

# Rasters

Manipulate rasters with 

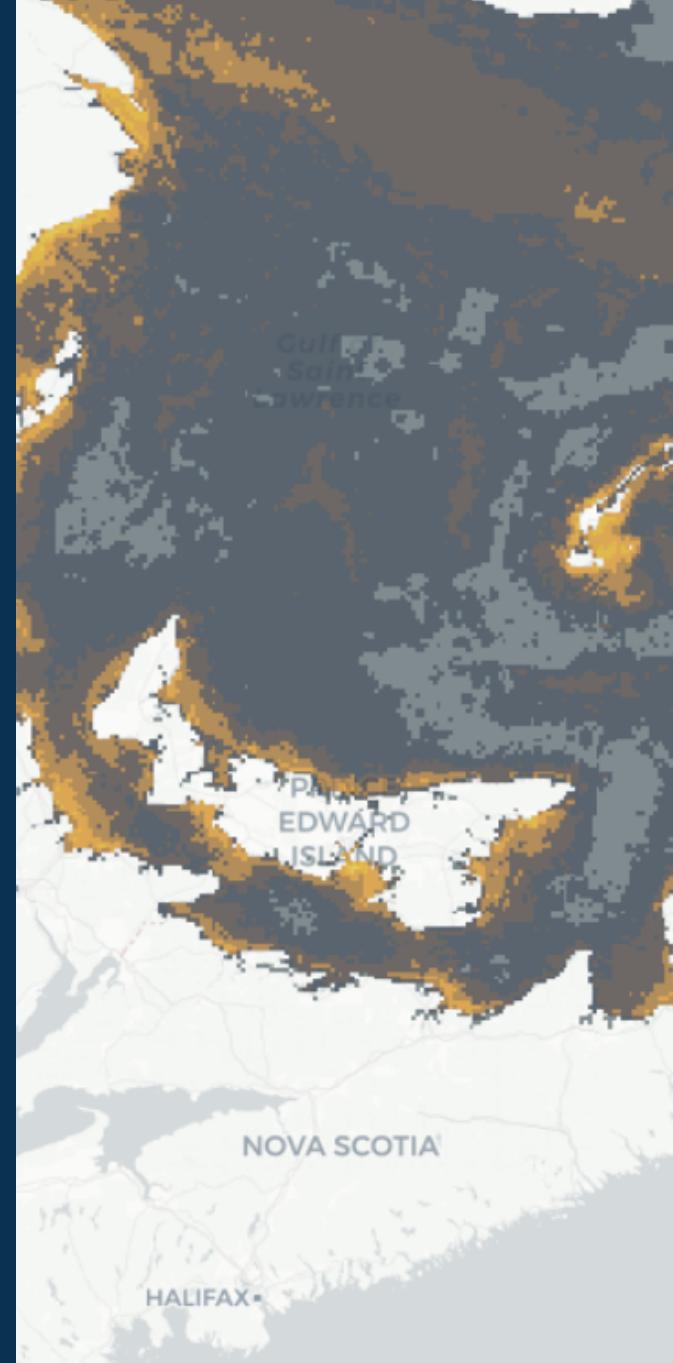
David Beauchesne & Kevin Cazelles

January 12, 2021



 deploy workshop passing

Content under  unless otherwise specified



# Learning objectives

---



# Learning objectives

Learn how to visualize raster interactively with 

1. Understand the benefits of manipulating raster files with 
2. Learn how to read and write raster files with 
3. Learn how to manipulate rasters 
4. Learn how to visualize rasters with 

## Rationalize

---

Why use  for rasters? (~15min)

# Raster files

Raster files are grids of equally sized (most often) cells (or pixels).

Every cell has one or several values (or missing values).

Cells are georeferenced: coordinates + CRS

Commonly used to represent continuous phenomena (e.g. temperature, elevation) and images (airborne or satellite imagery).

# Raster files

- Examples of raster file formats:
  - **GeoTIFF** (see <https://trac.osgeo.org/geotiff>)
  - **NetCDF** (see <https://www.unidata.ucar.edu/software/netcdf/>)
  - **KML** (also supports vector objects, see  
<https://www.ogc.org/standards/kml/>)
  - **GeoPackage** (also supports vector objects, see  
<https://www.ogc.org/standards/geopackage>)
-  More details :
  - <https://gisgeography.com/gis-formats/>
  - <https://gdal.org/drivers/raster/index.html>

R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. [Wikipedia](#)



# Coding vs clicking

1. Programming/coding allows to automate what is frequently done!
2. Allow to make analyses/reports reproducible
  - create data pipeline
  - for colleagues and future you
3. Increase efficiency (spend more time thinking)
4. Feeling empowered

# Why ?

1. Freeware + Open source
  2. **Package ecosystem** is rich and under active development
  3. Many packages connect  to other software/language
  4. Community is wide, quite involved, responsive
  5. You are most likely already using it! You are not starting from scratch
- **Very little cannot be done with **



# Input/Ouput

---

Read and write raster files (30min)

# packages ecosystem for spatial objects

## Classes and Methods

## Statistical Analyses

## Visualisation

## Data Retrieval

# R packages ecosystem for spatial objects

## Classes and Methods

### Vectors

- `sp`
- `sf` ✓

### Rasters (our main focus)

- `raster` ✓
- `stars` ✓
- `terra`

ℹ <https://keen-swartz-3146c4.netlify.app/raster.html#package-stars>

ℹ <https://github.com/rspatial> (15 packages)

# Raster packages

`raster` 03/2010 (1.0.0) // 11/2020 (3.3-7)

`terra` 03/2019 (0.5-2) // 11/2020 (0.9-11)

terra is an R package that replaces raster. It has a very similar interface, but it is simpler and much faster.

`stars` 07/2018 (0.1-1) // 07/2020 (0.4-3)

Reading, manipulating, writing and plotting spatiotemporal arrays (raster and vector data cubes) in 'R'.

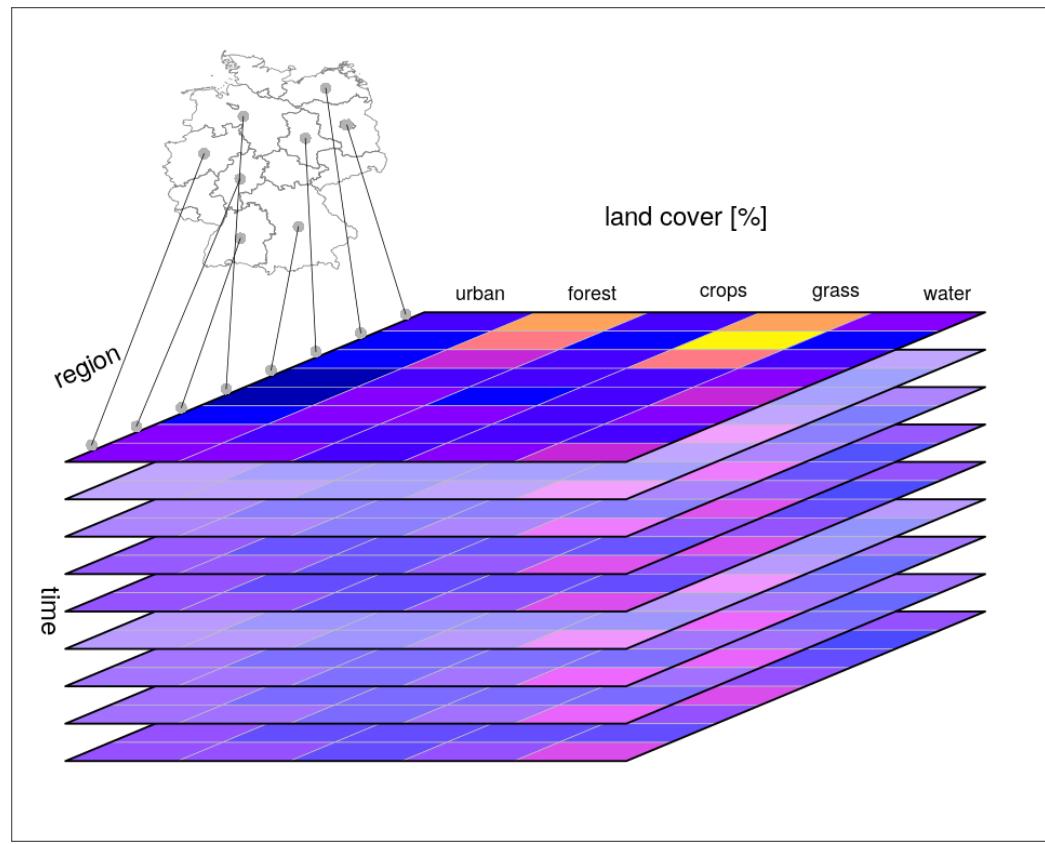
Depends on `sf`. Note that `stars` and `sf` are Tidyverse friendly.

# Package `raster`

- Well documented
  - see <https://rspatial.org/raster/>
  - many blog posts and tutorials (e.g. <https://datacarpentry.org/r-raster-vector-geospatial/01-raster-structure/>)
- 383 packages depend on or import `raster`
- Depends on `sp` but also works with `sf`.

# Package `stars`: SpatioTemporal ARrayS

Extent `sf` to Raster and Vector Datacubes



see <https://r-spatial.github.io/stars/index.html>

# Package `stars`: SpatioTemporal ARrayS

- Well documented too

The screenshot shows the documentation for the `stars` package. At the top, there's a navigation bar with tabs for "stars 0.4-4", "Home", "Reference", "Articles ▾", "Changelog", and a user icon. Below the navigation, there are two main sections: "Spatiotemporal" on the left and "Constructor Datacubes" on the right. The "Spatiotemporal" section contains a list of topics: 1. introduction, 2. stars proxy objects, 3. stars tidyverse methods, 4. stars data model, 5. vector-raster conversions, reprojection, warping, and 6. How 'raster' functions map to `stars` functions. The "Constructor Datacubes" section contains a list of topics: 1. introduction, 2. stars proxy objects, 3. stars tidyverse methods, 4. stars data model, 5. vector-raster conversions, reprojection, warping, and 6. How 'raster' functions map to `stars` functions. To the right of these sections is a "Links" sidebar with links to CRAN, source code, bug reports, a license (Apache License), and developer information.

see <https://r-spatial.github.io/stars/index.html>

- 9 packages depend on `stars`
- So far, less material available online.

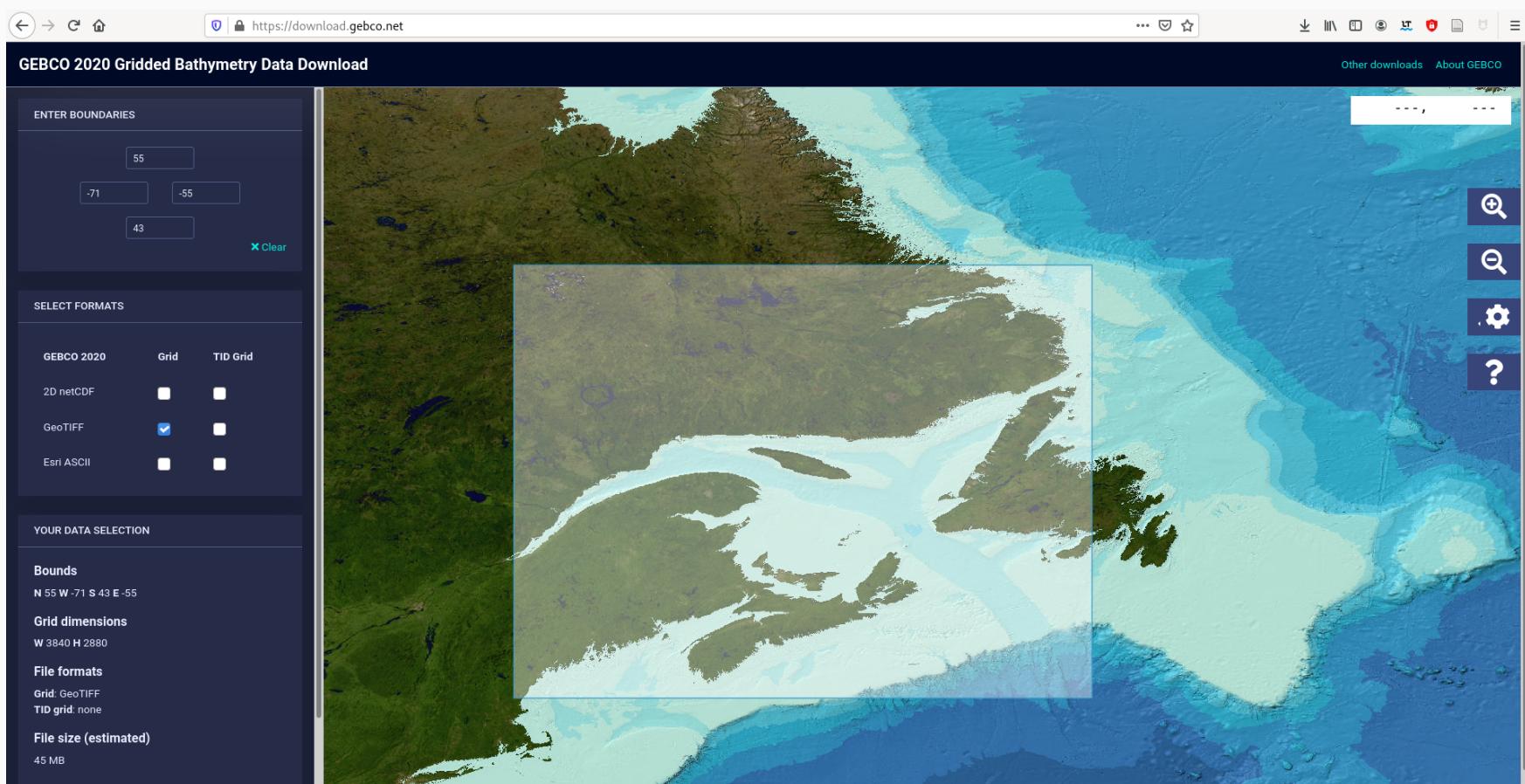
--  
We'll talk about both!

# Read a raster file



1. Files you created
2. Files you found on the Internet
3.  packages to download raster files

# Example file



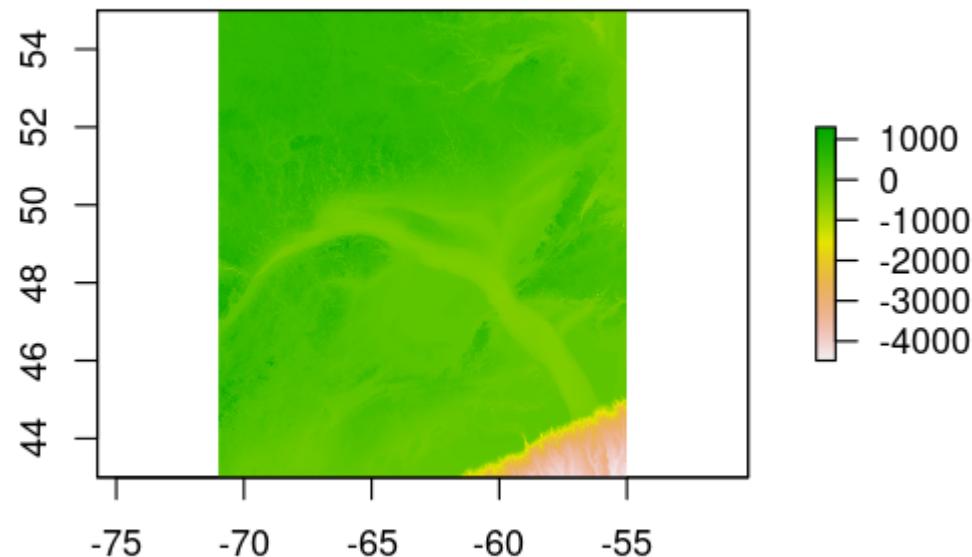
- <https://download.gebco.net/>
- [!\[\]\(1e590cb39b200e11edccc3e9ae0ab280\_img.jpg\) data\\_and\\_script.zip](#)

# Read with raster

```
library(raster)
rar <- raster("data/bathy.tif")
class(rar)
#> [1] "RasterLayer"
#> attr(,"package")
#> [1] "raster"
```

# Read with raster

```
plot(rar)
```

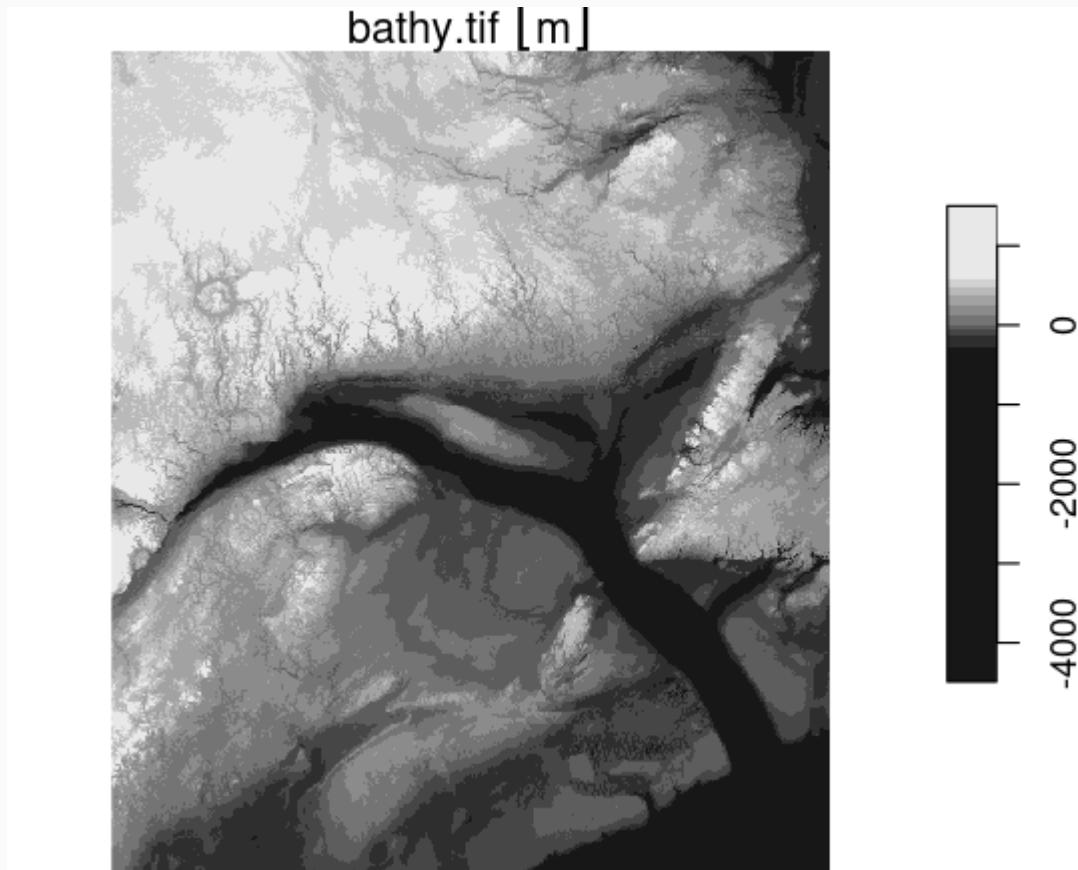


# Read with stars

```
library(stars)
ras ← read_stars("data/bathy.tif")    # argument `driver` to specify the driver
class(ras)
#> [1] "stars"
```

# Read with stars

```
plot(ras)
```



So ...

- **GeoTIFF ✓**
- **Other formats, how do we know?**

# Check read/write capabilities with `raster`

! `rgdal` is required

```
ra_dr ← rgdal::gdalDrivers()
head(ra_dr)
#>      name          long_name create  copy isRaster
#> 1 AAIGrid      Arc/Info ASCII Grid FALSE  TRUE   TRUE
#> 2 ACE2          ACE2        FALSE FALSE   TRUE
#> 3 ADRG  ARC Digitized Raster Graphics  TRUE FALSE   TRUE
#> 4 AIG           Arc/Info Binary Grid FALSE FALSE   TRUE
#> 5 AirSAR        AirSAR Polarimetric Image FALSE FALSE   TRUE
#> 6 ARG           Azavea Raster Grid format FALSE  TRUE   TRUE
```

```
ra_dr[which(ra_dr$name == "GTiff"), ]
#>      name long_name create copy isRaster
#> 57 GTiff  GeoTIFF    TRUE TRUE     TRUE
```

# Check read/write capabilities with stars

! sf is required

```
st_dr ← sf::st_drivers(what = "raster")
head(st_dr)
#>          name           long_name write
#> VRT      VRT           Virtual Raster  TRUE
#> DERIVED DERIVED Derived datasets using VRT pixel functions FALSE
#> GTiff    GTiff          GeoTIFF   TRUE
#> COG      COG           Cloud optimized GeoTIFF generator FALSE
#> NITF     NITF          National Imagery Transmission Format  TRUE
#> RPFTOC  RPFTOC         Raster Product Format TOC format FALSE
#>          copy is_raster is_vector  vsi
#> VRT      TRUE    TRUE    FALSE  TRUE
#> DERIVED FALSE   TRUE    FALSE  FALSE
#> GTiff    TRUE    TRUE    FALSE  TRUE
#> COG      TRUE    TRUE    FALSE  TRUE
#> NITF     TRUE    TRUE    FALSE  TRUE
#> RPFTOC  FALSE   TRUE    FALSE  TRUE
```

# Check read/write capabilities with stars

```
st_dr[which(st_dr$name == "GTiff"), ]  
#>      name long_name write copy is_raster is_vector  vsi  
#> GTiff  GTiff    Geotiff   TRUE  TRUE      TRUE      FALSE  TRUE
```

ⓘ <https://gdal.org/drivers/raster/index.html>

ⓘ <https://keen-swartz-3146c4.netlify.app/intro.html#reading>

# Write raster files



- Share the output file
- Use another software (e.g. QGIS)

! Create an `output` directory

```
dir.create("output", showWarnings = FALSE)
```

# Write with raster

```
ra_dr$name[ra_dr$create]
#> [1] "ADRG"        "BAG"          "BMP"          "BT"           "BYN"
#> [6] "CTable2"      "EHdr"         "ELAS"         "ENVI"         "ERS"
#> [11] "FITS"        "GPKG"         "GS7BG"        "GSBG"         "GTiff"
#> [16] "GTX"         "HDF4Image"    "HFA"          "IDA"          "ILWIS"
#> [21] "INGR"        "ISCE"          "ISIS2"        "ISIS3"        "KRO"
#> [26] "LAN"          "Leveller"      "MBTiles"      "MEM"          "MFF"
#> [31] "MFF2"         "MRF"           "netCDF"       "NGW"          "NITF"
#> [36] "NTv2"         "NWT_GRD"      "PAux"         "PCIDSK"      "PCRaster"
#> [41] "PDF"          "PDS4"          "PNM"          "RMF"          "ROI_PAC"
#> [46] "RRASTER"     "RST"           "SAGA"         "SGI"          "Terragen"
#> [51] "VICAR"        "VRT"
```

# Write with `raster`

```
writeRaster(rar, filename = "output/rar.gpkg", format = "GPKG", overwrite = TRUE)
```

# Write with stars

```
sort(st_dr$name[st_dr$write])  
#> [1] "ADRG"      "BAG"       "BMP"       "BT"        "BYN"  
#> [6] "CTable2"    "EHdr"      "ELAS"      "ENVI"      "ERS"  
#> [11] "FITS"      "GPKG"      "GS7BG"     "GSBG"      "GTiff"  
#> [16] "GTX"       "HDF4Image" "HFA"       "IDA"       "ILWIS"  
#> [21] "INGR"      "ISCE"      "ISIS2"     "ISIS3"     "KRO"  
#> [26] "LAN"        "Leveller"   "MBTiles"   "MEM"       "MFF"  
#> [31] "MFF2"      "MRF"       "netCDF"    "NGW"       "NITF"  
#> [36] "NTv2"       "NWT_GRD"   "PAux"      "PCIDSK"    "PCRaster"  
#> [41] "PDF"        "PDS4"      "PNM"       "RMF"       "ROI_PAC"  
#> [46] "RRASTER"   "RST"       "SAGA"      "SGI"       "Terragen"  
#> [51] "VICAR"     "VRT"
```

# Write with stars

```
write_stars(ras, dsn = "output/ras.gpkg", driver = "GPKG")
```

 dsn stands for **data source name**.

 Recap

Action	<i>raster</i>	<i>stars</i>
list drivers	<code>rgdal::gdalDrivers()</code>	<code>sf::st_drivers()</code>
read files	<code>raster()</code>	<code>read_stars()</code>
write files	<code>writeRaster()</code>	<code>write_stars()</code>

# >\_ R as a raster file converter

- ☆ Convert `bathy.tif` into `bathy.grd`  
(<https://gdal.org/drivers/raster/rraster.html#raster-rraster>)
- ☆ ☆ Create a function `conv_raster()` that handles any conversion supported.
- ☆ ☆ ☆ Create a function that converts a object of class `RasterLayer` object to a `stars` one (and conversely)

10:00

# Solution ☆

Convert `bathy.tif` into `bathy.grd`

```
# raster
rr ← raster("data/bathy.tif")
writeRaster(rr, filename = "output/rar.grd", overwrite = TRUE)
```

```
# stars
rs ← read_stars("data/bathy.tif")
write_stars(rs, dsn = "output/ras.grd", driver = "rraster")
```

# Solution ★ ★

Create a function `conv_raster()` that handles any conversion supported.

```
# raster
conv_raster_r <- function(input, output, ... ) {
  rr <- raster(input)
  writeRaster(rr, filename = output, ...)
}
conv_raster_r("data/bathy.tif", "output/rar2.grd", overwrite = TRUE)
```

```
# stars
conv_raster_s <- function(from, to, ... ) {
  rs <- read_stars("data/bathy.tif")
  write_stars(rs, dsn = "output/ras.grd", ...)
}
conv_raster_s("data/bathy.tif", "output/ras2.grd", driver = "rraster")
```

# Solution ★ ★ ★

Create a function that converts a object of class `RasterLayer` object to a `stars` one (and conversely)

```
conv_raster_stars <- function(x) {  
  if (class(x) == "RasterLayer") {  
    tmp <- tempfile(fileext = ".tif")  
    writeRaster(x, tmp)  
    read_stars(tmp)  
  } else if (class(x) == "stars") {  
    tmp <- tempfile(fileext = ".tif")  
    write_stars(x, tmp)  
    raster(tmp)  
  } else {  
    warning("x should be of class `stars` or `RasterLayer`")  
    NULL  
  }  
}  
ras_n <- conv_raster_stars(rar)  
rar_n <- conv_raster_stars(ras)  
# conv_raster_stars(1)
```

 Manipulate

---

Do whatever you need with raster files (~1h15)

# Common operations



# Common operations



- change projection
- basic calculations (e.g. average per regions, min/max)
- crop/mask
- extraction
- resample
- etc.

# What do we have? `raster`

```
class(rar)
#> [1] "RasterLayer"
#> attr(,"package")
#> [1] "raster"
rar
#> class      : RasterLayer
#> dimensions : 2880, 3840, 11059200 (nrow, ncol, ncell)
#> resolution : 0.004166667, 0.004166667 (x, y)
#> extent     : -71, -55, 43, 55 (xmin, xmax, ymin, ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : /home/kevcaz/Projects/inSilecoInc/workshop_raster/data/bathy.tif
#> names      : bathy
#> values     : -32768, 32767 (min, max)
```

# What do we have? raster

```
values(rar)
```

```
#> [1] 457 461 463 463 467 474 484 490 478 460 452 455 456 454 451  
#> [16] 450 449 447 446 442 443 453 459 452 437 429 432 440 446 451  
#> [31] 460 465 466 472 485 498 506 502 493 493 498 502 507 506 479  
#> [46] 457 453 453 453 455 458 460 460 457 455 453 454 456 457 464  
#> [61] 472 484 508 537 555 557 560 567 565 564 563 558 558 566 565  
#> [76] 559 555 547 541 549 557 560 556 557 568 575 573 567 564 565  
#> [91] 568 569 570 570 569 561 549 547 564 582 583 582 586 588 583  
#> [106] 572 562 554 547 543 540 540 551 560 560 555 562 576 573 561  
#> [121] 553 554 556 554 553 556 556 550 536 522 515 512 510 510 509  
#> [136] 507 506 505 505 507 510 514 519 524 529 536 539 531 522 520  
#> [151] 521 522 527 538 550 559 560 552 542 534 532 537 543 544 545  
#> [166] 545 542 535 533 533 535 544 556 568 580 590 596 603 604 604  
#> [181] 610 609 600 591 585 570 558 554 543 538 546 558 566 563 547  
#> [196] 537 538 541 536 524 518 518 514 506 504 505 506 507 512 522  
#> [211] 529 521 505 499 502 505 505 502 498 499 496 500 518 532 523  
#> [226] 503 496 492 489 492 505 522 531 515 498 502 511 520 519 511  
#> [241] 494 488 481 477 476 476 477 476 475 475 475 475 475 475 477  
#> [256] 482 489 492 495 499 504 506 505 507 512 513 504 492 488 495  
#> [271] 511 517 505 491 491 504 530 552 560 571 576 556 526 504 495  
#> [286] 490 483 476 471 469 473 480 482 483 483 483 482 480 481 483  
#> [301] 484 484 485 487 485 482 488 501 500 488 486 493 507 526 541  
#> [316] 550 563 566 557 550 549 553 561 558 546 532 511 489 484 497
```

# What do we have? `raster`

```
dim(rar)
#> [1] 2880 3840      1
ncell(rar)
#> [1] 11059200
projection(rar)
#> [1] "+proj=longlat +datum=WGS84 +no_defs"
bbox(rar)
#>   min max
#> s1 -71 -55
#> s2  43  55
extent(rar)
#> class       : Extent
#> xmin        : -71
#> xmax        : -55
#> ymin        : 43
#> ymax        : 55
```

# What do we have? stars

```
class(ras)
#> [1] "stars"
ras
#> stars object with 2 dimensions and 1 attribute
#> attribute(s), summary of first 1e+05 cells:
#>   bathy.tif [m]
#>   Min.    :-435.0
#>   1st Qu.: 23.0
#>   Median  : 479.0
#>   Mean    : 299.5
#>   3rd Qu.: 540.0
#>   Max.    : 812.0
#> dimension(s):
#>   from    to offset      delta refsys point values
#> x     1 3840     -71  0.00416667 WGS 84 FALSE   NULL [x]
#> y     1 2880      55 -0.00416667 WGS 84 FALSE   NULL [y]
```

# What do we have? stars

```
class(ras[[1]])
#> [1] "units"
ras[[1]]
#> Units: [m]
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
#>      [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21]
#>      [,22] [,23] [,24] [,25] [,26] [,27] [,28] [,29] [,30] [,31]
#>      [,32] [,33] [,34] [,35] [,36] [,37] [,38] [,39] [,40] [,41]
#>      [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50] [,51]
#>      [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61]
#>      [,62] [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71]
#>      [,72] [,73] [,74] [,75] [,76] [,77] [,78] [,79] [,80] [,81]
#>      [,82] [,83] [,84] [,85] [,86] [,87] [,88] [,89] [,90] [,91]
#>      [,92] [,93] [,94] [,95] [,96] [,97] [,98] [,99] [,100]
#>      [,101] [,102] [,103] [,104] [,105] [,106] [,107] [,108]
#>      [,109] [,110] [,111] [,112] [,113] [,114] [,115] [,116]
#>      [,117] [,118] [,119] [,120] [,121] [,122] [,123] [,124]
#>      [,125] [,126] [,127] [,128] [,129] [,130] [,131] [,132]
#>      [,133] [,134] [,135] [,136] [,137] [,138] [,139] [,140]
#>      [,141] [,142] [,143] [,144] [,145] [,146] [,147] [,148]
#>      [,149] [,150] [,151] [,152] [,153] [,154] [,155] [,156]
#>      [,157] [,158] [,159] [,160] [,161] [,162] [,163] [,164]
#>      [,165] [,166] [,167] [,168] [,169] [,170] [,171] [,172]
```

# What do we have? stars

```
dim(ras)
#>      x      y
#> 3840 2880
ncell(ras)
#>      x
#> 11059200
st_bbox(ras)
#> xmin ymin xmax ymax
#> -71   43   -55   55
```

# What do we have? stars

```
ras_crs ← st_crs(ras)
class(ras_crs)
#> [1] "crs"
ras_crs
#> Coordinate Reference System:
#>   User input: WGS 84
#>   wkt:
#> GEOGCRS["WGS 84",
#>   DATUM["World Geodetic System 1984",
#>     ELLIPSOID["WGS 84",6378137,298.257223563,
#>       LENGTHUNIT["metre",1]],
#>   PRIMEM["Greenwich",0,
#>     ANGLEUNIT["degree",0.0174532925199433]],
#>   CS[ellipsoidal,2],
#>     AXIS["geodetic latitude (Lat)",north,
#>       ORDER[1],
#>       ANGLEUNIT["degree",0.0174532925199433]],
#>     AXIS["geodetic longitude (Lon)",east,
#>       ORDER[2],
#>       ANGLEUNIT["degree",0.0174532925199433]],
#>   ID["EPSG",4326]]
```

# What do we have? stars

```
ras_crs$input  
#> [1] "WGS 84"  
ras_crs$proj4string  
#> [1] "+proj=longlat +datum=WGS84 +no_defs"  
ras_crs$epsg  
#> [1] 4326
```

- <https://epsg.io/>
- <https://epsg.io/4326>

# Conversion between stars and raster

Use a temporary file (see above)

```
rar_c ← st_as_stars(rar)
class(rar_c)
#> [1] "stars"
```

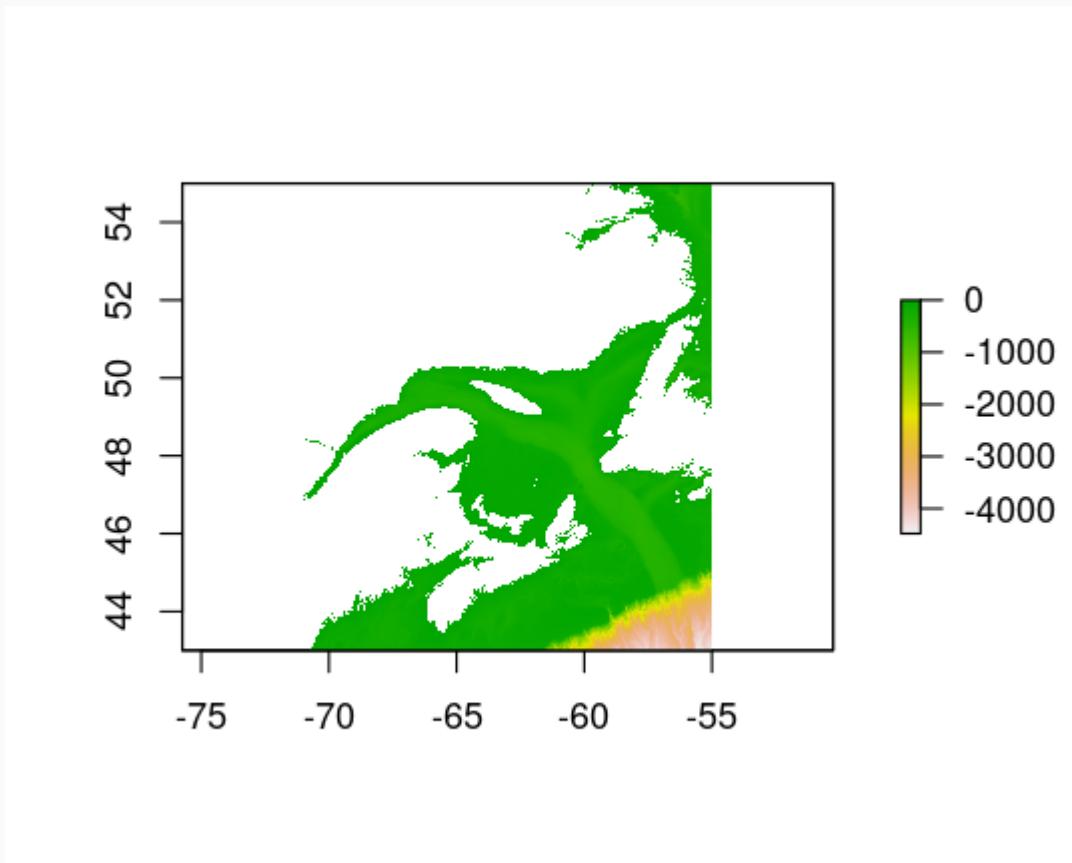
```
ras_c ← as(ras, "Raster")
class(ras_c)
#> [1] "RasterLayer"
#> attr(, "package")
#> [1] "raster"
```

# Basic computations with `raster`

```
class(rar[,])
#> [1] "integer"
rar[1, 1]
#>
#> 457
rar[1:10, 11:20]
#> [1] 452 455 456 454 451 450 449 447 446 442 493 502 512 507 484 472
#> [17] 468 462 450 440 529 532 536 523 491 468 459 454 440 431 504 507
#> [33] 503 485 464 451 441 435 431 428 459 455 445 438 434 431 429 429
#> [49] 429 428 444 439 434 431 434 435 431 430 433 441 438 435 436 443
#> [65] 449 445 441 441 445 449 439 444 447 449 450 447 443 440 439 440
#> [81] 441 442 443 444 444 444 441 441 440 443 452 469 460 448 443 443 447
#> [97] 446 445 447 454
mean(rar[,]) # or mean(as.matrix(rar))
#> [1] 59.65972
quantile(rar[,])
#>    0%   25%   50%   75% 100%
#> -4490    -94    114    439   1502
```

# Basic computations with `raster`

```
rar2 <- rar # create a copy  
rar2[rar > 0] <- NA # filter  
plot(rar2)
```



# Basic cell-based computations with stars

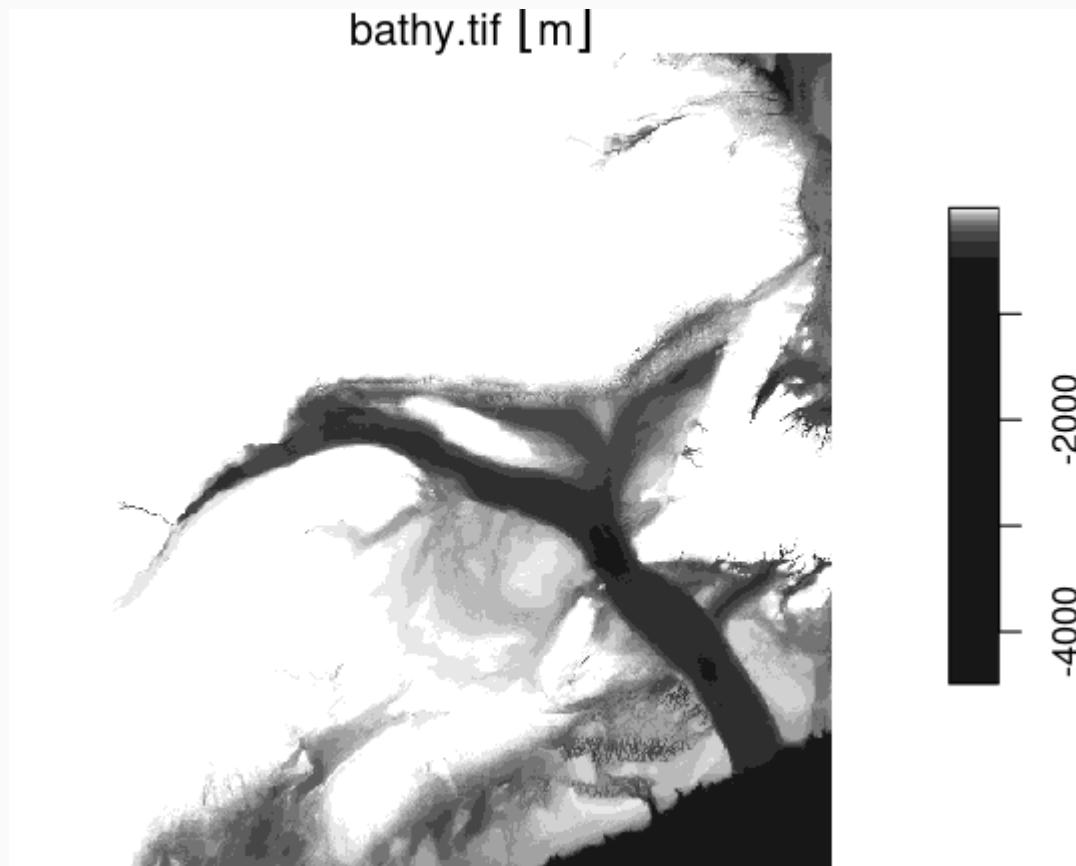
```
# extract value
class(ras[[1]])
#> [1] "units"
ras[[1]][1, 1]
#> 457 [m]
ras[[1]][1:10, 11:20]
#> Units: [m]
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
#> [1,] 455  475  479  476  474  473  488  529  525  521
#> [2,] 453  472  473  471  476  474  494  533  533  518
#> [3,] 454  471  471  471  477  475  493  539  536  515
#> [4,] 454  469  470  472  472  473  487  530  528  511
#> [5,] 456  469  470  475  472  476  493  519  520  508
#> [6,] 459  474  471  481  478  488  508  523  525  507
#> [7,] 459  484  473  494  487  502  519  537  537  513
#> [8,] 467  497  479  514  495  502  525  554  542  522
#> [9,] 483  506  483  526  497  495  531  554  535  525
#> [10,] 508  507  485  520  490  485  530  543  527  513
```

# Basic cell-based computations with stars

```
mean(ras[[1]])  
#> 59.65972 [m]  
quantile(ras[[1]])  
#> Units: [m]  
#>    0%   25%   50%   75% 100%  
#> -4490    -94    114    439  1502
```

# Basic cell-based computations with stars

```
ras2 <- ras # create a copy  
ras2[[1]][ras[[1]] > units::as_units(0, "m")] <- NA # filter  
plot(ras2)
```



# >\_ Mean along latitude and longitude

- ★ Compute the mean elevation (values > 0) and depth (values < 0) for `bathy.tif`.
- ★ ★ Create a function `mean_long_lat()` that computes mean elevation/depth along longitude (x axis) and latitude (y axis) (★ ★ ★ include an argument to set an elevation/depth filter).

15:00

# Solution ☆

Compute the mean elevation/depth along the latitude or/and longitude  
(optional: use elevation/depth filter)

```
# raster
rr_d <- rr_e <- rar
rr_d[rar > 0] <- NA
rr_e[rar < 0] <- NA
mean(values(rr_d), na.rm = TRUE)
#> [1] -392.727
mean(values(rr_e), na.rm = TRUE)
#> [1] 364.2634
```

# Solution ☆

Compute the mean elevation/depth along the latitude or/and longitude  
(optional: use elevation/depth filter)

```
# stars
rs_e <- rs_d <- ras
rs_d[[1]][ras[[1]] > units::as_units(0, "m")] <- NA # filter
rs_e[[1]][ras[[1]] < units::as_units(0, "m")] <- NA
mean(rs_d[[1]], na.rm = TRUE)
#> -392.727 [m]
mean(rs_e[[1]], na.rm = TRUE)
#> 364.2634 [m]
```

# Solution ★★ / ★★★

Create a function `mean_long_lat()` that computes mean elevation/depth along longitude and latitude (include an argument to set an elevation/depth filter).

```
# raster
mean_long_lat_r <- function(x, gt_na = NULL, lt_na = NULL) {
  m <- as.matrix(x)
  if (!is.null(lt_na)) m[m < lt_na] <- NA
  if (!is.null(gt_na)) m[m > gt_na] <- NA
  list(
    m_lat = apply(m, 1, mean, na.rm = TRUE),
    m_lon = apply(m, 2, mean, na.rm = TRUE)
  )
}
```

# Solution ★★ / ★★★

```
res1 ← mean_long_lat_r(rar)
par(mfrow = c(1, 2))
plot(res1$m_lat, main = "Along latitude", ylab = "elevation", xlab = "lat_index")
plot(res1$m_lon, main = "Along longitude", ylab = "elevation", xlab = "lon_index")
```

# Solution ★★ / ★★★

```
res2 ← mean_long_lat_r(rar, gt_na = 0)
par(mfrow = c(1, 2))
plot(res2$m_lat, main = "Along latitude", ylab = "elevation", xlab = "lat_index")
plot(res2$m_lon, main = "Along longitude", ylab = "elevation", xlab = "lon_index")
```

# Solution ★★ / ★★★

```
# stars
mean_long_lat_s <- function(x, gt_na = NULL, lt_na = NULL) {
  m <- units::drop_units(ras[[1]])
  if (!is.null(lt_na)) m[m < lt_na] <- NA
  if (!is.null(gt_na)) m[m > gt_na] <- NA
  list(
    m_lon = apply(m, 1, mean, na.rm = TRUE),
    m_lat = apply(m, 2, mean, na.rm = TRUE)
  )
}
```

# Solution ★★ / ★★★

```
res1 <- mean_long_lat_s(ras)
par(mfrow = c(1, 2))
plot(res1$m_lon, main = "Along latitude", ylab = "elevation", xlab = "lat_index")
plot(res1$m_lat, main = "Along longitude", ylab = "elevation", xlab = "lon_index")
```

# Solution ★★ / ★★★

```
res2 ← mean_long_lat_s(ras, gt_na = 0)
par(mfrow = c(1, 2))
plot(res2$m_lon, main = "Along latitude", ylab = "elevation", xlab = "lat_index")
plot(res2$m_lat, main = "Along longitude", ylab = "elevation", xlab = "lon_index")
```

# Change map projection

- Reminder
  - [W List of map projections](#)
  - [Interactive examples](#)
- Let's use the spherical Mercator
  - <https://epsg.io/>
  - <https://epsg.io/3857>

# Change map projection with `raster`

- `raster` uses Proj strings
- <https://epsg.io/3857>

```
rar_crs ← CRS( "+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +ellps=WGS84 +units=m +no_defs" )
```

# Change map projection with `raster`

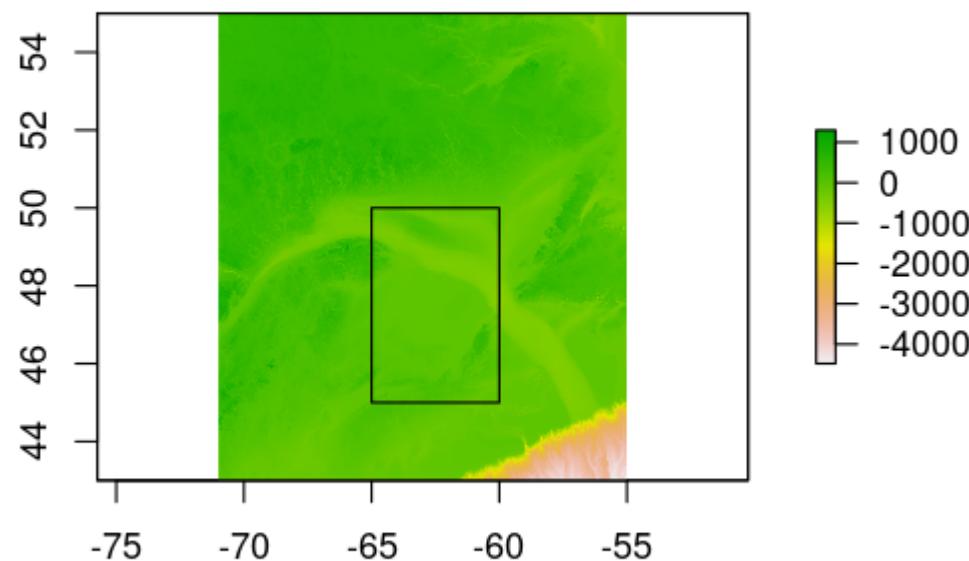
```
rar_t <- projectRaster(rar, crs = rar_crs)
projection(rar_t)
#> [1] "+proj=merc +a=6378137 +b=6378137 +lat_ts=0 +lon_0=0 +x_0=0 +y_0=0 +k=1 +units=m +
plot(rar)
```

# Change map projection with stars

```
ras_t ← st_transform(ras, crs = 3857)
st_crs(ras_t)
#> Coordinate Reference System:
#>   User input: EPSG:3857
#>   wkt:
#> PROJCRS["WGS 84 / Pseudo-Mercator",
#>   BASEGEOGCRS["WGS 84",
#>     DATUM["World Geodetic System 1984",
#>       ELLIPSOID["WGS 84",6378137,298.257223563,
#>         LENGTHUNIT["metre",1]],
#>     PRIMEM["Greenwich",0,
#>       ANGLEUNIT["degree",0.0174532925199433]],
#>     ID["EPSG",4326]],
#>   CONVERSION["Popular Visualisation Pseudo-Mercator",
#>     METHOD["Popular Visualisation Pseudo Mercator",
#>       ID["EPSG",1024]],
#>     PARAMETER["Latitude of natural origin",0,
#>       ANGLEUNIT["degree",0.0174532925199433],
#>       ID["EPSG",8801]],
#>     PARAMETER["Longitude of natural origin",0,
#>       ANGLEUNIT["degree",0.0174532925199433],
#>       ID["EPSG",8802]],
#>     PARAMETER["False easting",0,
```

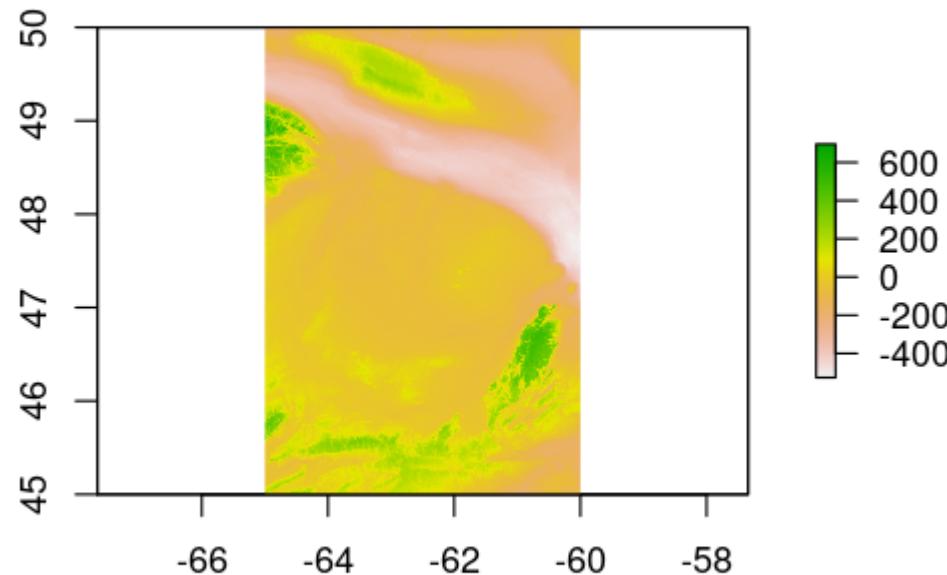
# Crop

```
plot(rar)
rect(-65, 45, -60, 50)
```



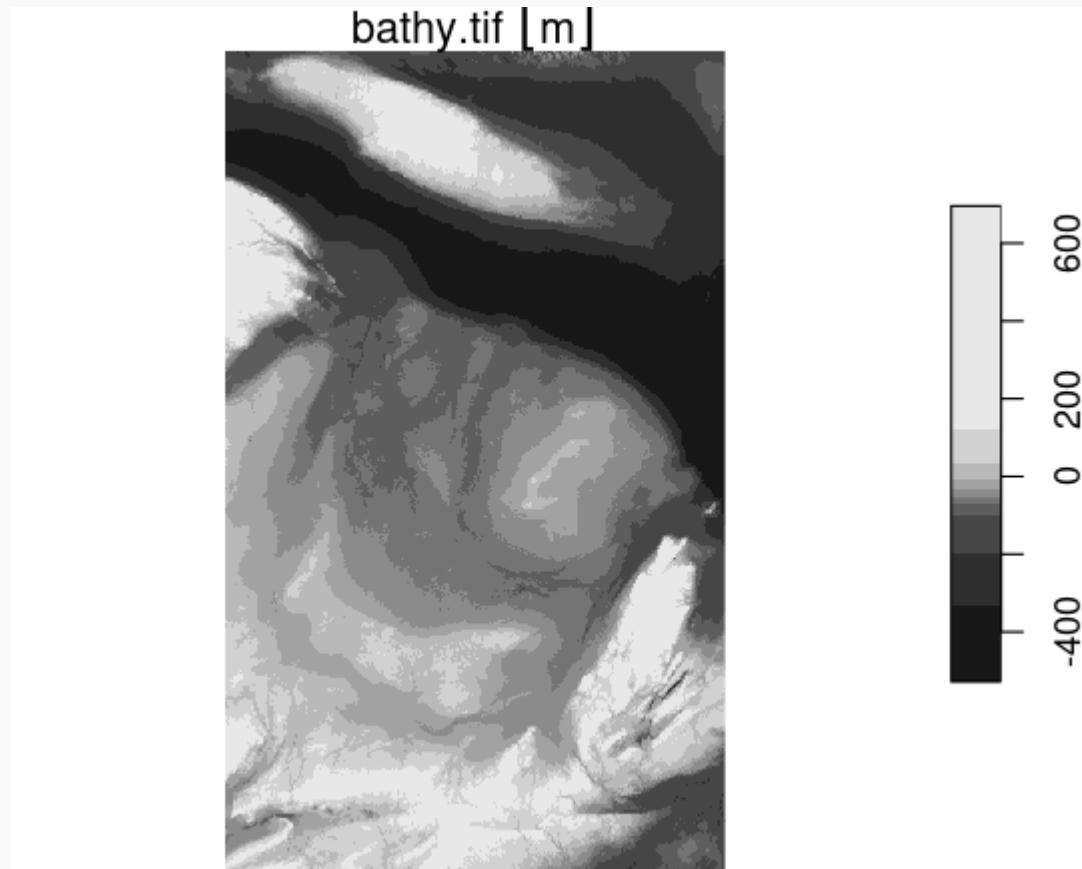
# Crop with raster

```
rar_c <- crop(rar, extent(-65, -60, 45, 50))  
plot(rar_c)
```



# Crop with stars

```
ext ← st_bbox(c(xmin = -65, xmax = -60, ymin = 45, ymax = 50), crs = 4326)
ras_c ← st_crop(ras, ext)
plot(ras_c)
```



# Mask

Let's use a shapefile of the Gulf of the St-Lawrence

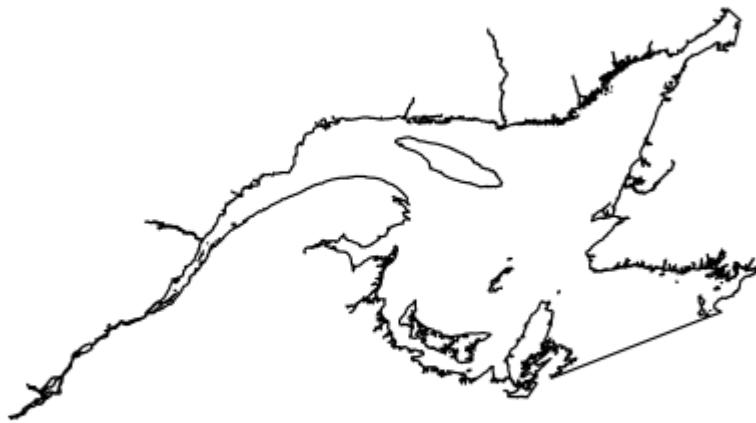
🔗 <https://www.marineregions.org/gazetteer.php?p=details&id=4290>

```
(stl ← sf::st_read("data/st_laurence.geojson"))
#> Reading layer `data_stl2` from data source `/home/kevcaz/Projects/inSilecoInc/workshop
#> Simple feature collection with 1 feature and 10 fields
#> geometry type:  POLYGON
#> dimension:      XY
#> bbox:           xmin: -74.86481 ymin: 44.9585 xmax: -54.70345 ymax: 52.22242
#> geographic CRS: WGS 84
#> Simple feature collection with 1 feature and 10 fields
#> geometry type:  POLYGON
#> dimension:      XY
#> bbox:           xmin: -74.86481 ymin: 44.9585 xmax: -54.70345 ymax: 52.22242
#> geographic CRS: WGS 84
#>
#>             name id longitude latitude     min_x   min_y
#> 1  Gulf of St. Lawrence 24 -61.56657 48.38072 -74.86481 44.9585
#>                 max_x   max_y    area mrgid          geometry
#> 1 -54.70345 52.22242 290874  4290 POLYGON ((-64.67242 47.7168 ...
```

# Mask

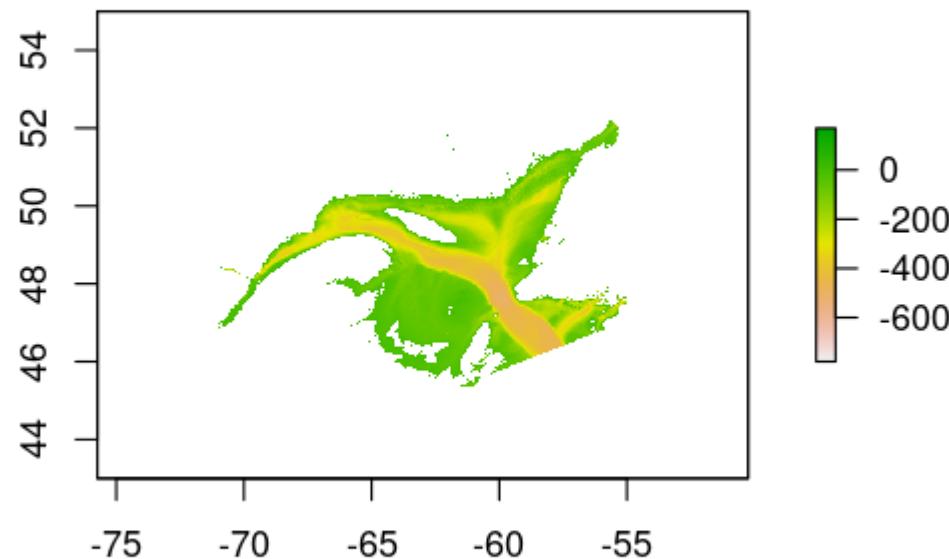
Values outside of a given geometry set to NA

```
plot(sf::st_geometry(stl))
```



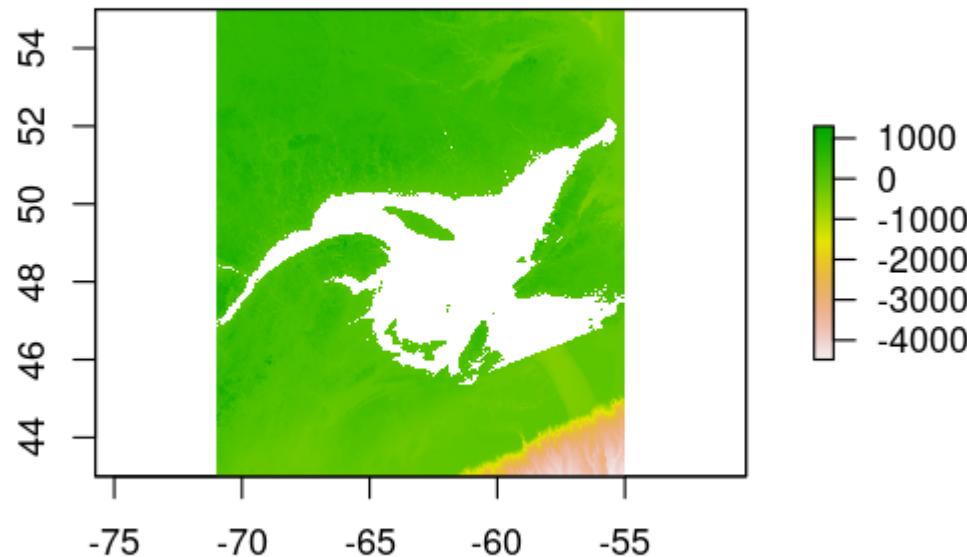
# Mask with raster

```
rar_m <- mask(rar, stl)  
plot(rar_m)
```



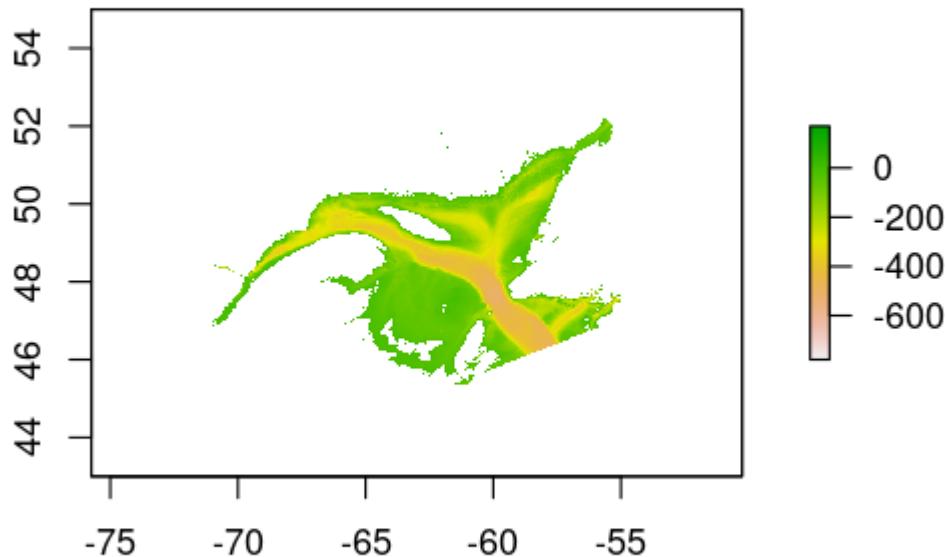
# Mask with raster

```
rar_mi <- mask(rar, stl, inverse = TRUE)  
plot(rar_mi)
```



# Mask with stars

```
ras_m <- ras[stl]  
plot(ras_m)
```



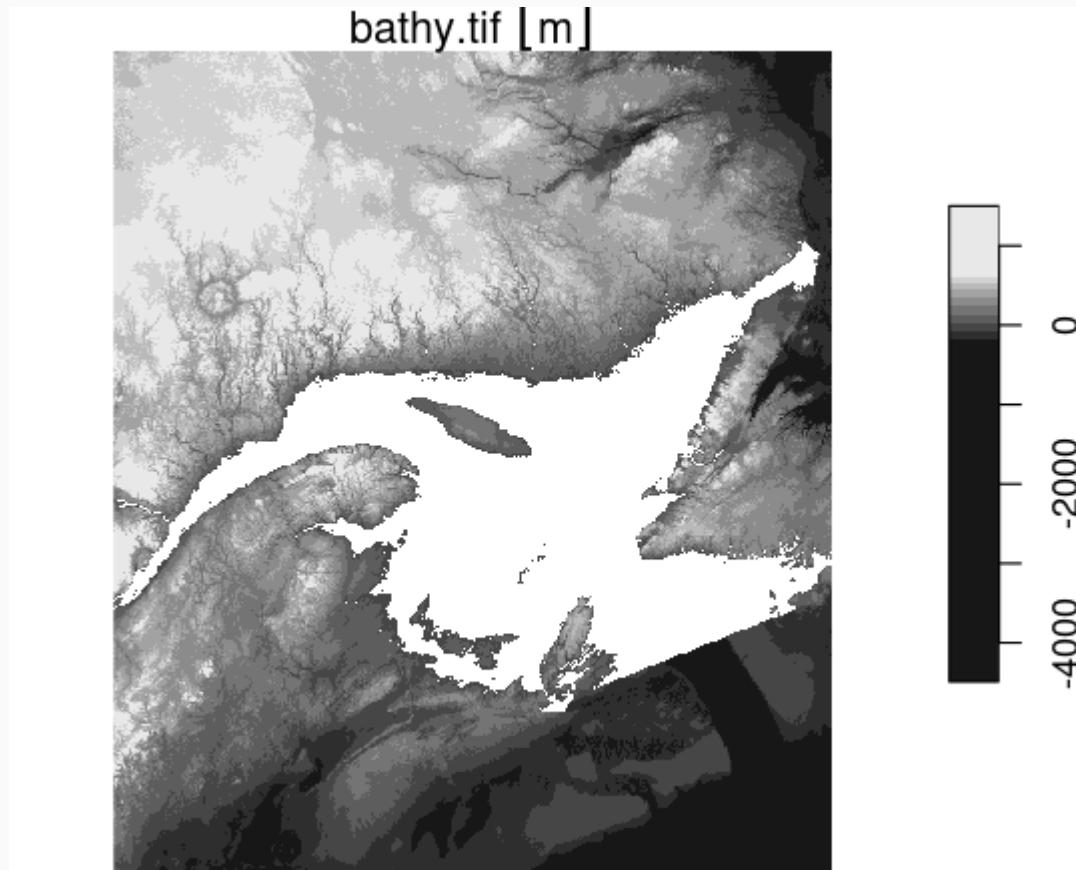
# Mask with stars

```
stl_i <- sf::st_difference(sf::st_as_sfc(st_bbox(ras)), stl)
plot(stl_i, col = 2)
```



# Mask with stars

```
ras_mi <- ras[stl_i]  
plot(ras_mi)
```



# > Mean depth of the St-Lawrence

- ★ Compute the mean depth of the St-Lawrence (Gulf and river)
- ★★ Compare the depth of the Gulf of the St-Lawrence and the depth of the St-Lawrence river (various solutions!)

10:00

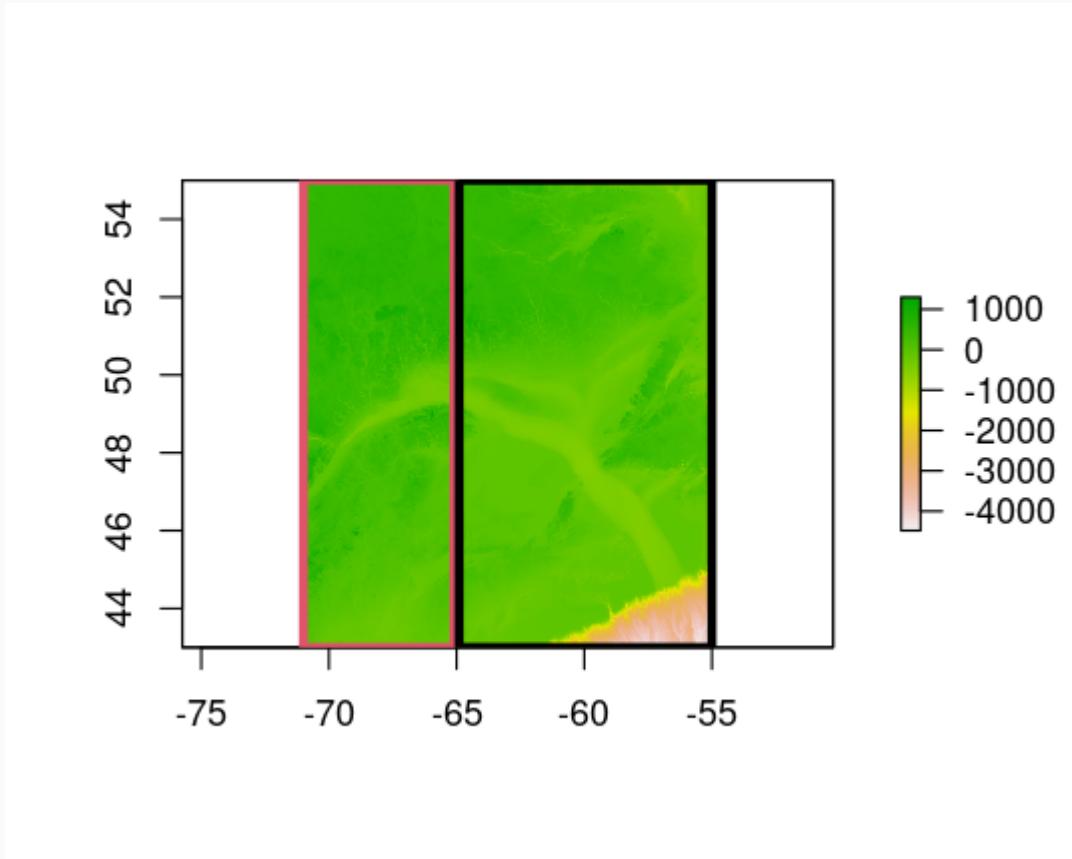
# Solution ☆

```
# raster
rar_m <- mask(rar, stl)
mean(rar_m[[1]], na.rm = TRUE)
#> class      : RasterLayer
#> dimensions : 2880, 3840, 11059200 (nrow, ncol, ncell)
#> resolution : 0.004166667, 0.004166667 (x, y)
#> extent     : -71, -55, 43, 55 (xmin, xmax, ymin, ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : memory
#> names      : layer
#> values     : -785, 282 (min, max)

# stars
ras_m <- ras[stl]
mean(ras_m[[1]], na.rm = TRUE)
#> -162.9211 [m]
```

# Solution ★ ★

```
plot(rar)
rect(-71, 43, -65.1, 55, border = 2, lwd = 4)
rect(-64.9, 43, -55, 55, lwd = 4)
```

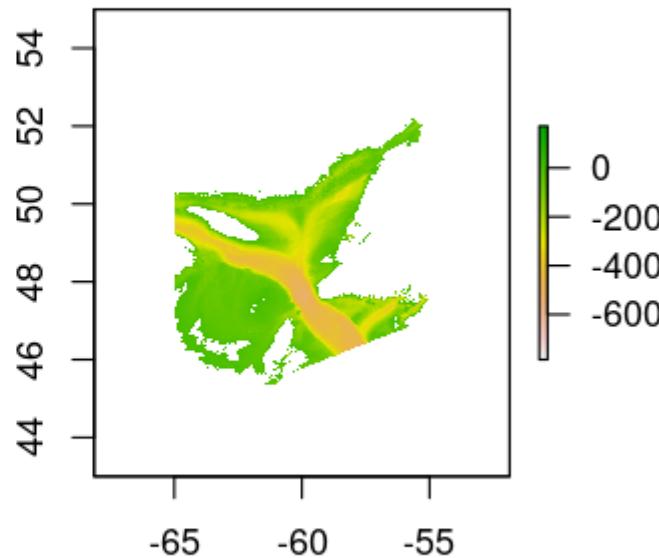
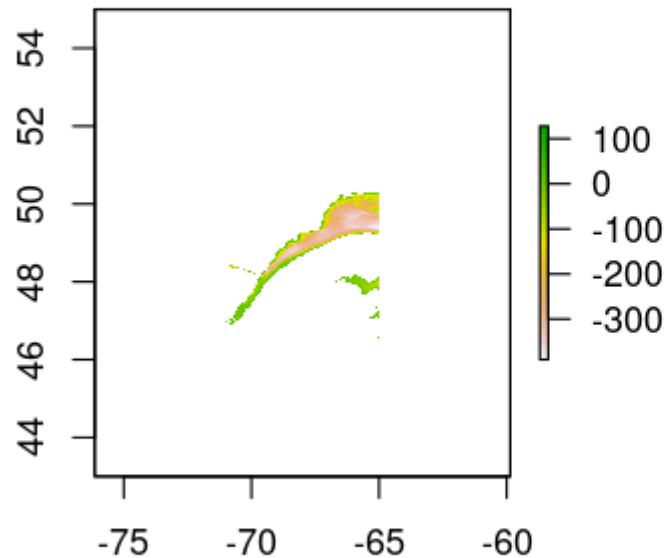


# Solution ★ ★

```
# raster
rar_m <- mask(rar, stl)
rar_river <- crop(rar_m, extent(-71, -65, 43, 55))
rar_gulf <- crop(rar_m, extent(-65, -55, 43, 55))
mean(values(rar_river), na.rm = TRUE)
#> [1] -176.0915
mean(values(rar_gulf), na.rm = TRUE)
#> [1] -161.125
```

# Solution ★ ★

```
plot(rar_river)  
plot(rar_gulf)
```



# Solution ★ ★

```
mean(values(rar_river), na.rm = TRUE)
#> [1] -176.0915
mean(values(rar_gulf), na.rm = TRUE)
#> [1] -161.125
```

# Solution ★ ★

```
# stars
ras_m <- ras[stl]
ras_river <- st_crop(ras_m,
  st_bbox(c(xmin = -71, xmax = -65, ymin = 43, ymax = 55), crs = 4326)
)
ras_gulf <- st_crop(ras_m,
  st_bbox(c(xmin = -65, xmax = -55, ymin = 43, ymax = 55), crs = 4326)
)
mean(ras_river[[1]], na.rm = TRUE)
#> -176.0915 [m]
mean(ras_gulf[[1]], na.rm = TRUE)
#> -161.125 [m]
```

# Warping/Resampling

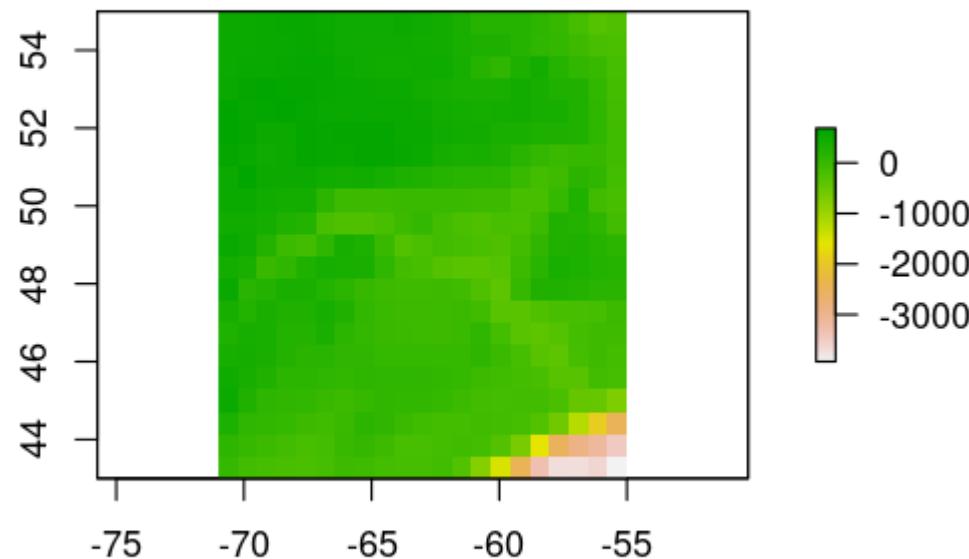
**Work with rasters of different resolutions**

# Warping/Resampling with `raster`

```
# create a raster with the given resolution
mr ← matrix(runif(21*21), 21, 21)
rar_template1 ← raster(mr, xmn = -71, xmx = -55, ymn = 43, ymx = 55,
  crs = projection(rar)) # create template
plot(rar_template1)
```

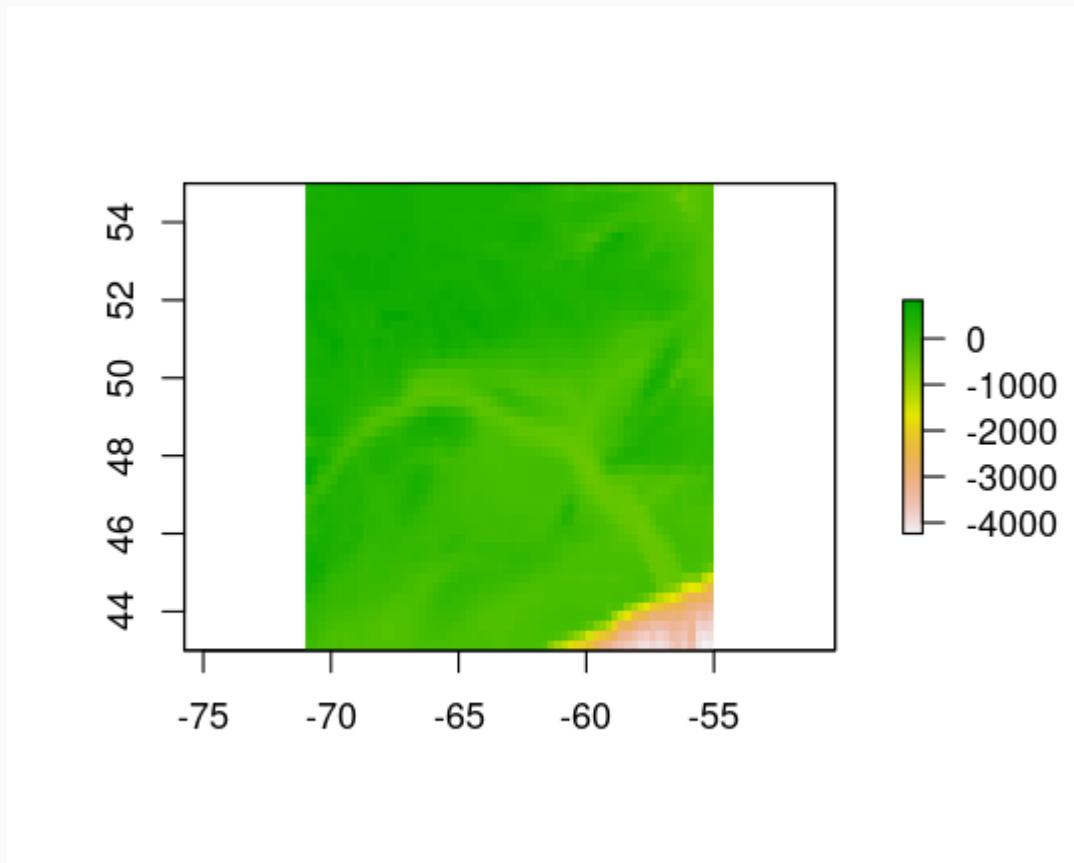
# Warping/Resampling with `raster`

```
rar_w1 <- resample(rar, rar_template1)  
plot(rar_w1)
```



# Warping/Resampling with `raster`

```
rar_template2 <- raster(xmn = -71, xmx = -55, ymn = 43, ymx = 55,  
  crs = projection(rar), resolution = .25) # create template  
plot(resample(rar, rar_template2))
```

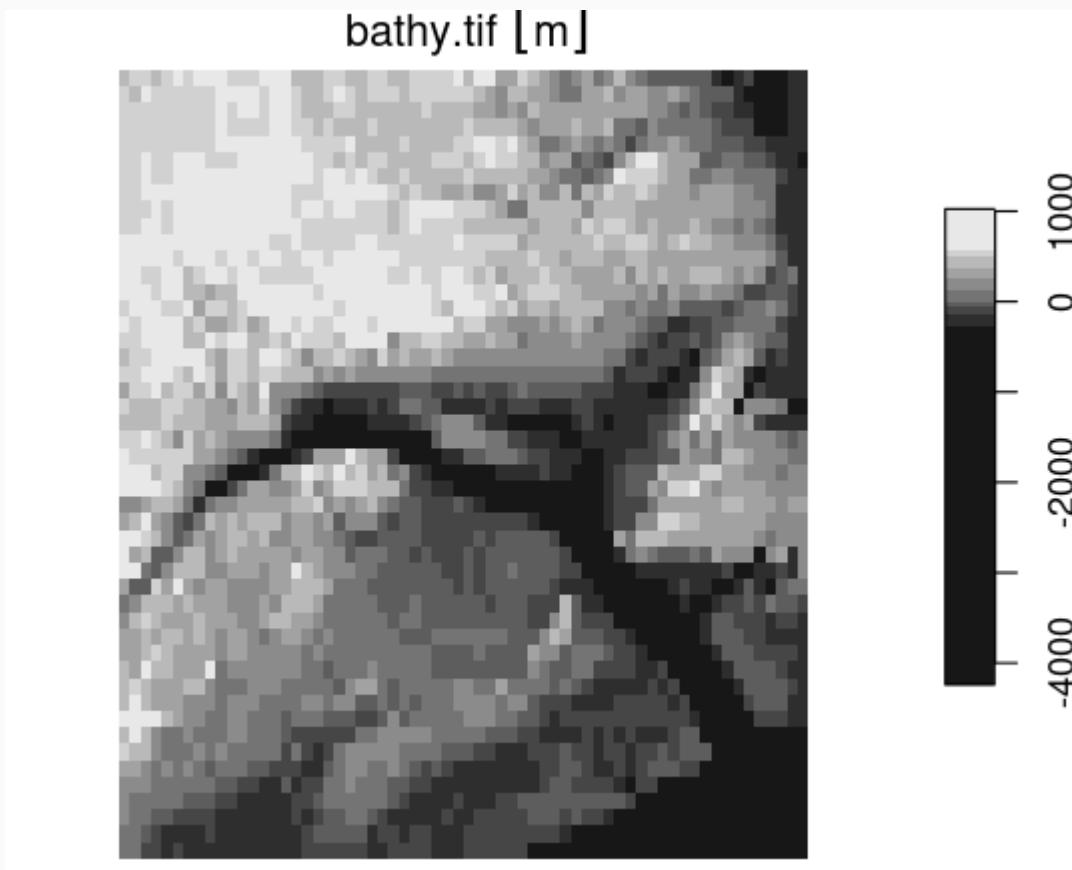


# Warping/Resampling with stars

```
ras_template1 <- st_as_stars(st_bbox(ras), nx = 21, ny = 21,  
    values = runif(21 * 21)) # create template  
ras_w1 <- st_warp(ras, ras_template1)  
plot(ras_w1)
```

# Warping/Resampling with stars

```
ras_w2 <- st_warp(ras, cellsize = 0.25, crs = st_crs(ras))  
plot(ras_w2)
```



# Rasterize

**Vectors → Raster**

**Create a raster template**

# Rasterize with `raster`

```
# not run  
stl_r <- rasterize(stl, rar)  
plot(stl_r)
```

# Rasterize with stars

```
stl_s <- st_rasterize(stl, dy = .1, dx = .1)  
plot(stl_s)
```

**longitude**



# Stack

- Collection of rasters
- Same extent/resolution
- Multi-band rasters (satellite images)
- Handle several rasters at once

# Stack with `raster`

2 classes : `RasterBrick` and `RasterStack`

The main difference is that a `RasterStack` is loose collection of `RasterLayer` objects that can refer to different files (but must all have the same extent and resolution), whereas a `RasterBrick` can only point to a single file.

ⓘ <https://rspatial.org/raster/spatial/4-rasterdata.html#rasterstack-and-rasterbrick>

We'll focus on `RasterStack`

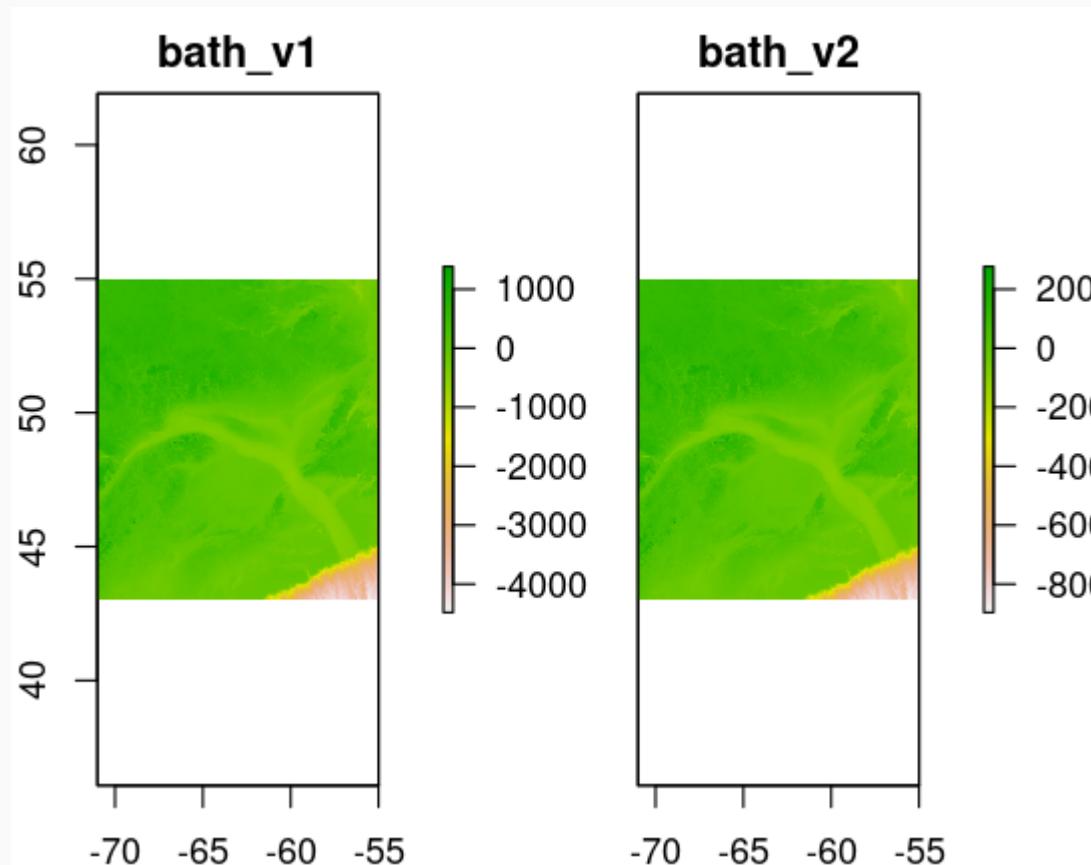
# Stack with raster

```
rar_stc ← stack(list(bath_v1 = rar, bath_v2 = rar * 2)) # could be a file path  
class(rar_stc)  
#> [1] "RasterStack"  
#> attr(, "package")  
#> [1] "raster"  
nlayers(rar_stc)  
#> [1] 2
```

```
class(rar_stc[[1]])  
#> [1] "RasterLayer"  
#> attr(, "package")  
#> [1] "raster"  
class(rar_stc[[2]])  
#> [1] "RasterLayer"  
#> attr(, "package")  
#> [1] "raster"
```

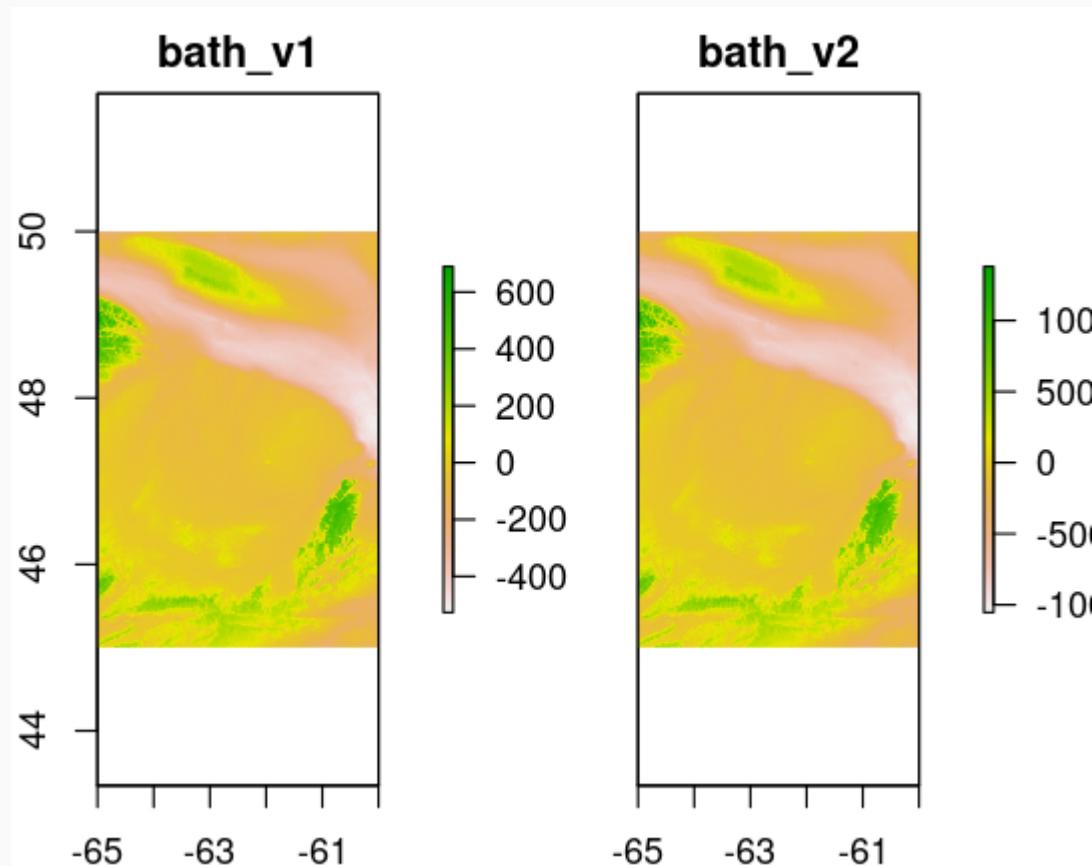
# Stack with raster

```
plot(rar_stc)
```



# Stack with raster

```
plot(crop(rar_stc, extent(-65, -60, 45, 50)))
```

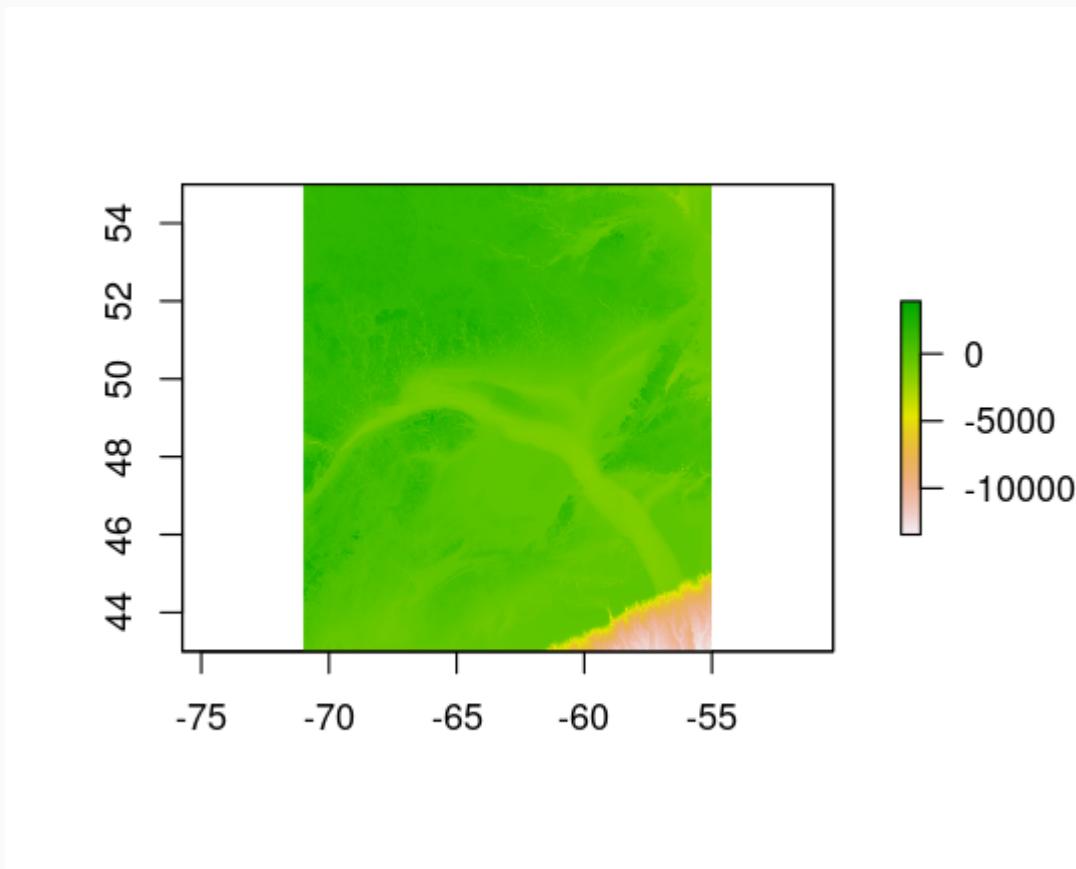


# Stack with raster

```
rar_sum ← calc(rar_stc, fun = sum)
rar_sum
#> class      : RasterLayer
#> dimensions : 2880, 3840, 11059200 (nrow, ncol, ncell)
#> resolution : 0.004166667, 0.004166667 (x, y)
#> extent     : -71, -55, 43, 55 (xmin, xmax, ymin, ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : memory
#> names      : layer
#> values     : -13470, 4506 (min, max)
```

# Stack with raster

```
plot(rar_sum)
```



# Stack with stars

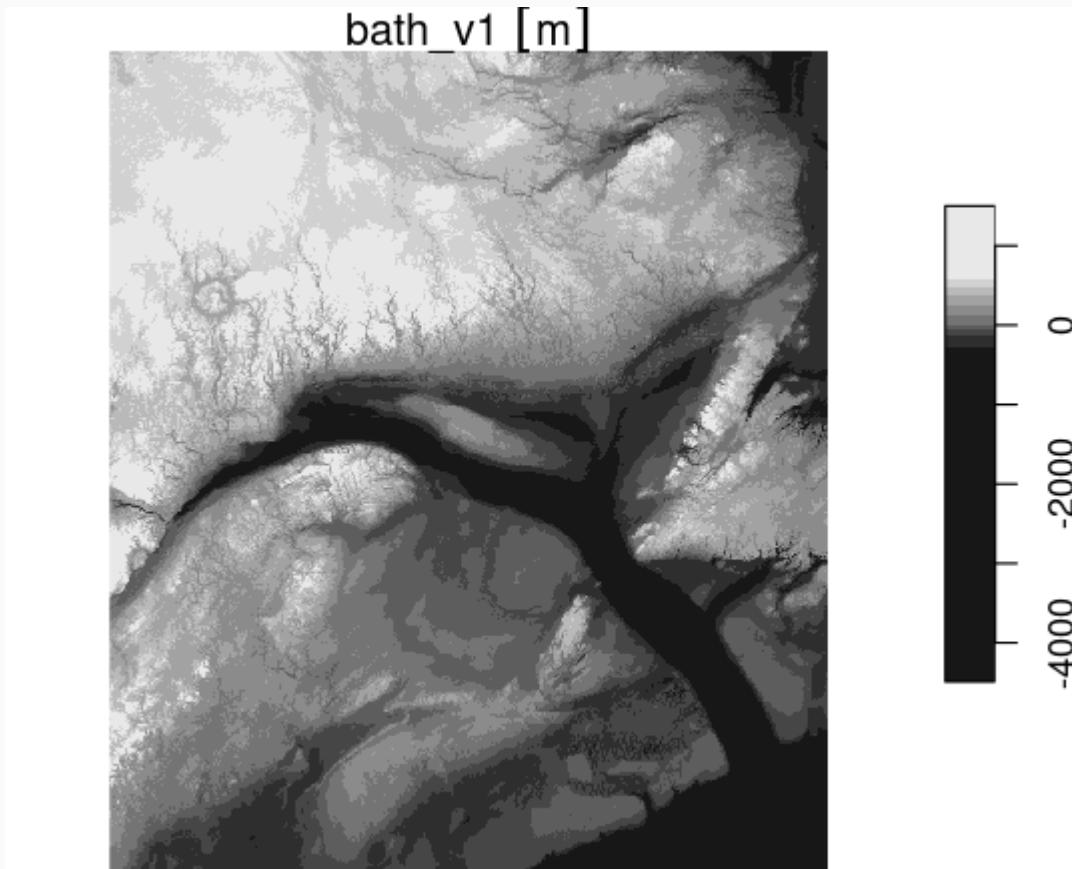
```
ras
#> stars object with 2 dimensions and 1 attribute
#> attribute(s), summary of first 1e+05 cells:
#>   bathy.tif [m]
#>   Min.    :-435.0
#>   1st Qu.: 23.0
#>   Median  : 479.0
#>   Mean    : 299.5
#>   3rd Qu.: 540.0
#>   Max.    : 812.0
#> dimension(s):
#>   from    to offset      delta refsys point values
#> x     1 3840     -71  0.00416667 WGS 84 FALSE   NULL [x]
#> y     1 2880      55 -0.00416667 WGS 84 FALSE   NULL [y]
```

# Stack with stars

```
ras_stc1 <- c(ras, ras * 2)
names(ras_stc1) <- c("bath_v1", "bath_v2")
ras_stc1
#> stars object with 2 dimensions and 2 attributes
#> attribute(s), summary of first 1e+05 cells:
#>   bath_v1 [m]      bath_v2 [m]
#>   Min.    :-435.0   Min.    :-870
#>   1st Qu.: 23.0    1st Qu.: 46
#>   Median : 479.0   Median : 958
#>   Mean    : 299.5   Mean    : 599
#>   3rd Qu.: 540.0   3rd Qu.: 1080
#>   Max.    : 812.0   Max.    : 1624
#> dimension(s):
#>   from    to offset      delta refsys point values
#> x     1 3840     -71  0.00416667 WGS 84 FALSE  NULL [x]
#> y     1 2880      55 -0.00416667 WGS 84 FALSE  NULL [y]
```

# Stack with stars

```
plot(ras_stc1)
```



# Stack with stars

```
ras_stc1[1] # or ras_stc1["bath_v1"]
#> stars object with 2 dimensions and 1 attribute
#> attribute(s), summary of first 1e+05 cells:
#>   bath_v1 [m]
#>   Min.    :-435.0
#>   1st Qu.: 23.0
#>   Median  : 479.0
#>   Mean    : 299.5
#>   3rd Qu.: 540.0
#>   Max.    : 812.0
#> dimension(s):
#>   from    to offset      delta refsys point values
#> x     1 3840     -71  0.00416667 WGS 84 FALSE   NULL [x]
#> y     1 2880      55 -0.00416667 WGS 84 FALSE   NULL [y]
```

# Stack with stars

```
ras_stc1[2] # or ras_stc1["bath_v2"]
#> stars object with 2 dimensions and 1 attribute
#> attribute(s), summary of first 1e+05 cells:
#>   bath_v2 [m]
#>   Min.    :-870
#>   1st Qu.: 46
#>   Median  : 958
#>   Mean    : 599
#>   3rd Qu.:1080
#>   Max.    :1624
#> dimension(s):
#>   from     to offset      delta refsys point values
#> x     1 3840     -71  0.00416667 WGS 84 FALSE   NULL [x]
#> y     1 2880      55 -0.00416667 WGS 84 FALSE   NULL [y]
```

# Stack with stars

```
library(dplyr)
ras_stc1 %>% select("bath_v1")
#> stars object with 2 dimensions and 1 attribute
#> attribute(s), summary of first 1e+05 cells:
#>   bath_v1 [m]
#>   Min.    :-435.0
#>   1st Qu.: 23.0
#>   Median  : 479.0
#>   Mean    : 299.5
#>   3rd Qu.: 540.0
#>   Max.    : 812.0
#> dimension(s):
#>   from    to offset      delta refsys point values
#> x     1 3840     -71  0.00416667 WGS 84 FALSE   NULL [x]
#> y     1 2880      55 -0.00416667 WGS 84 FALSE   NULL [y]
```

# Stack with stars

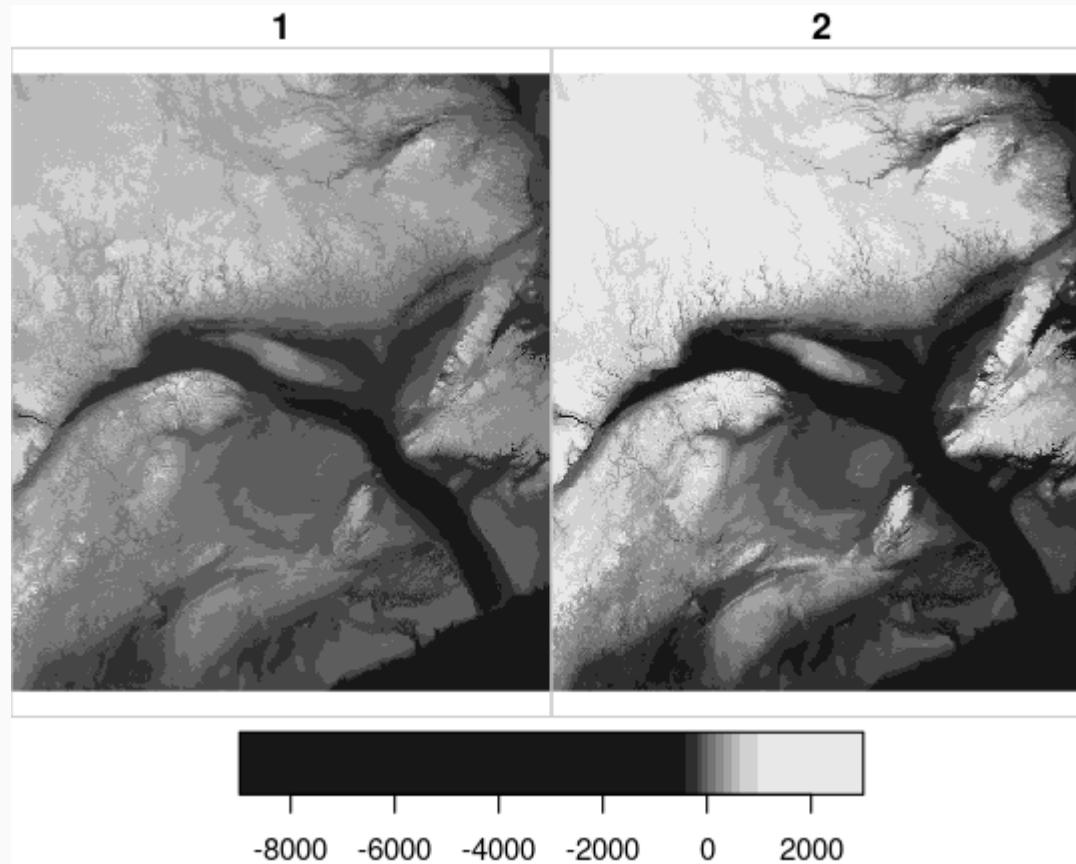
```
ras_stc1 %>% mutate(bath_v3 = bath_v2 * 2)
#> stars object with 2 dimensions and 3 attributes
#> attribute(s), summary of first 1e+05 cells:
#>   bath_v1 [m]      bath_v2 [m]      bath_v3 [m]
#>   Min.    :-435.0    Min.    :-870    Min.    :-1740
#>   1st Qu.: 23.0     1st Qu.: 46     1st Qu.:  92
#>   Median  : 479.0    Median  : 958    Median  : 1916
#>   Mean    : 299.5    Mean    : 599    Mean    : 1198
#>   3rd Qu.: 540.0    3rd Qu.:1080   3rd Qu.: 2160
#>   Max.    : 812.0    Max.    :1624    Max.    : 3248
#> dimension(s):
#>   from    to offset      delta refsys point values
#> x     1 3840     -71  0.00416667 WGS 84 FALSE    NULL [x]
#> y     1 2880      55 -0.00416667 WGS 84 FALSE    NULL [y]
```

# Stack with stars

```
ras_stc2 ← c(ras, ras*2, along = "z")
ras_stc2
#> stars object with 3 dimensions and 1 attribute
#> attribute(s), summary of first 1e+05 cells:
#>   bathy.tif [m]
#>   Min.    :-435.0
#>   1st Qu.: 23.0
#>   Median  : 479.0
#>   Mean    : 299.5
#>   3rd Qu.: 540.0
#>   Max.    : 812.0
#> dimension(s):
#>   from     to offset      delta refsys point values
#> x     1 3840      -71  0.00416667 WGS 84 FALSE    NULL [x]
#> y     1 2880       55 -0.00416667 WGS 84 FALSE    NULL [y]
#> z     1     2        NA        NA      NA      NA    NULL
```

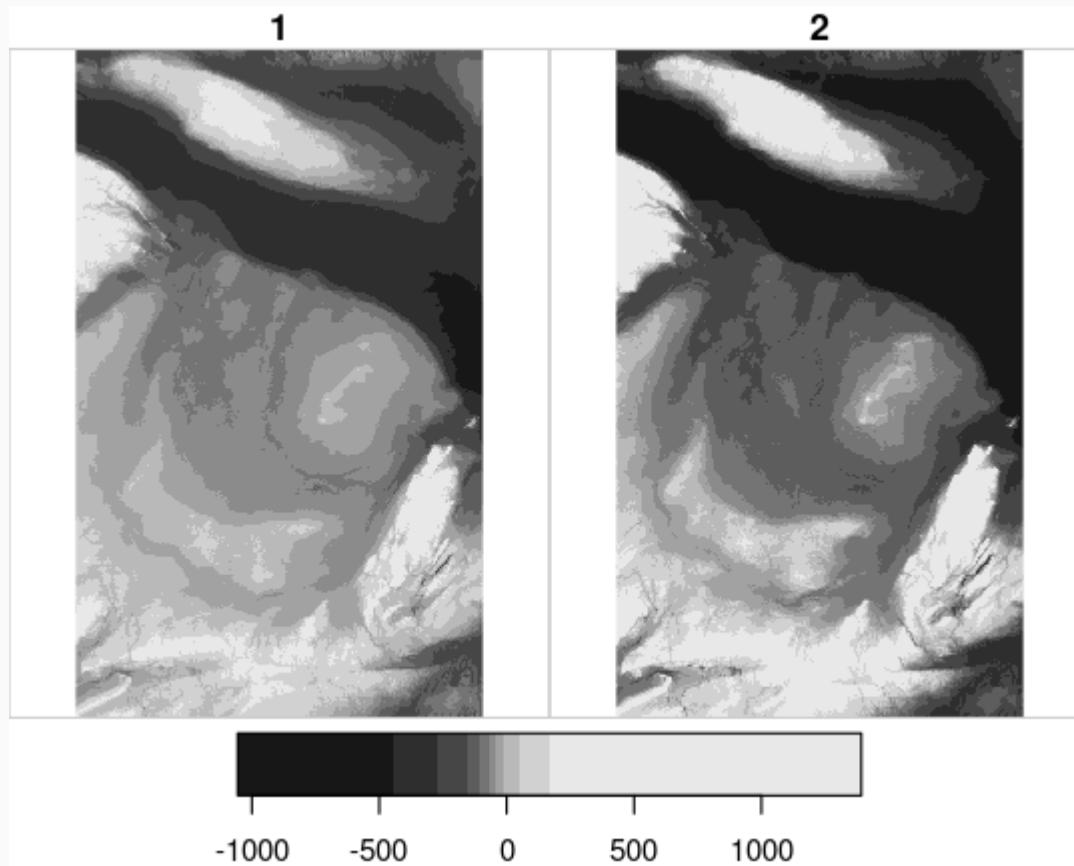
# Stack with stars

```
plot(ras_stc2)
```



# Stack with stars

```
ext <- st_bbox(c(xmin = -65, xmax = -60, ymin = 45, ymax = 50), crs = 4326)
plot(st_crop(ras_stc2, ext))
```

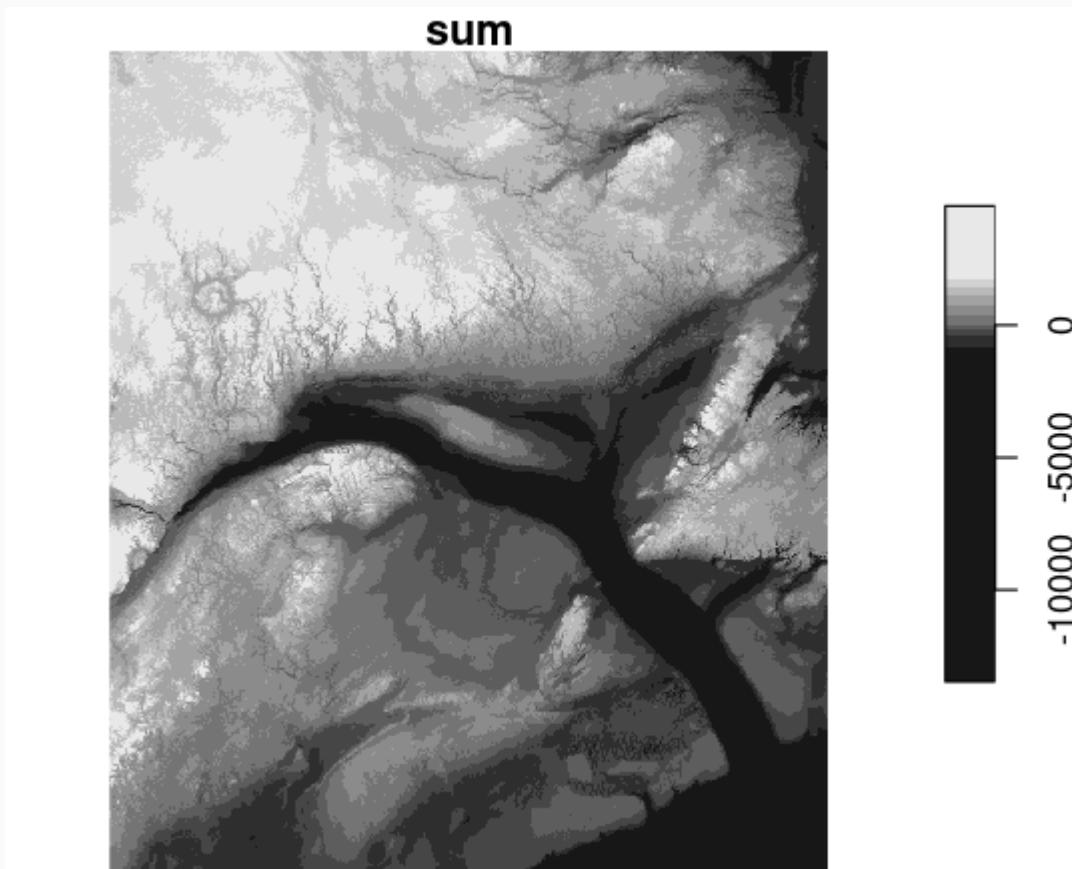


# Stack with stars

```
(ras_ap ← st_apply(ras_stc2, c(1, 2), sum))
#> stars object with 2 dimensions and 1 attribute
#> attribute(s), summary of first 1e+05 cells:
#>     sum
#> Min.   :-1305.0
#> 1st Qu.:  69.0
#> Median  : 1437.0
#> Mean    : 898.6
#> 3rd Qu.: 1620.0
#> Max.   : 2436.0
#> dimension(s):
#>   from    to offset      delta refsys point values
#> x     1 3840     -71  0.00416667 WGS 84 FALSE   NULL [x]
#> y     1 2880      55 -0.00416667 WGS 84 FALSE   NULL [y]
```

# Stack with stars

```
plot(ras_ap)
```



# 国旗 Recap

Action	raster	stars
Get projection	projection()	st_crs()
Get bounding box	bbox()	st_bbox()
Change projection	projectRaster()	st_transform()
Crop	crop()	st_crop()
Mask	mask()	[ ]←
Resample	projectRaster()	st_warp()
Rasterize	rasterize()	st_rasterize()
Stack	stack() or brick()	c(along= ... )
Apply functions on stacks	calc()	st_apply()

ℹ <https://r-spatial.github.io/stars/articles/stars6.html>



# Visualize

---

Static visualization (~45min)

# R packages for raster visualization

## Static

- `tmap` ✓
- `ggplot2`
  - ⓘ [https://r-spatial.github.io/stars/reference/geom\\_stars.html](https://r-spatial.github.io/stars/reference/geom_stars.html)
  - ⓘ [https://ggplot2.tidyverse.org/reference/geom\\_tile.html](https://ggplot2.tidyverse.org/reference/geom_tile.html)
- `ggmap`
  - ⓘ <https://github.com/dkahle/ggmap>
- `rasterVis`
  - ⓘ <https://oscarperpinan.github.io/rastervis/>

## Interactive

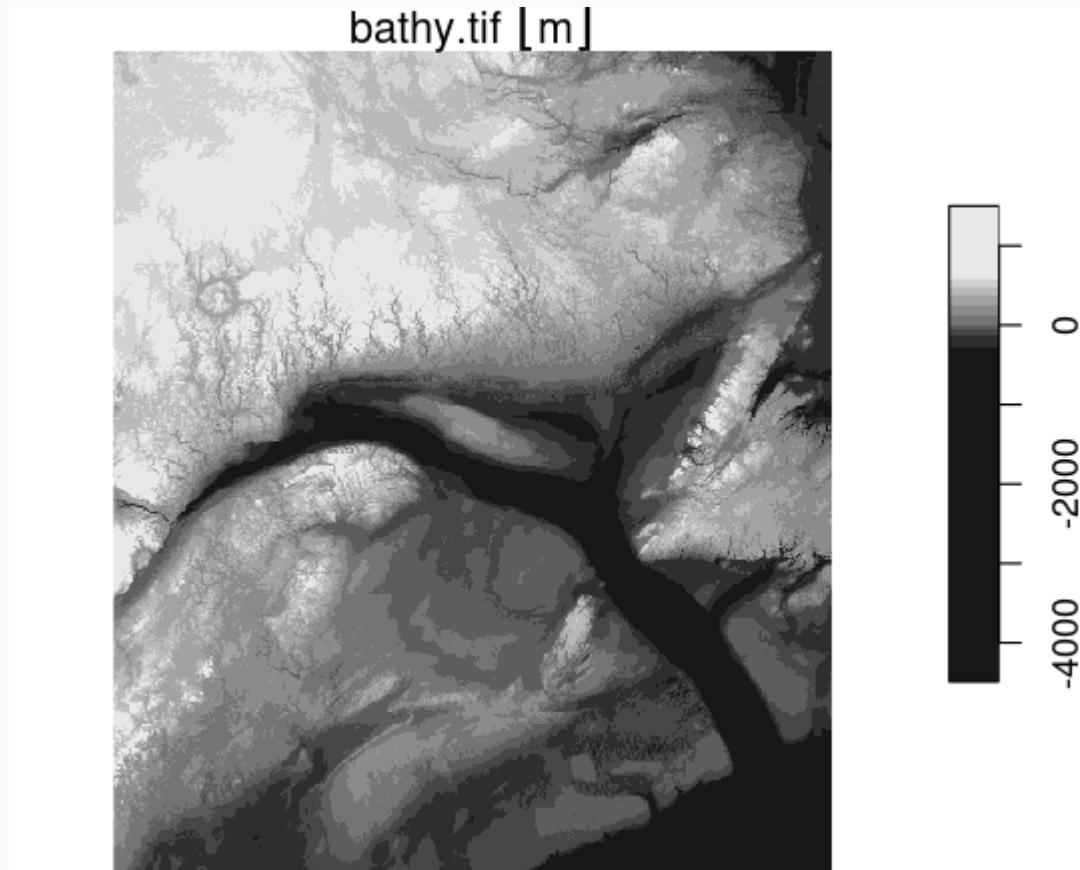
- `tmap`
- `leaflet`
- `mapview`

# Using `plot()` and `image()`

- `plot()`
  - → `?plot.stars`
  - → `?raster::plot`
- `image()`
  - → `?image.stars`
  - → `?raster::image`
- `par()` : margins, background color, ....

# Using `plot()` and `image()`

```
plot(ras)
```

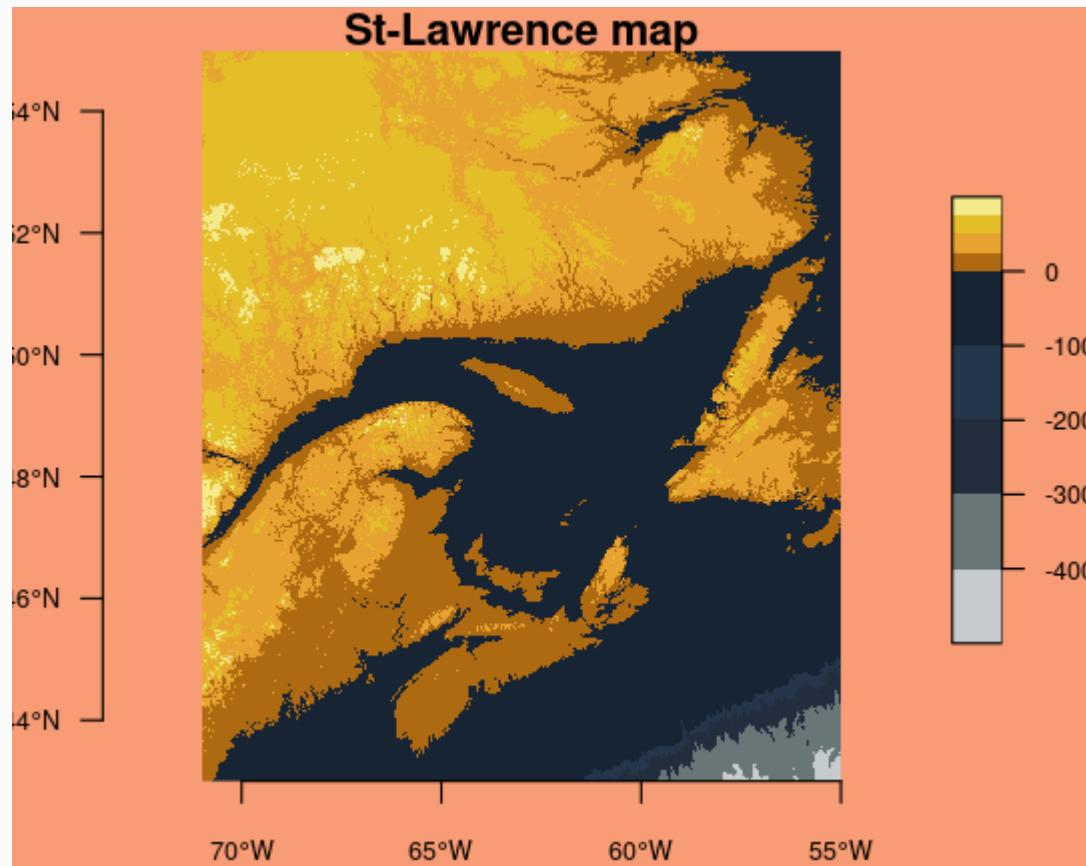


# Using `plot()` and `image()`

```
# breaks  
bks ← c(seq(-5000, 0, 1000), 250, 500, 750, 1000)  
# cols  
cls ← c("#c7cbce", "#687677", "#222d3d", "#25364a", "#172434",  
"#ad6a11", "#e6a331", "#e4be29", "#f2ea8b")
```

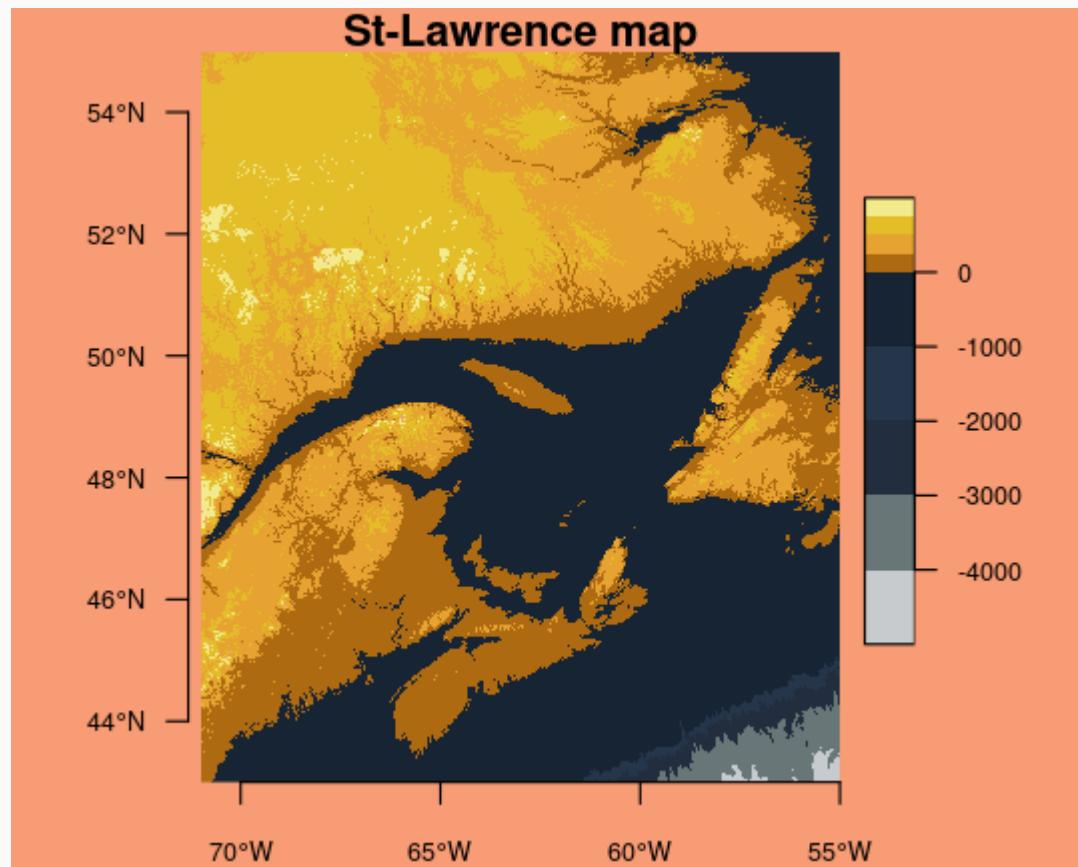
# Using `plot()` and `image()`

```
par(las = 1, bg = "#f79c74", cex.axis = .7, mar = c(2, 2, 2, 4))
plot(ras, breaks = bks, col = cls, main = "St-Lawrence map", axes = TRUE)
```



# Using `plot()` and `image()`

```
par(las = 1, bg = "#f79c74", cex.axis = .7, mar = c(2, 2, 2, 4), oma = c(0, 2, 0, 2))
plot(ras, breaks = bks, col = cls, main = "St-Lawrence map", axes = TRUE)
```



# mapedit

README.md

# mapedit

CRAN 0.6.0 downloads 2369/month downloads 90K

Interactive editing of spatial data in R | an [RConsortium](#) funded [project](#). For additional detail, please see the original [proposal](#).

## 🔗 Status

`mapedit` is still in active development. We would very much appreciate feedback, ideas, and use cases. The API has stabilized, and we will use semantic versioning with Github tagged releases to track changes and progress. All changes will also be documented in `NEWS.md`.

ℹ️ <https://github.com/r-spatial/mapedit>

# mapedit *live*

```
install.packages("mapedit")
library(mapedit)
anticosti <- mapedit()
```

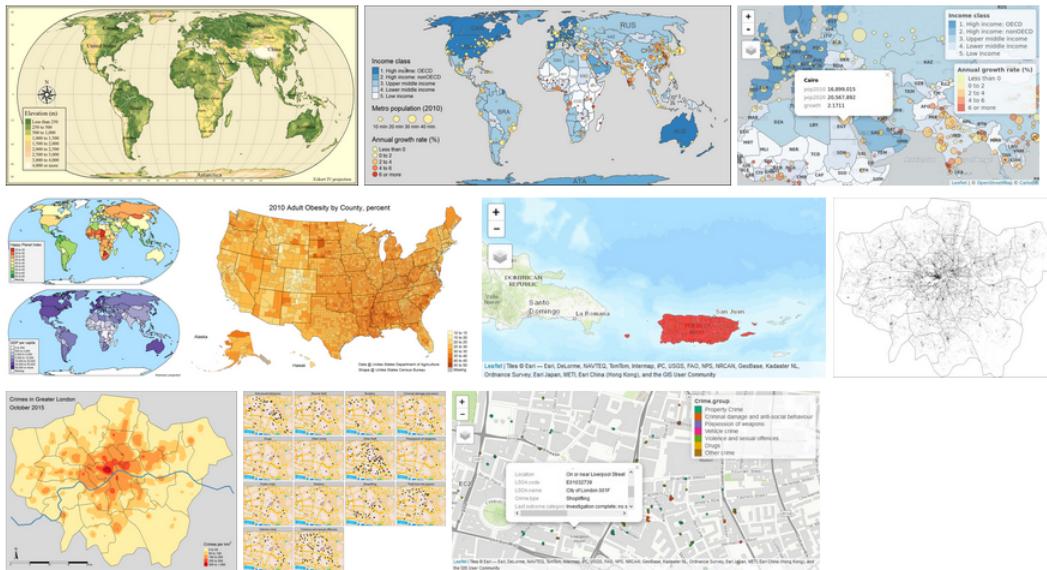
```
stl ← sf::st_read("data/st_laurence.geojson")
png("output/anticosti.png", width = 7, height = 5, units = "in", res = 300)
plot(st_geometry(stl))
plot(st_geometry(anticosti), col = 2, add = TRUE)
dev.off()
```

README.md

## tmap: thematic maps in R

 build failing License [GPL v3](#) CRAN 3.2 CRAN [NOTE](#) downloads 15K/month

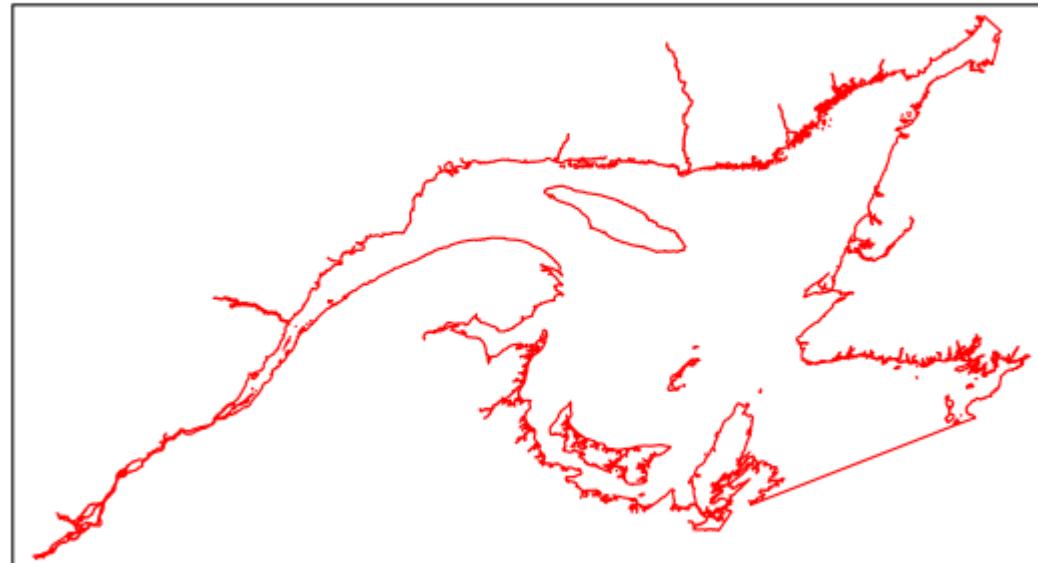
`tmap` is an actively maintained open-source R-library for drawing thematic maps. The API is based on [A Layered Grammar of Graphics](#) and resembles the syntax of `ggplot2`, a popular R-library for drawing charts.



See [below](#) the source code for these images.

 <https://github.com/mtennekes/tmap>

```
library(tmap)
map0 <- tm_shape(stl) + tm_borders(col = "red")
map0
```



```
map1 <- map0 + tm_compass(type = "8star", position = c("left", "top"))
map1
```

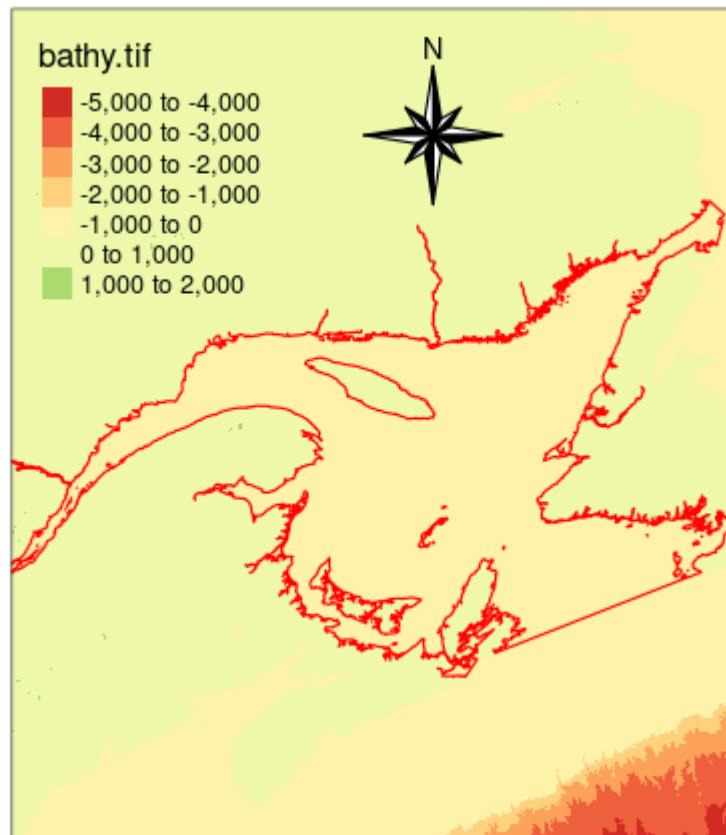


- works with `raster` and `stars` (the latter is recommended)
- let's focus on `stars`

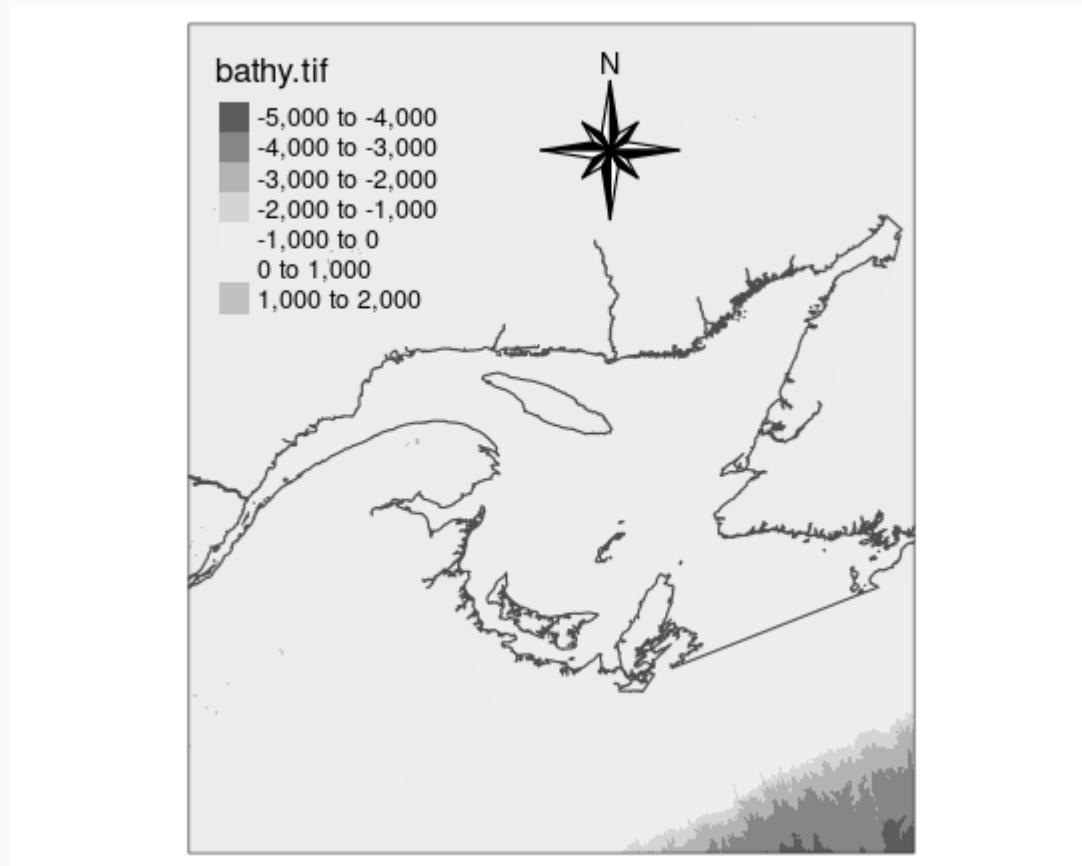
```
# create a stars object from a raster object  
st_as_stars(rar)
```

```
names(ras)
#> [1] "bathy.tif"
map2 <- tm_shape(ras) + tm_raster("bathy.tif")
```

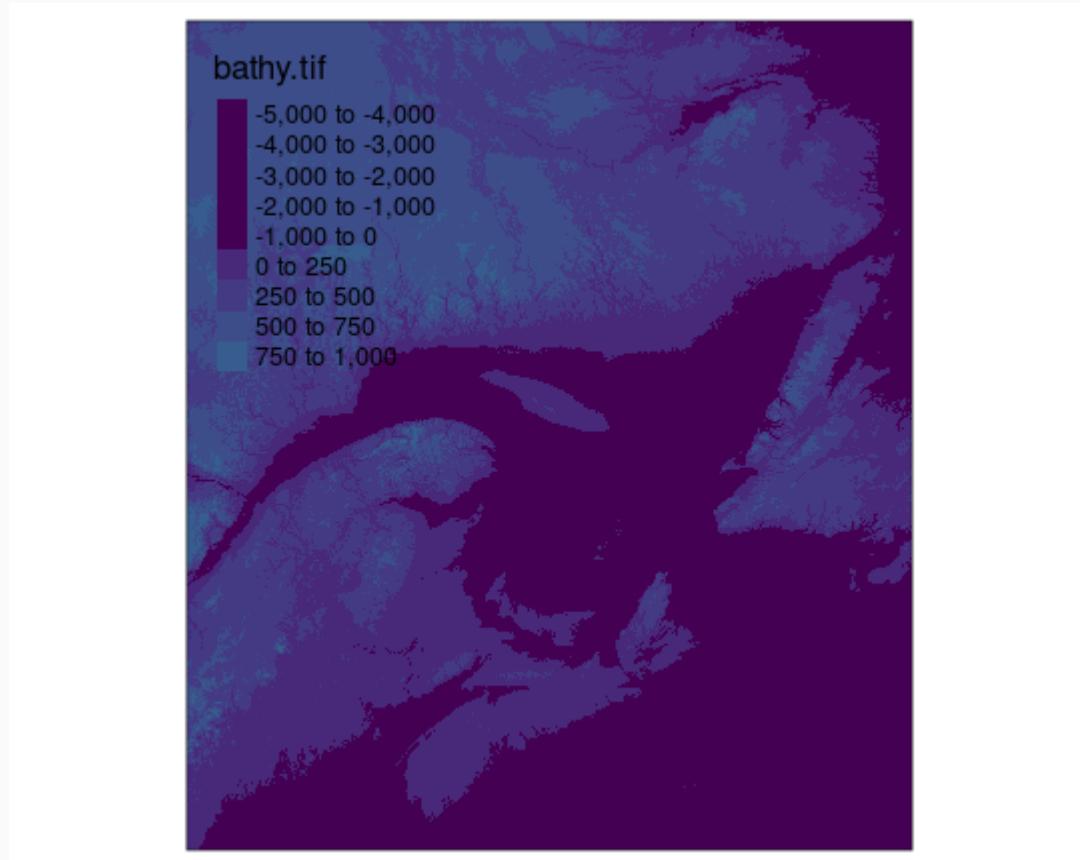
```
map3 <- map2 + map1 # the order matters  
map3
```



```
map4 <- map2 + map1 + tm_style("bw")  
map4
```



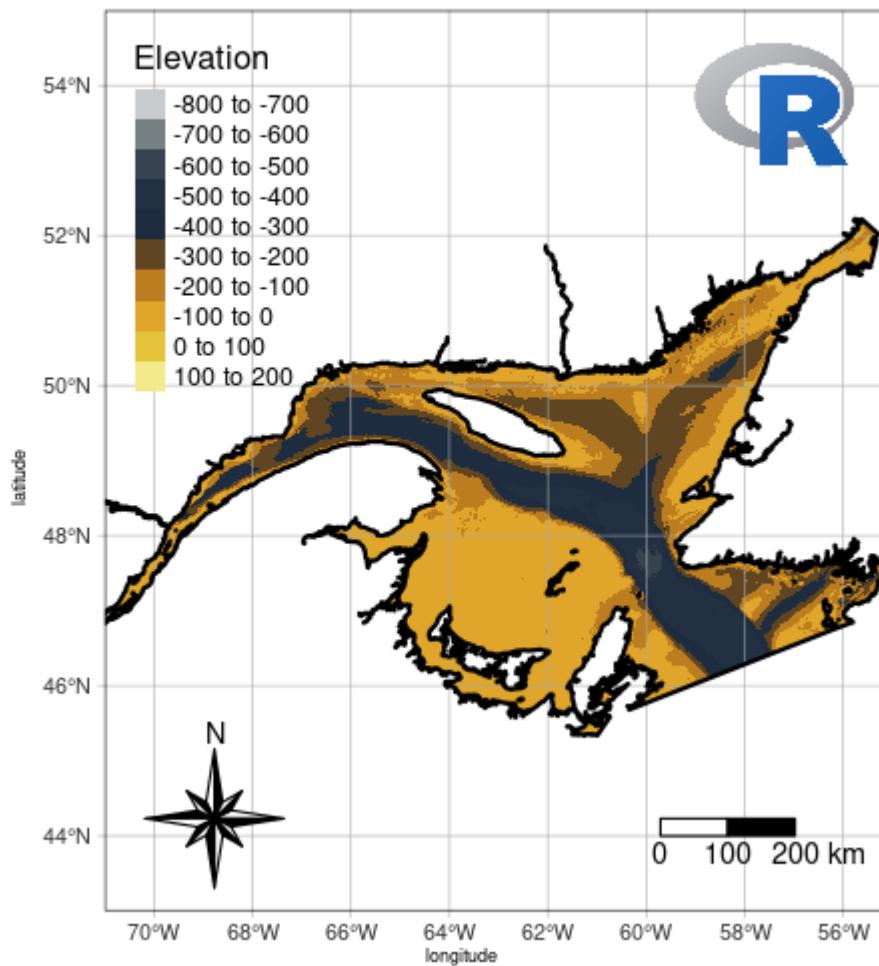
```
map5 <- tm_shape(ras) + tm_raster("bathy.tif", breaks = c(seq(-5000,  
0, 1000), 250, 500, 750, 1000), palette = "viridis")  
map5
```



- ⓘ <https://geocompr.robinlovelace.net/adv-map.html>
- ⓘ <https://github.com/mtennekes/tmap>
- ⓘ [https://www.rdocumentation.org/packages/tmap/versions/3.2/topics/tm\\_layout](https://www.rdocumentation.org/packages/tmap/versions/3.2/topics/tm_layout)

# ★ ★ Reproduce this map

St-Lawrence river & Gulf



15:00

# Solution

# Solution

# Resources

- Geocomputation with R
- Spatial Data Science
- Blog R-spatial
- Open Geospatial Consortium
- [https://nowosad.github.io/BioGIS\\_19/workshop/#64](https://nowosad.github.io/BioGIS_19/workshop/#64)