

Working with ECMWF data in Python

Building a new framework to interact with ECMWF data & services

Stephan Siemen, Iain Russell,
Fernando Ii, Sándor Kertész
Development Section, ECMWF



Python Software Foundation [US] | <https://pypi.org/projects/metview/>

Search projects

metview 0.9.1

`pip install metview`

Metview Python API.

Navigation

- Project description
- Release history
- Download files

Project description

Python interface to Metview examining, manipulating and plotting ECMWF data. <https://software.ecmwf.int/>

Requirements

```
In [2]: import metview as mv

In [29]: t = mv.read('2m_temperature.grib')
print(mv.datainfo(t))
[{'in': '2m_temperature.grib', 'cross_sect': 'polar', 'datainfo': {'date': '2013-03-25T00:00:00', 'date_desc': 'Monday 25 March 2013 00 UTC ecmf surface 2 metre temperature', 'db_info': '2m_temperature.grib', 'definition': '2m_temperature', 'describe': '2m_temperature', 'det': '2m_temperature', 'dictionary': '2m_temperature'}, 'polar': {'map_projection': 'polar_stereographic', 'map_area_definition': 'corners', 'area': [19.62, -31.44, 39.66, 80.1]}}
```

To view your plot in a Jupyter notebook, call "mv.setoutput('jupyter')" at some point before plotting

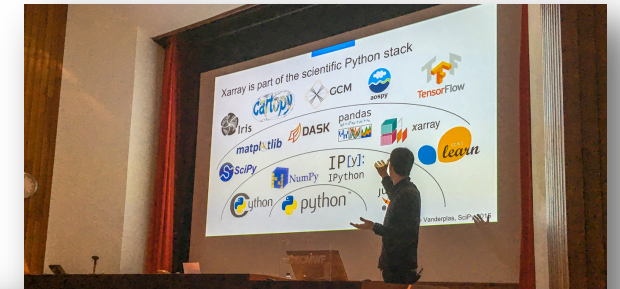
```
In [31]: mv.setoutput('jupyter')
mv.plot(polar, t, pal)
```

Out[31]:

Active engagement with community

We had now two workshops with wider Python community

- There are already many good efforts and solutions out there
 - Many good “wheels” which do not need to be reinvented
 - We want to allow easy interactions between frameworks
- Confirmation of our direction for developments
 - Using common Python packages for meteorological data
 - Handle fields through *xarray*; *pandas* for tables/time series
 - Build Python interfaces the Python way; not how legacy Fortran/C interfaces were done
 - We will take this to heart when moving to Python 3
- Building a community is more than just releasing software under Open Source
 - ‘Open Source’ versus ‘Open Development’ → embrace new culture



CDS (toolbox) released earlier this year

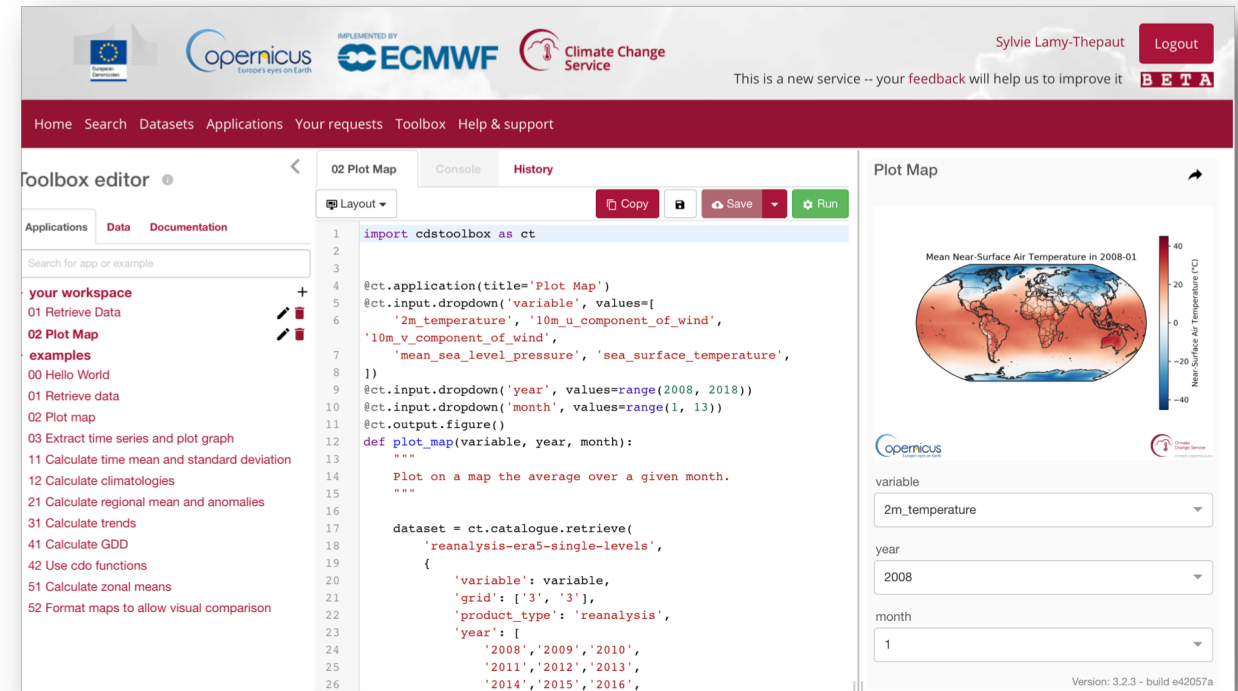


- New portal to find / download and work with Copernicus Climate data

- High-level descriptive Python interface
 - Allow non-domain users to build apps

- Try it out yourself:

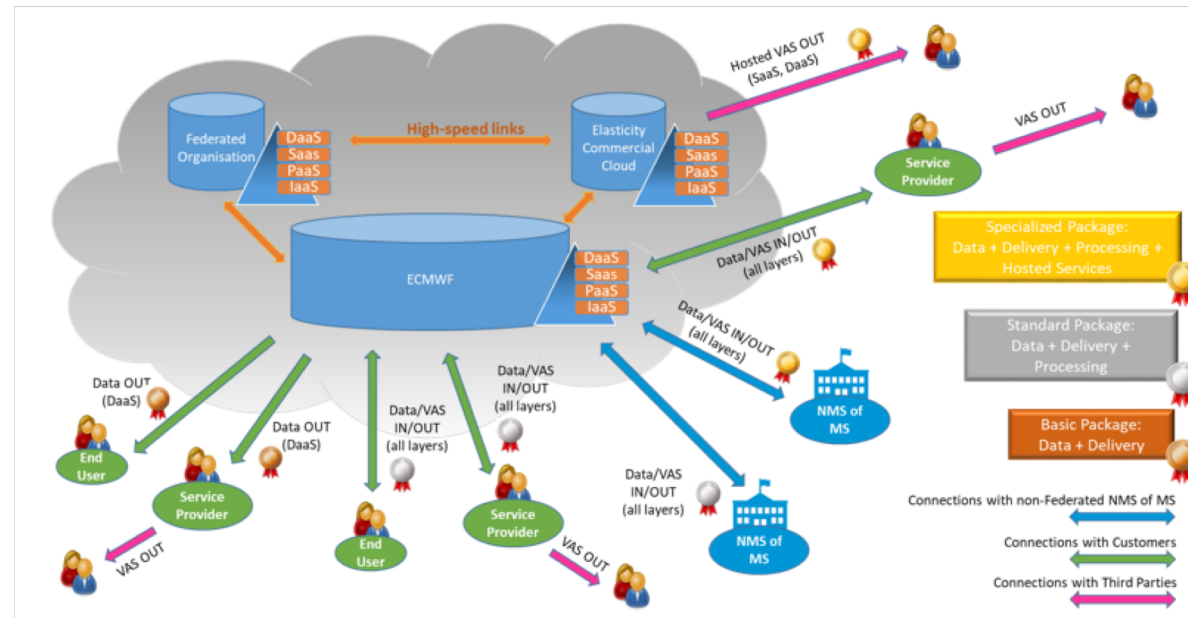
<https://cds.climate.copernicus.eu>





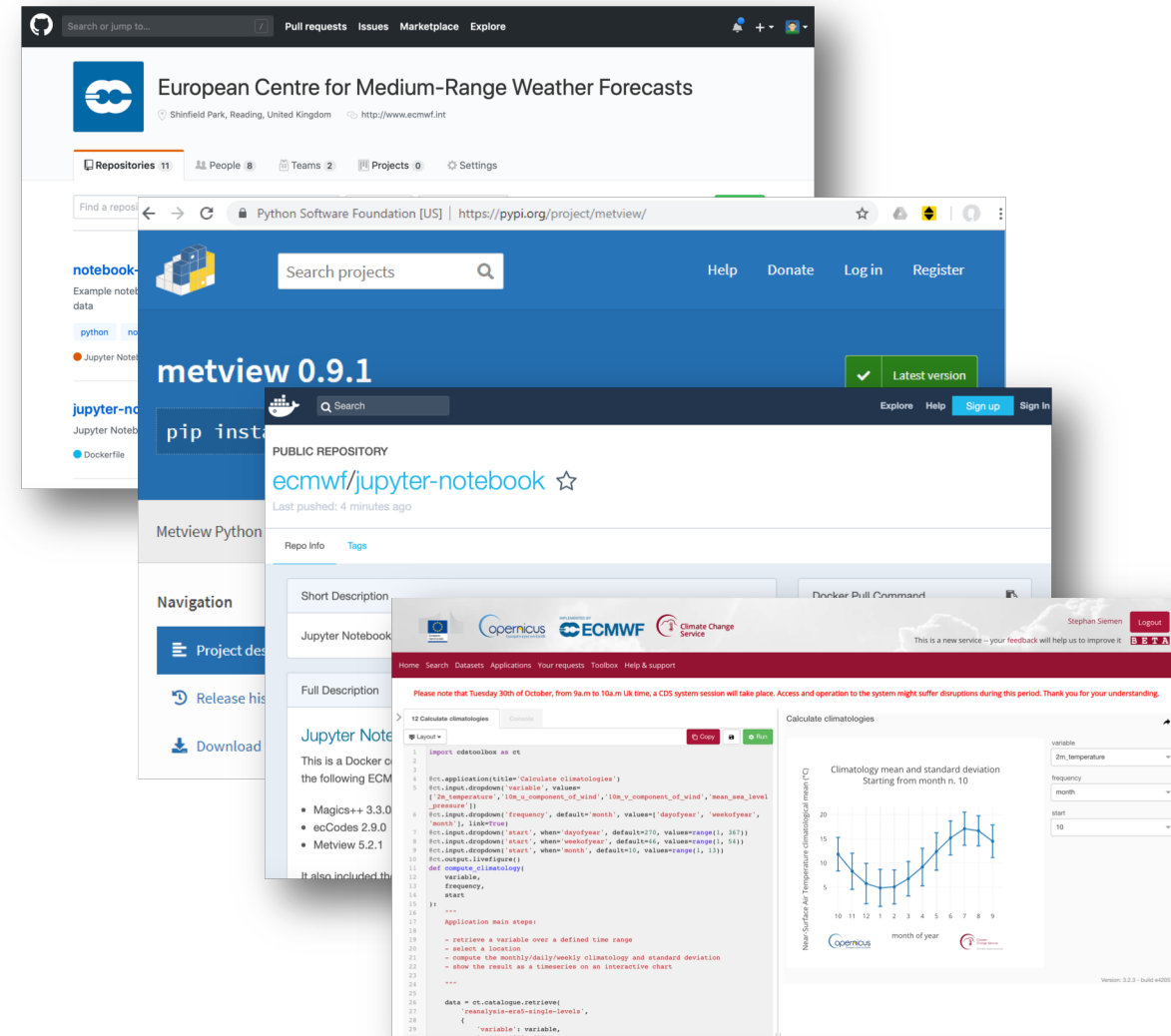
New opportunities through cloud services

- ECMWF looks together with its partners on providing private clouds close to data
 - *European Weather Cloud* with EUMETSAT
 - Copernicus *WEkEO* DIAS in co-operation with EUMETSAT & Mercator Ocean
- Making it easier for users to work with ECMWF forecast & Copernicus data
 - And Python will play an important role here
 - Fast deployment + high level interfaces to abstract technical implementations



Making software easily available within existing frameworks

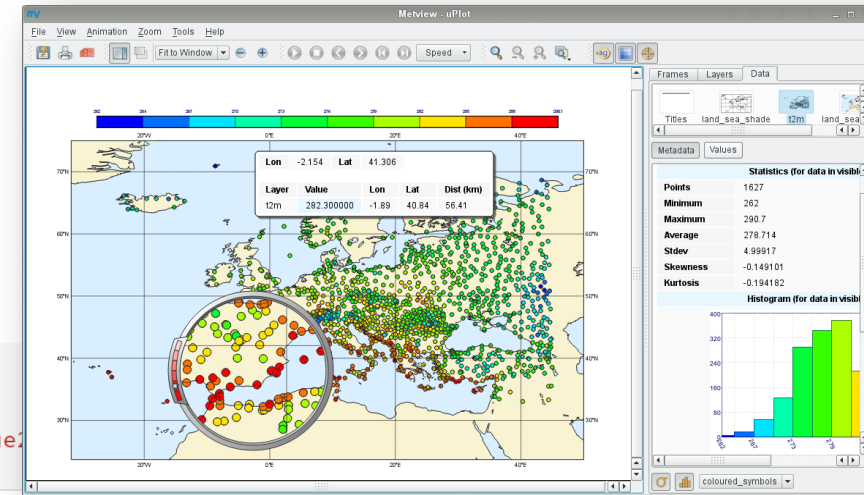
- Source code & examples on GitHub
- Packages need be on PyPi and Conda
- ECMWF Python software on DockerHub
- SaaS - CDS Toolbox



The Metview Python framework

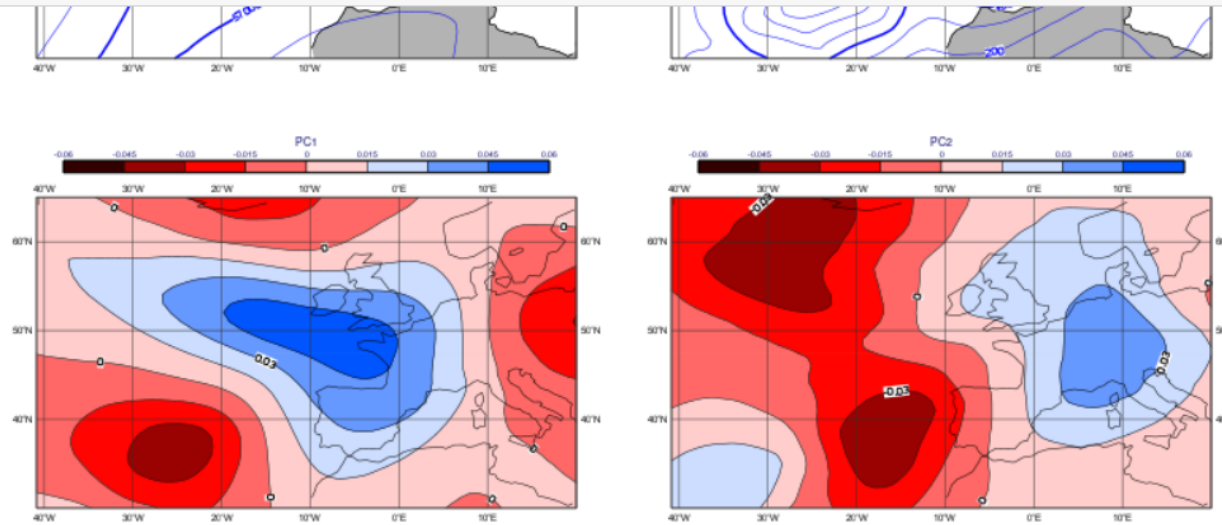
- A high-level Python 3 interface for processing and visualising ECMWF data
- Aim is to allow users of Metview to use easily the power of Python but still have all functionality of Metview; including visualisation
- Beta release - all Metview functionality available from Python 3
- Close co-operation CDS toolbox

```
contour_shade = "on",  
contour_shade_colour_method = "palette",  
contour_shade_method = "area_fill",  
contour_shade_palette_name= "eccharts_red_blue"
```

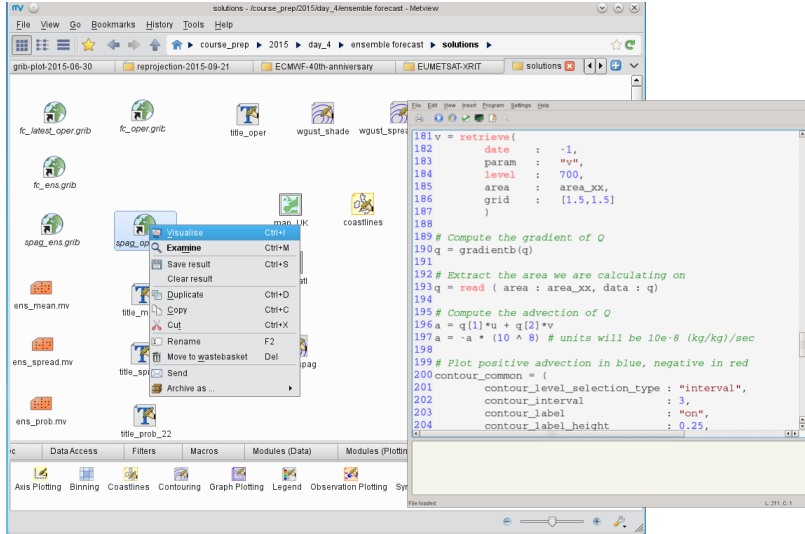


Finally, we plot each field with a custom title. We compute the ensemble mean and spread on the fly with fieldset functions from Metview.

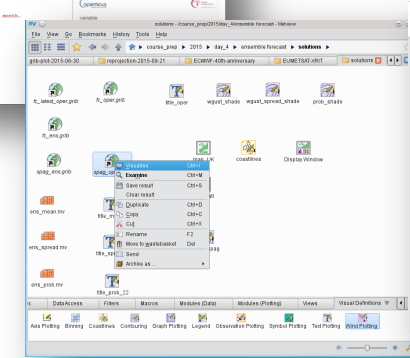
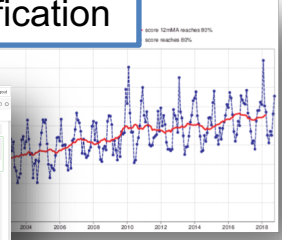
```
[12]: mv.plot(dw[0], fs.mean(), mv.mtext(text_line_1 = "ENS mean"),  
            dw[1], fs.stdev(), mv.mtext(text_line_1 = "ENS spread"),  
            dw[2], g[0], cont_pc, mv.mtext(text_line_1 = "PC1"),  
            dw[3], g[1], cont_pc, mv.mtext(text_line_1 = "PC2"))
```



Evolution of Metview to Python



Quaver verification



Keep - High-level interface to abstract technical details

Metview Macro

mars client, web_api, Magics, MIR

C++

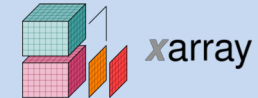
Bespoke implementation of data models and communication between services

We're here now

Metview Python

mars client, cdsapi, web_api, Magics, MIR

pandas



Easy extension through Python

→ Evolve – make internally more use of community packages and contribute to them →

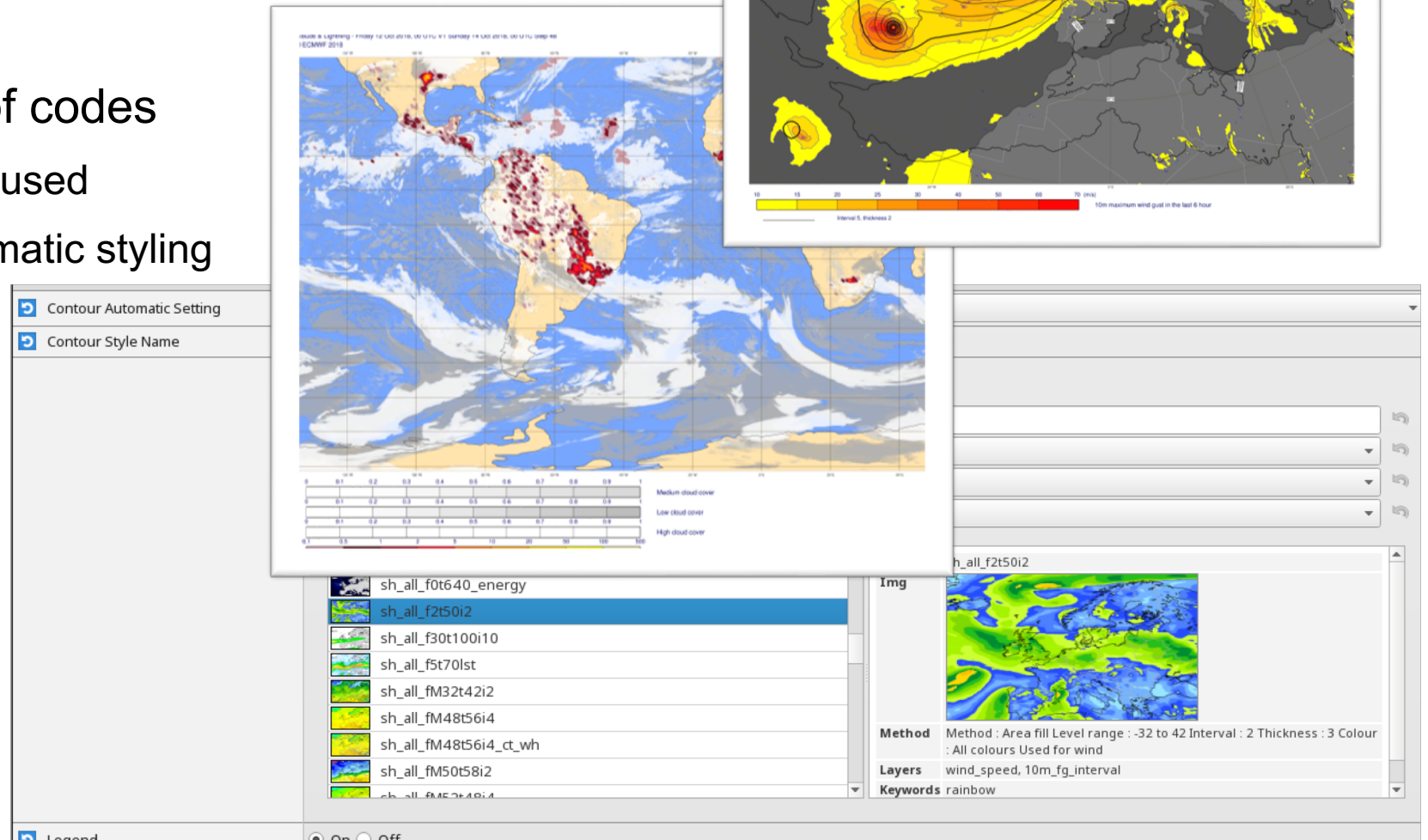
Generation of Python code

The image illustrates the process of generating Python code for a plot in the Metview environment. It shows three main components:

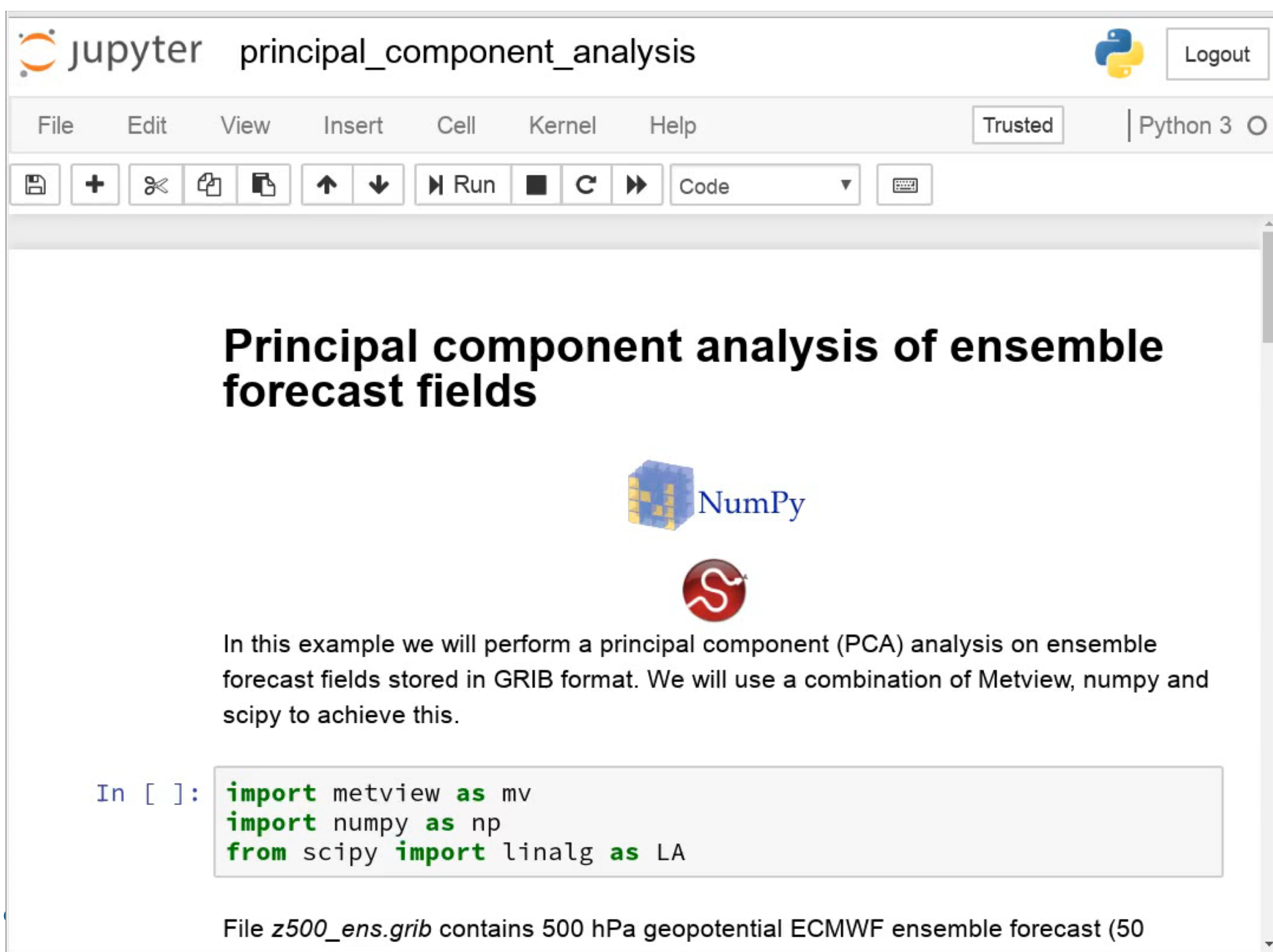
- Metview Main Window:** Displays a plot area with various data layers. The 'coloured_symbols' layer is highlighted, and a 'Python Script.py' icon is visible, indicating the generation of code for this layer.
- coloured_symbols Dialog:** A configuration dialog for the 'coloured_symbols' layer. It shows the 'Symbol Advanced Table Max Level Colour' set to Red and the 'Symbol Advanced Table Min Level Colour' set to Blue. The dialog also includes a color wheel and a grid for color selection, along with fields for 'HTML' (#0000ff) and 'Macro' ('RGB(0.0...').
- Python Script.py Editor:** A code editor window showing the generated Python code. The code includes the line `view = mv.geoview(`, which is highlighted by a blue arrow pointing from the 'Python Script.py' icon in the main window.

Other benefits of high level definitions

- Concept is also used by web framework & CDS
 - Easy way to migrate definitions between systems
- Allows reuse and sharing of codes
 - Definitions can easily be reused
 - Higher level use; e.g. automatic styling



Using NumPy and SciPy



The screenshot shows a Jupyter Notebook titled "principal_component_analysis". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help), a "Trusted" status indicator, and "Python 3" as the selected kernel. The toolbar contains icons for saving, adding cells, cutting, copying, pasting, navigating, running, and refreshing. The main content area features a large heading: "Principal component analysis of ensemble forecast fields". Below the heading are the logos for NumPy and SciPy. A paragraph explains the task: "In this example we will perform a principal component (PCA) analysis on ensemble forecast fields stored in GRIB format. We will use a combination of Metview, numpy and scipy to achieve this." A code cell is shown with the following Python code:

```
In [ ]: import metview as mv
import numpy as np
from scipy import linalg as LA
```

At the bottom of the notebook, a text box states: "File z500_ens.grib contains 500 hPa geopotential ECMWF ensemble forecast (50".

Using pandas

jupyter field_obs_difference

File Edit View Insert Cell Kernel Help Trusted Python 3

Difference between gridded field (GRIB) and scattered observations (BUFR)

pandas $y_i = \beta' x_i + \mu_i + \epsilon_i$

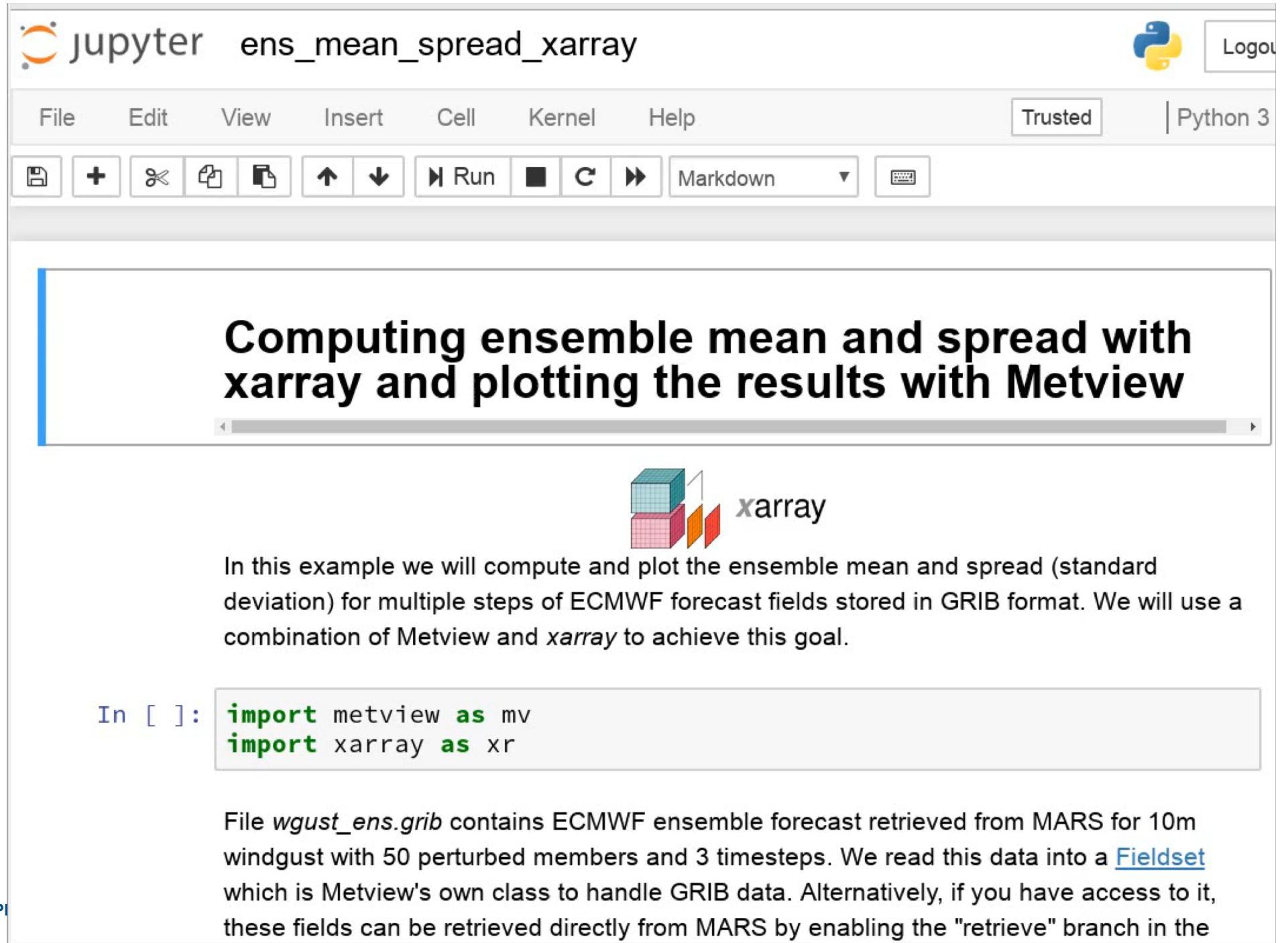
In this example we will load a gridded model field in GRIB format and a set of observation data in BUFR format. We will then use Metview to examine the data, and compute and plot their differences. Then we will export the set of differences into a pandas dataframe for further inspection.

```
In [ ]: import metview as mv
```

```
In [ ]: use_mars = False # if False, then read data from disk
```

Metview retrieves/reads GRIB data into its [Fieldset](#) class.

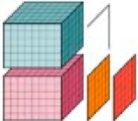
Using xarray



jupyter ens_mean_spread_xarray

File Edit View Insert Cell Kernel Help Trusted Python 3

Computing ensemble mean and spread with xarray and plotting the results with Metview



xarray

In this example we will compute and plot the ensemble mean and spread (standard deviation) for multiple steps of ECMWF forecast fields stored in GRIB format. We will use a combination of Metview and *xarray* to achieve this goal.

```
In [ ]: import metview as mv
import xarray as xr
```

File *wgust_ens.grib* contains ECMWF ensemble forecast retrieved from MARS for 10m windgust with 50 perturbed members and 3 timesteps. We read this data into a [Fieldset](#) which is Metview's own class to handle GRIB data. Alternatively, if you have access to it, these fields can be retrieved directly from MARS by enabling the "retrieve" branch in the

Benefitting the wider community

cfgrid – linking xarray and ecCodes

- Essential building block for Metview-Python
- To embrace xarray for all our field data, we needed to know that we could handle all our GRIB 1 & 2 data
 - Therefore it was important for us to have a solution based on ecCodes
- Open to the whole community
 - First user: CDS toolbox

The top screenshot shows the GitHub repository page for `ecmwf / cfgrid`. The repository description is "A Python interface to map GRIB files to the NetCDF Common Data Model following the CF Convention using ecCodes". It has 65 stars (circled in red), 13 watchers, and 7 forks. The repository statistics show 699 commits, 2 branches, 22 releases, and 2 contributors. The bottom screenshot shows a pull request in the `pydata / xarray` repository titled "Add a GRIB backend via ECMWF cfgrid / ecCodes #2476". The pull request is merged and was created by `alexamici` 11 days ago. The pull request description includes a checklist of items to be reviewed and implemented.

Metview and CDS data





jupyter wind_gust_nc_era5_cds

File Edit View Insert Cell Kernel Help Trusted Python 3

Run

Retrieving netCDF data from the CDS and plotting it with Metview



Demonstrates how to retrieve netCDF data from the [Climate Data Store](#) (CDS) and visualise it with [Metview](#) using automatic styling and a polar stereographic projection.

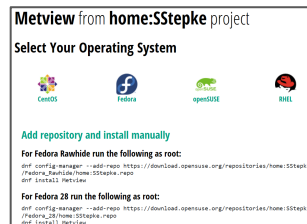
```
In [ ]: import metview as mv
import cdsapi
```

Retrieves ERA5 instantaneous 10 metre wind gust data in netCDF format using the [CDS API](#) (access needs to be set up first). If you do not have access to the CDS-API then initialise variable `use_cds = False`. A copy of the data is provided on disk.

```
In [ ]: use_cds = True
```

How can I use Metview Python right now?

- Documentation on Confluence
 - <https://confluence.ecmwf.int/metview/Metview's+Python+Interface>
- Docker image on DockerHub
 - <https://hub.docker.com/r/ecmwf/jupyter-notebook/>
- Available on github and PyPi
 - <https://github.com/ecmwf/metview-python>
 - `pip install metview`
 - Requires the Metview binaries to be installed too



- After installation, visit our Jupyter notebook examples at <https://github.com/ecmwf/notebook-examples>

At ECMWF

- Installed on all machines
- Use `module load metview-python`

The image shows two overlapping browser windows. The top window is the GitHub repository for `ecmwf/metview-python`, displaying the repository name, navigation tabs (Code, Pull requests, Wiki, Insights, Settings), and a sidebar with files like `docs`, `examples`, `metview`, and `requirements`. The bottom window is the PyPI project page for `metview 0.9.1`, showing the version number, a 'Latest version' badge, the installation command `pip install metview`, and a 'Last released: 5 minutes ago' timestamp. The page also includes a navigation menu and a project description.