Tidy geospatial data cubes

Emma Marshall, University of Utah
Deepak Cherian, NCAR
Scott Henderson, University of Washington

We acknowledge partial funding from NASA 20-OSTFL20-0006

Who are we?

Emma Marshall

Glaciology graduate student
University of Utah

- Using xarray for ~ 3 years
- 2022 NCAR SIParCS intern working on xarray
- My 2nd SciPy!

Scott Henderson

Geophysicist
University of Washington

- Using xarray for ~6 years
- Interested in facilitating research using satellite remote sensing datasets

Deepak Cherian

Oceanographer
National Center for Atmospheric Research

- Xarray maintainer
- First in-person SciPy!

Roadmap

What is the problem?

Geospatial raster data is complex - large amount of duplicated effort among users manipulating datasets into **analysis-ready data cubes**.

What do we hope to do?

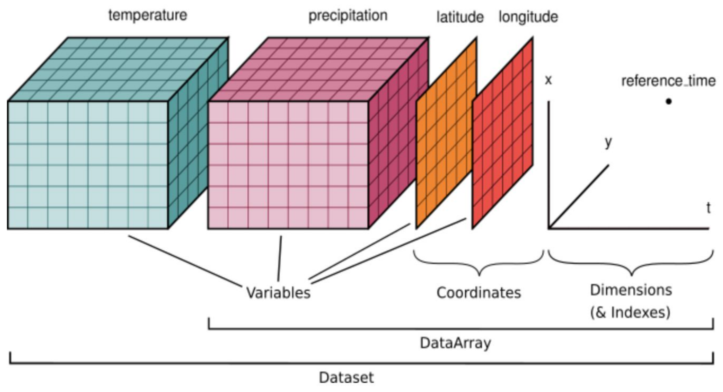
Borrowing from the tidy data framework for tabular data, propose 'tidy' guidelines for N-dimensional geospatial data, represented by **xarray objects**.

How will we do it?

Using examples of **real-world datasets**, we will explore how a tidy framework could make our processing & analysis smoother.

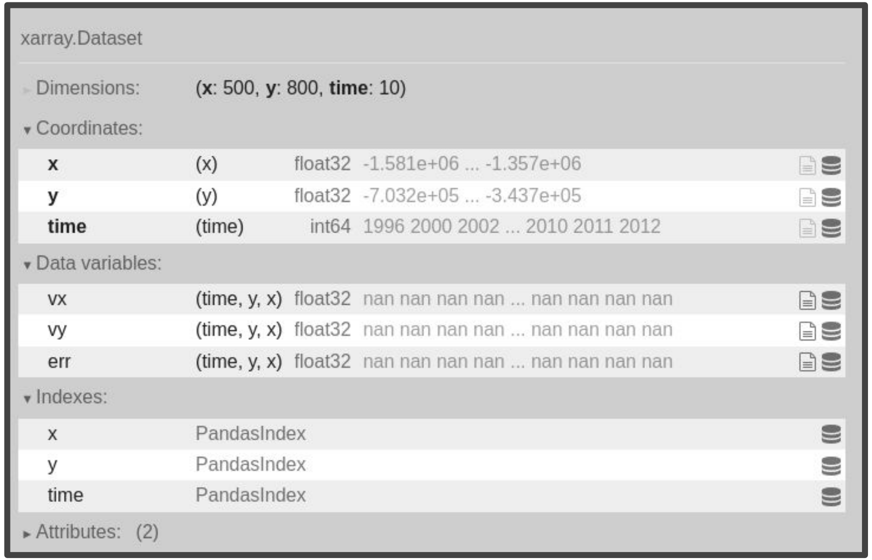
What is xarray?

“Xarray is an open source project and Python package that makes working with labelled multi-dimensional arrays simple, efficient, and fun”



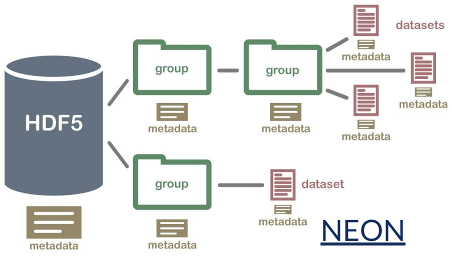
Schematic of xarray data structure

Real-world example of xarray dataset

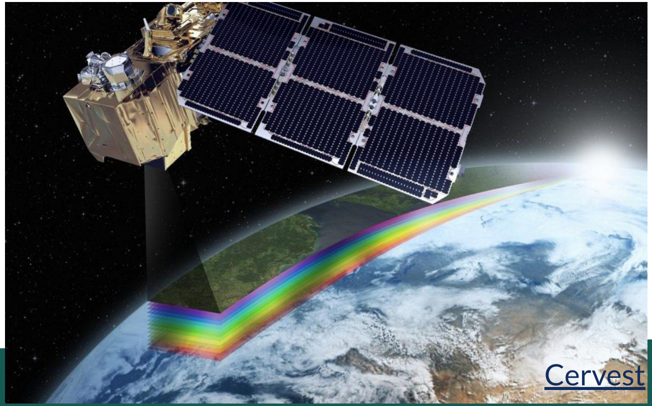


```
xarray.Dataset
▸ Dimensions:    (x: 500, y: 800, time: 10)
▾ Coordinates:
    x            (x)           float32  -1.581e+06 ... -1.357e+06
    y            (y)           float32  -7.032e+05 ... -3.437e+05
    time         (time)          int64  1996 2000 2002 ... 2010 2011 2012
▾ Data variables:
    vx           (time, y, x)  float32  nan nan nan nan ... nan nan nan nan
    vy           (time, y, x)  float32  nan nan nan nan ... nan nan nan nan
    err          (time, y, x)  float32  nan nan nan nan ... nan nan nan nan
▾ Indexes:
    x            PandasIndex
    y            PandasIndex
    time         PandasIndex
▸ Attributes:  (2)
```

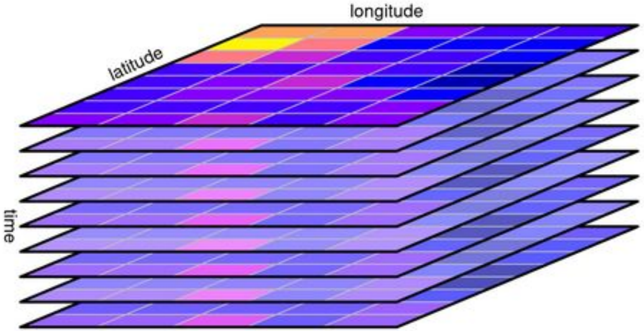
Geospatial datasets are large, complex and can be cumbersome to work with.



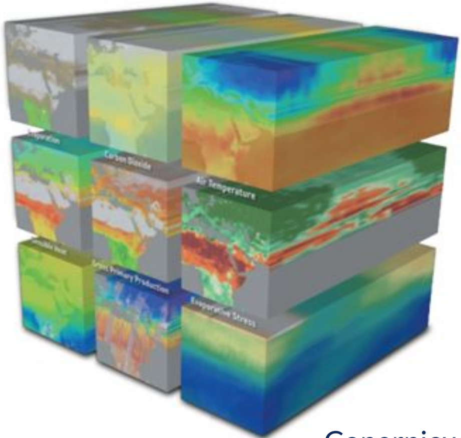
NEON



Cervest



Spatial data science with applications in R



Copernicus

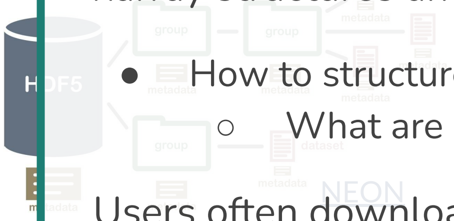
Geospatial datasets are large, complex and can be cumbersome to work with.

Community sentiment that the hardest part of learning to use xarray is conceptualizing xarray structures and how to coerce your data into them

- How to structure your dataset for easy analysis within the xarray ecosystem
  - What are coordinates, dimensions, variables? How do they all inter-relate?

Users often download a subset of data from archives as a number of individual files

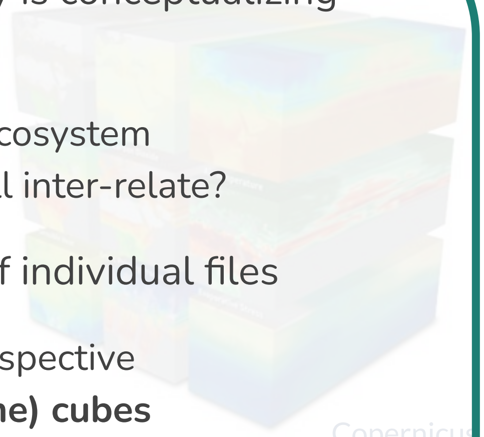
  - Good reason for this from a data-management, efficiency perspective
  - Most users will then need to compile these files into **(x,y,*,time) cubes**
  - Common hangup for new users, creates duplicated effort, introduces many decision points for which we hope to provide guidance.



NEON

Cervest

Spatial data science with applications in R



Copernicus

Tidy data *(H. Wickham, 2014)*

Data tidying: structuring datasets to facilitate analysis (Wickham, 2014)

"…tidy datasets are all alike but every messy dataset is messy in its own way. Tidy datasets provide a standardized way to link the **structure** of a dataset (its physical layout) with its **semantics** (its meaning)."

Tidy data principles for tabular datasets

1. Each variable forms a column
2. Each observation forms a row
3. Each type of observational unit forms a table

What would tidy data for gridded datasets look like?

Tidying real-world datasets

Guiding Q: How could subsequent analysis with this data be made easier?

Tidy data principles

Tabular data principles (from Wickham)

1. Every column is a variable
2. Every row is an observation
3. Every cell is a single value

N-dimensional data principles

Xarray variable = Physical observable(s) needed for analysis

Xarray dimension = Axes defining observable(s) domain

Xarray coordinate = Metadata that varies along dimension

Xarray attribute = Metadata that is static. Metadata attrs should be added such that dataset is self describing (following CF-conventions)

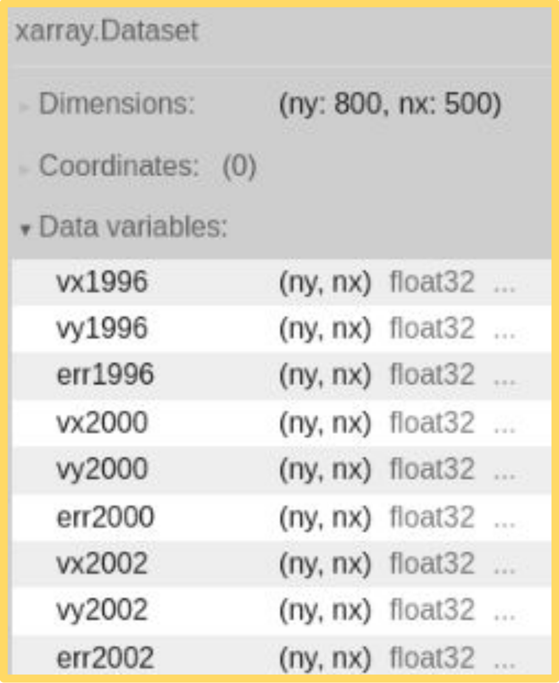
Tidying messy data: Examples

1. **Constructing data cubes**
   a. Organizing to a set of workable objects
2. **Making data cubes analysis-ready**
   a. Additional design elements that impact user experiences (especially when working with large datasets)
   b. Distinguishing observables vs metadata about observations

InSAR Ice Velocity

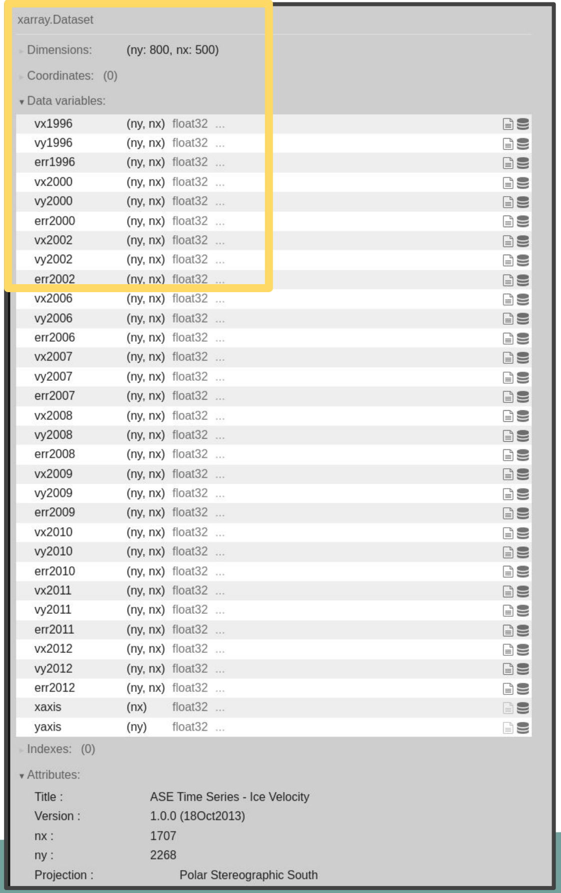
How could subsequent analysis with this data be made easier?

What do we want? A (time,x,y) cube with georeferenced x,y coordinates like lat, lon



```
xarray.Dataset
▸ Dimensions:      (ny: 800, nx: 500)
▸ Coordinates:  (0)
▾ Data variables:
   vx1996           (ny, nx)  float32  ...
   vy1996           (ny, nx)  float32  ...
   err1996          (ny, nx)  float32  ...
   vx2000           (ny, nx)  float32  ...
   vy2000           (ny, nx)  float32  ...
   err2000          (ny, nx)  float32  ...
   vx2002           (ny, nx)  float32  ...
   vy2002           (ny, nx)  float32  ...
   err2002          (ny, nx)  float32  ...
```

InSAR Ice Velocity



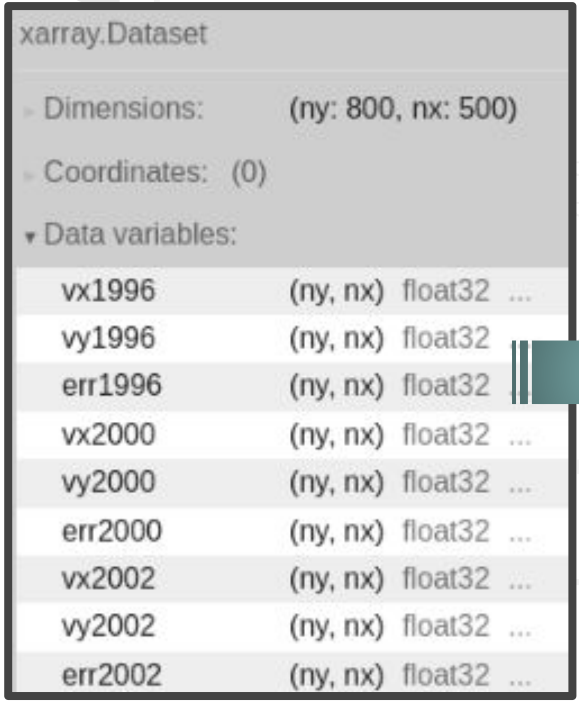
```
xarray.Dataset
▸ Dimensions:      (ny: 800, nx: 500)
▸ Coordinates:  (0)
▾ Data variables:
   vx1996      (ny, nx)  float32  ...
   vy1996      (ny, nx)  float32  ...
   err1996     (ny, nx)  float32  ...
   vx2000      (ny, nx)  float32  ...
   vy2000      (ny, nx)  float32  ...
   err2000     (ny, nx)  float32  ...
   vx2002      (ny, nx)  float32  ...
   vy2002      (ny, nx)  float32  ...
   err2002     (ny, nx)  float32  ...
   vx2006      (ny, nx)  float32  ...
   vy2006      (ny, nx)  float32  ...
   err2006     (ny, nx)  float32  ...
   vx2007      (ny, nx)  float32  ...
   vy2007      (ny, nx)  float32  ...
   err2007     (ny, nx)  float32  ...
   vx2008      (ny, nx)  float32  ...
   vy2008      (ny, nx)  float32  ...
   err2008     (ny, nx)  float32  ...
   vx2009      (ny, nx)  float32  ...
   vy2009      (ny, nx)  float32  ...
   err2009     (ny, nx)  float32  ...
   vx2010      (ny, nx)  float32  ...
   vy2010      (ny, nx)  float32  ...
   err2010     (ny, nx)  float32  ...
   vx2011      (ny, nx)  float32  ...
   vy2011      (ny, nx)  float32  ...
   err2011     (ny, nx)  float32  ...
   vx2012      (ny, nx)  float32  ...
   vy2012      (ny, nx)  float32  ...
   err2012     (ny, nx)  float32  ...
   xaxis       (nx)      float32  ...
   yaxis       (ny)      float32  ...
▸ Indexes:  (0)
▾ Attributes:
   Title :          ASE Time Series - Ice Velocity
   Version :        1.0.0 (18Oct2013)
   nx :             1707
   ny :             2268
   Projection :          Polar Stereographic South
```

https://tutorial.xarray.dev/data_cleaning/ice_velocity.html

InSAR Ice Velocity

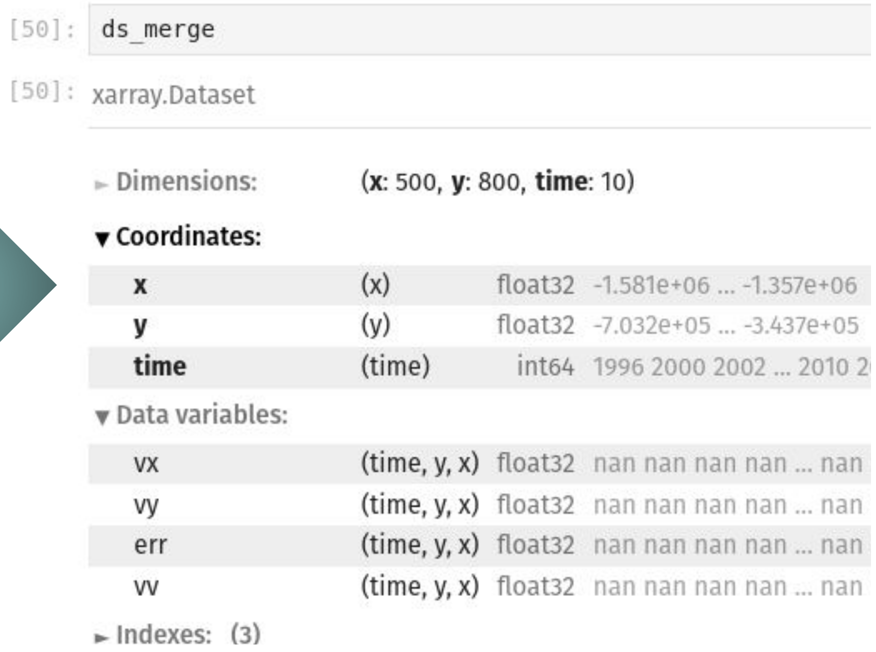
What do we need to do?

- Add time dimension
- Add coordinate variables
- Remove time dim from data variables



```
xarray.Dataset

► Dimensions:      (ny: 800, nx: 500)

► Coordinates:   (0)

▼ Data variables:

   vx1996          (ny, nx)  float32  ...
   vy1996          (ny, nx)  float32  ...
   err1996         (ny, nx)  float32  ...
   vx2000          (ny, nx)  float32  ...
   vy2000          (ny, nx)  float32  ...
   err2000         (ny, nx)  float32  ...
   vx2002          (ny, nx)  float32  ...
   vy2002          (ny, nx)  float32  ...
   err2002         (ny, nx)  float32  ...
```

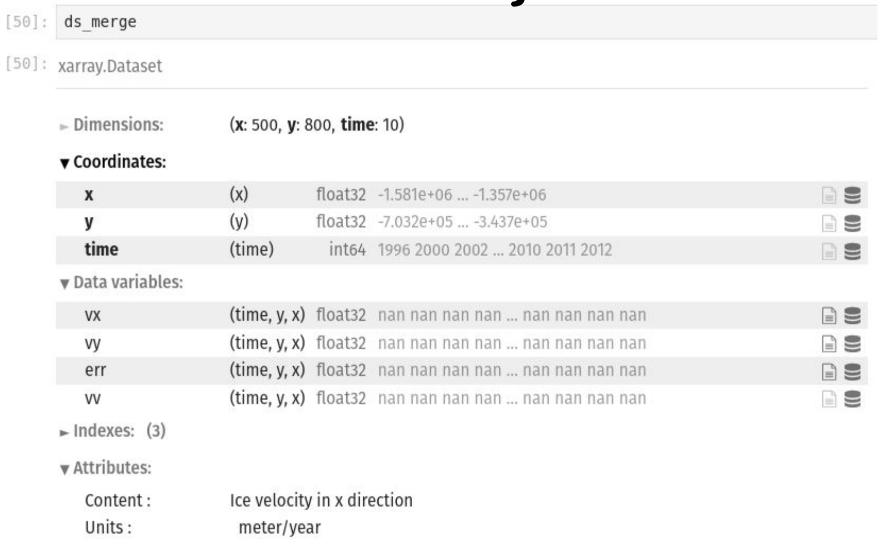
Tidied object



```
[50]: ds_merge

[50]: xarray.Dataset

  ► Dimensions:        (x: 500, y: 800, time: 10)

  ▼ Coordinates:

     x                 (x)           float32  -1.581e+06 ... -1.357e+06
     y                 (y)           float32  -7.032e+05 ... -3.437e+05
     time              (time)          int64  1996 2000 2002 ... 2010 2

  ▼ Data variables:

     vx                (time, y, x)  float32  nan nan nan nan ... nan
     vy                (time, y, x)  float32  nan nan nan nan ... nan
     err               (time, y, x)  float32  nan nan nan nan ... nan
     vv                (time, y, x)  float32  nan nan nan nan ... nan

  ► Indexes:  (3)
```

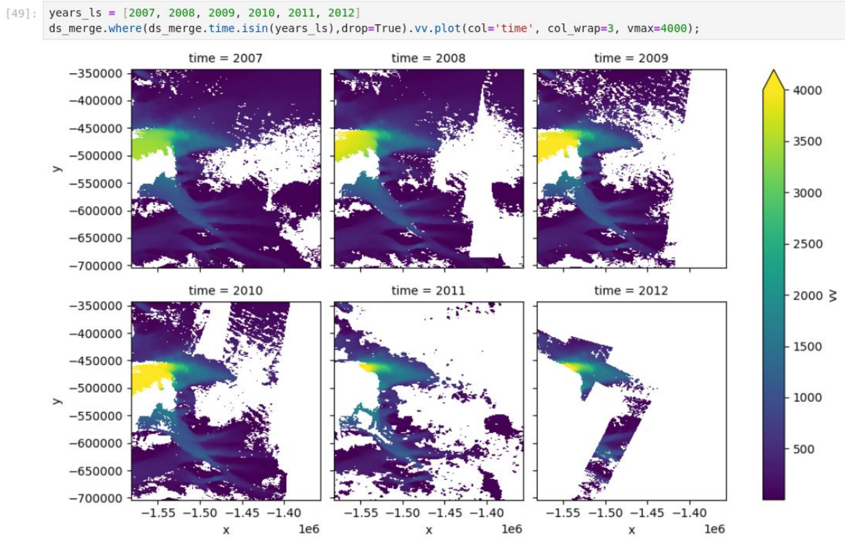
InSAR Ice Velocity

Tidied object



```
[50]: ds_merge
```

```
[50]: xarray.Dataset

► Dimensions:      (x: 500, y: 800, time: 10)
▼ Coordinates:
    x              (x)             float32  -1.581e+06 ... -1.357e+06
    y              (y)             float32  -7.032e+05 ... -3.437e+05
    time           (time)          int64    1996 2000 2002 ... 2010 2011 2012
▼ Data variables:
    vx             (time, y, x)    float32  nan nan nan nan ... nan nan nan nan
    vy             (time, y, x)    float32  nan nan nan nan ... nan nan nan nan
    err            (time, y, x)    float32  nan nan nan nan ... nan nan nan nan
    vv             (time, y, x)    float32  nan nan nan nan ... nan nan nan nan
► Indexes:  (3)
▼ Attributes:
    Content :      Ice velocity in x direction
    Units :          meter/year
```



```
[49]: years_ls = [2007, 2008, 2009, 2010, 2011, 2012]
      ds_merge.where(ds_merge.time.isin(years_ls),drop=True).vv.plot(col='time', col_wrap=3, vmax=4000);
```

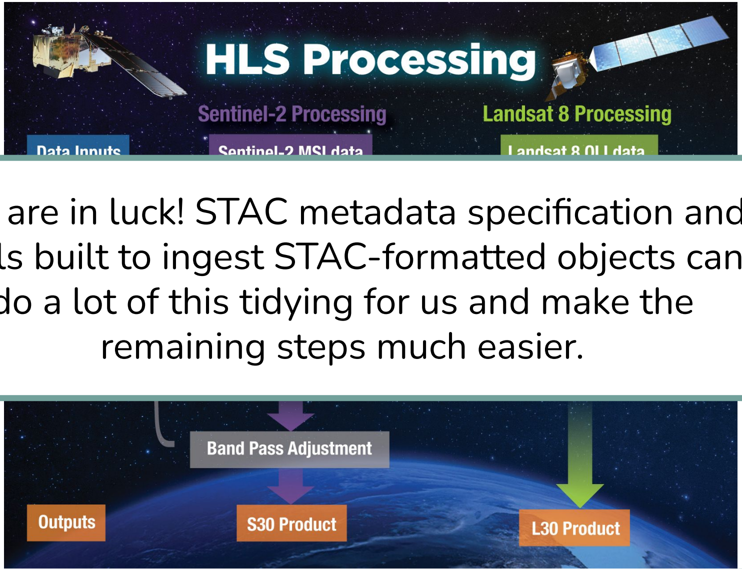
Example of efficient data visualization from a tidied object

Tidying messy data

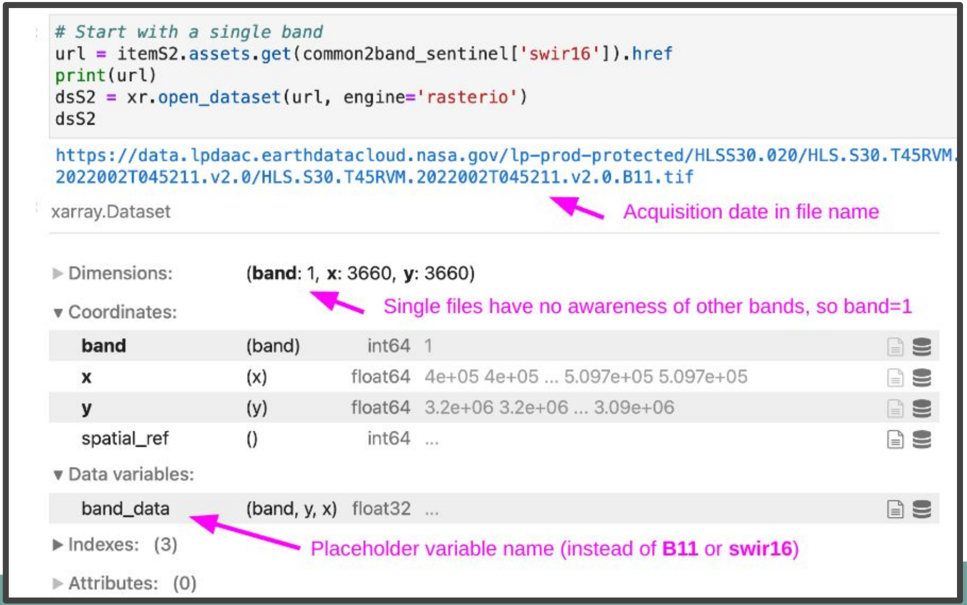
1. **Constructing data cubes**
   a. Organizing to a set of workable objects
2. **Making data cubes <u>analysis-ready</u>**
   a. Additional design elements that impact user experiences (especially when working with large datasets)
   b. Distinguishing observables vs metadata

Harmonized Landsat-Sentinel (HLS)

- Single satellite image read directly to xarray is not tidy:



We are in luck! STAC metadata specification and tools built to ingest STAC-formatted objects can do a lot of this tidying for us and make the remaining steps much easier.



```
# Start with a single band
url = itemS2.assets.get(common2band_sentinel['swir16']).href
print(url)
dsS2 = xr.open_dataset(url, engine='rasterio')
dsS2
```

```
https://data.lpdaac.earthdatacloud.nasa.gov/lp-prod-protected/HLSS30.020/HLS.S30.T45RVM.
2022002T045211.v2.0/HLS.S30.T45RVM.2022002T045211.v2.0.B11.tif

xarray.Dataset

▶ Dimensions:        (band: 1, x: 3660, y: 3660)
▼ Coordinates:
    band             (band)        int64   1
    x                (x)         float64   4e+05 4e+05 ... 5.097e+05 5.097e+05
    y                (y)         float64   3.2e+06 3.2e+06 ... 3.09e+06
    spatial_ref      ()            int64   ...
▼ Data variables:
    band_data        (band, y, x)  float32 ...
▶ Indexes:  (3)
▶ Attributes:  (0)
```

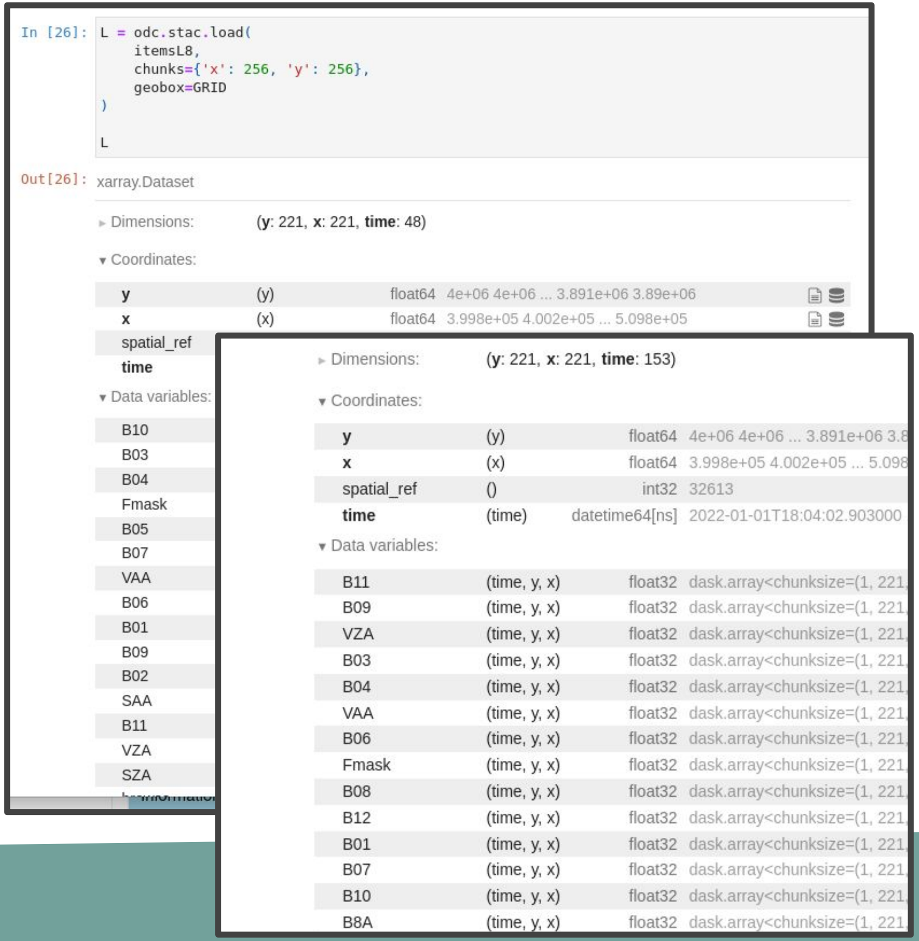
Acquisition date in file name

Single files have no awareness of other bands, so band=1

Placeholder variable name (instead of **B11** or **swir16**)

Harmonized Landsat-Sentinel (HLS)

- Reading in using `odc-stac` organizes the object into a datacube for us !
- Remaining issues:
    1. Add more contextual metadata to coordinates (e..g. platform, sun angle...)
    2. Data quality mask is bit-packed, hard to extract

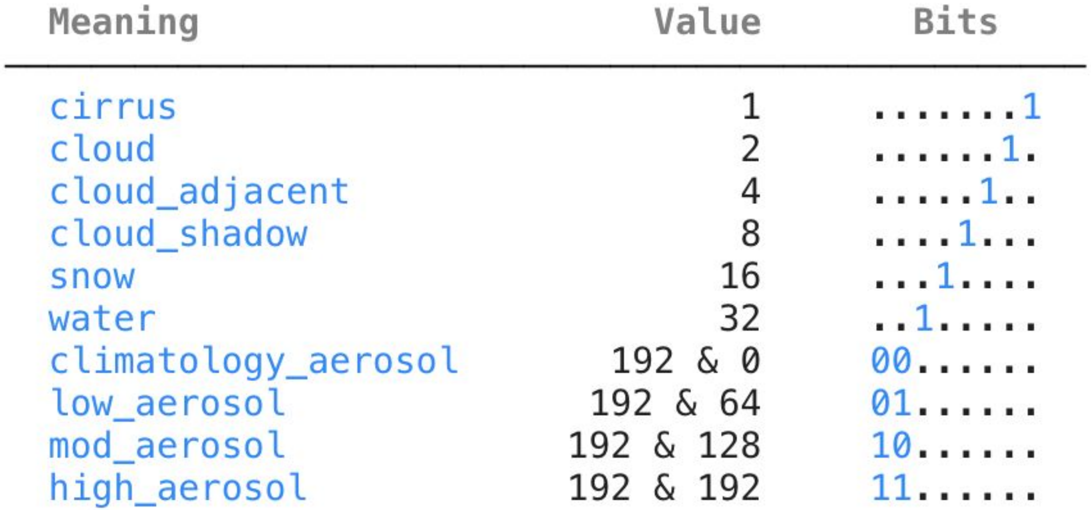


```
In [26]: L = odc.stac.load(
             itemsL8,
             chunks={'x': 256, 'y': 256},
             geobox=GRID
         )

         L
```

```
Out[26]: xarray.Dataset
         ▸ Dimensions:     (y: 221, x: 221, time: 48)
         ▾ Coordinates:
             y             (y)      float64  4e+06 4e+06 ... 3.891e+06 3.89e+06
             x             (x)      float64  3.998e+05 4.002e+05 ... 5.098e+05
             spatial_ref
             time
         ▾ Data variables:
             B10
             B03
             B04
             Fmask
             B05
             B07
             VAA
             B06
             B01
             B09
             B02
             SAA
             B11
             VZA
             SZA
```

```
▸ Dimensions:     (y: 221, x: 221, time: 153)
▾ Coordinates:
    y             (y)          float64  4e+06 4e+06 ... 3.891e+06 3.8
    x             (x)          float64  3.998e+05 4.002e+05 ... 5.098
    spatial_ref   ()             int32  32613
    time          (time)  datetime64[ns]  2022-01-01T18:04:02.903000
▾ Data variables:
    B11           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    B09           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    VZA           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    B03           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    B04           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    VAA           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    B06           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    Fmask         (time, y, x)  float32  dask.array<chunksize=(1, 221,
    B08           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    B12           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    B01           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    B07           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    B10           (time, y, x)  float32  dask.array<chunksize=(1, 221,
    B8A           (time, y, x)  float32  dask.array<chunksize=(1, 221,
```

HLS: Quality flags presented as a bit-packed mask! Not analysis-ready…



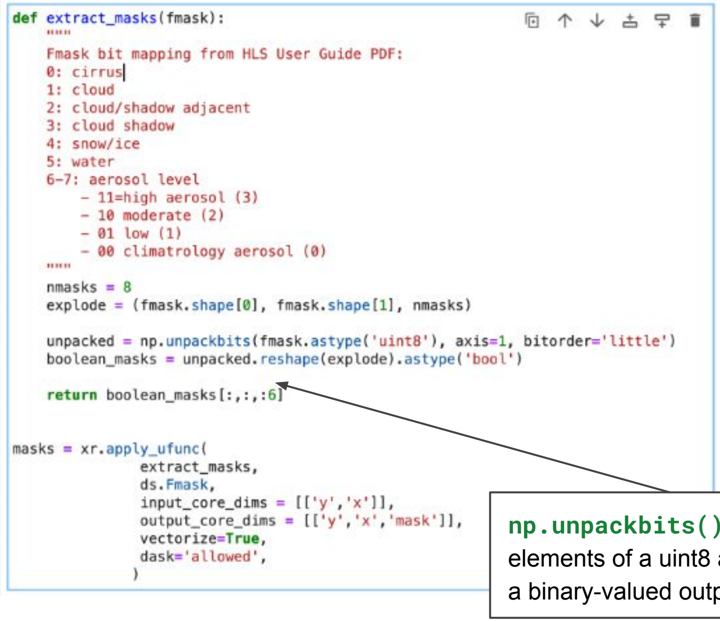
Meaning	Value	Bits
cirrus	1	1
cloud	2	......1.
cloud_adjacent	4	.....1..
cloud_shadow	8	....1...
snow	16	...1....
water	32	..1.....
climatology_aerosol	192 & 0	00......
low_aerosol	192 & 64	01......
mod_aerosol	192 & 128	10......
high_aerosol	192 & 192	11......

Harmonized Landsat-Sentinel (HLS)

- Extracting bit-packed masks is not user-friendly, requires advanced Xarray code

- Without a standardized format, this places a heavy burden on analysis-effort that will be duplicated across users and not necessarily reusable between datasets

Bit-packed masks for cloud, snow, water cover: **Extracting masks with custom function**



```
def extract_masks(fmask):
    """
    Fmask bit mapping from HLS User Guide PDF:
    0: cirrus
    1: cloud
    2: cloud/shadow adjacent
    3: cloud shadow
    4: snow/ice
    5: water
    6-7: aerosol level
        - 11=high aerosol (3)
        - 10 moderate (2)
        - 01 low (1)
        - 00 climatrology aerosol (0)
    """
    nmasks = 8
    explode = (fmask.shape[0], fmask.shape[1], nmasks)

    unpacked = np.unpackbits(fmask.astype('uint8'), axis=1, bitorder='little')
    boolean_masks = unpacked.reshape(explode).astype('bool')

    return boolean_masks[:,:,:6]


masks = xr.apply_ufunc(
               extract_masks,
               ds.Fmask,
               input_core_dims = [['y','x']],
               output_core_dims = [['y','x','mask']],
               vectorize=True,
               dask='allowed',
              )
```

`np.unpackbits()` unpacks elements of a uint8 array into a binary-valued output array.

Bit-packed masks are space efficient, but require:

1. Looking up bit mapping in documentation

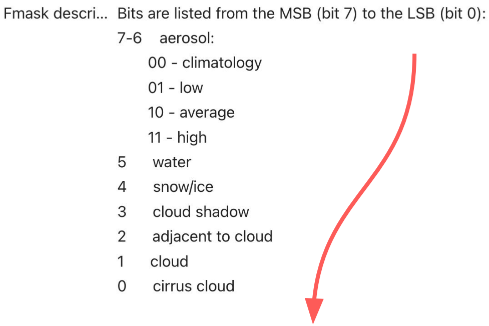
2. Fairly advanced code to readily use masks in workflows

Harmonized Landsat-Sentinel (HLS)

Bit-packed masks for cloud, snow, water cover:

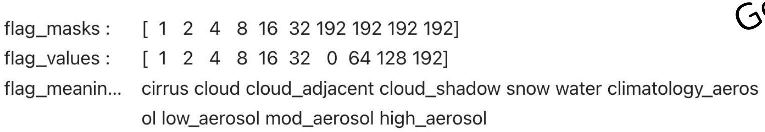
Extracting masks with CF-convention attributes

Essential metadata in non-standard attribute



```
Fmask descri...  Bits are listed from the MSB (bit 7) to the LSB (bit 0):
                 7-6   aerosol:
                       00 - climatology
                       01 - low
                       10 - average
                       11 - high
                 5     water
                 4     snow/ice
                 3     cloud shadow
                 2     adjacent to cloud
                 1     cloud
                 0     cirrus cloud
```

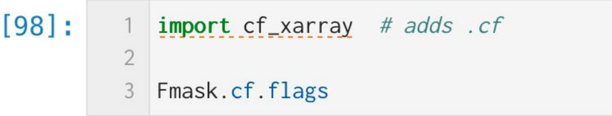
Instead use CF-conventions for “Flag masks and values”



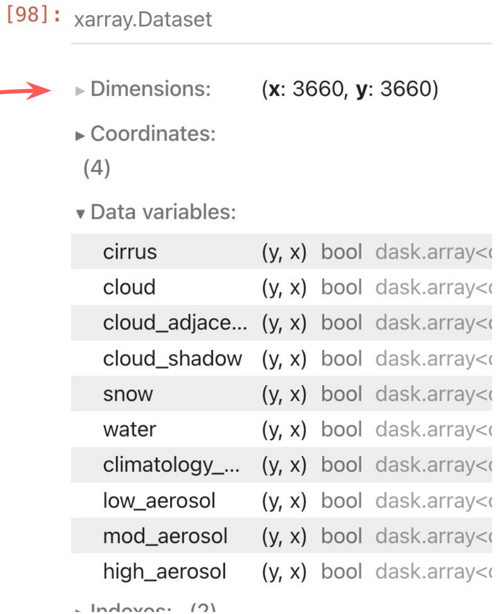
```
flag_masks :    [  1   2   4   8  16   32 192 192 192 192]
flag_values :   [  1   2   4   8  16   32   0  64 128 192]
flag_meanin...  cirrus cloud cloud_adjacent cloud_shadow snow water climatology_aeros
                ol low_aerosol mod_aerosol high_aerosol
```

Now interpretable by cf-xarray.

Get boolean mask array per flag

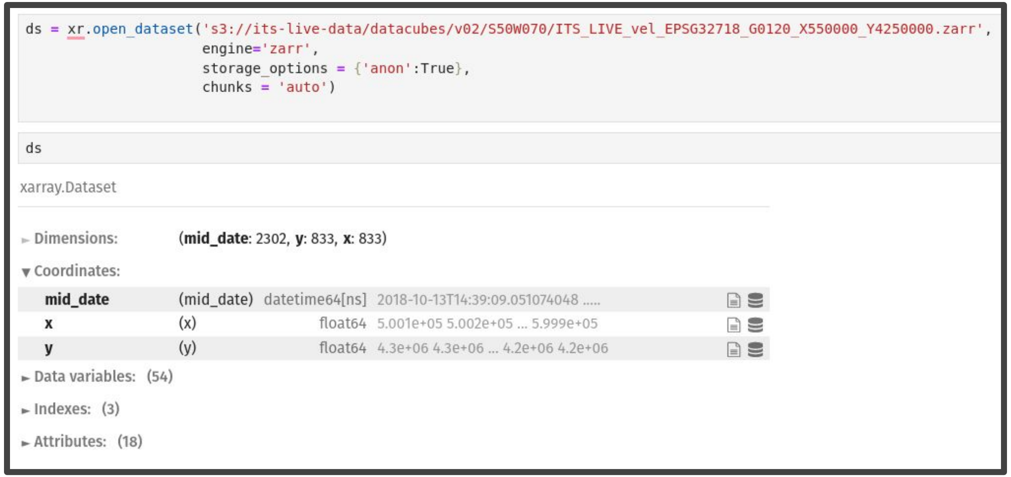


```
[98]:  import cf_xarray  # adds .cf

       Fmask.cf.flags
```



```
[98]:  xarray.Dataset

  ▸ Dimensions:       (x: 3660, y: 3660)
  ▸ Coordinates:
    (4)
  ▾ Data variables:
      cirrus          (y, x)  bool  dask.array<
      cloud           (y, x)  bool  dask.array<
      cloud_adjace... (y, x)  bool  dask.array<
      cloud_shadow    (y, x)  bool  dask.array<
      snow            (y, x)  bool  dask.array<
      water           (y, x)  bool  dask.array<
      climatology_... (y, x)  bool  dask.array<
      low_aerosol     (y, x)  bool  dask.array<
      mod_aerosol     (y, x)  bool  dask.array<
      high_aerosol    (y, x)  bool  dask.array<
  ▸ Indexes:  (2)
```

ITSLIVE

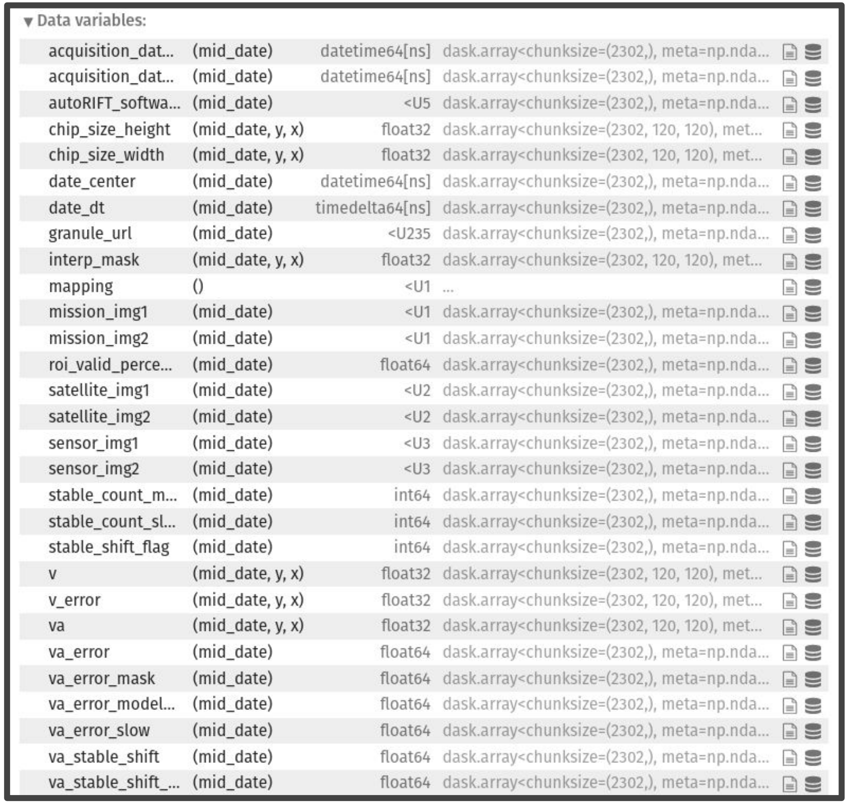


```
ds = xr.open_dataset('s3://its-live-data/datacubes/v02/S50W070/ITS_LIVE_vel_EPSG32718_G0120_X550000_Y4250000.zarr',
                     engine='zarr',
                     storage_options = {'anon':True},
                     chunks = 'auto')
```

```
ds
```

```
xarray.Dataset

► Dimensions:      (mid_date: 2302, y: 833, x: 833)
▼ Coordinates:
   mid_date        (mid_date)  datetime64[ns]  2018-10-13T14:39:09.051074048 .....
   x               (x)                float64  5.001e+05 5.002e+05 ... 5.999e+05
   y               (y)                float64  4.3e+06 4.3e+06 ... 4.2e+06 4.2e+06
► Data variables:  (54)
► Indexes:  (3)
► Attributes:  (18)
```

CF-conventions: ancillary data;   CF-conventions:coordinate types



```
▼ Data variables:
   acquisition_dat...   (mid_date)        datetime64[ns]  dask.array<chunksize=(2302,), meta=np.nda...
   acquisition_dat...   (mid_date)        datetime64[ns]  dask.array<chunksize=(2302,), meta=np.nda...
   autoRIFT_softwa...   (mid_date)                   <U5  dask.array<chunksize=(2302,), meta=np.nda...
   chip_size_height     (mid_date, y, x)         float32  dask.array<chunksize=(2302, 120, 120), met...
   chip_size_width      (mid_date, y, x)         float32  dask.array<chunksize=(2302, 120, 120), met...
   date_center          (mid_date)        datetime64[ns]  dask.array<chunksize=(2302,), meta=np.nda...
   date_dt              (mid_date)       timedelta64[ns]  dask.array<chunksize=(2302,), meta=np.nda...
   granule_url          (mid_date)                 <U235  dask.array<chunksize=(2302,), meta=np.nda...
   interp_mask          (mid_date, y, x)         float32  dask.array<chunksize=(2302, 120, 120), met...
   mapping              ()                           <U1  ...
   mission_img1         (mid_date)                   <U1  dask.array<chunksize=(2302,), meta=np.nda...
   mission_img2         (mid_date)                   <U1  dask.array<chunksize=(2302,), meta=np.nda...
   roi_valid_perce...   (mid_date)               float64  dask.array<chunksize=(2302,), meta=np.nda...
   satellite_img1       (mid_date)                   <U2  dask.array<chunksize=(2302,), meta=np.nda...
   satellite_img2       (mid_date)                   <U2  dask.array<chunksize=(2302,), meta=np.nda...
   sensor_img1          (mid_date)                   <U3  dask.array<chunksize=(2302,), meta=np.nda...
   sensor_img2          (mid_date)                   <U3  dask.array<chunksize=(2302,), meta=np.nda...
   stable_count_m...    (mid_date)                 int64  dask.array<chunksize=(2302,), meta=np.nda...
   stable_count_sl...   (mid_date)                 int64  dask.array<chunksize=(2302,), meta=np.nda...
   stable_shift_flag    (mid_date)                 int64  dask.array<chunksize=(2302,), meta=np.nda...
   v                    (mid_date, y, x)         float32  dask.array<chunksize=(2302, 120, 120), met...
   v_error              (mid_date, y, x)         float32  dask.array<chunksize=(2302, 120, 120), met...
   va                   (mid_date, y, x)         float32  dask.array<chunksize=(2302, 120, 120), met...
   va_error             (mid_date)               float64  dask.array<chunksize=(2302,), meta=np.nda...
   va_error_mask        (mid_date)               float64  dask.array<chunksize=(2302,), meta=np.nda...
   va_error_model...    (mid_date)               float64  dask.array<chunksize=(2302,), meta=np.nda...
   va_error_slow        (mid_date)               float64  dask.array<chunksize=(2302,), meta=np.nda...
   va_stable_shift      (mid_date)               float64  dask.array<chunksize=(2302,), meta=np.nda...
   va_stable_shift_...  (mid_date)               float64  dask.array<chunksize=(2302,), meta=np.nda...
```

- Tons of data variables: which should be coordinates and which should stay data variables?
- ITS_LIVE is an image pair dataset, indexed off of mid-date. But time-separation of image pair is fundamental to the observable – is there a better way to convey this?

Key takeaways

- Important data stored in filename = bad :(
- Non-descriptive variable names can create confusion
- Missing coordinate information makes datasets harder to use
- ‘Shape’/structure of a dataset can sometimes be embedded in variable names
  - This will make subsequent analysis more difficult
- Some variables contain data about your observable, some provide context about that observable
  - Separating these types of data into coordinate variables and data variables will make analysis easier
- Structuring data so that it adheres to common specifications (STAC, CF) let’s us work with already-existing tools, simplifies tidying work

Principles of tidy gridded data

1. **Dimensions**

- Minimize # of dimensional coords; only what is necessary to describe shape of your data

2. **Coordinates**

- Non-dimensional coordinates can be numerous. Each should exist along one or multiple dimensions

3. **Data Variables**

- These should be observables rather than contextual, each should exist along one or multiple dimensions

4. **Contextual information (metadata)**

- Metadata should only be stored as an attribute if it is static along the dimensions to which it is applied
- If metadata is dynamic, store as coordinate variable

5. **Variable, attribute naming**

- **Where possible, use cf-conventions for naming**
- Variable names should be descriptive
- Variable names should not contain information that belongs in a dimension or coordinate. (ie. information stored in variable name should be reduced only to observable)

6. **Make use of, work within the frameworks of other tools**

- Tools like STAC, open data cube, cf.xarray, pystac, stackstac [and many more] maker tidying possible (+ smoother), especially with large datasets

What could a tidy framework look like?

Dataset produces & consumers

- Dataset developers and users often have different needs/priorities
- Developers want: minimize storage
- Users want: easy access to information
  - *These goals are often at odds with one another*
- How can we create a framework/format that minimizes gap between these groups
  - We'd love feedback on this and what it could look like

Tools ecosystem

- Some tools already address this gap. Let's use them!
  - odc-stac, stackstac, cf-xarray, pint, etc.

Where do we go from here?

- Perspectives from users of other libraries
- Tidying examples, resources as educational resources?
- Domain-specific tidy specifications?
- Tidying examples, resources
- Tidy tools

Thank you!

Questions, comments, ideas?
emma.marshall@utah.edu
https://github.com/dcherian/tidy-xarray

Jupyter book

Principles of tidy gridded data

1. **Dimensions**

- Minimize # of dimensional coords; only what is necessary to describe shape of your data

2. **Coordinates**

- Non-dimensional coordinates can be numerous. Each should exist along one or multiple dimensions

3. **Data Variables**

- These should be observables rather than contextual, each should exist along one or multiple dimensions

4. **Contextual information (metadata)**

- Metadata should only be stored as an attribute if it is static along the dimensions to which it is applied
- If metadata is dynamic, store as coordinate variable

5. **Variable, attribute naming**

- **Where possible, use cf-conventions for naming**
- Variable names should be descriptive
- Variable names should not contain information that belongs in a dimension or coordinate. (ie. information stored in variable name should be reduced only to observable)

6. **Make use of, work within the frameworks of other tools**

- Tools like STAC, open data cube, cf.xarray, pystac, stackstac [and many more] maker tidying possible (+ smoother), especially with large datasets