Mapping Chile and gridded value

Reto Schmucki

03/12/2021

Map and grid

Spatial data are often represented on a regular grid. Similar to a raster that is the simplest form of a regular grid, a grid can also be represented as a series of polygons or regular shape, size and spacing. Here we will build a vector grid over a country's border, Chile in this particular case. In this first steps, we are using functions from the R package sf, together with functions from terra and rnaturalearth that use map data provided by Natural Earth. Like many coding tasks, their is many ways to achieve similar results and I am not claiming that this the best recipe to build a grid on a map, but I find it very powerful and nicely fit in my geospatial workflow using `sf` and `terra` packages.

I you have not yet installed these packages on your systems, I recommend you install `sf` first and then `terra` because sf will install some important dependencies (GDAL and PROJ) that are also needed by `terra`. Note that we are also using the `rnaturalearthdata` package that contains the actual data that `rnaturalearth` is using.

```
library(sf)
```

```
## Linking to GEOS 3.9.1, GDAL 3.2.1, PROJ 7.2.1
```

```
library(terra)
```

```
## terra version 1.4.11
```

```
library(rnaturalearth)
library(rnaturalearthdata)
```

Natural Earth data can be accessed with the function `ne_countries()` where the `scale = medium` correspond to border contours define at 50 meters resolution, `small` and `large` referring to 110m, and 10m. Depending how precise you need the border to be and the scale you want to draw your map, one option might be better for you. In this function, we specify that we want an `sf` object to be returned.

```
## get the country borders for the world
worldmap <- rnaturalearth::ne_countries(
    scale = "medium",
    type = "map_units",
    returnclass = "sf")

## examine the firs 5 columns of the object
worldmap[,1:4]
```

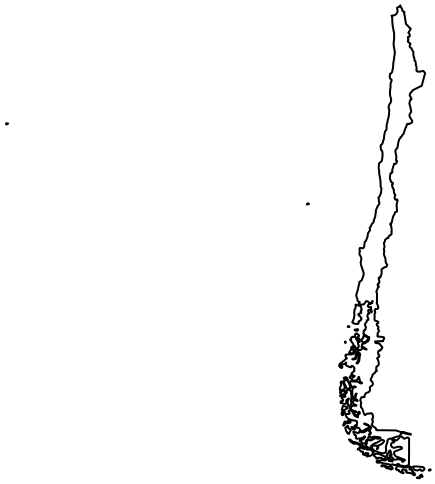
```
## Simple feature collection with 264 features and 4 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -180 ymin: -89.99893 xmax: 180 ymax: 83.59961
## CRS:           +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## First 10 features:
```

```
##   scalerank       featurecla labelrank           sovereignt
## 0         3 Admin-0 map unit         5          Netherlands
## 1         1 Admin-0 map unit         8  Antigua and Barbuda
## 2         3 Admin-0 map unit         8  Antigua and Barbuda
## 3         1 Admin-0 map unit         3          Afghanistan
## 4         1 Admin-0 map unit         3               Angola
## 5         1 Admin-0 map unit         6       United Kingdom
## 6         1 Admin-0 map unit         6              Albania
## 7         3 Admin-0 map unit         6              Finland
## 8         3 Admin-0 map unit         6              Andorra
## 9         1 Admin-0 map unit         4 United Arab Emirates
##                         geometry
## 0 MULTIPOLYGON (((-69.89912 1...
## 1 MULTIPOLYGON (((-61.71606 1...
## 2 MULTIPOLYGON (((-61.74712 1...
## 3 MULTIPOLYGON (((74.89131 37...
## 4 MULTIPOLYGON (((14.19082 -5...
## 5 MULTIPOLYGON (((-63.00122 1...
## 6 MULTIPOLYGON (((20.06396 42...
## 7 MULTIPOLYGON (((20.61133 60...
## 8 MULTIPOLYGON (((1.706055 42...
## 9 MULTIPOLYGON (((53.92783 24...
```

A first map

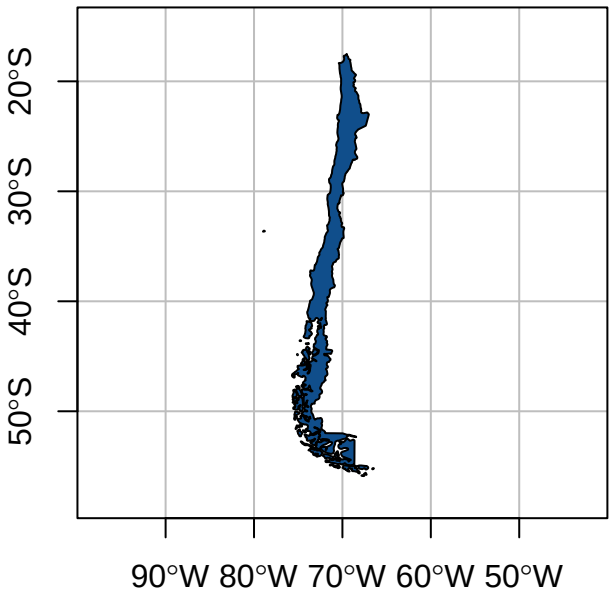
From this object, `worldmap`, we can subset a specific country and produce a first map using the plot method. Because the worldmap sf object contains a data.frame and a spatial feature, the plot function will produce a plot for each attribute contained in the data.frame. To only represent the spatial feature, the border, we use the function `st_geometry()` on the sf object. The alternative would be to name the geometry column in the object, but be aware that the name for the geometry column might differ. The default plot method in `sf` is powerful and gives you options to display graticules, select color, show axis values or set limit with extent (seed sf vignette). Other graphical environments are also available and I show how `sf` object can be used with ggplot2 packages below.

```
plot(st_geometry(worldmap[worldmap$name == "Chile", ]))
```



```r
## alternative
## plot(worldmap[worldmap$name == "Chile", "geometry"])

plot(st_geometry(worldmap[worldmap$name == "Chile", ]),
    graticule = TRUE,
    col = "dodgerblue4",
    axes = TRUE,
    extent = terra::ext(c(-75, -65, -58, -15)))
```



Building a regular grid

Building a grid is made very easy with the function `sf::st_make_grid()`. This function can build a grid using a simple feature such as the border of Chile the we used above or using the bounding box of another spatial feature, the extent of a raster of define by a vector. You noted that when mapping Chile, some islands in the Pacific are stretching the map far beyond the continental seashore. Here we might be interested in limiting the grid to a certain range of longitude. An easy way to do this is by defining the coordinates that define the bounding box of interest. We can define the bounding box with the `terra::ext()` function from the terra package, combined with the `sf::st_bbox()` function from `sf`. Note that we defined a vector for with the coordinates for the minimum and maximum values along the x and y axis, using c(xmin, xmax, ymin, ymax) in this order. We also provide a coordinate reference system (crs) for the longitude latitude system, using the value 4326 (epsg:4326). When building a grid, we can define the size of the grid with `cellsize` argument, here set to 1°, and define the type of objects we want, here `polygons`. By default the polygons are square, but they could be hexagonal if "square=FALSE". Note that if you only want 1 rectangle around the bounding box, you can use the argument `n=1` instead of cellsize. This specify the number of cell you want per dimension *see documentation*.

Because `sf::st_make_grid()` produce the geometry part of the simple feature, you can add data with the function `sf::st_as_sf()`, defining variables as you would to build a data.frame, adding them to the geometry. Here we add an id for each grid cell `gid` and a random variable that we name `value_x`. These variable should be the same length as the geometry column.

```
## build 1&deg; grid
chl_grid <- sf::st_make_grid(
                sf::st_bbox(terra::ext(c(-76, -66, -58, -16)), crs = 4326),
                cellsize = 1,
                what = "polygons",
```

```
                square = TRUE)

## build sf object with data for gid and value_x
chl_grid <- sf::st_as_sf(
                gid = seq_along(chl_grid),
                value_x = sample(c(1:10, NA),
                                 length(chl_grid),
                                 replace = TRUE),
                chl_grid)
```

Cropping or intersecting

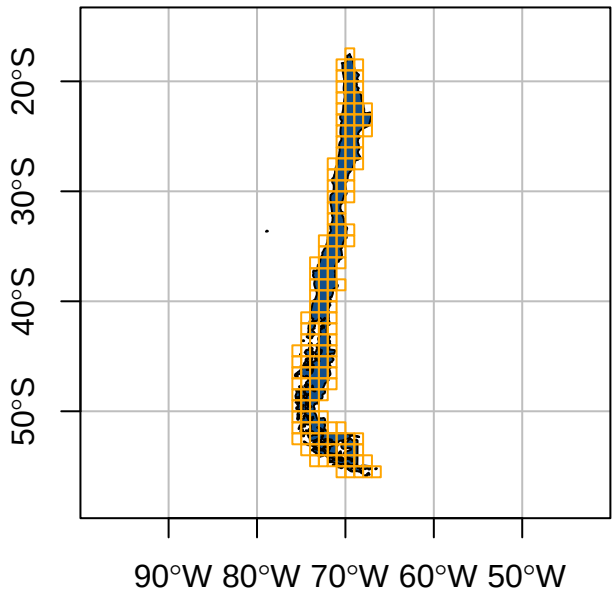
If you are interested in displaying only the terrestrial part of your grid, the fastest way to identify and subset those grid cell is by using `sf::st_intersects()` which return a logical value (TRUE or FALSE) if polygons in x intersects part of polygons in y. Be aware that st_intersects differs from st_intersection, the latter returning a new sf object with the polygons in x cropped to the portion that overlap y. Because we want to keep the entire grid cell we use st_intersects, which is also faster than st_intersection. To retrieve a vector of TRUE/FALSE, we set sparse = FALSE.

From the newly created grid, we can overlay the two objects on a map, using the plot method, with the argument `add=TRUE` to add the new layer over an already existing map.

```
## subset gridcell touching Chile
chl_grd_int <- chl_grid[st_intersects(chl_grid,
    worldmap[worldmap$name == "Chile", ],
    sparse = FALSE), ]

## plot country and grid
plot(st_geometry(worldmap[worldmap$name == "Chile", ]),
    graticule = TRUE,
    col = "dodgerblue4",
    axes = TRUE,
    extent = terra::ext(c(-75, -65, -58, -15)),
    reset = FALSE)

## add the terrestrial grid over Chile
plot(st_geometry(chl_grd_int),
    col = NA,
    border = 'orange',
    add = TRUE)
```



Mapping with ggplot2

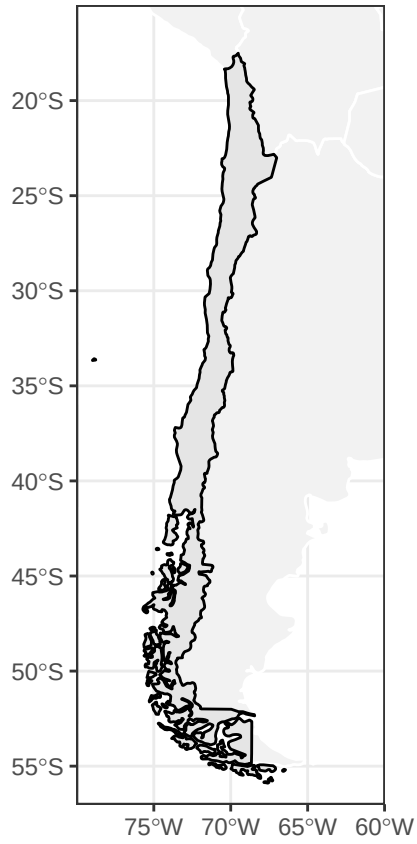
Many data visualization in R are build with ggplot2 and map production based on spatial objects is no exception. Although other frameworks exist and can be used to create beautiful maps in R, I will focus on the use of ggplot2. To add simple feature to a ggplot, we use the `geom_sf` that can use sf data and use various aesthetic to define the color to be used to fill area and draw borders (polygons contours). As we did above with the `plot()` function, we can add layer to our map by using `geom_sf()` with several sf objects. We can also define the limits of the coordinates we want to represent with the `coord_sf()` and use the standard ggplot functions to define other aspect of the figure such as the theme.

In our example, we can build map Chile with its surrounding country, using a light grey color for other country and a darker shade to highlight Chile. To keep our map minimal, we can use white borders and black and with background (`theme_bw()`).

```
# map with geom_sf and ggplot2
library(ggplot2)

p1 <- ggplot() +
        geom_sf(data = worldmap[worldmap$name == "Argentina", ],
            fill = 'grey95', col='white') +
        geom_sf(data = worldmap[worldmap$name == "Peru", ],
            fill = 'grey95', col='white') +
        geom_sf(data = worldmap[worldmap$name == "Bolivia", ],
            fill = 'grey95', col='white') +
        geom_sf(data = worldmap[worldmap$name == "Paraguay", ],
            fill = 'grey95', col='white') +
        geom_sf(data = worldmap[worldmap$name == "Brazil", ],
```

```
            fill = 'grey95', col='white') +
        geom_sf(data = worldmap[worldmap$name == "Chile", ],
            col='black') +
        coord_sf(xlim = c(-80, -60), ylim = c(-15,-57), expand = FALSE)  +
        theme_bw()
p1
```



Multilayer map with legend

Adding data to the a base map is just a matter of adding new layers with `geom_sf()`. In this way, we can add the grid that cover the continental part of Chile and highlight a specific variable contained in simple feature. We can display the `value_x` to as a continuous or a discrete variable. Many R packages can help you handle colour and define gradient in ggplot2. One that I find particularly good is paletteer. Selecting the right set of colour is an art and it can be helpful to explore some available pallets at r-chart website.

```
library(cowplot)
library(patchwork)
```

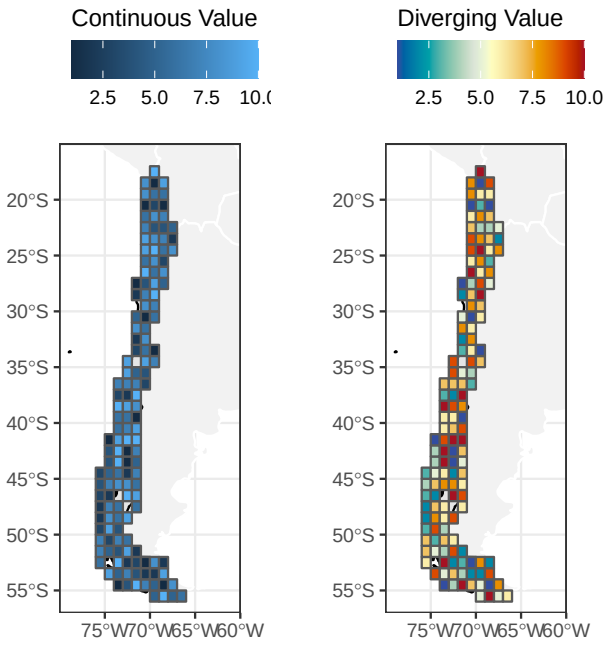
```
##
## Attaching package: 'patchwork'
## The following object is masked from 'package:cowplot':
##
##     align_plots
## The following object is masked from 'package:terra':
##
```

```
##     area
```

```
# with scale_fill_gradient
p2 <- ggplot() +
    geom_sf(data = worldmap[worldmap$name == "Argentina", ],
        fill = "grey95", col = "white") +
    geom_sf(data = worldmap[worldmap$name == "Peru", ],
        fill = "grey95", col = "white") +
    geom_sf(data = worldmap[worldmap$name == "Bolivia", ],
        fill = "grey95", col = "white") +
    geom_sf(data = worldmap[worldmap$name == "Paraguay", ],
        fill = "grey95", col = "white") +
    geom_sf(data = worldmap[worldmap$name == "Brazil", ],
        fill = "grey95", col = "white") +
    geom_sf(data = worldmap[worldmap$name == "Chile", ], col = "black") +
    geom_sf(data = chl_grd_int[!is.na(chl_grd_int$value_x),],
        aes(fill = value_x)) +
    scale_fill_gradient(na.value = NA) +
    coord_sf(xlim = c(-80, -60), ylim = c(-15, -57), expand = FALSE) +
    guides(fill = guide_colorbar(
                title = "Continuous Value",
                        title.position = "top")) +
    theme_bw() +
    theme(legend.position = "top",
    legend.justification = "left",
    plot.margin = margin(t = 0.5, r = 0.5, b = 0, l = 1, unit = "cm"))

# with paletteer::scale_fill_paletteer_c()
p3 <- ggplot() +
    geom_sf(data = worldmap[worldmap$name == "Argentina", ],
        fill = "grey95", col = "white") +
    geom_sf(data = worldmap[worldmap$name == "Peru", ],
        fill = "grey95", col = "white") +
    geom_sf(data = worldmap[worldmap$name == "Bolivia", ],
        fill = "grey95", col = "white") +
    geom_sf(data = worldmap[worldmap$name == "Paraguay", ],
        fill = "grey95", col = "white") +
    geom_sf(data = worldmap[worldmap$name == "Brazil", ],
        fill = "grey95", col = "white") +
    geom_sf(data = worldmap[worldmap$name == "Chile", ], col = "black") +
    geom_sf(data = chl_grd_int[!is.na(chl_grd_int$value_x),],
     aes(fill = value_x)) +
    paletteer::scale_fill_paletteer_c("grDevices::RdYlBu", na.value = NA,
        direction = -1) +
    coord_sf(xlim = c(-80, -60), ylim = c(-15, -57), expand = FALSE) +
    guides(fill = guide_colorbar(
                title = "Diverging Value",
                        title.position = "top")) +
    theme_bw() +
    theme(legend.position = "top",
    legend.justification = "left",
    plot.margin = margin(t = 0.5, r = 0.5, b = 0, l = 1, unit = "cm"))

(p2 | p3)
```


Continuous Value
2.5
5.0
7.5
10.0
Diverging Value
2.5
5.0
7.5
10.0
20°S
25°S
30°S
35°S
40°S
45°S
50°S
55°S
75°W
70°W
65°W
60°W