The R-spatial package ecosystem and openEO for analysing Earth Observation data

E. Pebesma, M. Mohr, F. Lahn, P. Zellner, M. Rossi, A. Jacob, P. Griffiths

Why do we use Data Science languages, and Open Source?

The aim of science is communication:

- DS languages summarise and abstract problems, and allow reproducing the computational steps of a study
- open source lets anyone scrutinize computation, it allows to
  - learn,
  - understand,
  - critisize and
  - extend

`https://github.com/edzer/LPS22`

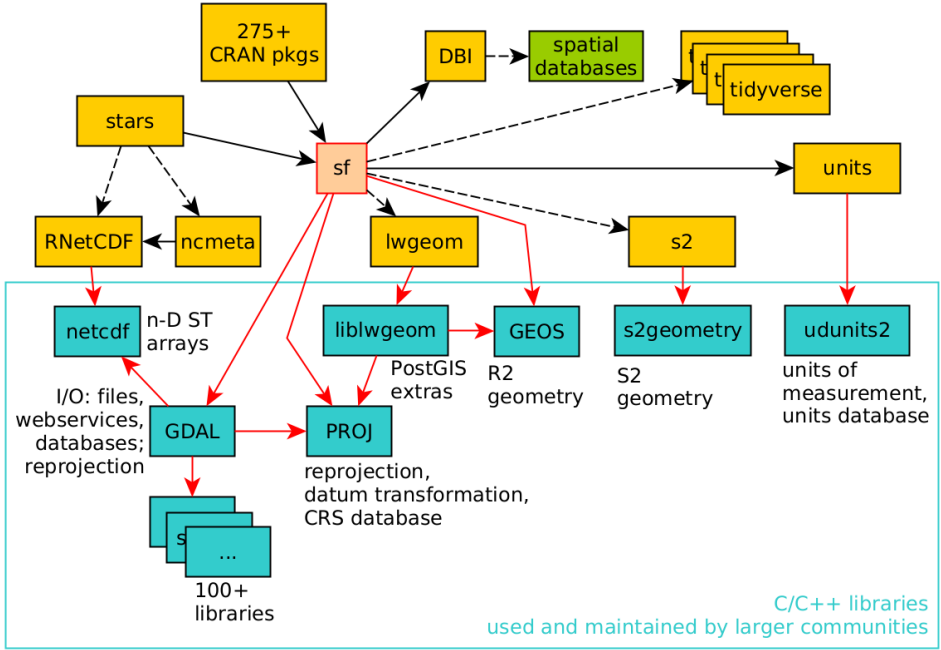
What is R, and R-Spatial?

Why do people use R?

- .. it is a free software environment for statistical computing and graphics
- it is extendible
- it runs on Win/OSX/Linux, and has no package install hell
- it has strong support for spatial data, and spatial statistics

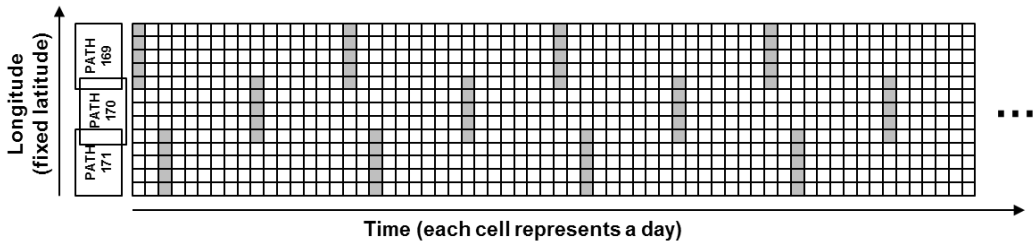
Why do people use R-Spatial?

- it solves problems, they know how
- many problems need *statistical* analysis, e.g. for model inference or assessing prediction errors
- complex graphs are easy to make
- friendly and diverse commmunity support


275+
CRAN pkgs
DBI
spatial
databases
tidyverse
stars
sf
units
RNetCDF
ncmeta
lwgeom
s2
netcdf
n-D ST
arrays
liblwgeom
GEOS
s2geometry
udunits2
PostGIS
extras
R2
geometry
S2
geometry
units of
measurement,
units database
I/O: files,
webservices,
databases;
reprojection
GDAL
PROJ
reprojection,
datum transformation,
CRS database
...
100+
libraries
C/C++ libraries
used and maintained by larger communities

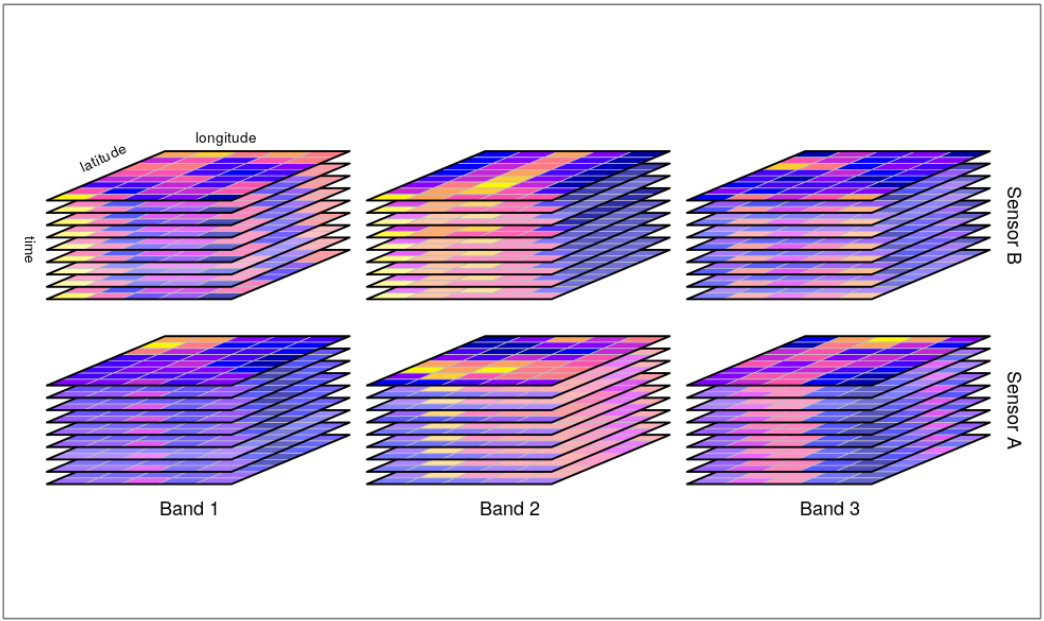
Data cubes, image collections

What we have:



Appel, M., Lahn, F., Buytaert, W. & Pebesma, E. (2018). Open and scalable analytics of large Earth observation datasets: from scenes to multidimensional arrays using SciDB and GDAL. ISPRS Journal of Photogrammetry and Remote Sensing, 138, 47-56.

What we want:


longitude
latitude
time
Sensor B
Sensor A
Band 1
Band 2
Band 3

Or even:

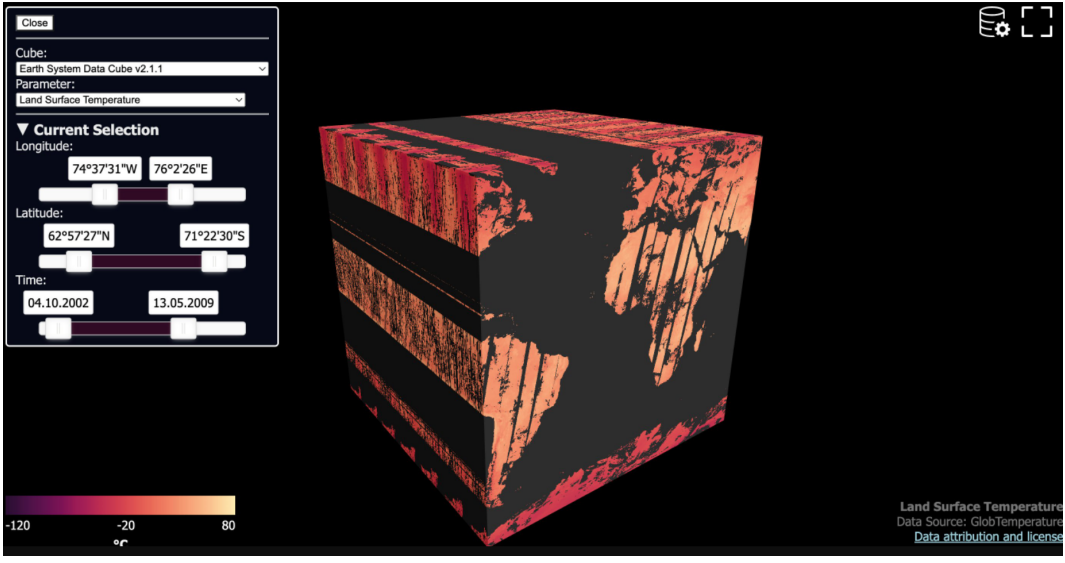


Figure 1: https://twitter.com/miguelmahechag1/status/1528653878798991360

How to cube?

Need to choose:

- spatial resolution: not everything needs to be done always at observation resolution
- target CRS, in case e.g. multiple UTM zones are covered
- spatial and/or temporal aggregation/interpolation *methods*

It seems unlikely that major EO datasets are *distributed* as cubes, because

- there is no one-fits-all
- the “best” cube seems application dependent

Creating cubes from image collections

- on-the-fly, GEE, openEO: data cube *views*
- explicitly: e.g. using R package `gdalcubes`

package gdalcubes

```
library(gdalcubes)
L8.files = list.files("/home/edzer/data/L8_cropped", pattern = ".tif",
                recursive = TRUE, full.names = TRUE)
L8.col = create_image_collection(L8.files, format = "L8_SR",
                out_file = "L8.db")
v.overview = cube_view(srs="EPSG:3857", extent=L8.col,
        dx = 500, dy = 500, dt = "P1Y", resampling = "nearest",
        aggregation = "median")
```

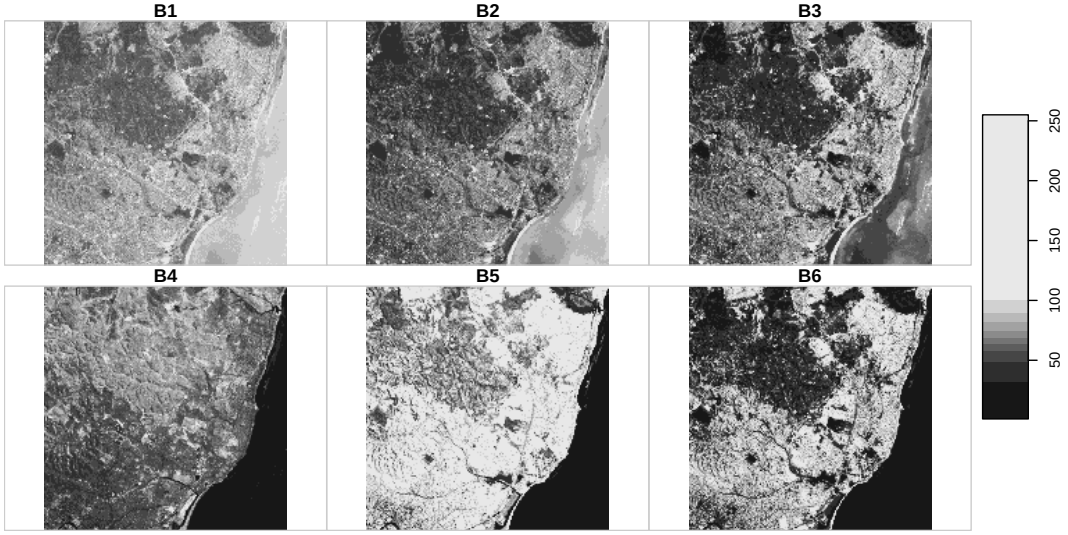
```
v.overview
## A data cube view object
##
## Dimensions:
##                  low              high count pixel_size
## t               2013              2019     7        P1Y
## y -1096876.57818711 -993876.578187112   206        500
## x -7370181.54841633 -7235181.54841633   270        500
##
## SRS: "EPSG:3857"
## Temporal aggregation method: "median"
## Spatial resampling method: "near"
L8.cube.overview = raster_cube(L8.col, v.overview)
L8.cube.overview.rgb = select_bands(L8.cube.overview,
    c("B02", "B03", "B04"))
# write_ncdf(L8.cube.overview.rgb, "L8.nc")
```

gdalcubes understands STAC collections, using R package rstac

package stars

```
library(stars)
file_name = system.file("tif/L7_ETMs.tif", package = "stars")
(r = read_stars(file_name))
## stars object with 3 dimensions and 1 attribute
## attribute(s):
##               Min. 1st Qu. Median     Mean 3rd Qu. Max.
## L7_ETMs.tif      1      54     69 68.91242      86  255
## dimension(s):
##      from  to  offset delta                       refsys point values x/y
## x       1 349  288776  28.5 SIRGAS 2000 / UTM zone 25S FALSE   NULL [x]
## y       1 352 9120761 -28.5 SIRGAS 2000 / UTM zone 25S FALSE   NULL [y]
## band    1   6      NA    NA                           NA    NA   NULL
```

```
plot(r)
```



```
read_stars(file_name, proxy = TRUE) |>
   st_set_dimensions("band", c("B1", "B2", "B3", "B4", "B5", "B6")) -> r
r
## stars_proxy object with 1 attribute in 1 file(s):
## $L7_ETMs.tif
## [1] "[...]/L7_ETMs.tif"
##
## dimension(s):
##      from  to  offset delta                       refsys point    values
## x       1 349  288776  28.5 SIRGAS 2000 / UTM zone 25S FALSE      NULL
## y       1 352 9120761 -28.5 SIRGAS 2000 / UTM zone 25S FALSE      NULL
## band    1   6      NA    NA                           NA    NA B1,...,B6
```

- lazy: delays reading pixels and subsequent computations until needed (plot, download)

Vector data cubes

```
r |> st_bbox() |> st_as_sfc() |> st_sample(10) -> pts
(e <- st_extract(r, pts))
## stars object with 2 dimensions and 1 attribute
## attribute(s):
##               Min. 1st Qu. Median  Mean 3rd Qu. Max.
## L7_ETMs.tif    11   44.25   60.5 59.35      79  131
## dimension(s):
##          from to offset delta                       refsys point
## geometry    1 10     NA    NA SIRGAS 2000 / UTM zone 25S  TRUE
## band        1  6     NA    NA                           NA    NA
##                                                        values
## geometry POINT (296565.3 9112856),...,POINT (291353.6 9115558)
## band                                                  B1,...,B6
```

```r
as.data.frame(e) |> dim()
## [1] 60  3
as.data.frame(e) |> head(3)                # "long"
##                   geometry band L7_ETMs.tif
## 1 POINT (296565.3 9112856)   B1          89
## 2   POINT (295004 9118257)   B1          69
## 3   POINT (290961 9116739)   B1          61
st_as_sf(e) |> as.data.frame() |> dim()
## [1] 10  7
st_as_sf(e) |> as.data.frame() |> head(3) # "wide"
##   B1 B2 B3 B4 B5 B6                  geometry
## 1 89 76 48 12 13 13 POINT (296565.3 9112856)
## 2 69 60 55 80 89 53   POINT (295004 9118257)
## 3 61 47 36 66 62 33   POINT (290961 9116739)
```

openEO: R client

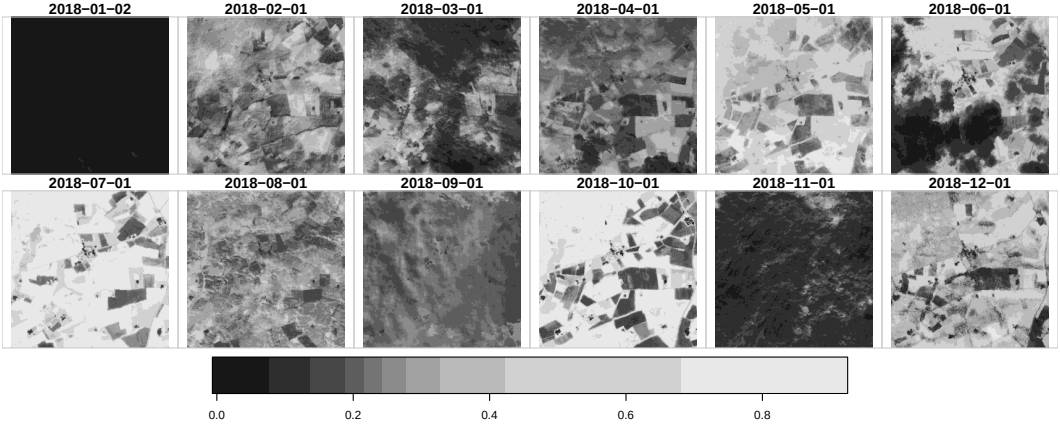
Connect, load collection:

```
library(openeo)
con = connect("https://openeo.cloud")
login()
# list_collections()
collection = "SENTINEL2_L2A"
bbox = list(west = 7, east = 7.01, south = 52, north = 52.01)
bands = c("B04", "B08")
time_range = list("2018-01-01", "2019-01-01")
p = processes()
data = p$load_collection(id = collection, spatial_extent = bbox,
          temporal_extent = time_range, bands = bands)
```

Process (asynchronous):

```
ndvi = function(data, context) {
  red = data[1]; nir = data[2]; (nir-red)/(nir+red)
}
calc_ndvi = p$reduce_dimension(data = data, dimension = "bands",
    reducer = ndvi)
intervals = ... ; labels = ...
temp_period = p$aggregate_temporal(data = calc_ndvi,
    reducer = function(data, context) p$median(data),
    intervals = intervals, labels = labels, dimension = "t")
result = p$save_result(data = temp_period, format="NetCDF")
job = create_job(graph = result,
    title = "ndvi.nc", description = "ndvi.nc",
    format = "netCDF")
start_job(job = job$id) # use the id of the job (job$id) to start the job
status(job)
dwnld = download_results(job = job$id, folder = "./") # when finished
```

```
r = read_stars("openEO.nc")
plot(r)
```



UDF: user-defined functions

local exploration, remote execution

1. obtain a small cube section, using `openeo::get_sample()`, as `stars` object
2. explore locally, developing an analysis function
3. push *exactly* the same function as UDF to the openEO backend
4. show, explore, or download the results

- At the back-end, the "full" data cube is chunked, and pulled through the UDF, again as `stars` sub-cubes.
- Currently works with `reduce_dimension` and `apply_dimension`, so that the backend *knows* which chunking strategy to use

Vision

1. Minimize modification of the script

- extend `stars_proxy` objects to "cubed" remote image collections
- running locally or in cloud, with minimal adaptation
- use UDF: test locally, deploy remotely

2. Add syntactic sugar

```
# after connect()/login():
get_collection("SENTINEL2_L2A") |>
    filter(bbox, time_range, bands) |>
    st_apply(~band, ndvi) |>
    aggregate("months", median) |>
    mapview()
```

Further packages:

- `raster`, now `terra`: stop at 3 dimensions, directly read GDAL datasets, no vector data cubes; focus on scalable and high performance
- `sits`: trains ML models and predicts using EO *time series* ; used operationally by INPE for mapping land use change in Brazil; see https://e-sensing.github.io/sitsbook/

Discussion

- vector data cubes arise naturally from raster data cubes (by sampling, aggregating over polygons)
- Data Science is multi-lingual; language cross fertilization is useful
- R-spatial welcomes new contributions, and developers willing to take responsibility
- use or search for `#rspatial` on Twitter
- Get involved, get in touch!