Lab 8: Working with climate data in Julia

2026-11-13

In this lab, you will learn to download, load, subset, and plot climate model output in Julia with the `YAXArrays.jl` package. You will

1. work through the package's tutorials, then
2. use what you learned on a CMIP6 simulation of near-surface air temperature.

Background

Python's `xarray` package has changed how climate scientists work with labeled multi-dimensional arrays. It was developed at The Climate Corporation, released as open source in 2014, and is now the standard tool for climate data analysis in Python. Julia is less widely used for climate data, but the Climate Modeling Alliance is building an Earth system model from scratch in Julia, and that work is driving the development of climate data tools in Julia.

Julia has advantages for this course. Julia is fast, so you do not need to switch to C, Fortran, or C++ for code that has to run quickly. You can write a statistical method in plain Julia and apply it directly to climate data. This means you can write your own downscaling algorithms, bias corrections, and statistical models without learning a specialized library's syntax or calling one language from another. Code that stays in one language is often easier to understand and maintain.

`xarray` remains excellent and widely used in practice. For this course, which focuses on implementing and understanding statistical methods, Julia's speed and readability make it easier to write, test, and apply your own algorithms to real climate data.

Objectives

By the end of this lab, you will be able to:

1. Load and explore climate model output using `YAXArrays.jl`.
2. Work with NetCDF files and understand how CMIP6 data are organized.
3. Extract and plot climate model data for specific locations and time periods.

Before class

All the packages you need are in this lab's environment, and the setup cell below installs them the first time you run it. The first run can take 15 minutes or more.

The analysis after the tutorials uses daily near-surface air temperature (`tas`, in kelvin) from the CMIP6 archive hosted on Google Cloud, the same `gs://cmip6` bucket as the Plotting Maps tutorial. The data are CMIP6 simulations under SSP5-8.5, a very high emissions scenario, from two models, MPI-ESM1-2-LR (Max Planck Institute) and CanESM5 (Canadian Centre for Climate Modelling and Analysis).

Each dataset in the archive has its own path, called a store, such as `store = "gs://cmip6/CMIP6/ScenarioMIP/DKRZ/MPI-ESM1-2-HR/ssp585/r1i1p1f1/3hr/tas/gn/v20190710/"`, which the Plotting Maps tutorial reads. Each part of that path names one choice, following the CMIP6 Data Reference Syntax:

- `cmip6`: the top-level Google Cloud storage bucket (hence `gs`).
- `CMIP6`: the root directory for the project.
- `ScenarioMIP`: the Model Intercomparison Project (MIP).
- `DKRZ`: the institution, here the German Climate Computing Center.
- `MPI-ESM1-2-HR`: the model, here the Max Planck Institute Earth System Model, version 1.2, high-resolution version.
- `ssp585`: the experiment, here Shared Socioeconomic Pathway 5-8.5, a very high emissions scenario.
- `r1i1p1f1`: the variant label. `r1` is realization 1, which changes from one ensemble member (one run of the same model and scenario, started from different initial conditions) to the next. `i1` is initialization method 1, `p1` is physics version 1, and `f1` is forcing index 1.
- `3hr`: the time frequency, 3-hourly.
- `tas`: the variable, near-surface air temperature.
- `gn`: the grid label, here the model's native grid.
- `v20190710`: the version date.

The files you download below come from stores like this one, with the frequency `day`, so each value is one day's mean. For MPI-ESM1-2-LR, the low-resolution version of the Max Planck model, the institution is `MPI-M`. For CanESM5, the institution is `CCCma` and the version is `v20190429`.

The day before class, run the five cells in this section, in order. The download cell saves only your box, under 10 MB in total in the `data/` folder in this lab, but each cloud file covers the whole globe, so it takes about 30 minutes. Each file is saved in the Zarr format, which is a folder of compressed chunks rather than a single file. If you run it again, it skips every file already downloaded, so it finishes in seconds. The file names include the box, so if you change the box, run the download cell again, which takes about 30 minutes. The files for the old box stay in `data/` and are not used.

```
using Pkg
here = dirname(@__FILE__)
Pkg.activate(here)
Pkg.instantiate()
```

The next cell loads the packages used below.

```
using Dates, DimensionalData, IntervalSets, Statistics, YAXArrays, Zarr
```

The next cell names the variable, ensemble members, periods, and 5 x 5 degree box you will work with. Choose your box now: set `box_lon` and `box_lat` to any 5 x 5 degree box you like.

```
variable = "tas"
ensemble_members = ["r1i1p1f1", "r2i1p1f1", "r3i1p1f1"]
early = 2015:2034
late = 2081:2100
```

```
# Longitudes run from 0 to 360, so Houston is at about 265 degrees east.
box_lon = 265 .. 270
box_lat = 28 .. 33
```

The next cell defines two functions. `cmip6_path` builds a data file's path from a model, an ensemble member, a variable, and a range of years, and `load_cmip6` opens that file as a cube, the labeled array that the Getting Started guide below introduces.

```
data_dir = joinpath(here, "data")

# Return the path of the local file for one model, member, variable, and range of years,
in your box.
function cmip6_path(model, member, variable, years)
    box = "lon$(box_lon.left)-$(box_lon.right)_lat$(box_lat.left)-$(box_lat.right)"
    file = "$(model)_$(member)_$(variable)_$(first(years))-$(last(years))_$(box).zarr"
    return joinpath(data_dir, file)
end

# Open a downloaded file and return its data as a cube.
function load_cmip6(model, member, variable, years)
    return Cube(open_dataset(cmip6_path(model, member, variable, years)))
end
```

The next cell downloads the data for the last tasks. It saves the data in the `data/` folder of this lab, which is not committed to Git. It does not run when this document is rendered (`eval: false`), so you must run it yourself the day before class, with the **Run Cell** button above it or `Shift+Enter`. `cmip6_store` builds the path of a cloud store like the one above. `download_cmip6` reads one model, member, variable, and range of years from that store, keeps only your box, and writes it to a local file, skipping a file that already exists.

```
# Return the cloud store for daily SSP5-8.5 output of one model, member, and variable.
function cmip6_store(model, member, variable)
    institution = Dict("MPI-ESM1-2-LR" => "MPI-M", "CanESM5" => "CCCma")[model]
    version = Dict("MPI-ESM1-2-LR" => "v20190710", "CanESM5" => "v20190429")[model]
       return  "gs://cmip6/CMIP6/ScenarioMIP/$institution/$model/ssp585/$member/day/$
variable/gn/$version/"
end

# Download one model, member, variable, and range of years in your box, unless already
saved.
function download_cmip6(model, member, variable, years)
    path = cmip6_path(model, member, variable, years)
    if !isdir(path)
        store = zopen(cmip6_store(model, member, variable); consolidated=true)
        cube = open_dataset(store)[variable]
        subset = cube[time=Where(t -> year(t) in years), lon=box_lon, lat=box_lat]
```

```
        rm(path * ".partial"; force=true, recursive=true)
        savecube(subset, path * ".partial"; driver=:zarr)
        mv(path * ".partial", path)
    end
end

mkpath(data_dir)
for model in ["MPI-ESM1-2-LR", "CanESM5"], member in ensemble_members, years in (early,
late)
    download_cmip6(model, member, variable, years)
end
```

Tutorials

Work through the three tutorials below in the Julia REPL or in a separate scratch file, not in this notebook. They are not part of what you turn in. You can copy and paste the code blocks, but make sure you understand what each line does. The `using` line under Before class needs only the packages the analysis steps use, so load a tutorial's packages in the REPL.

Getting started

Work through the Getting Started with `YAXArrays.jl` user guide.

Selecting data

Much of the functionality of `YAXArrays.jl` comes from `DimensionalData.jl`, **including the functions for selecting subsets of data and for grouping and aggregating it**.

- Read the Dimensions, Selectors, and other sections of the `DimensionalData.jl` documentation under "Getting Started".
- Work through the `YAXArrays.jl` tutorial on selecting data. **To avoid downloading the file again every time you run the code, start the REPL in this lab's folder and replace `path = download(url, fname)` with `mkpath("data")`, then `path = joinpath("data", fname)`, then `isfile(path) || download(url, path)`, which saves the file in `data/`.**

You do not need to work through every `YAXArrays.jl` user guide. Do take a few minutes to browse the other guides, so you know what the package can do when you need it later.

Plotting maps

Work through the Plotting Maps tutorial.

- Use `CairoMakie.jl` instead of `GLMakie.jl`: replace `using GLMakie` and `using GLMakie.GeometryBasics` with `using CairoMakie` and `using CairoMakie.GeometryBasics`. The `Makie.jl` documentation explains the backends.
- You can skip the `AlgebraOfGraphics.jl` part, although the package is installed if you want to try it.

Daily temperature in a 5 x 5 degree box

This step and the three after it each have a cell for your code, above the step's **Your answer** line; add more cells there as you need them. Open the file for member `r1i1p1f1` of MPI-ESM1-2-LR, which holds your box, and compute the average `tas` over it for every day in each of the two periods, 2015-2034 and

2081-2100. If you want to go further, weight each grid cell by the cosine of its latitude, since grid cells shrink toward the poles, as in this `xarray` example. Plot the two daily time series of `tas` for your box.

To open a downloaded file, call `load_cmip6("MPI-ESM1-2-LR", "r1i1p1f1", "tas", early)`.

```
# Put your code for the daily temperature in your box here.
```

Your answer: Which 5 x 5 degree box did you choose, and how did you compute its daily average `tas`, including whether you weighted grid cells by latitude?

Hottest and coldest days

From those two time series, find the hottest and coldest days across both periods for your box, and report their dates.

```
# Put your code for the hottest and coldest days here.
```

Your answer: On which dates did the hottest and coldest days in your box fall, which period was each in, and how far was each from that period's mean?

Early and late 21st century

Plot a histogram of the daily average `tas` over your box for 2015 to 2034, pooling the three ensemble members `r1i1p1f1`, `r2i1p1f1`, and `r3i1p1f1` of MPI-ESM1-2-LR. Use normalized histograms, so that distributions built from different numbers of days compare fairly. Make the same histogram for 2081 to 2100 and draw it over the first one, to compare the early and the late 21st century.

To pool members, join their daily series end to end with `vcat`. For a normalized histogram, pass `normalization=:pdf` to `hist!`.

```
# Put your code for the early and late 21st century histograms here.
```

Your answer: How does the distribution of daily average `tas` in your box change between 2015-2034 and 2081-2100, both in its center and in its hot and cold tails?

A second model

Make the same early and late histograms for CanESM5 with the same box and its three ensemble members, and compare its histogram for one of the two periods to the MPI-ESM1-2-LR histogram for that period.

```
# Put your code for the CanESM5 histograms here.
```

Your answer: For the same period, which model is warmer in your box, by about how much, and which has the wider spread?

Turning it in

This lab is due on Canvas on Friday, November 20. It is worth 3 points, graded on your answers in the **Your answer** blocks rather than on your code.

1. Check that your notebook answers each **Your answer** question.

2. Render the notebook to a PDF. In the terminal, from the lab folder, run:

   ```
   quarto render index.qmd --to typst
   ```

   This writes `index.pdf` beside the notebook. It runs the whole notebook from the top in a fresh process, so it also confirms your work does not depend on leftover REPL state. The download cell does not run in the render, so it needs the files in `data/` from the day before class.

3. Commit and push your work, with GitHub Desktop or by asking Claude to commit and push. The downloaded data in `data/` is ignored by Git, so it is not part of your repository.

4. Upload `index.pdf` to the Lab 8 assignment on Canvas.

5. Paste the link to your repository in the same Canvas submission.

If the render fails and you cannot fix it, upload what you have and say where it broke. A lab that does not render is worth more to me than a lab you did not turn in.