Parallel Processing

Data Wrangling, Session 7d

Kieran Healy
Code Horizons

January 2026

Load the packages, as always

```
library(here)      # manage file paths
library(socviz)    # data and some useful functions
library(tidyverse) # your friend and mine

## Magic new package
# install.packages("furrr")
library(furrr) # Also loads `future`
```

```
Loading required package: future
```

Split, Apply, Combine

A lot of analysis has this pattern

We start with a dataset

We **split** it into pieces, usually according to some feature or categorical variable, or by file or something.

We do something—the *same* thing—to each of those pieces. That is we **apply** a function or procedure to the pieces. That procedure returns some result for each piece. The result will be of the same form: a number, a vector of counts, summary statistics, a model, a list, whatever.

Finally we **combine** those results into a final piece of output, usually a tibble or somesuch.

For example

`dplyr` is all about this.

```
gss_sm |>
  count(bigregion)
```

```
# A tibble: 4 × 2
  bigregion     n
  <fct>     <int>
1 Northeast   488
2 Midwest     695
3 South      1052
4 West        632
```

We *split* into groups, *apply* the `sum()` function within the groups, and *combine* the results into a new tibble showing the resulting sum per group. The various dplyr functions are oriented to doing this in a way that gives you a consistent set of outputs.

For example: split

We can split, apply, combine in various ways.

Base R has the `split()` function:

```r
out <- mtcars |>
  split(mtcars$cyl)
summary(out) # mtcars split into a list of data frames by the `cyl` variable
```

```
  Length Class      Mode
4 11     data.frame list
6 11     data.frame list
8 11     data.frame list
```

For example: split

Tidyverse has `group_split()`:

```r
out <- mtcars |>
  group_split(cyl)
out
```

```
<list_of<
  tbl_df<
    mpg : double
    cyl : double
    disp: double
    hp  : double
    drat: double
    wt  : double
    qsec: double
    vs  : double
    am  : double
    gear: double
    carb: double
  >
>[3]>
[[1]]
# A tibble: 11 × 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
```

For example: apply

The application step is "I want to fit a linear model to each piece"

```
out <- mtcars |>
  group_split(cyl) |>
  map(\(df) lm(mpg ~ wt + hp + gear, data = df))
```

For example: apply

The application step is “I want to fit a linear model to each piece” and get a summary

```r
mtcars |>
  group_split(cyl) |>
  map(\(df) lm(mpg ~ wt + hp + gear, data = df)) |>
  map(summary) |>
  map_dbl("r.squared")
```

```
[1] 0.7301860 0.6597413 0.4995237
```

For example: combine

In this case the “combine” step is implicitly at the end: we get a vector of R squareds back, and it’s as long as the number of groups.

```r
mtcars |>
  group_split(cyl) |>
  map(\(df) lm(mpg ~ wt + hp + gear, data = df))  |>
  map(summary) |>
  map_dbl("r.squared")
```

```
[1] 0.7301860 0.6597413 0.4995237
```

For example: apply

This is also what we're doing more elegantly (staying within a tibble structure) if we `nest()` and use `broom` to get a summary out.

```r
mtcars |>
  group_by(cyl) |>
  nest() |>
  mutate(model = map(data, \(df) lm(mpg ~ wt + hp + gear, data = df)),
         perf = map(model, broom::glance)) |>
  unnest(perf)
```

```
# A tibble: 3 × 15
# Groups:   cyl [3]
    cyl data     model  r.squared adj.r.squared sigma statistic p.value    df
  <dbl> <list>   <list>     <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>
1     6 <tibble> <lm>       0.660         0.319  1.20      1.94  0.300      3
2     4 <tibble> <lm>       0.730         0.615  2.80      6.31  0.0211     3
3     8 <tibble> <lm>       0.500         0.349  2.06      3.33  0.0648     3
# ℹ 6 more variables: logLik <dbl>, AIC <dbl>, BIC <dbl>, deviance <dbl>,
#   df.residual <int>, nobs <int>
```

How this happens

In each of these cases, the data is processed *sequentially* or *serially*. R splits the data according to your instructions, applies the function or procedure to each one in turn, and combines the outputs in order out the other side. Your computer's processor is handed each piece in turn.

How this happens

For small tasks that's fine. But for bigger tasks it gets inefficient quickly.

```
## From Henrik Bengtsson's documentation for future/furrr
slow_sum <- function(x) {
  sum <- 0
  for (value in x) {
    Sys.sleep(1.0)  ## one-second slowdown per value
    sum <- sum + value
  }
  sum
}

# This takes > ten seconds to run.
tictoc::tic()
slow_sum(1:10)
```

```
[1] 55
```

```
tictoc::toc()
```

```
10.052 sec elapsed
```

If *this* is the sort of task we have to apply to a bunch of things, it's going to take ages.

That’s Embarrassing

A feature of many split-apply-combine activities is that it *does not matter* what order the “apply” part happens to the groups. All that matters is that we can combine the results at the end, no matter what order they come back in. E.g. in the `mtcars` example the model being fit to the 4-cylinder cars group doesn’t depend in any way on the results of the model being fit to the 8-cylinder group.

This sort of situation is *embarrassingly parallel*.

That’s Embarrassing

When a task is embarrassingly parallel, and the number of pieces or groups is large enough or complex enough, then if we can do them at the same time then we should. There is some overhead—we have to keep track of where each piece was sent and when the results come back in—but if that’s low enough in comparison to doing things serially, then we should parallelize the task.

Multicore Computing

Parallel computing used to mean “I have a cluster of computers at my disposal”. But modern CPUs are constructed in a semi-modular fashion. They have some number of “cores”, each one of which is like a small processor in its own right. In effect you have a computer cluster already.

Some Terms

Socket: The physical connection on your computer that houses the processor. These days, mostly there's just one.

Core: The part of the processor that actually performs the computation. Most modern processors have multiple cores. Each one can do wholly independent work.

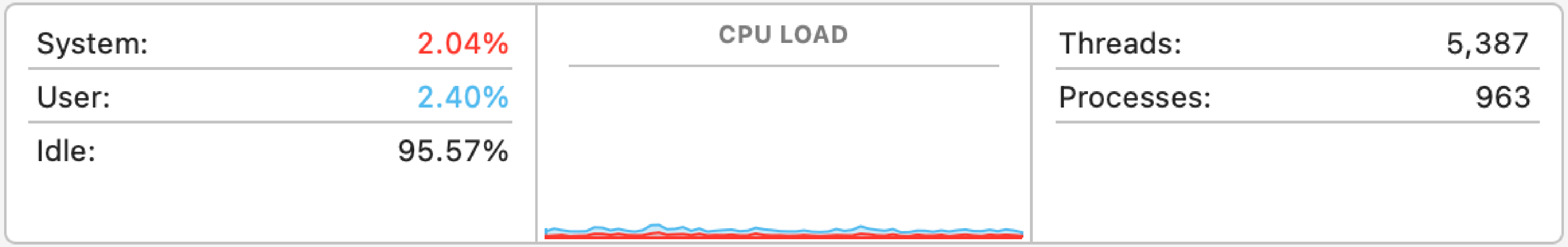
Process: A single instance of a running task or program (R, Slack, Chrome, etc). A core can only run one process at a time. But, cores are *fast*. And so, they can run many *threads*

Thread: A piece of a process that can share memory and resources with other threads. If you have enough power you can do something Intel called **hyperthreading**, which is a way of dividing up a physical core into (usually) two *logical* cores that work on the same clock cycle and share some resources on the physical core.

Cluster: A collection of things that are capable of hosting cores. Might be a single socket (on your laptop) or an old-fashioned room full of many physical computers that can be made to act as if they were a single machine.

Multicore Computing

Most of the time, even when we are using them, our computers sit around doing PRETTY MUCH NOTHING.




```r
## How many cores do we have?
parallelly::availableCores()
```

```
system 
    16 
```

Remember, processor clock cycles are *really fast*. They're measured in billions of cycles per second.

We need to put those cores to work!

Previously: Future and furrr

`future` and `furrr` made parallel computing *way* more straightforward than it used to be. In particular, for Tidyverse-centric workflows, `furrr` provides a set of `future_` functions that are drop-in replacements for `map` and friends. So `map()` becomes `future_map()` and so on.

However, `purrr` has recently added support for parallelization via the `in_parallel()` function. It uses `crate` and `mirai` under the hood.

Parallel purrr

Update your R packages with `remotes::update_packages()`.

```
library(tidyverse)
library(here)

# Set up parallel processing
library(mirai)

status()
```

```
$connections
[1] 0

$daemons
[1] 0
```

```
# Summon daemons. Do not summon more daemons than you have cores. People often choose one less than the number of
daemons(15)

status()
```

```
$connections
[1] 15

$daemons
[1] "ipc:///tmp/2223b6a0a0c3faee6151d441"

$mirai
 awaiting executing completed
        0         0         0
```

Toy Example

```r
# Another slow function (from
# Grant McDermott this time)
slow_square <- function(x = 1) {
    x_sq <- x^2
    ## We explicitly namespace all our package function calls
    out <-  tibble::tibble(value = x, value_sq = x_sq)
    Sys.sleep(2) # Zzzz
    out
}

tictoc::tic("Serially")
## This is of course way slower than just writing
## slow_square(1:20) but nvm that for now
serial_out <- map(1:20, slow_square) |>
  list_rbind()
tictoc::toc()
```

```
Serially: 40.13 sec elapsed
```

Toy Example

```r
tictoc::tic("Parallelized")
parallel_out <- 1:20 |>
  map(in_parallel(\(x) slow_square(x),
      slow_square = slow_square)) |>
        list_rbind()
tictoc::toc()
```

```
Parallelized: 4.076 sec elapsed
```

```r
identical(serial_out, parallel_out)
```

```
[1] TRUE
```

Using `in_parallel()`

If you use `in_parallel()` but don't set `daemons()`, then the map will just proceed sequentially, as if `in_parallel()` were not there.

You have to explicitly pass through the names of any local functions you're using, as well as any local objects. This can make the code a little more verbose, but with more complex jobs it's clear what's being passed in and from where.

If you have written a local function and are passing that along, you must make sure any package functions it calls are fully namespaced (e.g. `tibble::tibble()` rather than just `tibble()`).

Using `in_parallel()`

E.g. From the help:

```r
# X This won't work - external dependencies not declared
my_data <- c(1, 2, 3)
map(1:3, in_parallel(\(x) mean(my_data)))

# ✅ This works - dependencies explicitly provided
my_data <- c(1, 2, 3)
map(1:3, in_parallel(\(x) mean(my_data), my_data = my_data))

# ✅ Package functions need explicit namespacing
map(1:3, in_parallel(\(x) vctrs::vec_init(integer(), x)))

# ✅ Or load packages within the function
map(
  1:3,
  in_parallel(\(x) {
    library(vctrs)
    vec_init(integer(), x)
  })
```

NOAA Temperature Data

See: https://github.com/kjhealy/noaa_ncei/. This function gets one folder full of NCDF files from NOAA.

```
#' Get a year-month folder of NCDF Files from NOAA
#'
#' @param url The endpoint URL of the AVHRR data, <https://www.ncei.noaa.gov/data/sea-surface-temperature-optimum
#' @param local A local file path for the raw data folders, i.e. where all the year-month dirs go. Defaults to a
#' @param subdir The subdirectory of monthly data to get. A character string of digits, of the form "YYYYMM". No
#'
#' @return  A directory of NCDF files.
#' @export
#'
#'
get_nc_files <- function(
  url = "https://www.ncei.noaa.gov/data/sea-surface-temperature-optimum-interpolation/v2.1/access/avhrr/",
  local = here::here(
    "raw/www.ncei.noaa.gov/data/sea-surface-temperature-optimum-interpolation/v2.1/access/avhrr/"
  ),
  subdir
) {
  localdir <- here::here(local, subdir)

  if (!fs::dir_exists(localdir)) {
    fs::dir_create(localdir)
  }

  files <- rvest::read_html(paste0(url, subdir)) |>
```

NOAA Temperature Data

Initial get:

```r
## Functions, incl. actual get_nc_files() function to get 1 year-month's batch of files.
source(here("R", "functions.R"))

### Initial get. Only have to do this once.
## We try to be nice.

# Data collection starts in September 1981
first_yr <- paste0("1981", sprintf('%0.2d', 9:12))
yrs <- 1982:2024
months <- sprintf('%0.2d', 1:12)
subdirs <- c(first_yr, paste0(rep(yrs, each = 12), months))

slowly_get_nc_files <- slowly(get_nc_files)

walk(subdirs, \(x) slowly_get_nc_files(subdir = x))
```

This tries to be polite with the NOAA: it enforces a wait time and in addition randomizes it to make it variably longer. There are a lot of files (>15,000). Doing it this way will take several *days* in real time (though much much less in actual transfer time of course).

NOAA Temperature Data

Raw data directories locally:

```r
basename(fs::dir_ls(here::here("avhrr")))
```

```
 [1] "198109" "198110" "198111" "198112" "198201" "198202" "198203" "198204"
 [9] "198205" "198206" "198207" "198208" "198209" "198210" "198211" "198212"
[17] "198301" "198302" "198303" "198304" "198305" "198306" "198307" "198308"
[25] "198309" "198310" "198311" "198312" "198401" "198402" "198403" "198404"
[33] "198405" "198406" "198407" "198408" "198409" "198410" "198411" "198412"
[41] "198501" "198502" "198503" "198504" "198505" "198506" "198507" "198508"
[49] "198509" "198510" "198511" "198512" "198601" "198602" "198603" "198604"
[57] "198605" "198606" "198607" "198608" "198609" "198610" "198611" "198612"
[65] "198701" "198702" "198703" "198704" "198705" "198706" "198707" "198708"
[73] "198709" "198710" "198711" "198712" "198801" "198802" "198803" "198804"
[81] "198805" "198806" "198807" "198808" "198809" "198810" "198811" "198812"
[89] "198901" "198902" "198903" "198904" "198905" "198906" "198907" "198908"
[97] "198909" "198910" "198911" "198912"
[ reached 'max' / getOption("max.print") -- omitted 428 entries ]
```

NOAA Temperature Data

Raw data files, in netCDF (Version 4) format

```r
basename(fs::dir_ls(here::here("avhrr", "202402")))
```

```
 [1] "oisst-avhrr-v02r01.20240201.nc" "oisst-avhrr-v02r01.20240202.nc"
 [3] "oisst-avhrr-v02r01.20240203.nc" "oisst-avhrr-v02r01.20240204.nc"
 [5] "oisst-avhrr-v02r01.20240205.nc" "oisst-avhrr-v02r01.20240206.nc"
 [7] "oisst-avhrr-v02r01.20240207.nc" "oisst-avhrr-v02r01.20240208.nc"
 [9] "oisst-avhrr-v02r01.20240209.nc" "oisst-avhrr-v02r01.20240210.nc"
[11] "oisst-avhrr-v02r01.20240211.nc" "oisst-avhrr-v02r01.20240212.nc"
[13] "oisst-avhrr-v02r01.20240213.nc" "oisst-avhrr-v02r01.20240214.nc"
[15] "oisst-avhrr-v02r01.20240215.nc" "oisst-avhrr-v02r01.20240216.nc"
[17] "oisst-avhrr-v02r01.20240217.nc" "oisst-avhrr-v02r01.20240218.nc"
[19] "oisst-avhrr-v02r01.20240219.nc" "oisst-avhrr-v02r01.20240220.nc"
[21] "oisst-avhrr-v02r01.20240221.nc" "oisst-avhrr-v02r01.20240222.nc"
[23] "oisst-avhrr-v02r01.20240223.nc" "oisst-avhrr-v02r01.20240224.nc"
[25] "oisst-avhrr-v02r01.20240225.nc" "oisst-avhrr-v02r01.20240226.nc"
[27] "oisst-avhrr-v02r01.20240227.nc" "oisst-avhrr-v02r01.20240228.nc"
[29] "oisst-avhrr-v02r01.20240229.nc"
```

Some Prep

```r
## Seasons for plotting
season <-  function(in_date){
  br = yday(as.Date(c("2019-03-01",
                      "2019-06-01",
                      "2019-09-01",
                      "2019-12-01")))
  x = yday(in_date)
  x = cut(x, breaks = c(0, br, 366))
  levels(x) = c("Winter", "Spring", "Summer", "Autumn", "Winter")
  x
}

season_lab <-  tibble(yrday = yday(as.Date(c("2019-03-01",
                                             "2019-06-01",
                                             "2019-09-01",
                                             "2019-12-01"))),
                      lab = c("Spring", "Summer", "Autumn", "Winter"))
```

NOAA Temperature Data

Raw data files

```r
#install.packages("ncdf4")
#install.packages("terra")
library(terra)

## For the filename processing
## This one gives you an unknown number of chunks each with approx n elements
chunk <- function(x, n) split(x, ceiling(seq_along(x)/n))

## This one gives you n chunks each with an approx equal but unknown number of elements
chunk2 <- function(x, n) split(x, cut(seq_along(x), n, labels = FALSE))

## All the daily .nc files:
all_fnames <- as.character(fs::dir_ls(here("avhrr"), recurse = TRUE, glob = "*.nc"))
chunked_fnames <- chunk(all_fnames, 25)

length(all_fnames)
```

```
[1] 16065
```

```r
length(chunked_fnames)
```

```
[1] 643
```

NOAA Temperature Data

The data is in netCDF (Version 4) format. An interesting self-documenting format. Read one in with the netCDF reader.

```r
tmp ← ncdf4::nc_open(all_fnames[10000])
tmp
```

```
File /Users/kjhealy/Documents/courses/data_wrangling/avhrr/200901/oisst-avhrr-v02r01.20090120.nc
(NC_FORMAT_NETCDF4):

     4 variables (excluding dimension variables):
        short anom[lon,lat,zlev,time]   (Chunking: [1440,720,1,1])  (Compression: shuffle,level 4)
            long_name: Daily sea surface temperature anomalies
            _FillValue: -999
            add_offset: 0
            scale_factor: 0.00999999977648258
            valid_min: -1200
            valid_max: 1200
            units: Celsius
        short err[lon,lat,zlev,time]   (Chunking: [1440,720,1,1])  (Compression: shuffle,level 4)
            long_name: Estimated error standard deviation of analysed_sst
            units: Celsius
            _FillValue: -999
            add_offset: 0
            scale_factor: 0.00999999977648258
            valid_min: 0
```

NOAA Temperature Data

We use the `terra` package, which understands many GIS formats. Each day is a grid or *raster* of data that's 1440 x 720 in size. It has several *layers*, each one a specific measure—sea-surface temperature, sea ice, etc.

```r
tmp <- terra::rast(all_fnames[10000])
tmp
```

```
class       : SpatRaster
size        : 720, 1440, 4  (nrow, ncol, nlyr)
resolution  : 0.25, 0.25  (x, y)
extent      : 0, 360, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84 (CRS84) (OGC:CRS84)
sources     : oisst-avhrr-v02r01.20090120.nc:anom
              oisst-avhrr-v02r01.20090120.nc:err
              oisst-avhrr-v02r01.20090120.nc:ice
              oisst-avhrr-v02r01.20090120.nc:sst
varnames    : anom (Daily sea surface temperature anomalies)
              err (Estimated error standard deviation of analysed_sst)
              ice (Sea ice concentration)
               ...
names       : anom_zlev=0, err_zlev=0, ice_zlev=0, sst_zlev=0
unit        :     Celsius,    Celsius,          %,    Celsius
depth       : 0
time (days) : 2009-01-20
```

Terra can understand rasters aggregated into slabs or “bricks” covering the same area repeatedly, and has methods for aggregating these arrays in

NOAA Temperature Data

Read one in with `terra`. Get the `sst` (Sea Surface Temperature) layer only.

```r
tmp <- terra::rast(all_fnames[10000])["sst"]
tmp
```

```
class       : SpatRaster
size        : 720, 1440, 1  (nrow, ncol, nlyr)
resolution  : 0.25, 0.25  (x, y)
extent      : 0, 360, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84 (CRS84) (OGC:CRS84)
source      : oisst-avhrr-v02r01.20090120.nc:sst
varname     : sst (Daily sea surface temperature)
name        : sst_zlev=0
unit        :    Celsius
depth       : 0
time (days) : 2009-01-20
```

NOAA Temperature Data

What reading a *chunk* of filenames (with all their layers) does:

```r
tmp2 <- terra::rast(chunked_fnames[[10]])
tmp2
```

```
class       : SpatRaster
size        : 720, 1440, 100  (nrow, ncol, nlyr)
resolution  : 0.25, 0.25  (x, y)
extent      : 0, 360, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84 (CRS84) (OGC:CRS84)
sources     : oisst-avhrr-v02r01.19820414.nc:anom
              oisst-avhrr-v02r01.19820414.nc:err
              oisst-avhrr-v02r01.19820414.nc:ice
              ... and 97 more sources
varnames    : anom (Daily sea surface temperature anomalies)
              err (Estimated error standard deviation of analysed_sst)
              ice (Sea ice concentration)
              ...
names       : anom_zlev=0, err_zlev=0, ice_zlev=0, sst_zlev=0, anom_zlev=0, err_zlev=0, ...
unit        :     Celsius,    Celsius,          %,    Celsius,     Celsius,    Celsius, ...
depth       : 0
time (days) : 1982-04-14 to 1982-05-08 (25 steps)
```

NOAA Temperature Data

Write a function to get a file, read in the SST raster, and get its area-weighted mean, for the North Atlantic region only.

```r
#' Process NCDF temperature rasters
#'
#' Use terra to process a temperature NCDF file
#'
#' @param fnames
#' @param crop_area
#' @param layerinfo
#'
#' @returns
#' @export
#'
#' @examples
process_raster <- function(
  fnames,
  crop_area = c(-80, 0, 0, 60),
  layerinfo = NULL
) {
  nc_layerinfo <- tibble::tibble(
    num = c(1:4),
    raw_name = c("anom_zlev=0", "err_zlev=0", "ice_zlev=0", "sst_zlev=0"),
    name = c("anom", "err", "ice", "sst")
  )

  if (!is.null(layerinfo)) {
```

NOAA Temperature Data

Try it on one data file:

```r
process_raster(all_fnames[1000]) |>
  as_tibble()
```

```
# A tibble: 1 × 5
  date          anom   err   ice   sst
  <date>       <dbl> <dbl> <dbl> <dbl>
1 1984-05-27 -0.372 0.182 0.437  20.6
```

Try it on one *chunk* of data files:

```r
process_raster(chunked_fnames[[500]]) |>
  as_tibble()
```

```
# A tibble: 25 × 5
   date        anom   err   ice   sst
   <date>     <dbl> <dbl> <dbl> <dbl>
 1 2015-11-02 0.566 0.160 0.267  22.8
 2 2015-11-03 0.561 0.174 0.264  22.8
 3 2015-11-04 0.559 0.167 0.271  22.8
 4 2015-11-05 0.571 0.157 0.271  22.7
 5 2015-11-06 0.588 0.154 0.273  22.7
 6 2015-11-07 0.581 0.159 0.274  22.7
 7 2015-11-08 0.559 0.162 0.275  22.6
 8 2015-11-09 0.529 0.157 0.300  22.5
 9 2015-11-10 0.492 0.156 0.311  22.5
10 2015-11-11 0.476 0.156 0.311  22.4
# ℹ 15 more rows
```

NOAA Temperature Data

Do it in parallel for all files:

```r
tictoc::tic("Terra Method")
df <- chunked_fnames |>
  map(in_parallel(
    \(x) process_raster(x),
    process_raster = process_raster
  )) |>
  list_rbind() |>
  as_tibble() |>
  mutate(
    date = ymd(date),
    year = lubridate::year(date),
    month = lubridate::month(date),
    day = lubridate::day(date),
    yrday = lubridate::yday(date),
    season = season(date)
  )

dim(df)
```

```
[1] 16065    10
```

```r
tictoc::toc()
```

```
Terra Method: 163.588 sec elapsed
```

NOAA Temperature Data

```r
df |>
  slice_sample(n = 10)
```

```
# A tibble: 10 × 10
   date         anom   err   ice   sst  year month   day yrday season
   <date>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <fct> 
 1 1991-10-19 -0.262 0.181 0.169  22.5  1991    10    19   292 Autumn
 2 1993-03-03 -0.307 0.189 0.621  18.9  1993     3     3    62 Spring
 3 1985-10-18 -0.344 0.186 0.102  22.5  1985    10    18   291 Autumn
 4 1983-07-27 -0.117 0.219 0.206  23.2  1983     7    27   208 Summer
 5 2020-10-10  0.734 0.162 0.233  23.8  2020    10    10   284 Autumn
 6 2016-11-13  0.586 0.159 0.183  22.4  2016    11    13   318 Autumn
 7 2006-04-10  0.230 0.171 0.579  19.7  2006     4    10   100 Spring
 8 1986-07-12 -0.296 0.187 0.163  22.6  1986     7    12   193 Summer
 9 2024-01-07  0.990 0.130 0.453  20.9  2024     1     7     7 Winter
10 2005-05-28  0.461 0.155 0.495  21.5  2005     5    28   148 Spring
```

NOAA Temperature Data

Make our cores work even harder

```r
# All the seas with rasterize() and zonal()
# Seas of the world polygons from https://www.marineregions.org/downloads.php,
# IHO Sea Areas V3 shapefile.
seas <- sf::read_sf(here("seas"))

seas
```

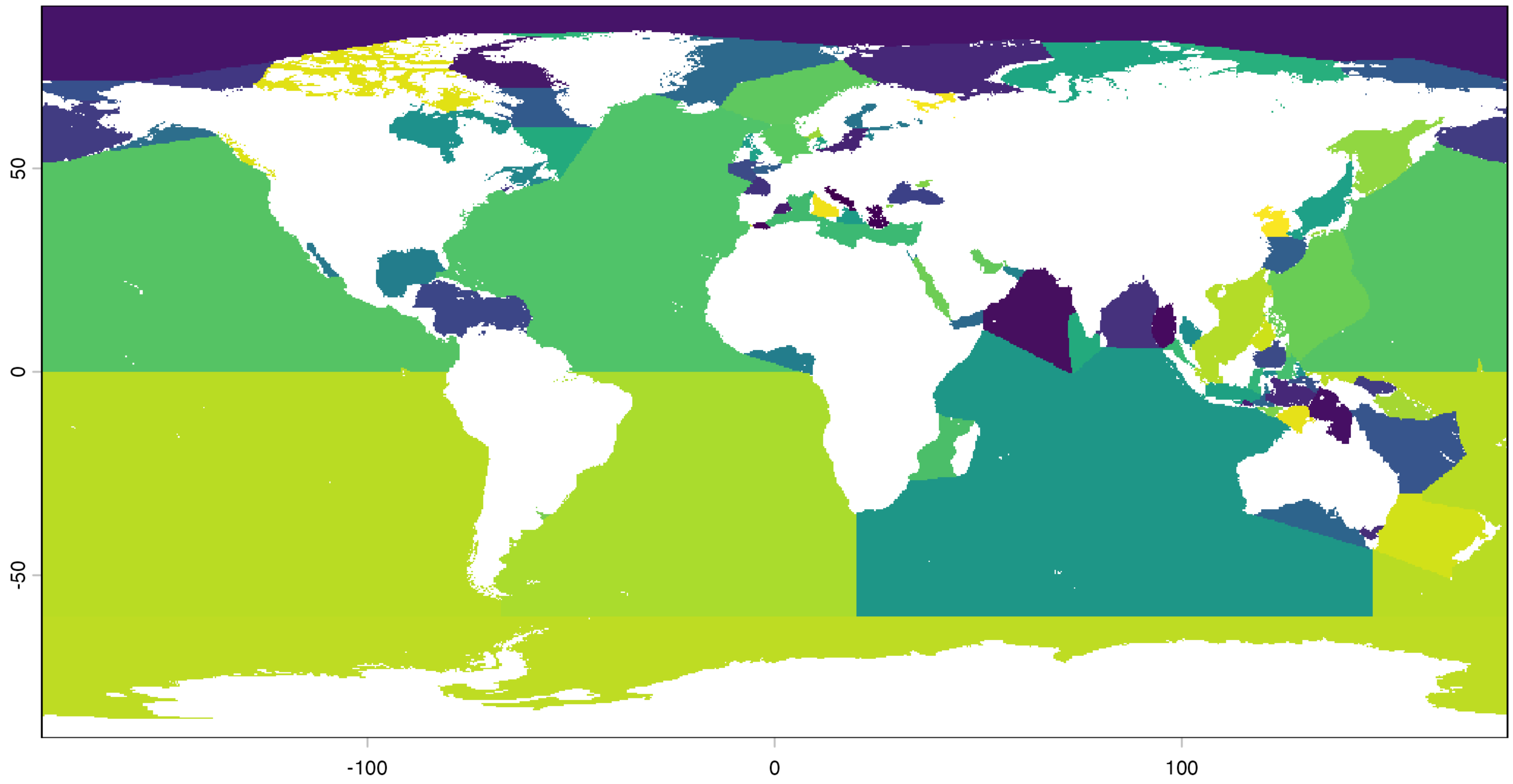
```
Simple feature collection with 101 features and 10 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -180 ymin: -85.5625 xmax: 180 ymax: 90
Geodetic CRS:  WGS 84
# A tibble: 101 × 11
   NAME           ID     Longitude Latitude min_X  min_Y max_X  max_Y   area MRGID
   <chr>          <chr>      <dbl>    <dbl> <dbl>  <dbl> <dbl>  <dbl>  <dbl> <dbl>
 1 Rio de La Pl… 33         -56.8    -35.1 -59.8 -36.4  -54.9 -31.5  3.18e4  4325
 2 Bass Strait   62A        146.     -39.5 144.  -41.4  150.  -37.5  1.13e5  4366
 3 Great Austra… 62         133.     -36.7 118.  -43.6  146.  -31.5  1.33e6  4276
 4 Tasman Sea    63         161.     -39.7 147.  -50.9  175.  -30    3.34e6  4365
 5 Mozambique C… 45A         40.9    -19.3  32.4 -26.8   49.2 -10.5  1.39e6  4261
 6 Savu Sea      48o        122.      -9.48 119.  -10.9  125.   -8.21 1.06e5  4343
 7 Timor Sea     48i        128.     -11.2  123.  -15.8  133.   -8.18 4.34e5  4344
 8 Bali Sea      48l        116.      -7.93 114.   -9.00 117.   -7.01 3.99e4  4340
 9 Coral Sea     64         157.     -18.2  141.  -30.0  170.   -6.79 4.13e6  4364
10 Flores Sea    48j        120.      -7.51 117.   -8.74 123.   -5.51 1.03e5  4341
# ℹ 91 more rows
```

NOAA Temperature Data

```r
## Rasterize the seas polygons using one of the nc files
## as a reference grid for the rasterization process
one_raster <- all_fnames[1]
seas_vect <- terra::vect(seas)
tmp_tdf_seas <- terra::rast(one_raster)["sst"] |>
  rotate()
seas_zonal <- rasterize(seas_vect, tmp_tdf_seas, "NAME")
```

NOAA Temperature Data

```
plot(seas_zonal, legend = FALSE)
```



Pointers and wrapping

We'll need to apply this grid of seas and oceans repeatedly — once for each `.nc` file — which means it has to get passed to all our worker cores. But it is stored as a pointer, so we can't do that directly. We need to wrap it:

```
# If we don't do this it can't be passed around
# across the processes that in_parallel() will spawn
seas_zonal_wrapped <- terra::wrap(seas_zonal)
```

NOAA Temperature Data

Write a function very similar to the other one.

```r
#' Process raster zonally
#'
#' Use terra to process the NCDF files
#'
#' @param fnames Vector of filenames
#' @param wrapped_seas wrapped object containing rasterized seas data
#'
#' @returns
#' @export
#'
#' @examples
process_raster_zonal <- function(fnames, wrapped_seas = seas_zonal_wrapped) {
  d <- terra::rast(fnames)
  wts <- terra::cellSize(d, unit = "km") # For scaling

  layer_varnames <- terra::varnames(d) # vector of layers
  date_seq <- rep(terra::time(d)) # vector of dates

  # New colnames for use post zonal calculation below
  new_colnames <- c("sea", paste(layer_varnames, date_seq, sep = "_"))

  # Better colnames
  tdf_seas <- d |>
    terra::rotate() |> # Convert 0 to 360 lon to -180 to +180 lon
```

NOAA Temperature Data

Try it on one record:

```r
process_raster_zonal(all_fnames[10000])
```

```
# A tibble: 101 × 6
   sea                       date          anom   err     ice   sst
   <chr>                     <chr>        <dbl> <dbl>   <dbl> <dbl>
 1 Adriatic Sea              2009-01-20  0.174  0.227 NaN     13.5
 2 Aegean Sea                2009-01-20  0.796  0.146 NaN     15.9
 3 Alboran Sea               2009-01-20 -0.692  0.168 NaN     14.8
 4 Andaman or Burma Sea      2009-01-20 -0.548  0.124 NaN     27.1
 5 Arabian Sea               2009-01-20  0.267  0.133 NaN     26.6
 6 Arafura Sea               2009-01-20  0.112  0.293 NaN     29.6
 7 Arctic Ocean              2009-01-20  0.0366 0.298   0.979 -1.74
 8 Baffin Bay                2009-01-20  0.130  0.279   0.953 -1.58
 9 Balearic (Iberian Sea)    2009-01-20  0.0595 0.130 NaN     14.0
10 Bali Sea                  2009-01-20 -0.134  0.138 NaN     28.7
# ℹ 91 more rows
```

We’ll tidy the date later. You can see we get 101 summary records per day.

NOAA Temperature Data

Try it on one *chunk* of records:

```r
process_raster_zonal(chunked_fnames[[1]])
```

```
# A tibble: 2,525 × 6
   sea          date         anom   err   ice   sst
   <chr>        <chr>       <dbl> <dbl> <dbl> <dbl>
 1 Adriatic Sea 1981-09-01 -0.737 0.167   NaN  23.0
 2 Adriatic Sea 1981-09-02 -0.645 0.176   NaN  23.1
 3 Adriatic Sea 1981-09-03 -0.698 0.176   NaN  22.9
 4 Adriatic Sea 1981-09-04 -0.708 0.248   NaN  22.9
 5 Adriatic Sea 1981-09-05 -1.05  0.189   NaN  22.5
 6 Adriatic Sea 1981-09-06 -1.02  0.147   NaN  22.4
 7 Adriatic Sea 1981-09-07 -0.920 0.141   NaN  22.4
 8 Adriatic Sea 1981-09-08 -0.832 0.140   NaN  22.5
 9 Adriatic Sea 1981-09-09 -0.665 0.162   NaN  22.6
10 Adriatic Sea 1981-09-10 -0.637 0.268   NaN  22.5
# ℹ 2,515 more rows
```

NOAA Temperature Data

Now parallelize it:

```r
tictoc::tic("Big op")
seameans_df <- chunked_fnames |>
  map(in_parallel(
    \(x) process_raster_zonal(x),
    process_raster_zonal = process_raster_zonal,
    seas_zonal_wrapped = seas_zonal_wrapped
  )) |>
  list_rbind() |>
  mutate(
    date = ymd(date),
    year = lubridate::year(date),
    month = lubridate::month(date),
    day = lubridate::day(date),
    yrday = lubridate::yday(date),
    season = season(date)
  )
tictoc::toc()
```

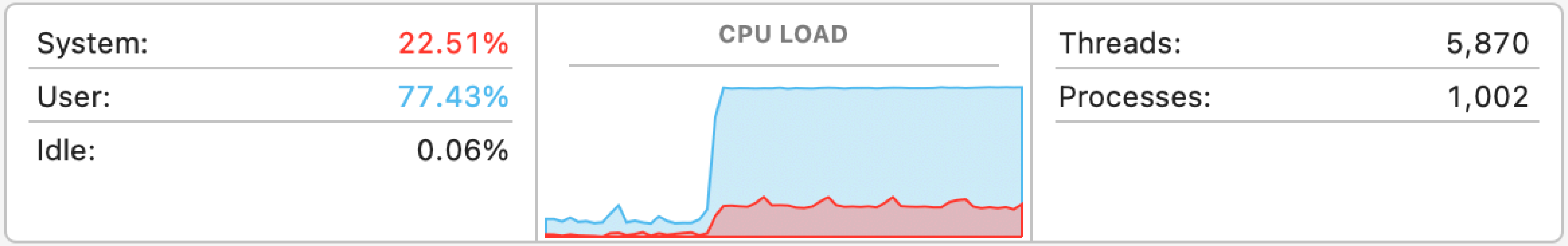
```
Big op: 141.848 sec elapsed
```

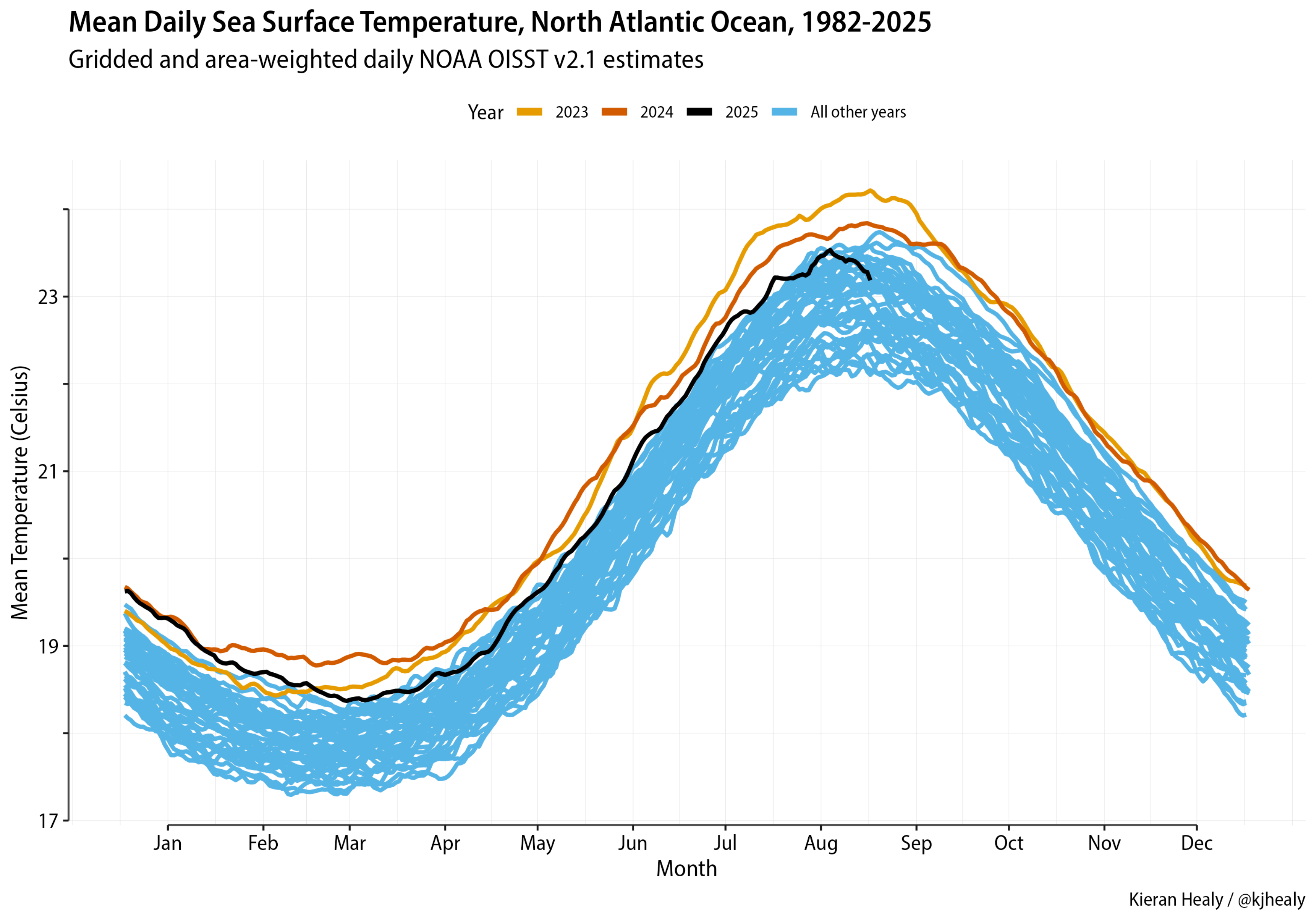
NOAA Temperature Data

```r
seameans_df
```

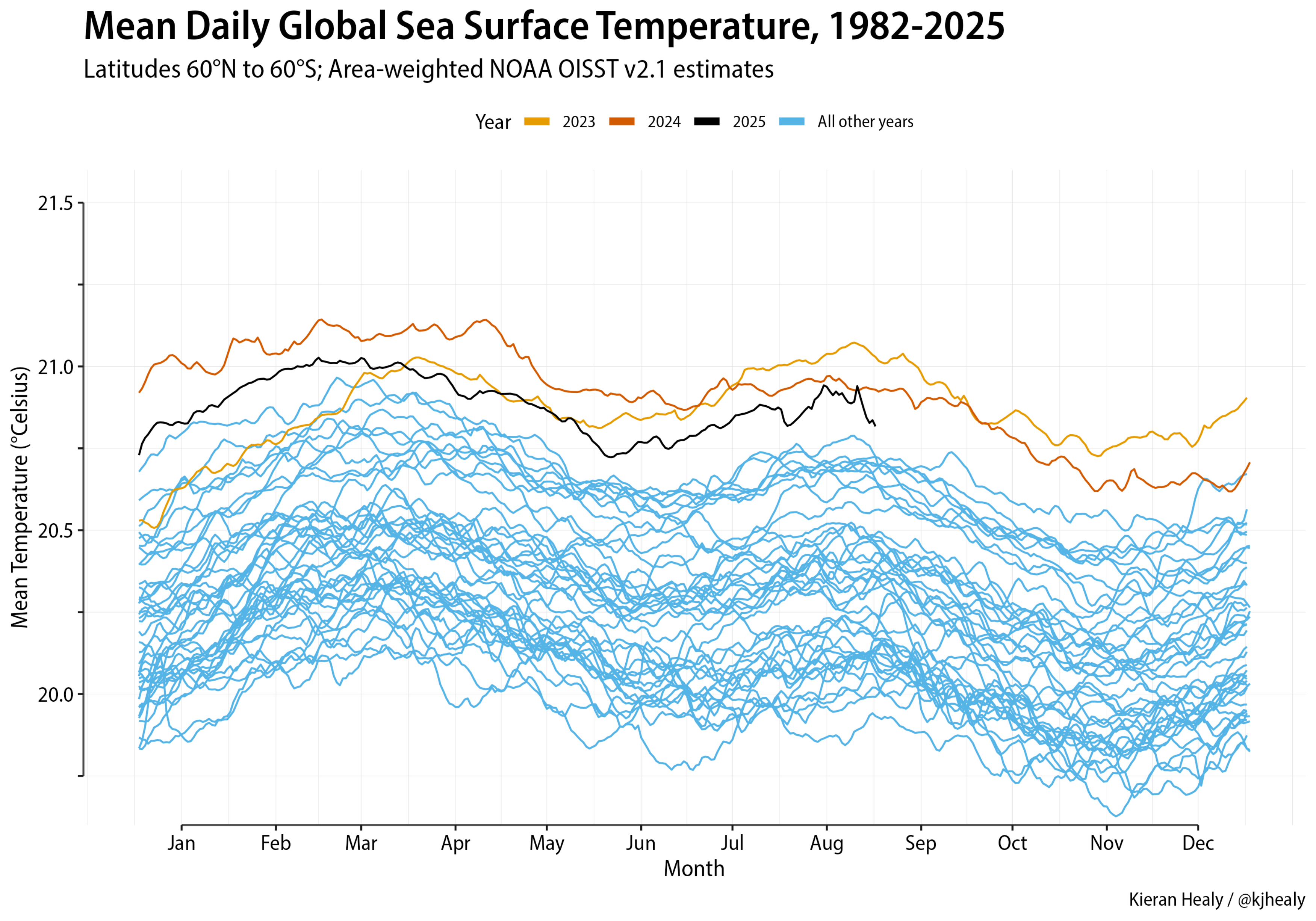
```
# A tibble: 1,622,565 × 11
   sea        date         anom   err   ice   sst  year month   day yrday season
   <chr>      <date>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <fct>
 1 Adriatic … 1981-09-01 -0.737 0.167   NaN  23.0  1981     9     1   244 Summer
 2 Adriatic … 1981-09-02 -0.645 0.176   NaN  23.1  1981     9     2   245 Autumn
 3 Adriatic … 1981-09-03 -0.698 0.176   NaN  22.9  1981     9     3   246 Autumn
 4 Adriatic … 1981-09-04 -0.708 0.248   NaN  22.9  1981     9     4   247 Autumn
 5 Adriatic … 1981-09-05 -1.05  0.189   NaN  22.5  1981     9     5   248 Autumn
 6 Adriatic … 1981-09-06 -1.02  0.147   NaN  22.4  1981     9     6   249 Autumn
 7 Adriatic … 1981-09-07 -0.920 0.141   NaN  22.4  1981     9     7   250 Autumn
 8 Adriatic … 1981-09-08 -0.832 0.140   NaN  22.5  1981     9     8   251 Autumn
 9 Adriatic … 1981-09-09 -0.665 0.162   NaN  22.6  1981     9     9   252 Autumn
10 Adriatic … 1981-09-10 -0.637 0.268   NaN  22.5  1981     9    10   253 Autumn
# ℹ 1,622,555 more rows
```

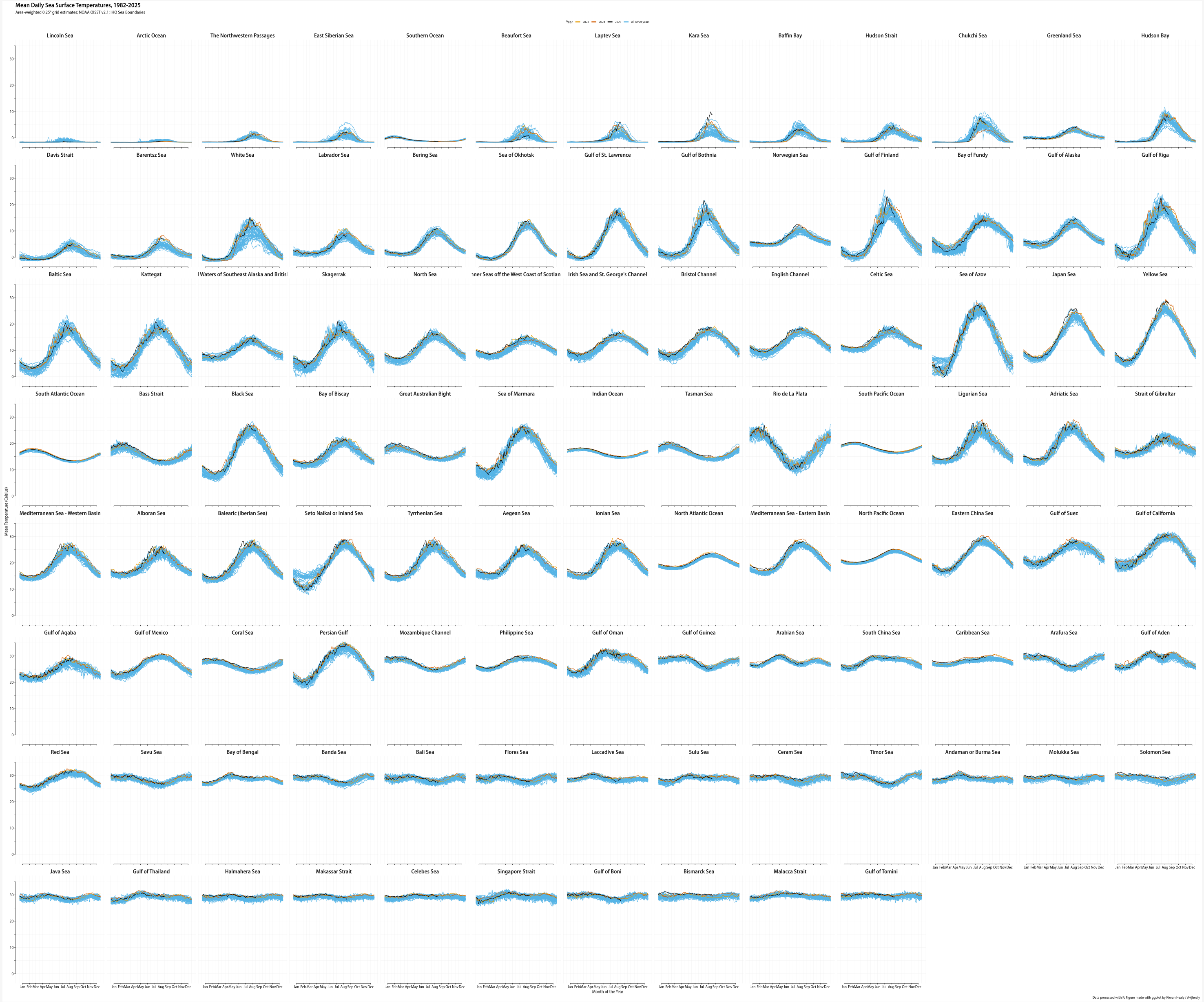
NOAA that's more like it





Mean Daily Sea Surface Temperature, North Atlantic Ocean, 1982-2025
Gridded and area-weighted daily NOAA OISST v2.1 estimates
Year
2023
2024
2025
All other years
Mean Temperature (Celsius)
23
21
19
17
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month
Kieran Healy / @kjhealy


Mean Daily Global Sea Surface Temperature, 1982-2025
Latitudes 60°N to 60°S; Area-weighted NOAA OISST v2.1 estimates
Year
2023
2024
2025
All other years
21.5
21.0
20.5
20.0
Mean Temperature (°Celsius)
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month
Kieran Healy / @kjhealy


Mean Daily Sea Surface Temperatures, 1982-2025
Area-weighted 0.25° grid estimates; NOAA OISST v2.1; IHO Sea Boundaries
Year 2023 2024 2025 All other years
Lincoln Sea
Arctic Ocean
The Northwestern Passages
East Siberian Sea
Southern Ocean
Beaufort Sea
Laptev Sea
Kara Sea
Baffin Bay
Hudson Strait
Chukchi Sea
Greenland Sea
Hudson Bay
Davis Strait
Barentsz Sea
White Sea
Labrador Sea
Bering Sea
Sea of Okhotsk
Gulf of St. Lawrence
Gulf of Bothnia
Norwegian Sea
Gulf of Finland
Bay of Fundy
Gulf of Alaska
Gulf of Riga
Baltic Sea
Kattegat
l Waters of Southeast Alaska and Britisl
Skagerrak
North Sea
ner Seas off the West Coast of Scotlan
Irish Sea and St. George's Channel
Bristol Channel
English Channel
Celtic Sea
Sea of Azov
Japan Sea
Yellow Sea
South Atlantic Ocean
Bass Strait
Black Sea
Bay of Biscay
Great Australian Bight
Sea of Marmara
Indian Ocean
Tasman Sea
Rio de La Plata
South Pacific Ocean
Ligurian Sea
Adriatic Sea
Strait of Gibraltar
Mediterranean Sea - Western Basin
Alboran Sea
Balearic (Iberian Sea)
Seto Naikai or Inland Sea
Tyrrhenian Sea
Aegean Sea
Ionian Sea
North Atlantic Ocean
Mediterranean Sea - Eastern Basin
North Pacific Ocean
Eastern China Sea
Gulf of Suez
Gulf of California
Gulf of Aqaba
Gulf of Mexico
Coral Sea
Persian Gulf
Mozambique Channel
Philippine Sea
Gulf of Oman
Gulf of Guinea
Arabian Sea
South China Sea
Caribbean Sea
Arafura Sea
Gulf of Aden
Red Sea
Savu Sea
Bay of Bengal
Banda Sea
Bali Sea
Flores Sea
Laccadive Sea
Sulu Sea
Ceram Sea
Timor Sea
Andaman or Burma Sea
Molukka Sea
Solomon Sea
Java Sea
Gulf of Thailand
Halmahera Sea
Makassar Strait
Celebes Sea
Singapore Strait
Gulf of Boni
Bismarck Sea
Malacca Strait
Gulf of Tomini
Mean Temperature (Celsius)
0
10
20
30
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Month of the Year
Data processed with R; Figure made with ggplot by Kieran Healy / @kjhealy