OPEN DATA CUBE

Loading and analysing Earth Observation data with the Open Data Cube

Website: www.opendatacube.org
Datacube docs: https://datacube-core.readthedocs.io/en/latest/
odc-geo docs: https://odc-geo.readthedocs.io/en/latest/

Getting started

Import Python packages and connect to database:

```
import datacube  # used for querying and loading data
import odc.geo.xr  # enables additional geospatial tools

dc = datacube.Datacube()
```

List available products in the datacube:

```
dc.list_products()
```

List the measurements (e.g. bands or variables) available for each datacube product:

```
dc.list_measurements()
```

Loading data

Load a specific product and measurements:

```
ds = dc.load(
      product="ga_ls8c_ard_3",
      measurements=["nbart_red", "nbart_blue", "fmask"], ...)
```

Load data for a specific spatial extent:

Degrees lat/lon coordinates (WGS84/EPSG:4326):

```
dc.load(...
     y=(-32.2, -32.5),
     x=(142.2, 142.5))
```

Custom coordinate reference system (e.g. Australian Albers):

```
dc.load(...
     x=(948280, 981840),
     y=(-3546480, -3584720),
     crs="EPSG:3577")
```

Loading data by time:

From a specific date:

```
dc.load(...
     time="2020-01-01")
```

From an entire year:

```
dc.load(...
     time="2020")
```

All data from 2020 to 2022 (inclusive of start and end):

```
dc.load(...
     time=("2020", "2022"))
```

All data from 2020 onward (inclusive of start):

```
dc.load(...
     time=("2020", None))
```

Group sequential images captured along each satellite path into daily timesteps:
(only required for products with daily acquisitions, e.g. Landsat or Sentinel-2; not required for summary products like annual or monthly datasets)

```
dc.load(..., group_by="solar_day")
```

Load and reproject data into a custom coordinate reference system and resolution grid, e.g. UTM Zone 55 S, 200 metre resolution:
(for most CRSs, the first value is negative by convention)

```
dc.load(...
      output_crs="EPSG:32755",
      resolution=(-200, 200))  # -y, x
```

Apply custom resampling when reprojecting (default is “nearest”):

Use “average” resampling for all bands:

```
dc.load(...
     resampling="average")
```

Use “nearest” resampling for the “fmask” band, “average” for all others:

```
dc.load(...
     resampling={
        "fmask": "nearest",
        "*": "average"})
```

Lazily load data using Dask:
(used for parallelization and managing memory; chunk sizes will depend on data)

```
dc.load(..., dask_chunks={"y": 2048, "x": 2048})
```

Preparing data for analysis

Inspect nodata attributes and cloud masking band flags:

```
ds.nbart_red.odc.nodata
ds.fmask.attrs["flags_definition"]
```

Setting nodata pixels (e.g. -999) to NaN:

```
ds_masked = datacube.utils.masking.mask_invalid_data(ds)
```

Convert a cloud masking band into a boolean mask and apply to a dataset (setting cloud pixels to NaN):

```
cloud_mask = datacube.utils.masking.make_mask(
    ds.fmask, fmask="cloud")
ds_masked = ds.where(~cloud_mask)
```

Basic analysis with xarray

Selecting a subset of data:

Use “.isel()” for “index selection”, e.g. select first 5 values along the data’s y and x dimensions:

```
ds.isel(
     y=slice(0, 5),
     x=slice(0, 5))
```

Use “.sel()” for “coordinate selection”, e.g. select all pixels between specific y and x coordinates:

```
ds.sel(
     y=slice(-3867375, -3867350),
     x=slice(1516200, 1541300))
```

Aggregating data (e.g. min, max, mean, median, std):

Calculate means for every pixel across time, producing a 2D image:

```
ds.mean(dim="time")
```

Calculate means across all pixels in each timestep, producing a 1D timeseries:

```
ds.mean(dim=["y", "x"])
```

Plotting and exporting data

Plot on an interactive map for rapid data exploration:

```
ds.isel(time=0).odc.explore()  # also works for single bands
```

Plotting single bands as a static plot:

Plot a single timestep:

```
ds.fmask.isel(time=0).plot()
```

Plot multiple timesteps:

```
ds.fmask.plot(
     col="time", col_wrap=4)
```

Plotting multiple bands as an RGB image:
(will auto-guess red, green and blue bands if they exist in the data)

```
ds.isel(time=0).odc.to_rgba().plot.imshow()
```

Export data as a cloud optimised GeoTIFF raster file:

```
ds.isel(time=0).fmask.odc.write_cog("output_filename.tif")
```

GeoBox and geospatial tools

View a dataset’s “GeoBox” defining its spatial pixel grid:

```
ds.odc.geobox
ds.odc.geobox.crs  # coordinate reference system (CRS)
ds.odc.geobox.resolution  # spatial pixel resolution
ds.odc.geobox.boundingbox  # spatial extent of data
```

Reproject a loaded dataset:

Reproject to a different CRS:

```
ds.odc.reproject(
      how="EPSG:32755")
```

Reproject to another dataset’s GeoBox:

```
ds_wgs84  # data in another CRS
ds.odc.reproject(
      how=ds_wgs84.odc.geobox)
```

Mask or crop a dataset to the extent of a polygon:

```
from odc.geo.geom import Geometry
geopolygon = Geometry(<shapely_polygon>, crs="EPSG:4326")

# Mask data to set pixels outside polygon to NaN
ds_masked = ds.odc.mask(poly=geopolygon)

# Crop data to extent of polygon (and optionally mask)
ds_cropped = ds.odc.crop(poly=geopolygon, apply_mask=True)
```

Designed: Robbi Bishop-Taylor (@SatelliteSci), Geoscience Australia. Modified: Feb 2024 (datacube==1.8.17, odc-geo==0.4.2)