

# 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 <https://software.ecmwf.int/>

Requirements

Jupyter Untitled Last Checkpoint: an hour ago (unsaved changes)

```
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', 'proportion_present': '1', 'proportion_missing': '0'}]
```

```
In [30]: pal = mv.palette(
    mv.date,
    mv.db_info,
    mv.definition,
    mv.describe,
    mv.det,
    mv.dictionary)

polar = mv.geoview(
    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

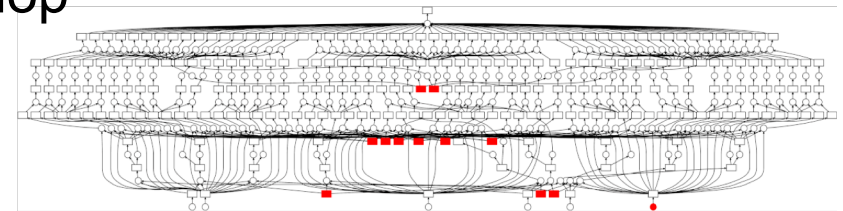
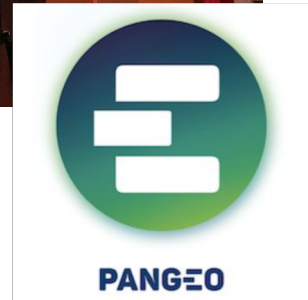
- There are already many good efforts and solutions out there
  - Many good “wheels” which do not need to be reinvented
  - We want to allow easy interactions between frameworks
- Confirmation of our direction for developments
  - Using common Python packages for meteorological data
    - Handle fields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



# Looking at data processing at scale



- Python has long been seen as not capable of processing large amounts of data efficiently
- This has changed in recent years with the emergence of packages like pandas, xarray, dask, ...
- We are now looking at how we can use Python itself for larger datasets and provide tutorials & cookbooks for users how to process our data
- We had a very good session on this topic at the last workshop
  - E.g the work of the Pangeo initiative here is very relevant



- These and many other interesting presentations from this year's workshop you can find at <https://www.ecmwf.int/en/learning/workshops/2018-workshop-developing-python-frameworks-earth-system-sciences>

# Copernicus Climate Data Store (CDS) released earlier this year

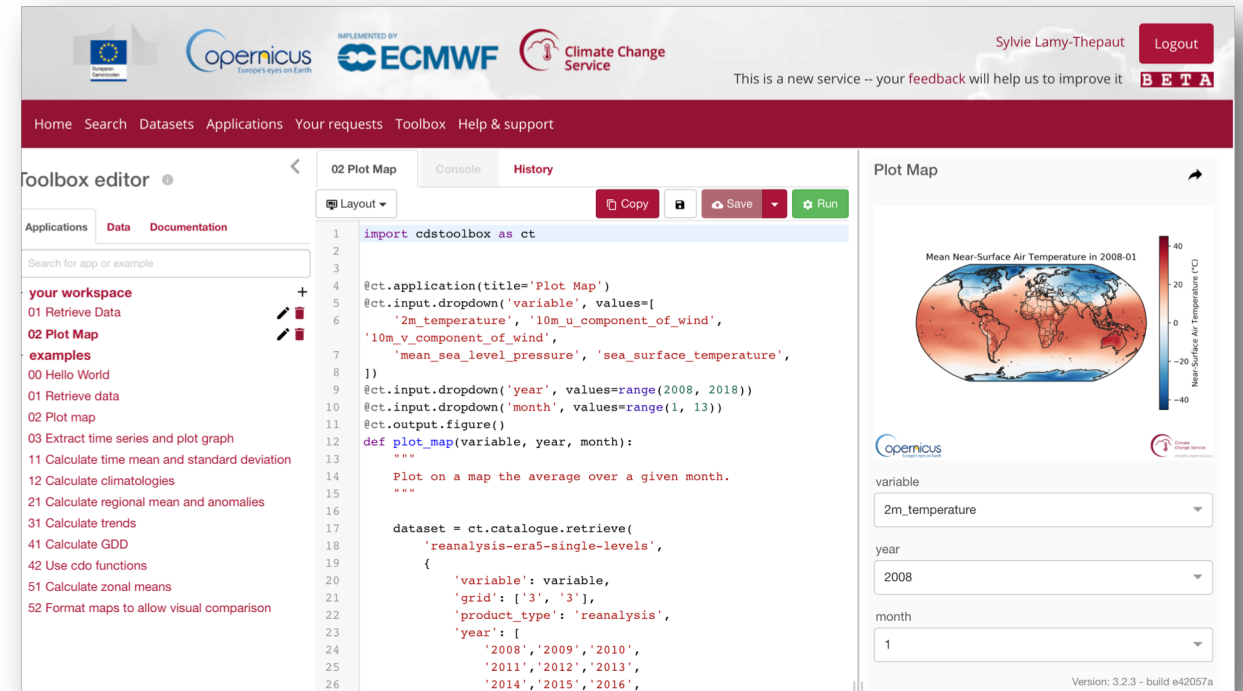


- New portal to find / download and work with Copernicus climate data

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

- Try it out yourself:

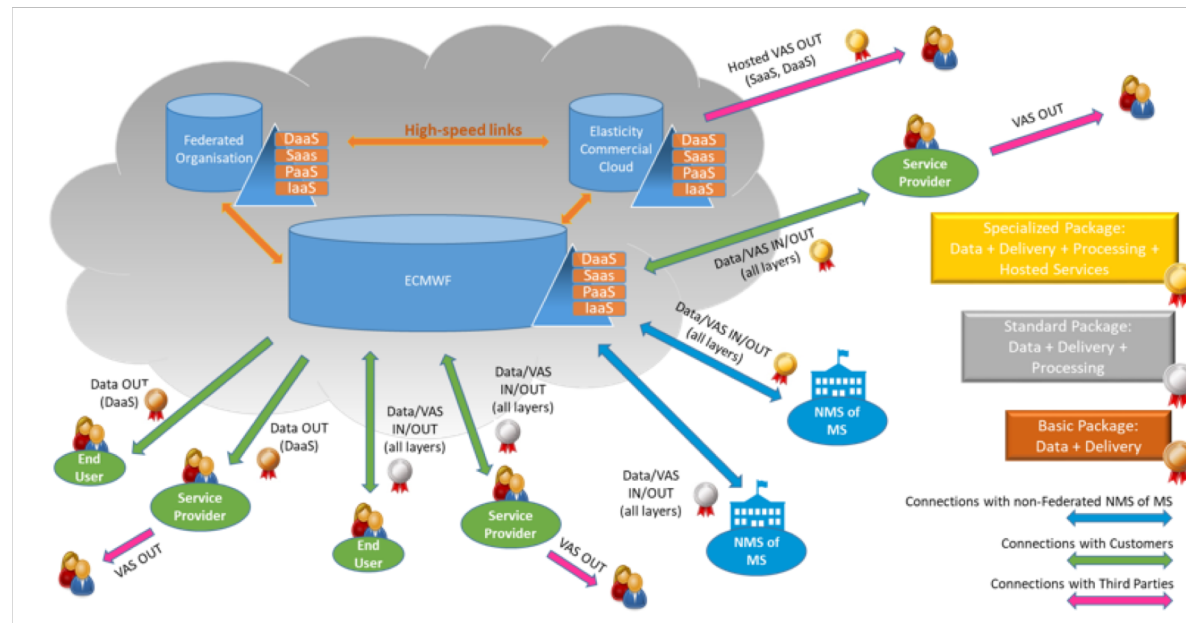
<https://cds.climate.copernicus.eu>





# New opportunities through cloud services

- ECMWF looks together with its partners on providing private clouds close to data
  - E.g. 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

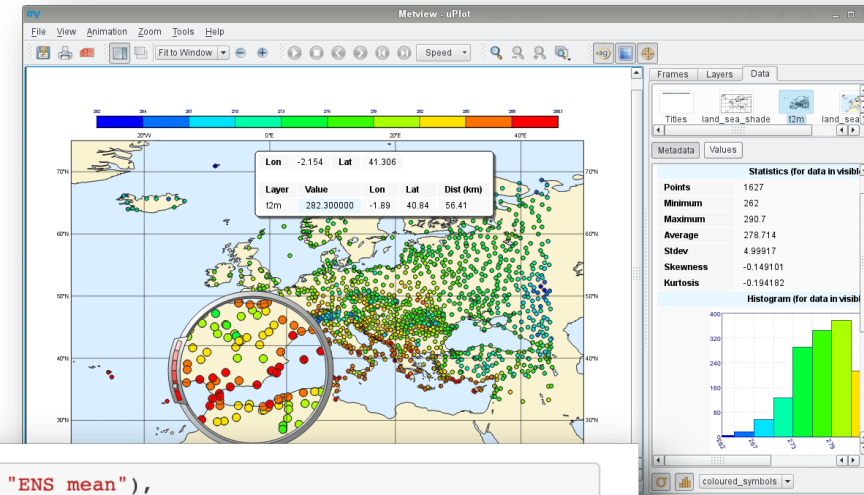
- Source code & examples on GitHub
- Packages need be on PyPi and Conda
- ECMWF Python software on DockerHub
- SaaS - CDS Toolbox

The image is a collage of four screenshots illustrating the integration of ECMWF software into various frameworks:

- Top:** GitHub profile for the European Centre for Medium-Range Weather Forecasts (ECMWF).
- Middle-left:** PyPI page for 'metview 0.9.1', showing it is the latest version.
- Middle-right:** DockerHub repository for 'ecmwf/jupyter-notebook'.
- Bottom:** Copernicus CDS Toolbox interface. It shows a code editor with Python code for calculating climatologies and an interactive chart displaying 'Climatology mean and standard deviation' for '2m\_temperature' over a 10-month period.

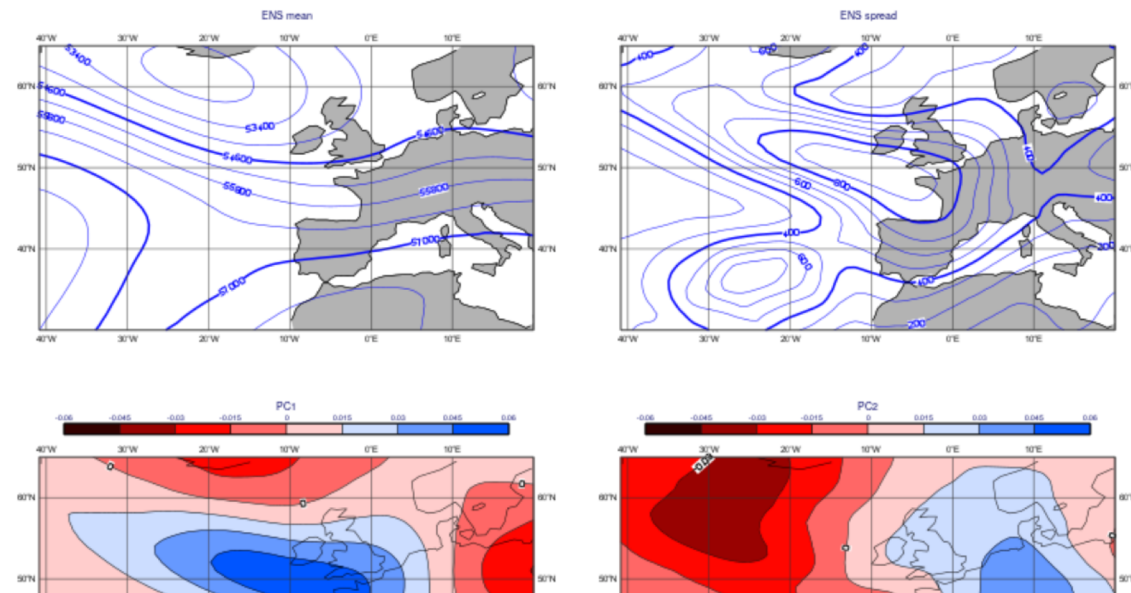
# The Metview Python framework

- 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

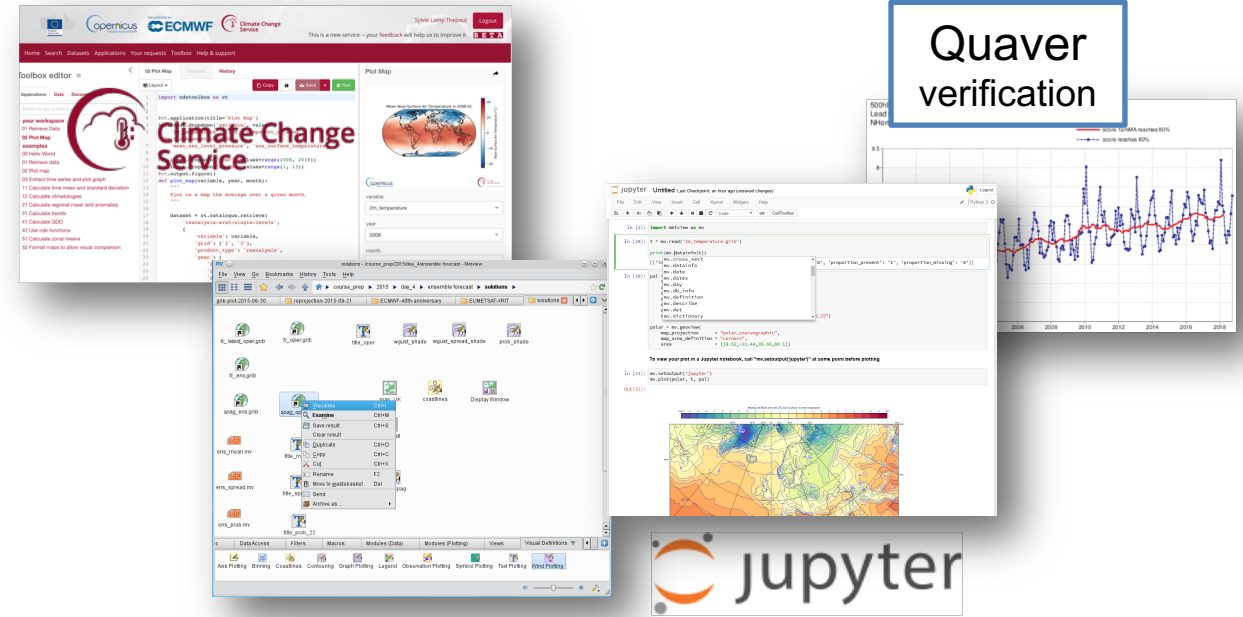
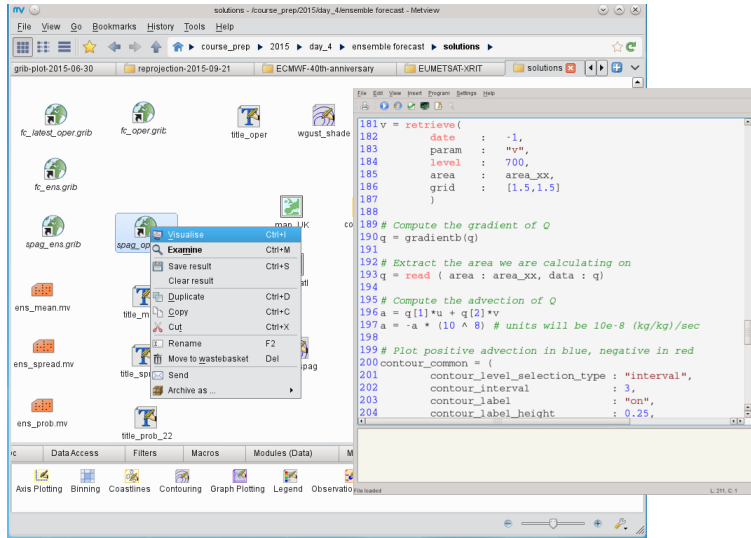


```
In [12]: mv.plot(dw[0], fs.mean(), mv.mtext(text_line_1 = "ENS mean"),  
dw[1], fs.stdev(), mv.mtext(text_line_1 = "ENS spread"),  
dw[2], g[0], cont_pc, mv.mtext(text_line_1 = "PC1"),  
dw[3], g[1], cont_pc, mv.mtext(text_line_1 = "PC2"))
```

Out[12]:



# Evolution of Metview to Python



Keep - High-level interface to abstract technical details

## Metview Macro

mars client, web\_api, Magics, MIR

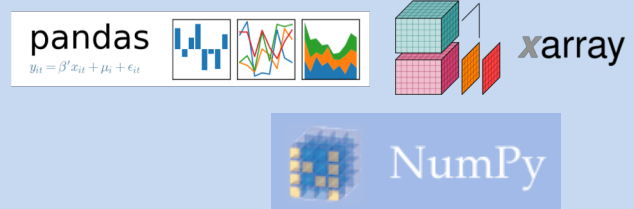
C++

Bespoke implementation of data models and communication between services



## Metview Python

mars client, cdsapi, web\_api, Magics, MIR



Easy extension through Python

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



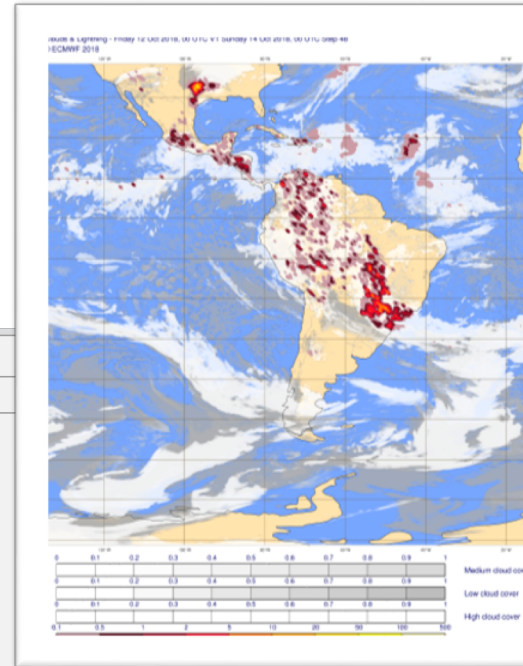
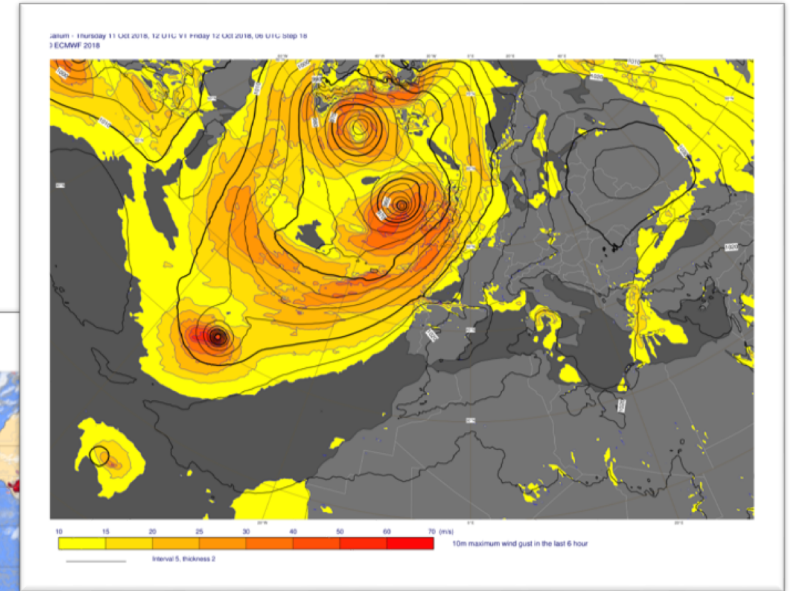
# Generation of Python code

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

- Metview Project Window:** Displays the project 'obs-filter-and-plot' with various modules and views. The 'coloured\_symbols' module is highlighted, and a 'Python Script.py' icon is shown next to it.
- Symbol Properties Dialog:** A dialog box for the 'coloured\_symbols' icon, showing parameters for 'Symbol Advanced Table Max Level Colour' (Red) and 'Symbol Advanced Table Min Level Colour' (Blue). It also includes a color wheel and a 'Symbol Advanced Table Colour Direction' set to 'Clockwise'.
- Python Script Editor:** Shows the resulting Python code for the 'view' object, including the line `view = mv.geoview(` and subsequent lines for the 'coloured\_symbols' module.

# 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



Contour Automatic Setting

Contour Style Name

sh\_all\_f0t640\_energy

sh\_all\_f2t50i2

sh\_all\_f30t100i10

sh\_all\_f5t70lst

sh\_all\_fm32t42i2

sh\_all\_fm48t56i4

sh\_all\_fm48t56i4\_ct\_wh

sh\_all\_fm50t58i2

sh\_all\_fm52t48i4

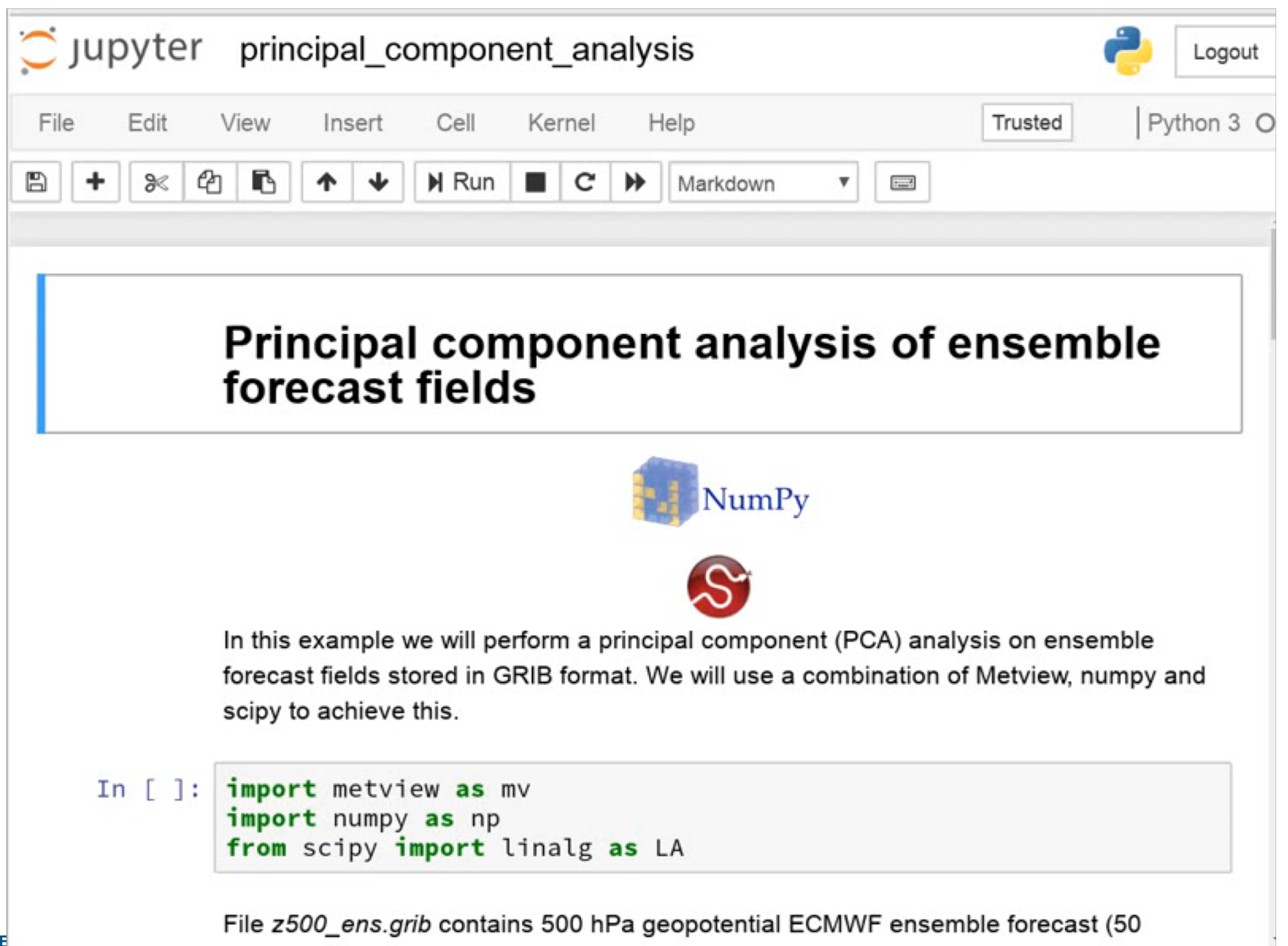
sh\_all\_f2t50i2

Method : Method : Area fill Level range : -32 to 42 Interval : 2 Thickness : 3 Colour : All colours Used for wind

Layers wind\_speed, 10m\_fg\_interval

Keywords rainbow

# Using NumPy and SciPy

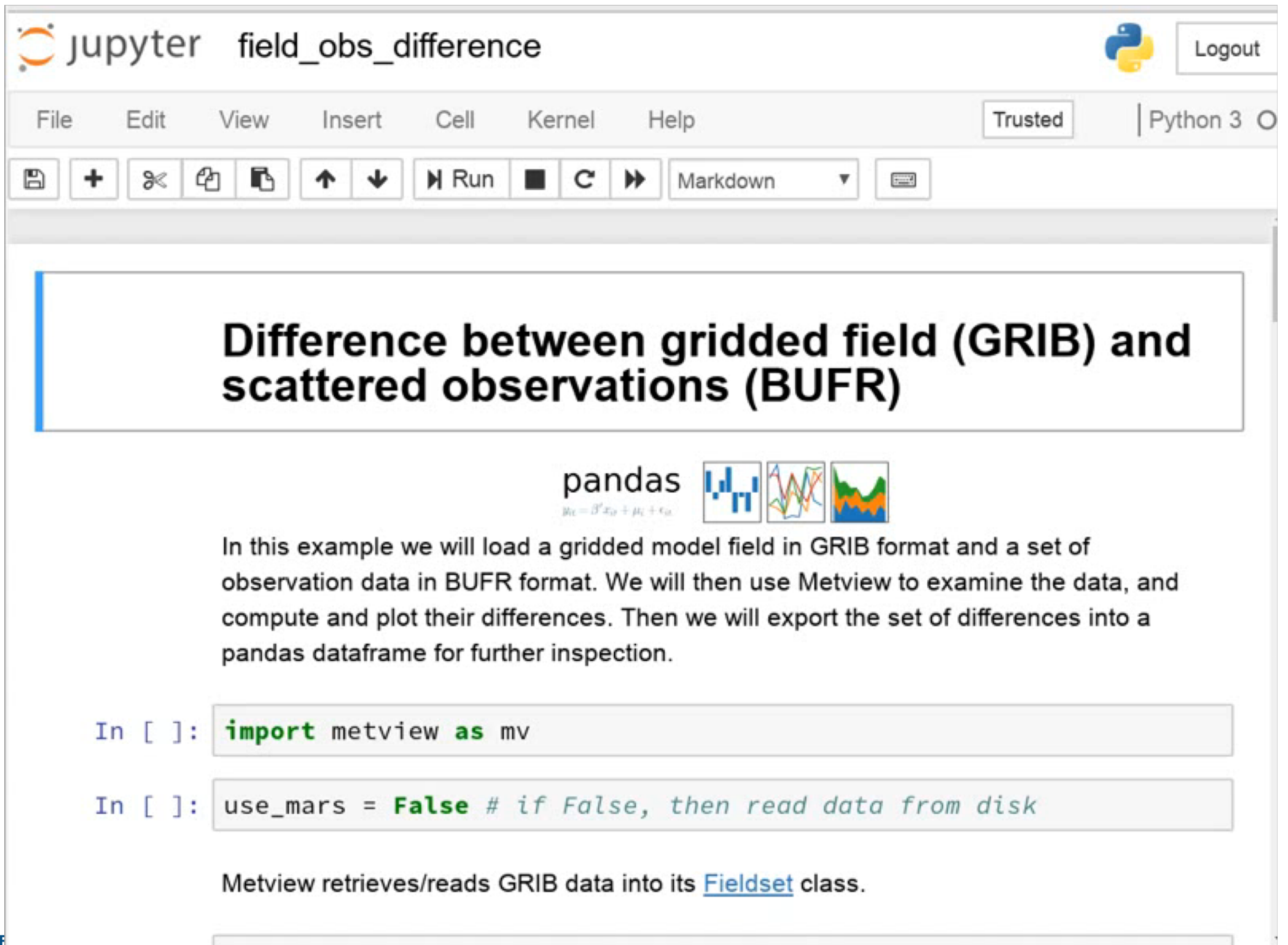


The screenshot shows a Jupyter Notebook titled "principal\_component\_analysis". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help), a toolbar with icons for file operations and execution, and a "Trusted" status indicator. The notebook content features a title "Principal component analysis of ensemble forecast fields", followed by the NumPy and SciPy logos. A text block explains the task: performing a PCA analysis on ensemble forecast fields in GRIB format using Metview, numpy, and scipy. Below this is a code cell with the following Python code:

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

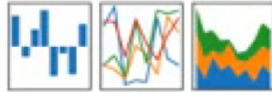
At the bottom of the notebook, a text block states: "File z500\_ens.grib contains 500 hPa geopotential ECMWF ensemble forecast (50".

# Using pandas



The screenshot shows a Jupyter Notebook window titled "jupyter field\_obs\_difference". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help), a "Trusted" status indicator, and "Python 3" as the selected kernel. Below the menu is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The main content area displays a slide with the following text:

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

**pandas**   
 $y_t = \beta' x_t + \mu_t + \epsilon_t$

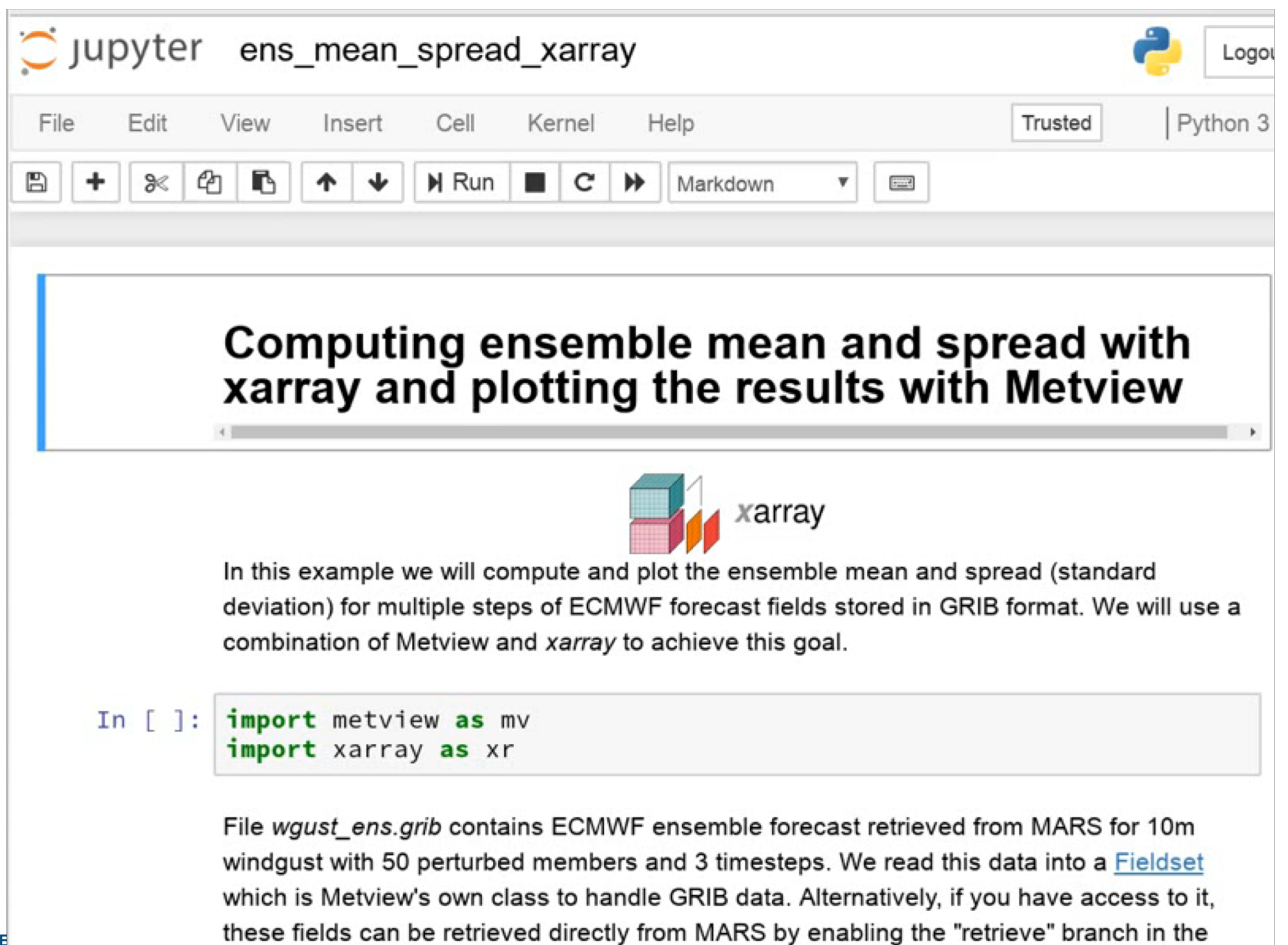
In this example we will load a gridded model field in GRIB format and a set of observation data in BUFR format. We will then use Metview to examine the data, and compute and plot their differences. Then we will export the set of differences into a pandas dataframe for further inspection.

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

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

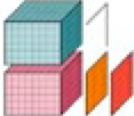
Metview retrieves/reads GRIB data into its [Fieldset](#) class.

# Using xarray



The screenshot shows a Jupyter Notebook window with the title 'ens\_mean\_spread\_xarray'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help), a 'Trusted' status indicator, and 'Python 3' as the kernel. Below the menu is a toolbar with icons for saving, adding, deleting, copying, pasting, and running code. The main content area displays a slide with the following text:

## 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.

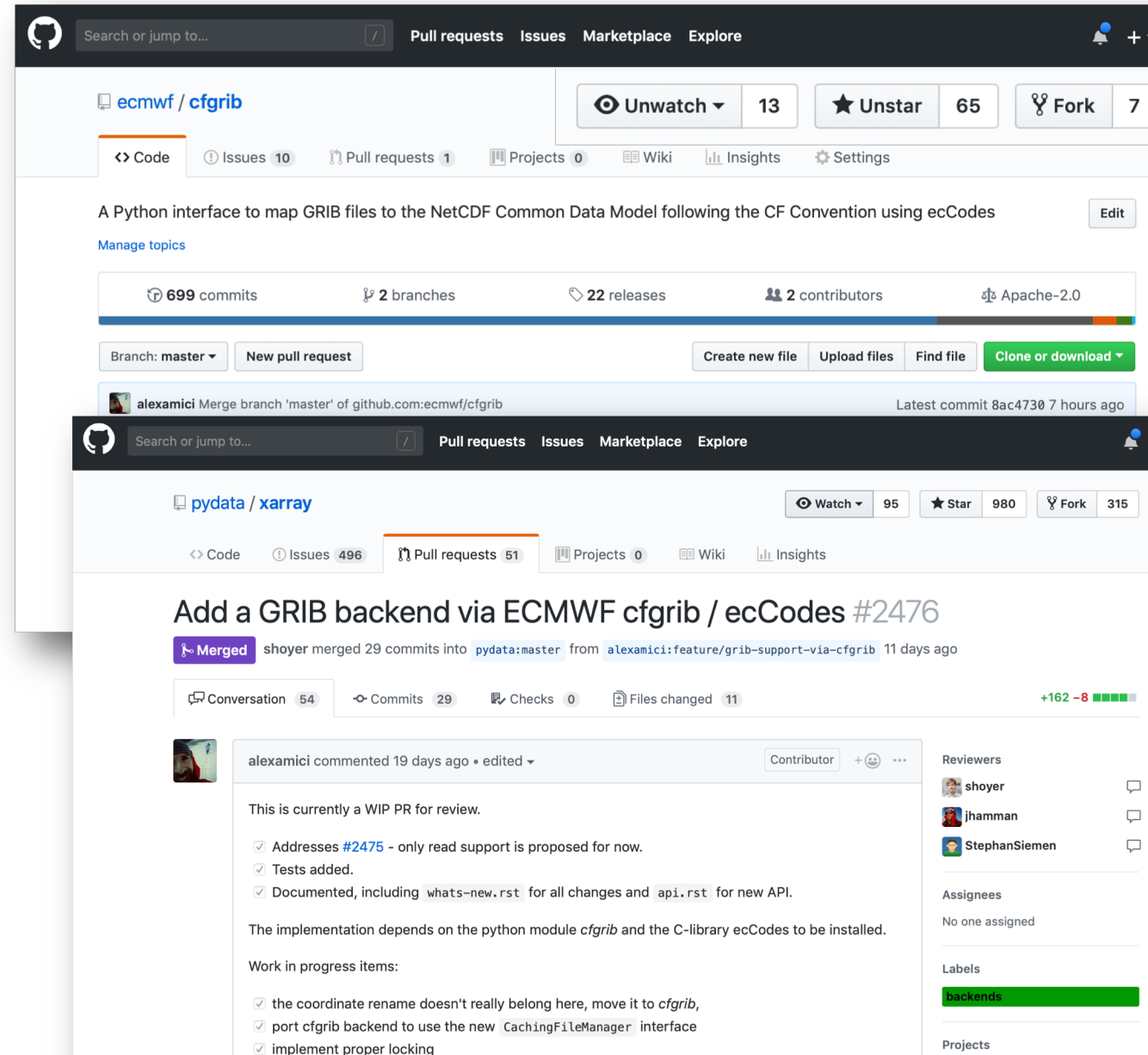
```
In [ ]: import metview as mv
import xarray as xr
```

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
  - Developed with our partners at 
- Open to the whole community
  - First user: CDS toolbox



The image shows two screenshots of GitHub. The top screenshot is the repository page for `ecmwf/cfgrid`, which is a Python interface to map GRIB files to the NetCDF Common Data Model. It shows 699 commits, 2 branches, 22 releases, and 2 contributors. The bottom screenshot shows a pull request titled "Add a GRIB backend via ECMWF cfgrid / ecCodes #2476" in the `pydata/xarray` repository. The PR is merged and includes a comment from alexamici stating it is a WIP PR for review. The comment lists several items that have been addressed: read support for GRIB 1 & 2, tests, and documentation. It also lists work in progress items: renaming coordinates, porting the backend to use the new `CachingFileManager` interface, and implementing proper locking.

# Metview and CDS data





jupyter wind\_gust\_nc\_era5\_cds  Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Save + Copy Paste Undo Redo Run Stop Refresh Markdown

## Retrieving netCDF data from the CDS and plotting it with Metview

 Climate Change Service



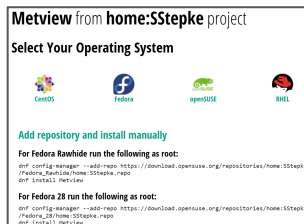
Demonstrates how to retrieve netCDF data from the [Climate Data Store](#) (CDS) and visualise it with [Metview](#) using automatic styling and a polar stereographic projection.

```
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.

# How can I use Metview Python right now?

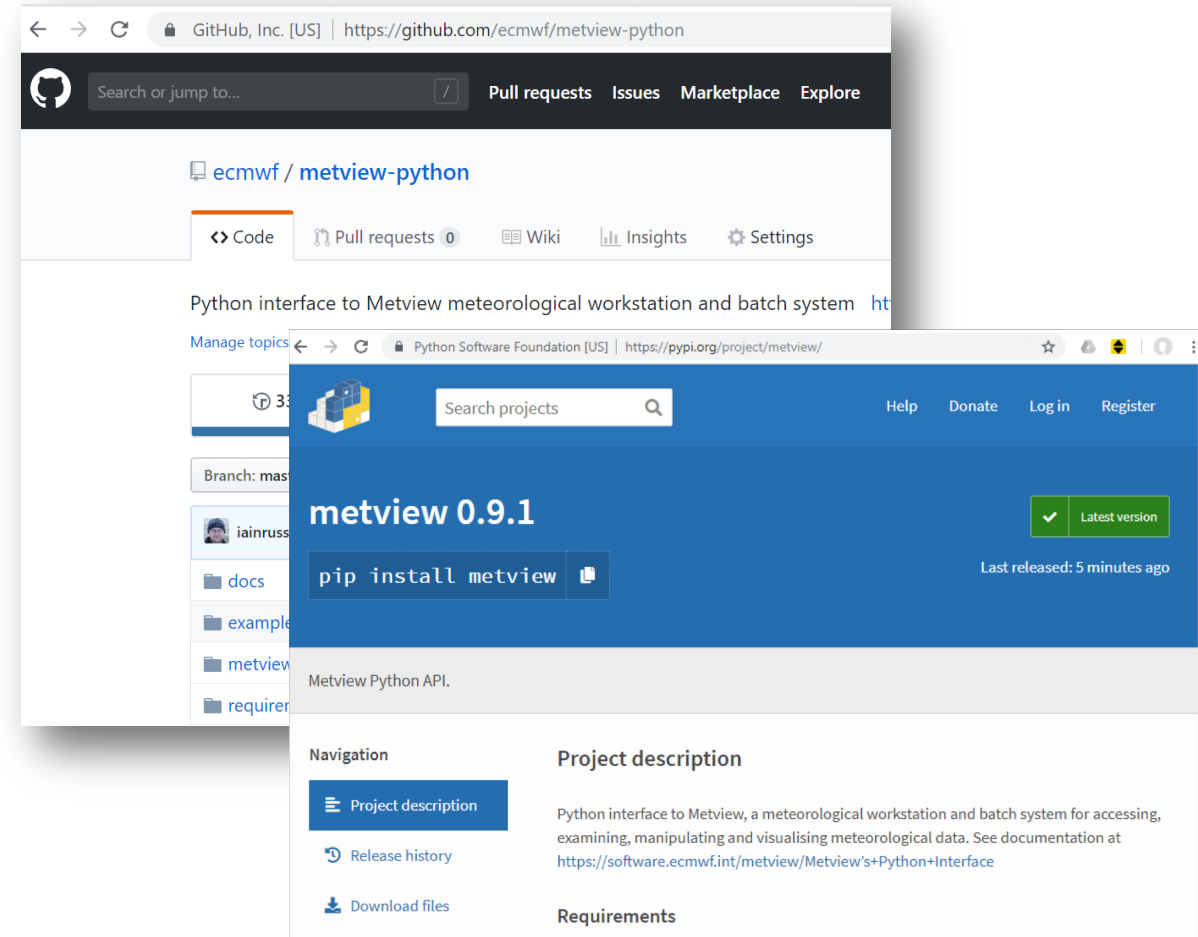
- Documentation on Confluence
  - <https://confluence.ecmwf.int/metview/Metview's+Python+Interface>
- Docker image on DockerHub
  - <https://hub.docker.com/r/ecmwf/jupyter-notebook/>
- Available on github and PyPi
  - <https://github.com/ecmwf/metview-python>
  - `pip install metview`
  - Requires the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



- Check out 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 page for `ecmwf/metview-python`. It displays the repository name, a search bar, and navigation links for Code, Pull requests, Wiki, Insights, and Settings. The description reads: "Python interface to Metview meteorological workstation and batch system". The bottom window is the PyPI project page for `metview`. It shows the project name "metview 0.9.1" with a "Latest version" badge and a green checkmark. Below the name is a large button with the text `pip install metview`. The page also includes a navigation menu with "Project description", "Release history", and "Download files", and a "Project description" section with the text: "Python interface to Metview, a meteorological workstation and batch system for accessing, examining, manipulating and visualising meteorological data. See documentation at <https://software.ecmwf.int/metview/Metview's+Python+Interface>".