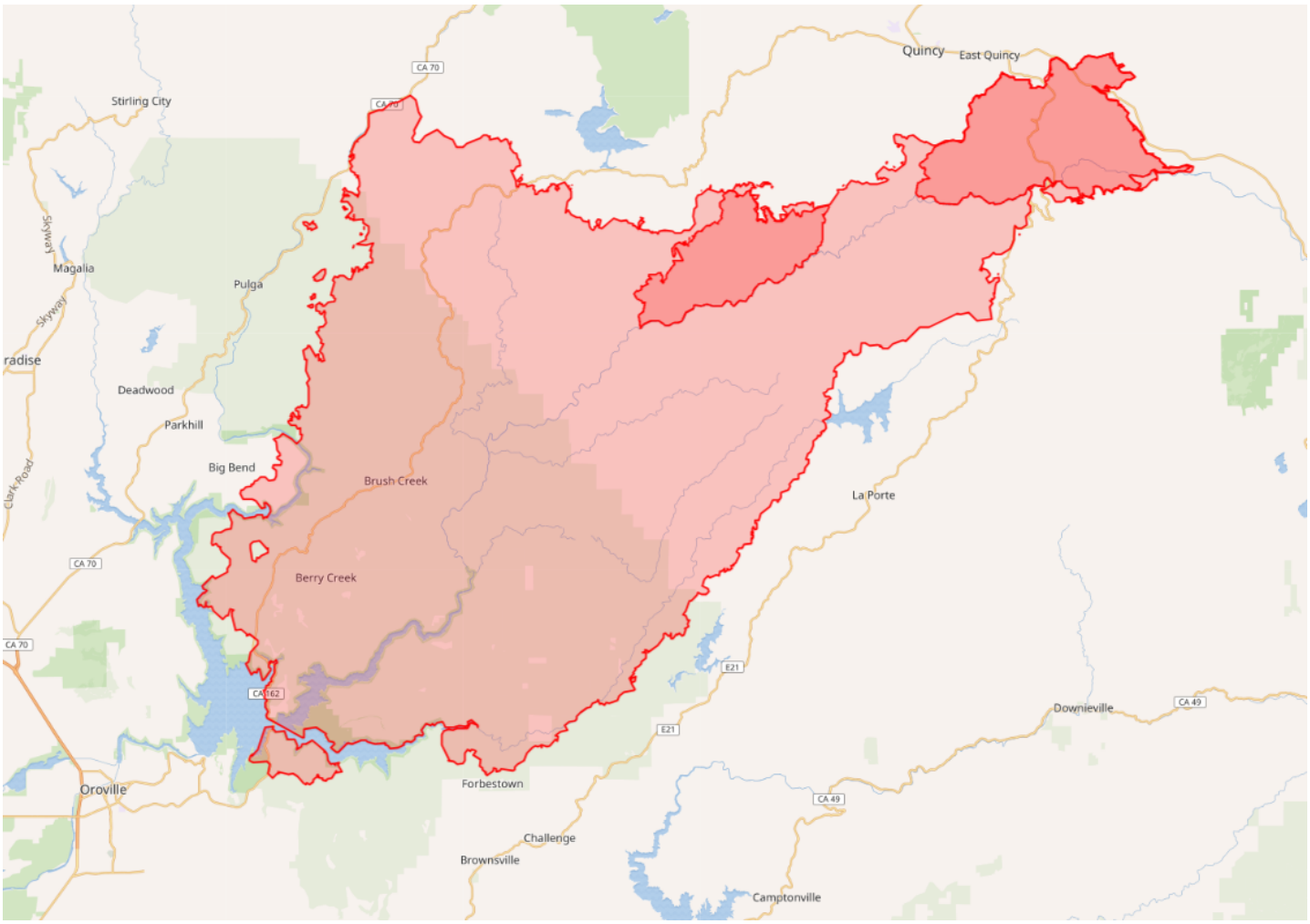
Geospatial Data File Formats examples

This Jupyter Notebook provide examples on how to read different data file formats.

Updated: 01/22/2024

Carlos Lizarraga


Quincy
East Quincy
Stirling City
CA 70
Magalia
Pulga
Deadwood
Parkhill
Big Bend
Brush Creek
La Porte
Berry Creek
Oroville
Forbestown
Challenge
Brownsville
Downieville
Camptonville
E21
CA 49

North Complex Fire. Aug 17, 2020 - Dec 3, 2020. Area burned: 318,935 acres (129,068 ha)

Sentinel-2 on the Planetary Computer

This notebook explores Sentinel-2 data on Microsoft's Planetary Computer using:

- Planetary Computer STAC API, catalog of public data
- Planetary Computer Hub for running Jupyter Notebooks in the cloud
- pystac-client for searching and access data
- OpenDataCube and odc-stac for loading STAC assets and representing geospatial data as XArrays
- XArray, pandas and geopandas for manipulating data
- Dask for performing parallel, distributed computing
- hvplot for visualization

Shown will be how find data for an area of interest, explore the resulting metadata, perform calculations, and visualize the results.

Created by Element 84

Installation of supplementary Python libraries

Next, for our future we install a list of required Python libraries in Google Colab using the *pip* command. See the Pypi.org library repository.

The installation is sequential, because of software dependencies the order is important.

```
# Visualization library: https://pypi.org/project/holoviews/
!pip install "holoviews[recommended]" --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 6.7 MB/s eta 0:00:00
```

```
# High-level plotting API: https://pypi.org/project/hvplot/
!pip install hvplot --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 12.2 MB/s eta 0:00:00
```

```
# Library for working with SpatioTemporal Asset Catalog (STAC): https://pypi.org/project/pystac/
!pip install pystac --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 181.6/181.6 kB 1.7 MB/s eta 0:00:00
```

```
# Python client for working with STAC: https://pypi.org/project/pystac-client/
!pip install pystac-client --quiet
```

```
# Cartographic visualizations with Matplotlib: https://pypi.org/project/Cartopy/
!pip install cartopy --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.8/11.8 MB 29.8 MB/s eta 0:00:00
```

```
# Library for exploring and visualize geographical data: https://pypi.org/project/geoviews/
!pip install geoviews --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 511.2/511.2 kB 3.3 MB/s eta 0:00:00
```

```
# Tool for STAC metadata: https://pypi.org/project/odc-stac/
!pip install odc-stac --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.7/56.7 kB 857.7 kB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 122.4/122.4 kB 2.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.6/20.6 MB 40.5 MB/s eta 0:00:00
```

```
# lightning fast ASGI server: https://pypi.org/project/uvicorn/
!pip install uvicorn --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.6/60.6 kB 800.5 kB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 2.3 MB/s eta 0:00:00
```

```
# A streamimg multipart parser: https://pypi.org/project/python-multipart/
!pip install python-multipart --quiet
```

```
# Fast API framework: https://pypi.org/project/fastapi/
!pip install fastapi --quiet
```

```
# Static image export for web visualization: https://pypi.org/project/kaleido/
!pip install kaleido --quiet
```

```
# Miscellaneous algorithmic helper methods: https://pypi.org/project/odc-algo/
!pip install odc-algo --quiet
```

```
     ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the followir
     bigframes 0.19.1 requires pandas<2.1.4,>=1.5.0, but you have pandas 2.2.0 which is incompatible.
     google-colab 1.0.0 requires pandas==1.5.3, but you have pandas 2.2.0 which is incompatible.
```

```
# Python Imaging Library: https://pypi.org/project/pillow/10.0.1/
!pip install pillow==10.0.1 --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.6/3.6 MB 10.9 MB/s eta 0:00:00
```

```
# Render xarray timestacks: https://pypi.org/project/geogif/
!pip install geogif --quiet
```

```
# Planetary Computer SDK: https://pypi.org/project/planetary-computer/
!pip install planetary-computer --quiet
```

```
# Install Python interface for PROJ: https://pypi.org/project/pyproj/
!pip install pyproj --quiet
```

```
# Install Shapely for manipulating geometric objects; https://pypi.org/project/shapely/
!pip install shapely --quiet
```

```
# Install Geopandas, geographic pandas extensions: https://pypi.org/project/geopandas/
!pip install geopandas --quiet
```

```
# Installl mapclassify
#!pip install mapclassify --quiet
```

```
# initial imports and reusable functions

import holoviews as hv
hv.extension('bokeh')

from copy import deepcopy
import geopandas as gpd
import hvplot.pandas
import pandas as pd
import pystac
from shapely.geometry import shape

# create a function for later reuse
def plot_polygons(data, *args, **kwargs):
    return data.hvplot.paths(*args, geo=True, tiles='OSM', xaxis=None, yaxis=None,
                             frame_width=600, frame_height=600,
                             line_width=3, **kwargs)

# convert a list of STAC Items into a GeoDataFrame
def items_to_geodataframe(items):
    _items = []
    for i in items:
        _i = deepcopy(i)
        _i['geometry'] = shape(_i['geometry'])
        _items.append(_i)
    gdf = gpd.GeoDataFrame(pd.json_normalize(_items))
    for field in ['properties.datetime', 'properties.created', 'properties.updated']:
        if field in gdf:
            gdf[field] = pd.to_datetime(gdf[field])
    gdf.set_index('properties.datetime', inplace=True)
    return gdf

# set pystac_client logger to DEBUG to see API calls
import logging
logging.basicConfig()
logger = logging.getLogger('pystac_client')
logger.setLevel(logging.INFO)

# Dismiss future warnings
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
# Open the Planetary Computer STAC API

from pystac_client import Client
URL = 'https://planetarycomputer.microsoft.com/api/stac/v1'
cat = Client.open(URL)
cat
```


type "Catalog"
id "microsoft-pc"
stac_version "1.0.0"
description "Searchable spatiotemporal metadata describing Earth science datasets hosted by the Microsoft Planetary Computer"
▶ links [] 130 items
▶ conformsTo [] 15 items
title "Microsoft Planetary Computer STAC API"


```
# Link: 68 for Sentinel 2 data
collection = cat.get_collection('sentinel-2-l2a')

pd.DataFrame.from_dict(collection.to_dict()['item_assets'], orient='index')
```

	gsd	type	roles	title	eo:bands
AOT	10.0	image/tiff; application=geotiff; profile=cloud...	[data]	Aerosol optical thickness (AOT)	NaN
B01	60.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 1 - Coastal aerosol - 60m	[{'name': 'B01', 'common_name': 'coastal', 'de...
B02	10.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 2 - Blue - 10m	[{'name': 'B02', 'common_name': 'blue', 'descr...
B03	10.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 3 - Green - 10m	[{'name': 'B03', 'common_name': 'green', 'desc...
B04	10.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 4 - Red - 10m	[{'name': 'B04', 'common_name': 'red', 'descri...
B05	20.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 5 - Vegetation red edge 1 - 20m	[{'name': 'B05', 'common_name': 'rededge', 'de...
B06	20.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 6 - Vegetation red edge 2 - 20m	[{'name': 'B06', 'common_name': 'rededge', 'de...
B07	20.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 7 - Vegetation red edge 3 - 20m	[{'name': 'B07', 'common_name': 'rededge', 'de...
B08	10.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 8 - NIR - 10m	[{'name': 'B08', 'common_name': 'nir', 'descri...
B09	60.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 9 - Water vapor - 60m	[{'name': 'B09', 'description': 'Band 9 - Wate...
B11	20.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 11 - SWIR (1.6) - 20m	[{'name': 'B11', 'common_name': 'swir16', 'des...
B12	20.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 12 - SWIR (2.2) - 20m	[{'name': 'B12', 'common_name': 'swir22', 'des...
B8A	20.0	image/tiff; application=geotiff; profile=cloud...	[data]	Band 8A - Vegetation red edge 4 - 20m	[{'name': 'B8A', 'common_name': 'rededge', 'de...
SCL	20.0	image/tiff; application=geotiff; profile=cloud...	[data]	Scene classfication map (SCL)	NaN
WVP	10.0	image/tiff; application=geotiff; profile=cloud...	[data]	Water vapour (WVP)	NaN
visual	10.0	image/tiff; application=geotiff; profile=cloud...	[data]	True color image	[{'name': 'B04', 'common_name': 'red', 'descri...
preview	NaN	image/tiff; application=geotiff; profile=cloud...	[thumbnail]	Thumbnail	NaN
safe-manifest	NaN	application/xml	[metadata]	SAFE manifest	NaN
granule-metadata	NaN	application/xml	[metadata]	Granule metadata	NaN
inspire-metadata	NaN	application/xml	[metadata]	INSPIRE metadata	NaN
product-metadata	NaN	application/xml	[metadata]	Product metadata	NaN
datastrip-metadata	NaN	application/xml	[metadata]	Datastrip metadata	NaN

```
#Change the Area of Interest here

import geopandas as gpd
import json

aoi = gpd.read_file('/content/bear-fire.geojson')
geom = aoi['geometry'][0] # < shapely geometry object

# limit sets the # of items per page so we can see multiple pages getting fetched
search = cat.search(
    collections = ["sentinel-2-l2a"],
    intersects = geom,
    datetime = "2019-10-01/2019-10-31",
    query = ["eo:cloud_cover<25"],
    limit = 100
)
```

Use GeoPandas to view footprints

The cell below fetches all the STAC Items, then creates a GeoDataFrame for visualizing the footprints.

```
# Get all items as a dictionary
items_dict = search.get_all_items_as_dict()['features']

# Create GeoDataFrame from Items
items_gdf = items_to_geodataframe(items_dict)

print(f"{len(items_dict)} items found")

items_gdf.head()
```

```
12 items found
```

properties.datetime	id	bbox	type	links	geomet
2019-10-27 18:55:01.024000+00:00	S2A_MSIL2A_20191027T185501_R113_T10TFK_2020100...	[-121.834335, 39.63587147039632, -120.51955, 4...	Feature	[{'rel': 'collection', 'type': 'application/js...	POLYGC ((-121.817 40.6448 -120.519 40.6
2019-10-27 18:55:01.024000+00:00	S2A_MSIL2A_20191027T185501_R113_T10SFJ_2020100...	[-121.84912, 38.73593643111066, -120.55211, 39...	Feature	[{'rel': 'collection', 'type': 'application/js...	POLYGC ((-121.832 39.7444 -120.552 39.7
2019-10-22 18:54:29.024000+00:00	S2B_MSIL2A_20191022T185429_R113_T10TFK_2020100...	[-121.834335, 39.63587147039632, -120.51955, 4...	Feature	[{'rel': 'collection', 'type': 'application/js...	POLYGC ((-121.817 40.6448 -120.519 40.6
2019-10-22 18:54:29.024000+00:00	S2B_MSIL2A_20191022T185429_R113_T10SFJ_2020100...	[-121.84912, 38.73593643111066, -120.55211, 39...	Feature	[{'rel': 'collection', 'type': 'application/js...	POLYGC ((-121.832 39.7444 -120.552 39.7
2019-10-17 18:54:01.024000+00:00	S2A_MSIL2A_20191017T185401_R113_T10TFK_2020100...	[-121.834335, 39.63587147039632, -120.51955, 4...	Feature	[{'rel': 'collection', 'type': 'application/js...	POLYGC ((-121.817 40.6448 -120.519 40.6

5 rows × 214 columns

```
import hvplot
import cartopy
import geoviews

plot_polygons(aoi) * items_gdf.hvplot.paths(geo=True)
```

OpenDataCube

Now we'll turn the set of scenes into a virtual datacube. None of the data will actually be read yet.

The configuration string (`cfg`) is for providing additional info not currently available in the STAC Items, but will be in the future.

```
import yaml

cfg = """---
sentinel-2-l2a:
  assets:
    '*':
      data_type: uint16
      nodata: 0
      unit: '1'
"""
cfg = yaml.load(cfg, Loader=yaml.CSafeLoader)
```

Here we load as a DataCube. A PySTAC ItemCollection is created from the found STAC Items, and we specify various parameters, such as bands of interest and chunk size. We are requesting to only load pixels within a bounding box of the requested geometry (`bbox=geom.bounds`).

```
%%time

from odc.stac import stac_load
import planetary_computer as pc

# Create PySTAC ItemCollection
item_collection = pystac.ItemCollection(items_dict)

dc = stac_load(item_collection,
               measurements=['B02', 'B03', 'B04', 'B08'],
               chunks={"x": 2048, "y": 2048},
               bbox=geom.bounds,
               stac_cfg=cfg,
               patch_url=pc.sign
)
dc
```

```
CPU times: user 861 ms, sys: 183 ms, total: 1.04 s
Wall time: 2.19 s
<xarray.Dataset>
Dimensions:      (y: 1017, x: 1206, time: 6)
Coordinates:
  * y            (y) float64 4.395e+06 4.395e+06 ... 4.385e+06 4.385e+06
  * x            (x) float64 6.331e+05 6.331e+05 ... 6.451e+05 6.451e+05
    spatial_ref  int32 32610
  * time         (time) datetime64[ns] 2019-10-02T18:52:09.024000 ... 2019-10...
Data variables:
    B02          (time, y, x) uint16 dask.array<chunksize=(1, 1017, 1206), meta=np.ndarray>
    B03          (time, y, x) uint16 dask.array<chunksize=(1, 1017, 1206), meta=np.ndarray>
    B04          (time, y, x) uint16 dask.array<chunksize=(1, 1017, 1206), meta=np.ndarray>
    B08          (time, y, x) uint16 dask.array<chunksize=(1, 1017, 1206), meta=np.ndarray>
```

xarray.Dataset

▶ Dimensions: (**y**: 1017, **x**: 1206, **time**: 6)

▼ Coordinates:

y	(y)	float64	4.395e+06 4.395e+06 ... 4.385e+06
x	(x)	float64	6.331e+05 6.331e+05 ... 6.451e+05
spatial_ref	()	int32	32610
time	(time)	datetime64[ns]	2019-10-02T18:52:09.024000 ... 2...

▼ Data variables:

B02	(time, y, x)	uint16	dask.array<chunksize=(1, 1017, 1206), meta=np....
B03	(time, y, x)	uint16	dask.array<chunksize=(1, 1017, 1206), meta=np....
B04	(time, y, x)	uint16	dask.array<chunksize=(1, 1017, 1206), meta=np....
B08	(time, y, x)	uint16	dask.array<chunksize=(1, 1017, 1206), meta=np....

▶ Indexes: (3)

▶ Attributes: (0)

Calculations

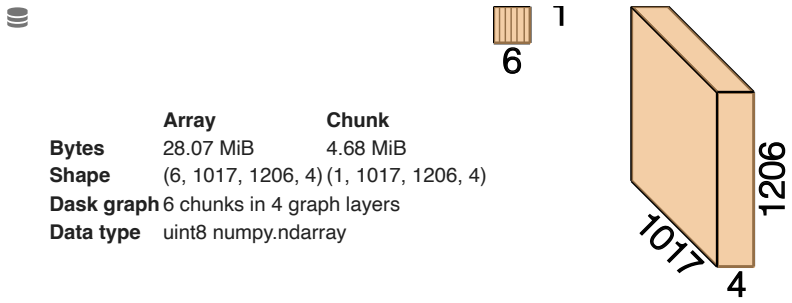
We will create an RGBA datacube representation (`nodata` values have `alpha=0`), and generate an NDVI datacube.

```
from odc.algo import to_rgba

RGB = ['B04', 'B03', 'B02']
vis = to_rgba(dc, clamp=(1, 3000), bands=RGB)
vis
```

```
<xarray.DataArray 'ro_rgba-db0b19e7-7c26eba5b7cfb877331a02c60a54e8ca' (time: 6,
                                                                       y: 1017,
                                                                       x: 1206,
                                                                       band: 4)>
dask.array<ro_rgba-db0b19e7, shape=(6, 1017, 1206, 4), dtype=uint8, chunksize=(1, 1017, 1206, 4), chunktype=numpy.ndarray>
Coordinates:
  * y           (y) float64 4.395e+06 4.395e+06 ... 4.385e+06 4.385e+06
  * x           (x) float64 6.331e+05 6.331e+05 ... 6.451e+05 6.451e+05
    spatial_ref  int32 32610
  * time        (time) datetime64[ns] 2019-10-02T18:52:09.024000 ... 2019-10...
  * band        (band) <U1 'r' 'g' 'b' 'a'
Attributes:
    crs:       PROJCRS["WGS 84 / UTM zone 10N",BASEGEOGCRS["WGS 84",ENSEMBLE["...
```

xarray.DataArray 'ro_rgba-db0b19e7-7c26eba5b7cfb877331a02c60a54e8ca'
(**time**: 6, **y**: 1017, **x**: 1206, **band**: 4)



	Array	Chunk
Bytes	28.07 MiB	4.68 MiB
Shape	(6, 1017, 1206, 4)	(1, 1017, 1206, 4)
Dask graph	6 chunks in 4 graph layers	
Data type	uint8 numpy.ndarray	

▼ Coordinates:

y	(y)	float64	4.395e+06 4.395e+06 ... 4.385e+06
x	(x)	float64	6.331e+05 6.331e+05 ... 6.451e+05
spatial_ref	()	int32	32610
time	(time)	datetime64[ns]	2019-10-02T18:52:09.024000 ... 2...
band	(band)	<U1	'r' 'g' 'b' 'a'

▶ Indexes: (4)

▼ Attributes:

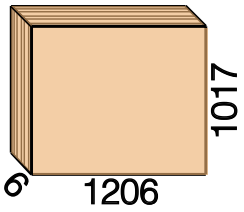
crs :
PROJCRS["WGS 84 / UTM zone 10N",BASEGEOGCRS["WGS 84",ENSEMBLE["World Geodetic System 1984 ensemble",MEMBER["World Geodetic System 1984 (Transit)"],MEMBER["World Geodetic System 1984 (G730)"],MEMBER["World Geodetic System 1984 (G873)"],MEMBER["World Geodetic System 1984 (G1150)"],MEMBER["World Geodetic System 1984 (G1674)"],MEMBER["World Geodetic System 1984 (G1762)"],MEMBER["World Geodetic System 1984 (G2139)"],ELLIPSOID["WGS 84",6378137,298.257223563,LENGTHUNIT["metre",1]],ENSEMBLEACCURACY[2.0]],PRIMEM["Greenwich",0,ANGLEUNIT["degree",0.0174532925199433]],ID["EPSG",4326]],CONVERSION["UTM zone 10N",METHOD["Transverse Mercator",ID["EPSG",9807]],PARAMETER["Latitude of natural origin",0,ANGLEUNIT["degree",0.0174532925199433],ID["EPSG",8801]],PARAMETER["Longitude of natural origin",-123,ANGLEUNIT["degree",0.0174532925199433],ID["EPSG",8802]],PARAMETER["Scale factor at natural origin",0.9996,SCALEUNIT["unity",1],ID["EPSG",8805]],PARAMETER["False easting",500000,LENGTHUNIT["metre",1],ID["EPSG",8806]],PARAMETER["False northing",0,LENGTHUNIT["metre",1],ID["EPSG",8807]]],CS[Cartesian,2],AXIS["(E)",east,ORDER[1],LENGTHUNIT["metre",1]],AXIS["(N)",north,ORDER[2],LENGTHUNIT["metre",1]],USAGE[SCOPE["Navigation and medium accuracy spatial referencing."],AREA["Between 126°W and 120°W, northern hemisphere between equator and 84°N, onshore and offshore. Canada - British Columbia (BC); Northwest Territories (NWT); Nunavut; Yukon. United States (USA) - Alaska (AK)."],BBOX[0,-126,84,-120]],ID["EPSG",32610]]

```
ndvi = ((dc['B08'] - dc['B04']) / (dc['B08'] + dc['B04'])).clip(0, 1).rename("ndvi")
ndvi
```

```
<xarray.DataArray 'ndvi' (time: 6, y: 1017, x: 1206)>
dask.array<clip, shape=(6, 1017, 1206), dtype=float64, chunksize=(1, 1017, 1206), chunktype=numpy.ndarray>
Coordinates:
  * y           (y) float64 4.395e+06 4.395e+06 ... 4.385e+06 4.385e+06
  * x           (x) float64 6.331e+05 6.331e+05 ... 6.451e+05 6.451e+05
    spatial_ref  int32 32610
  * time        (time) datetime64[ns] 2019-10-02T18:52:09.024000 ... 2019-10...
```

xarray.DataArray 'ndvi' (**time**: 6, **y**: 1017, **x**: 1206)

	Array	Chunk
Bytes	56.14 MiB	9.36 MiB
Shape	(6, 1017, 1206)	(1, 1017, 1206)
Dask graph	6 chunks in 6 graph layers	
Data type	float64 numpy.ndarray	



▼ Coordinates:

y	(y)	float64	4.395e+06 4.395e+06 ... 4.385e+06
x	(x)	float64	6.331e+05 6.331e+05 ... 6.451e+05
spatial_ref	()	int32	32610
time	(time)	datetime64[ns]	2019-10-02T18:52:09.024000 ... 2...

▶ Indexes: (3)

▶ Attributes: (0)

```
from dask.distributed import Client
client = Client()
client
```

```
INFO:distributed.http.proxy:To route to workers diagnostics web server please install jupyter-server-proxy: python -m pip install jupyter-server-prox
INFO:distributed.scheduler:State start
INFO:distributed.scheduler:  Scheduler at:     tcp://127.0.0.1:38107
INFO:distributed.scheduler:  dashboard at:  http://127.0.0.1:8787/status
INFO:distributed.nanny:        Start Nanny at: 'tcp://127.0.0.1:36247'
INFO:distributed.nanny:        Start Nanny at: 'tcp://127.0.0.1:33235'
INFO:distributed.scheduler:Register worker <WorkerState 'tcp://127.0.0.1:43969', name: 1, status: init, memory: 0, processing: 0>
INFO:distributed.scheduler:Starting worker compute stream, tcp://127.0.0.1:43969
INFO:distributed.core:Starting established connection to tcp://127.0.0.1:45780
INFO:distributed.scheduler:Register worker <WorkerState 'tcp://127.0.0.1:43099', name: 0, status: init, memory: 0, processing: 0>
INFO:distributed.scheduler:Starting worker compute stream, tcp://127.0.0.1:43099
INFO:distributed.core:Starting established connection to tcp://127.0.0.1:45786
INFO:distributed.scheduler:Receive client connection: Client-273ea471-b9b1-11ee-808b-0242ac1c000c
INFO:distributed.core:Starting established connection to tcp://127.0.0.1:45792
```

Client

Client-273ea471-b9b1-11ee-808b-0242ac1c000c

Connection method: Cluster object **Cluster type:** distributed.LocalCluster

Dashboard: http://127.0.0.1:8787/status

▸ **Cluster Info**

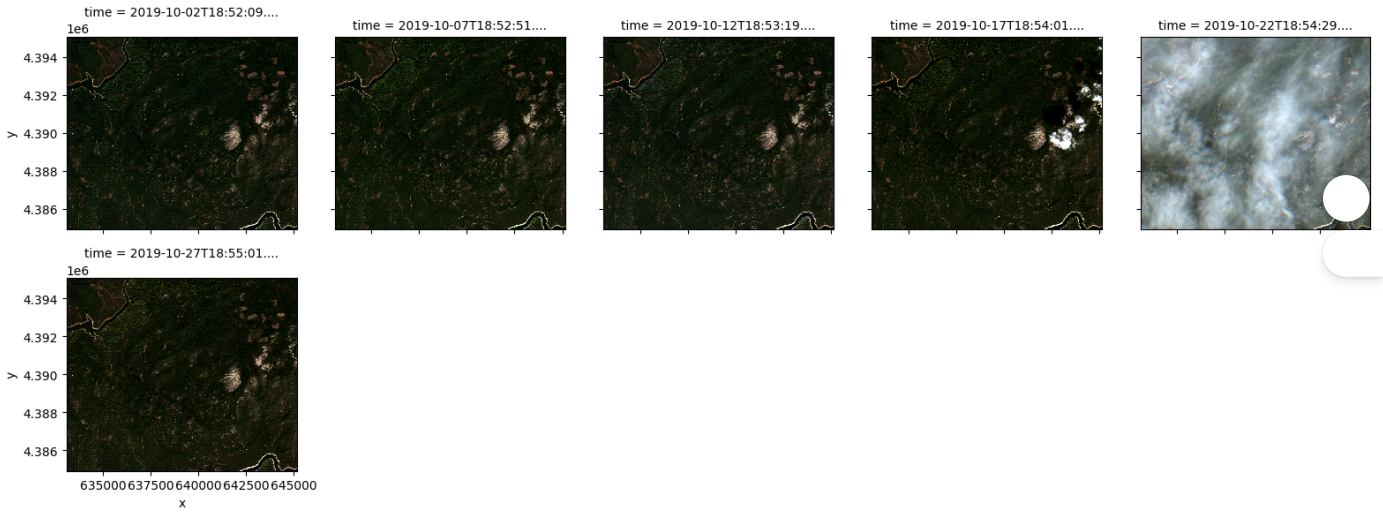
```
%%time
from dask.distributed import wait

ndvi, vis = client.persist([ndvi, vis])
_ = wait([ndvi, vis])
```

```
CPU times: user 1.69 s, sys: 203 ms, total: 1.89 s
Wall time: 18.6 s
```

```
%%time
vis_ = vis.compute()
vis_.plot.imshow(col='time', rgb='band', col_wrap=5, robust=True)
```

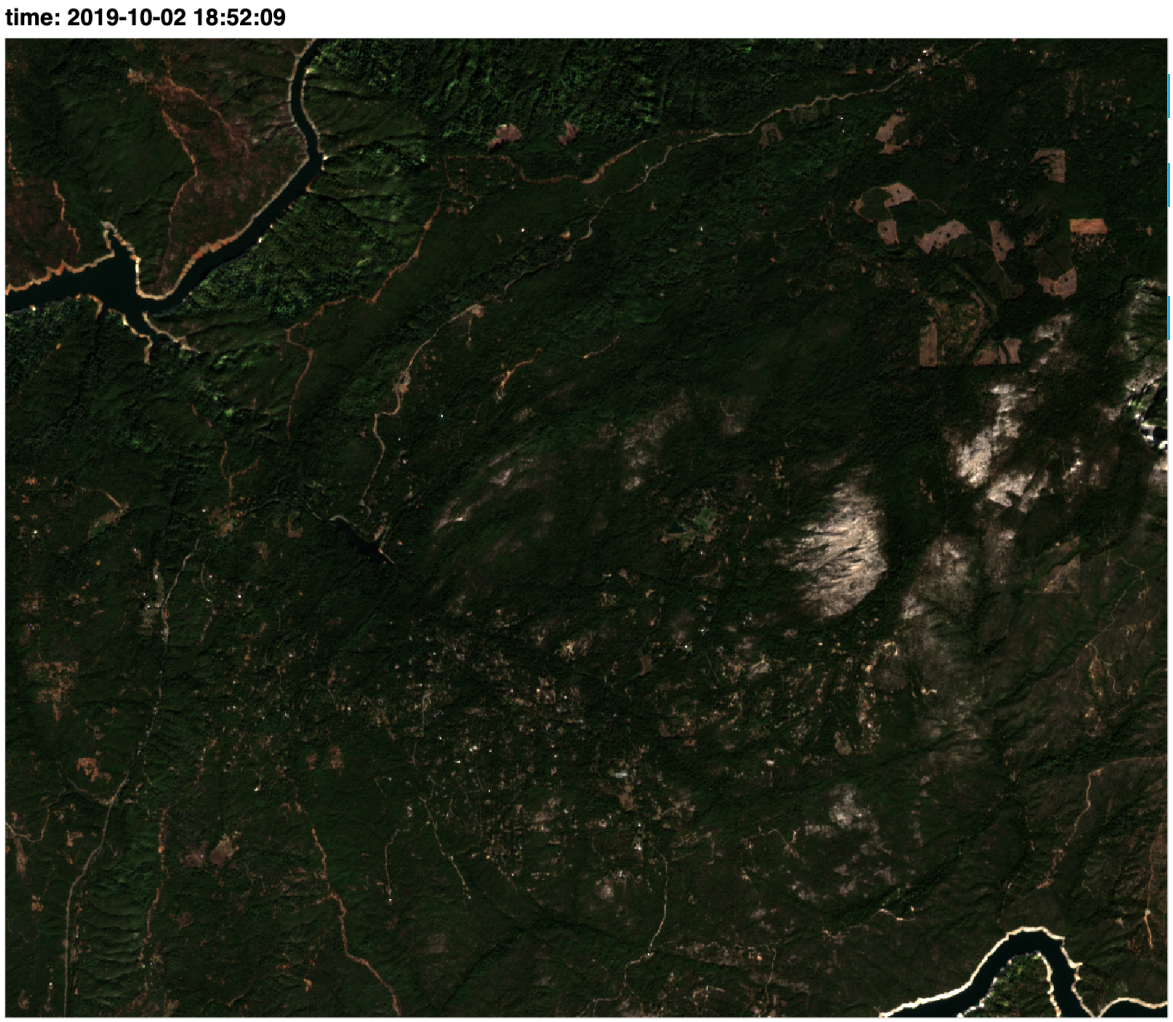
```
CPU times: user 2.67 s, sys: 1.4 s, total: 4.07 s
Wall time: 3.8 s
<xarray.plot.facetgrid.FacetGrid at 0x78a738233af0>
```



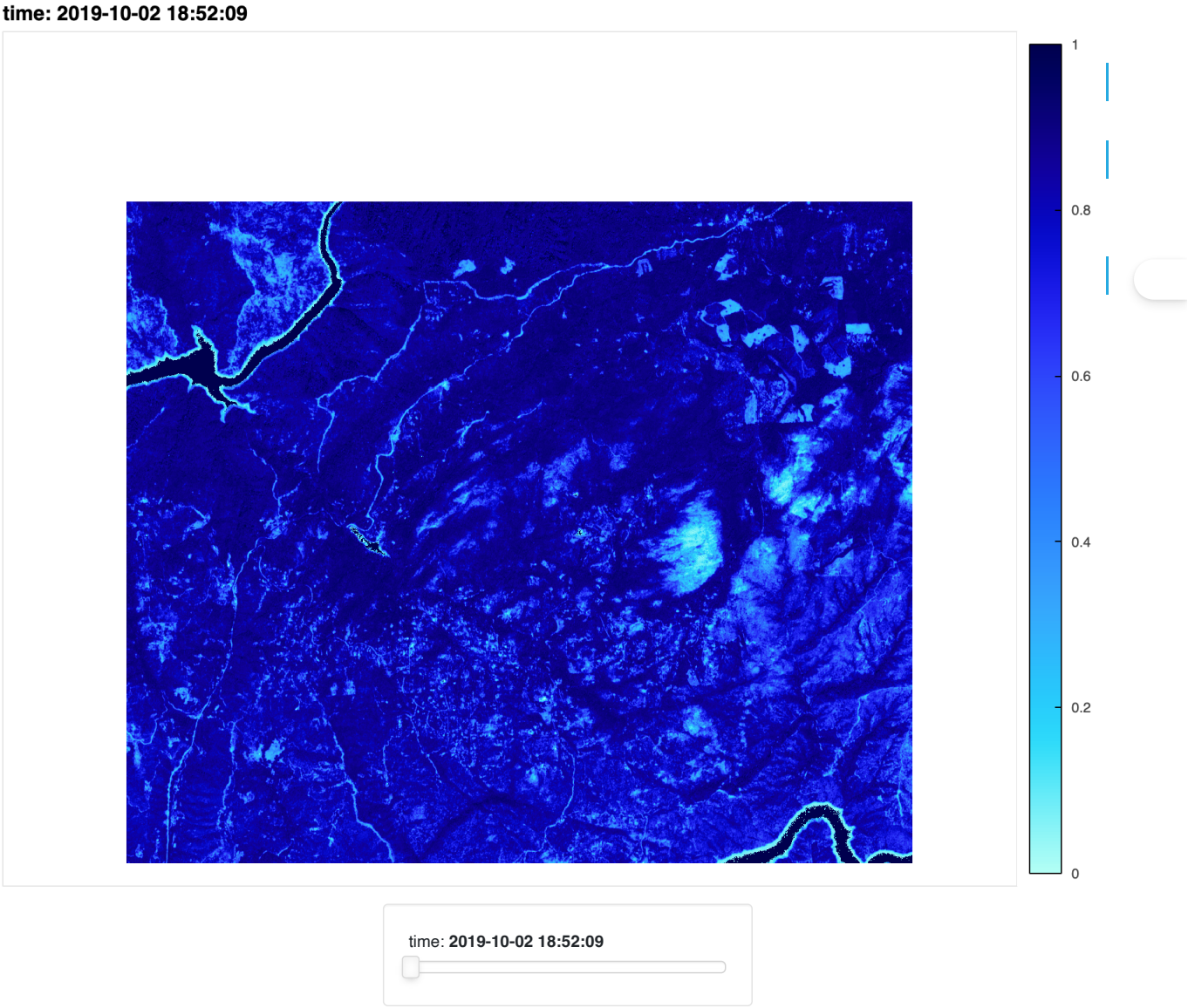

```
import hvplot.xarray

hvplot_kwargs = {
    "frame_width": 800,
    "xaxis": None,
    "yaxis": None,
    "widget_location": "bottom",
    "aspect": len(vis.x)/len(vis.y)
}

vis_.hvplot.rgb('x', 'y', bands='band', groupby='time', **hvplot_kwargs)
```




```
ndvi_ = ndvi.compute()
ndvi_.hvplot('x', 'y', groupby='time', **hvplot_kwargs)
```



Create an animated GIF of NDVI over time using `geogif` with the fetched results.

```
%%time
from geogif import gif, dgif

gif(ndvi_, fps=5, cmap='YlGn')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<timed exec> in <module>

/usr/local/lib/python3.10/dist-packages/geogif/gif.py in gif(arr, to, fps, robust, vmin, vmax, cmap, date_format, date_position, date_color, date_bg)
    236
    237     out = to if to is not None else io.BytesIO()
--> 238     imgs[0].save(
    239         out,
    240         format="gif",

                                4 frames
/usr/local/lib/python3.10/dist-packages/PIL/GifImagePlugin.py in _getbbox(base_im, im_frame)
    572             im_frame.convert("RGBA"), base_im.convert("RGBA")
    573         )
--> 574     return delta.getbbox(alpha_only=False)
    575
    576

TypeError: Image.getbbox() got an unexpected keyword argument 'alpha_only'
```

EXPLAIN ERROR

```
%%time
ndvi_mean = ndvi.mean(dim=['x', 'y']).compute()
ndvi_mean.hvplot()
```

```
CPU times: user 126 ms, sys: 9.41 ms, total: 135 ms
Wall time: 321 ms
```

```
if "cluster" in locals():
    cluster.close()
```

Exploring data interoperability with STAC and the Microsoft Planetary Computer

By Pete Gadomski.

Initially presented at PECORA 2022 on 26 Oct in Denver.

Requirements

Install into your Python environment, e.g. w/ pip.

- pystac-client
- rich
- folium
- shapely
- odc-stac
- planetary-computer
- dask-gateway (optional)

Discovery

First, let's find out what collections are available to us on the Planetary Computer. We'll use pystac-client to query the Planetary Computer's STAC API.

```
from pystac_client import Client
from rich.table import Table

PLANETARY_COMPUTER = "https://planetarycomputer.microsoft.com/api/stac/v1"
client = Client.open(PLANETARY_COMPUTER)
collections = list(client.get_all_collections())
collections.sort(key=lambda c: c.id)
table = Table("ID", "Title", title="Planetary Computer collections")
for collection in collections:
    table.add_row(collection.id, collection.title)
table
```

Planetary Computer collections

ID	Title
3dep-lidar-classification	USGS 3DEP Lidar Classification
3dep-lidar-copc	USGS 3DEP Lidar Point Cloud
3dep-lidar-dsm	USGS 3DEP Lidar Digital Surface Model
3dep-lidar-dtm	USGS 3DEP Lidar Digital Terrain Model
3dep-lidar-dtm-native	USGS 3DEP Lidar Digital Terrain Model (Native)
3dep-lidar-hag	USGS 3DEP Lidar Height above Ground
3dep-lidar-intensity	USGS 3DEP Lidar Intensity
3dep-lidar-pointsourceid	USGS 3DEP Lidar Point Source
3dep-lidar-returns	USGS 3DEP Lidar Returns
3dep-seamless	USGS 3DEP Seamless DEMs
alos-dem	ALOS World 3D-30m
alos-fnf-mosaic	ALOS Forest/Non-Forest Annual Mosaic
alos-palsar-mosaic	ALOS PALSAR Annual Mosaic
aster-l1t	ASTER L1T
chesapeake-lc-13	Chesapeake Land Cover (13-class)
chesapeake-lc-7	Chesapeake Land Cover (7-class)
chesapeake-lu	Chesapeake Land Use
chloris-biomass	Chloris Biomass
cil-gdpcir-cc-by	CIL Global Downscaled Projections for Climate Impacts Research (CC-BY-4.0)
cil-gdpcir-cc-by-sa	CIL Global Downscaled Projections for Climate Impacts Research (CC-BY-SA-4.0)
cil-gdpcir-cc0	CIL Global Downscaled Projections for Climate Impacts Research (CC0-1.0)
cop-dem-glo-30	Copernicus DEM GLO-30
cop-dem-glo-90	Copernicus DEM GLO-90
daymet-annual-hi	Daymet Annual Hawaii
daymet-annual-na	Daymet Annual North America
daymet-annual-pr	Daymet Annual Puerto Rico
daymet-daily-hi	Daymet Daily Hawaii
daymet-daily-na	Daymet Daily North America
daymet-daily-pr	Daymet Daily Puerto Rico
daymet-monthly-hi	Daymet Monthly Hawaii
daymet-monthly-na	Daymet Monthly North America
daymet-monthly-pr	Daymet Monthly Puerto Rico
deltares-floods	Deltares Global Flood Maps
deltares-water-availability	Deltares Global Water Availability
drcog-lulc	Denver Regional Council of Governments Land Use Land Cover
eclipse	Urban Innovation Eclipse Sensor Data
ecmwf-forecast	ECMWF Open Data (real-time)
era5-pds	ERA5 - PDS
esa-cci-lc	ESA Climate Change Initiative Land Cover Maps (Cloud Optimized GeoTIFF)
esa-cci-lc-netcdf	ESA Climate Change Initiative Land Cover Maps (NetCDF)
esa-worldcover	ESA WorldCover
fia	Forest Inventory and Analysis
fws-nwi	FWS National Wetlands Inventory
gap	USGS Gap Land Cover
gbif	Global Biodiversity Information Facility (GBIF)
gnatsgo-rasters	gNATSGO Soil Database - Rasters
gnatsgo-tables	gNATSGO Soil Database - Tables
goes-cmi	GOES-R Cloud & Moisture Imagery
goes-glm	GOES-R Lightning Detection
gpm-imerg-hhr	GPM IMERG
gridmet	gridMET
hgb	HGB: Harmonized Global Biomass for 2010
hrea	HREA: High Resolution Electricity Access
io-biodiversity	Biodiversity Intactness
io-lulc	Esri 10-Meter Land Cover (10-class)
io-lulc-9-class	10m Annual Land Use Land Cover (9-class)
jrc-gsw	JRC Global Surface Water
kaza-hydroforecast	HydroForecast - Kwando & Upper Zambezi Rivers
landsat-c2-l1	Landsat Collection 2 Level-1
landsat-c2-l2	Landsat Collection 2 Level-2
mobi	MoBI: Map of Biodiversity Importance
modis-09A1-061	MODIS Surface Reflectance 8-Day (500m)
modis-09Q1-061	MODIS Surface Reflectance 8-Day (250m)
modis-10A1-061	MODIS Snow Cover Daily
modis-10A2-061	MODIS Snow Cover 8-day
modis-11A1-061	MODIS Land Surface Temperature/Emissivity Daily
modis-11A2-061	MODIS Land Surface Temperature/Emissivity 8-Day
modis-13A1-061	MODIS Vegetation Indices 16-Day (500m)
modis-13Q1-061	MODIS Vegetation Indices 16-Day (250m)
modis-14A1-061	MODIS Thermal Anomalies/Fire Daily
modis-14A2-061	MODIS Thermal Anomalies/Fire 8-Day
modis-15A2H-061	MODIS Leaf Area Index/FPAR 8-Day
modis-15A3H-061	MODIS Leaf Area Index/FPAR 4-Day
modis-16A3GF-061	MODIS Net Evapotranspiration Yearly Gap-Filled
modis-17A2H-061	MODIS Gross Primary Productivity 8-Day

modis-17A2HGF-061	MODIS Gross Primary Productivity 8-Day Gap-Filled
modis-17A3HGF-061	MODIS Net Primary Production Yearly Gap-Filled
modis-21A2-061	MODIS Land Surface Temperature/3-Band Emissivity 8-Day
modis-43A4-061	MODIS Nadir BRDF-Adjusted Reflectance (NBAR) Daily
modis-64A1-061	MODIS Burned Area Monthly
ms-buildings	Microsoft Building Footprints
mtbs	MTBS: Monitoring Trends in Burn Severity
naip	NAIP: National Agriculture Imagery Program
nasa-nex-gddp-cmip6	Earth Exchange Global Daily Downscaled Projections (NEX-GDDP-CMIP6)
nasadem	NASADEM HGT v001
noaa-c-cap	C-CAP Regional Land Cover and Change
noaa-cdr-ocean-heat-content	Global Ocean Heat Content CDR
noaa-cdr-ocean-heat-content-netcdf	Global Ocean Heat Content CDR NetCDFs
noaa-cdr-sea-surface-temperature-optimum-interpolation	Sea Surface Temperature - Optimum Interpolation CD
noaa-cdr-sea-surface-temperature-whoi	Sea Surface Temperature - WHOI CDR
noaa-cdr-sea-surface-temperature-whoi-netcdf	Sea Surface Temperature - WHOI CDR NetCDFs
noaa-climate-normals-gridded	NOAA US Gridded Climate Normals (Cloud-Optimized GeoTIFF)
noaa-climate-normals-netcdf	NOAA US Gridded Climate Normals (NetCDF)
noaa-climate-normals-tabular	NOAA US Tabular Climate Normals
noaa-mrms-qpe-1h-pass1	NOAA MRMS QPE 1-Hour Pass 1
noaa-mrms-qpe-1h-pass2	NOAA MRMS QPE 1-Hour Pass 2
noaa-mrms-qpe-24h-pass2	NOAA MRMS QPE 24-Hour Pass 2
noaa-nclimgrid-monthly	Monthly NOAA U.S. Climate Gridded Dataset (NClimGrid)
nrcan-landcover	Land Cover of Canada
planet-nicfi-analytic	Planet-NICFI Basemaps (Analytic)
planet-nicfi-visual	Planet-NICFI Basemaps (Visual)
sentinel-1-grd	Sentinel 1 Level-1 Ground Range Detected (GRD)
sentinel-1-rtc	Sentinel 1 Radiometrically Terrain Corrected (RTC)
sentinel-2-l2a	Sentinel-2 Level-2A
sentinel-3-olci-lfr-l2-netcdf	Sentinel-3 Land (Full Resolution)
sentinel-3-olci-wfr-l2-netcdf	Sentinel-3 Water (Full Resolution)
sentinel-3-slstr-frp-l2-netcdf	Sentinel-3 Fire Radiative Power
sentinel-3-slstr-lst-l2-netcdf	Sentinel-3 Land Surface Temperature
sentinel-3-slstr-wst-l2-netcdf	Sentinel-3 Sea Surface Temperature
sentinel-3-sral-lan-l2-netcdf	Sentinel-3 Land Radar Altimetry
sentinel-3-sral-wat-l2-netcdf	Sentinel-3 Ocean Radar Altimetry
sentinel-3-synergy-aod-l2-netcdf	Sentinel-3 Global Aerosol
sentinel-3-synergy-syn-l2-netcdf	Sentinel-3 Land Surface Reflectance and Aerosol
sentinel-3-synergy-v10-l2-netcdf	Sentinel-3 10-Day Surface Reflectance and NDVI (SPOT VEGETATION)
sentinel-3-synergy-vg1-l2-netcdf	Sentinel-3 1-Day Surface Reflectance and NDVI (SPOT VEGETATION)
sentinel-3-synergy-vgp-l2-netcdf	Sentinel-3 Top of Atmosphere Reflectance (SPOT VEGETATION)
sentinel-5p-l2-netcdf	Sentinel-5P Level-2
terraclimate	TerraClimate
us-census	US Census
usda-cdl	USDA Cropland Data Layers (CDLs)
usgs-lcmap-conus-v13	USGS LCMAP CONUS Collection 1.3
usgs-lcmap-hawaii-v10	USGS LCMAP Hawaii Collection 1.0

Getting our Landsat collection id

As you can see, there's a ton of collections. Because this is PECORA, let's narrow in on the Landsat collections for now.

```
from rich.markdown import Markdown

landsat_collections = [c for c in collections if c.id.startswith("landsat")]
table = Table("ID", "Title", "Description",
              title="Planetary Computer Landsat collections")
for collection in landsat_collections:
    table.add_row(collection.id,
                  collection.title,
                  Markdown(collection.description))
table
```

Planetary Computer Landsat collections

ID	Title	Description
landsat-c2-l1	Landsat Collection 2 Level-1	Landsat Collection 2 Level-1 data, consisting of quantized and calibrated scaled Digital Numbers (DN) representing the multispectral image data. These Level-1 data can be rescaled to top of atmosphere (TOA) reflectance and/or radiance. Thermal band data can be rescaled to TOA brightness temperature. This dataset represents the global archive of Level-1 data from Landsat Collection 2 acquired by the Multispectral Scanner System onboard Landsat 1 through Landsat 5 from July 7, 1972 to January 7, 2013. Images are stored in cloud-optimized GeoTIFF format.
landsat-c2-l2	Landsat Collection 2 Level-2	Landsat Collection 2 Level-2 Science Products, consisting of atmospherically corrected surface reflectance and surface temperature image data. Collection 2 Level-2 Science Products are available from August 22, 1982 to present. This dataset represents the global archive of Level-2 data from Landsat Collection 2 acquired by the Thematic Mapper onboard Landsat 4 and 5, the Enhanced Thematic Mapper onboard Landsat 7, and the Operatational Land Imager and Thermal Infrared Sensor onboard Landsat 8 and 9. Images are stored in cloud-optimized GeoTIFF format.

Choosing the correct Landsat collection

As you can see, there's currently three Landsat collections in the Planetary Computer. Which one to use? The two `landsat-c2-*` collections seem to be of a pair, but what's going on with `landsat-8-c2-l2`?

Turns out that `landsat-8-c2-l2` is an old collection that has been superseded by `landsat-c2-l2`. There's no way to know this from the STAC metadata, though we (being the Planetary Computer team) should probably fix that by marking it deprecated with the version extension. The deprecated collection is hidden from the website interface, if you use that.

But, for now, we'll use `landsat-c2-l2` for Landsat Level 2 data.

Fetching and displaying data

First, let's demonstrate loading data using **pystac-client** and odc-stac. **odc-stac** is part of the OpenDataCube, ecosystem, but stands alone as its own software package. If you're familiar with OpenDataCube, you know that it's primary product is a database-cum-analysis software stack used for analyzing geospatial data, not dissimilar to the Planetary Computer. **odc-stac** represents a coming together between the OpenDataCube and the STAC ecosystems, because it take STAC items and turns them into analysis-ready xarrays using routines developed for the OpenDataCube. **odc-stac** *does not* require a database, or any significant dependencies.

We'll define an area around Steamboat Springs, Colorado, and a datetime range during the winter. We'll also search for items that have a sufficiently low amount enough of cloud cover to create a good mosaic. First, let's find the STAC items that match our criteria.

```
LANDSAT_COLLECTION = "landsat-c2-l2"
# this is a simplified datetime syntax that pystac-client understands
DATETIME = "2022-04"
CLOUD_COVER_THRESHOLD = 10  # percent
#BBOX = [-107.381493, 40.118423, -106.331366, 40.960106]  # Steamboat Springs, CO
# You can use geojson.io tool to select an area of interest or bounding box
BBOX = [-110.9701438117958, 32.28943920624256,  -110.57661859175865, 32.593051001253556]  # Catalina Mountains, Tucson,

item_search = client.search(
    collections=[LANDSAT_COLLECTION],
    bbox=BBOX,
    datetime=DATETIME,
    query=[f"eo:cloud_cover<={CLOUD_COVER_THRESHOLD}"],
)
item_collection = item_search.item_collection()
print(f"Found {len(item_collection)} items")
```

```
Found 14 items
```

Showing the STAC footprints

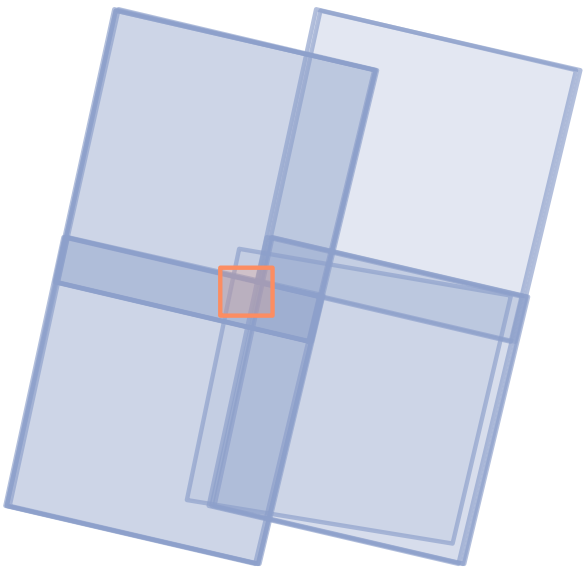
So far, we've only fetched STAC metadata, which are just JSON objects. We haven't fetched any raster data at all. How do the footprints of our STAC items relate to our bounding box of interest? Let's look, using **folium**.

```
from folium import Map, GeoJson
from shapely.geometry import box

item_style_function = lambda s: {
    "color": "#8da0cbaa"
}
bbox_style_function = lambda s: {
    "color": "#fc8d62"
}

map = Map(tiles="Stamen Watercolor")
minx = 180
maxx = -180
miny = 90
maxy = -90
for item in item_collection:
    if item.bbox[0] < minx:
        minx = item.bbox[0]
    if item.bbox[1] < miny:
        miny = item.bbox[1]
    if item.bbox[2] > maxx:
        maxx = item.bbox[2]
    if item.bbox[3] > maxy:
        maxy = item.bbox[3]
    GeoJson(item.to_dict(), style_function=item_style_function).add_to(map)
GeoJson(box(*BBOX), style_function=bbox_style_function).add_to(map)
map.fit_bounds([[miny, minx], [maxy, maxx]])
display(map)
```

Make this Notebook Trusted to load map: File -> Trust Notebook



Leaflet (https://leafletjs.com) | Map tiles by Stamen Design (http://stamen.com), under CC BY 3.0 (http://creativecommons.org/licenses/by/3.0). Data by © OpenStreetMap (http://openstreetmap.org), under CC BY SA (http://creativecommons.org/licenses/by-sa/3.0).

Reading data only in our area of interest

As you can see, the item bounds extend well beyond our bounding box of interest. While we could load all the data from all items, that's inefficient. **odc-stac** can use our bounding box and the fact that our Landsat data is stored as Cloud-Optimized GeoTIFFs (COGs) to only read the bits we need, not all of the data. Let's show that in action, loading only the RGB bands for quick visualization. Note that we use the Planetary Computer API to sign the asset hrefs, which appends a Shared Access Signature (SAS) to allow access to the data assets.

We load the data at 50m resolution to speed up display operations. For analysis, you'll most likely want to load at native resolution.

```
import odc.stac
import planetary_computer

item_collection = planetary_computer.sign_item_collection(item_collection)
data = odc.stac.load(
    item_collection,
    bands=["red", "green", "blue"],
    groupby="solar_day",
    bbox=BBOX,
    resolution=50
)
data
```

```
<xarray.Dataset>
Dimensions:      (y: 676, x: 742, time: 7)
Coordinates:
  * y            (y) float64 3.606e+06 3.606e+06 ... 3.573e+06 3.572e+06
  * x            (x) float64 5.028e+05 5.028e+05 ... 5.398e+05 5.398e+05
    spatial_ref  int32 32612
  * time         (time) datetime64[ns] 2022-04-01T17:57:33.928182 ... 2022-04...
Data variables:
    red          (time, y, x) uint16 11190 11682 12112 ... 13693 14161 14305
    green        (time, y, x) uint16 10535 10869 11169 ... 12407 12823 12852
    blue         (time, y, x) uint16 9432 9630 9801 9931 ... 11079 11329 11323
```

xarray.Dataset

▶ Dimensions: (**y**: 676, **x**: 742, **time**: 7)

▼ Coordinates:

y	(y)	float64	3.606e+06 3.606e+06 ... 3.572e+06
x	(x)	float64	5.028e+05 5.028e+05 ... 5.398e+05
spatial_ref	()	int32	32612
time	(time)	datetime64[ns]	2022-04-01T17:57:33.928182 ... 2...

▼ Data variables:

red	(time, y, x)	uint16	11190 11682 12112 ... 14161 14305
green	(time, y, x)	uint16	10535 10869 11169 ... 12823 12852
blue	(time, y, x)	uint16	9432 9630 9801 ... 11329 11323

▶ Indexes: (3)

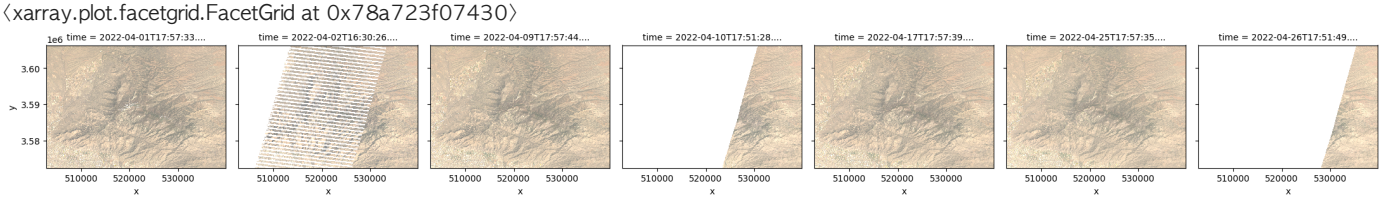
▶ Attributes: (0)

Plotting RGB data

There's nothing better than visualizing your data as full-color images.

```
rgb_data = data.odc.to_rgba(vmin=1, vmax=15000, bands=["red", "green", "blue"])
rgb_data.plot.imshow(col="time", rgb="band", robust=True)
```

```
<xarray.plot.facetgrid.FacetGrid at 0x78a723f07430>
```



FilmDrop Analytics with Sentinel-2

This notebook explores Sentinel-2 data on Earth Search from within Element 84's FilmDrop environment.

- Earth Search STAC API: https://earth-search.aws.element84.com/v1/, catalog of public data
- pystac-client: https://pystac-client.readthedocs.io/, for searching and access data
- OpenDataCube: https://www.opendatacube.org/ and odc-stac https://odc-stac.readthedocs.io/ for loading STAC assets and representing geospatial data as XArrays
- XArray: http://xarray.pydata.org/en/stable/, pandas https://pandas.pydata.org/ and geopandas https://geopandas.org/ for manipulating data
- Dask: https://dask.org/ for performing parallel, distributed computing
- Folium https://python-visualization.github.io/folium/index.html and hvplot https://hvplot.holoviz.org/ for visualization

Shown will be how find data for an area of interest, explore the resulting metadata, perform calculations like NDVI, and visualize the results.

```
# set pystac_client logger to DEBUG to see API calls
import logging
logging.basicConfig()
logger = logging.getLogger('pystac_client')
logger.setLevel(logging.INFO)

# ItemMap class for display on a slippy map

import folium
import requests

class ItemMap(object):
    _colors = {
        'red': '#fc0f03',
        'green': '#27AD0C',
        'blue': '#0f03fc'
    }

    def __init__(self, item, tiles='Stamen Watercolor'):
        self.item = item
        self.m = folium.Map(tiles=tiles)
        sw = item.bbox[1], item.bbox[0]
        ne = item.bbox[3], item.bbox[2]
        self.m.fit_bounds([sw, ne])
        self.legend = {}

    def display(self):
        return self.m

    @classmethod
    def create_map(cls, item, **kwargs):
        m = cls(item[0], **kwargs)

        # original footprint
        m.add_item(item, name='', color='red', weight=6)

        # add image asset
        #href = item.assets[asset].href
        #m.add_asset(href)
        return m

    def add_item(self, item, name, color='red', weight=2):
        style = {
            'fillColor': '#00000000',
            'color': self._colors[color],
            'weight': weight
        }
        folium.GeoJson(item.to_dict(), style_function=lambda x: style).add_to(self.m)
        #label = f"{name} {summarize_geometry(item)}"
        #self.legend[label] = self._colors[color]

    def add_geom(self, geom, name, color='blue', weight=2):
        style = {
            'fillColor': '#00000000',
            'color': self._colors[color],
            'weight': weight
        }
        folium.GeoJson(geom, style_function=lambda x: style).add_to(self.m)

    def add_asset(self, href):
        # add image
        stats = requests.get(f"http://titiler:8000/cog/statistics?url=" + href).json()['1']
        tileset = "http://127.0.0.1:8000/cog/tiles/{z}/{x}/{y}?&url=" + href
        tileset = tileset + f"&rescale={stats['percentile_2']},{stats['percentile_98']}"
        tile_layer = folium.TileLayer(
            tiles = tileset,
            attr=item.id
        )
        tile_layer.add_to(self.m)
```

```
# Use pystac-client to find data in the Earth Search STAC API.
#
# Open the Earth Search STAC API

from pystac_client import Client
URL = 'https://earth-search.aws.element84.com/v1/'
api = Client.open(URL)
print(api)
```

```
〈Client id=earth-search-aws〉
```

```
# Fetch the collection of interest and print the assets that are available.
import pandas as pd

collection = api.get_collection('sentinel-2-l2a')
pd.DataFrame.from_dict(collection.to_dict()['item_assets'], orient='index')
```

	type	title	proj:shape	proj:transform	raster:bands	roles	eo:bands	gsd
aot	image/tiff; application=geotiff; profile=cloud...	Aerosol optical thickness (AOT)	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	NaN	NaN
blue	image/tiff; application=geotiff; profile=cloud...	Blue (band 2) - 10m	[10980, 10980]	[10, 0, 399960, 0, -10, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'blue', 'common_name': 'blue', 'desc...	10.0
coastal	image/tiff; application=geotiff; profile=cloud...	Coastal aerosol (band 1) - 60m	[1830, 1830]	[60, 0, 399960, 0, -60, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'coastal', 'common_name': 'coastal',...	60.0
granule_metadata	application/xml	NaN	NaN	NaN	NaN	[metadata]	NaN	NaN
green	image/tiff; application=geotiff; profile=cloud...	Green (band 3) - 10m	[10980, 10980]	[10, 0, 399960, 0, -10, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'green', 'common_name': 'green', 'de...	10.0
nir	image/tiff; application=geotiff; profile=cloud...	NIR 1 (band 8) - 10m	[10980, 10980]	[10, 0, 399960, 0, -10, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'nir', 'common_name': 'nir', 'descri...	10.0
nir08	image/tiff; application=geotiff; profile=cloud...	NIR 2 (band 8A) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'nir08', 'common_name': 'nir08', 'de...	20.0
nir09	image/tiff; application=geotiff; profile=cloud...	NIR 3 (band 9) - 60m	[1830, 1830]	[60, 0, 399960, 0, -60, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'nir09', 'common_name': 'nir09', 'de...	60.0
red	image/tiff; application=geotiff; profile=cloud...	Red (band 4) - 10m	[10980, 10980]	[10, 0, 399960, 0, -10, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'red', 'common_name': 'red', 'descri...	10.0
rededge1	image/tiff; application=geotiff; profile=cloud...	Red edge 1 (band 5) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'rededge1', 'common_name': 'rededge'...	20.0
rededge2	image/tiff; application=geotiff; profile=cloud...	Red edge 2 (band 6) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'rededge2', 'common_name': 'rededge'...	20.0
rededge3	image/tiff; application=geotiff; profile=cloud...	Red edge 3 (band 7) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'rededge3', 'common_name': 'rededge'...	20.0
scl	image/tiff; application=geotiff; profile=cloud...	Scene classification map (SCL)	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint8', 'spatial_...	[data, reflectance]	NaN	NaN
swir16	image/tiff; application=geotiff; profile=cloud...	SWIR 1 (band 11) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'swir16', 'common_name': 'swir16', '...	20.0
swir22	image/tiff; application=geotiff; profile=cloud...	SWIR 2 (band 12) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'swir22', 'common_name': 'swir22', '...	20.0
thumbnail	image/jpeg	Thumbnail image	NaN	NaN	NaN	[thumbnail]	NaN	NaN
tileinfo_metadata	application/json	NaN	NaN	NaN	NaN	[metadata]	NaN	NaN
visual	image/tiff; application=geotiff;	True color	[10980,	[10, 0, 399960, 0, -10	NaN	[visual]	[{'name': 'red', 'common_name':	NaN

visual	application=geotiff; profile=cloud...	image	10980]	0, -10, 4900020]	NaN	[visual]	common_name': 'red', 'descri...	NaN
wvp	image/tiff; application=geotiff; profile=cloud...	Water vapour (WVP)	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	NaN	NaN
aot-jp2	image/jp2	Aerosol optical thickness (AOT)	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	NaN	NaN
blue-jp2	image/jp2	Blue (band 2) - 10m	[10980, 10980]	[10, 0, 399960, 0, -10, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'blue', 'common_name': 'blue', 'desc...	10.0
coastal-jp2	image/jp2	Coastal aerosol (band 1) - 60m	[1830, 1830]	[60, 0, 399960, 0, -60, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'coastal', 'common_name': 'coastal',...	60.0
green-jp2	image/jp2	Green (band 3) - 10m	[10980, 10980]	[10, 0, 399960, 0, -10, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'green', 'common_name': 'green', 'de...	10.0
nir-jp2	image/jp2	NIR 1 (band 8) - 10m	[10980, 10980]	[10, 0, 399960, 0, -10, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'nir', 'common_name': 'nir', 'descri...	10.0
nir08-jp2	image/jp2	NIR 2 (band 8A) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'nir08', 'common_name': 'nir08', 'de...	20.0
nir09-jp2	image/jp2	NIR 3 (band 9) - 60m	[1830, 1830]	[60, 0, 399960, 0, -60, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'nir09', 'common_name': 'nir09', 'de...	60.0
red-jp2	image/jp2	Red (band 4) - 10m	[10980, 10980]	[10, 0, 399960, 0, -10, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'red', 'common_name': 'red', 'descri...	10.0
rededge1-jp2	image/jp2	Red edge 1 (band 5) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'rededge1', 'common_name': 'rededge'...	20.0
rededge2-jp2	image/jp2	Red edge 2 (band 6) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'rededge2', 'common_name': 'rededge'...	20.0
rededge3-jp2	image/jp2	Red edge 3 (band 7) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'rededge3', 'common_name': 'rededge'...	20.0
scl-jp2	image/jp2	Scene classification map (SCL)	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint8', 'spatial_...	[data, reflectance]	NaN	NaN
swir16-jp2	image/jp2	SWIR 1 (band 11) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'swir16', 'common_name': 'swir16', '...	20.0
swir22-jp2	image/jp2	SWIR 2 (band 12) - 20m	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	[{'name': 'swir22', 'common_name': 'swir22', '...	20.0
visual-jp2	image/jp2	True color image	[10980, 10980]	[10, 0, 399960, 0, -10, 4900020]	NaN	[visual]	[{'name': 'red', 'common_name': 'red', 'descri...	NaN

wvp-jp2	image/jp2	Water vapour (WVP)	[5490, 5490]	[20, 0, 399960, 0, -20, 4900020]	[{'nodata': 0, 'data_type': 'uint16', 'bits_pe...	[data, reflectance]	NaN	NaN

```
# load the geometry of the AOI (GeoJSON Feature)
filename = "/content/bear-fire.geojson"
from pathlib import Path
from json import loads
geom = loads(Path(filename).read_text())['geometry']

import geopandas as gpd
aoi = gpd.read_file(filename)['geometry'][0]

query = api.search(
    collections=["sentinel-2-l2a"],
    intersects=geom,
    datetime="2019-10-01/2021-10-01",
    limit=100,
    query = [
        "eo:cloud_cover<10"
    ]
)
item_collection = query.item_collection()

print(f"Found: {len(item_collection):d} STAC Items")
item_collection
```

```
Found: 303 STAC Items
```

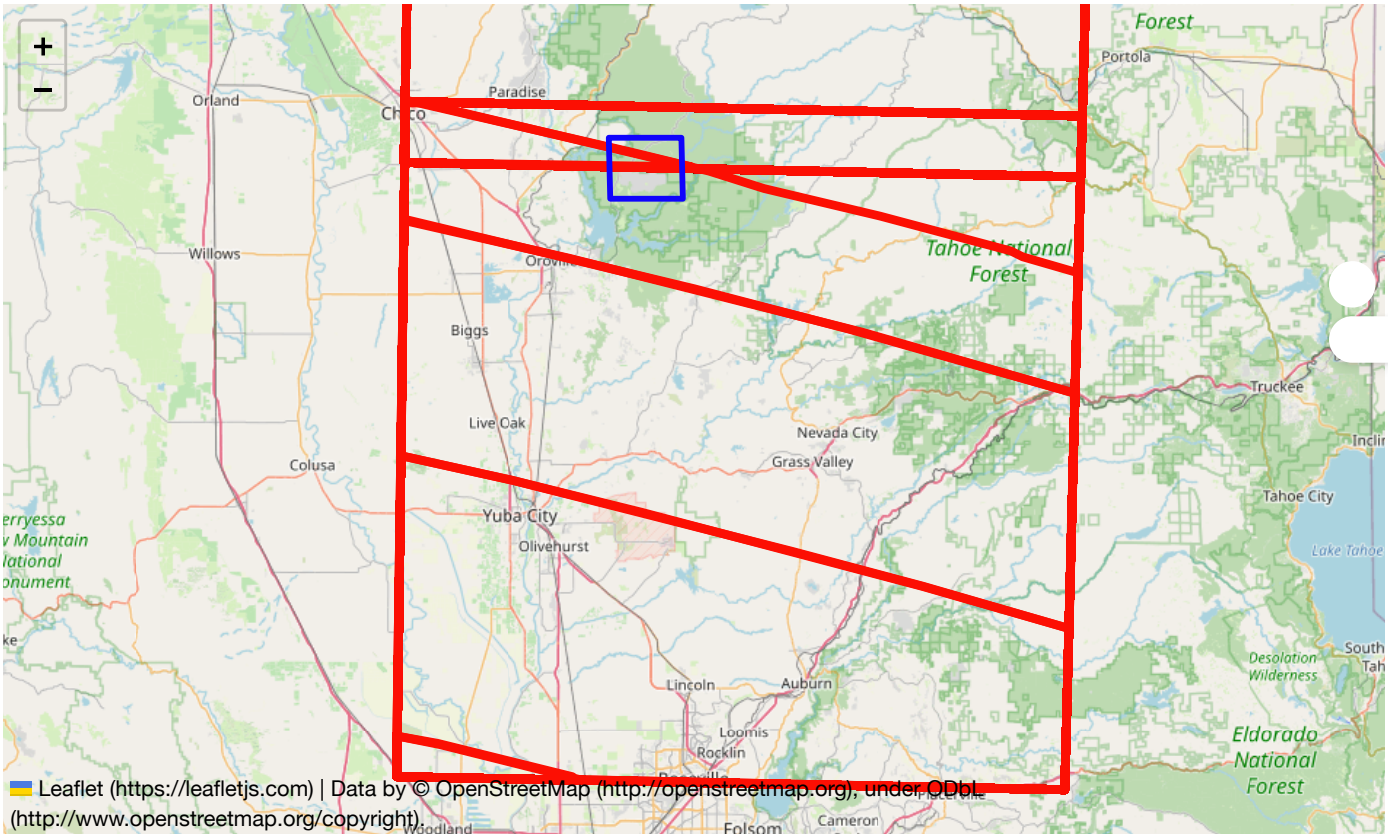
```
# view footprints

asset = 'visual'

m = ItemMap.create_map(item_collection, tiles='OpenStreetMap')

m.add_geom(geom, 'aoi', color='blue', weight=4)

display(m.display())
```




```
%%time
# Here we load as a DataCube. A PySTAC ItemCollection is created from the found STAC Items,
# and we specify various parameters, such as bands of interest and chunk size.
# We are requesting to only load pixels within a bounding box of the requested
# geometry (`bbox=geom.bounds`).

from odc.stac import stac_load

dc = stac_load(item_collection,
               measurements=['red', 'green', 'blue', 'nir'],
               chunks={"x": 1024, "y": 1024},
               bbox=aoi.bounds,
               groupby='solar_day',
)
dc
```