Mapa bivariante dasimétrico

Brown Bag Seminar - InnovaLab | Dr. Dominic Royé

04 mayo 2022

Índice

1. Consideraciones iniciales

Una desventaja de los mapas coropletas es que estos suelen distorsionar la relación entre la verdadera geografía subyacente y la variable representada. Se debe a que las divisiones administrativas no suelen coincidir con la realidad geográfica, donde la gente vive. Además, grandes áreas aparentan tener un peso con poca población que no tienen realmente. Para reflejar mejor la realidad se hace uso de distribuciones más realista de la población como puede ser el uso de suelo. Con técnicas de Sistemas de Información Geográfica es posible redistribuir la variable de interés en función de una variable a menor unidad espacial.

Cuando disponemos de datos de puntos, el proceso de redistribución simplemente es recortar áreas de puntos con población a base del uso de suelo, normalmente clasificado como urbano. En caso de polígonos también podríamos recortar con polígonos de uso de suelo, pero una alternativa interesante son los mismos datos en formato raster. Veremos cómo podemos realizar un mapa dasimétrico usando datos raster con una resolución de 100 m. En este post usaremos datos de secciones censales de la renta media y el índice de Gini de España. No sólo haremos un mapa dasimétrico, sino también bivariante, representando con dos gamas de colores ambas variables en el mismo mapa.

2. Paquetes

En este ejemplos añadimos tres nuevos paquetes: *{biscale}* para crear una variable bivariante, *{patchwork}* para combinar diferentes gráficos basado en *{ggplot2]* y *{janitor}* que ayuda en limpiar, entre otras cosas, los nombres de columnas.

```
# paquetes
library(tidyverse)
library(sf)
library(biscale)
library(patchwork)
library(terra)
library(janitor)
```

3. Preparación

3.1. Datos

- CORINE Land Cover 2018 (geotiff) [COPERNICUS]
- Datos de renta, edad media (csv) [INE]
- Límites censales y municipios de España (vectorial) [INE]

3.2. Importar

Lo primero que hacemos es importar el raster del uso de suelo, los datos de renta y edad media, además de los límites censales y municipales. La función `clean_names()` convierte nombres de columnas en un formato límpio.

```
# raster de CORINE LAND COVER 2018
urb <- rast("./data/U2018_CLC2018_V2020_20u1.tif")

# datos de renta y Gini
renta <- read_csv2("./data/30824.csv") %>% clean_names()
edad <- read_csv2("./data/30832.csv") %>% clean_names()

# límites censales del INE y municipios
limits <- st_read("./data/SECC_CE_20200101.shp")
```

```
## Reading layer 'SECC_CE_20200101' from data source 'C:\Users\xeo19\OneDrive - Universidade de Santiag
## Simple feature collection with 36309 features and 20 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -1004502 ymin: 3132130 xmax: 1126932 ymax: 4859240
## Projected CRS: ETRS89 / UTM zone 30N
```

```
muni <- st_read("./data/au_AdministrativeUnit_4thOrder0.gml")
```

```
## Reading layer 'AdministrativeUnit' from data source 'C:\Users\xeo19\OneDrive - Universidade de Santi
## Simple feature collection with 8217 features and 15 fields
```

```
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -18.16118 ymin: 27.63772 xmax: 4.327785 ymax: 43.79238
## Geodetic CRS:  ETRS89
```

3.3. Usos de suelo

En este primer paso filtramos las secciones censales para obtener aquellas de la Comunidad Autónoma de Madrid.

```
# filtramos la Comunidad Autónoma de Madrid
limits <- filter(limits, NCA == "Comunidad de Madrid")
```

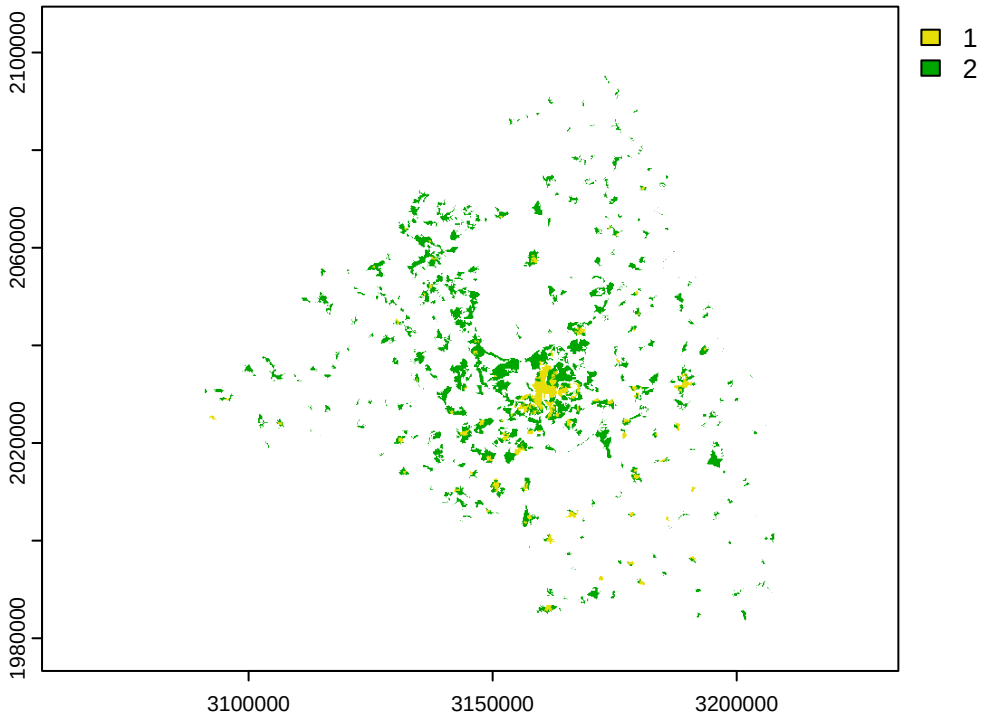
En el siguiente paso recortamos el raster de uso de suelo con los límites de Madrid. Recomiendo usar siempre primero la función `crop()` y después `mask()`, la primera recorta a la extensión requerida y la segunda enmascara. Posteriormente, eliminamos todos los valores que correspondan a 1 o 2 (urbano continuo, discontinuo). Por último, proyectamos el raster.

```
# proyectamos los límites
limits_prj <- st_transform(limits, crs(urb))

# acortamos y enmascaramos
urb_mad <- crop(urb, limits_prj) %>%
             mask(vect(limits_prj))

# eliminamos píxeles no urbanos
urb_mad[!urb_mad %in% 1:2] <- NA

# plot del raster
plot(urb_mad)
```




```
# proyectamos
urb_mad <- project(urb_mad, "EPSG:25830")
```

En este siguiente paso, convertimos los datos raster en un objeto `sf` de puntos.

```
# transformamos el raster a xyz y objeto sf
urb_mad_sf <- as.data.frame(urb_mad, xy = TRUE) %>%
                st_as_sf(coords = c("x", "y"), crs = 25830)
```

3.4. Renta y edad media

Lo que nos queda es crear una columna con los códigos de las secciones censales y excluir datos que corresponden a otro nivel administrativo.

```
## datos renta y edad media

renta_sec <- mutate(renta, GEOCODE = str_extract(unidades_territoriales,
                                                 "[0-9]{5,10}"),
                    nc_len = str_length(GEOCODE),
                    mun_name = str_remove(unidades_territoriales,
                                          GEOCODE) %>% str_trim()) %>%
             filter(nc_len > 5)

edad_sec <- mutate(edad, GEOCODE = str_extract(unidades_territoriales,
```

```
                                              "[0-9]{5,10}"),
              nc_len = str_length(GEOCODE),
              mun_name = str_remove(unidades_territoriales,
                                    GEOCODE) %>% str_trim()) %>%
            filter(nc_len > 5)
```

En el siguiente paso filtramos las variables y el año de interés; unimos ambas tablas con las secciones censales usando `left_join()` y convertimos columnas de interés en modo numérico.

```
renta_sec <- filter(renta_sec,
                    indicadores_de_renta_media == "Renta neta media por persona",
                    periodo == 2019) %>%
            select(GEOCODE, renta = total) %>%
             mutate(renta = parse_number(renta,
                                         locale = locale(decimal_mark = ",",
                                                         grouping_mark = "."))))

edad_sec <- filter(edad_sec,
                    indicadores_demograficos == "Edad media de la población",
                    periodo == 2019) %>%
            select(GEOCODE, edad = total) %>%
            mutate(edad = parse_number(edad,
                                         locale = locale(decimal_mark = ",",
                                                         grouping_mark = "."))))
```

```
# unimos ambas tablas de renta y edad media con los límites censales
mad <- left_join(limits, renta_sec, by = c("CUSEC"="GEOCODE")) %>%
          left_join(edad_sec, by = c("CUSEC"="GEOCODE"))
```

3.5. Variable bivariante

Para crear un mapa bivariante debemos construir una única variable que combina diferentes clases de dos variables. Normalmente son tres de cada una lo que lleva a nueve clases en total. En nuestro caso, la renta media y la edad media. El paquete `biscale` incluye funciones auxiliares para llevar a cabo este proceso. Con la función `bi_class()` creamos esta variable de clasificación usando cuantiles como algoritmo. Dado que en ambas variables encontramos valores ausentes, corregimos aquellas combinaciones entre ambas variables donde aparece un `NA`.

```
## creamos clasificación bivariante
mapbivar <- bi_class(mad, renta, edad, style = "quantile", dim = 3) %>%
             mutate(bi_class = ifelse(str_detect(bi_class, "NA"), NA, bi_class))

# resultado
#head(dplyr::select(mapbivar, renta, edad, bi_class))
```

Terminamos redistribuyendo la variable bivariante sobre los píxeles (o puntos) del uso de suelo urbano. La función `st_join()` une los datos con los puntos del uso de suelo por su ubicación geográfica.

```
## redistribuimos los píxeles urbanos a la desigualdad
mapdasi <- st_join(urb_mad_sf, mapbivar)
```

4. Construcción del mapa

4.1. Leyenda y fuente

Antes de construir ambos mapas debemos crear la leyenda usando la función `bi_legend()`. En la función definimos los títulos para cada variable, el número de dimensiones y la gama de colores.

```
# leyenda bivariante
legend2 <- bi_legend(pal = "DkViolet",
                     dim = 3,
                     xlab = "Más renta",
                     ylab = "Más edad",
                     size = 9)
```

4.2. Mapa dasimétrico

Primero preparamos los límites municipales de Madrid. Filtramos nuestro datos usando parte del código nacional que identifica cada municipio:

- 34: Pais
- 13: Comunidad Autónoma. Madrid
- 28: Provincia (Segovia)
- 28079: Código Municipio. Madrid

Además, reproyectamos los municipios.

```
muni <- filter(muni, str_sub(nationalCode, 3, 4) == 13) %>%
            st_transform(25830)
```

Para situar la leyenda en su posición debemos pasar las coordinadas geográficas a la proyección en uso (EPSG:25830).

```
legend_pos <- st_bbox(c(xmin = -3.25,  ymin = 40.55,
                        xmax = -2.65, ymax = 40.95), crs = 4326) %>%
                st_as_sfc() %>%
                st_transform(25830) %>%
                st_bbox()

legend_pos
```

```
##      xmin      ymin      xmax      ymax
##  478832.3 4488834.8  529634.8 4533265.6
```

Este mapa construimos usando `geom_tile()` para los píxeles y `geom_sf()` para los límites municipales. Además, será el mapa de la derecha donde ubicamos también la leyenda. Para añadir la leyenda hacemos uso de la función `annotation_custom()` indicando la posición en las coordenadas geográficas del mapa. El paquete *{biscale}* también nos ayuda con la definición del color a través de la función `bi_scale_fill()`.

```
p2 <- ggplot(mapdasi) +
  geom_tile(aes(fill = bi_class,
                geometry = geometry),
            stat = "sf_coordinates",
            show.legend = FALSE) +
  geom_sf(data = muni,
          color = "grey70",
          fill = NA,
         size = 0.1) +
  annotation_custom(ggplotGrob(legend2),
                    xmin = legend_pos[1], xmax = legend_pos[3],
                    ymin = legend_pos[2], ymax = legend_pos[4]) +
  bi_scale_fill(pal = "DkViolet",
                dim = 3,
                na.value = "grey90") +
  labs(title = "dasimétrico", x = "", y = "") +
  theme_void(base_family = "Bahnschrift") +
  theme(plot.title = element_text(hjust = .5,
                                  size = 30,
                                  face = "bold")

        ) +
  coord_sf()
```

4.3. Mapa coropleta

El mapa coropleta se construye de forma similar al mapa anterior con la diferencia de que usamos `geom_sf()`.

```
p1 <- ggplot(mapbivar) +
  geom_sf(aes(fill = bi_class),
          colour = NA,
          size = .1,
          show.legend = FALSE) +
  geom_sf(data = muni,
          color = "white",
          fill = NA,
          size = 0.1) +
  bi_scale_fill(pal = "DkViolet",
                dim = 3,
                na.value = "grey90") +
  labs(title = "coroplético",  x = "", y ="") +
  bi_theme(base_family = "Bahnschrift") +
  theme(plot.title = element_text(hjust = .5,
                                  size = 30,
                                  face = "bold")

        ) +
  coord_sf()
```

4.4. Combinar ambos mapas

Con ayuda del paquete `patchwork` combinamos ambos mapas en una única fila, primero el mapa coropleta y a su derecha el mapa dasimétrico. Más detalles de la gramática que se usa para la combinación de gráficos aquí.

```
# Combinamos
p <- p1 | p2

p
```