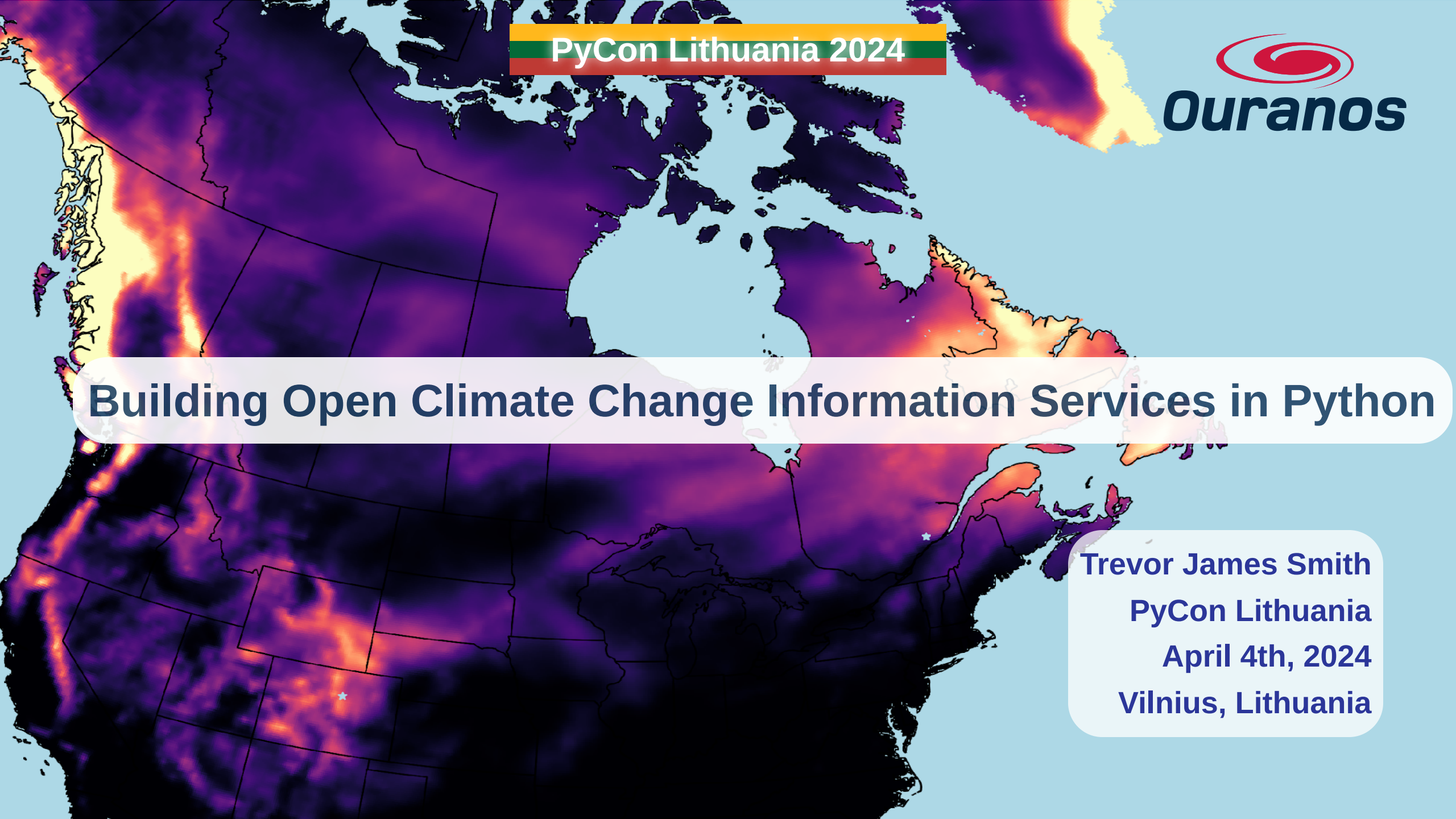


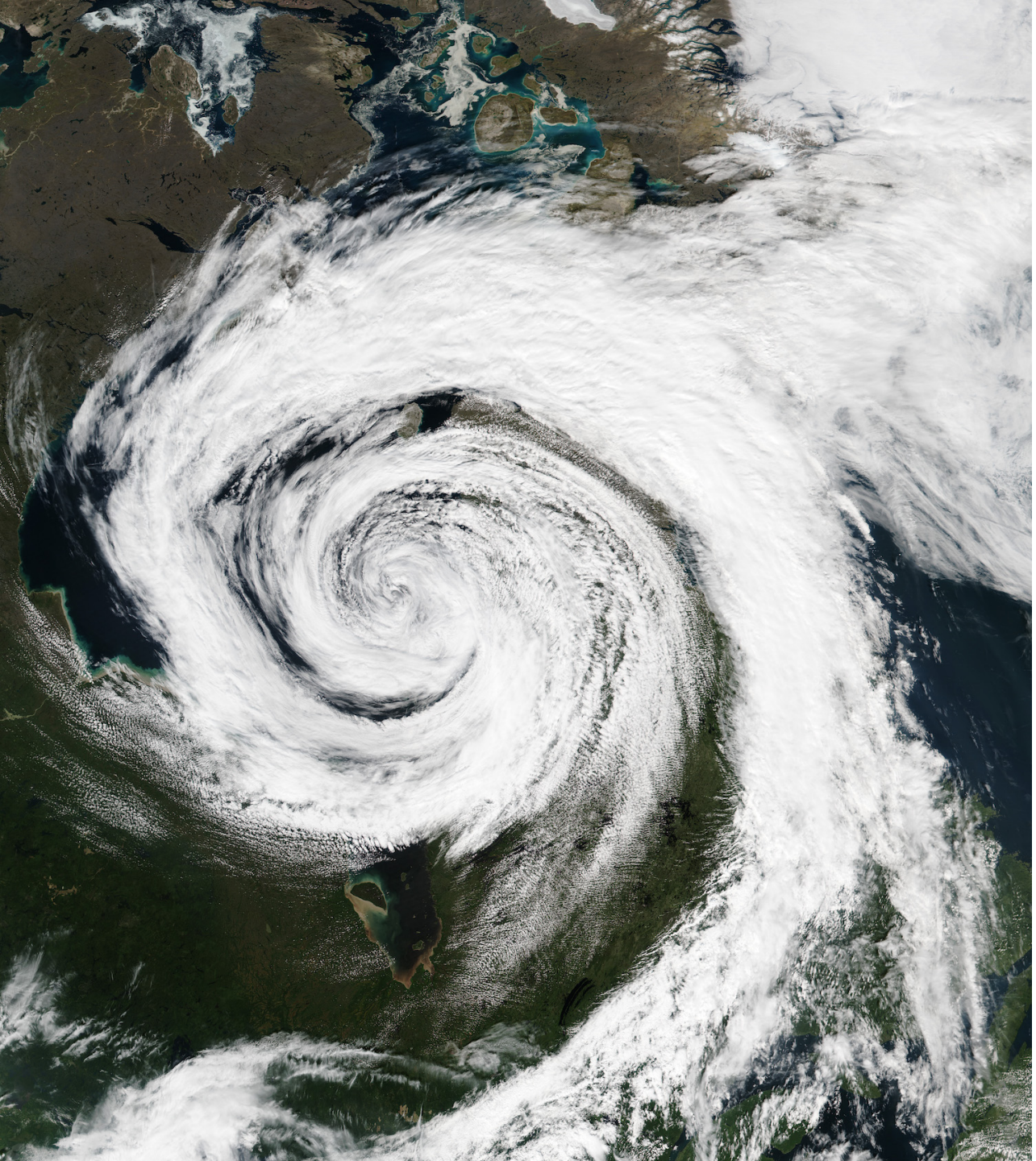
Building Open Climate Change Information Services in Python

Trevor James Smith

PyCon Lithuania

April 4th, 2024

Vilnius, Lithuania



Presentation Outline

- Who am I? / What is Ouranos?
- What's our context?
- Climate Services?
- `xclim`: climate operations
- `finch`: `xclim` as a Service
- Climate WPS Frontends
- Open Source Climate Services
- Acknowledgements

Photo: Extratropical Cyclone over Hudson Bay, Canada, August 2016. Credit: NASA Earth Observatory.

Who am I?

Trevor James Smith

github.com/Zeitsperre

Zeit@techhub.social

- Research software developer/packager/maintainer from Montréal, Québec, Canada 🇨🇦
- Studied climate change impacts on wine viticulture 🍇 in Southern Québec
- Making stuff with Python 🐍 for ~6.5 years
- Užupio Respublikos 🖐️ pilietis (nuo 2024 m.)

What is Ouranos?

- Non-profit research consortium established in 2003 in Montréal, Québec, Canada
    - Created in response to the January 1998 North American Ice Storm
- Climate Change Adaptation Planning
- Climate Model Data Producer/Provider
- **Climate Information Services**

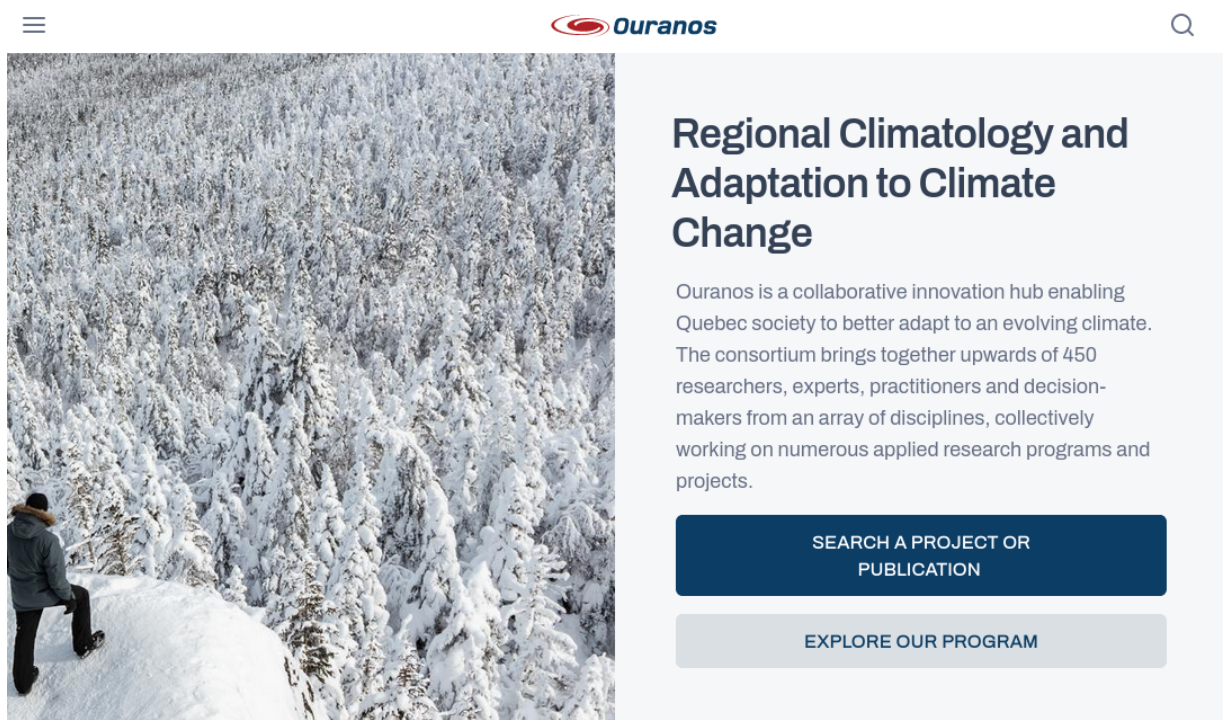
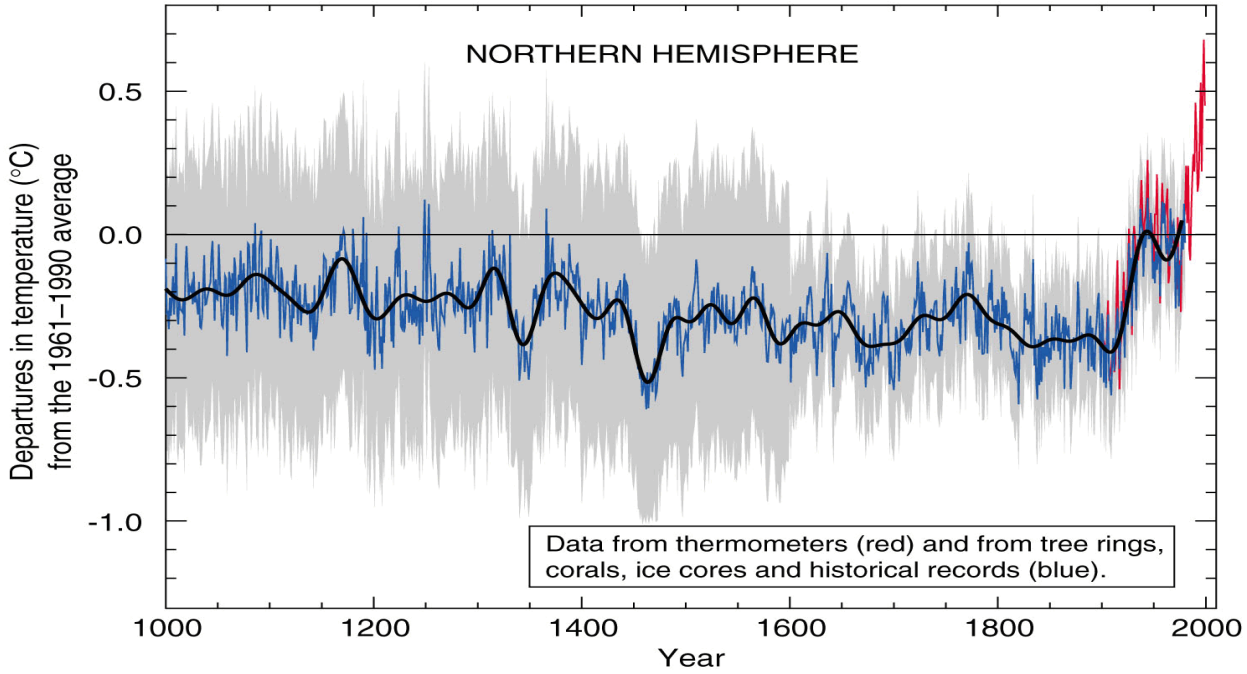
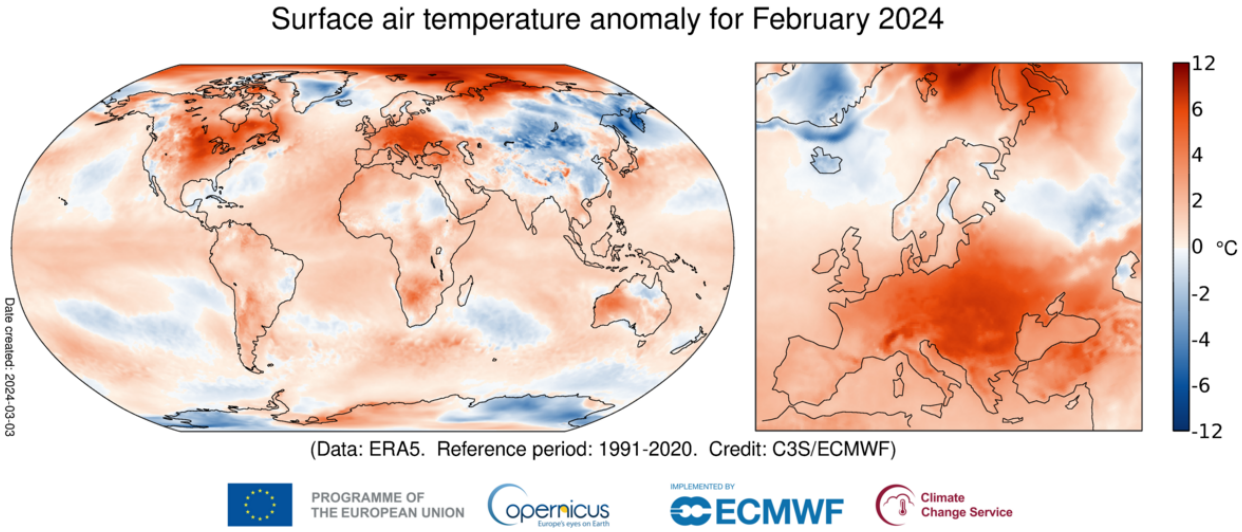


Photo credit: https://www.communitystories.ca/v2/grand-verglas-saint-jean-sur-richelieu_ice-storm/





What's the climate situation?

- Climate Change is having major impacts on Earth's environmental systems
- IPCC: **Global average temperature has increased > +1.1 °C since 1850s**.
  - **> +1.5 °C is considered to be beyond a safe limit**

What's the climate data situation?

Climate science is a "Big Data" problem

- New climate models being developed every year
- More climate simulations being produced every day
- Higher resolution input and output datasets (gridded data)
- Specialised analyses and more personalized user needs

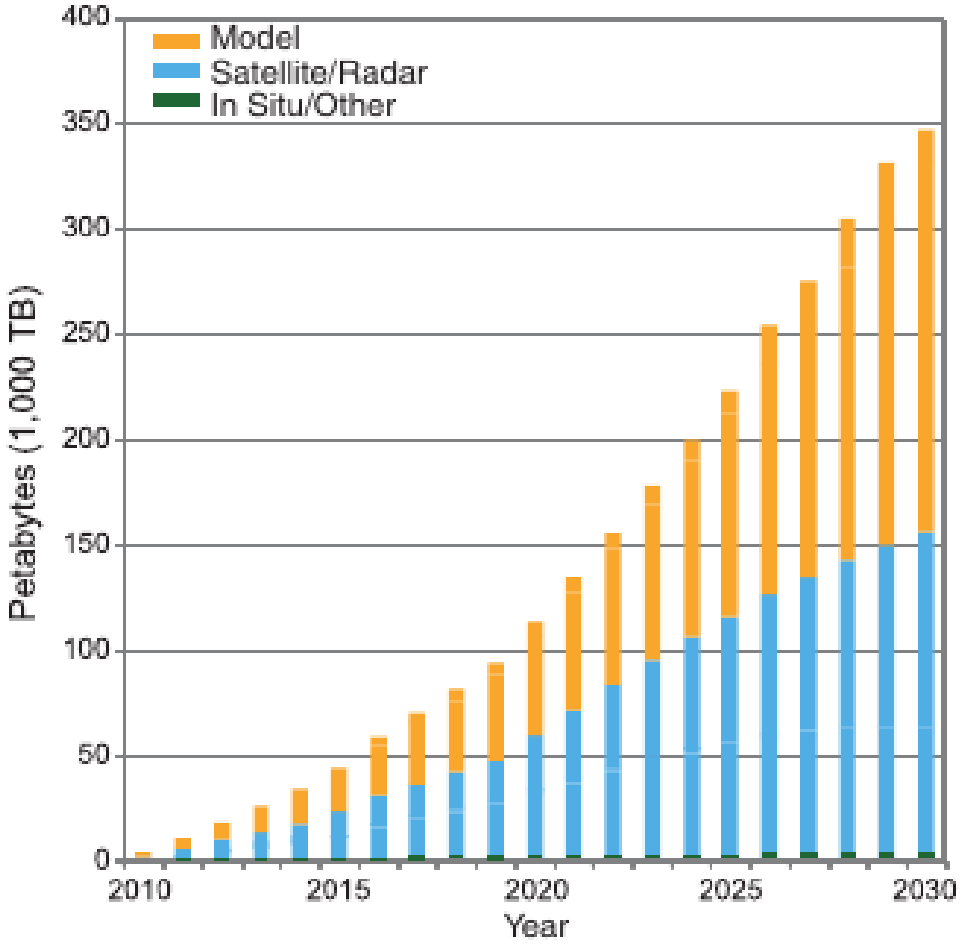


Fig. 2. The volume of worldwide climate data is expanding rapidly, creating challenges for both physical archiving and sharing, as well as for ease of access and finding what's needed, particularly if you are not a climate scientist. The figure shows the projected increase in global climate data holdings for climate models, remotely sensed data, and in situ instrumental/proxy data.

Climate Services

What do they provide?

- Tailoring objectives and information to different user needs
- Providing access to **climate information**
- Building local mitigation/adaptation capacity
- Offering training and support
- Making sense of **Big *climate* Data**

Infographic from The Canadian Centre for Climate Services (CCCS)

What information do Climate Services provide?

Climate Indicators, e.g.:

- **Hot Days** (Days with temperature >= 22 deg Celsius) 🌡️
- **Beginning / End / Length of the growing season** 🌷
- **Average seasonal rainfall** (3-Month moving average precipitation) ☔
- *Many more examples*

Planning Tools, e.g. :

- Maps 🗺️
- Point estimates at geographic locations 📈
- Gridded values 🌐
- **Not really sure what they need?** ❓ **Guidance from experts!**

Climate Services in the 2010s

- `MATLAB`-based in-house libraries (**proprietary** 💰)
  - No source code review
- Issues with data storage / access / processing 😫
  - Small team unable to meet demand
  - Lack of output data uniformity between researchers ⁉️
  - Lots of bugs 🐛 and human error 🤦
- Data analysis/requests served manually ⏳
- Software testing + data validation? Not really. 😱

Building a Climate Services library?

What are the requirements?

What does it need to perform?

- **Climate Indicators**
  - Units management
  - Metadata management
- **Ensemble statistics**;
- **Bias Adjustment**;
- **Data Quality Assurance Checks**

Implementation goals?

- **Operational** : Capable of handling very large ensembles of climate data
- **Foolproof** : Automatic verification of data and metadata validity by default
- **Extensible** : Flexibility of use and able to easily provide custom indicators, as needed

Is there Python in this talk?

- **Yes**

Why build a Climate Services library in Python?

- **Robust**, **trustworthy**, and **fast** scientific Python libraries
- Python's Readability / Reviewability (**Peer Review**)
- Growing demand for climate services / products
  - **Let the users help themselves**
- *The timing was right*
  - Internal and external demand for common tools
- **Less time writing code, more time spent doing research**

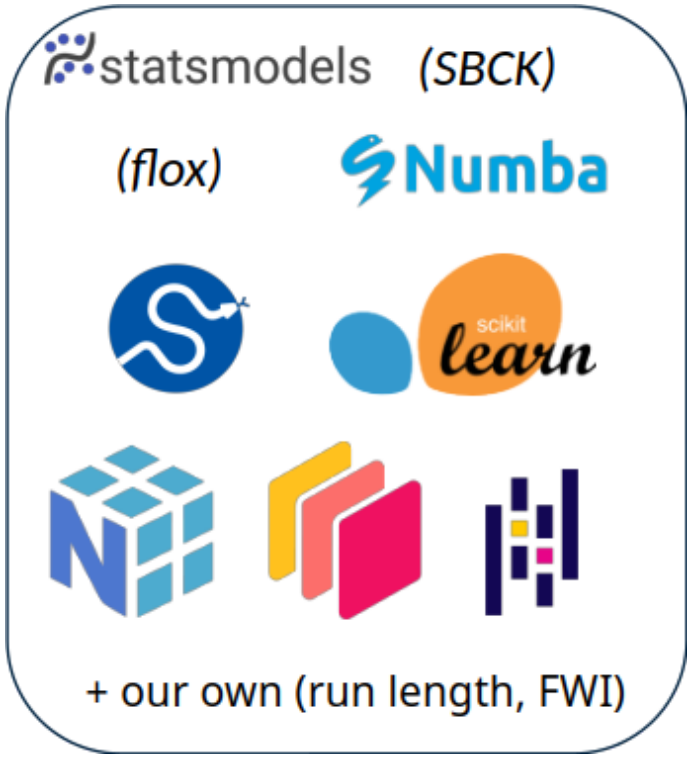
xclim: Climate services library

Versions	pypi v0.48.2 · conda-forge v0.48.2 · python 3.9 | 3.10 | 3.11 | 3.12
Documentation and Support	docs passing · GitHub Discussions
Open Source	license Apache-2.0 · openssf scorecard 8.5 · DOI 10.5281/zenodo.10710942 · PyOpenSci Peer Reviewed · JOSS 10.21105/joss.05415
Coding Standards	code style black · Ruff · pre-commit.ci passed · openssf best practices in progress 99% · license scan passing
Development Status	repo status Active · xclim Testing Suite passing · coverage 90%

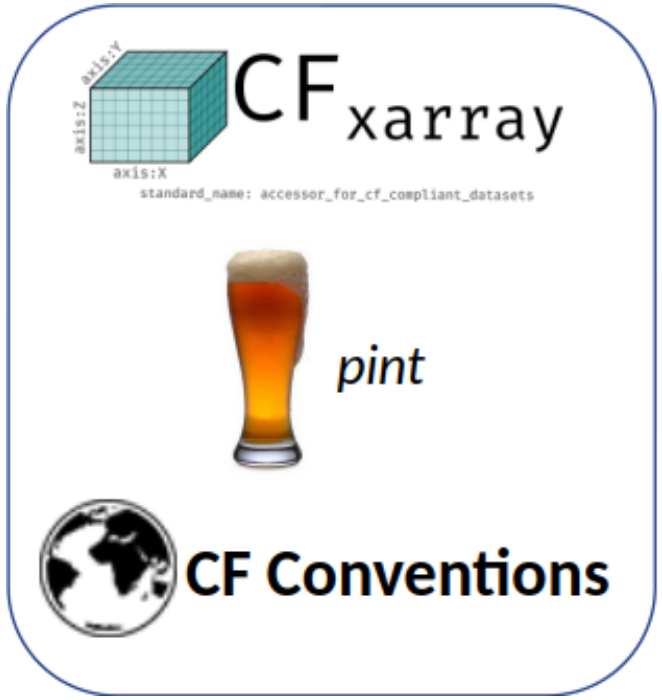
xclim is an operational Python library for climate services, providing numerous climate-related indicator tools with an extensible framework for constructing custom climate indicators, statistical downscaling and bias adjustment of climate model simulations, as well as climate model ensemble analysis tools.

How did we build Xclim?

- **Data Structure**



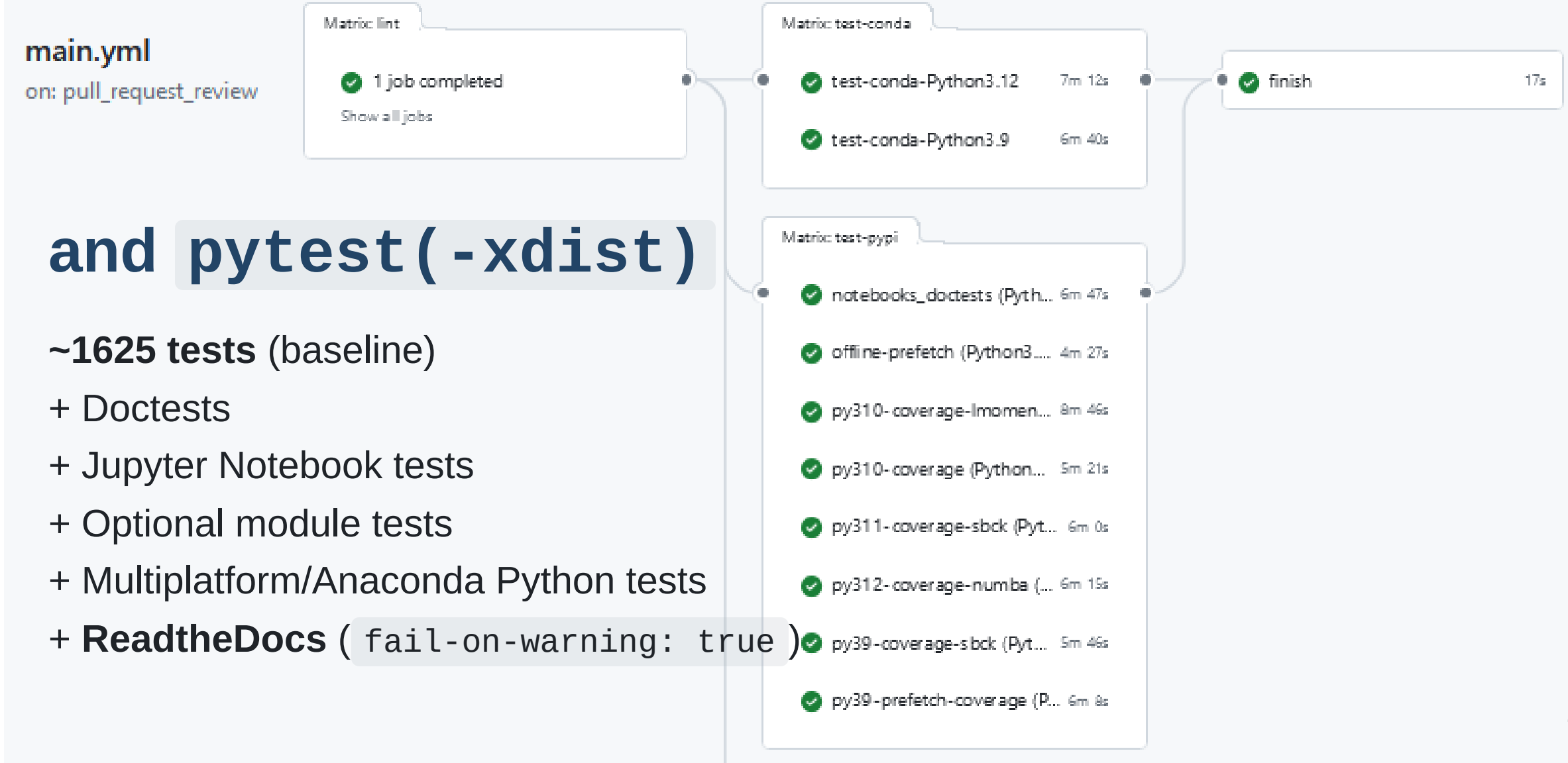
- **Algorithms**



- **Data and Metdata Conventions**

Triggered via pull request review last week | Status | Total duration | Artifacts

Zeitsperre submitted 6b4cec0 | **Success** | **10m 41s** | –



and `pytest(-xdist)`

~1625 tests (baseline)

\+ Doctests

\+ Jupyter Notebook tests

\+ Optional module tests

\+ Multiplatform/Anaconda Python tests

\+ **ReadtheDocs** (`fail-on-warning: true`)

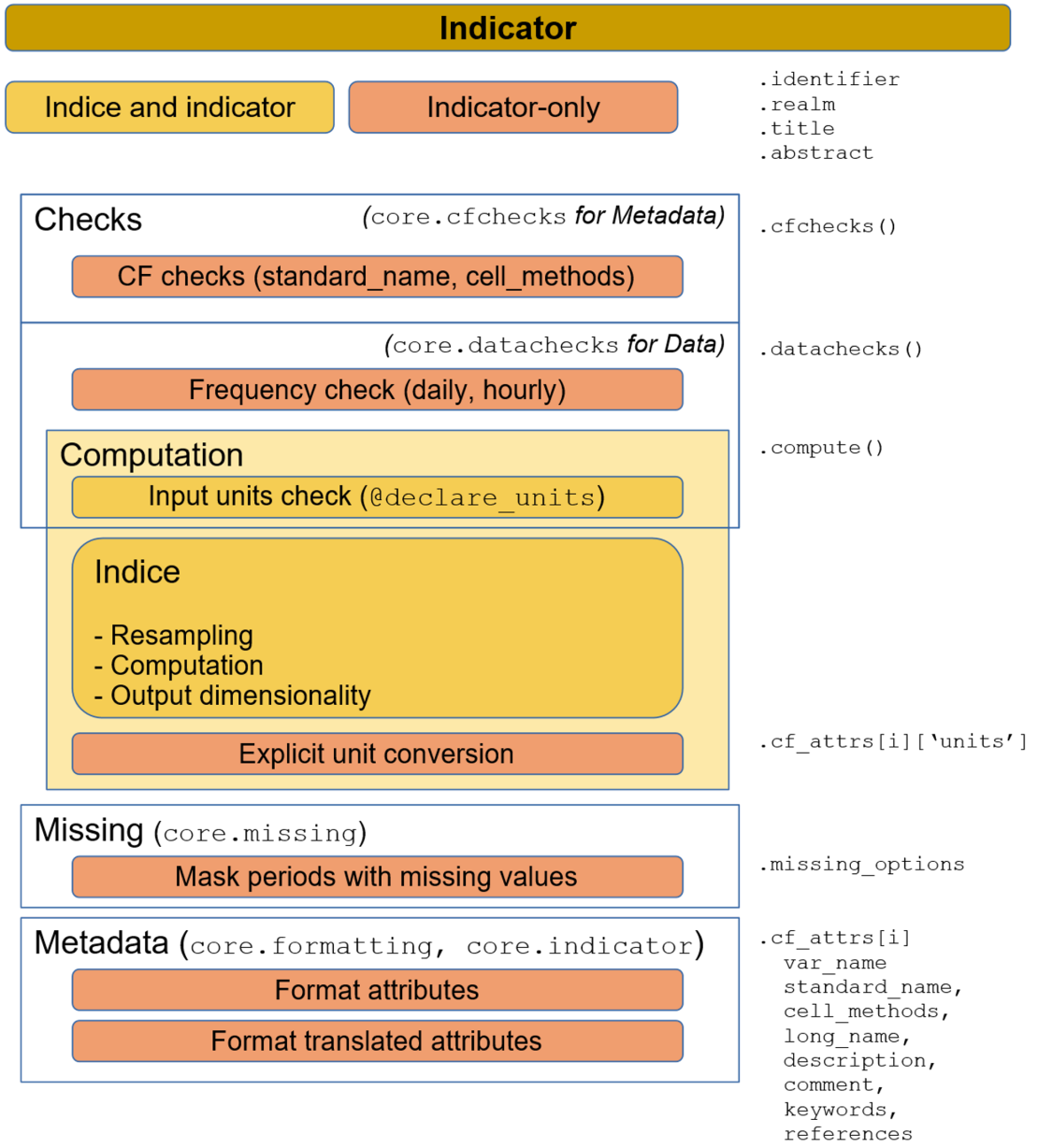
Climate Indicator Example - Average Snow Depth

```python
@declare_units(snd="[length]")
def snow_depth(
    snd: xarray.DataArray,
    freq: str = "YS",
) -> xarray.DataArray:
    """Mean of daily average snow depth.

    Resample the original daily mean snow depth series by taking the mean over each period.

    Parameters
    ----------
    snd : xarray.DataArray
        Mean daily snow depth.
    freq : str
        Resampling frequency.

    Returns
    -------
    xarray.DataArray, [same units as snd]
        The mean daily snow depth at the given time frequency
    """
    return snd.resample(time=freq).mean(dim="time").assign_attrs(units=snd.units)
```

Xclim algorithm design

Two ways of calculating indicators

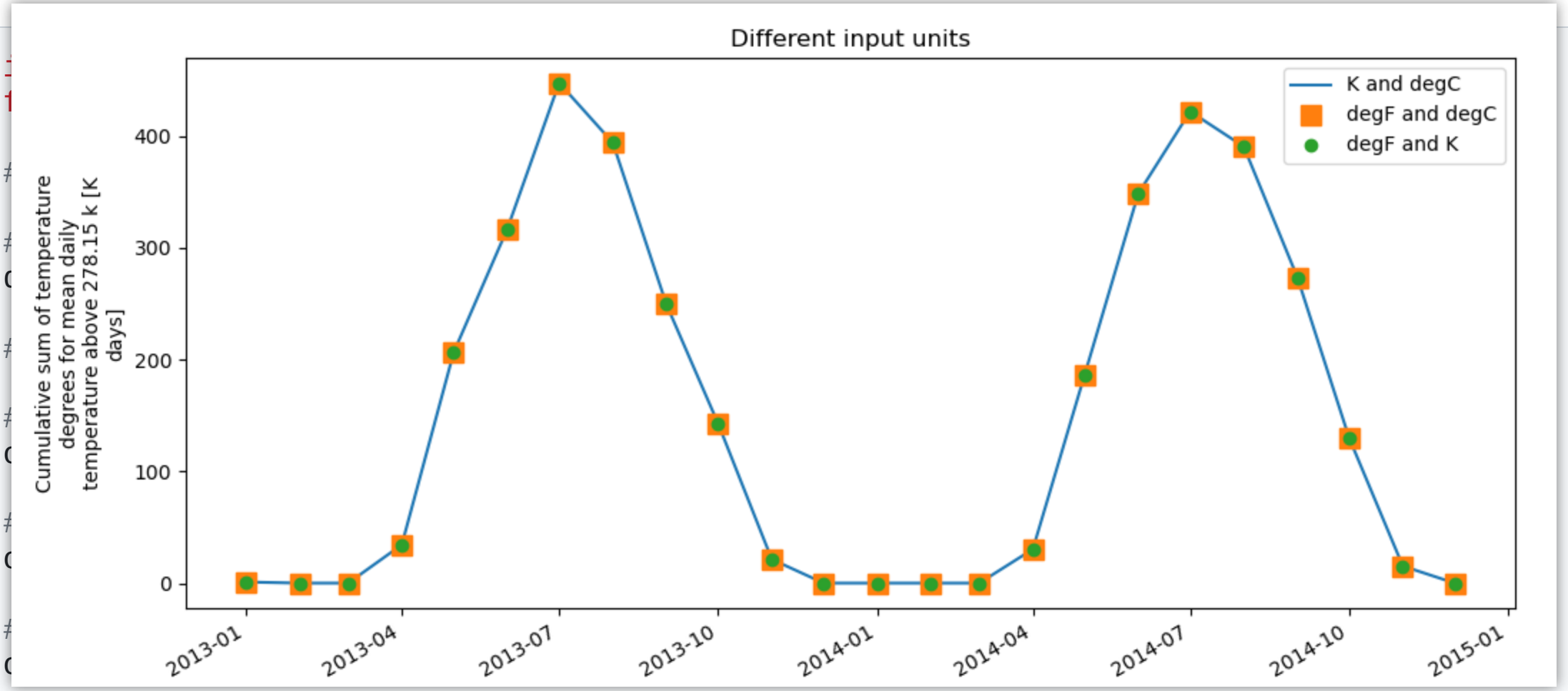
- `indicators` (**End-User API**)
  - Metadata standards checks
  - Data quality checks
  - Time frequency checks
  - Missing data-compliance
  - Calendar-compliance
- `indice` (**Core API**)
  - For users that don't care for the standards and quality checks



What does Xclim do? 🤔 Units Management

```python
import xclim
from clisops.core import subset

# Data is in Kelvin, threshold is in Celsius, and other combinations

# Extract a single point location for the example
ds_pt = subset.subset_gridpoint(ds, lon=-73, lat=44)

# Calculate indicators with different units

# Kelvin and Celsius
out1 = xclim.atmos.growing_degree_days(tas=ds_pt.tas, thresh="5 degC", freq="MS")

# Fahrenheit and Celsius
out2 = xclim.atmos.growing_degree_days(tas=ds_pt.tas_F, thresh="5 degC", freq="MS")

# Fahrenheit and Kelvin
out3 = xclim.atmos.growing_degree_days(tas=ds_pt.tas_F, thresh="278.15 K", freq="MS")
```

What does Xclim do? Units Management



What does Xclim do? □ Missing Data and Metadata Locales

```python
import xarray as xr
import xclim

ds = xr.open_dataset("my_dataset.nc")

with xclim.set_options(
    # Drop timesteps with more than 5% of missing data
    set_missing="pct", missing_options=dict(pct={"tolerance": 0.05}),

    metadata_locales=["fr"] # Add French language metadata
):
    # Calculate Annual Frost Days (days with min temperature < 0 °C)
    FD = xclim.atmos.frost_days(ds.tas, freq="YS")
```

What does Xclim do? Missing Data and Metadata Locales

```
import
import

ds = xr

with xc
    # D
    set
                                                    ),
    met
):
    # C
    FD
                                                    )
```

xarray.DataArray 'fd' (**time**: 2, **lat**: 25, **lon**: 53)

302.0 306.0 308.0 314.0 317.0 322.0 323.0 ... 0.0 0.0 0.0 0.0 0.0 0.0

▾ Coordinates:

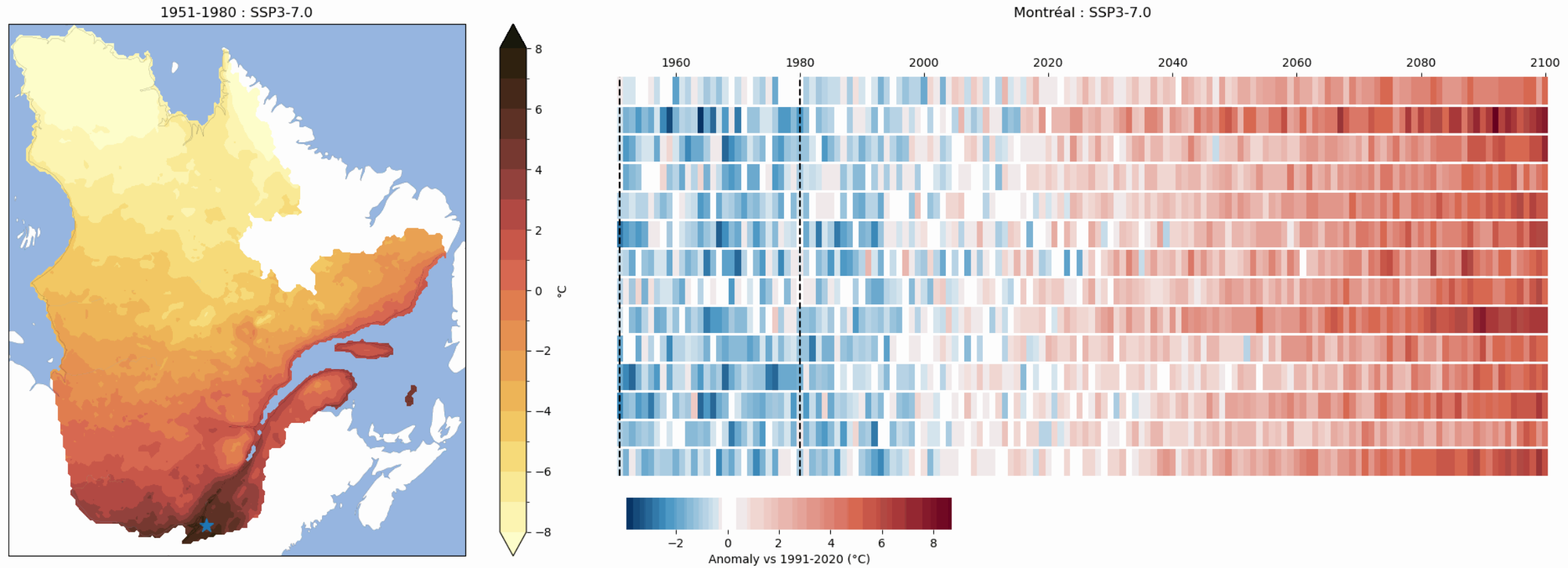
lat	(lat)	float32	75.0 72.5 70.0 ... 20.0 17.5 15.0	
lon	(lon)	float32	200.0 202.5 205.0 ... 327.5 330.0	
time	(time)	datetime64[ns]	2013-01-01 2014-01-01	

▸ Indexes: (3)

▾ Attributes:

units :	days
cell_methods :	time: sum over days
history :	[2022-12-23 11:14:55] frost_days: FROST_DAYS(tasmin=air, thresh='0 degC', freq='YS') with options check_missing=pct, missing_options={'tolerance': 0.05} - xclim version: 0.39.0
long_name :	Number of frost days (tmin < 0°c)
description :	Annual number of days where the daily minimum temperature is below 0 degc.
long_name_fr :	Nombre de jours où la température minimale quotidienne est sous 0 degC
description_fr :	Nombre annuel de jours où la température minimale quotidienne est sous 0 degC.

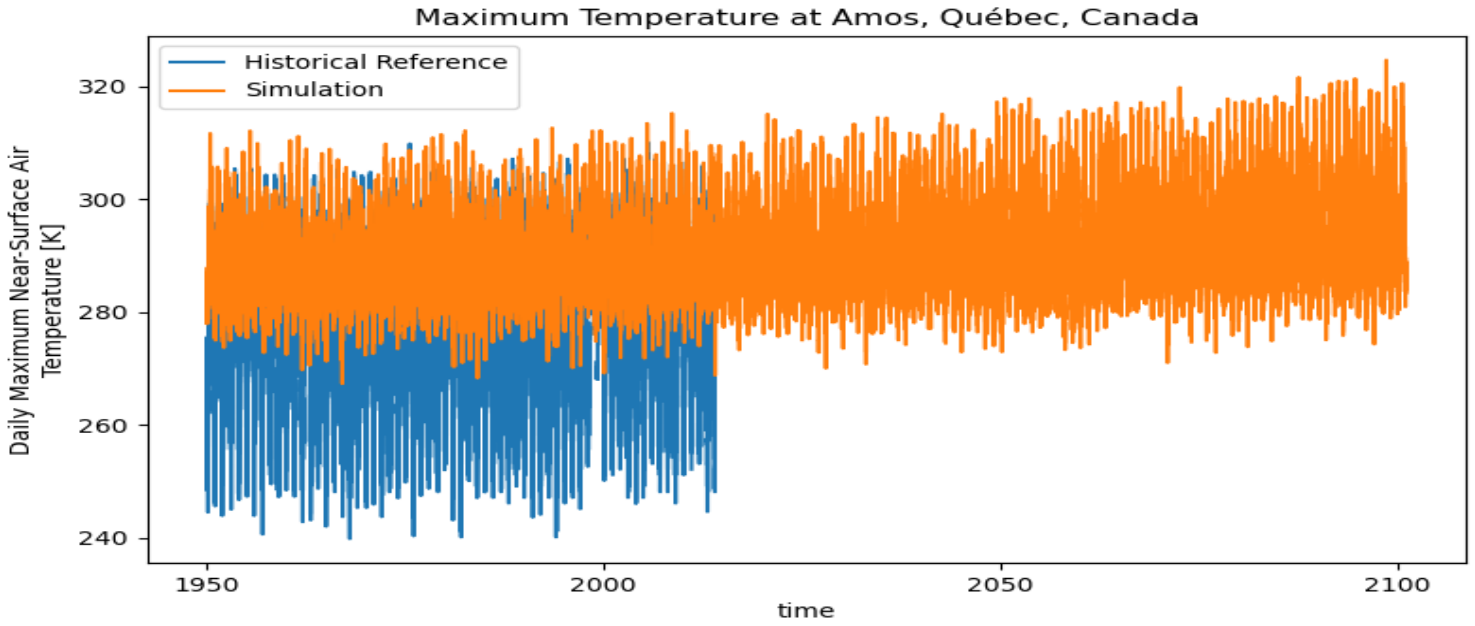
What does Xclim do ? Climate Ensemble Mean Analysis

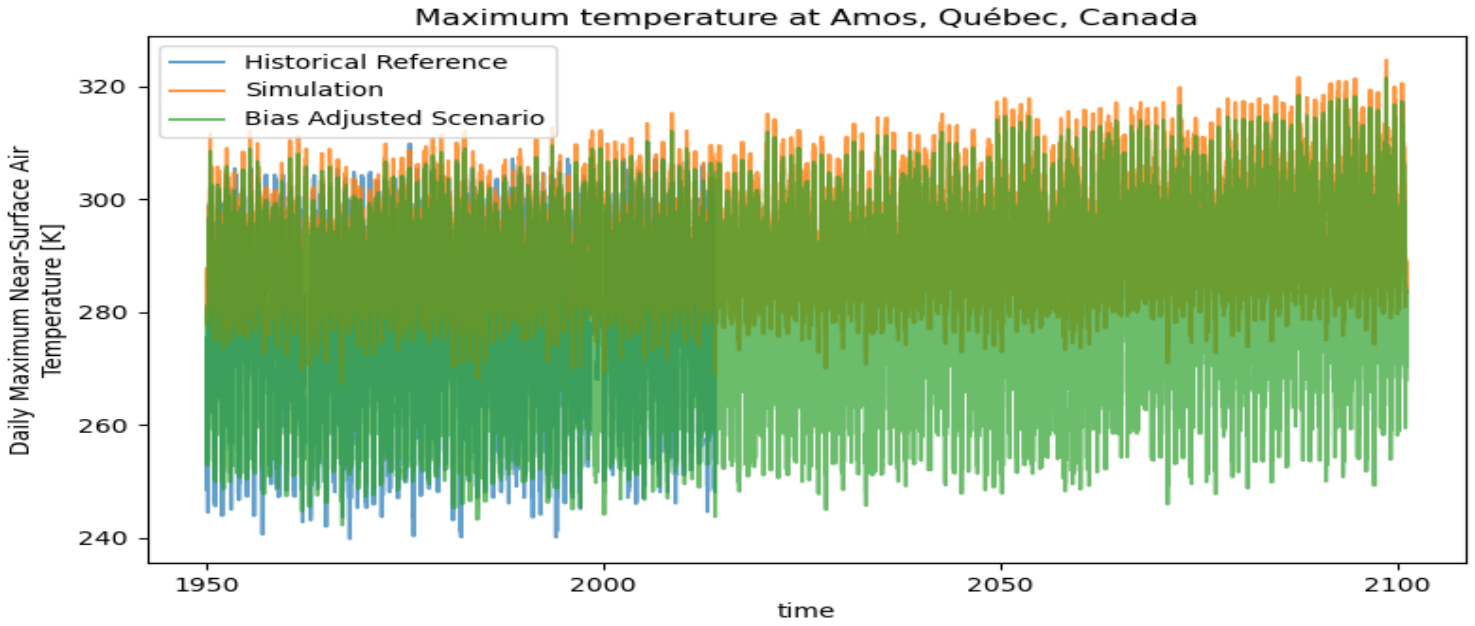


Average temperature from the years 1991-2020 average across 14 Regional Climate Models (*extreme warming scenario: SSP3-7.0*)

What Does Xclim do? Bias Adjustment

- Model `train` / `adjust` approach





Upstream contributions from Xclim

- Non-standard calendar (`cftime`) support in `xarray.groupby`
- Quantile methods in `xarray.groupby`
- Non-standard calendar conversion migrated from `xclim` to `xarray`
- Climate and Forecasting (CF) unit definitions inspired from `MetPy`
  - Inspiring work in `cf-xarray`
- Weighted variance, standard deviations, and quantiles in `xarray` (for ensemble statistics)
- Faster `NaN`-aware quantiles in `numpy`
- Initial polyfit function in `xarray`
- Also, we help maintain `xESMF`, `intake-esm`, `cf-xarray`, `xncml`, `climpred` and others for `xclim`-related tools

That's great and all, but what if...

- There's just too much data that we need to crunch :
  - The data could be spread across servers globally
  - Local computing power is not powerful enough for the analyses
- The user knows programming but not Python :
  - A biologist who uses `R` or a different program for their work
  - An engineer who just needs a range of estimates for future rainfall
- The user just wants to see some custom maps :
  - Agronomist who is curious about average growing conditions in 10 years?

Tools

xclim

XSCEN

MIRANDA

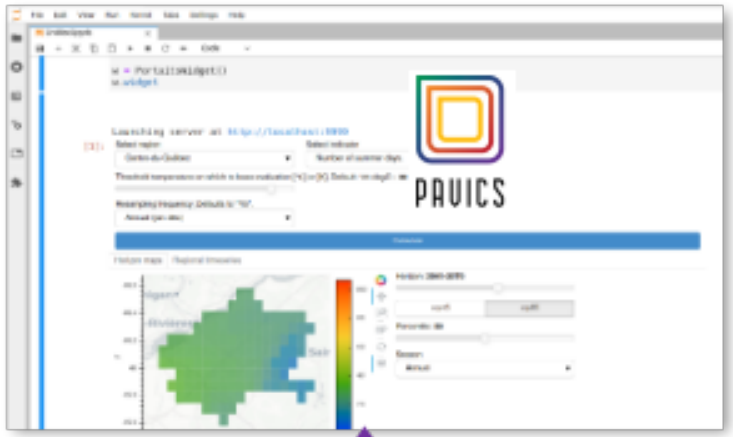
Raven

RavenPy

Platforms

PAVICS

Dashboards



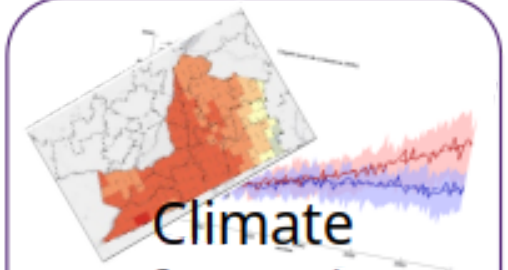
Climate Portraits

```
def frequency_analysis(
    da: xr.DataArray,
    mode: str,
    t: Union[int, Sequence[int]],
    dist: str,
    window: int = 1,
    freq: Optional[str] = None,
    **indexer,
):
```

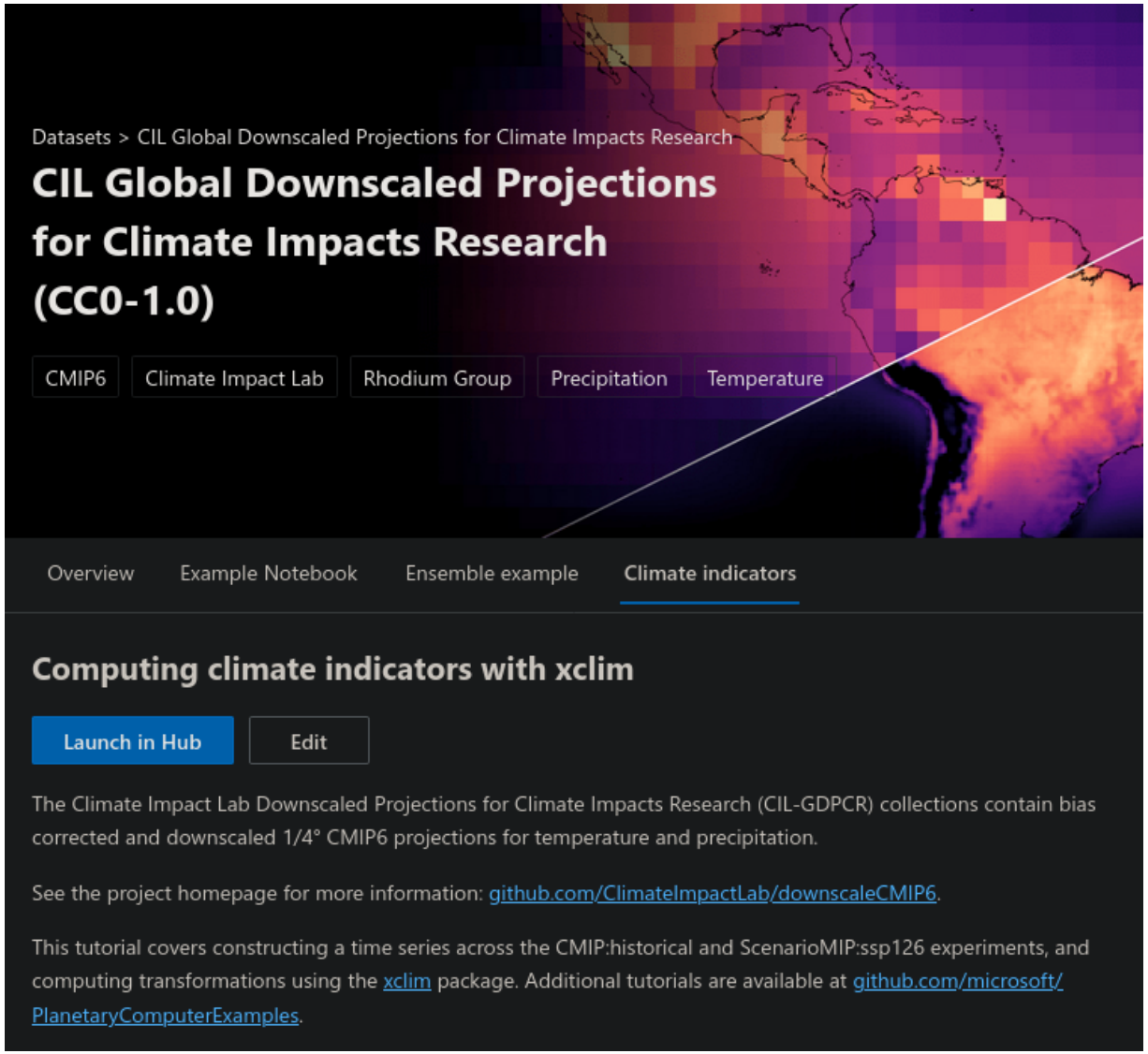
Software and workflows

2TB 2TB

Climate Data



Climate Information



Xclim on Computation Platforms

Microsoft Planetary Computer

- Computing Climate Indicators with xclim

Enhancing Accessibility : Web Services

- **WMS** : Web Mapping Service
  - **Google Maps**
- **WFS** : Web Feature Service
- **WCS** : Web Coverage Service
- **WPS** : ***Web Processing Service***
  - Running geospatial analyses over the internet

Finch : Climate Indicator Web Processing Service

github.com/Bird-house/Finch

Finch

Navigation

Indicators Processes

Processes for indicators

class finch.processes.virtual.indicators.**base_flow_index_Indicator_Process**

Bases: **XclimIndicatorBase**

base_flow_index Base flow index (v0.1)

Return the base flow index, defined as the minimum 7-day average flow divided by the mean flow.

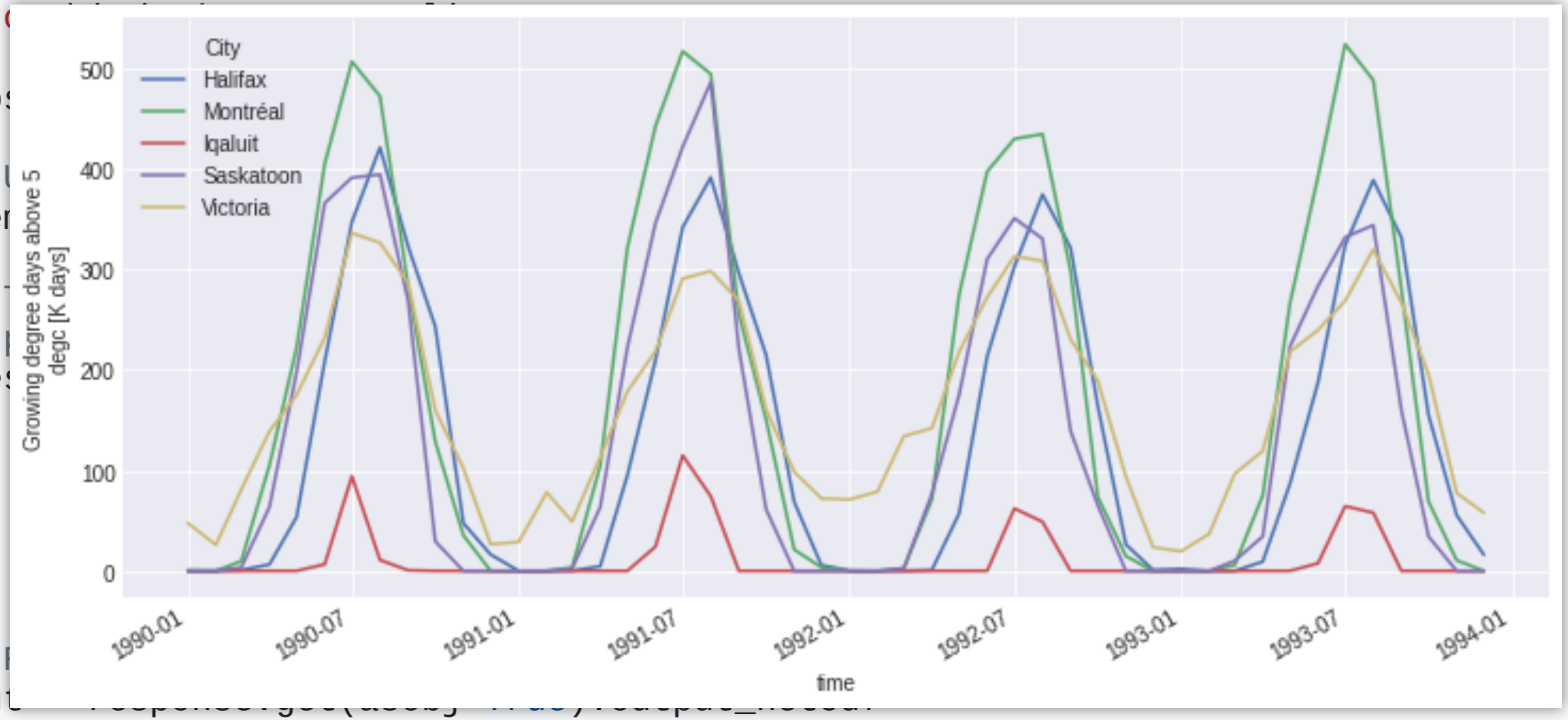
Dynamically-generated indicators from `xclim` (~430 Indicators in total)

Using remote Finch Web Service from Python (with `birdy`)

```python
from birdy import WPSClient

wps = WPSClient("https://ouranos.ca/example/finch/wps")

# Using the OPeNDAP protocol
remote_dataset = "www.exampledata.lt/climate.ncml"

# The indicator call looks a lot like the one from `xclim` but
# passing a url instead of an `xarray` object.
response = wps.growing_degree_days(
    remote_dataset,
    thresh='10 degC',
    freq='MS',
    variable='tas'
)

# Returned as a streaming `xarray` data object
out = response.get(asobj=True).output_netcdf

out.growing_degree_days.plot(hue='location')
```

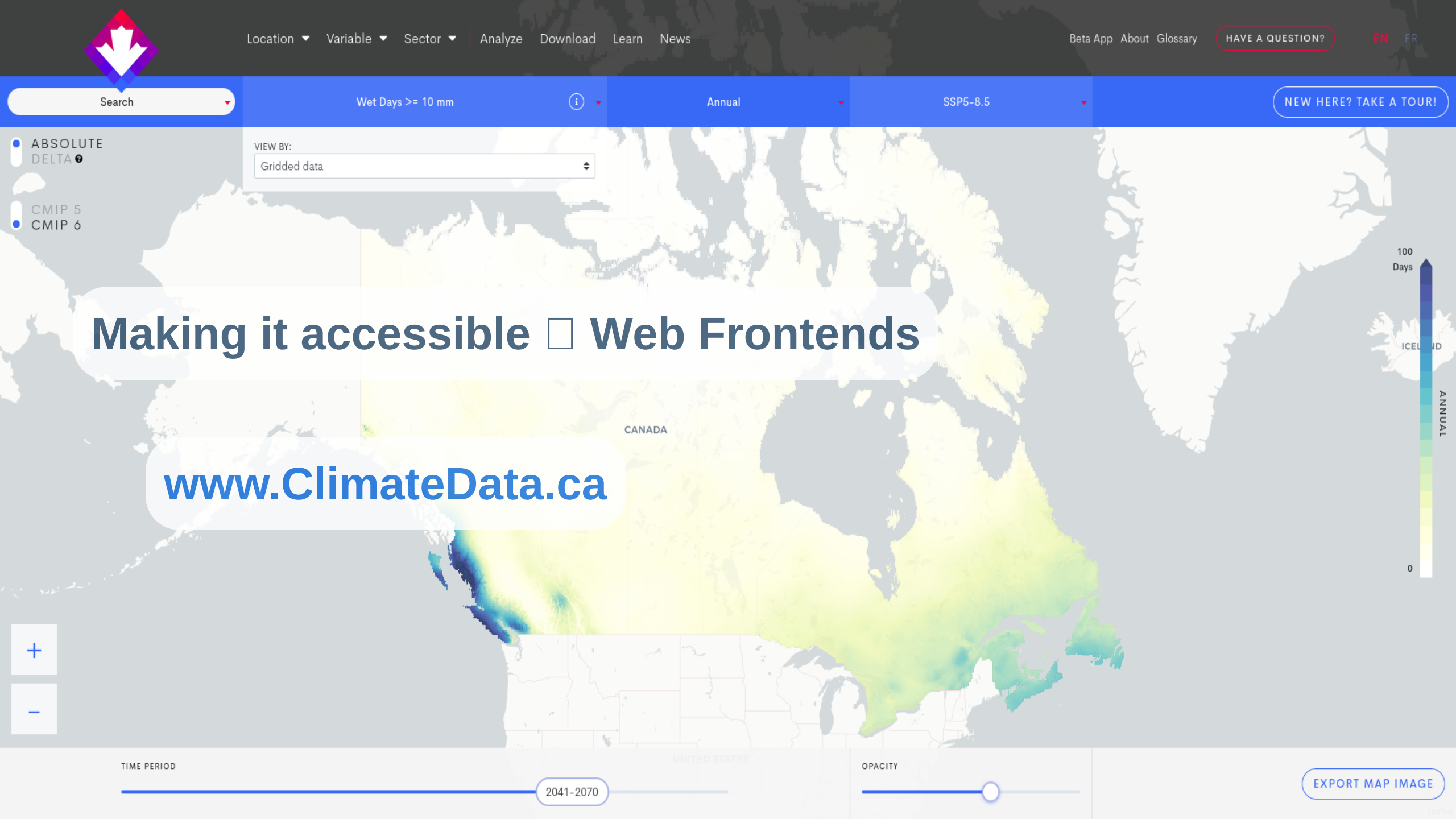
Bird-house/birdy -> PyWPS Helper Library

Using remote Finch Web Service from Python (`birdy`)




```
out.growing_degree_days.plot(hue='location')
```

Bird-house/birdy -> PyWPS Helper Library


Location
Variable
Sector
Analyze
Download
Learn
News
Beta App
About
Glossary
HAVE A QUESTION?
EN
FR
Search
Wet Days >= 10 mm
Annual
SSP5-8.5
NEW HERE? TAKE A TOUR!
ABSOLUTE
DELTA
VIEW BY:
Gridded data
CMIP 5
CMIP 6
100
Days
0
ANNUAL
CANADA
Making it accessible □ Web Frontends
www.ClimateData.ca
TIME PERIOD
2041-2070
OPACITY
EXPORT MAP IMAGE

1 CHOOSE A DATASET

2 SELECT LOCATIONS

3 CUSTOMIZE VARIABLES

4 CHOOSE A TIMEFRAME

5 ADVANCED

Models

Full ensemble

Emissions Scenarios

SSP1-2.6 SSP2-4.5 SSP5-8.5

Percentiles (Unselect all to receive output from individual models)

5 10 25 50 75 90 95

Temporal Frequency

Annual Monthly

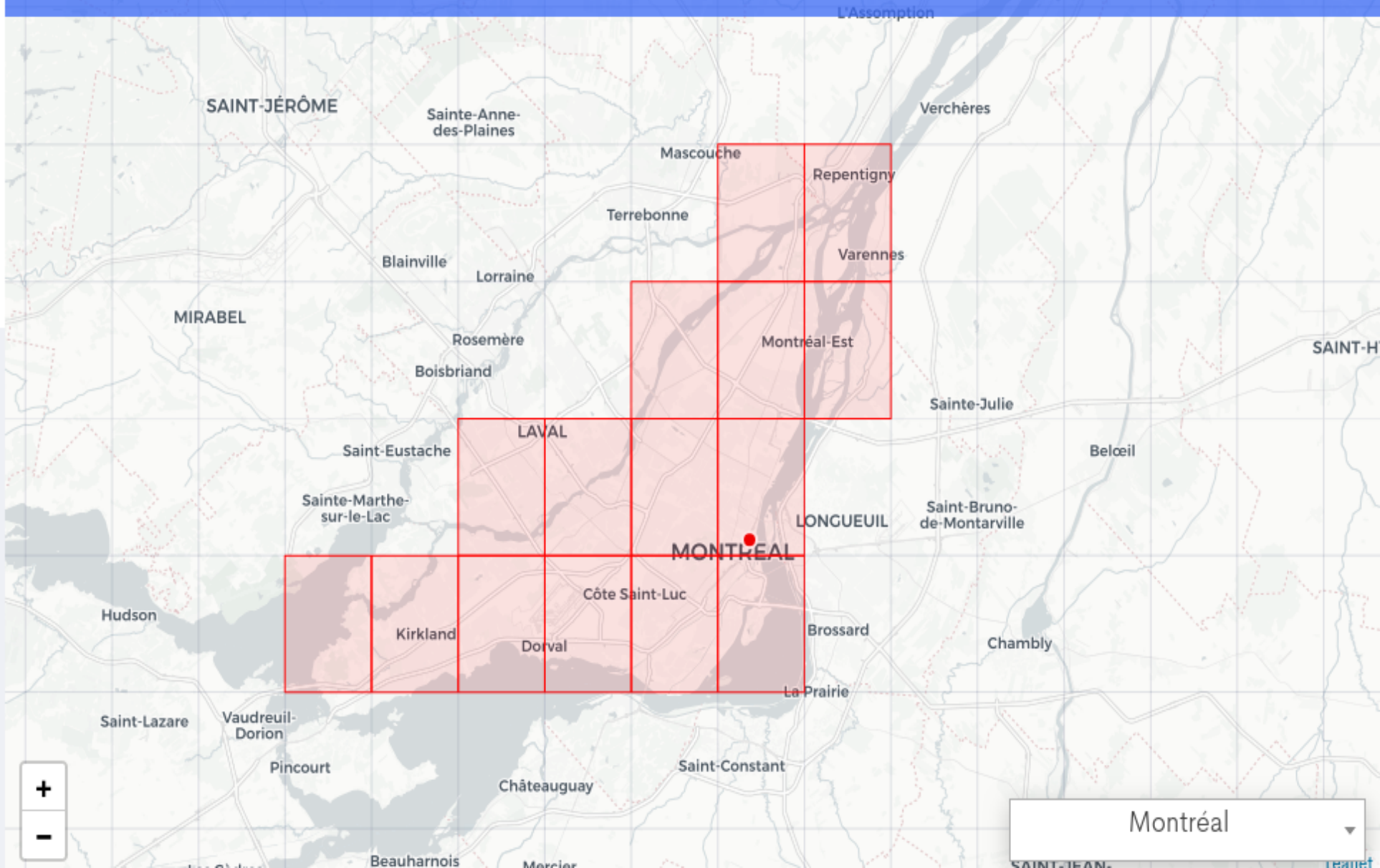
DATASET: CMIP6 (CanDCS-U6)

LOCATION: 15 Grids

VARIABLE(S): Degree Days Below a Threshold

TIMEFRAME: 2030 – 2060

OPTIONS: 1 Emissions Scenario | 3 Percentiles



Modern-day Climate Services with Python

- **Open Source Python** libraries (`numpy`, `sklearn`, `xarray`, etc.)
- **Multithreading** and **streaming** data formats (e.g. `OPeNDAP` and `ZARR`)
- **Common tools** built collaboratively and **shared widely** (`xclim`, `finch`)
- **Docker**-deployed **Web-Service**-based infrastructure
- **Testing**, **CI/CD** pipelines, and **validation workflows**
- **Peer-Reviewed** software (**pyOpenSci** and **JOSS**)

Thanks! Ačiū!

Colleagues and Collaborators

- Pascal Bourgault
- David Huard
- Travis Logan
- Abel Aoun
- Juliette Lavoie
- Éric Dupuis
- Gabriel Rondeau-Genesse
- Carsten Ehbrecht
- Long Vu
- Sarah Gammon
- David Caron

and many more contributors!

Have a great rest of PyCon Lithuania!

github.com/Ouranosinc/xclim

JOSS 10.21105/joss.05415

DOI 10.5281/zenodo.10710942

github.com/Bird-house/finch

DOI 10.5281/zenodo.10870939

This presentation:

https://zeitsperre.github.io/PyConLT2024/