Science & Technology Facilities Council | UK Research and Innovation | NERC SCIENCE OF THE ENVIRONMENT

Python data analysis

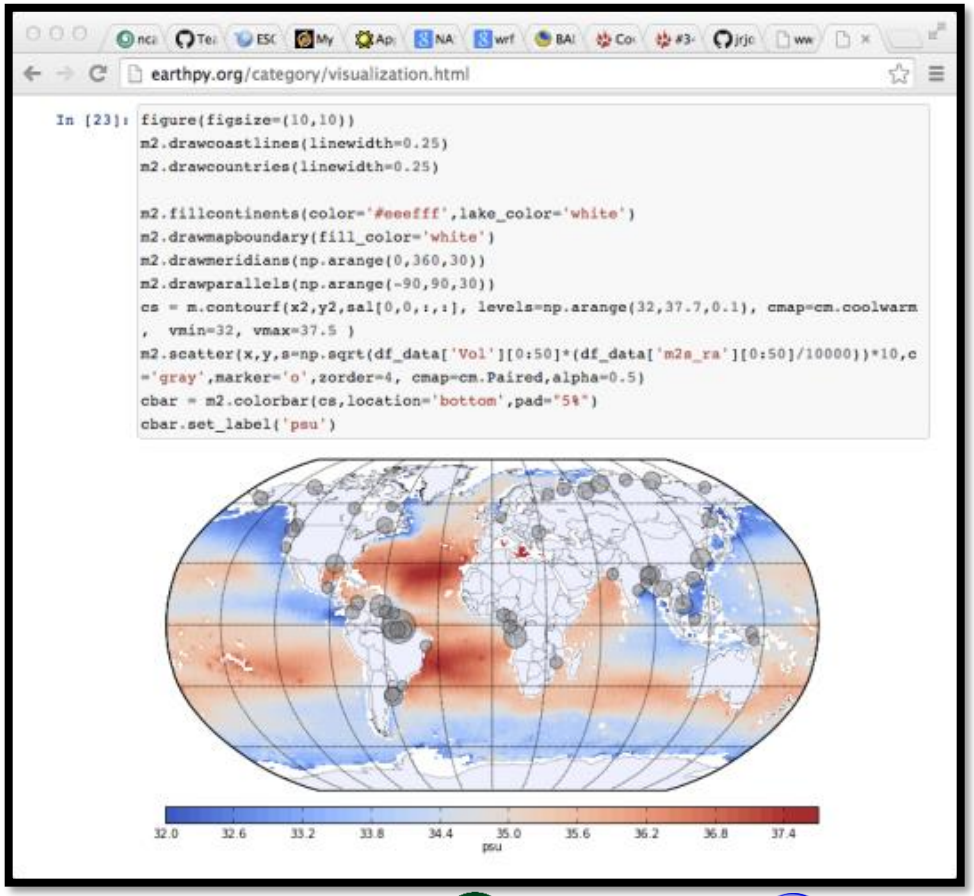
Iris

There are lots of tools

Many (Atmospheric) Science libraries are available for Python:

- `netCDF4-python`
- `cdat`
- `cf-python`
- `Iris`
- `pyNGL`

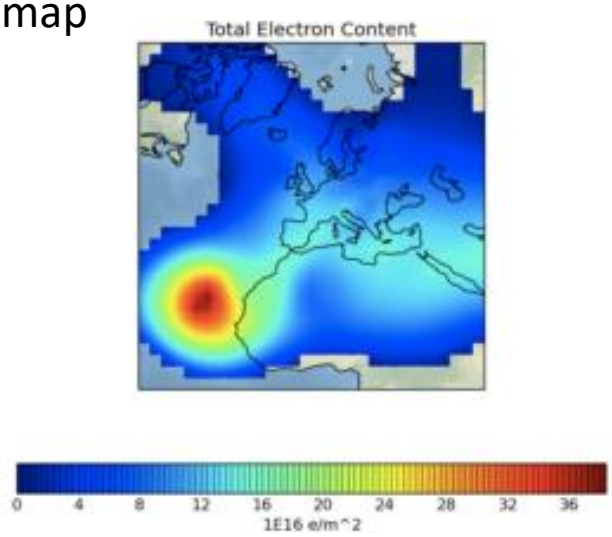
- Many others
  - `OpenClimateGIS`
  - `pyTroll`
  - …



Higher-level data tools

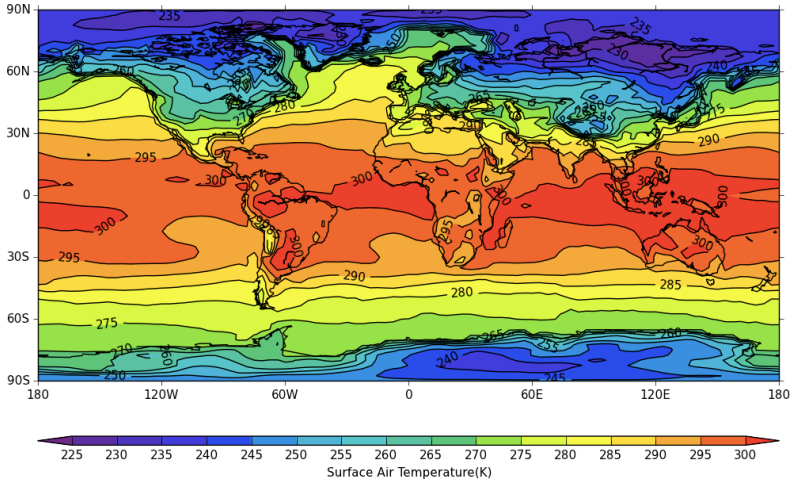
Iris

- Developed by the Met Office
- Reads NetCDF, PP and Grib
- Supports CF-conventions via the "Cube"
- Plotting via cartopy or matplotlib + basemap



cf-python

- Developed by Uni. Reading
- Reads NetCDF and PP
- Strict interpretation of the CF-conventions
- Plotting via cfplot or matplotlib + basemap



What makes these tools useful?

- The biggest gain from using these tools is that you can work with higher-level objects that know about real-world coordinate systems.

- Hence you can subset a variable based on temporal, spatial and other constraints rather than using slicing in index space.

Introducing Iris

A Python package for data analysis and visualisation

Iris

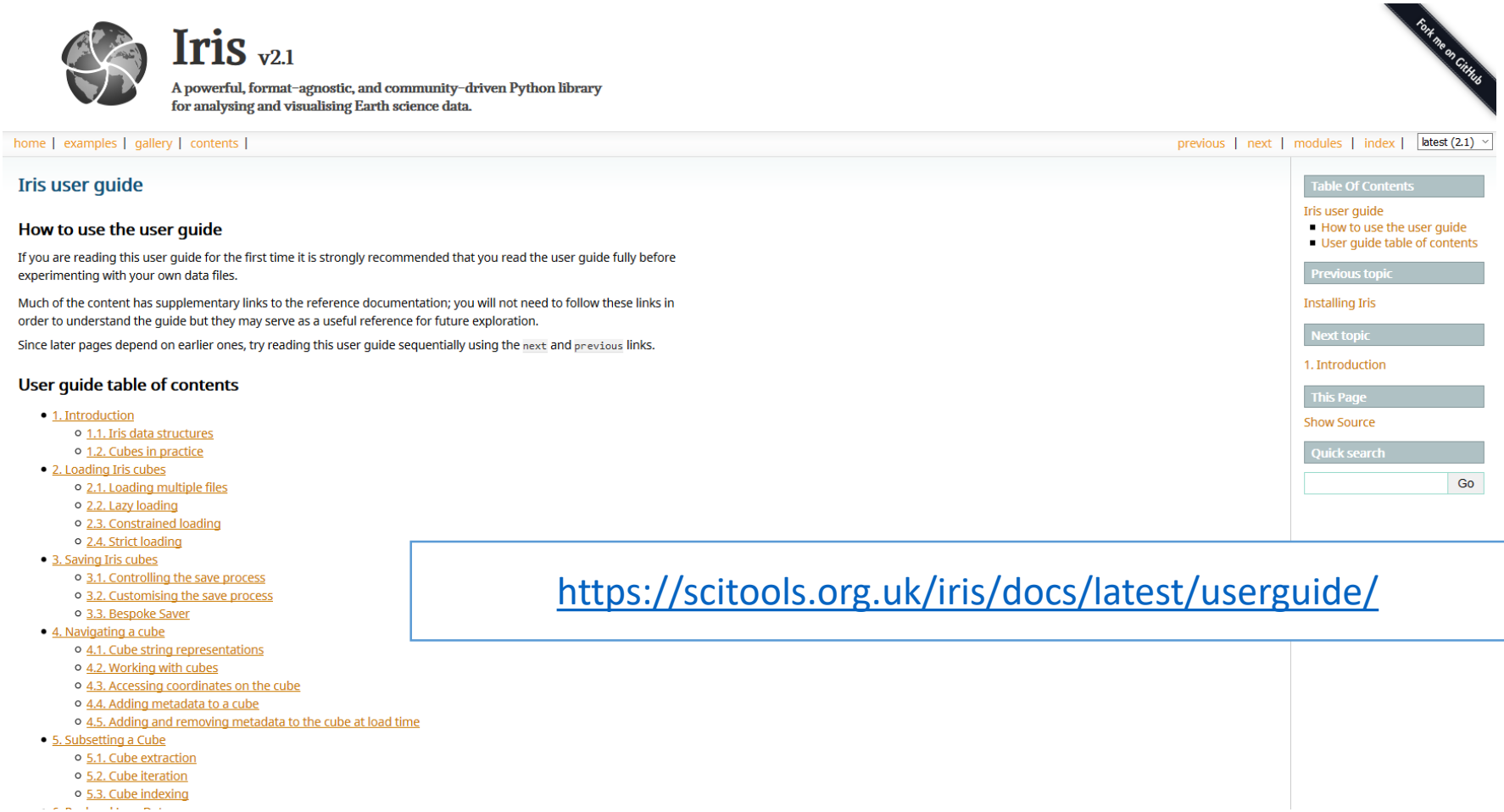
What is Iris?

Iris is publicised as:

"A Python library for Meteorology and Climatology"

- Implements the CF-netCDF Data Model in its "cube" design.
- Supports read/write access to a range of data formats (including CF-netCDF, GRIB, and PP).
- Fundamental data manipulation operations, such as arithmetic, interpolation, and statistics;
- A range of integrated plotting options.

Documentation



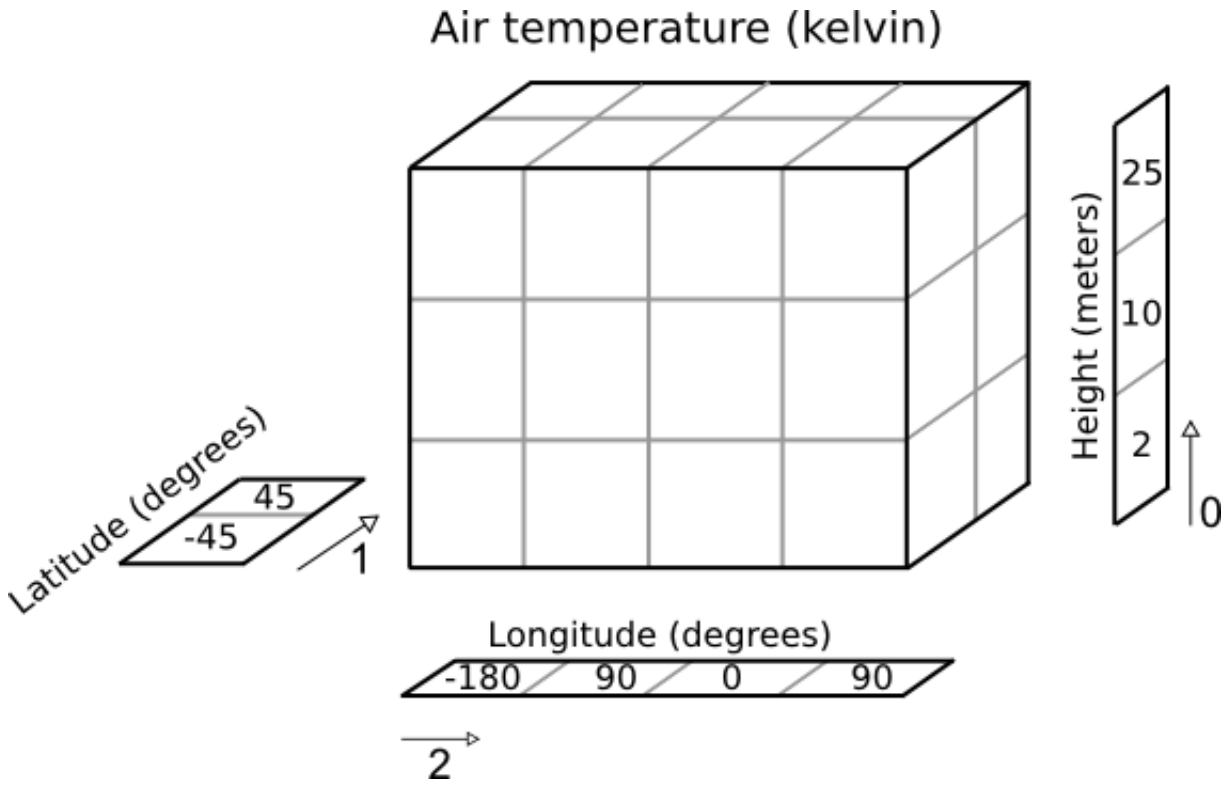
https://scitools.org.uk/iris/docs/latest/userguide/

Main concept - the "cube"

A cube consists of:

- a standard name and/or a long name and unit;
- a data array;
- a collection of coordinates and associated data dimensions on the cube's data array;
- an attributes dictionary for metadata;
- a list of cell methods (e.g. "mean over time")
- a list of coordinate "factories" used to derive coordinates from the values of other coordinates in the cube;

The "cube" - in a picture



So what can Iris do?

There are too many features to describe in detail. Here are some things that extend functionality we have seen in lower level libraries:

Loading data from multiple files:

```
import iris
filename = iris.sample_data_path('GloSea4', '*.nc')
cubes = iris.load(filename)
```

Constrained loading

Constrained by CF standard name:

```
filename = iris.sample_data_path('uk_hires.nc')
cubes = iris.load(filename,
['air_potential_temperature', 'specific_humidity'])
```

Constrained by coordinate selection:

```
filename = iris.sample_data_path('uk_hires.nc')
level_10_or_12_fp_6 = iris.Constraint(
model_level_number=[10, 16], forecast_period=6)
cubes = iris.load(filename, level_10_or_16_fp_6)
```

Cube slicing/indexing - like numpy

Cubes can be sliced and indexed like numpy arrays:

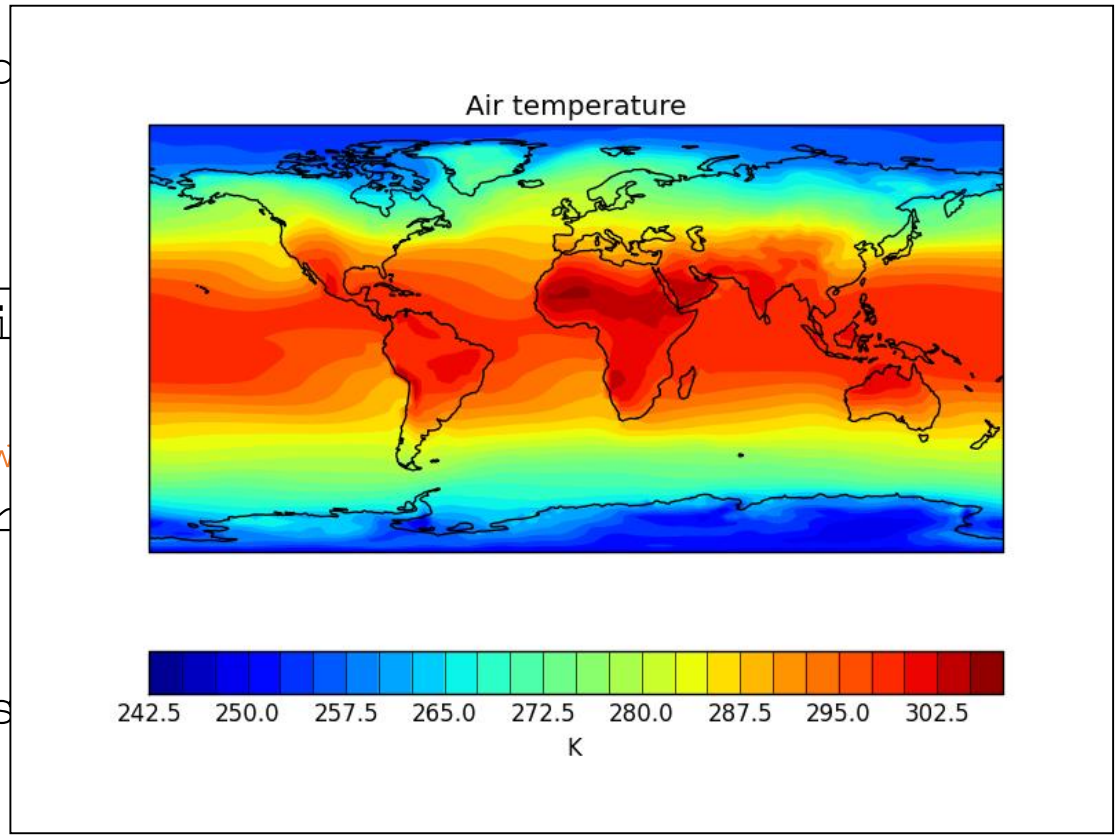
```
# get the first element of the first dimension
# (+ every other dimension)
print(cube[0])

# get the first 4 elements of the first dimension
# (+ every other dimension)
print(cube[0:4])

# Get the first element of the first and third
dimension (+ every other dimension)
print(cube[0, :, 0])
```

Plotting a cube

```
import matplotlib.pyplot as plt

import iris
import iris.quickplot as qplt

# Load the data
fname = iris.sample_data_path('air_temp.pp')
temperature_cube = iris.load_cube(fname)

# Draw the contour with 25 levels.
qplt.contourf(temperature_cube, 25)

# Add coastlines to the map created by contourf.
plt.gca().coastlines()

plt.show()
```

Plotting a cube

```
import matplotlib.pyplot as plt

import iris
import iris.quickplo

# Load the data
fname = iris.sample_
temperature_cube = i

# Draw the contour w
qplt.contourf(temper

# Add coastlines to
plt.gca().coastlines

plt.show()
```



Collapsing cubes

Cubes can be collapsed using various statistical/mathematical operations.

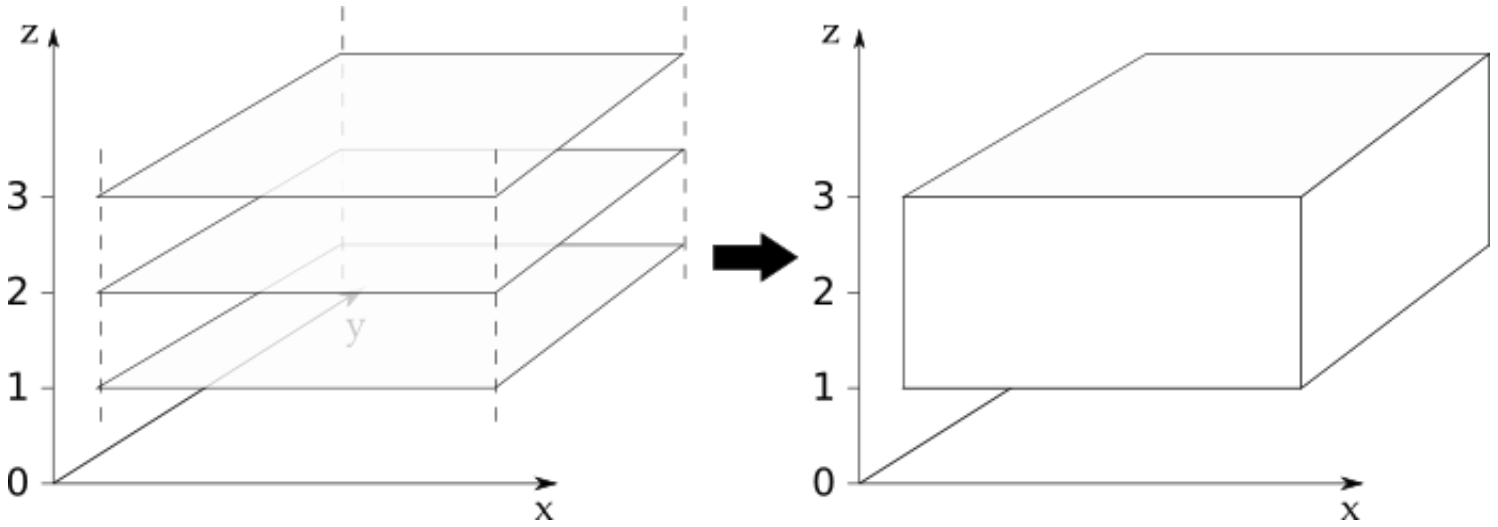
Calculate a time-series mean:

```
air_temp_mean = air_temp.collapsed('time',
iris.analysis.MEAN)
```

Merging cubes

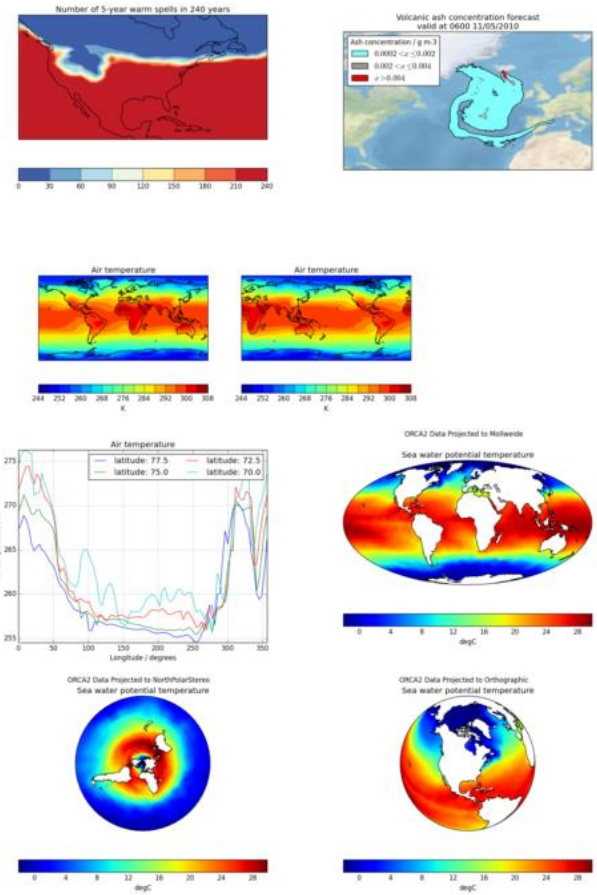
```
>>> print(cubes)
0: air_temperature / (kelvin)                (y: 4; x: 5)
1: air_temperature / (kelvin)                (y: 4; x: 5)
2: air_temperature / (kelvin)                (y: 4; x: 5)

>>> print(cubes[0])
air_temperature / (kelvin)                   (y: 4; x: 5)
 ...       z: 1 meters
>>> print(cubes[1])
air_temperature / (kelvin)                   (y: 4; x: 5)
 ...       z: 2 meters
>>> print(cubes[2])
air_temperature / (kelvin)                   (y: 4; x: 5)
 ...       z: 3 meters

>>> print(cubes.merge())
0: air_temperature / (kelvin)                (z: 3; y: 4; x: 5)
```

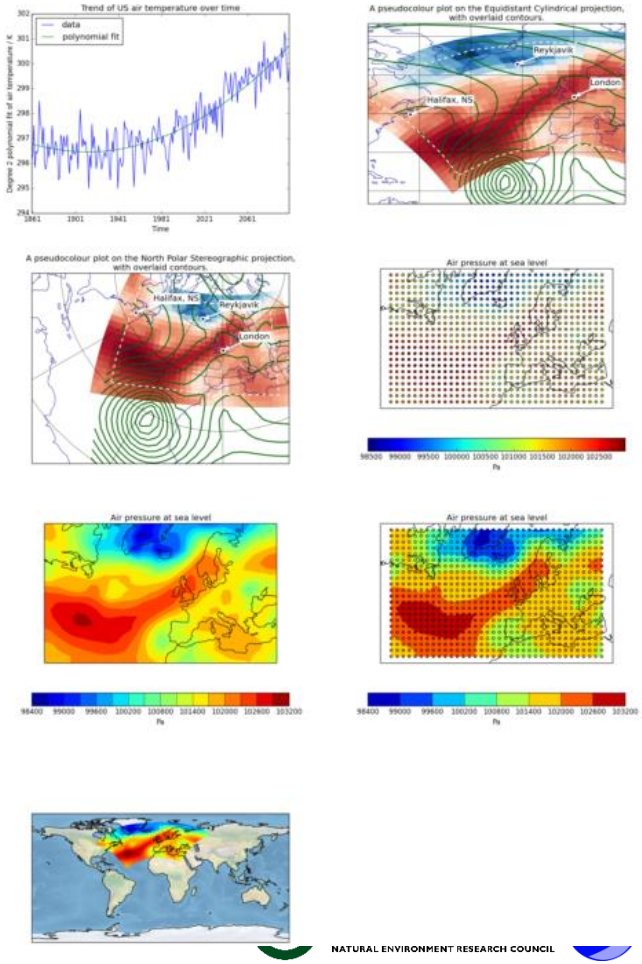
Merging cubes



And plotting

See: https://scitools.org.uk/iris/docs/latest/gallery





Further reading

Iris documentation:
https://scitools.org.uk/iris/docs/latest

Iris image gallery:
https://scitools.org.uk/iris/docs/latest/gallery