Geospatial data analysis - Part 2

2023-10-03

Note: This resource is adapted from the book: "Spatial Data Science with R and"terra" (https://rspatial.org/index.html 2019-2023. License: CC BY-SA 4.0.), "The terra package for raster and vector data" tutorial (https://www.paulamoraga.com/tutorial-terra), and "leatlet for R" (https://rstudio.github.io/leaflet/).

I. Raster data

Continuous spatial data or fields (such as elevation) are commonly represented using a **raster data** structure. A raster divides the world into a grid of equally sized rectangles (referred to as cells or, in the context of satellite remote sensing, pixels) that all have one or more values (or missing values) for the variables of interest. A raster cell value should normally represent the average (or majority) value for the area it covers. However, in some cases the values are actually estimates for the center of the cell (in essence becoming a regular set of points with an attribute).

`SpatRaster` objects from scratch (terra package):

A `SpatRaster`represents multi-layer (multi-variable) raster data. A `SpatRaster` stores a number of fundamental parameters describing its geometry, such as the number of columns and rows, the spatial extent, and the Coordinate Reference System.

- Use the function `rast` to create a `SpatRaster` object.

```
library(terra)
# Create a raster with 100 rows and 100 columns (10000 cells)
r <- rast(ncol=100, nrow=100, xmin=-150, xmax=-80, ymin=20, ymax=60)
r
```

Note: At this stage the raster object only has the geometry of a raster data set. That is, it knows about its location, resolution, etc., but there are no values associated with it.

- Use the function `values` to assign values to a `SpatRaster` object. In this example we assign some random data, but in reality raster values should be associated to meaningful information about the spatial data linked to (e.g. elevation, temperature, etc.).

```
# Assigning random numbers
values(r) <- runif(ncell(r))
r
plot(r)
```

- You can use a matrix to assign values to a `SpatRaster` object provided the number of row and column in the matrix corresponds to the number of rows and columns of the `SpatRaster`. In this example we use a matrix to assign data from a 2D function to our `SpatRaster`.

Try it yourself: Change the amplitudes and frequencies in the function to create different patterns and plot them.

```
Rdata <- matrix(ncol = 100, nrow = 100)
for (i in 1:nrow(Rdata)) {
  for (j in 1:ncol(Rdata)) {
    Rdata[i,j] <- 2*sin(i/15)+3*cos(j/15)
  }
}

values(r) <- Rdata
terra::plot(r)
```

- You can add polygons and points to a `SpatRaster` object.

```
terra::plot(r)
# add polygon and points
lon <- c(-116.8, -114.2, -112.9, -111.9, -114.2, -115.4, -117.7)
lat <- c(41.3, 42.9, 42.4, 39.8, 37.6, 38.3, 37.6)
lonlat <- cbind(id=1, part=1, lon, lat)
pts <- vect(lonlat)
pols <- vect(lonlat, type="polygons", crs="+proj=longlat +datum=WGS84")
points(pts, col="red", pch=20, cex=3)
lines(pols, col="blue", lwd=2)
```

- You can concatenate `SpatRaster` objects to create a multilayer object. In the following example we create a new `SpatRaster` object with 3 layers. You can subset layers with the `[[ ]]` operator.

```r
r2 <- r * r
r3  <- sqrt(r)
s <- c(r, r2, r3)
s

#plot all layers
terra::plot(s)

#plot only layer 2
terra::plot(s[[2]])
```

Reading spatial raster files

While we can create a `SpatRaster` from scratch, it is more common to do so from a file. The `terra` package can use raster files in several formats, including GeoTiff, ESRI, ENVI, and ERDAS.

For this example we are using the "DIF.tiff" file from the course website. This file is a geoTIFF map containing Colombia's long-term yearly average of diffuse horizontal irradiation (DIF) in kWh/m2, covering the period 1999-2018. Data was downloaded from https://globalsolaratlas.info/download/colombia (***Gis data - LTAym_AvgDailyTotals (GeoTIFF).zip***).

- Following code assumes that you have downloaded the "DIF.tiff" into your current working directory. Use the `rast()` function to load a raster data. You can verify whether the imported raster file has data by using the `hasValues()` function. You can retrieve the values from a `SpatRaster` object by using the function `as.data.frame()`.

```r
library(terra)
f <- paste0(getwd(),"/DIF.tif")
basename(f)
colombiaDIF <- terra::rast(f)
hasValues(colombiaDIF)

# get raster values
data <- as.data.frame(colombiaDIF)
```

The (`terra`) `plot()` function accepts colour palettes in the same way that the `plot()` R base function. Use the `col` parameter to set the colour property of your plot.

- R base palettes include: `cm.colors`, `topo.colors`, `terrain.colors`, `heat.colors`, and `rainbow`. In this example we use 10 colours from the `terrain.colors` palette.

```
terra::plot(colombiaDIF,col= terrain.colors(10))
```

- The `RColorBrewer` package includes several further palette options. use the function `display.brewer.all()` to show them. Use the `brewer.pal(n, "name")` to select the desired palette.

```
library(RColorBrewer)
display.brewer.all()
terra::plot(colombiaDIF,col= brewer.pal(11, "Spectral"))
```

- Add borders to the raster with the `lines()` function

```
# download Colombian shape with rnaturalearth package
library(rnaturalearth)
colombiaShape <- terra::vect(ne_states(country = "colombia"
                                       ,returnclass = "sf"))

terra::plot(colombiaDIF,col= brewer.pal(11, "Spectral"))
terra::lines(colombiaShape)
```

Basic raster manipulation

Many generic functions that allow for simple and elegant raster algebra have been implemented for `Raster` objects, including the normal algebraic operators such as +, −, ∗, /, logical operators such as >, >=, <, ==, `!` and functions like `abs`, `round`, `ceiling`, `floor`, `trunc`, `sqrt`, `log`, `log10`, `exp`, `cos`, `sin`, `atan`, `tan`, `max`, `min`, `range`, `prod`, `sum`, `any`, `all`. In these functions you can mix `raster` objects with numbers, as long as the first argument is a `raster` object.

- In the following code we performed a data transformation to get Colombian diffuse horizontal irradiation (DIF) in log(Watt*h/m2).

```
terra::plot(colombiaDIF
      , col = brewer.pal(11, "Spectral")
      , main="Colombian diffuse horizontal irradiation (DIF) in kWh/m2")


colombiaDIFwatts <- log(1000*colombiaDIF)
terra::plot(colombiaDIFwatts
  ,col = brewer.pal(11, "Spectral")
  ,main="Colombian diffuse horizontal irradiation (DIF) in Wh/m2 (log scale)")
```

- Use the function `extract()` the retrieve the values of a `SpatRaster` object for a set of locations. The locations can be a SpatVector (points, lines, polygons), a matrix with (x, y) or (longitude, latitude – in that order!) coordinates, or a vector with cell numbers.

```
# Extract raster cells within each polygon
aux <- terra::extract(x=colombiaDIF, y=colombiaShape, na.rm = TRUE)
head(aux)
```

- You can use summary functions to get summary values per geometry. Use the `fun` parameter to set the summary function to be used.

```
aux <- extract(colombiaDIF, colombiaShape, fun = "mean", na.rm = TRUE)
colombiaShape$meanDIF <- aux$DIF
head(colombiaShape$meanDIF)
```

- Use the `rasterize` function to create a `SpatRaster` object from a `SpatVector`. Additionally, you can use the `text()` function to add some text to inside the polygons of a `SpatVector` object.

```
#create an empty raster. Grid resolution: 1000 x 1000
colombiaRaster <-  rast(colombiaShape, ncol=1000, nrow=1000)

#Use a feature from the SpatVector as raster values
colombiaRaster <- rasterize(colombiaShape,colombiaRaster,"meanDIF")
terra::plot(colombiaRaster, col=brewer.pal(11, "Spectral"))
terra::lines(colombiaShape)
text(colombiaShape, colombiaShape$name, inside=T, cex=0.6)
```

- Use `writeRaster` to write raster data. You must provide a SpatRaster and a filename. The file format will be guessed from the filename extension. If that does not work you can provide an argument like `format=GTiff`. Following code saves the new Colombia raster, with the average of DIF per administrative department.

```
x <- writeRaster(colombiaRaster, "colombiaRaster.tif", overwrite=TRUE)
```

II. Useful packages for maps.

The `maptiles` package

- You can get many different base-maps with the **`maptiles`** package. Use the `provider` parameter to select the server from which to get the map (e.g. OpenStreetMap-*default*, Stamen, Esri, CARTO, or Thunderforest). Further information can be found at https://github.com/riatelab/maptiles/

- Set `crop=TRUE` if results should be cropped to the specified x extent.
- You can set the `zoom` parameter to adjust the map's resolution. Use the correct zoom level as described in https://wiki.openstreetmap.org/wiki/Zoom_levels
- You can use an `ext(xmin,xmax,ymin,ymax)` to get the tiles. `x` refers to longitude, while `y` refers to latitude.

```
library(maptiles)
bg <- get_tiles(ext(colombiaShape))
plotRGB(bg)
lines(colombiaShape, col="blue", lwd=1)

bg <- get_tiles(ext(colombiaShape), crop=TRUE, zoom=5)
plotRGB(bg)
lines(colombiaShape, col="blue", lwd=1)

# You can use an extend object to get the tiles. ext(xmin,xmax,ymin,ymax)
bogotaExt <- terra::ext(-74.2,-73.98, 4.45, 4.84)
bg <- get_tiles(bogotaExt,crop=TRUE, zoom = 11)
plotRGB(bg)
```

The `leaflet` package

You can use the `leaflet` package to make interactive maps.

- Use the `plet` function to plot the values of a SpatRaster or SpatVector to make an interactive leaflet map that is displayed in a browser.

```
library(leaflet)
#keep only 2 features to be shown when the map is clicked.
colombiaShape2 <- colombiaShape[c("adm1_code","name")]

#create interactive map
m <- plet(colombiaShape2)
m
```

- Use `addMarkers()` to a add marker in the given coordinates. You can use the `popup` parameter to assign a message to be displayed when the marker is clicked.

```
m <- leaflet() %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addMarkers(lng=174.768, lat=-36.852, popup="The birthplace of R")
m  # Print the map
```

- You can use a data frame to add more than one marker at a time. Order of coordinates in the data frame needs to be *longitude*, *latitude*.

```
library(leaflet)
data(quakes)
head(quakes)

df <- quakes[1:20,c("long","lat")]
leaflet(df) %>% addMarkers() %>% addTiles()
```

- You can use functions to add the markers, and additional information.

```
leaflet(data = quakes[1:20,]) %>% addTiles() %>%
  addMarkers(~long, ~lat, popup = ~as.character(mag), label = ~as.character(mag))
```

- Use `addAwesomeMarkers()` to specify custom colors for the markers as well as icons from the Font Awesome, Bootstrap Glyphicons, and Ion icons icon libraries.

```
df.20 <- quakes[1:20,]

getColor <- function(quakes) {
  sapply(quakes$mag, function(mag) {
  if(mag <= 4) {
    "green"
  } else if(mag <= 5) {
    "orange"
  } else {
    "red"
  } })
}

icons <- awesomeIcons(
  icon = 'ios-close',
  iconColor = 'black',
  library = 'ion',
  markerColor = getColor(df.20)
)

leaflet(df.20) %>% addTiles() %>%
  addAwesomeMarkers(~long, ~lat, icon=icons, label=~as.character(mag))
```

- You can use the `saveWidget()` function from the `htmlwidgets` package to save your map as html.

```
m <- leaflet(df.20) %>% addTiles() %>%
  addAwesomeMarkers(~long, ~lat, icon=icons, label=~as.character(mag))

library(htmlwidgets)
saveWidget(m, file="m.html")
```

- Alternatively, if you run your code for a script or the console, you can use RStudio GUI to save it as html.

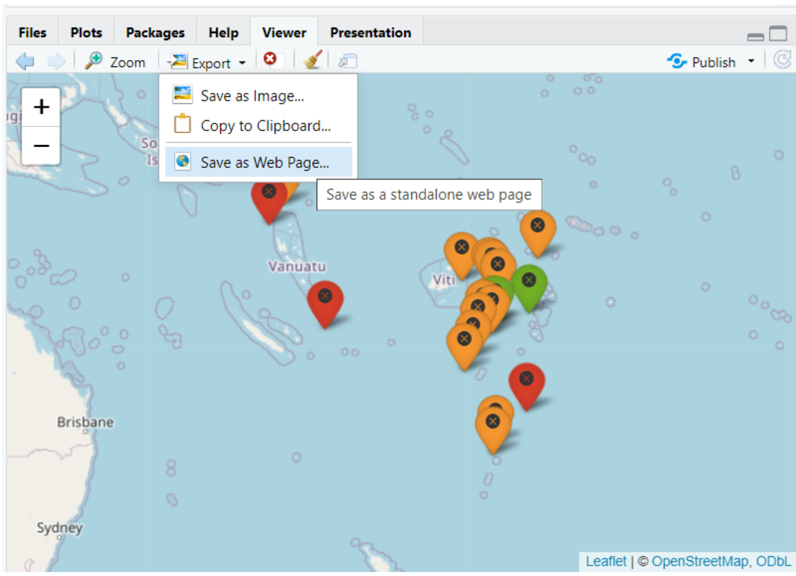

Figure 1: Saving an interactive map from RStudio