Introduction to rasters in R terra

Anna Stroh

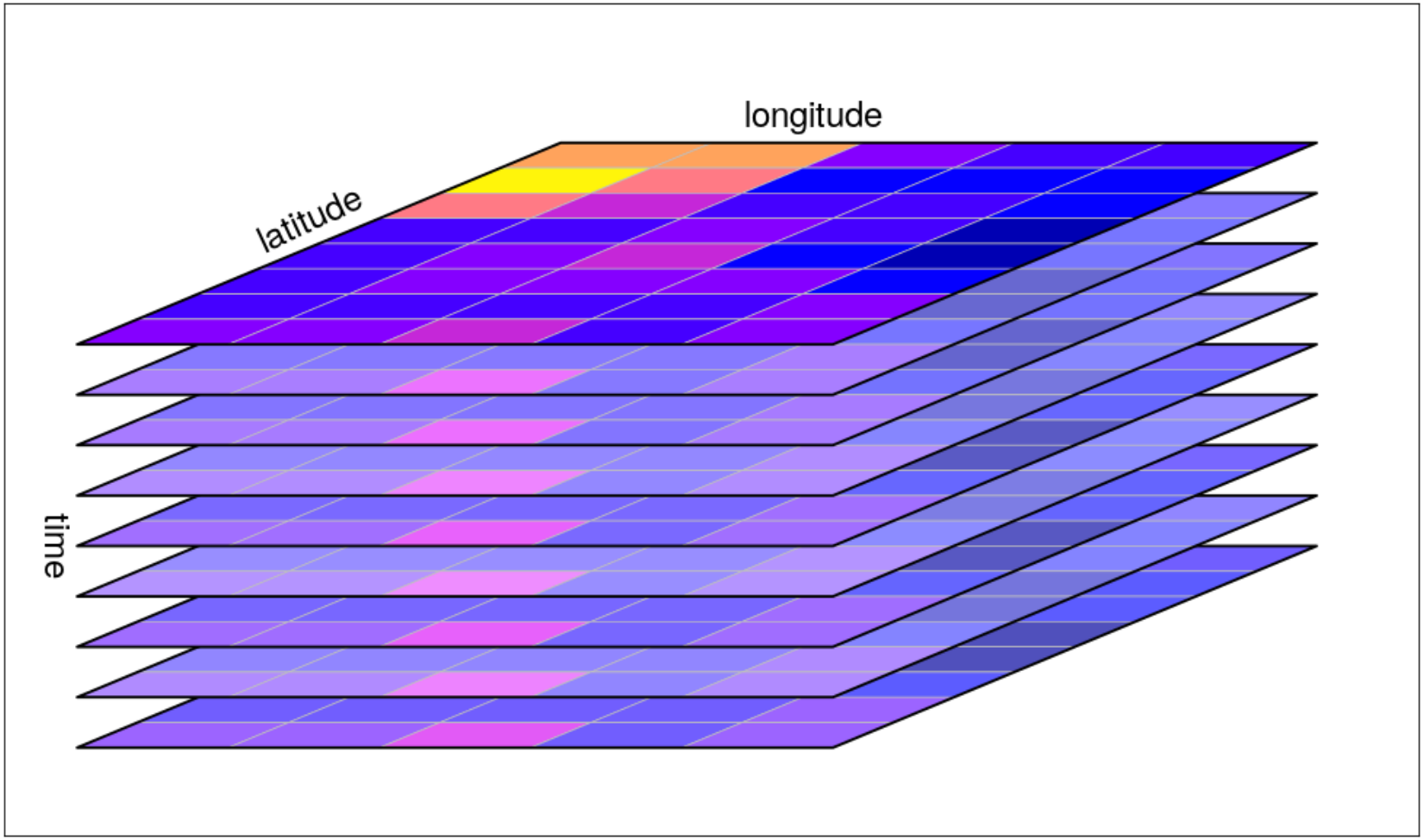
MFRC, Atlantic Technological University

R terra tutorial
November 6, 2024

Contents

What are rasters?

- Typically used to store spatially continuous phenomena
- If these phenomena vary over a period of time (i.e., day, month, year), data is stored within spatiotemporal rasters
- Raster data is aligned on a grid of equally sized rectangles or squares ("cells")
- Cells are aligned on the x and y axes (Easting and Northing, respectively)



Raster data cube

Reference: https://r-spatial.github.io/stars/

Extracting grid data from online servers

Spatiotemporal grid data can be accessed within R using the `rerddap` R package. ERDDAP servers archive environmental, meteorological, oceanographic and biological data. More information on the `rerddap` package see `https://docs.ropensci.org/rerddap/articles/Using_rerddap.html`.

See a full demonstration of this in SpatiotemporalRasterProcessing.Rmd, based on code created by Dr Cóilín Minto.

The terra package

- Provides methods for spatial data analysis with raster (grid) and vector (points, lines, polygons) data
- Unites many relevant functions in a singular package
- Compatible with other popular R packages for spatial data analysis (i.e. `sf`)
- Supports large raster files through optimised raster file storage
  - `terra` stores a raster on the disc, and only loads features once needed

Comparison terra vs stars

When working with (spatiotemporal) rasters in R, you will find two popular packages: `terra` and `stars`. Both are excellent packages and have their advantages. A full comparison can be found here and here.

Personally, I find `terra` more efficient, easier to understand if you are a beginner, and sufficient for common spatiotemporal raster processing.

Reading rasters

The function `rast()` reads various raster file types using a GDAL driver

```
## Load example
library(terra)
f <- system.file("ex/elev.tif", package="terra")
example <- rast(f)
example
```

See R code: TerraBasics.R

```
## terra 1.7.71

##
## Attaching package: 'terra'

## The following object is masked from
'package:knitr':
##
##     spin

## class       : SpatRaster
## dimensions  : 90, 95, 1  (nrow, ncol, nlyr)
## resolution  : 0.008333333, 0.008333333  (x, y)
## extent      : 5.741667, 6.533333, 49.44167, 50.19167  (x
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## source      : elev.tif
## name        : elevation
## min value   :       141
## max value   :       547
```

In `terra`, any file being read with `rast()` creates a `SpatRaster` object, no matter whether the raster has single or multiple layers

Key functions to get and set SpatRaster elements:

```
dim() # number of rows, cols, and layers
ncell() # number of cells = nrow*ncol
ext() # spatial extent
res() # spatial resolution
crs() # coordinate reference system (crs)
names() # layer names
varnames() # names of variables stored in raster
time() # time band
```

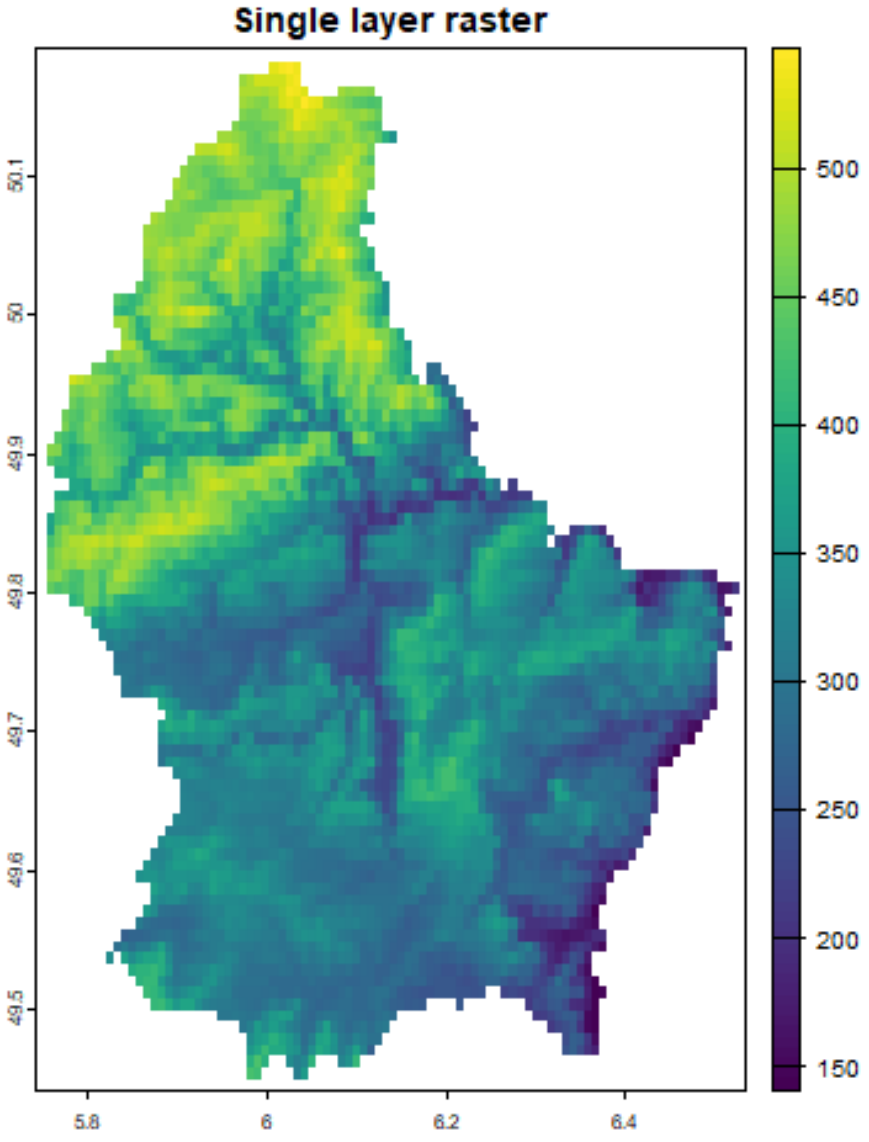
Summarising rasters

These functions allow to get general summaries of the values stored in a raster object in terra:

```
## Careful with very large rasters
global() # summary statistics for a set of layers
zonal() # summary statistics for regions
```

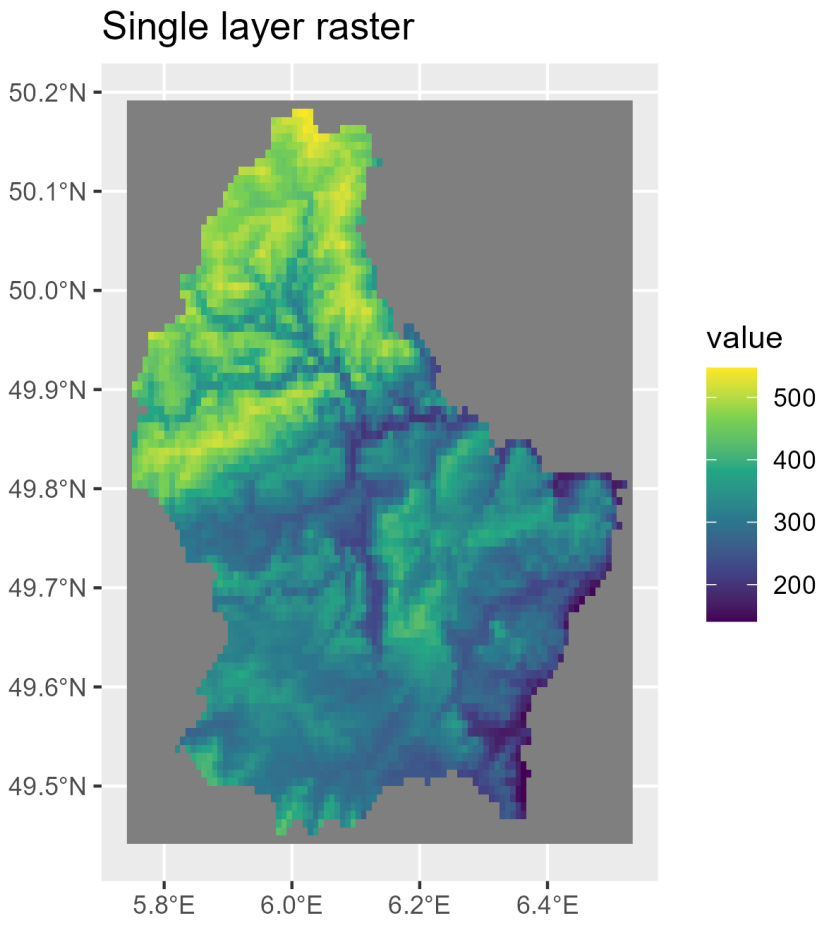
Plotting in terra

```
plot(example, main = "Single layer raster")
```



Plotting in ggplot2

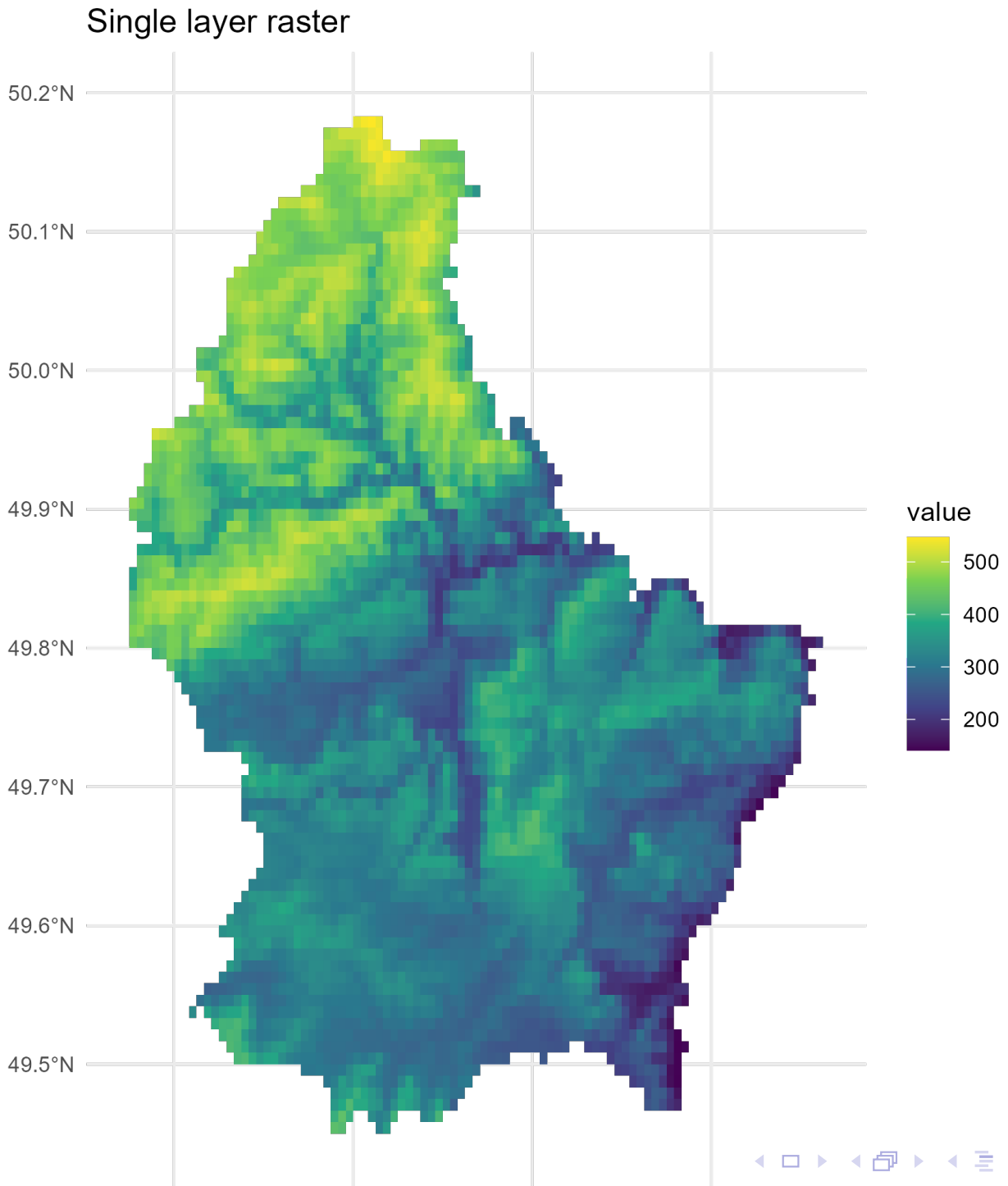
```
library(ggplot2)
library(tidyterra) # package extension
library(viridis)

ggplot() +
    geom_spatraster(data = example, # plot raster
                    aes(fill = elevation)) + # layer name
    scale_fill_viridis_c() +
    ggtitle("Single layer raster")
```


Single layer raster
50.2°N
50.1°N
50.0°N
49.9°N
49.8°N
49.7°N
49.6°N
49.5°N
5.8°E
6.0°E
6.2°E
6.4°E
value
500
400
300
200

Mapping SpatRasters in ggplot2 requires additional layout formatting

```
ggplot() +
    geom_spatraster(data = example,
                    aes(fill = elevation)) +
    # removes background
    scale_fill_viridis_c(na.value = "transparent") +
    coord_sf(crs = 4326) +
    labs(title = "Single layer raster",
         fill = "elevation") +
    theme_minimal()
```

See R code: TerraBasics.R


Single layer raster
50.2°N
50.1°N
50.0°N
49.9°N
49.8°N
49.7°N
49.6°N
49.5°N
value
500
400
300
200

Mapping SpatRasters with multiple layers in ggplot2 similar to faceting. Use "lyr" argument to plot rasters with multiple layers.

```
ggplot() +
geom_spatraster(data = mpa_rast[[1:4]],
                maxcell = ncell(mpa_rast[[1:4]])) +
    scale_fill_viridis_c(
    labels = scales::label_number(suffix = "º"),
                     na.value = "transparent") +
    facet_wrap(~ lyr, # "~lyr" = layers
               ncol = 2, # customise layout
               nrow = 2)
```

See R vignette: SpatiotemporalRasterProcessing.Rmd

Manipulation - Subset

Raster layers can be subset in two ways:

```
## Subset using [[ layer index/indices ]]
red <- multilayer[[1]] # subsets to first layer
red_green <- multilayer[[1:2]] # or multiple layers

## Subset using subset()
red2 <- subset(multilayer, 1) # also layer index
```

The same methods can be applied to the time band, and rasters with multiple variables. See SpatiotemporalRasterProcessing.Rmd for more applications (incl. time band).

Manipulation - Clamping

Raster values are subset through clamping:

```
## Subset raster values using clamp()
start <- 0 # assume interest in lower elevations
end <- 200
low_elev <- clamp(example, start, end, values = TRUE)
```

See R code: TerraBasics.R

Manipulation - Cropping

Assume you need to alter the geographic range of a given raster. `terra` provides several ways to achieve that:

```
crop() # geographic subset
drawExtent() # visually determine mapping extent
trim() # remove outer rows/columns that contain NA values
```

See example for `crop()` in SpatiotemporalRasterProcessing.Rmd.

Manipulation - Aggregation

Spatiotemporal rasters may come in different temporal and spatial resolutions. Time periods or raster cells are aggregated, thereby lowering the temporal/spatial resolution.

`tapp()` aggregates raster values over chosen time interval, creating a new raster with lower temporal resolution.

```
# Aggregate time periods
tapp()
```

`aggregate()` aggregates groups of cells to create larger cells, creating a new raster with lower spatial resolution.

```
# Aggregate spatial resolution
aggregate()
```

See example for `tapp()` in SpatiotemporalRasterProcessing.Rmd.

Manipulation - Rasterisation

Increasing the spatial resolution can be useful particularly during a rasterisation process, where a vector is converted into a raster.

```
## This is generalised code, and not yet included in GitHu
v <- vect(polygon) # read vector data in terra
v <- project(v, "EPSG:4326") # reproject vec crs to WGS84(
r <- rast(v) # create a raster "frame" for rasterisation
r <- project(r, crs(v),
             # proportionally increase resolution!
             res = 1/200)
r_rast <- rasterise(v, r)
```

Data extraction

Raster data can be directly linked to biological observation by extraction for their spatial location. To improve the efficiency and speed of data extraction, it is recommended to inspect and prepare raster data prior to extracting data for a given set of locations. `extract()` can be used to extract for points, lines, and polygons. A full example on how to extract time-specific temperature data for point locations is given in SpatiotemporalRasterProcessing.Rmd.

Alternative applications of `extract()` include:

```
extractAlong() # orders raster data along set of line vecto
extractRange() # extract for specific set of layers
```

Any questions?