Boundary_Box}.''')
print('\n')

############ endpoint ############

# Endpoint = "https://data.csw.met.no"                # The endpoint to u
                                                      # when data is made
                                                      # publicly availabl

Endpoint = "https://csw.s-enda-staging.k8s.met.no"    # Endpoint used in
                                                      # original version
                                                      # - only accessible
                                                      # internally at MET
                                                      # Norway
''' The site at which the data is located '''

print(f"Searching for data with endpoint set to: {Endpoint}.")
print('\n')
```

```
############ Finding the Corresponding datasets ############

sar = SearchCSW(time = time,
                dt = dt,
                text = Text,
                bbox = Boundary_Box,
                endpoint = Endpoint)




############### How many files are found ####################
if len(sar.urls) == 0:
    print('No data match the chosen credentials...')
elif len(sar.urls) == 1:
    print(f'''
There is {len(sar.urls)} file which match the chosen credentials!''')
else:
    print(f'''
There are {len(sar.urls)} files which match the chosen credentials!''')

print('\n')



################ Provide the found URLs ######################
sar.urls.sort()  # Sorts the list of files
print('''These are the Opendap-URLs of the datasets
which match the chosen credentials:''')
sar.urls
```

```
Finding data within the timespan of:
      2012-02-14 12:00:00 and
      2012-02-15 12:00:00.


Finding data with titles containing "Doppler".


Finding data intersected by this specified boundary box:
                [34.9, 80.9, 35.1, 81].


Searching for data with endpoint set to: https://csw.s-enda-staging.k8s.me
```

```
There are 5 files which match the chosen credentials!


These are the Opendap-URLs of the datasets
which match the chosen credentials:
```

```
['https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/2
 'https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/2
 'https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/2
 'https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/2
 'https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/2
```

Get Parent Datasets and their Children (or Dataset Series in ISO 19115) with OGC CSW

- Change identifier when no longer on staging site - 5 ALTERSATIONS REQUIRED!
- Change endpoint (in all links) when data is available on data.met.no: https://csw.s-enda-staging.k8s.met.no –> https://data.csw.met.no.

MET Norway organises datasets in parent-child relationships. A parent can be a set of Calibrated geophysical ENVISAT ASAR wide-swath range Doppler frequency shift retrievals, where the hyperlinklink provides the OGC CSW result of a search for “ASAR”.

The same search but with results provided in ISO format: https://csw.s-enda-staging.k8s.met.no/csw?SERVICE=CSW&VERSION=2.0.2&REQUEST=GetRecords&RESULTTYPE=results&TYPENAMES=csw:Record&ElementSetName=full&q=ASAR&outputschema=http://www.isotc211.org/2005/gmd.

Here, a field gmd:parentIdentifier provides the metadata identification of the parent dataset, i.e., no.met.staging:e19b9c36-a9dc-4e13-8827-c998b9045b54. CHANGE HERE

Note: If this document is opened as a PDF, all the links below will be incomplete. To see full links below open the document as a HTML or a jupyter notebook.

Get the parent dataset:

```
https://csw.s-enda-staging.k8s.met.no/csw?service=CSW&version=2.0.2&reque
CHANGE HERE
```

Get all its children:

```
https://csw.s-enda-staging.k8s.met.no/csw?SERVICE=CSW&VERSION=2.0.2&REQUE
CHANGE HERE
```

To find all parent datasets:

```
https://csw.s-enda-staging.k8s.met.no/csw?SERVICE=CSW&VERSION=2.0.2&REQUE
```

Find Data with OpenSearch

- Need to change the endpoint of all links below: https://csw.s-enda-staging.k8s.met.no… –> https://data.csw.met.no…

OpenSearch is a way for websites and search engines to publish search results in a standard and accessible format.

To find all datasets in the catalogue (Note: To see full links below open the page as a HTML or a jupyter notebook):

```
https://csw.s-enda-staging.k8s.met.no/?mode=opensearch&service=CSW&versio
```

Or datasets within a given time span (for instance: from 2012-02-01 to 2012-02-05):

```
https://csw.s-enda-staging.k8s.met.no/?mode=opensearch&service=CSW&versio
```

Or datasets within a geographical domain (defined as a box with parameters min_longitude, min_latitude, max_longitude, max_latitude - for instance [0, 70, 10, 80]):

```
https://csw.s-enda-staging.k8s.met.no/?mode=opensearch&service=CSW&versio
```

Or datasets with “ENVISAT ASAR wide-swath range Doppler frequency shift” in the title:

```
https://csw.s-enda-staging.k8s.met.no/?mode=opensearch&service=CSW&versior
```

Or datasets with all the three spesifications above:

```
https://csw.s-enda-staging.k8s.met.no/?mode=opensearch&service=CSW&versior
```

More Advanced Geographical Search with OGC CSW

PyCSW opensearch only supports geographical searches querying for a box. For more advanced geographical searches, one must write specific XML files.

The XML-files listed below are also available in the current notebooks-folder. Also, they are visible in their entirety if document is open as a HTML or as a jupyter notebook.

Here are some examples:

- To find all datasets containing a point:
  - XML-file name: my_xml_request_containing_a_point.xml
  - Here the coordinates of the point is 59.0 degrees north and 4.0 degrees east.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<csw:GetRecords
    xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    service="CSW"
    version="2.0.2"
    resultType="results"
    maxRecords="10"
    outputFormat="application/xml"
    outputSchema="http://www.opengis.net/cat/csw/2.0.2"
    xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2 http://schema
  <csw:Query typeNames="csw:Record">
    <csw:ElementSetName>full</csw:ElementSetName>
    <csw:Constraint version="1.1.0">
      <ogc:Filter>
        <ogc:Contains>
          <ogc:PropertyName>ows:BoundingBox</ogc:PropertyName>
          <gml:Point>
            <gml:pos srsDimension="2">59.0 4.0</gml:pos>
          </gml:Point>
        </ogc:Contains>
      </ogc:Filter>
    </csw:Constraint>
  </csw:Query>
</csw:GetRecords>
```

- To find all datasets intersecting a polygon:
  - XML-file name: my_xml_request_intersecting_a_polygon.xml
  - Here the polygon is [westernmost lon, southernmost lat, easternmost lon, northernmost lat] = [-5.00, -47.00, 20.00, 55.00]. The first and last coupled coordinate is the same to close the polygon.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<csw:GetRecords
    xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    service="CSW"
    version="2.0.2"
    resultType="results"
    maxRecords="10"
    outputFormat="application/xml"
    outputSchema="http://www.opengis.net/cat/csw/2.0.2"
    xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2 http://schema
  <csw:Query typeNames="csw:Record">
    <csw:ElementSetName>full</csw:ElementSetName>
    <csw:Constraint version="1.1.0">
      <ogc:Filter>
        <ogc:Intersects>
          <ogc:PropertyName>ows:BoundingBox</ogc:PropertyName>
          <gml:Polygon>
            <gml:exterior>
              <gml:LinearRing>
                <gml:posList>
                  47.00 -5.00 55.00 -5.00 55.00 20.00 47.00 20.00 47.00 -5
                </gml:posList>
              </gml:LinearRing>
            </gml:exterior>
          </gml:Polygon>
        </ogc:Intersects>
      </ogc:Filter>
    </csw:Constraint>
  </csw:Query>
</csw:GetRecords>
```

- To find all datasets intersecting a polygon within a given time span:
    - XML-file name: my_xml_request_intersecting_a_polygon_within_a_given_time_span.xml
    - Here the polygon is [westernmost lon, southernmost lat, easternmost lon, northernmost lat] = [-10.00, 70.00, 10.00, 80.00]. The first and last coupled coordinate is the same to close the polygon.
    - Here the start time is 2018-01-01 00:00.
    - Here the end tim is 2022-01-01 00:00.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<csw:GetRecords
    xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    service="CSW"
    version="2.0.2"
    resultType="results"
    maxRecords="100"
    outputFormat="application/xml"
    outputSchema="http://www.opengis.net/cat/csw/2.0.2"
    xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2 http://schema
  <csw:Query typeNames="csw:Record">
    <csw:ElementSetName>summary</csw:ElementSetName>
    <csw:Constraint version="1.1.0">
      <ogc:Filter>
        <ogc:And>
          <ogc:Intersects>
            <ogc:PropertyName>ows:BoundingBox</ogc:PropertyName>
            <gml:Polygon>
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList>
                    70.00 -10.00 80.00 -10.00 80.00 10.00 70.00 10.00 70.0
                  </gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
          </ogc:Intersects>
          <ogc:PropertyIsGreaterThanOrEqualTo>
            <ogc:PropertyName>apiso:TempExtent_begin</ogc:PropertyName>
            <ogc:Literal>2018-01-01 00:00</ogc:Literal>
          </ogc:PropertyIsGreaterThanOrEqualTo>
          <ogc:PropertyIsLessThanOrEqualTo>
            <ogc:PropertyName>apiso:TempExtent_end</ogc:PropertyName>
            <ogc:Literal>2022-01-01 00:00</ogc:Literal>
          </ogc:PropertyIsLessThanOrEqualTo>
        </ogc:And>
      </ogc:Filter>
    </csw:Constraint>
  </csw:Query>
</csw:GetRecords>
```

- To find all datasets intersecting a polygon within a given time span and with a certain text string:
    - XML-file name: my_xml_request_intersecting_a_polygon_within_a_given_time_span_and_certain_
    - Here the polygon is [westernmost lon, southernmost lat, easternmost lon, northernmost lat] = [-10.00, 70.00, 10.00, 80.00]. The first and last coupled coordinate is the same to close the polygon.

- Here the start time is 2012-02-01 00:00.
- Here the end tim is 2012-02-03 00:00.
- The recognizable string is “ENVISAT ASAR”.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<csw:GetRecords
    xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    service="CSW"
    version="2.0.2"
    resultType="results"
    maxRecords="100"
    outputFormat="application/xml"
    outputSchema="http://www.opengis.net/cat/csw/2.0.2"
    xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2 http://schema
  <csw:Query typeNames="csw:Record">
    <csw:ElementSetName>summary</csw:ElementSetName>
    <csw:Constraint version="1.1.0">
      <ogc:Filter>
        <ogc:And>
          <ogc:Intersects>
            <ogc:PropertyName>ows:BoundingBox</ogc:PropertyName>
            <gml:Polygon>
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList>
                    70.00 -10.00 80.00 -10.00 80.00 10.00 70.00 10.00 70.0
                  </gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
          </ogc:Intersects>
          <ogc:PropertyIsGreaterThanOrEqualTo>
            <ogc:PropertyName>apiso:TempExtent_begin</ogc:PropertyName>
            <ogc:Literal>2012-02-01 00:00</ogc:Literal>
          </ogc:PropertyIsGreaterThanOrEqualTo>
          <ogc:PropertyIsLessThanOrEqualTo>
            <ogc:PropertyName>apiso:TempExtent_end</ogc:PropertyName>
            <ogc:Literal>2012-02-03 00:00</ogc:Literal>
          </ogc:PropertyIsLessThanOrEqualTo>
          <ogc:PropertyIsLike wildCard="%" singleChar="_" escapeChar="\\">
            <ogc:PropertyName>dc:title</ogc:PropertyName>
            <ogc:Literal>%ENVISAT ASAR%</ogc:Literal>
          </ogc:PropertyIsLike>
        </ogc:And>
      </ogc:Filter>
    </csw:Constraint>
  </csw:Query>
</csw:GetRecords>
```

Query CSW Endpoint by the Use of Python

- Then, you can query the CSW endpoint and print the response text using, e.g., python (alter endpoint from 'https://csw.s-enda-staging.k8s.met.no' to https://data.csw.met.no):

```
import requests
import xarray as xr
import re
import sys

### Define the headers
headers = {'Content-Type': 'application/xml'}

### Specify the xml-file that should be used for the search
  # - As mentioned all the XML-files listed above can be found
  #   in the notebooks folder.

# my_xml_request = 'my_xml_request_containing_a_point.xml'

# my_xml_request = 'my_xml_request_intersecting_a_polygon.xml'

# my_xml_request = 'my_xml_request_intersecting_a_polygon_within_a_given_

my_xml_request = 'my_xml_request_intersecting_a_polygon_within_a_given_tir

# Open and read the XML file
with open(my_xml_request, 'r') as file:
    xml_data = file.read()

### Send the POST request


# response = requests.post('https://data.csw.met.no',
#                          data=xml_data,
#                          headers=headers)

response = requests.post('https://csw.s-enda-staging.k8s.met.no',
                         data=xml_data,
                         headers=headers)

# The response text
print(response.text)
print('\n')
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- pycsw 2.7.dev0 -->
<csw:GetRecordsResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" x
```

Extract the OPENDAP urls

Having recieved the response text, it is possible to extract the OPENDAP-urls. This can be read from the response text, but can also be easily extracted using the code snippet below:

```
''' The pattern 'https.*?\.nc(?:ml)?' is
"https://thredds.met.no/thredds/dodsC/{regardless_of_what_is_in_between}.n
where the "ml" ending is inculded only if found. '''

### Opendap url format
my_pattern= r'https://thredds.met.no/thredds/dodsC/.*?\.nc(?:ml)?'


### findall() function returns all non-overlapping matches of
  # my_pattern in string, as a list of strings
opendap_urls = re.findall(my_pattern, response.text)

# Sort the list of OPENDAP-urls by date and time
opendap_urls.sort()

# List of OPENDAP urls
print(f'List contains {len(opendap_urls)} urls:')
for url in opendap_urls:
    print(url)

# Check if there are any files - Statement if not
if len(opendap_urls) > 0:

    # Open the first dataset in the list of urls
    print('\n')
    print("Opening the first dataset with xarray:")
    ds = xr.open_dataset(opendap_urls[0])

else:
    ds = "No file(s) match the search criterias."


ds
```

```
List contains 10 urls:
https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/201
https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/201
https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/201
https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/201
https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/201
https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/201
https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/201
https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/201
https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/201
https://thredds.met.no/thredds/dodsC/remotesensingenvisat/asar-doppler/201


Opening the first dataset with xarray:
```

xarray.Dataset

▸ Dimensions: (y: 605, x: 852, **zero_doppler_time**: 605)

▾ Coordinates:

zero_dopple… (zero_doppler_time) datetime64[ns] 2012-02-01T10:11:3…

▸ Data variables: (23)

▸ Indexes: (1)

▸ Attributes: (60)

NOTE: There seems to be a server-side limit on the number of records returned in a single response, regardless of the “maxRecords” value in the resquest. It’s not uncommon for servers to have such limits to prevent excessively large responses. Here this limit appears to be 10 records for every request.

To retrieve the rest of the records, you can make use of the startPosition attribute. By setting startPosition=”11”, you can retrieve the next set of records starting from the 11th record.

Here’s how you would add it to one of the XML files listed above:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<csw:GetRecords
    xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    service="CSW"
    version="2.0.2"
    resultType="results"
    maxRecords="100"
    startPosition="11"
    outputFormat="application/xml"
    outputSchema="http://www.opengis.net/cat/csw/2.0.2"
    xsi:schemaLocation="http://www.opengis.net/cat/csw/2.0.2 http://schema
  <!-- rest of the XML file -->
</csw:GetRecords>
```

This way, you can “paginate” through the records by making multiple requests and incrementing startPosition each time.

Query CSW Endpoint by the Use an HTTP POST (From the Terminal)

- Alternatively, one can also use an HTTP POST request to query to the PyCSW server directly from the terminal. The steps are as follows:
    1. Make sure that you have one of the listed XML-files above saved, or one that you have composed for your search.

    2. Then, use curl (a command-line tool for making HTTP requests) to send a POST request to the PyCSW server. An example might look like (alter endpoint from ‘https://csw.s-enda-staging.k8s.met.no’ to https://data.csw.met.no):

```
### The original bash commands:

'''
%%bash
# curl -X POST -H "Content-Type: application/xml" -d \
# @my_xml_request_intersecting_a_polygon_within_a_given_time_span_and_cer
# https://data.csw.met.no
'''

'''
curl -X POST -H "Content-Type: application/xml" -d \
@my_xml_request_intersecting_a_polygon_within_a_given_time_span_and_certa
https://csw.s-enda-staging.k8s.met.no
'''

### Using subprocess just to make the output readable when opened as a HT

import subprocess

# Define the curl command
curl_command = [
    "curl", "-X", "POST", "-H", "Content-Type: application/xml", "-d",
    "@my_xml_request_intersecting_a_polygon_within_a_given_time_span_and_
    "https://csw.s-enda-staging.k8s.met.no"
]

# Run the curl command and capture the output
result = subprocess.run(curl_command, capture_output=True, text=True)

# Print the output
print("Standard Output:\n")
print(result.stdout)
print("\nStandard Error:\n")
print(result.stderr)
```

```
Standard Output:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- pycsw 2.7.dev0 -->
<csw:GetRecordsResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2" x

Standard Error:

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Cu
                                 Dload  Upload   Total   Spent    Left  Sp

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
100 25859  100 23977  100  1882  85775   6732 --:--:-- --:--:-- --:--:-- 9
```

In this example:

- https://csw.s-enda-staging.k8s.met.no (https://data.csw.met.no) is the URL of the PyCSW server.

- The -X POST option specifies that this is a POST request.

- The -H “Content-Type: application/xml” option sets the content type of the request to XML.

- The -d @my_xml_request_intersecting_a_polygon_within_a_given_time_span_and_certain_te option attaches the contents of the querying XML file to the request.

The server will respond with an XML document containing the search results. You can save this document to a file using the -o option with curl:

```
### The original bash commands:

'''
%%bash
# curl -X POST -H "Content-Type: application/xml" -d \
# @my_xml_request_intersecting_a_polygon_within_a_given_time_span_and_cer
# -o \
# query_results.xml https://data.csw.met.no
'''

'''
curl -X POST -H "Content-Type: application/xml" -d \
@my_xml_request_intersecting_a_polygon_within_a_given_time_span_and_certa
-o query_results.xml https://csw.s-enda-staging.k8s.met.no
'''

### Using subprocess just to make the output readable when opened as a HT

import subprocess

# Define the curl command
curl_command = [
    "curl", "-X", "POST", "-H", "Content-Type: application/xml", "-d",
    "@my_xml_request_intersecting_a_polygon_within_a_given_time_span_and_
    "-o", "query_results.xml",
    "https://csw.s-enda-staging.k8s.met.no"
]

# Run the curl command
result = subprocess.run(curl_command, capture_output=True, text=True)

# Print the output
print("Standard Output:\n")
print(result.stdout)
print("\nStandard Error:\n")
print(result.stderr)
```

```
Standard Output:



Standard Error:

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Cu
                                 Dload  Upload   Total   Spent    Left  Sp

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
100  1882    0     0  100  1882      0   9360 --:--:-- --:--:-- --:--:--
100 25859  100 23977  100  1882  76177   5979 --:--:-- --:--:-- --:--:-- 8
```

NB! In this example, the search results are (as intended) saved to query_results.xml.

Parameter descriptions

Contents

Below is a list of variables available in the SAR datasets with some adherent descriptions.

```
%%capture
!python3 -m venv .venv      # sets up a virtual environment
!source .venv/bin/activate  # Activates said environment

# Installing the required packages to current environment
!pip install -r ../requirements.txt

# The "%%capture" command ensures that the output
# (in this case directories) are not displayed.
```

List of Variables

sensor_view_angle

The angle between a sensor’s line of sight and the vertical downward direction. A zero angle indicates the sensor is looking directly downward.

sigma0

The uncalibrated and normalized radar cross section.

The radar cross section is a measure of the detectability of an object. It’s the ratio of the

power scattered by the object in a specific direction to the power that would be scattered by an ideal reflector.

The scattering/absorption/attenuation coefficient is assumed to cover all wavelengths, unless specified by the the radiation_wavelength coordinate. It pertains to the radiation's deflection without energy loss. Backwards scattering involves scattering into all backward angles, i.e. those exceeding pi/2 radians. This quantity should not specify a scattering_angle.

subswath_number

Per pixel subswath number.

Each image is composed of 5 subswaths. The variable provides the subswath number of each image pixel.

incidence_angle

The angle between the radar beam center and the normal to the local topography is the incidence angle.

sensor_azimuth

The horizontal angle measured clockwise from north to the direction the sensor is pointing.

dc

The Doppler Centroid frequency shift.

dc_std

The standard deviation of Doppler Centroid frequency shift.

topographic_height

The vertical distance or height of a point above sea level.

valid_land_doppler

Valid land pixels (pixels near water boundaries have been masked).

valid_sea_doppler

Valid water covering pixels (near land boundaries have been masked).

valid_doppler

Valid Doppler pixels (0 for invalid, 1 for land, 2 for ocean).

electronic_mispointing

The Doppler frequency shift due to electronic mispointing.

geometric_doppler

The Doppler frequency shift due to orbit geometry.

wind_waves_doppler

The Radar Doppler frequency shift due to wind waves.

This is an experimental variable calculated from ERA5 wind fields and an empirical model (CDOP). Because the wind field is uncertain, the variable is often inaccurate.

std_wind_waves_doppler

The standard deviation of radar Doppler frequency shift due to wind waves.

wind_direction

SAR look relative ERA5 reanalysis wind-from direction used in CDOP calculation.

wind_speed

ERA5 reanalysis wind speed used in CDOP calculation.

geophysical_doppler

The radar Doppler frequency shift due to surface velocity.

ground_range_current

The sea surface current velocity in ground range direction.

By subtracting wind_waves_doppler from geophysical_doppler (geophysical_doppler - wind_waves_doppler) you are left with the Doppler shift caused by the sea surface current. This is then transformed to a measure of the current in the ground range direction, namely “ground_range_current”.

The wind_waves_doppler is an experimental variable calculated from ERA5 wind fields and an empirical model (CDOP). Because the wind field is uncertain, the wind_waves_doppler is often inaccurate. As an implication, the ground_range_current is therefore also often inaccurate.

std_ground_range_current

The standard deviation of the sea surface current velocity in the ground range direction.

longitude

Longitude as degrees east.

latitude

Latitude as degrees north.

zero_doppler_time

This variable provides the time of zero Doppler. It can be used to merge datasets in azimuth.

Example Overview of Variables

```
import xarray as xr

''' The backslashes serves as line shifts '''

ds = xr.open_dataset('''https://thredds.met.no/thredds/dodsC/\
remotesensingenvisat/asar-doppler/2012/01/27/\
ASA_WSDV2PRNMI20120127_215005_000612433111_00101_51839_0000.nc''')
```

Printed Overview of Variables

```
ds.data_vars
```

```
Data variables:
    sensor_view_angle         (y, x) float32 2MB ...
    sigma0                    (y, x) float32 2MB ...
    subswath_number           (y, x) float32 2MB ...
    incidence_angle           (y, x) float32 2MB ...
    sensor_azimuth            (y, x) float32 2MB ...
    dc                        (y, x) float32 2MB ...
    dc_std                    (y, x) float32 2MB ...
    topographic_height        (y, x) float32 2MB ...
    valid_land_doppler        (y, x) float32 2MB ...
    valid_sea_doppler         (y, x) float32 2MB ...
    valid_doppler             (y, x) float32 2MB ...
    electronic_mispointing    (y, x) float32 2MB ...
    geometric_doppler         (y, x) float32 2MB ...
    wind_waves_doppler        (y, x) float32 2MB ...
    std_wind_waves_doppler    (y, x) float32 2MB ...
    wind_direction            (y, x) float32 2MB ...
    wind_speed                (y, x) float32 2MB ...
    geophysical_doppler       (y, x) float32 2MB ...
    ground_range_current      (y, x) float32 2MB ...
    std_ground_range_current  (y, x) float32 2MB ...
    longitude                 (y, x) float32 2MB ...
    latitude                  (y, x) float32 2MB ...
    crs                       int32 4B ...
```

Interactive Overview of Dataset

```
# Interactive if opened as a HTML, jupyter notebook or jupyter-book
ds
```

xarray.Dataset

▸ Dimensions: (y: 602, x: 851, **zero_doppler_time**: 602)

▾ Coordinates:

zero_dopple… (zero_doppler_time) datetime64[ns] 2012-01-27T21:50:0…

▸ Data variables: (23)

▸ Indexes: (1)

▸ Attributes: (60)

Plot Data on a Map

Contents


```
%%capture
!python3 -m venv .venv      # sets up a virtual environment
!source .venv/bin/activate  # Activates said environment

# Installing the required packages to current environment
!pip install -r ../requirements.txt

# The "%%capture" command ensures that the output
# (in this case directories) are not displayed.
```

This notebook is to serve as a manual on how to plot the SAR data. This can be done in numerous ways, but in the following some spesific examples are presented. These functions rely on the use of the python libraries cartopy and xarray, along with some other packages. Thefore, the first thing to do is to ensure that the required packages are installed and imported:

```
# Import the required packages:

import os
import glob
import xarray as xr
import matplotlib.pyplot as plt
import numpy as np
import cartopy.crs as ccrs
import cartopy.feature as cfeature

from matplotlib.colors import TwoSlopeNorm
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
```

Next up is to select the dataset which contain the data that is to be plotted:

```
# Open a dataset using its OPENDAP-url
# (file from 2012-01-27T21:50:05)

''' The backslashes serves as line shifts '''

ds = xr.open_dataset('''https://thredds.met.no/thredds/dodsC/\
remotesensingenvisat/asar-doppler/2012/01/27/\
ASA_WSDV2PRNMI20120127_215005_000612433111_00101_51839_0000.nc''')

ds
```

xarray.Dataset

▸ Dimensions: (y: 602, x: 851, **zero_doppler_time**: 602)

▾ Coordinates:

zero_dopple… (zero_doppler_time) datetime64[ns] 2012-01-27T21:50:0…

▸ Data variables: (23)

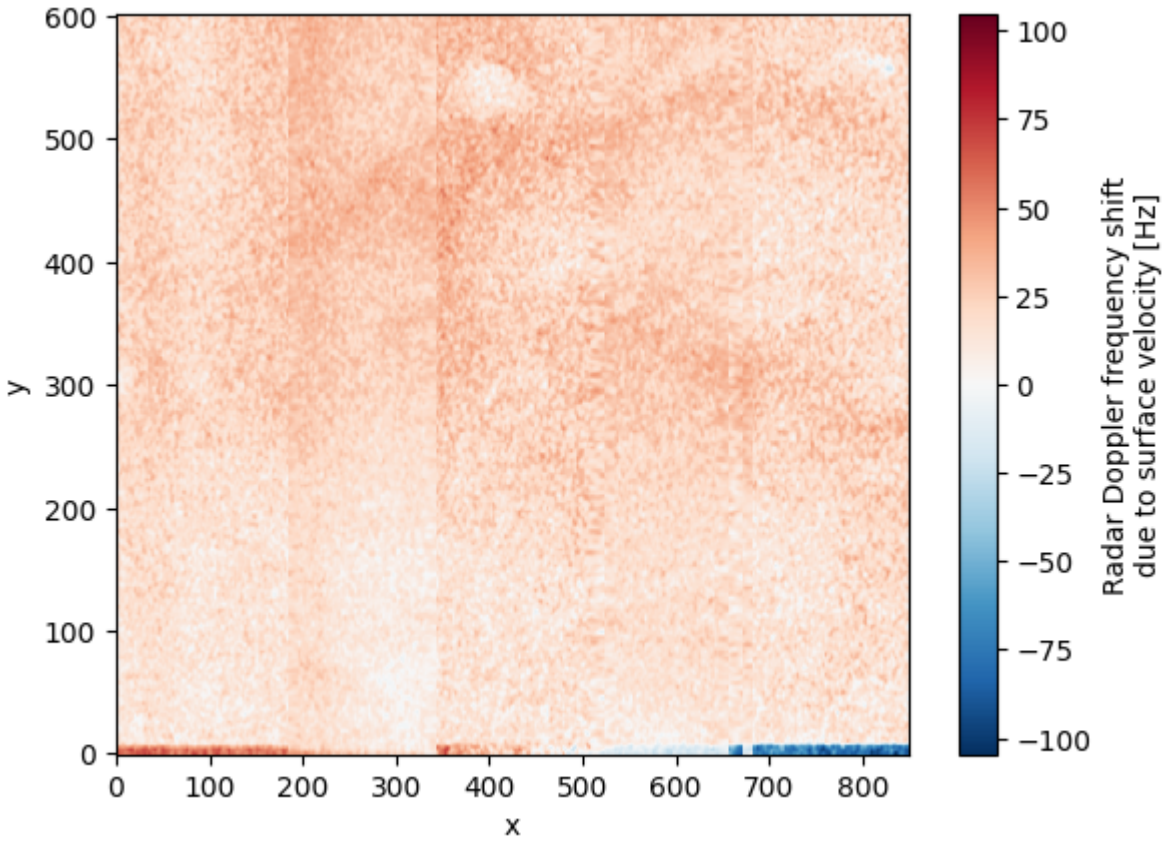
▸ Indexes: (1)

▸ Attributes: (60)

The netCDF file format combined with the xarray python library makes it easy to plot different variables. The example below is arguably the easiest and most efficient way of plotting a variable from an opened xarray.Dataset. The metadata of the dataset in question is used to provide axis- and colorbar labels.

In the following example the variable “geophysical_doppler” from the xarray.Dataset (defined as ds) is plotted by simply using .plot(). Here the axis labels are automatically named “x” and “y” as these are the dimensions of “geophysical_doppler”. From the “Data variables” of the xarray.Dataset (see the interactive menu above), both the “standard_name” and the “units” (in square brackets) of the plotted variable “geophysical_doppler” are also automatically added as the colorbar label.

```
# Plotting a selected variable from the dataset opened above:

ds.geophysical_doppler.plot()
plt.show()
```



Below is a code snippet which easily extracts the wanted files from the current directory (alternativly from “Downloads”) given a recognicable part of the filename. This is meant for extracting files which have been downloaded locally. This can be used when files have been dowloaded from i.e. data.met.no, see the Visualize Data with WMS section on the Sar Dataset Accessibility page.

```
### Opens Downloaded file
# ds_arctic = xr.open_dataset('''ASA_WSDV2PRNMI20120127_215005
# _000612433111_00101_51839_0000
# .nc''')
''' Downloaded file found in the current notebook folder '''

### Stores the path of the current directoy
path2files = os.getcwd()

'''
# Alternative for when a dataset is downloaded locally and stored in
# "Downloads" - which is the default placement
### Get the path of the home directory
home_dir = os.path.expanduser("~")

#### Use os.path.join to combine paths
path2files = os.path.join(home_dir, "Downloads")
'''

### Specify something to recognice the desired nc-files,
  # extract them and put them in a list
desired_files_paths = glob.glob(os.path.join(path2files, "ASA_*.nc"))
''' Where "ASA_*.nc" is
    "filename_starts_with_str*filename_ends_with"
'''

# Open the first dataset
ds = xr.open_dataset(desired_files_paths[0])

# Interactive overview of the dataset
# - when opened as a jupyter notebook or a jupyter book
ds
```

xarray.Dataset

▸ Dimensions: (y: 602, x: 851, **zero_doppler_time**: 602)

▾ Coordinates:

zero_dopple… (zero_doppler_time) datetime64[ns] 2012-01-27T21:50:0…

▸ Data variables:

(23)

▸ Indexes: (1)

▸ Attributes: (60)

Plotting Function

Below is an easy to use plotting function which visualize your desired data variable on a map. This function relies on the python library cartopy, along several other packages that are imported on the very begining of this Plot Data on a Map section. There is also possible to set the spatial extent of the plot yourself - default is plotting the spatial extent of the variable itself as defined in the metadata.

```
def plotting_variables_on_a_map(
ds,
variable,
lat_min = 'default', lat_max = 'default',
lon_min = 'default', lon_max = 'default',
number_of_colorbar_intervals = 100,
fractional_title_spacing = 0.05,
title_fontsize = 15,
cbar_label_fontsize = 12):


    ### Create a new xarray DataArray with latitude and longitude
      # as coordinates
    plotable_variable = variable.assign_coords(
        {'lat': (('y', 'x'), ds.latitude.values),
         'lon': (('y', 'x'), ds.longitude.values)
         })

    ### Extract the geospatial extent of the variable that
      # is plotted
    if lat_min == 'default':
        lat_min = float(plotable_variable.lat.min())
    else:
        lat_min = lat_min

    if lat_max == 'default':
      lat_max = float(plotable_variable.lat.max())
    else:
        lat_max = lat_max

    if lon_min == 'default':
      lon_min = float(plotable_variable.lon.min())
    else:
       lon_min = lon_min

    if lon_max == 'default':
      lon_max = float(plotable_variable.lon.max())
    else:
       lon_max = lon_max

    #### Ready to plot the variable on a map using subplot package
       # of matplotlib.pyplot
       # Create the main plot and set the projection
    fig, ax = plt.subplots(
                1,
                figsize=(16, 12),
                subplot_kw={'projection': ccrs.PlateCarree()})

    ax.axis('off')   # Turn off axes
    ax.coastlines()  # Add coastlines


    ### Create the gridlines and configure the labels
    gl = ax.gridlines(draw_labels=True)
    gl.bottom_labels = True # Default     - Include bottom labels
    gl.left_labels = True   # Default     - Include LH side labels
    gl.top_labels = True    # Default     - Include top labels
```

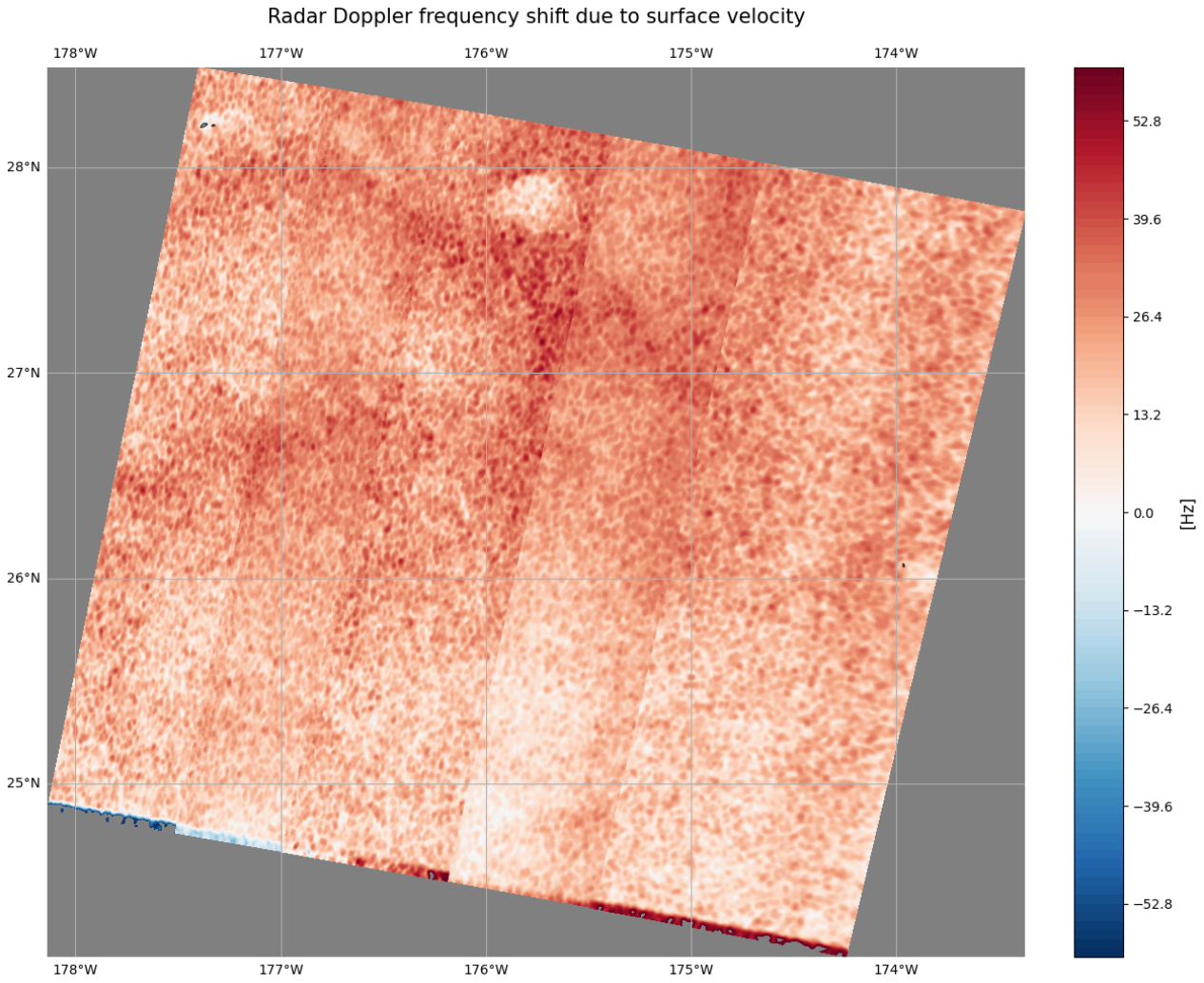
```
    gl.right_labels = False # Not default - Exclude RH side labels

    ### Defining var_min, var_max and var_step
      # using the variable attribute minmax
      # to extract var_min and var_max
    var_min = int(plotable_variable.minmax.split(' ')[0])
    var_max = int(plotable_variable.minmax.split(' ')[1])

    ### Defining the colorbar intervals based on desired
      # number of colorbars
    var_step = ((var_max - var_min)/number_of_colorbar_intervals)

    ### Redefining the max val to improve visuals of colorbar
      #  around center
    var_max=int(plotable_variable.minmax.split(' ')[1])+var_step

    ### Add features for land and ocean
    ax.add_feature(cfeature.LAND, facecolor='tan', zorder=2, alpha = 0.5)
    ''' Tan color for land
        - Placed third in line for plotting '''

    ax.add_feature(cfeature.OCEAN, facecolor='grey', zorder=0)
    ''' Grey color for ocean
        - Placed first in line for plotting '''

    ### Plot the data
    cax = plotable_variable.plot.contourf(
        ax=ax, x='lon', y='lat',
        extend='neither',
        norm=TwoSlopeNorm(vmin=var_min,
                          vcenter=0,
                          vmax=var_max),
        levels=np.arange(var_min, var_max, var_step),
        add_colorbar=False,  # Do not automatically add colorbar
        zorder = 1
        )

    ### Create and inset axes for the colorbar
    cbar_ax = inset_axes(
        ax,
        width="5%",  # width = 5% of parent_bbox width
        height="100%",  # height : 100% of parent_bbox height
        loc='lower left',
        bbox_to_anchor=(1.05, 0., 1, 1),
        bbox_transform=ax.transAxes,
        borderpad=0
        )

    ### Create colorbar
    cbar = fig.colorbar(cax,
                        cax=cbar_ax,
                        orientation='vertical')

    ### Extract units from the plotted variable to use as
      #  label for colorbar
    cbar.set_label(f'[{plotable_variable.units}]',
                   fontsize = cbar_label_fontsize)
```

```
    # Set and adjust title
    mid_lat = (lat_min + lat_max) / 2
    mid_lon = (lon_min + lon_max) / 2
    ax.text(
        mid_lon,
        lat_max+((lat_max - lat_min)*fractional_title_spacing),
          f'{plotable_variable.long_name}',
            fontsize=title_fontsize,
              ha='center')

    # Setting the extent on the map that is plotted
    ax.set_ylim(lat_min, lat_max)
    ax.set_xlim(lon_min, lon_max)

    plt.show()

    return
```

Plotting Examples

In the following, some examples on how to utilize the plotting function above (and their results) is shown.

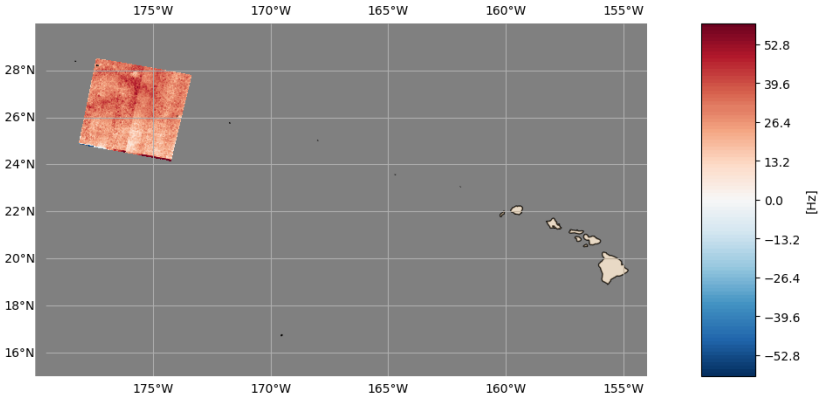
```
### Plotting the desired variable within its geographical extent
  # - By default using the metadata for the geographical extent
  # as lat/lon min/max values.

### Plotting geophysical_doppler
plotting_variables_on_a_map(
    ds = ds,
    variable = ds.geophysical_doppler,
    number_of_colorbar_intervals = 100,  # Default value
    title_fontsize = 15,                 # Default value
    cbar_label_fontsize = 12             # Default value
    )
```



Then, some examples on how to alter the spatial extent of the map at which the selected variable is plotted on (three examples showing Hawaii in the middle of the Pacific Ocean):

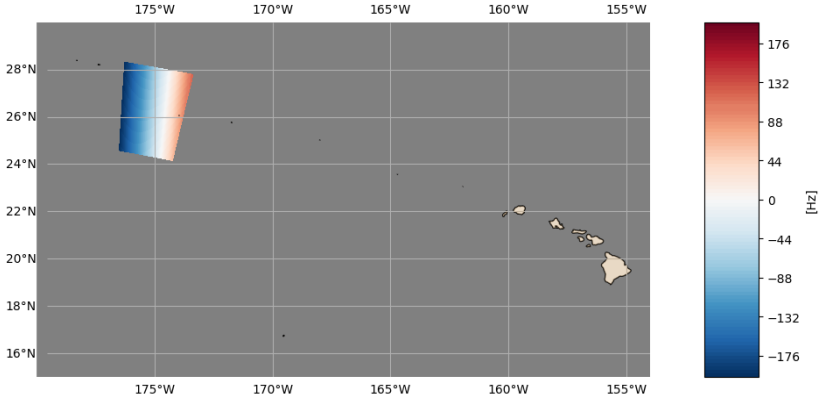
```
### Plotting geophysical_doppler
plotting_variables_on_a_map(
    ds = ds,
    variable = ds.geophysical_doppler,
    lat_min = 15, lat_max = 30,
    lon_min = -200, lon_max = -154,
    number_of_colorbar_intervals = 100,  # Default value
    fractional_title_spacing = 0.10,     # Default 0.05
    title_fontsize = 30,                 # Default value
    cbar_label_fontsize = 10             # Default 12
    )
```

Radar Doppler frequency shift due to surface velocity

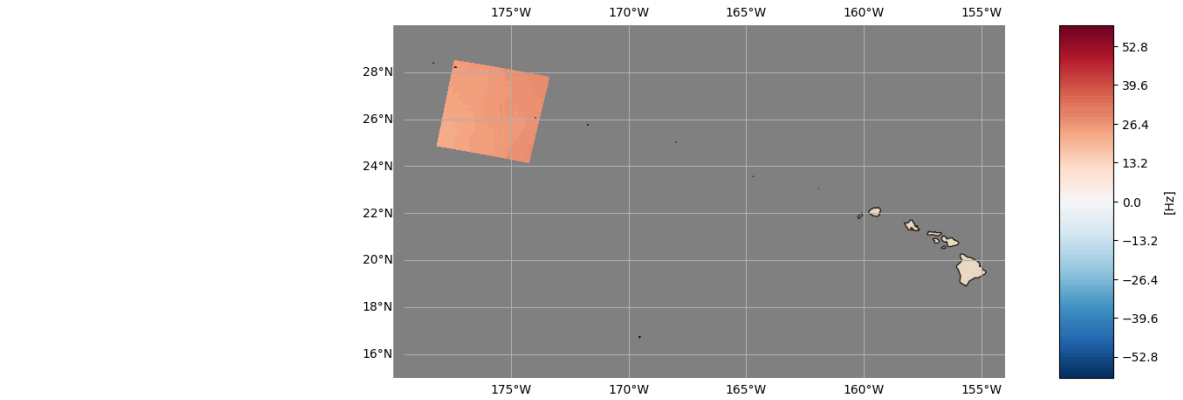



```
### Plotting geometric_doppler
plotting_variables_on_a_map(
    ds = ds,
    variable = ds.geometric_doppler,
    lat_min = 15, lat_max = 30,
    lon_min = -200, lon_max = -154,
    number_of_colorbar_intervals = 100,  # Default value
    fractional_title_spacing = 0.10,     # Default 0.05
    title_fontsize = 30,                 # Default value
    cbar_label_fontsize = 10             # Default 12
    )
```

Doppler frequency shift due to orbit geometry

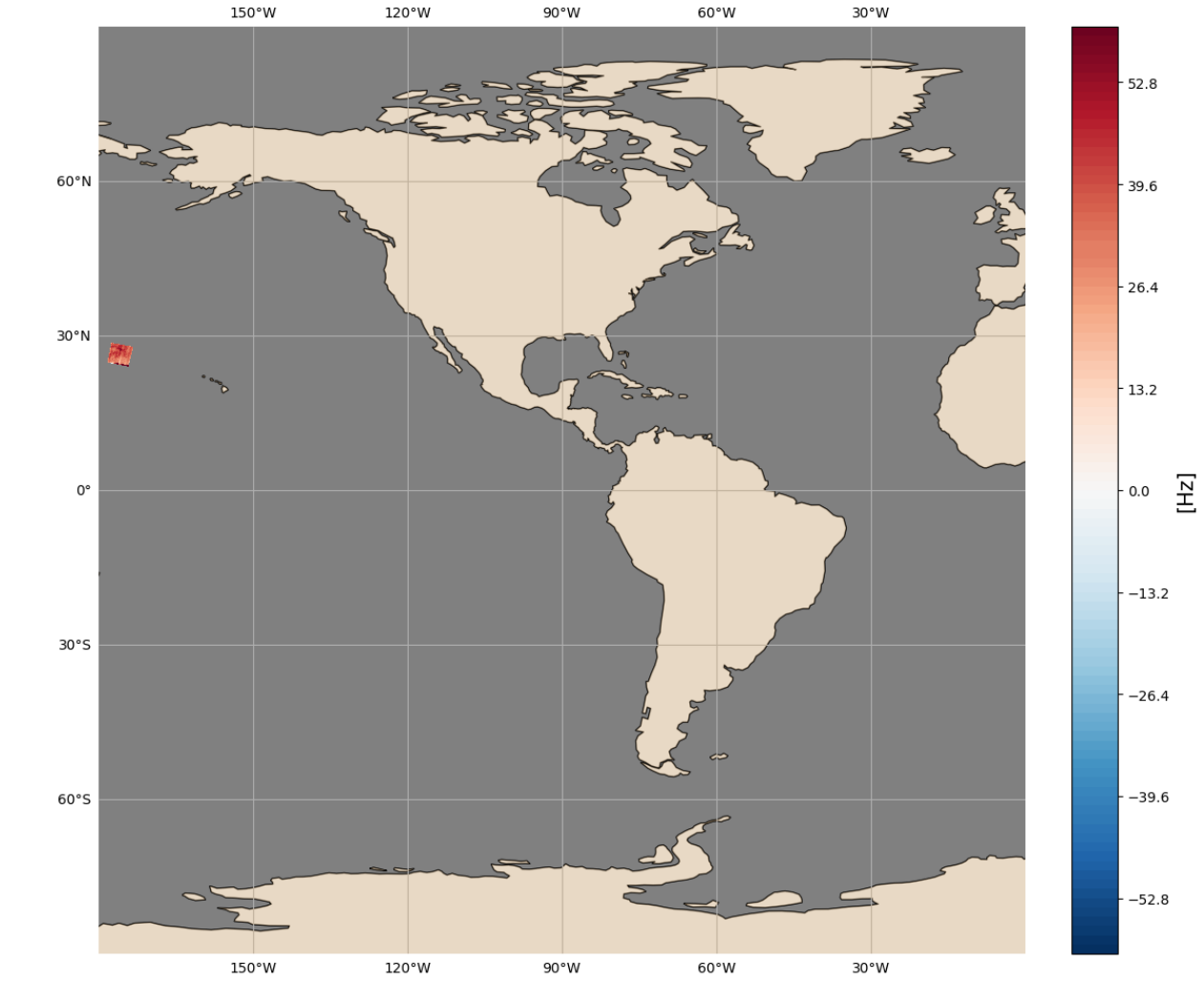



```
### Plotting wind_waves_doppler
plotting_variables_on_a_map(
    ds = ds,
    variable = ds.wind_waves_doppler,
    lat_min = 15, lat_max = 30,
    lon_min = -200, lon_max = -154,
    number_of_colorbar_intervals = 100,  # Default value
    fractional_title_spacing = 0.20,     # Default 0.05
    title_fontsize = 30,                 # Default value
    cbar_label_fontsize = 10             # Default 12
    )
```



And finally, some data variable plotted on (half of) the world map:

```
### Plotting geophysical_doppler
plotting_variables_on_a_map(
    ds = ds,
    variable = ds.geophysical_doppler,
    lat_min = -90, lat_max = 90,
    lon_min = -180, lon_max = 0,
    number_of_colorbar_intervals = 100,  # Default value
    fractional_title_spacing = 0.08,     # Default 0.05
    title_fontsize = 30,                 # Default 15
    cbar_label_fontsize = 15             # Default 12
    )
```



Plotting with a Polar Projection

As the SAR Doppler data is a great asset when researching areas at high latitudes, there is also provided an additional example function which extracts and plots desired variables on a map with an Arctic projection. Again the python library cartopy is utilized.

```
def ArcticPlot_variables_on_a_map(ds,
                                  variable,
                                  map_color,
                                  map_start_longitude = -180,
                                  map_end_longitude = 180,
                                  map_edge_latitude = 60,
                                  map_center_latitude = 90,
                                  title_fontsize = 15,
                                  log_scale_option = False):


    ### Create a new xarray DataArray with
      # latitude and longitude as coordinates
    if log_scale_option == True:

      plotable_variable = (np.log10(variable)*10).assign_coords(
        {'lat': (('y', 'x'), ds.latitude.values),
         'lon': (('y', 'x'), ds.longitude.values)
         })
      '''converting the linear backscatter values to dB (decibel)'''

    else:

        plotable_variable = variable.assign_coords(
        {'lat': (('y', 'x'), ds.latitude.values),
         'lon': (('y', 'x'), ds.longitude.values)
         })

    fig, ax = plt.subplots(1, figsize=(10, 6))
    fig.suptitle(f'{variable.long_name}',
                 fontsize = title_fontsize)
    ax.axis('off')   # Turn off axis

    # Set the projection - NorthPolarStereo
    ax = plt.axes(projection=ccrs.NorthPolarStereo())
    ax.coastlines()   # Adding coastlines
    ax.set_extent([map_start_longitude,
                   map_end_longitude,
                   map_edge_latitude,
                   map_center_latitude],
                   crs=ccrs.PlateCarree())

    # Add features for land and ocean
    ax.add_feature(cfeature.LAND,
                   facecolor='tan',
                   zorder=2,
                   alpha = 0.5)
    ''' Tan color for land - Placed third in line for plotting '''

    ax.add_feature(cfeature.OCEAN,
                   facecolor='grey',
                   zorder=0)
    ''' Grey color for ocean - Placed first in line for plotting'''


    if log_scale_option == True:
        plotable_variable.plot.pcolormesh(
```

```
        ax=ax, x='lon', y='lat',
        transform=ccrs.PlateCarree(),  # added transform argument
        extend='neither',
        cbar_kwargs = {'label':'Decibel [dB]'},
        cmap=map_color,
        zorder = 1    # Placed second in line for plotting
        )

    else:
        plotable_variable.plot.pcolormesh(
        ax=ax, x='lon', y='lat',
        transform=ccrs.PlateCarree(),  # added transform argument
        extend='neither',
        cbar_kwargs = {'label': f'[{variable.units}]'},
        cmap=map_color,
        zorder = 1   # Placed second in line for plotting
        )


    # Draw gridlines on the map
    gl = ax.gridlines(draw_labels=True)

    plt.tight_layout()
    plt.show()

    return
```

By getting hold of a dataset from somewhere close to the Arctic, the plotting of different variables can be done as follows:

```
### Opens Downloaded file
# ds_arctic = xr.open_dataset('''ASA_WSDH2PRNMI20120322_111545
# _000623843113_00023_52623_0000
# .nc''')
''' Downloaded file found in the notebook folder '''
'''     The backslashes serves as line shifts    '''



# Opens dataset through the associated OPENDAP-link
ds_arctic = xr.open_dataset('''https://thredds.met.no/thredds/dodsC/\
remotesensingenvisat/asar-doppler/2012/03/22/\
ASA_WSDH2PRNMI20120322_111545_000623843113_00023_52623_0000.nc''')

ds_arctic
```

xarray.Dataset

▸ Dimensions: (y: 611, x: 847, **zero_doppler_time**: 611)

▾ Coordinates:

zero_dopple… (zero_doppler_time) datetime64[ns] 2012-03-22T11:15:4…

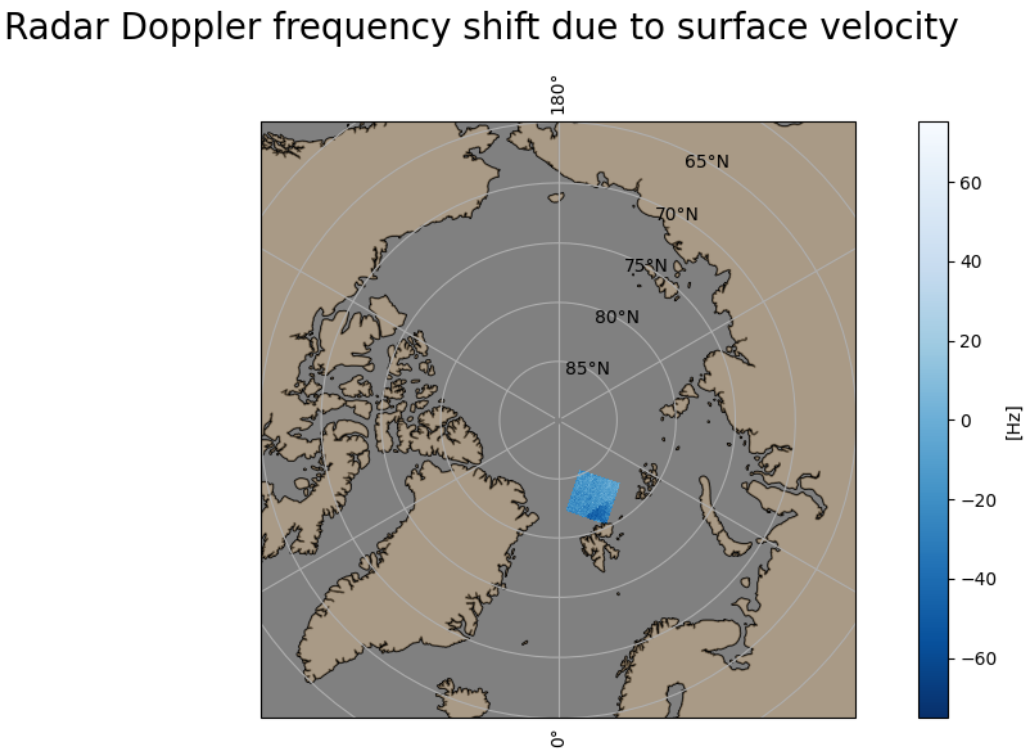
▸ Data variables: (23)

▸ Indexes: (1)

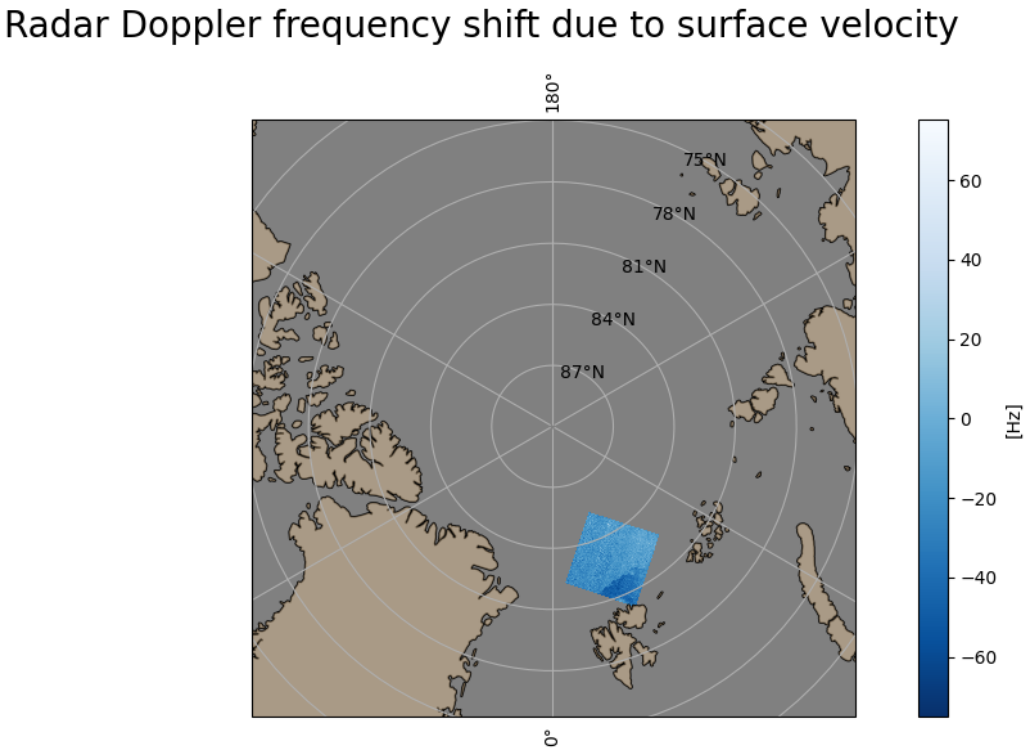
▸ Attributes: (60)

Underneath there are again some examples on how to plot a specified variable on the Arctic map. For each example the map is more and more zoomed in on the data that is plotted.

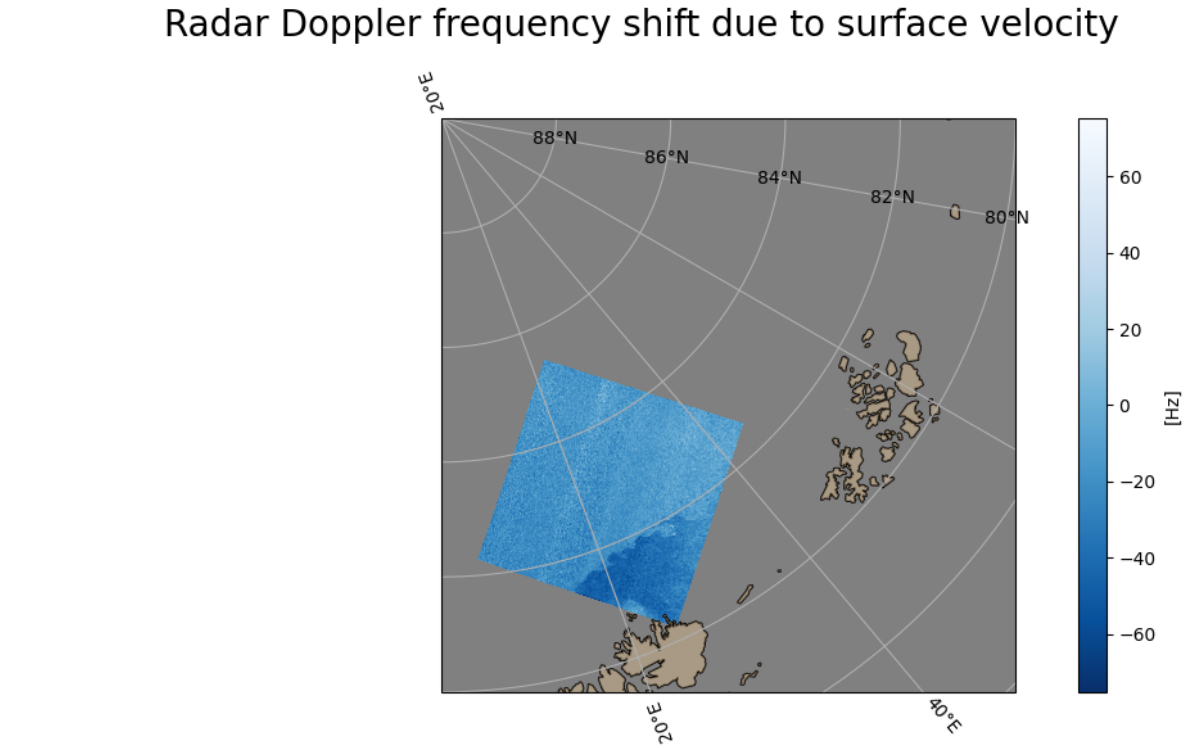
```
# The entire Arctic

ArcticPlot_variables_on_a_map(ds = ds_arctic,
                              variable = ds_arctic.geophysical_doppler,
                              map_color = 'Blues_r',
                              map_start_longitude = -180,
                              map_end_longitude = 180,
                              map_edge_latitude = 65,     # Default 60
                              map_center_latitude = 90,   # Default 90
                              title_fontsize = 20         # Default 15
                              )
```



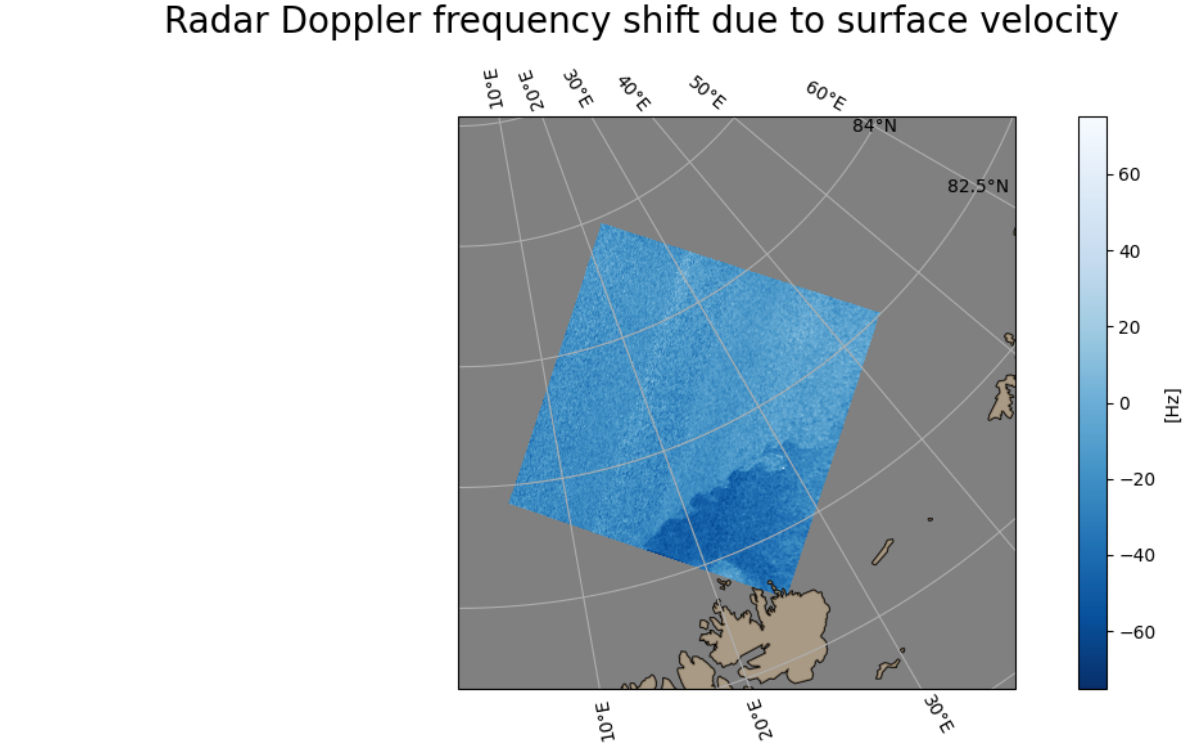
```
# Zooming in on the Arctic Ocean

ArcticPlot_variables_on_a_map(ds = ds_arctic,
                              variable = ds_arctic.geophysical_doppler,
                              map_color = 'Blues_r',
                              map_start_longitude = -180,
                              map_end_longitude = 180,
                              map_edge_latitude = 75,     # Default 60
                              map_center_latitude = 90,  # Default 90
                              title_fontsize = 20         # Default 15
                              )
```




```
# A quarter of the sphere covering the Arctic Ocean

ArcticPlot_variables_on_a_map(ds = ds_arctic,
                              variable = ds_arctic.geophysical_doppler,
                              map_color = 'Blues_r',
                              map_start_longitude = 0,
                              map_end_longitude = 90,
                              map_center_latitude = 90,  # Default 90
                              map_edge_latitude = 80,  # Default 60
                              title_fontsize = 20         # Default 15
                              )
```




```
# Close up of the plotted variable

ArcticPlot_variables_on_a_map(ds = ds_arctic,
                              variable = ds_arctic.geophysical_doppler,
                              map_color = 'Blues_r',
                              map_start_longitude = 0,
                              map_end_longitude = 44,
                              map_center_latitude = 86,   # Default 90
                              map_edge_latitude = 80,     # Default 60
                              title_fontsize = 20         # Default 15
                              )
```



As an additional option, the plot below show that the selected variable, in this instance backscatter (sigma0), is attainable to plot on a log scale [dB] through setting the log_scale_option as True.

```
ArcticPlot_variables_on_a_map(ds = ds_arctic,
                              variable = ds_arctic.sigma0,
                              map_color = 'gray_r',
                              map_start_longitude = 0,
                              map_end_longitude = 44,
                              map_center_latitude = 86,  # Default 90
                              map_edge_latitude = 80,    # Default 60
                              title_fontsize = 20,        # Default 15
                              log_scale_option = True
                              )
```