The Bureau of Meteorology

Scores: an open source Python package based on xarray*

Nicholas Loveday, Tennessee Leeuwenburg, Aidan Griffiths, John Sharples, Mohammadreza Khanarmuei, Robert Taggart, Harrison Cook and Elizabeth Ebert

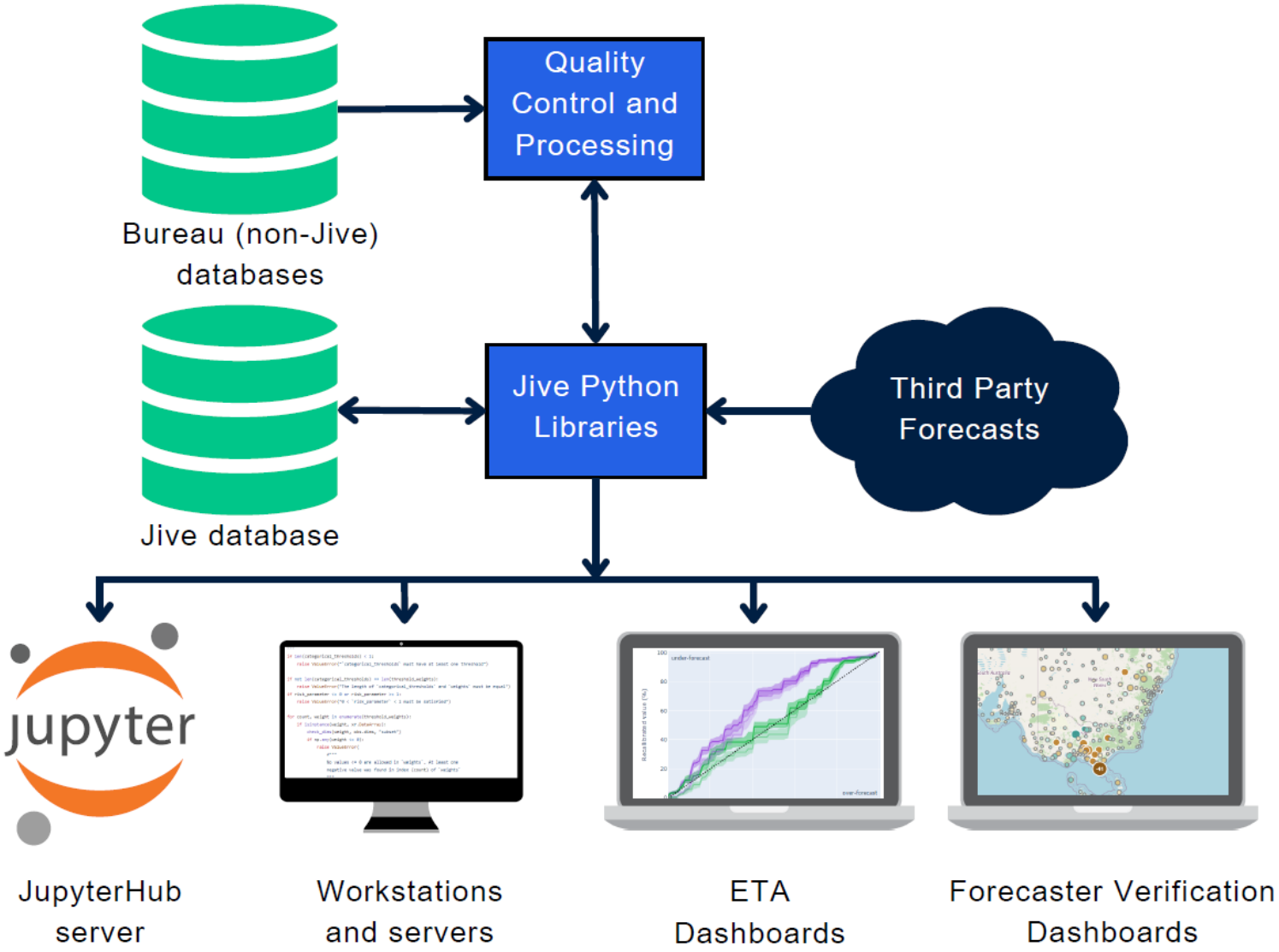
With help from Deryn Griffiths, Maree Carroll, Nikeeth Ramanathan and Steph Chong

*with some support for pandas

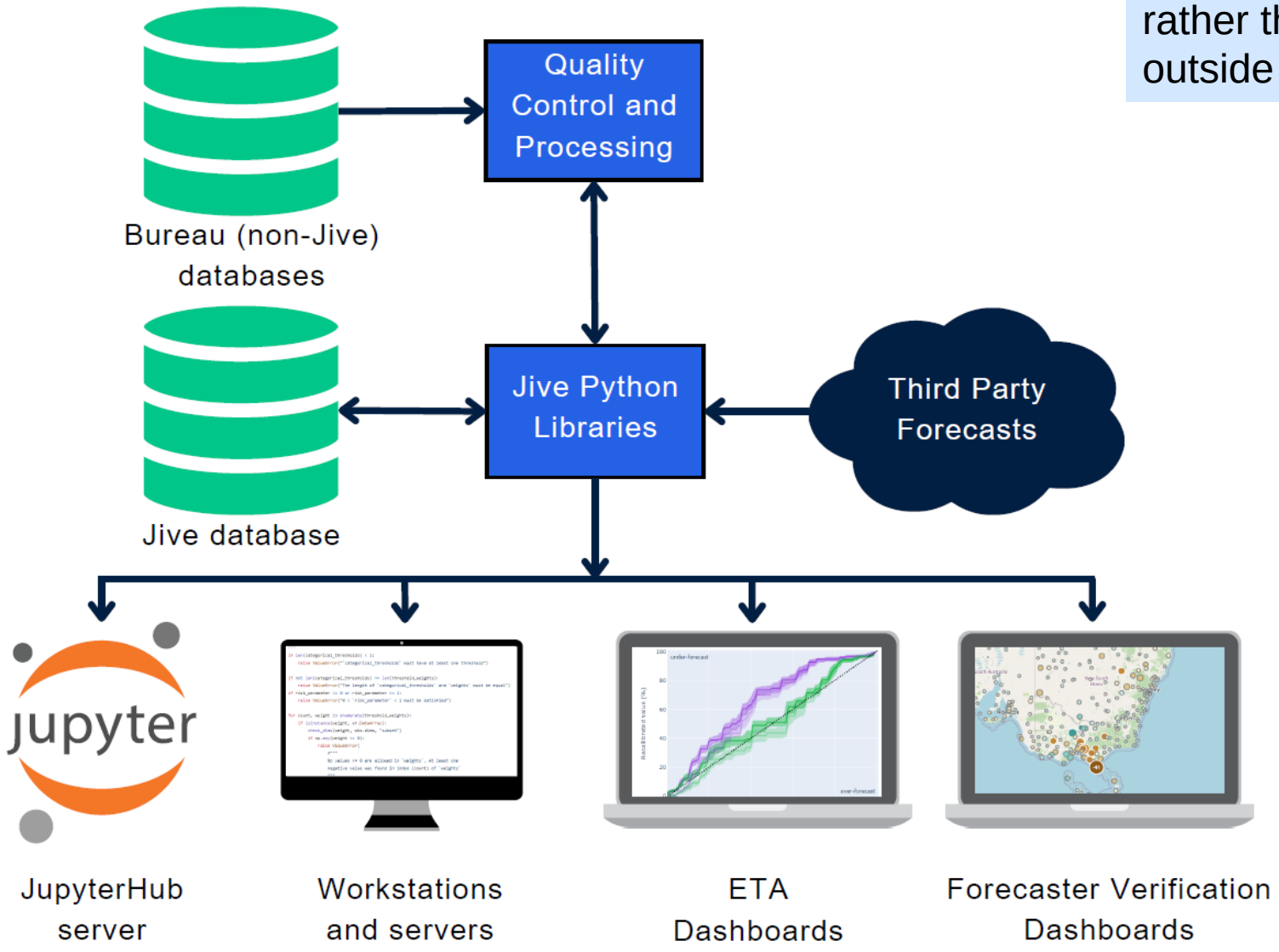
Background


Need for an improved verification tool for evaluating public weather forecasts
Research project to build this new system called Jive
Jive becomes operational
2015
2022

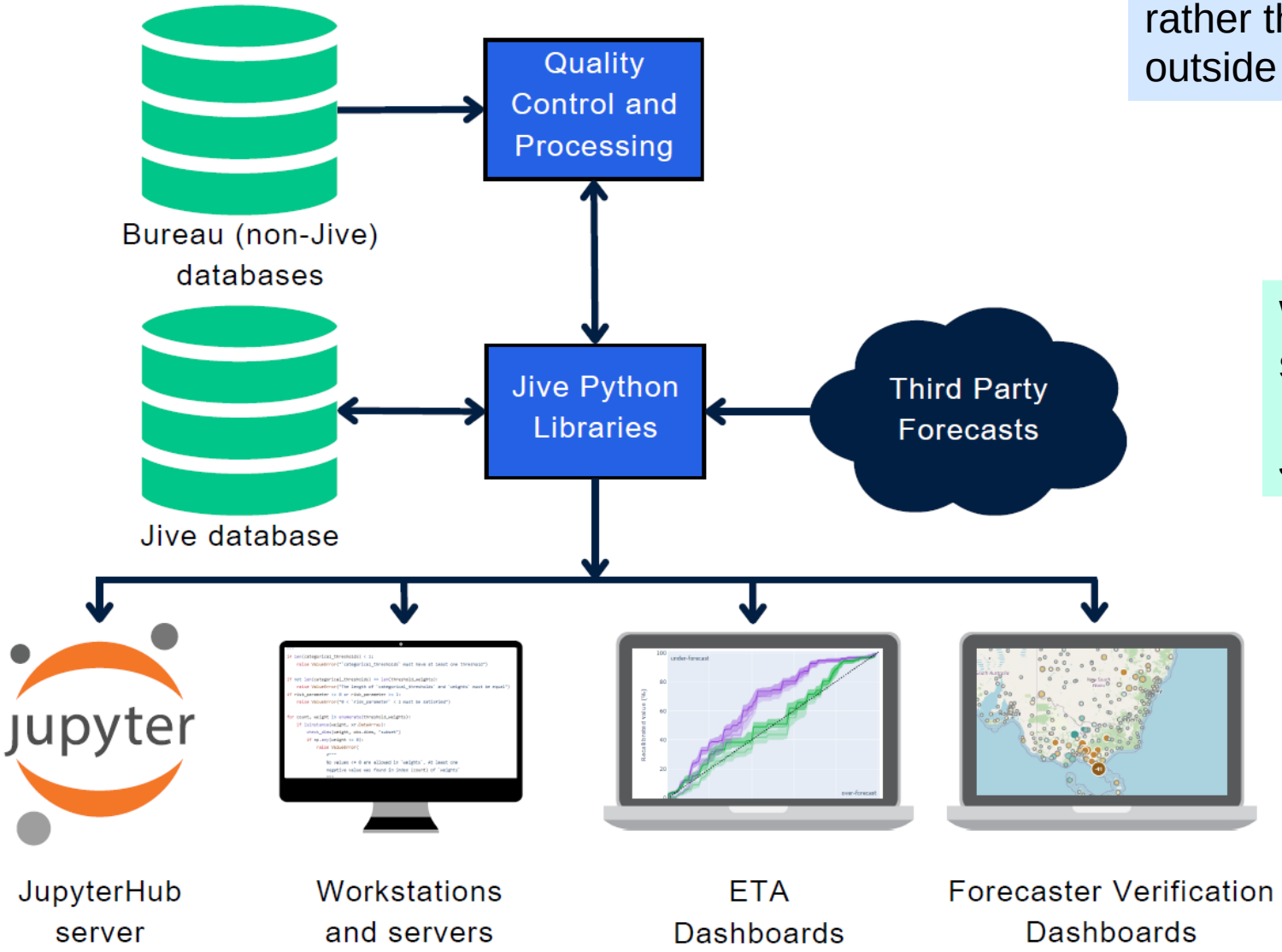
Jive became operational in 2022



People also wanted to use the metrics code rather than the whole system, including outside the Bureau's internal network.



People also wanted to use the metrics code rather than the whole system, including outside the Bureau's internal network.



We got approval for the first open source software in the Bureau!

Scores

What is scores

A flexible, lightweight xarray-based python package for forecast verification calculations.

What it is not:

It is not a complete verification system with a database, plotting tools and dashboards etc

The focus is:

- Verification scores
- Statistical tests
- Preprocessing

Why might you be interested in using scores?

Ease of use: lightweight, flexible, scalable

Science: Novel and familiar metrics

Open source: we encourage contributions

```
from scores.categorical import firm
firm(fcst, obs, risk_parameter=0.7, categorical_thresholds=[34, 48], weights=[2, 1])
```

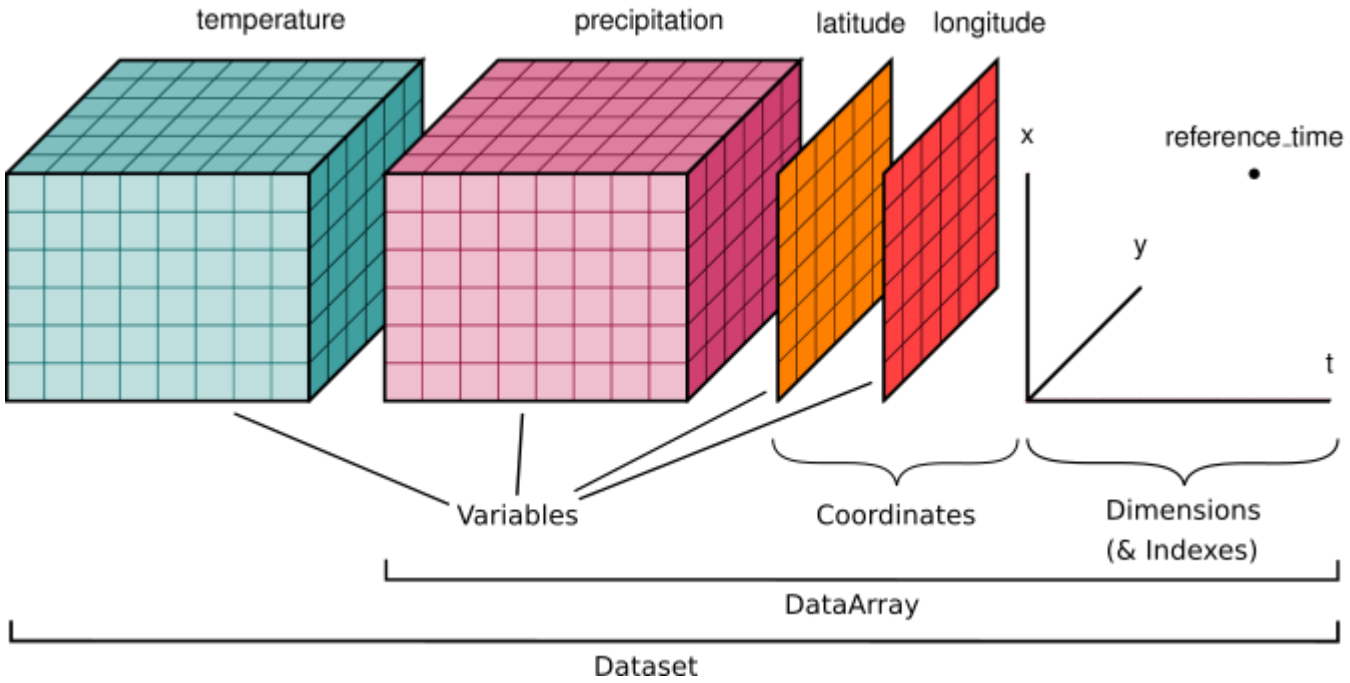

Current and upcoming metrics and tools

Continuous	Probability	Categorical	Statistical tests	Processing
• MAE • MSE • RMSE • **Flip-Flop Index** • Quantile Score • **Isotonic Regression** • Correlation Coefficient • Additive/Multiplicative Bias • **Murphy Diagrams**	• CRPS for CDF • **twCRPS for CDF** • CRPS for ensemble • ROC • Brier Score	• **FIRM** • Binary contingency table metrics	• **Diebold Mariano**	• Matching data • Discretisation • **CDF/ensemble handling tools**

Continuous	Probability	Statistical tests	Spatial
• **Threshold weighted scores** • LEPS • **Consistent scores for Huber, expectile, and quantile functionals** • NSE • Percent within X • Spearman rank correlation	• **PIT** • REV • Brier score components • quantile interval score	• Block bootstrapping • Student's t-test	• FSS

Bold metrics indicate something novel

Scores – a verification tool based on xarray



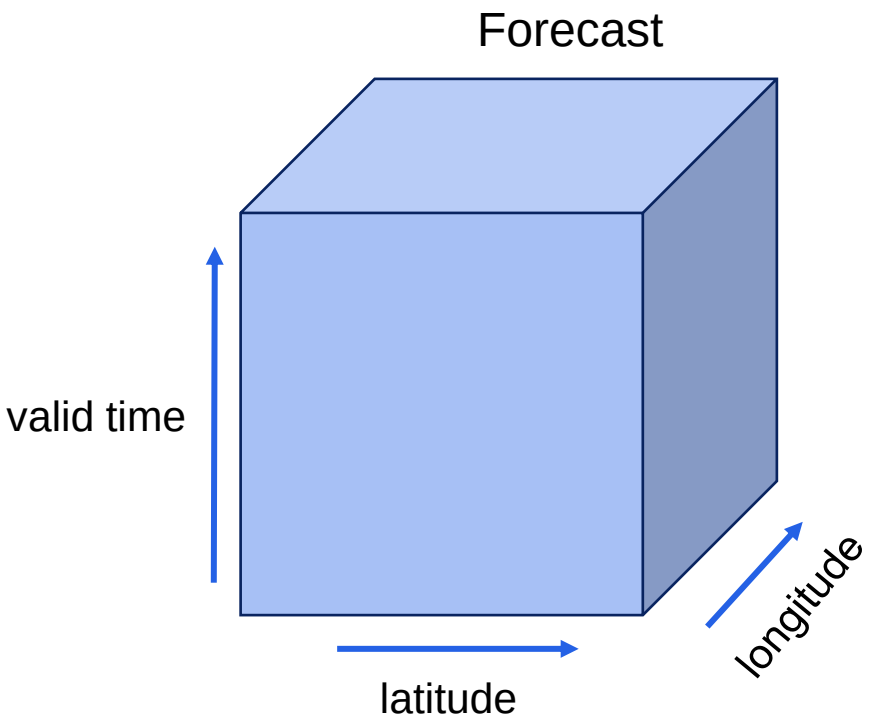
Xarray allows us to do verification on N-dimensional labelled arrays.

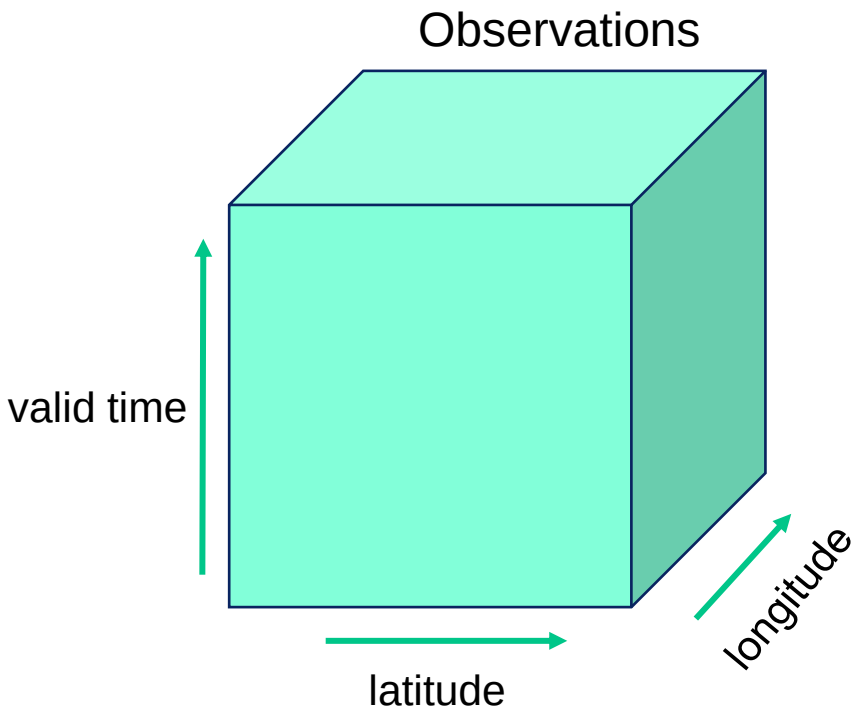
- A timeseries for a wind speed forecast at Cape Town Airport = 1D.
- An NWP forecast over South Africa with (x, y, z, time, lead time, ensemble member) = 6D.

It follows the netCDF structure, but cf-compliance is optional.

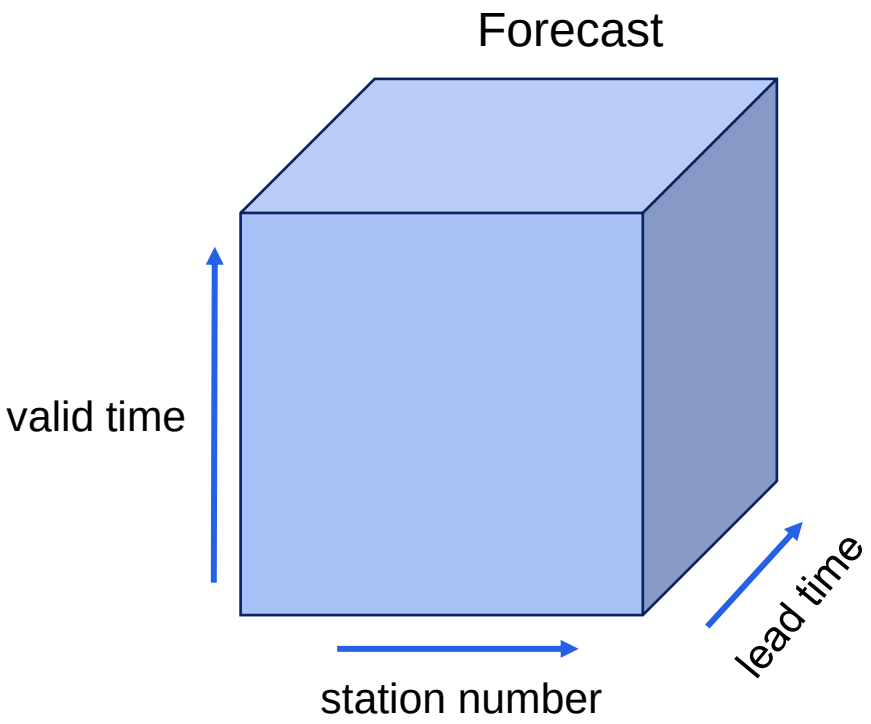
It allows us to work with netCDF, Grib, HDF5, Zarr data. Iris data can easily be converted.

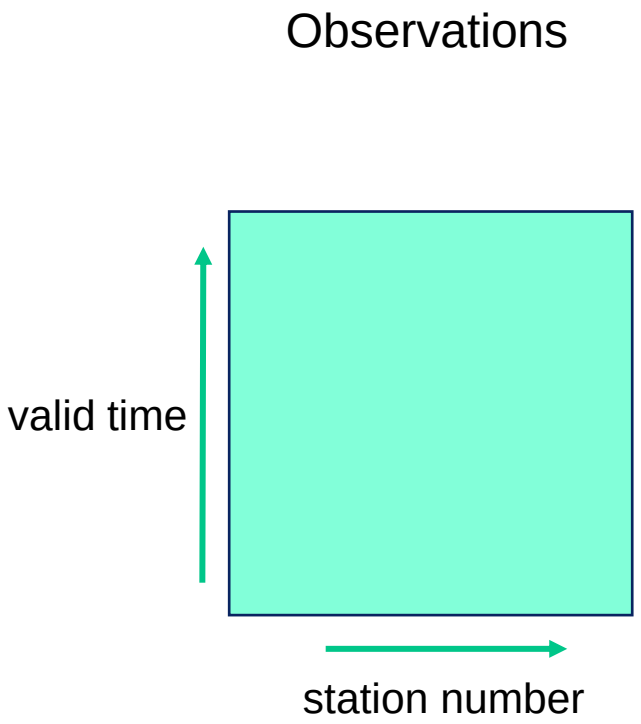
Data with the same dimensions



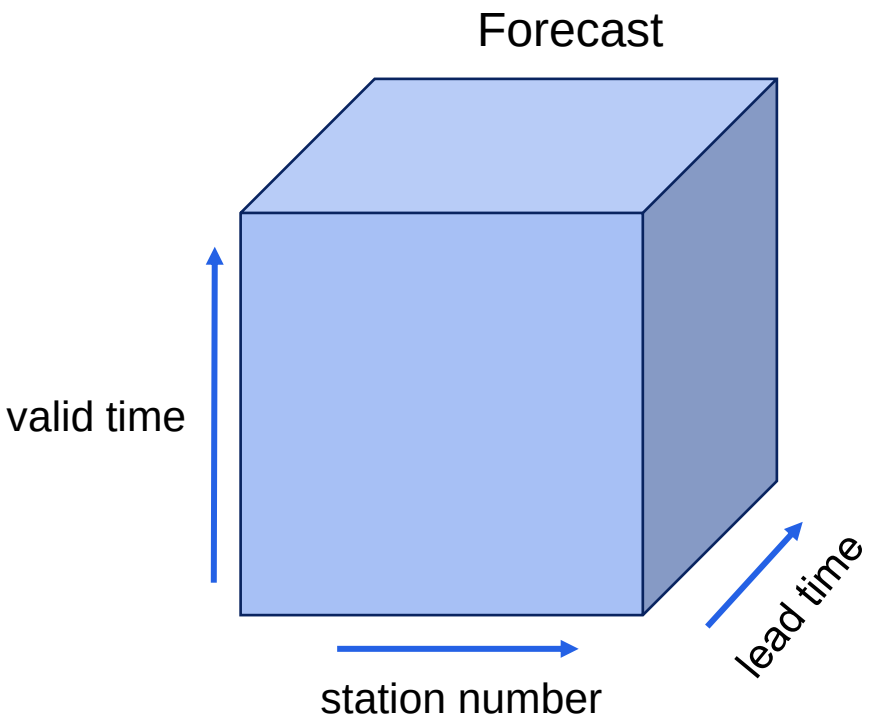


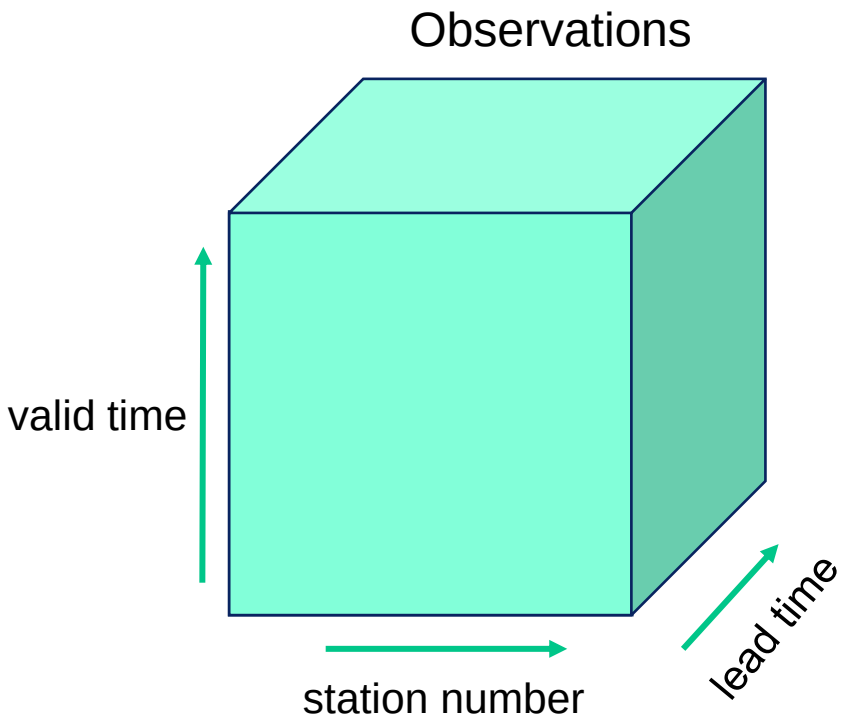
Data with different dimensions





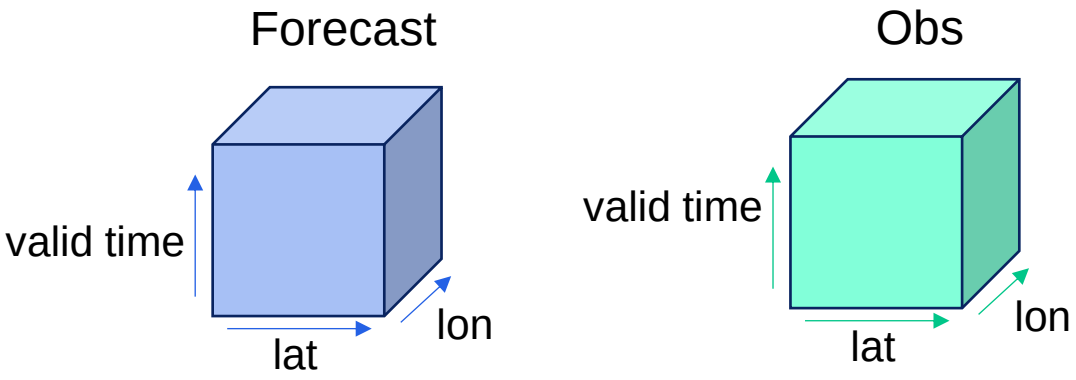
Data with different dimensions





2D array is broadcast to be 3D

Working with scores

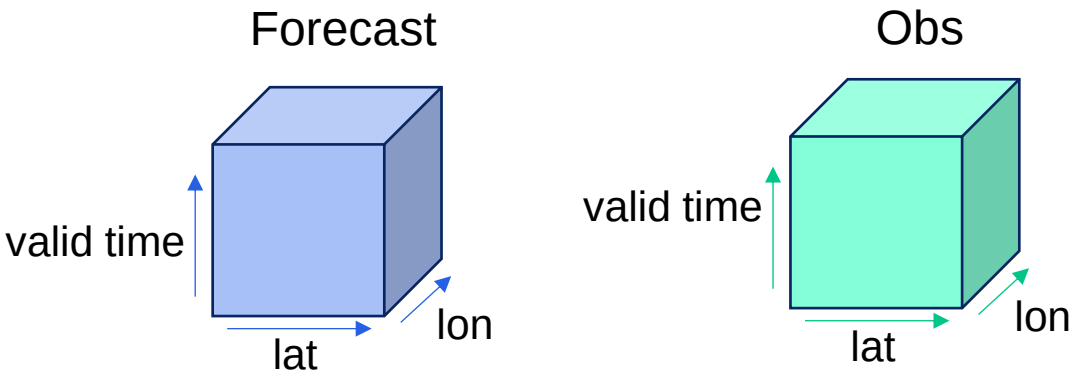



```
from scores.probability import brier_score

brier_score(forecast, observations)
```

Output -> A single value

Working with scores

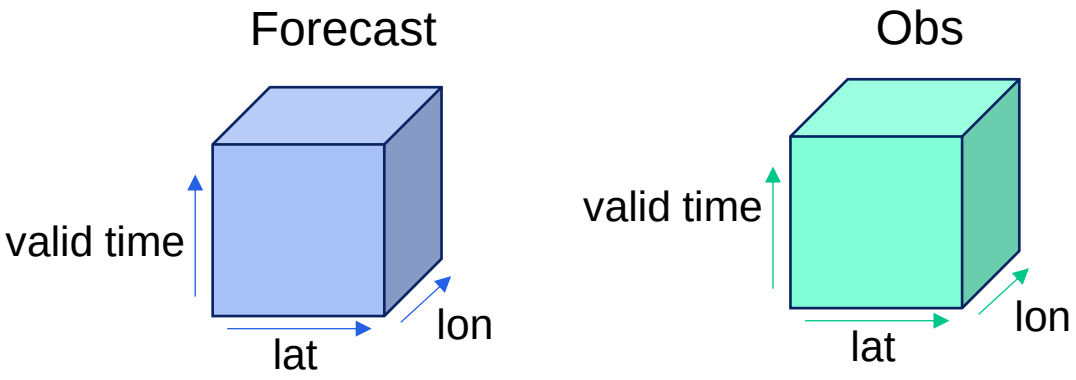



```python
from scores.probability import brier_score

brier_score(forecast, observations, preserve_dims="valid_time")
```

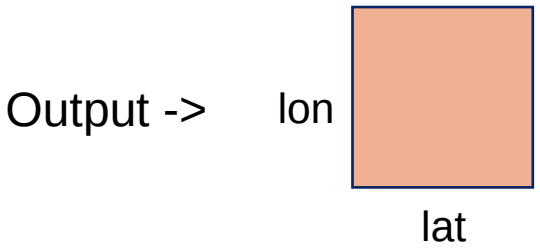
Output ->

Working with scores

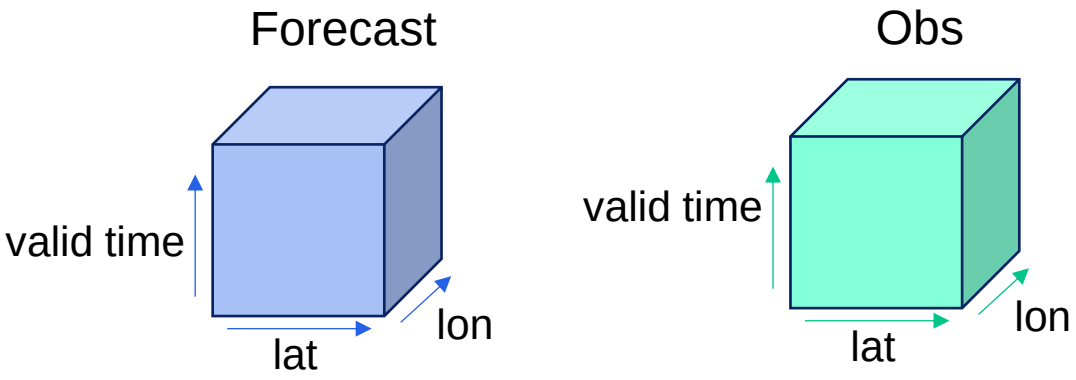



```python
from scores.probability import brier_score

brier_score(forecast, observations, reduce_dims="valid_time")
```



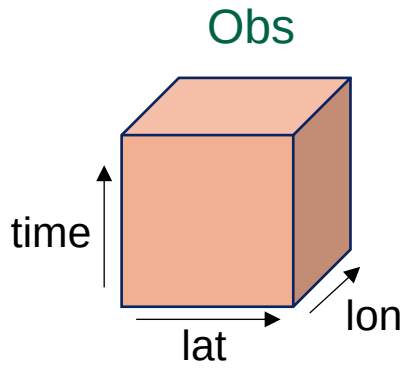
Working with scores




```python
from scores.probability import brier_score

brier_score(forecast, observations, preserve_dims="all")
```

Output ->



What if…

Your data is too big to fit into memory

and/or

you want parallelise your code on HPC?

Dask

```
forecast = xr.open_mfdataset("fcst_*.nc")
```

	Array	Chunk
Bytes	23.84 GiB	268.29 MiB
Shape	(9, 728, 685, 713)	(9, 8, 685, 713)
Dask graph	91 chunks in 249 graph layers	
Data type	float64 numpy.ndarray	

▼ Coordinates:

lead_day	(lead_day)	int64	0 1 2 3 4 5 6 7 8
y	(y)	float64	-5.557e+06 -5.55e+06 ... -1.021e+06
x	(x)	float64	-2.539e+06 -2.533e+06 ... 2.181e+06
valid_start	(valid_start)	datetime64[ns]	2023-11-30T15:00:00 ... 2024-02-...

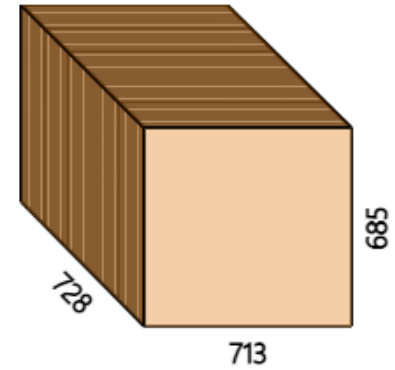
Data organised in chunks – not brought into memory yet

Dask

```
forecast = xr.open_mfdataset("fcst_*.nc")
```

```
bs = brier_score(forecast, observations)
```

	Array	Chunk
Bytes	23.84 GiB	268.29 MiB
Shape	(9, 728, 685, 713)	(9, 8, 685, 713)
Dask graph	91 chunks in 249 graph layers	
Data type	float64 numpy.ndarray	



▼ Coordinates:

lead_day	(lead_day)	int64	0 1 2 3 4 5 6 7 8
y	(y)	float64	-5.557e+06 -5.55e+06 ... -1.021e+06
x	(x)	float64	-2.539e+06 -2.533e+06 ... 2.181e+06
valid_start	(valid_start)	datetime64[ns]	2023-11-30T15:00:00 ... 2024-02-...

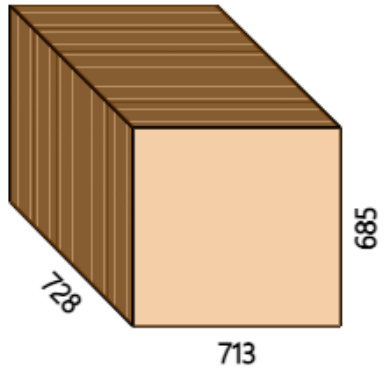
Produces "task graph" for the computation

Data organised in chunks – not brought into memory yet

Dask

```
forecast = xr.open_mfdataset("fcst_*.nc")
```

	Array	Chunk
Bytes	23.84 GiB	268.29 MiB
Shape	(9, 728, 685, 713)	(9, 8, 685, 713)
Dask graph	91 chunks in 249 graph layers	
Data type	float64 numpy.ndarray	



▼ Coordinates:

lead_day	(lead_day)	int64	0 1 2 3 4 5 6 7 8
y	(y)	float64	-5.557e+06 -5.55e+06 ... -1.021e+06
x	(x)	float64	-2.539e+06 -2.533e+06 ... 2.181e+06
valid_start	(valid_start)	datetime64[ns]	2023-11-30T15:00:00 ... 2024-02-...

Data organised in chunks – not brought into memory yet

```
bs = brier_score(forecast, observations)
```

Produces "task graph" for the computation

```
bs.compute()
```

Does computation

Output -> A single value

Alternatively save data to disk rather than creating an in-memory object of the output

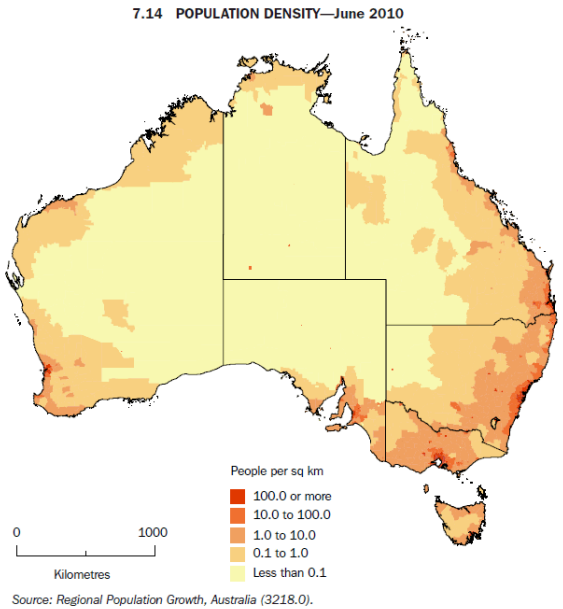
Other capabilities

Angular data

```
mae(fcst, obs, is_angular=True)
```

Weights

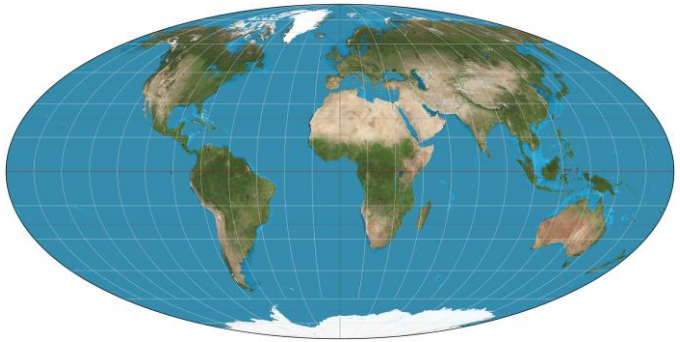
```
mae(fcst, obs, weights=weights)
```



Map projections

Use xarray's inbuilt capability

Use rioxarray for reprojection



Tutorials

Murphy Diagrams

The Murphy diagram is a powerful tool for understanding how forecast performance differs for various user decision thresholds (Ehm et al., 2016).

Given many forecasts, the Murphy diagram shows the mean score for as a function of potential user decision thresholds θ. It uses the relevant elementary score for the forecast, be it an

- expectile (e.g. mean)
- quantile (e.g. median)
- Huber quantile, or
- probabilistic forecast of a binary outcome.

For information about elementary scores, see the end of this tutorial.

We'll use `scores` to create a Murphy diagram comparing two sets of synthetic temperature forecast data.

[1]:
```
from scores.continuous import murphy_score, murphy_thetas, mse
import numpy as np
import xarray as xr
from scipy.stats import skewnorm
import matplotlib.pyplot as plt
import plotly.express as px

np.random.seed(100)
```

[2]:
```
# Read the doc string for the murphy_score
# help(murphy_score)
```

[4]:
```
# Read the doc string for murphy_thetas
# help(murphy_thetas)
```

[5]:
```
# Generate some synthetic observations between 0 and 40 repesenting temperature in Celsius
N = 1000
obs = xr.DataArray(data=40 * np.random.random(N), dims=["time"], coords={"time": np.arange(0, N)}

# Generate synthetic forecasts by adding noise to each observation
```

Tutorials

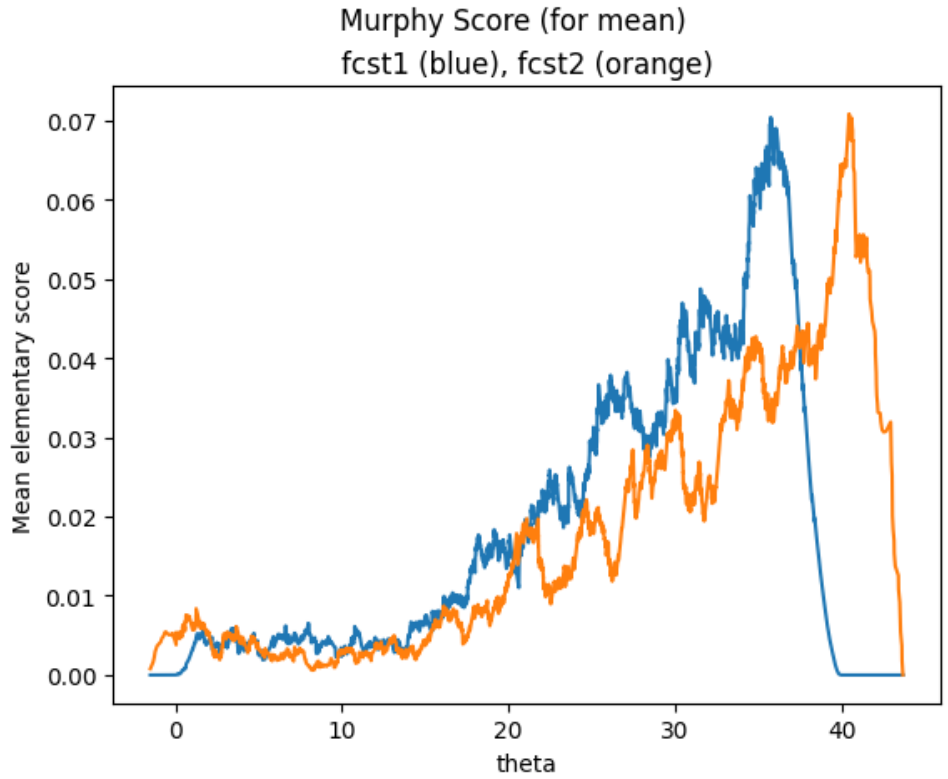
[8]:
```
# Generate a list of thresholds of interest
thetas = murphy_thetas([fcst1, fcst2], obs, "expectile")

# Calculate the average elementary score for the mean (0.5 expectile) for each threshold theta
ms1 = murphy_score(fcst1, obs, thetas, functional="expectile", alpha=0.5)
ms2 = murphy_score(fcst2, obs, thetas, functional="expectile", alpha=0.5)

# Rename date variable for plotting
ms1 = ms1.rename({"total": "Mean elementary score"})
ms2 = ms2.rename({"total": "Mean elementary score"})

# Plot the results
ms1["Mean elementary score"].plot()
ms2["Mean elementary score"].plot()
plt.title('fcst1 (blue), fcst2 (orange)')
plt.suptitle('Murphy Score (for mean)')
```

[8]: `Text(0.5, 0.98, 'Murphy Score (for mean)')`



Tutorials

An introduction to elementary scores

The MSE is a consistent scoring function for forecasting the mean (also called the expected value or 0.5 expectile) of a predicitive distribution. But did you know that there is a whole family of proper scoring rules that are consistent for the mean?

Savage (1971) showed that any function is consistent for the mean if and only if

$$S(x, y) = \phi(y) - \phi(x) - \phi'(x)(y - x)$$

where x is the forecast value and y is the observation, where ϕ is convex with subgradient ϕ'.

Ehm et al. (2016) showed that when a scoring function S satisfies the above equation it can be written as

$$S(x, y) = \int_{-\infty}^{\infty} S_\theta(x, y) dH(\theta)$$

where H is a non-negative measure, then

$$S_\theta(x, y) = \begin{cases} |y - \theta|, & \min(x, y) \leq \theta < \max(x, y) \\ 0, & \text{otherwise} \end{cases}$$

is the elementary scoring function.

This means we can construct an infinite amount of different scoring rules that are consistent for forecasting the mean by varying H. All will be optimized by predicting the mean, but some will give more weighting to certain thresholds.

The Murphy diagram shows the average elementary score S_θ plotted across all potential user decision thresholds θ.

Different elemetary scoring functions are relevant to

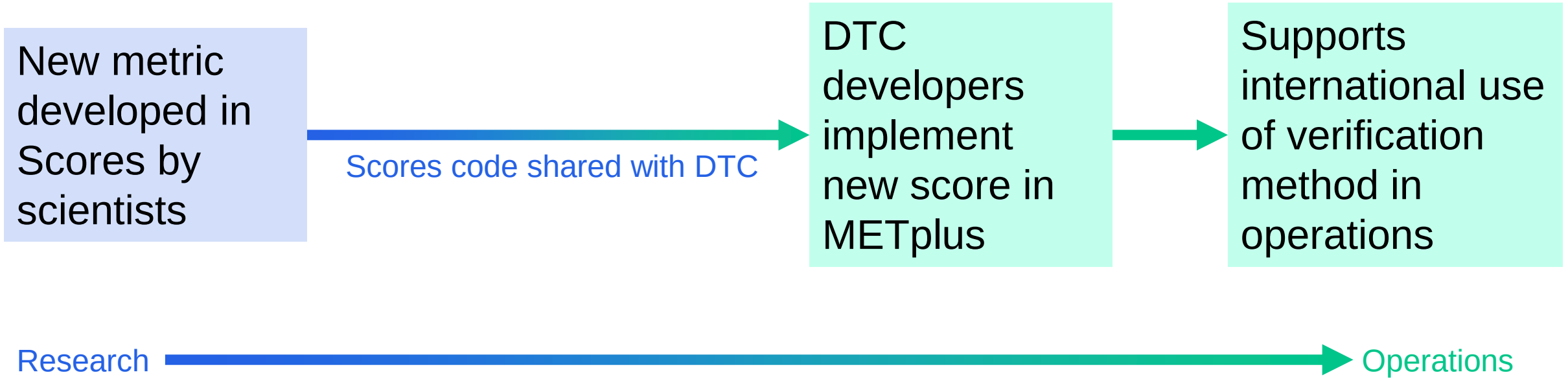
Scores/Jive and METplus in the Bureau

One system can't meet every need – we focus on two system to capture most of our needs.

Scores strengths	METplus strengths
Verification of post-processed forecasts expressed as CDFs, quantile, or mean-valued forecasts	Verification of NWP – Particularly spatial metrics (MODE, intensity scale, HiRA, distance mapping methods)
Scientists can easily add new metrics in Python	Fair comparison between Met agencies (e.g., sharing config files to evaluate NWP)
More flexible with data inputs	Large international uptake
A lightweight tool that can be incorporated into other Python packages (e.g., `from scores.continuous import mse`)	Tested with LFRic grids
Option to scale with Dask	TC verification

Scores and METplus in the Bureau

An opportunity to support the R2O process

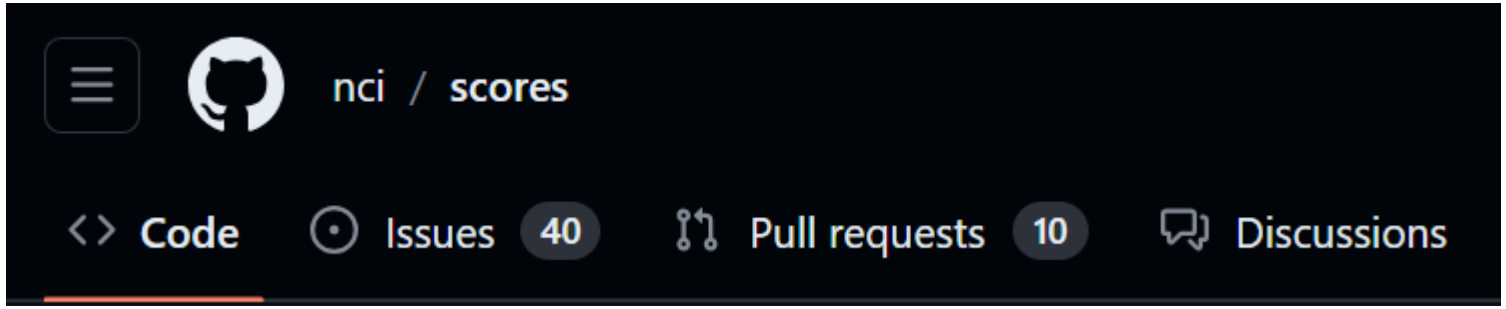


What's next?

- Add more scores
- Journal of Open Source Software (JOSS) paper
- Expand Pandas support
- Leverage improvements and new features in xarray (e.g. unstructured grids?)
- Increase support for machine learning library integration

How to get involved?

https://github.com/nci/scores



- Add new metrics, tools, statistical tests
- Get involved with science/code reviews
- Improve the tutorial notebooks

Summary – why scores?

- Lightweight
- Flexible
- Scalable
- Developed by software developers and scientists
- Can be used standalone or imported into other software packages
- Familiar and novel metrics
- Open to the international community to contribute

Please contribute!

Github: https://github.com/nci/scores
Documentation: https://scores.readthedocs.io/en/latest/
Contact: nicholas.loveday@bom.gov.au

An image depicting scores generated by DALL·E 3