How to convert NetCDF to Points/ Polygons

What is NetCDF?

Dong-Hoon, Lee (이동훈)
dhl@gaia3d.com

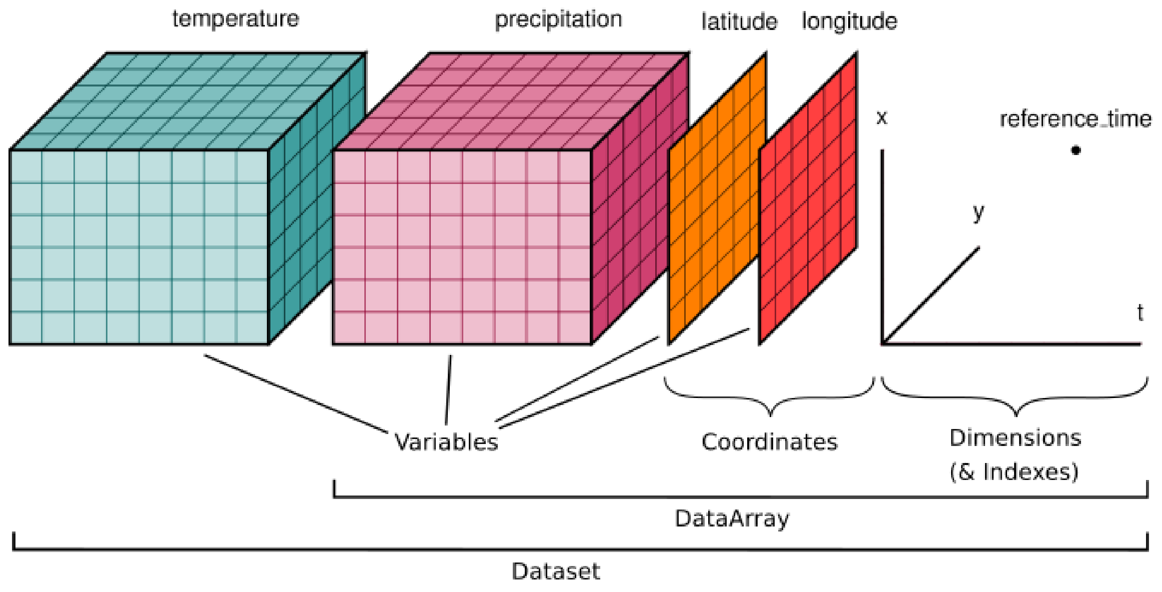
What id NetCDF?

- NetCDF (Network Common Data Form) is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data
- It is commonly used in climatology, meteorology and oceanography applications (e.g., weather forecasting, climate change) and GIS applications.
- https://en.wikipedia.org/wiki/NetCDF

Climatology & Oceanography require time-series and three-dimensional measured information, NetCDF is so useful.

Multidimensional spatio-temporal data

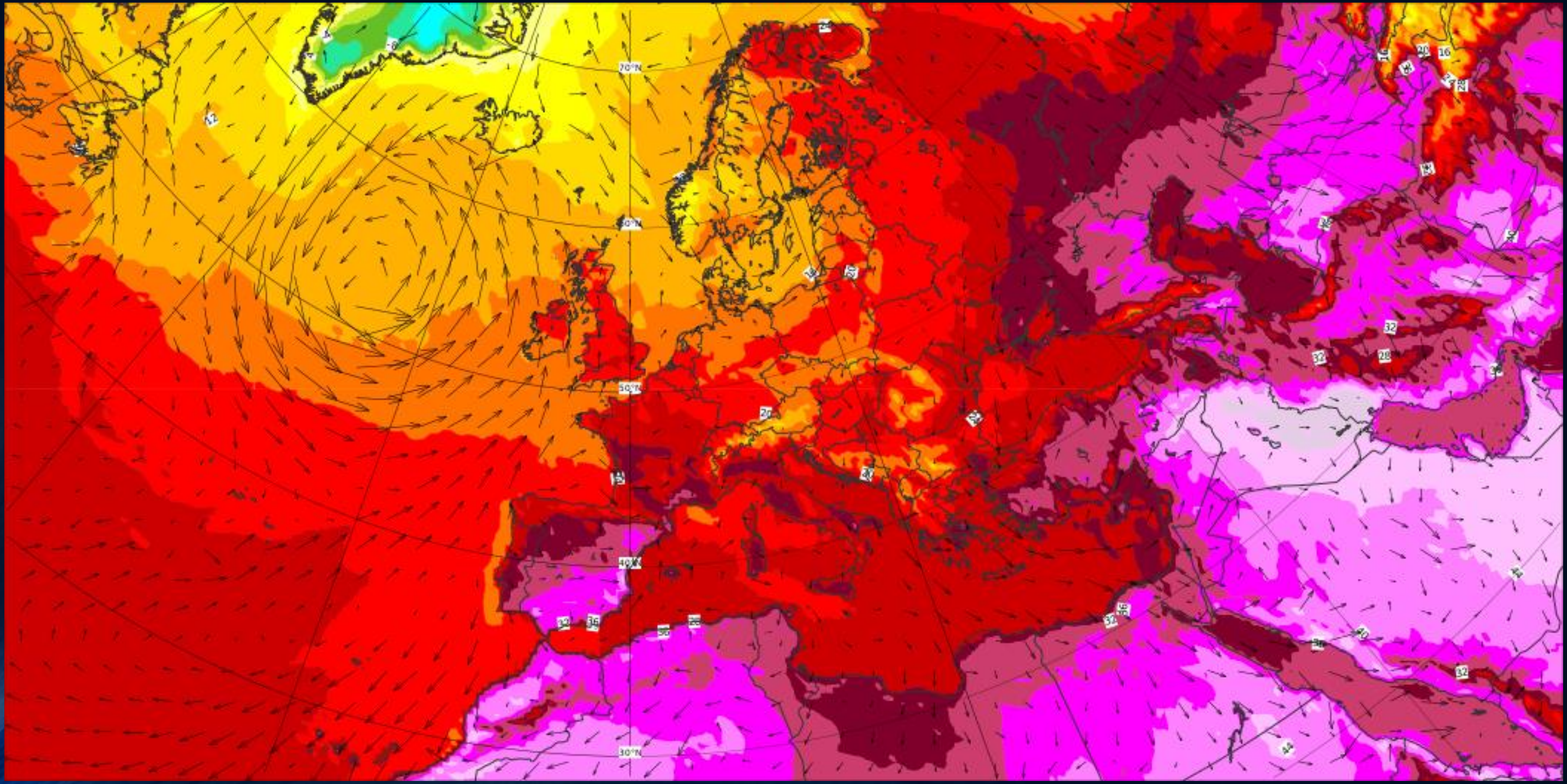
- NetCDF can store multidimensional spatio-temporal information
- Xarray is a Python package that can use multidimensional spatio-temporal data including NetCDF



https://docs.xarray.dev/en/stable/user-guide/data-structures.html

Where can you get that Data?

- ECMWF (https://www.ecmwf.int/)
  - ECMWF is the European Centre for Medium-Range Weather Forecasts



Copernicus (https://www.copernicus.eu/)

ATMOSPHERE
CLIMATE CHANGE
EMERGENCY - CEMS-EFAS
EMERGENCY - CEMS-EFFIS
EMERGENCY - CEMS-GLOFAS
LAND - GLOBAL
LAND - PAN-EUROPEAN/LOCAL LAND
MARINE

Multi Observation Global Ocean 3D Temperature Salinity Height Geostrophic Current and MLD

Home > Marine Data Store > Product

- Description
- Notifications
- Data access
- Contacts

DOCUMENTATION

- User Manual
- Quality Information Document
- Licence
- How to cite

DOI

10.48670/moi-00052

Data access and mapping services

There are multiple ways to download data from this product:

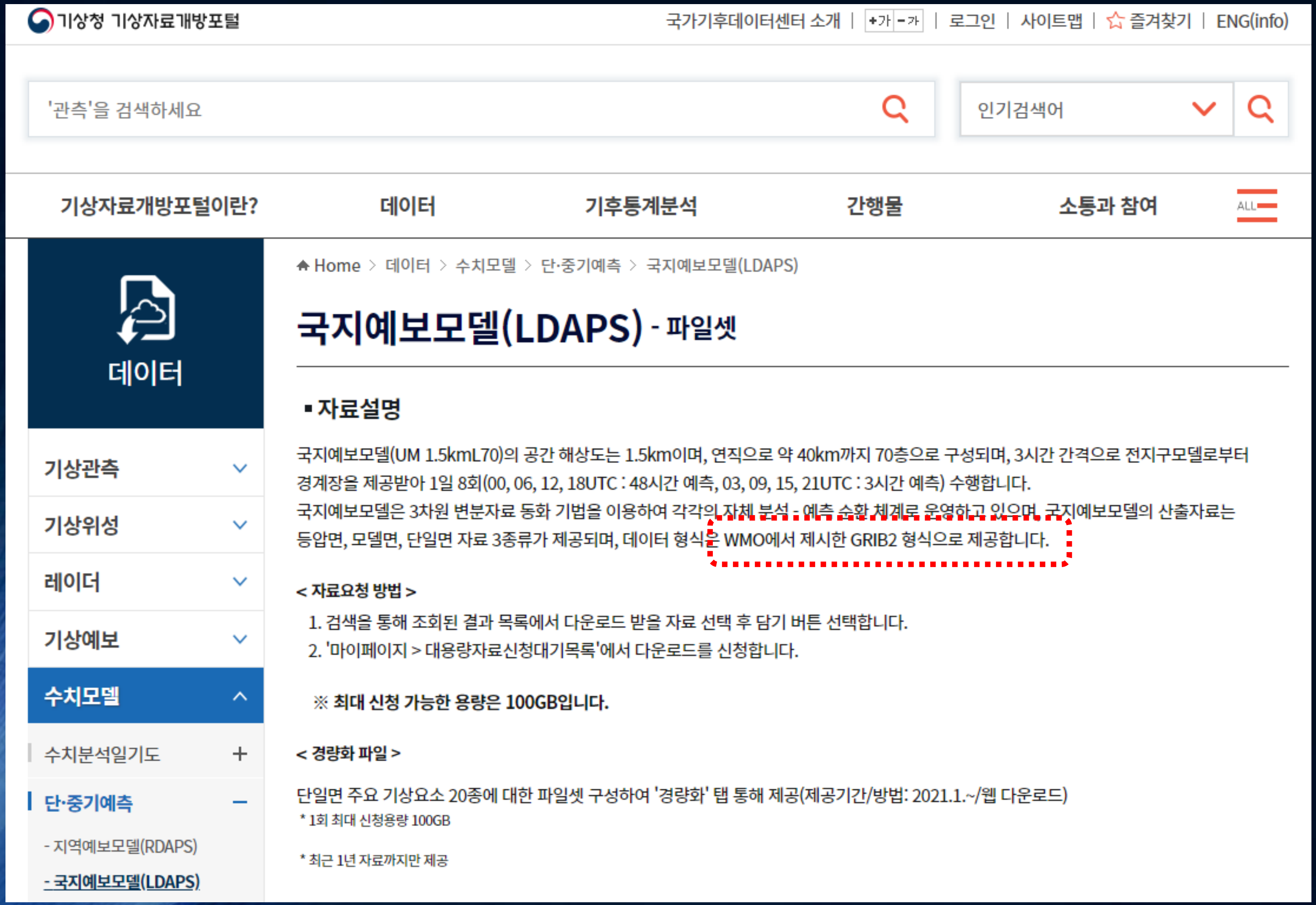
- If you prefer a graphical tool, click on above.
- To subset data in time and/or space, choose MOTU.
- If you use an OPeNDAP client such as *netCDF4/xarray (Python)*, *ferret*, or *MATLAB*, choose OPeNDAP or ERDDAP.
- **Raw files:** Use FTP or the web-based File Browser.
- **Maps:** To request maps from QGIS or similar tools, use our Web Mapping Service (WMS).

Dataset	MOTU	OPeNDAP	ERDDAP	Raw files	Maps
dataset-armor-3d-nrt-weekly	Link	Link	-	FTP Browse	WMS
dataset-armor-3d-nrt-monthly	Link	Link	-	FTP Browse	WMS
dataset-armor-3d-rep-weekly	Link	Link	-	FTP Browse	WMS
dataset-armor-3d-rep-monthly	Link	Link	-	FTP Browse	WMS

https://data.marine.copernicus.eu/product/MULTIOBS_GLO_PHY_TSUV_3D_MYNRT_015_012/services

LDAPS and ETC…

- https://data.kma.go.kr/data/rmt/rmtList.do?code=340&pgmNo=65



GRIB (GRIdded Binary or General Regularly-distributed Information in Binary form[1]) is a concise data format commonly used in meteorology to store historical and forecast weather data. It is standardized by the World Meteorological Organization's Commission for Basic Systems, known under number GRIB FM 92-IX, described in WMO Manual on Codes No.306

https://en.wikipedia.org/wiki/GRIB

NOAA

Physical Sciences Laboratory

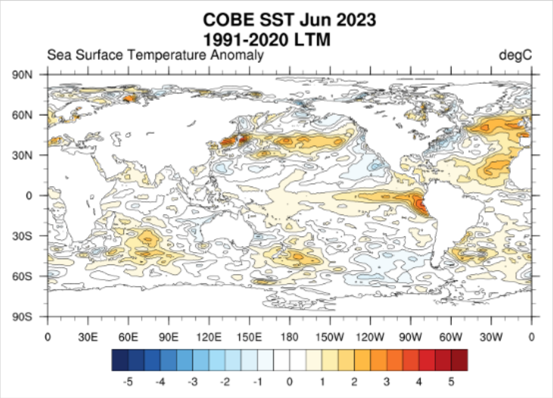
https://psl.noaa.gov/data/gridded/index.html

- https://psl.noaa.gov/data/gridded/data.cobe.html

Home » Data » Gridded Climate » COBE Sea Surface Temperature

COBE Sea Surface Temperature

Monthly 1° x1° SST dataset from 1891 to present from the JMA. Datasets uses 3D Var to fill gaps.



SPECIFICATIONS

Temporal Coverage

- Monthly Means: 1891/01 to 2023/06
- Long Term Mean: 1991-2020 (default; 1981-2010 also available)

Spatial Coverage

- 1.0° latitude x 1.0° longitude global grid (360x180)
- 89.5N - 89.5S, 0.5E - 359.5E

Levels:

- Sea Level

Update Schedule:

- Monthly

DESCRIPTION +

USAGE & CITATION +

SOURCE & REFERENCES +

VARIABLES +

Download and Plot Data

Data Help

Search Dataset Variables (start with * to match anywhere) | Clear

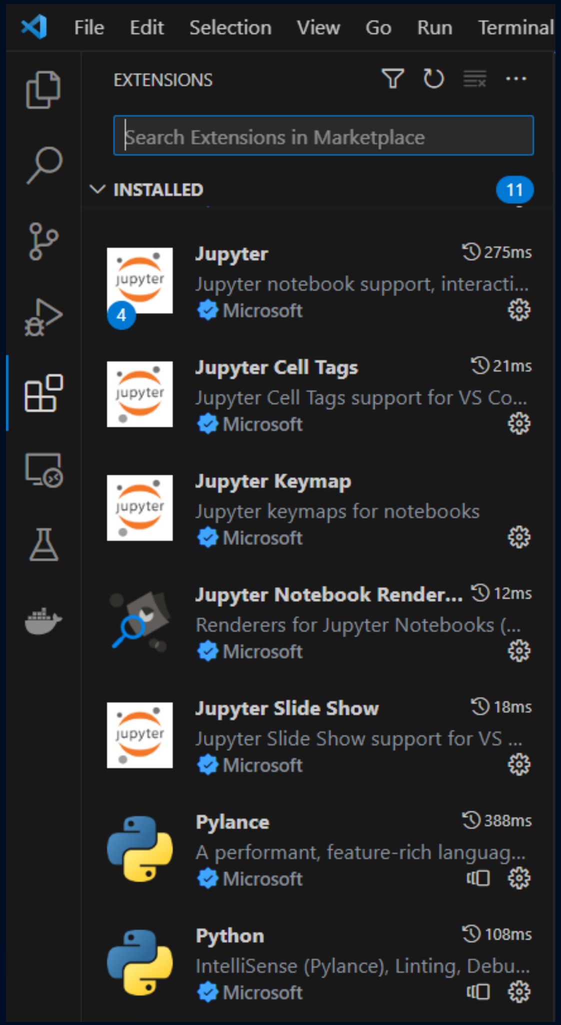
+ / -

	Variable	Statistic	Level	TimeScale	Options
	Sea Surface Temperature				
2	Sea Surface Temperature	Long Term Mean	Surface	Monthly	
1	Sea Surface Temperature	Mean	Surface	Monthly	
	/Datasets/COBE/sst.mon.mean.nc				

Thredds Catalog: Download: Plot/Subset:

sst.mon.mean.nc 160,775KB

Data Processing Environment



Loading NetCDF by using Xarray

```python
import xarray as xr  #netcdf와 같은 다차원 배열 데이터 처리용 패키지
```
✓ 0.0s

```python
ds = xr.open_dataset('sst.mon.mean.nc')
ds
```
✓ 0.0s

xarray.Dataset

- Dimensions: (**lat**: 180, **lon**: 360, **time**: 1590)
- Coordinates:

lat	(lat)	float32	89.5 88.5 87.5 ... -88.5 -89.5
lon	(lon)	float32	0.5 1.5 2.5 ... 357.5 358.5 359.5
time	(time)	datetime64[ns]	1891-01-01 ... 2023-06-01

- Data variables:

sst	(time, lat, lon)	float32	...

- Indexes: (3)
- Attributes:

title :	COBE Sea Surface Temperature Analysis
history :	created June 2011 at ESRL/PSD using the grib data from JRA.
platform :	Analyses
original_source :	http://near-goos1.jodc.go.jp/cgi-bin/1997/near_goos_catalog?projectname=NEAR-GOOS
Conventions :	CF-1.2
institution :	NOAA ESRL/PSD
comment :	recent values (w/i last year) may change as dataset is updated. It is a monitoring dataset.
dataset_title :	COBE Sea Surface Temperature
References :	https://www.psl.noaa.gov/data/gridded/data.cobe.html

- import xarray package
- loading data & View

Dimension

Z(Depth/Height) Dimension also can be added

Variables – Sea Surface Temperature

Metadata

Extract data by temporal ranges

```python
# import package for deal with datetime data
import datetime
```

✓ 0.0s

```python
# extract the data by time - recent 2 years
time_start = datetime.datetime(2022, 1, 1, 0, 0, 0)
time_end = datetime.datetime(2022, 12, 31, 23, 59, 59)
ds2 = ds.sel(time=slice(time_start, time_end))
ds2
```

✓ 0.0s

dataset.sel(time index=slice(start, end))

xarray.Dataset

► Dimensions: (**lat**: 180, **lon**: 360, **time**: 12)

▼ Coordinates:

lat	(lat)	float32	89.5 88.5 87.5 ... -88.5 -89.5
lon	(lon)	float32	0.5 1.5 2.5 ... 357.5 358.5 359.5
time	(time)	datetime64[ns]	2022-01-01 ... 2022-12-01

▼ Data variables:

sst	(time, lat, lon)	float32	...

► Indexes: (3)

► Attributes: (9)

Extract data by spatial ranges

```
# extract the data by spatial - zoom in asia
min_lon = 120
min_lat = 30
max_lon = 140
max_lat = 45
# mask the coordinates
mask_lon = (ds2.lon >= min_lon) & (ds2.lon <= max_lon)
mask_lat = (ds2.lat >= min_lat) & (ds2.lat <= max_lat)
# clip the dataset
ds3 = ds2.where(mask_lon & mask_lat, drop=True)
ds3
```

✓ 0.0s

dataset.where(coordinates range)

xarray.Dataset

► Dimensions: (**time**: 12, **lat**: 15, **lon**: 20)

▼ Coordinates:

lat	(lat)	float32	44.5 43.5 42.5 ... 32.5 31.5 30.5
lon	(lon)	float32	120.5 121.5 122.5 ... 138.5 139.5
time	(time)	datetime64[ns]	2022-01-01 ... 2022-12-01

▼ Data variables:

sst	(time, lat, lon)	float32	nan nan nan ... 21.97 22.07 22.16

► Indexes: (3)

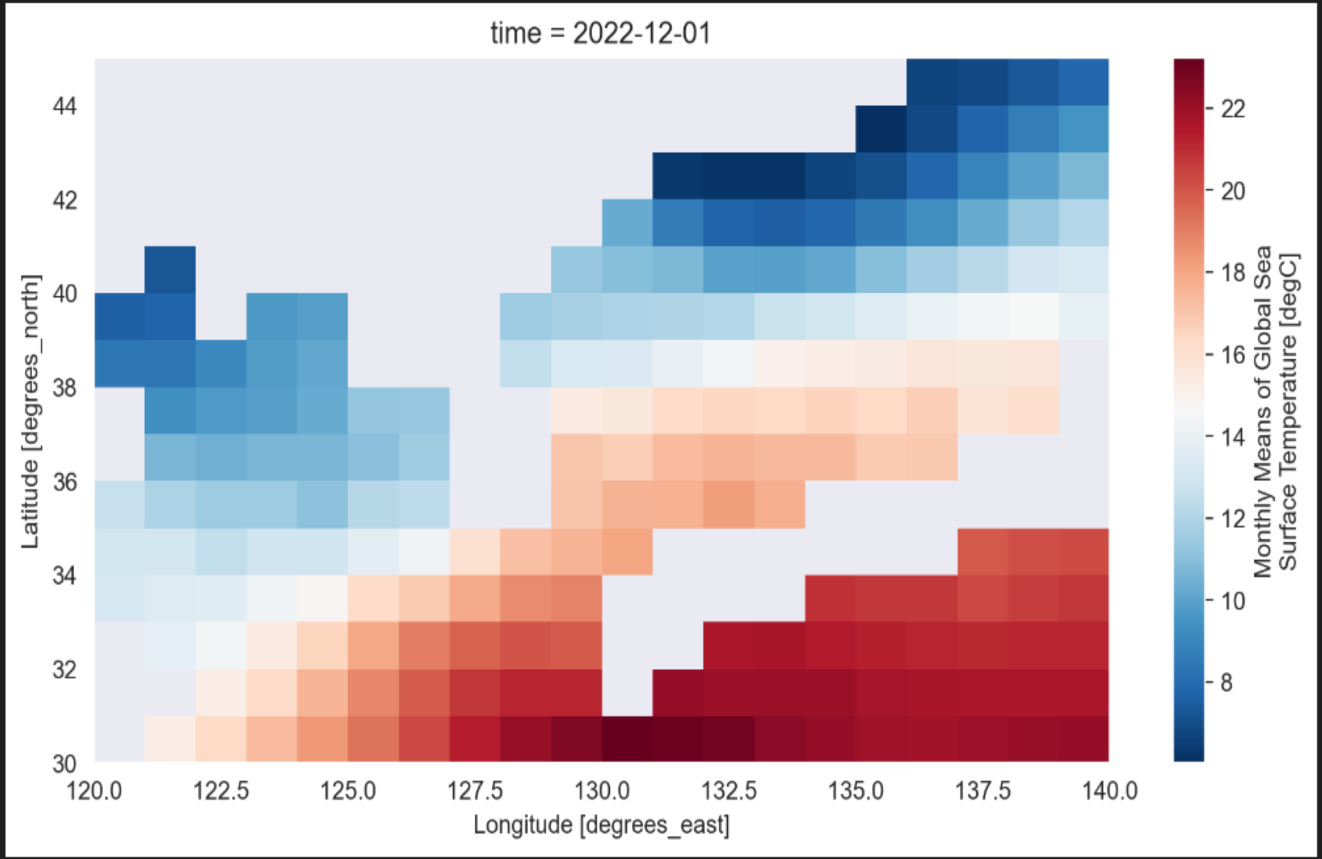
► Attributes: (9)

Plot the snapshot of dataset

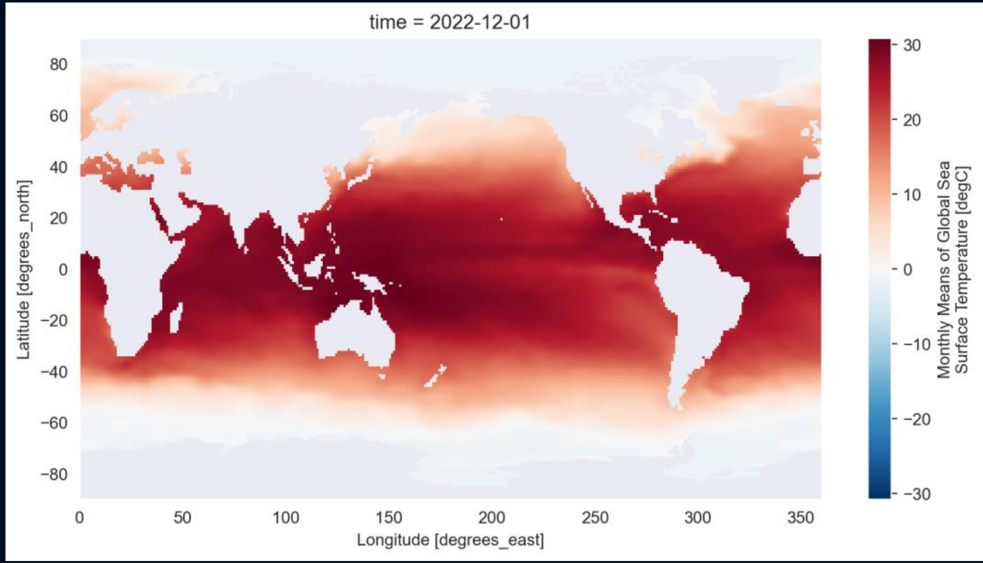
```python
# you can plot the snapshot of dataset
snapshot = datetime.datetime(2022, 12, 1, 0, 0, 0)
ds3.sel(time = snapshot).sst.plot(cmap='RdBu_r',figsize=(10,5))
```

✓ 0.5s

```
<matplotlib.collections.QuadMesh at 0x1e888380d10>
```




```python
# you can plot the snapshot of dataset
snapshot = datetime.datetime(2022, 12, 1, 0, 0, 0)
ds.sel(time = snapshot).sst.plot(cmap='RdBu_r',figsize=(10,5))
```



Now, convert NetCDF into Pandas DataFrame

- using Xarray's to_dataframe

```
# Latitude, longitude and time are indexed.
# You can convert it to a general attribute column by using reset_index.
df = ds3.to_dataframe().reset_index()
df.info()
df.tail()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3600 entries, 0 to 3599
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   time    3600 non-null   datetime64[ns]
 1   lat     3600 non-null   float32
 2   lon     3600 non-null   float32
 3   sst     2364 non-null   float32
dtypes: datetime64[ns](1), float32(3)
memory usage: 70.4 KB
```

	time	lat	lon	sst
3595	2022-12-01	30.5	135.5	21.897499
3596	2022-12-01	30.5	136.5	21.852501
3597	2022-12-01	30.5	137.5	21.965000
3598	2022-12-01	30.5	138.5	22.072500
3599	2022-12-01	30.5	139.5	22.162500

Make Point GeoDataFrame from lat, lon coordinates

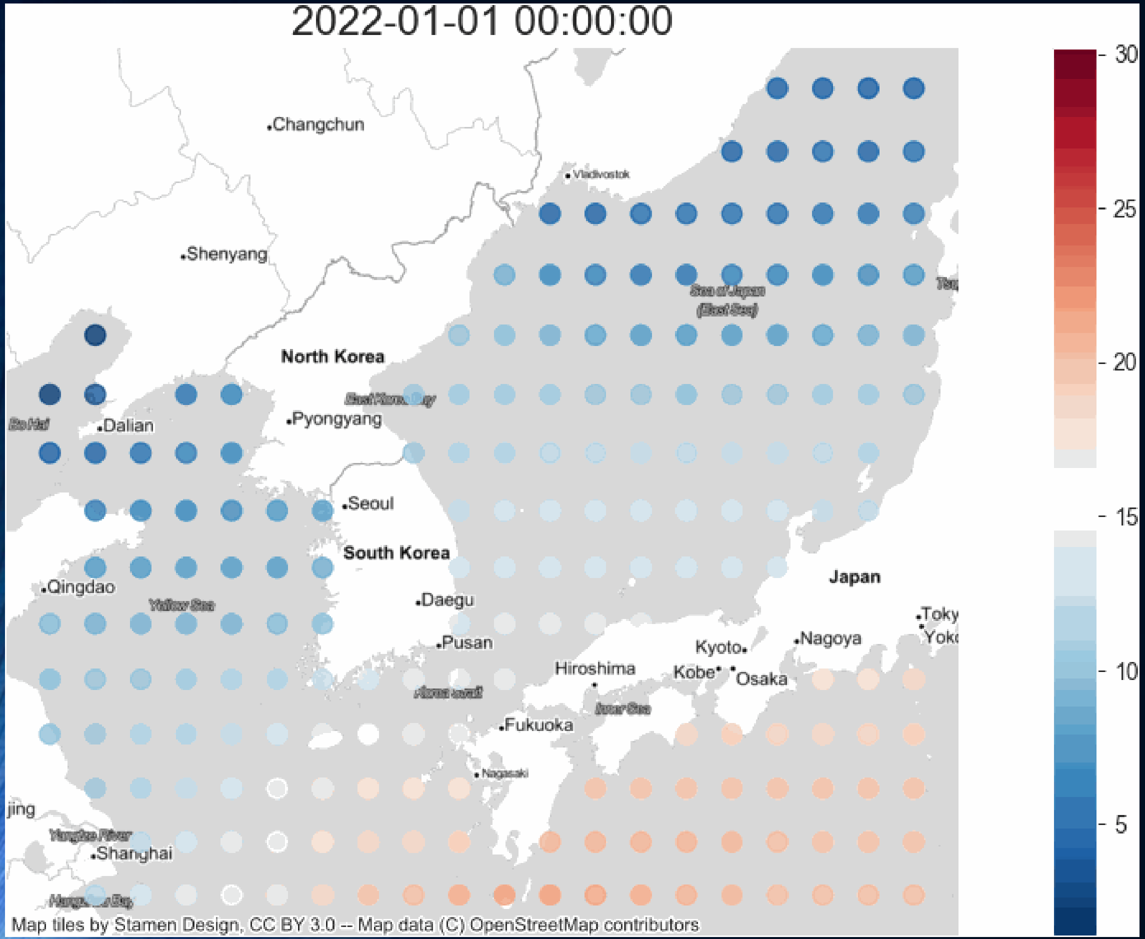
```python
# Make Point GeoDataFrame from lat, lon coordinates
# make point geometry
geom = gpd.points_from_xy(df_nnull.lon, df_nnull.lat)
gdf_pt4326 = gpd.GeoDataFrame(df_nnull, geometry=geom, crs='EPSG:4326')
gdf_pt4326.info()
gdf_pt4326.head()
```

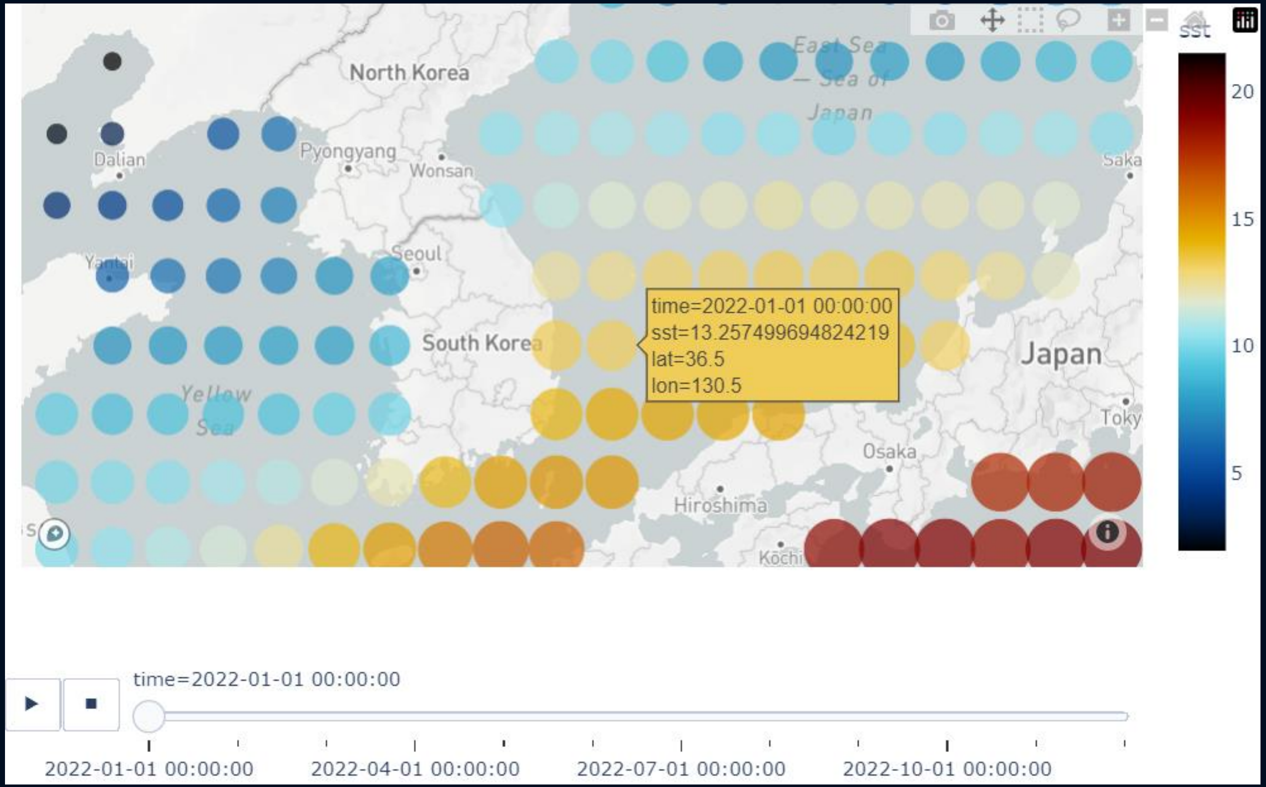
✓ 0.0s

```
<class 'geopandas.geodataframe.GeoDataFrame'>
Index: 2364 entries, 16 to 3599
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   time      2364 non-null   datetime64[ns]
 1   lat       2364 non-null   float32
 2   lon       2364 non-null   float32
 3   sst       2364 non-null   float64
 4   geometry  2364 non-null   geometry
dtypes: datetime64[ns](1), float32(2), float64(1), geometry(1)
memory usage: 92.3 KB
```

	time	lat	lon	sst	geometry
16	2022-01-01	44.5	136.5	4.140001	POINT (136.50000 44.50000)
17	2022-01-01	44.5	137.5	3.992501	POINT (137.50000 44.50000)
18	2022-01-01	44.5	138.5	3.752501	POINT (138.50000 44.50000)
19	2022-01-01	44.5	139.5	4.035001	POINT (139.50000 44.50000)
35	2022-01-01	43.5	135.5	4.190001	POINT (135.50000 43.50000)

Make Map with Point (Time Animation)





Make Polygon (Grid) Data

```python
# user function for make grid ploygon by 1 degree
def create_polygon(lat, lon):
    return Polygon([(lon, lat),
                    (lon + 1, lat),
                    (lon + 1, lat + 1),
                    (lon, lat + 1)])
```

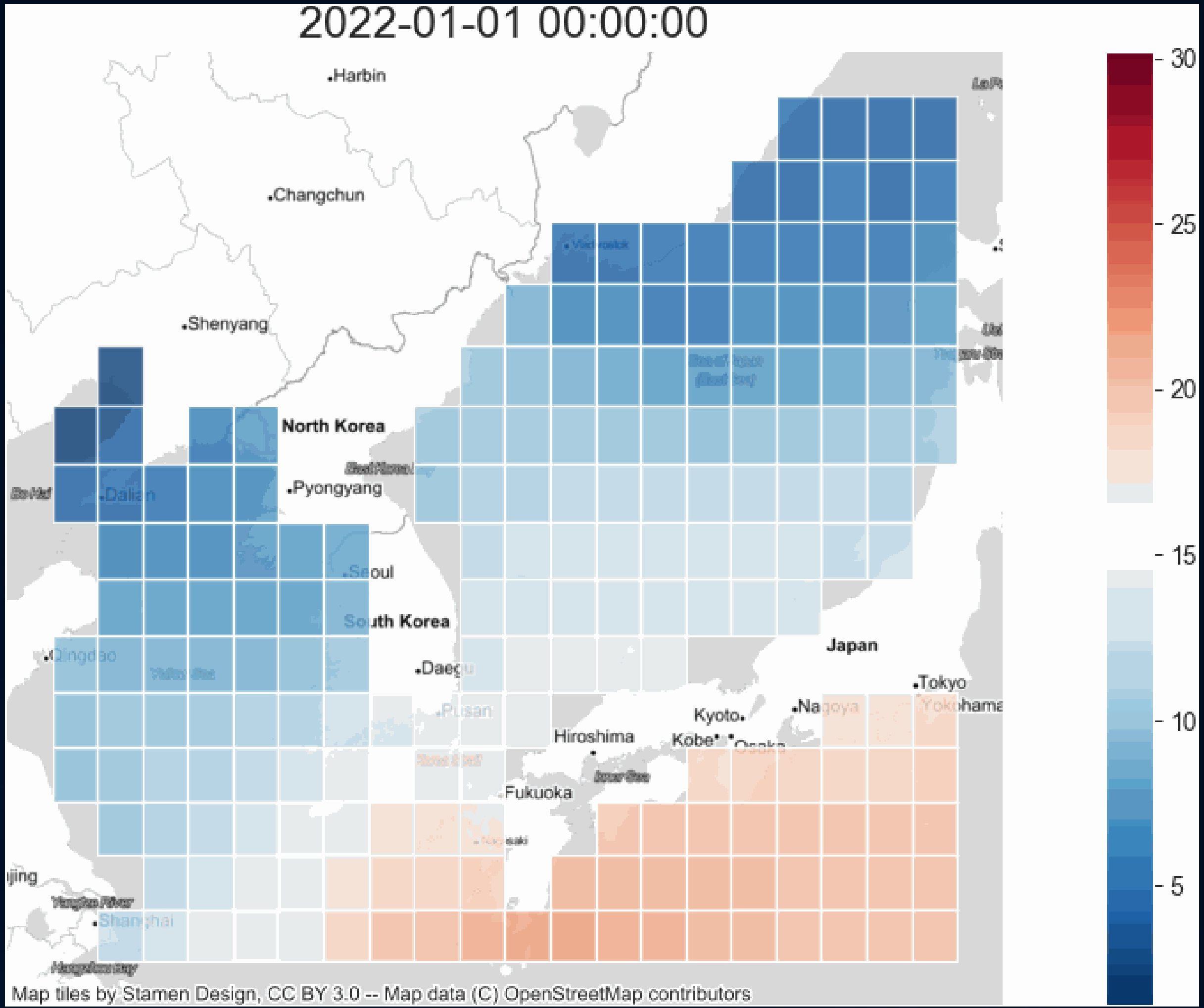
✓ 0.0s

```python
# Make polygon geometry with "create_polygon" function
geom_pg = df.apply(lambda row: create_polygon(row['lat'], row['lon']), axis=1)

# Make Polygon GeoDataFrame with DataFrame & Polygon geometry
gdf_pg4326 = gpd.GeoDataFrame(df, geometry=geom_pg, crs='4326')
```

✓ 0.2s

Make Map with Polygon Grid



Save GIS File

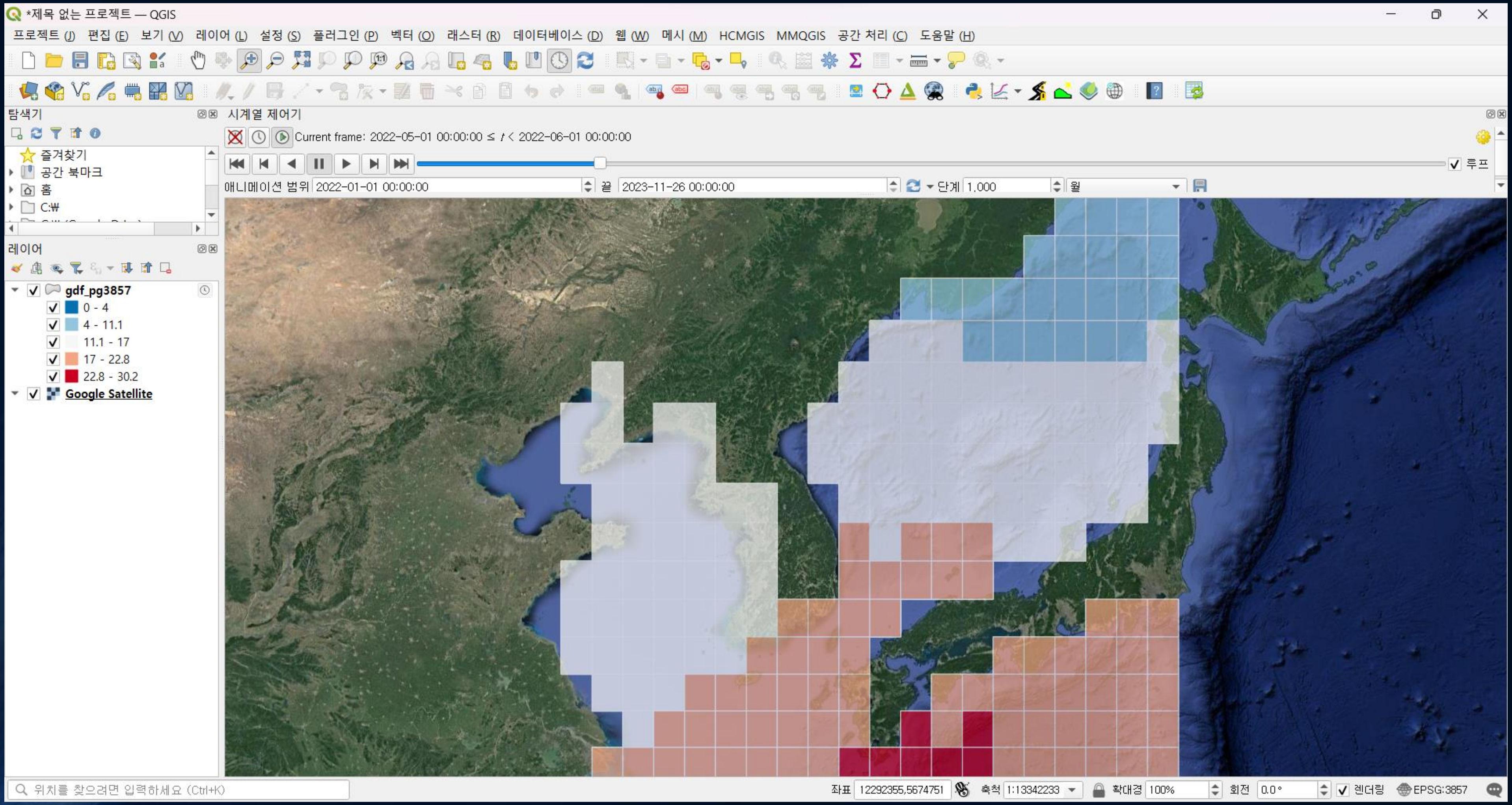
- Using GeoPandas

```python
# save shp file
gdf_pt3857.to_file('shp/gdf_pt3857.gpkg', driver="GPKG")
gdf_pg3857.to_file('shp/gdf_pg3857.gpkg', driver="GPKG")
```

✓ 1.1s

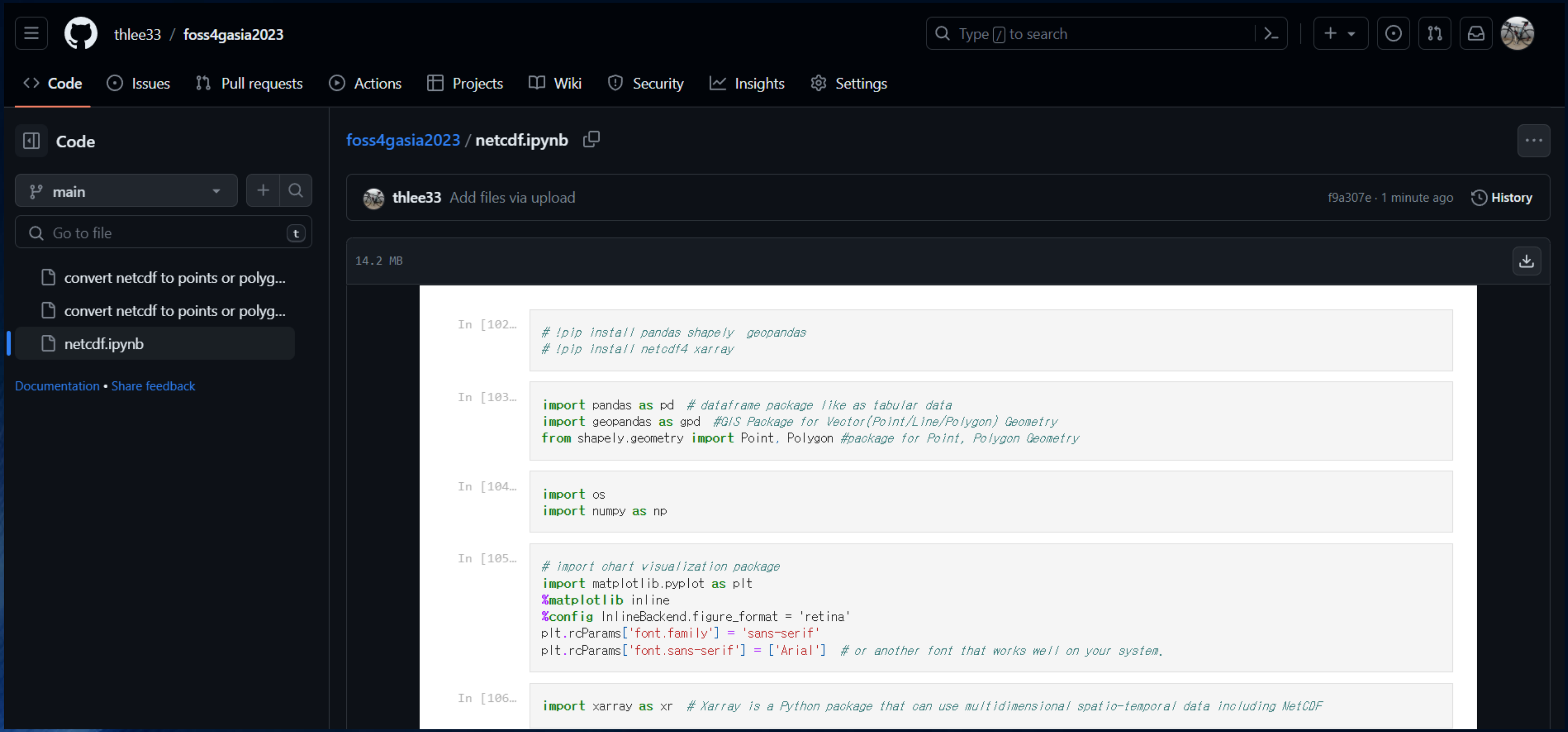
gdf_pg3857.gpkg	944KB
gdf_pt3857.gpkg	416KB

Using GeoPackage in QGIS or ETC



Get Jupyter Notebook

- https://github.com/thlee33/foss4gasia2023



Thank You

Dong-Hoon, Lee (이동훈)
dhl@gaia3d.com