

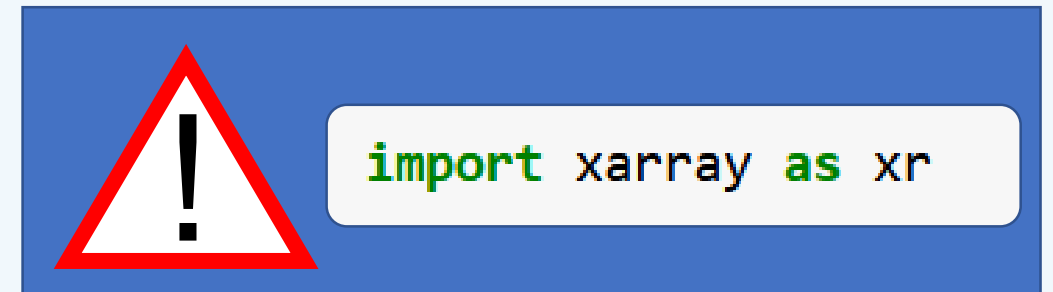
xarray

Introduction à xarray

Traitement de données sur grilles

Plan de la présentation

- Présentation et gestion des objets
 - `Xr.DataArray`, `xr.Dataset`
- Premiers traitements
- Création et gestion des figures
- Mise à l'échelle des calculs avec `dask`



Présentation des objets

xr.DataArray et xr.Dataset

Comparaison à pandas

Objet multivariées, alignées sur les mêmes axes

- **pd.DataFrame**
- **xr.Dataset**

Objet à variable unique:

- **pd.Series**
- **xr.DataArray**

Objet à variable unique:

- **pd.Series**
- **xr.DataArray**

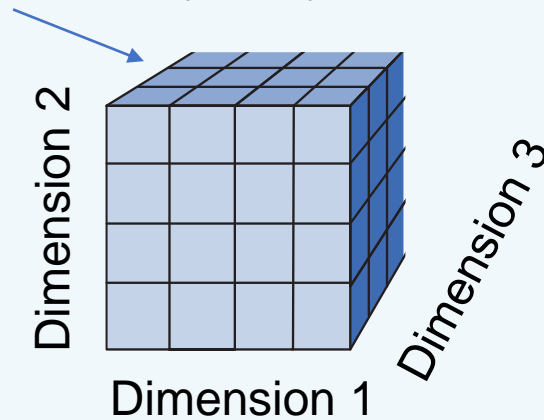
Objet à variable unique:

- **pd.Series**
- **xr.DataArray**

xr.DataArray : présentation

- Tableau de valeurs d'une variable unique:
 - emballage de numpy array

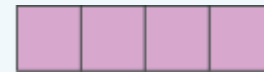
Tableau de valeurs: numpy array



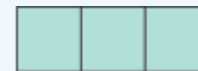
Coordonnées 1



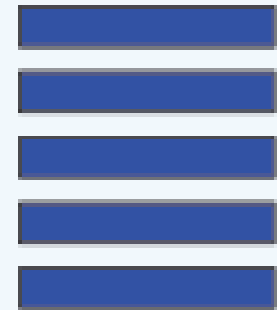
Coordonnées 2



Coordonnées 3




Attributs: méta données
supplémentaire



xr.DataArray : présentation

- Exemple d'un dataarray

```
xr.DataArray 'thetao' (time: 312, depth: 13, latitude: 157, longitude: 265)
```

 [168749880 values with dtype=float32]

▼ Coordinates:

depth	(depth)	float32	13.47 15.81 18.5 ... 77.85 92.33		
latitude	(latitude)	float32	-5.0 -4.917 -4.833 ... 7.917 8.0		
time	(time)	datetime64[ns]	1993-01-16T12:00:00 ... 2018-12-...		
longitude	(longitude)	float32	-180.0 -179.9 ... -158.1 -158.0		

▼ Attributes:

long_name : Temperature
standard_name : sea_water_potential_temperature
units : degrees_C
unit_long : Degrees Celsius
cell_methods : area: mean
_ChunkSizes : [1 7 341 720]

xr.DataArray: Création d'un objet

- Source: un tableau numpy a n dimension + n listes de coordonnées

```
data = np.array([[1, 2, 3],  
                 [5, 6, 7]])  
longitude = [10, 15, 20]  
latitude = [0, 5]
```

- Utiliser **xr.DataArray()**

```
dataarray = xr.DataArray(data,  
                          dims = ['latitude', 'longitude'],  
                          coords = {'longitude': longitude,  
                                    'latitude': latitude  
                                    },  
                          )
```

xarray.DataArray (latitude: 2, longitude: 3)

array([[1, 2, 3],
 [5, 6, 7]])

▼ Coordinates:

longitude	(longitude)	int32	10 15 20
latitude	(latitude)	int32	0 5

► Attributes: (0)

xr.DataArray: Ajouter des attributs

- `dataarray.attrs` est un dictionnaire que l'on peut modifier

```
dataarray.attrs['units'] = '°C'  
dataarray.attrs['description'] = "Température de l'OMP"
```

```
xarray.DataArray (latitude: 2, longitude: 3)  
  
array([[1, 2, 3],  
       [5, 6, 7]])  
  
▼ Coordinates:  
  
longitude (longitude) int32 10 15 20  
latitude (latitude) int32 0 5  
  
▼ Attributes:  
  
units : °C  
description : Température de l'OMP
```


xr.DataArray: Accéder aux valeurs

- **Accéder par les coordonnées:
la base de xarray**

```
dataarray.sel(longitude = 15)
```

xarray.DataArray (latitude: 2)

array([2, 6])

▼ Coordinates:

longitude	()	int32	15
latitude	(latitude)	int32	0 5

- Fonctionne aussi en liste et pour plusieurs coordonnées

```
dataarray.sel(longitude = [15, 10], latitude = 5)
```

- Sélection par position

```
dataarray.isel(longitude = [0, 2])
```

- Accéder au tableau de valeurs

















```
dataarray.values
```

xr.Dataset: présentation

- Une collection de DataArrays avec des dimensions communes

xr.DataArray

Attributs globaux

xarray.Dataset				
► Dimensions: (depth: 13, latitude: 157, time: 312, longitude: 265)				
▼ Coordinates:				
depth	(depth)	float32	13.47 15.81 18.5 ... 77.85 9...	 
latitude	(latitude)	float32	-5.0 -4.917 -4.833 ... 7.917 8.0	 
time	(time)	datetime64[ns]	1993-01-16T12:00:00 ... 201...	 
longitude	(longitude)	float32	-180.0 -179.9 ... -158.1 -158.0	 
▼ Data variables:				
vo	(time, depth, latitude, longitude)	float32	...	 
thetao	(time, depth, latitude, longitude)	float32	29.57 29.59 29.61 ... 20.22 ...	 
uo	(time, depth, latitude, longitude)	float32	...	 
thetao_profile	(time, depth)	float32	28.46 28.45 28.45 ... 28.32 ...	 
► Attributes: (17)				

Dimensions de chaque variable

xr.Dataset: Création d'un objet

- Sources:
 - Un ensemble de DataArrays
- xr.Dataset()

```
dataset = xr.Dataset({'temperature': data_temperature,  
                      'salinité': data_salinite,  
                      'precipitation': data_precipitation})
```

xr.Dataset: Accès aux données

- Accéder aux valeurs d'un dataarray

```
dataset.temperature  
dataset['temperature']
```

- Autres accès: comme un dataarray:
 - dataset.sel(latitude=...)
 - Dataset.isel(longitude=[..])

xr.Dataset: Ajouter des données

- **Ajouter une variable**
 - On ajoute une variable comme si on voulait y accéder:

```
dataset['ensoleillement'] = dataarray
```

- On peut aussi écraser la variable de cette manière

Combiner des DataArrays

- Concaténer par des dimensions existantes
 - Exemple: pas de temps, domaines géographiques différents

```
xr.concat([ds1, ds2], dim = 'latitude')
```

- Concaténer sur des nouvelles dimensions

```
xr.concat([ds1, ds2, ds3, ds4], dim = 'modele')
```

Résumé : objets xarray

Englobe des tableaux numpy avec :

- Des **dimensions** nommées (lon, lat, temps, profondeur, altitude, ...)
- Des **coordonnées** numériques
- Des **attributs**

DataArray : pour une seule variable (équivalent pd.Series)

Dataset : pour plusieurs variables sur les même grilles (équivalent pd.DataFrame)

Accès aux données par les coordonnées et le nom de la variable

```
ds.temperature.sel(lon=54, lat=[12,13,14])
```

Lire et écrire des fichiers

Format supportés

- Xarray supporte de nombreux formats en lecture et écriture
 - **netCDF**
 - Autres possibles
 - GRIB
 - geoTIFF
 - zarr
 - ...
- Optimisé pour tout ce qui est données sur grilles
- Possibilité d'ouvrir directement depuis internet

Ouvrir un fichier

- Pour ouvrir un fichier netcdf on utilise `xr.open_dataset()`

```
ds = xr.open_dataset('../data/GLORYS_ocean-temp-currents_1993-2019.nc')
```

- On peut aussi ouvrir un `DataArray` s'il n'y a qu'une variable

Sauvegarder un fichier

- Pour sauvegarder un fichier on utilise `dataset.to_format()`

```
ds.to_netcdf('path_to_file.nc')
```

Présentation du dataset exemple

Présentation des données









xarray.Dataset

► Dimensions: (depth: 13, latitude: 157, time: 312, longitude: 265)

▼ Coordinates:

depth	(depth)	float32	13.47 15.81 18.5 ... 77.85 9...		
latitude	(latitude)	float32	-5.0 -4.917 -4.833 ... 7.917 8.0		
time	(time)	datetime64[ns]	1993-01-16T12:00:00 ... 201...		
longitude	(longitude)	float32	-180.0 -179.9 ... -158.1 -158.0		

▼ Data variables:

vo	(time, depth, latitude, longitude)	float32	...		
thetao	(time, depth, latitude, longitude)	float32	29.57 29.59 29.61 ... 20.22 ...		
uo	(time, depth, latitude, longitude)	float32	...		
thetao_profile	(time, depth)	float32	28.46 28.45 28.45 ... 28.32 ...		

► Attributes: (17)

Premières analyses

Premières analyses

Sélections de données

Sélection de données

- Sélection par coordonnées

```
ds.sel(latitude=-4.75)
```

- Sélection par position

```
ds.isel(depth=5)
```

- Sélection unique

```
ds.sel(latitude=-4.75)
```

- Sélection par liste

```
ds.isel(depth=[5,8])
```

```
ds.isel(longitude = np.arange(3,9))
```

- Sélection multicritère

```
ds.isel(depth=[5,8], longitude = 12)
```


Sélection de données

- Plage de données par coordonnées: **slice(début, fin)**
 - Prend toutes les valeurs comprises entre début et fin

```
ds.sel(latitude=slice(-3,3))
```

- Sélection temporelle:

- Sélection exacte

```
ds.sel(time='1993-01-16')
```

- Sélection d'une période

```
ds.sel(time='2015')
```

- Sélection d'une plage de données

```
ds.sel(time=slice('2001', '2015-04'))
```

Sélection de données

- Masque des données avec where
 - Choisit les valeurs respectant une condition.
 - La condition est un dataarray de booléen (True/False) sur les mêmes

dimensions

```
ds.where(ds.uo>0.1)
```

```
ds.where(ds.uo>0.1, other=999)
```

Premières analyses

Quelques opérations

Opérations sur les dimensions

- Calcul sous la forme `DataArray.operation('dimension')`

```
ds.thetao.mean('depth')
```

- Fonctionne aussi selon plusieurs dimensions:

```
ds.thetao.mean(['longitude', 'latitude'])
```

- Idem pour les dataset

```
ds.mean(['longitude', 'latitude'])
```

- Globalement les mêmes opérations qu'avec pandas:

- `mean()`, `sum()`, `min()`, `max()`, `median()`
- `idxmax()`, `idxmin()`, `argmin()`, `argmax()`
- `quantile([q1,q2, ...])`
- `count()` → valeurs non nan

Quelques opérations avancées

- Différence d'un pas à l'autre

```
ds.diff('time')
```

- Somme cumulative

```
ds.cumsum('time')
```

- Gradient, intégrale

```
ds.differentiate('time')  
ds.integrate('time')
```

« fitter » un DataArray ou un Dataset

- On peut réaliser directement des fits polynomiaux ou des fits sur des fonctions personnalisées comme avec scipy/numpy

```
fit = ds.isel(depth=0).polyfit("time", deg=1)
```

xarray.Dataset

► Dimensions: (degree: 2, latitude: 157, longitude: 265)

▼ Coordinates:

degree	(degree)	int32	1 0	
latitude	(latitude)	float64	-5.0 -4.917 -4.833 ... 7.917 8.0	
longitude	(longitude)	float64	-180.0 -179.9 ... -158.1 -158.0	

▼ Data variables:

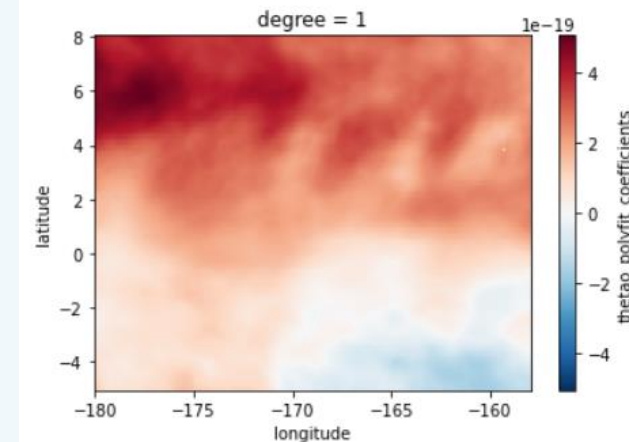
vo_polyfit_coef...	(degree, latitude, longitude)	float64	-3.185e-20 -3.72e-20 ... 0.05072	
thetao_polyfit_c...	(degree, latitude, longitude)	float64	1.37e-19 1.262e-19 ... 27.78 27.78	
uo_polyfit_coef...	(degree, latitude, longitude)	float64	-5.6e-20 -5.629e-20 ... 0.105 0.106	

► Attributes: (17)

```
def f(x, a,b):  
    return a*x+b  
fit = ds.isel(depth=0, longitude=0).curvefit("time",f)
```

```
fit.thetao_polyfit_coefficients.sel(degree=1).plot()
```

<matplotlib.collections.QuadMesh at 0x1d29a8f54c0>



Opérations pondérées

- On peut faire des opérations avec des poids
- Les poids sont un dataarray sur les même dimensions
- Exemple:
 - Moyenne globale pondérée par la taille des cellules d'un modèle de

climat `ds.weighted(taille_cellule).mean(['longitude', 'latitude'])`

Premières analyses

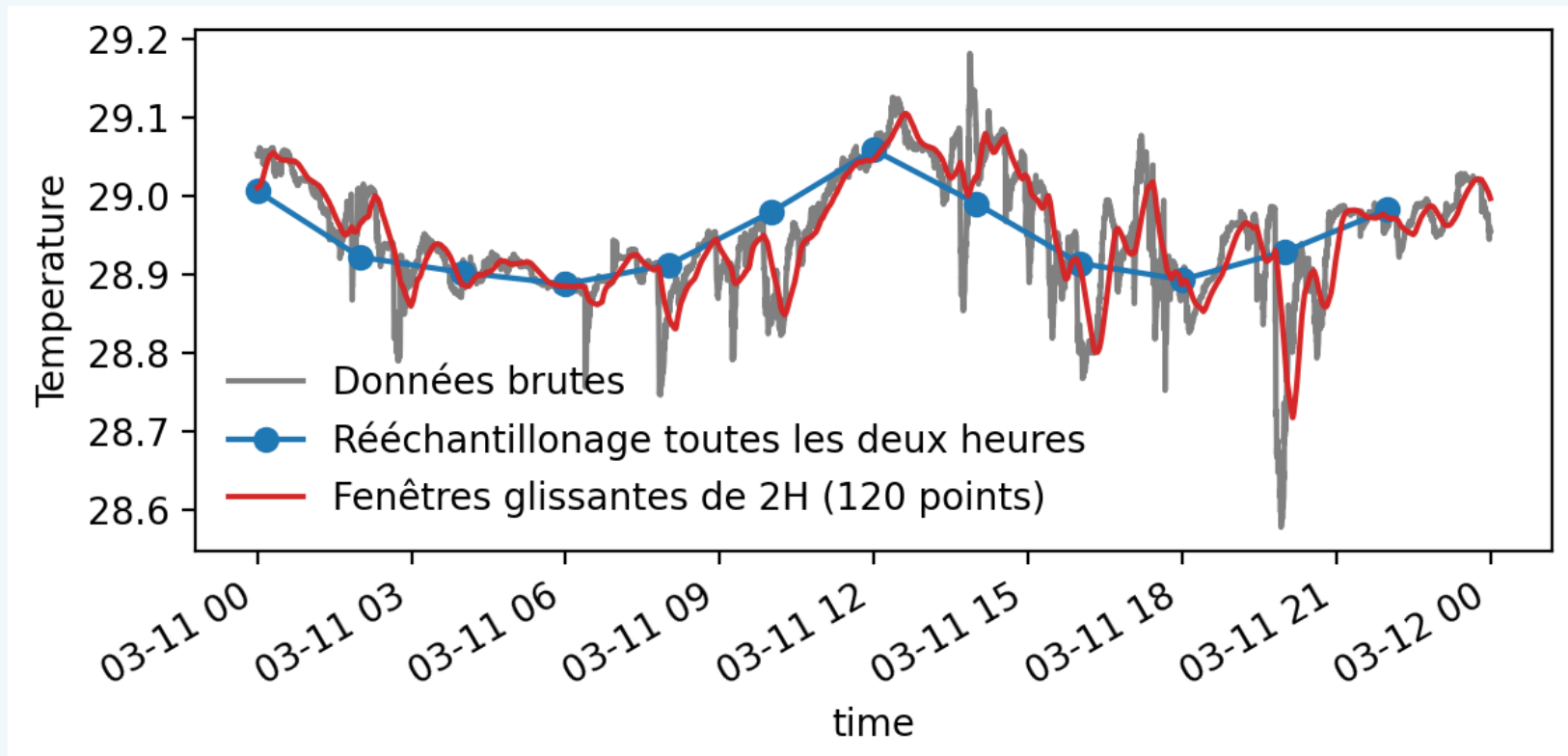
Agrégations : rééchantillonnage, fenêtres glissantes, groupby et gestion des grilles

Rééchantillonnage et fenêtres glissantes

En temporel : comme pandas

Rééchantillonnage : `ds.resample(time="2H").mean()`

Fenêtres glissantes : `ds.rolling(time=120).median()`

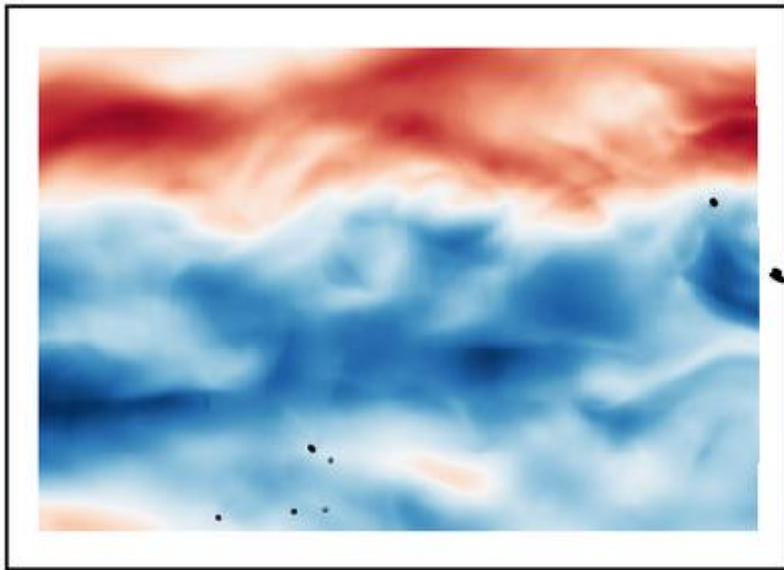


Rééchantillonnage et fenêtres glissantes

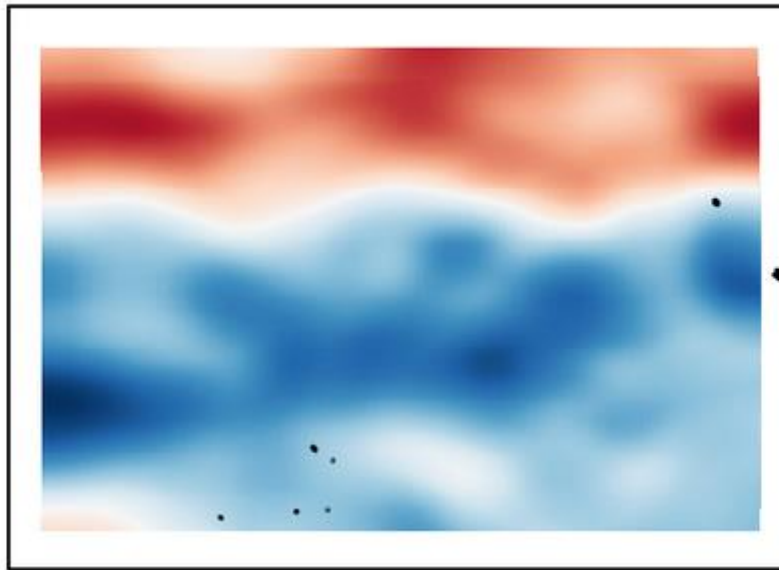
Rééchantillonnage autre dimension : `ds.coarsen(longitude=20, latitude=20, boundary='pad').mean()`

Fenêtres glissantes : comme time `ds.rolling(longitude=20, latitude=20, center=True).max()`

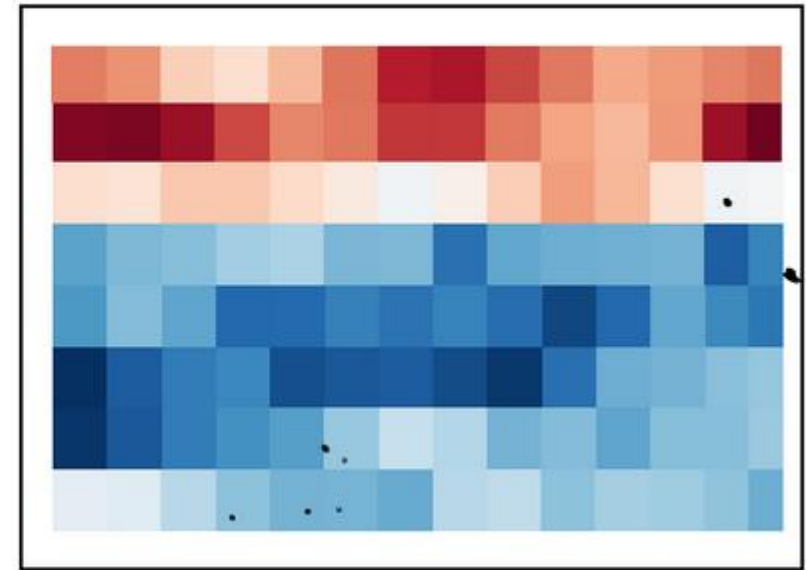
Données brutes



rolling(lat=20, lon=20)



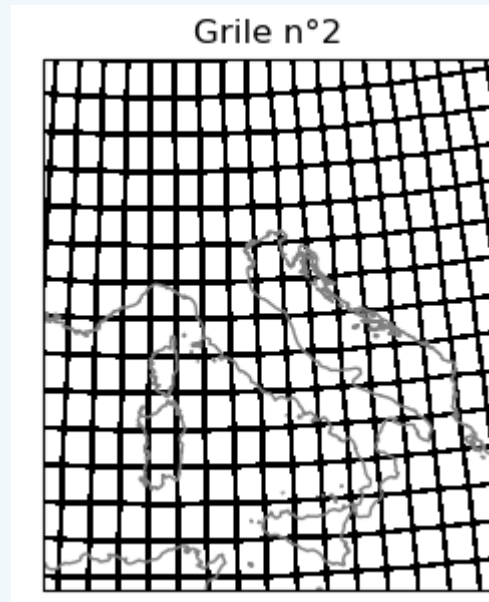
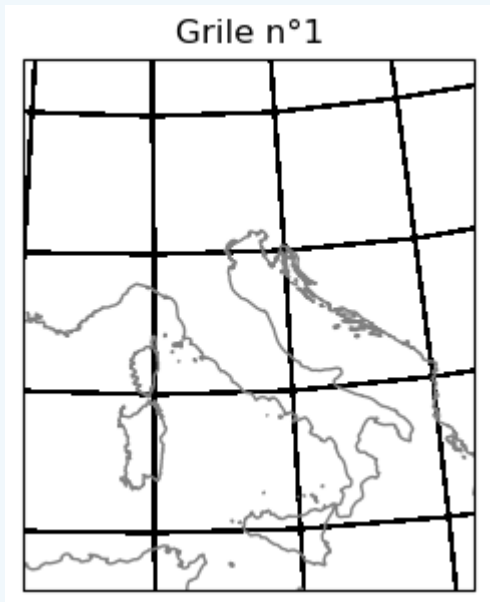
coarsen(lat=20, lon=20)



Bonus : Regrillage avec xesmf

xESMF: Universal Regridder for Geospatial Data

xESMF is a Python package for [regridding](#). It is



- **Grilles curviligne** (e.g. NEMO, modèle d'océan global)
- **Grilles rectiligne** (grille régulière lon/lat)

Interpolation des données

- Données manquantes: remplir les NaN:

```
ds.interpolate_na('latitude', method='cubic')
```

- Interpoler sur des nouvelles valeurs

```
ds.interp(latitude=np.arange(-5, 5, 0.1), method='linear')
```

Groupes de données avec groupby

- On peut réaliser des groupes comme pandas en utilisant groupby
 - On applique en suite des opérations sur les groupes

```
small_ds.groupby(small_ds.wo//0.1).mean()
```

- Groupby temporels:

```
ds.groupby('time.month')
```

Résumé : premières analyses

Sélection par les coordonnées

```
ds.sel(longitude=12, latitude=slice(0,40), time="2012")
```

Masquage par condition

```
ds.where(ds.temperature > 18)
```

Opération selon une dimension

```
ds.max("depth")
```

Opération pondérées

```
ds.weighted(taille_cellule).mean(["longitude", "latitude"])
```

Rééchantillonnage temporel

```
ds.resample(time="2H").min( )
```

Rééchantillonnage autre

```
ds.coarsen(longitude=10, latitude=5, boundary="trim").max( )
```

Fenêtres glissantes

```
ds.rolling(profondeur=5, center=True)
```

Interpolation des données

```
ds.interp(longitude = [10, 20, 30], latitude=18 )
```

Conversion en DataFrame/Series

- On peut convertir directement les objets xarray en objet pandas

```
ds.thetao.to_series()
```

```
ds.to_dataframe()
```

« One liners »

- Tout comme pandas, une commande renvoie un nouvel objet (Dataset, DataArray)
- On peut donc enchaîner les commandes sur une seule ligne

```
ds.thetao.where(ds.uo>0.1)\  
    .resample(time='Y').mean()\  
    .sel(latitude=slice(-2,2))\  
    .mean('longitude')\  
    .integrate('depth')
```


Premiers plots

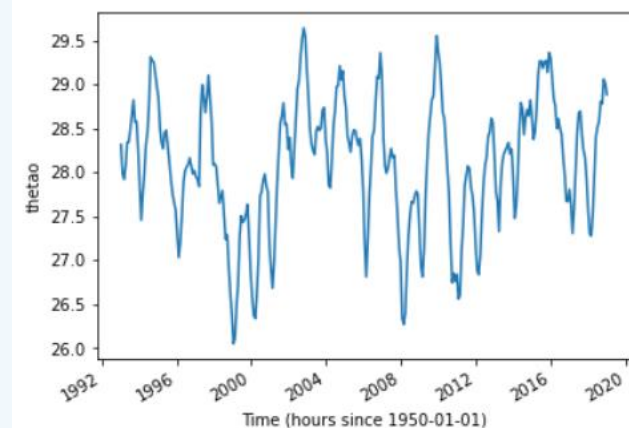
Plots de DataArrays

- Xarray est directement intégré avec matplotlib (autres backend possibles: hvplot, ...).
- Par défaut DataArray.plot() fait:

- Un line plot pour du 1d

```
ds.thetao.mean(['longitude', 'latitude', 'depth']).plot()
```

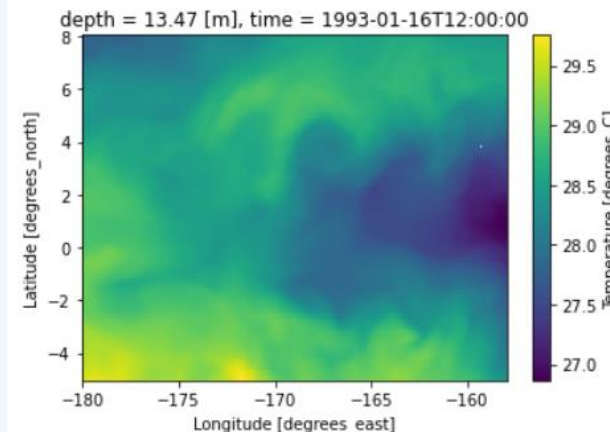
```
[<matplotlib.lines.Line2D at 0x183d4da0fa0>]
```



- Un quadmesh pour du 2D

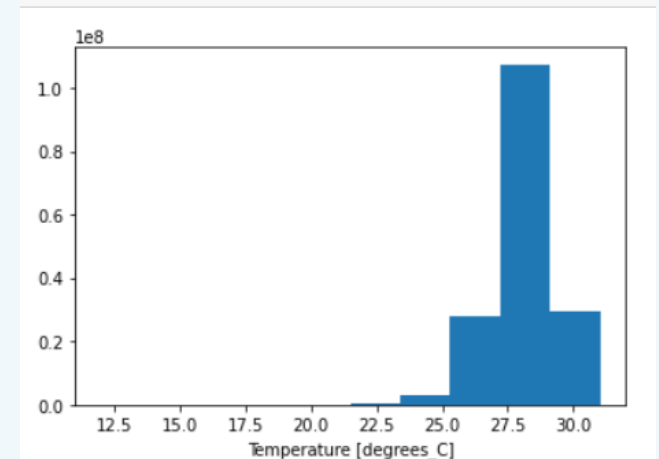
```
ds.thetao.isel(time=0, depth=0).plot()
```

```
<matplotlib.collections.QuadMesh at 0x183d4e58430>
```



- Un histogramme sinon

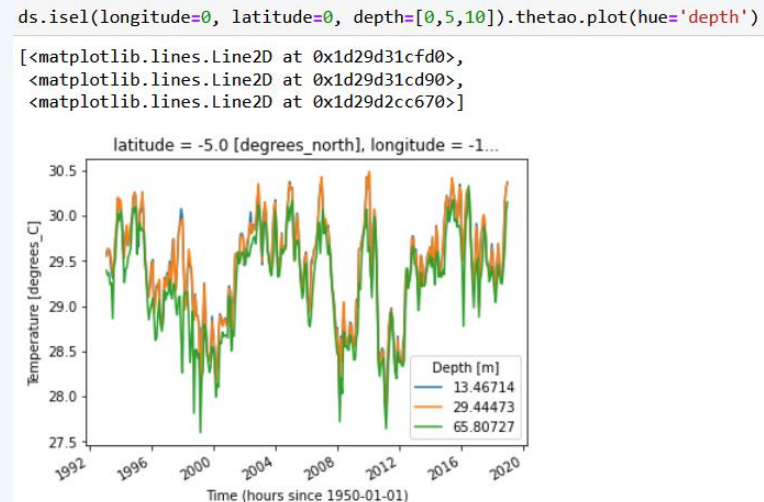
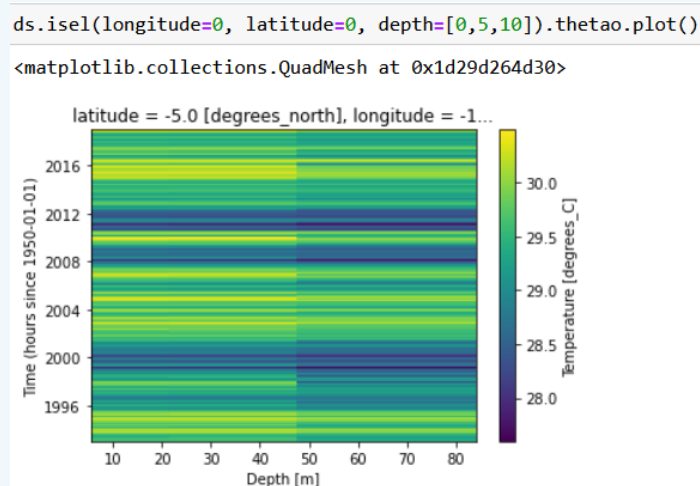
```
ds.thetao.plot();
```



- On peut en général utiliser les arguments clefs de matplotlib

Plots de DataArrays: ajouter des dimensions

- `hue=` : permet d'ajouter une dimension couleur au plot

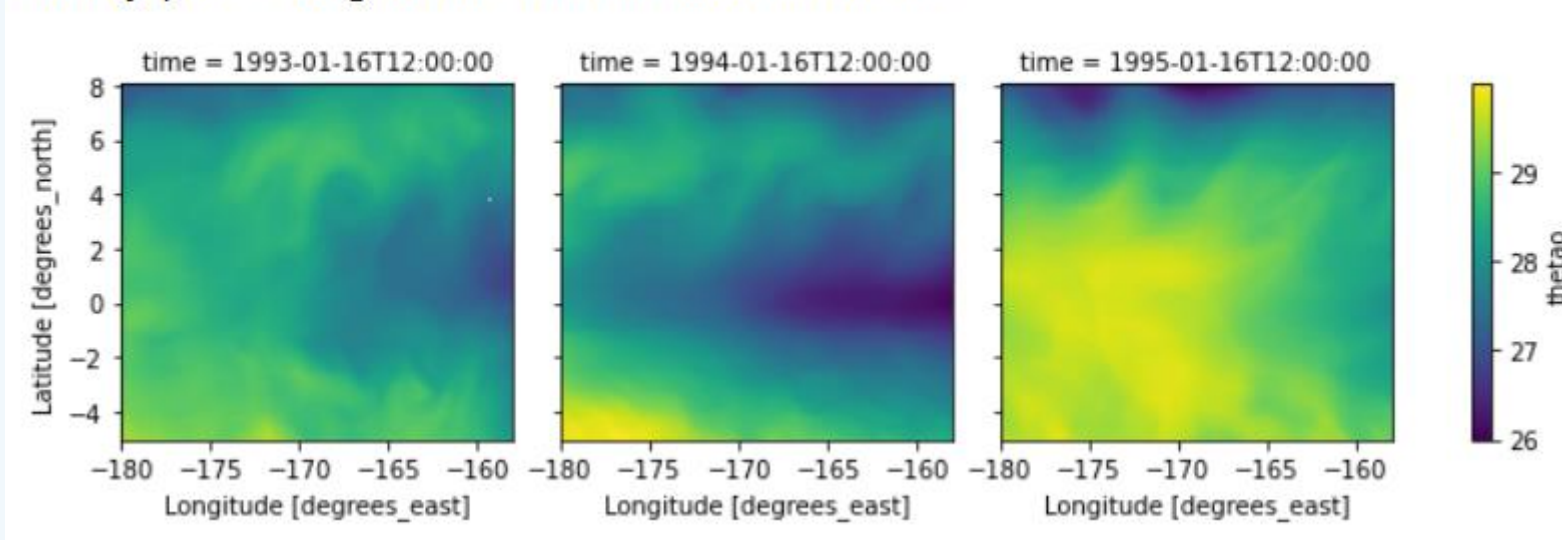


Plots de DataArrays: ajouter des dimensions

- col/row= : permet de faire des subplots automatiquement

```
ds.mean('depth').isel(time=[0,12,24]).thetao.plot(col='time')
```

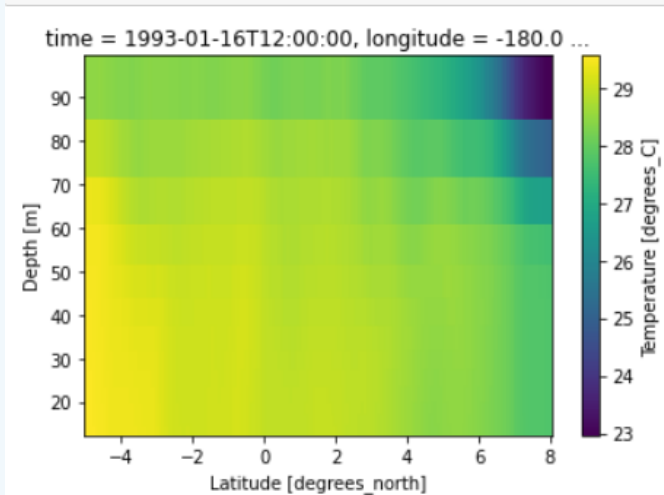
<xarray.plot.facetgrid.FacetGrid at 0x1d29d307df0>



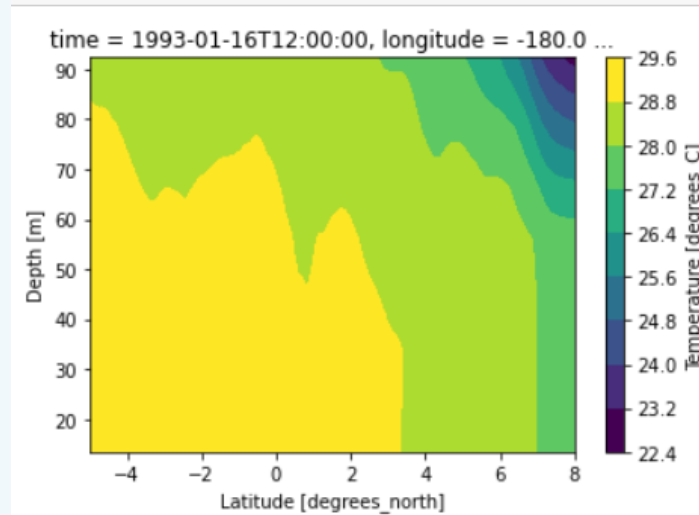
- On peut utiliser col_wrap=5 si on a beaucoup de subplots

Différents plots 2d

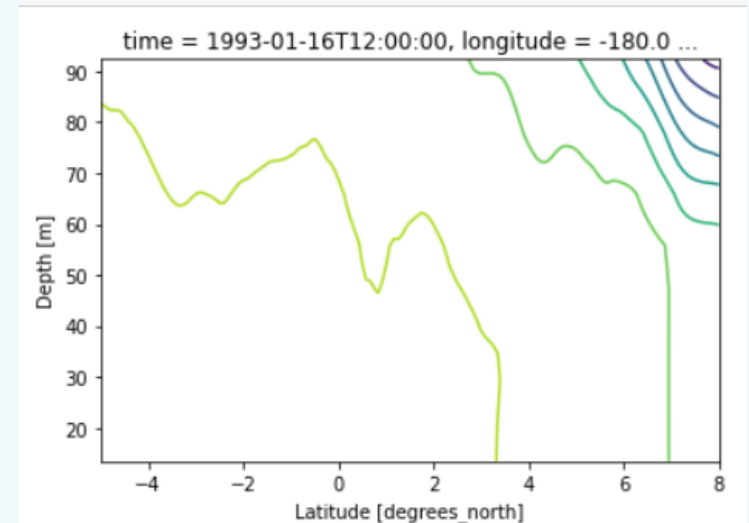
```
data.plot();
```



```
data.plot.contourf(levels=10);
```



```
data.plot.contour(levels=10);
```



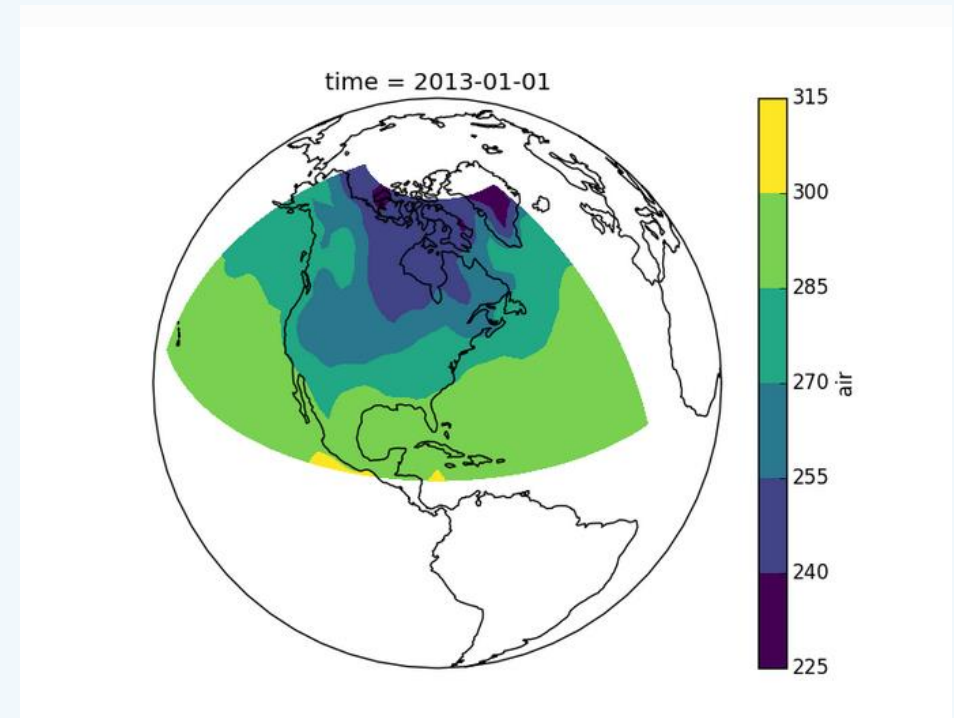
- Mots clefs importants
 - $x=$, $y=$
 - levels = liste ou nombre de niveaux, cmap=
 - add_colorbar = True/False

Intégration à cartopy

- Exemple tirée de la documentation

```
import cartopy.crs as ccrs

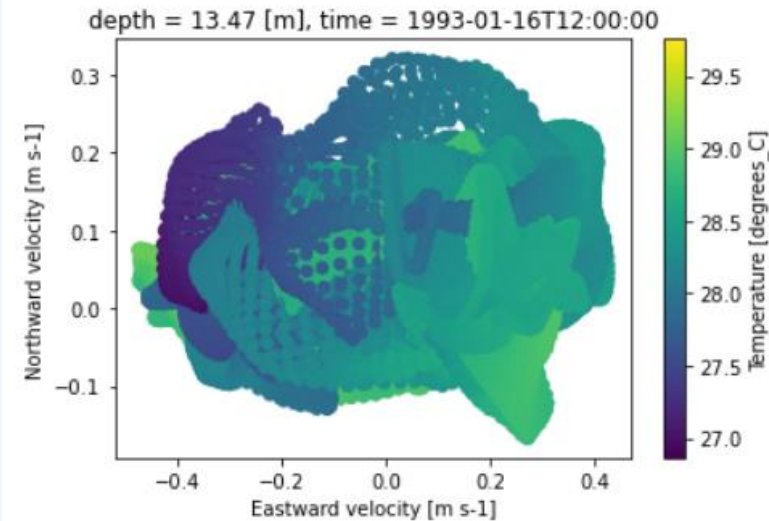
ax = plt.axes(projection=ccrs.Orthographic(-80, 35))
ax.set_global()
ds.plot.contourf(ax=ax, transform=ccrs.PlateCarree())
ax.coastlines()
```



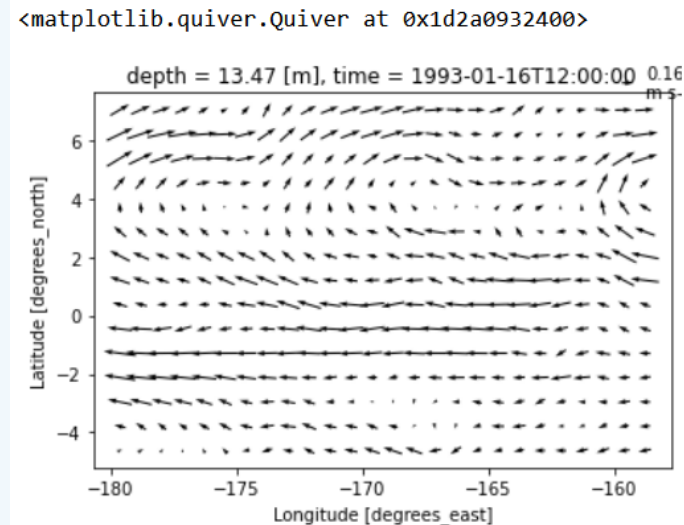
Plots avec les datasets

- On peut réaliser des plots avec les différentes variables d'un dataset

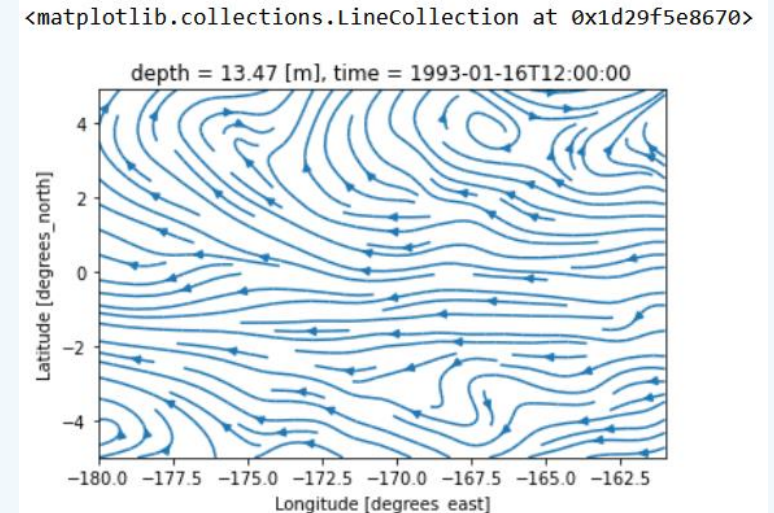
```
ds.isel(time=0, depth=0).plot.scatter(x='uo',  
                                       y='vo',  
                                       hue='thetao');
```



```
ds.isel(time=0, depth=0).plot.quiver(x='longitude',  
                                     y='latitude',  
                                     u='uo',  
                                     v='vo')
```



```
ds.isel(time=0, depth=0).plot.streamplot(x='longitude',  
                                          y='latitude',  
                                          u='uo',  
                                          v='vo')
```



Résumé : premiers plots

Intégrations de matplotlib

```
ds.temperature.plot( )
```


Pour aller plus loin

Passage à l'échelle avec dask

- On peut ouvrir tout un dossier de fichiers qui peuvent s'aligner sur une dimension

```
ds = xr.open_mfdataset('../data/temperature_files/*', concat_dim='time')
```

Voir Dask demain:

- Optimiser la mémoire
- Calcul parallèle



Tech Blog

Processing a 250 TB dataset with Coiled, Dask, and Xarray #

<https://blog.coiled.io/blog/coiled-xarray.html>

Extension de xarray

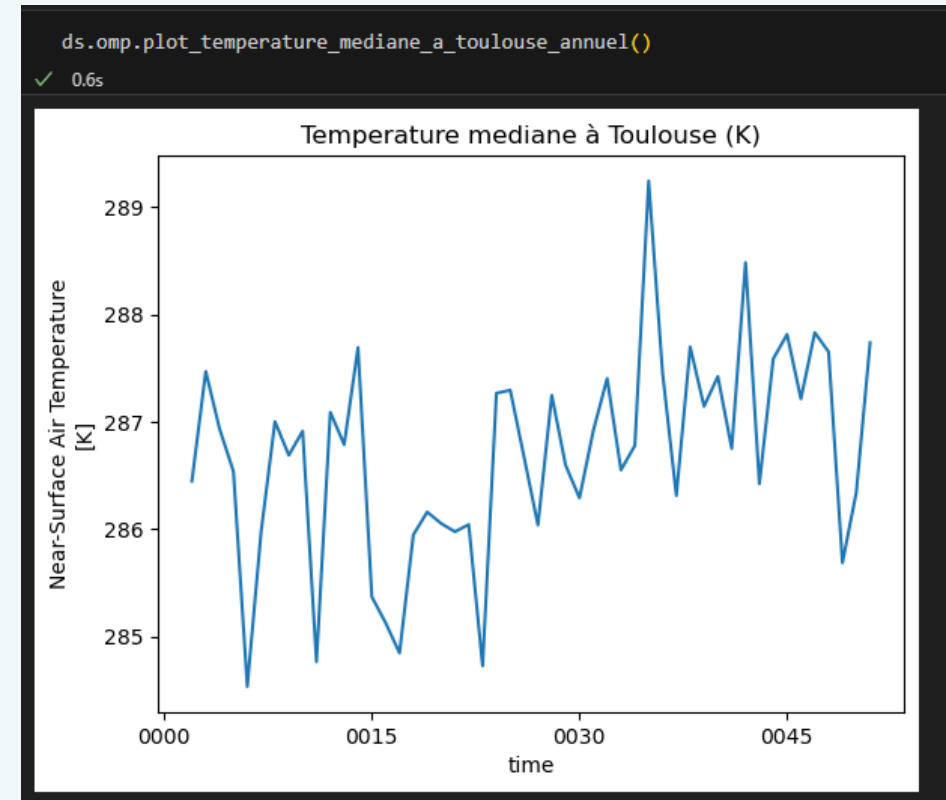
Xarray est indépendant du domaine scientifique d'application !

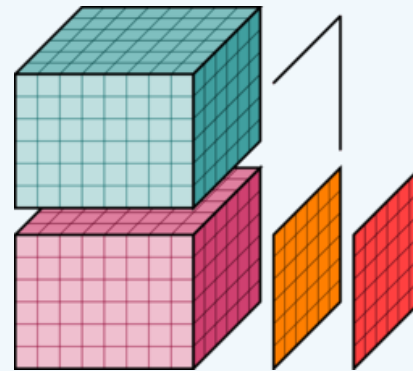
Besoin de fonctions supplémentaires?

« **Accesseur** » personnalisé

```
@xr.register_dataset_accessor("omp")
class OmpAccessor:
    def __init__(self, xarray_obj):
        self._obj = xarray_obj
        self.lon_toulouse = 1.43
        self.lat_toulouse = 43.60

    def plot_temperature_mediane_a_toulouse_annuel(self):
        import matplotlib.pyplot as plt
        """Return the geographic center point of this dataset."""
        temperature_toulouse = self._obj.tas.sel(lat=self.lat_toulouse,
                                                  lon=self.lon_toulouse,
                                                  method='nearest')
        mediane_annuelle = temperature_toulouse.resample(time='Y').median('time')
        fig, ax = plt.subplots()
        mediane_annuelle.plot(ax=ax)
        ax.set_title("Temperature mediane à Toulouse (K)")
```





xarray

Des questions?

