

Datos espacio-temporales I: Vectorial

Curso de Invierno UCM - Analizando datos con R | Dr. Dominic Royé

02 febrero 2022

Índice

1. El paquete sf	1
1.1. Lectura	2
1.2. Visualizar	4
1.3. Proyectar y coordenadas	6
1.4. Manipulación de los atributos	7
1.5. Funciones de análisis espacial	15
2. Cambiar entre formatos: sp, sf, data.frame	20
3. Exportación de datos	21
4. Visualización vectorial con ggplot2	21
5. Paquetes útiles	26

En general, para importar y manipular shapefiles (ESRI), geojsons, geodatabases, etc con R podemos utilizar el paquete más moderno llamado *{sf}*.

La forma clásica para importar, exportar y manipular datos vectoriales fue hace unos años vía el uso de *{sp}*. No obstante, hoy en día ya está consolidado el paquete *{sf}* (Simple Features) que une también los paquetes *{rgdal}* y *{maptools}*. No obstante, todavía es necesario conocer las formas de trabajo anteriores debido a que ciertas funciones y librerías aún no han cambiado al nuevo formato de *sf*. En el futuro, *sf* reemplazará las tres librerías mencionadas. ¿Qué ventajas tiene *sf*? 1) Simple Features es una forma estándar (ISO 19125-1:2004) que une datos espaciales con atributos no espaciales 2) Una única clase para todos los tipos de datos vectoriales, 3) La estructura es de tipo data.frame 4) Se instala más fácilmente 5) Es más rápido y es compatible con *{tidyverse}*.

1. El paquete sf

El paquete **sf** trabaja con una forma de escribir código más intuitiva, utilizando estructuras de datos nativas de R (*matrix*, *vector*, *list*, etc.). Algunas características de su sintaxis:

- Todos los objetos son **data.frame** o **tibble** con las coordenadas espaciales que le corresponden en una columna. Por tanto, el acceso a las variables es tan simple como si fuese una tabla normal que lleva una columna con geometría.
- Todas las funciones empiezan por **st_**, haciendo referencia al carácter espacial de su aplicación.
- Éstas pueden ser combinadas con el operador *pipe* (`%>%`), que facilita la aplicación de una secuencia de funciones sobre un conjunto de datos.
- Las funciones también son fácilmente combinables con funciones del paquete **tidyverse**.
- Se puede encontrar una lista de funciones equivalentes entre **sp** y **sf** en (<https://github.com/r-spatial/sf/wiki/migrating>).

Crear un objeto espacial:

```
datos <- data.frame(lon = c(-3.703889, -8.533333, -5.9925, 2.166667,
-3.8, -1.644167, -0.9), lat = c(40.4125, 42.883333, 37.3925,
41.4, 43.466667, 42.818333, 41.65), nombre = c("Madrid",
"Santiago", "Sevilla", "Barcelona", "Santander", "Pamplona",
"Zaragoza"))
```

`sf` permite consultar el tipo de geometría con `st_geometry()`. La proyección de nuestros puntos podemos definir con el código EPSG (<https://epsg.io>) o el *proj4string*.

```
library(sf)
points_sf <- st_as_sf(datos, coords = c("lon", "lat"), crs = 4326)

class(points_sf) #clase del objeto
```

```
## [1] "sf"           "data.frame"
st_geometry(points_sf) #tipo de geometría

## Geometry set for 7 features
## Geometry type: POINT
## Dimension: XY
## Bounding box: xmin: -8.533333 ymin: 37.3925 xmax: 2.166667 ymax: 43.46667
## Geodetic CRS: WGS 84
## First 5 geometries:
```

1.1. Lectura

Podemos importar un *shapefile* de *ESRI* o cualquier otro archivo de datos vectoriales con la función `st_read()`. El objeto cargado en memoria tendrá una estructura de datos en formato tabla con una columna indicando el tipo de geometría espacial. Así que es posible trabajar con él como si fuese un `data.frame`, pero con la ventaja de tener toda la información espacial necesaria.

```
paises <- st_read("./data/world_countries.shp")

## Reading layer `world_countries' from data source
##   `D:\OneDriveUSC\OneDrive - Universidade de Santiago de Compostela\Escritorio\GIS con R_Madrid\01_v...
##   using driver `ESRI Shapefile'
## Simple feature collection with 255 features and 94 fields
## Geometry type: MULTIPOLYGON
## Dimension: XY
## Bounding box: xmin: -180 ymin: -90 xmax: 180 ymax: 83.6341
## Geodetic CRS: WGS 84

ciudades <- st_read("./data/World_Cities.shp")

## Reading layer `World_Cities' from data source
##   `D:\OneDriveUSC\OneDrive - Universidade de Santiago de Compostela\Escritorio\GIS con R_Madrid\01_v...
##   using driver `ESRI Shapefile'
## Simple feature collection with 2540 features and 13 fields
## Geometry type: POINT
## Dimension: XY
## Bounding box: xmin: -176.1516 ymin: -54.792 xmax: 179.2219 ymax: 78.2
## Geodetic CRS: WGS 84

paises[, 1:3]

## Simple feature collection with 255 features and 3 fields
```

```

## Geometry type: MULTIPOLYGON
## Dimension: XY
## Bounding box: xmin: -180 ymin: -90 xmax: 180 ymax: 83.6341
## Geodetic CRS: WGS 84
## First 10 features:
##   featurecla scalerank labelrank           geometry
## 1 Admin-0 country      5          2 MULTIPOLYGON (((117.7036 4....
## 2 Admin-0 country      5          3 MULTIPOLYGON (((117.7036 4....
## 3 Admin-0 country      6          2 MULTIPOLYGON (((-69.51009 -...
## 4 Admin-0 country      0          3 MULTIPOLYGON (((-69.51009 -...
## 5 Admin-0 country      0          2 MULTIPOLYGON (((-69.51009 -...
## 6 Admin-0 country      0          2 MULTIPOLYGON (((-67.28475 -...
## 7 Admin-0 country      3          3 MULTIPOLYGON (((33.78094 34...
## 8 Admin-0 country      6          5 MULTIPOLYGON (((33.78183 34...
## 9 Admin-0 country      0          2 MULTIPOLYGON (((77.80035 35...
## 10 Admin-0 country     0          2 MULTIPOLYGON (((78.91595 33...

ciudades[, 1:3]

## Simple feature collection with 2540 features and 3 fields
## Geometry type: POINT
## Dimension: XY
## Bounding box: xmin: -176.1516 ymin: -54.792 xmax: 179.2219 ymax: 78.2
## Geodetic CRS: WGS 84
## First 10 features:
##   FID ObjectID CITY_NAME           geometry
## 1 2001    2000 Kamphaeng Phet POINT (99.529 16.473)
## 2 2002    2324 Cabinda    POINT (12.19 -5.556996)
## 3 2003    2001 Phichit     POINT (100.349 16.439)
## 4 2004    2325 Onjiva      POINT (15.7833 -17.05)
## 5 2005    2432 Kimberley   POINT (24.83 -28.66)
## 6 2006    2108 Long Xuyen   POINT (105.438 10.382)
## 7 2007    2216 Krasnoyarsk POINT (92.83334 56.01667)
## 8 2008    2002 Nakhon Sawan POINT (100.134 15.701)
## 9 2009    2433 Hlotse      POINT (28.056 -28.878)
## 10 2010   2326 Benguela    POINT (13.406 -12.577)

```

Los metadatos de los objetos `sf` nos resumen toda la información relevante del objeto espacial. En la cabecera veremos el tipo de clase (*simple feature collection*), el número de filas (*features*), el número de columnas (*fields*), el tipo de geometría (*geometry type*), las dimensiones (*XY*), la extensión (*bbox*) y la proyección (*epsg, proj4string*). Cuando imprimimos un objeto `sf`, R también resume los atributos.

filas columnas

##	Simple feature collection with 255 features and 3 fields		
##	geometry type: MULTIPOLYGON		
##	dimension: XY		
##	bbox: xmin: -180 ymin: -90 xmax: 180 ymax: 83.6341		
##	epsg (SRID): 4326		
##	proj4string: +proj=longlat +datum=WGS84 +no_defs		
##	First 10 features:		
##	featurecla	scalerank	labelrank
## 1	Admin-0 country	5	2 MULTIPOLYGON (((117.7036 4...
## 2	Admin-0 country	5	3 MULTIPOLYGON (((117.7036 4...
## 3	Admin-0 country	6	2 MULTIPOLYGON ((((-69.51009 -...
## 4	Admin-0 country	0	3 MULTIPOLYGON ((((-69.51009 -...
## 5	Admin-0 country	0	2 MULTIPOLYGON ((((-69.51009 -...
## 6	Admin-0 country	0	2 MULTIPOLYGON ((((-67.28475 -...

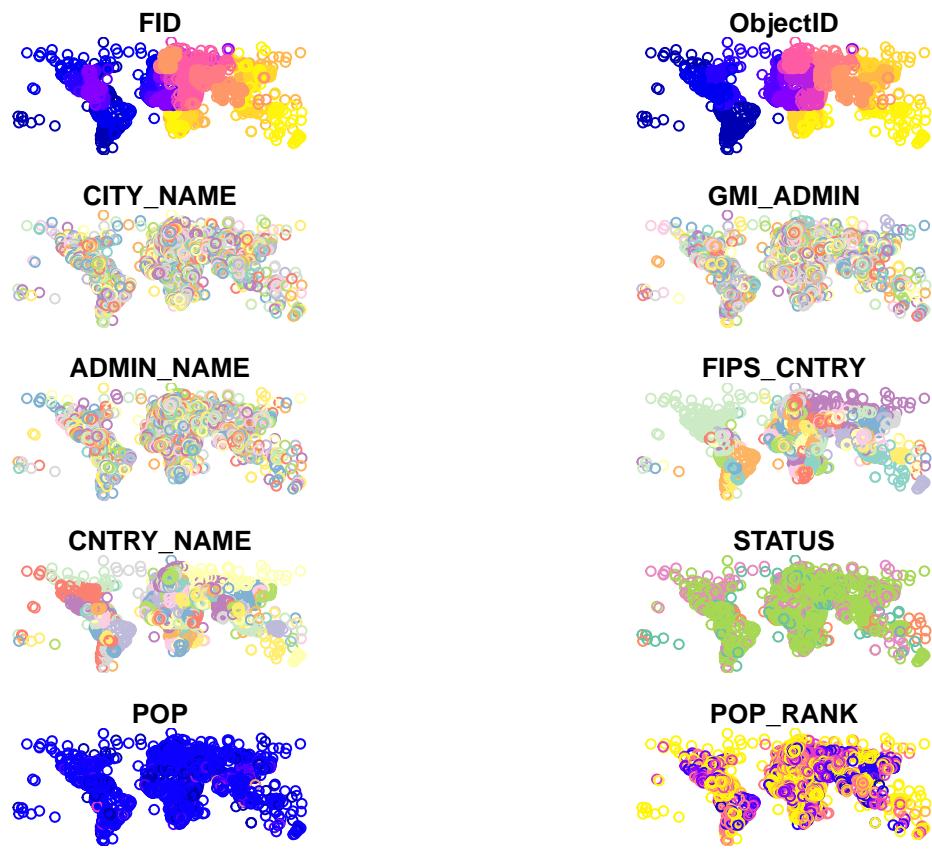
En el caso de una *geodatabase* es necesario indicar el nombre del *layer* como argumento. Por defecto, importa el primer *layer* y da aviso en caso de que exista más de uno. La función `st_layers()` nos permite conocer los *layers* de una *geodatabase*.

1.2. Visualizar

Para cartografiar podemos usar directamente la función `plot()`. Más adelante veremos cómo usar el paquete `{ggplot2}`.

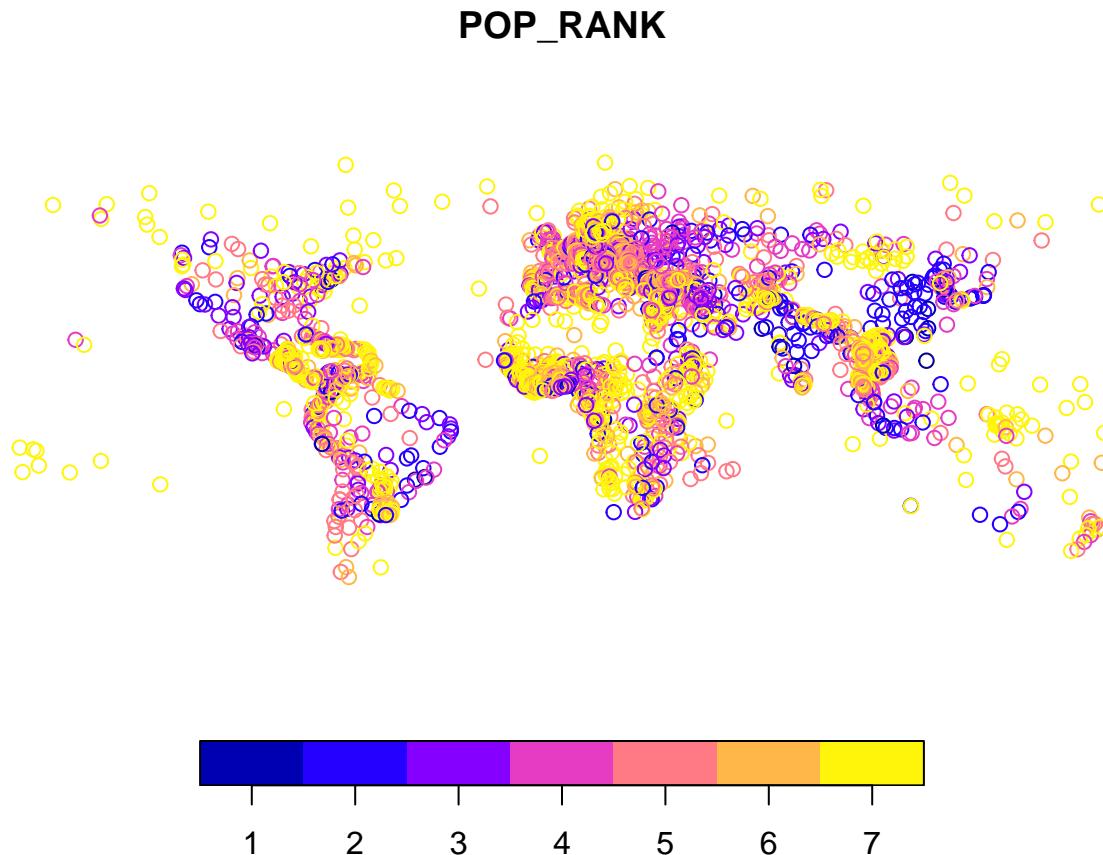
```
plot(ciudades)
```

```
## Warning: plotting the first 10 out of 13 attributes; use max.plot = 13 to plot
## all
```



Si queremos representar solo una variable, hay que definirla.

```
plot(ciudades["POP_RANK"])
```



1.3. Proyectar y coordenadas

Las funciones fundamentales para proyectar, extraer las coordenadas o extraer el sistema de coordenadas son: `st_coordinates()`, `st_crs()` y `st_transform()`. Para asignar una proyección es suficiente con indicar el código *EPSG*.

```
# extraer coordenadas
st_coordinates(ciudades) %>%
  head()

##          X         Y
## 1 99.5290 16.472997
## 2 12.1900 -5.556996
## 3 100.3490 16.439004
## 4 15.7833 -17.049996
## 5 24.8300 -28.659996
## 6 105.4380 10.382004

# extraer CRS
st_crs(ciudades)

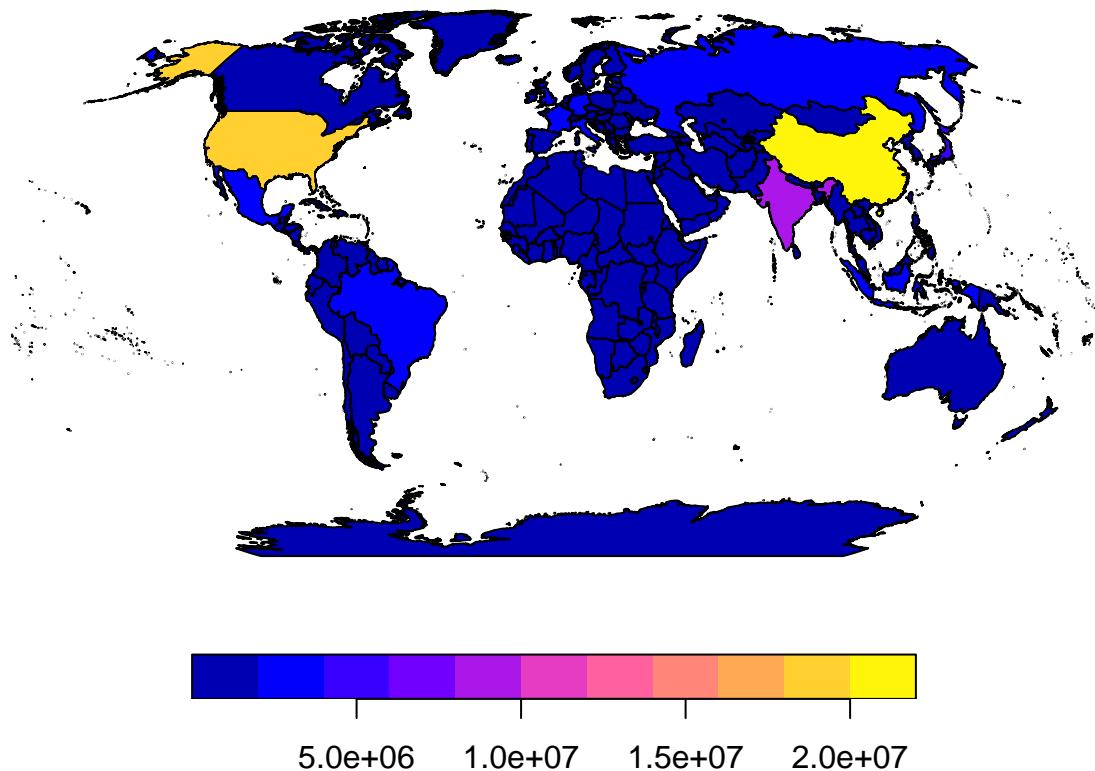
## Coordinate Reference System:
##   User input: WGS 84
##   wkt:
##   GEOGCRS["WGS 84",
##             DATUM["World Geodetic System 1984",
##                   ELLIPSOID["WGS 84",6378137,298.257223563,
##                             LENGTHUNIT["metre",1]]],
```

```

##     PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["latitude",north,
##               ORDER[1],
##               ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["longitude",east,
##               ORDER[2],
##               ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4326]

# proyectar
paises_rob <- st_transform(paises, "ESRI:54030") # Robinson EPSG
plot(paises_rob["gdp_md_est"])

```

gdp_md_est

1.4. Manipulación de los atributos

Como `sf` interpreta los objetos espaciales como un `data.frame`, el acceso a las variables se hace exactamente igual.

1.4.1. Manipular columnas y seleccionar

La manipulación de los atributos de la clase `sf` es igual a la que conocemos con la colección de paquetes `tidyverse`. Una lista de funciones compatibles encontramos en <https://r-spatial.github.io/sf/reference/tidye.html> (no usamos el sufijo `.sf`).

En el siguiente ejemplo modificamos los atributos (variables) del objeto `paises` para crear una columna con el área, otra con la población y otra con la densidad de población. La función `select()` nos permite

seleccionar y extraer las variables que queramos y, en combinación con `slice()`, podemos extraer también filas (unidades espaciales) concretas. Cuando usamos `select()` para seleccionar columnas, la columna que contiene la geometría se mantiene sin necesidad de especificarla.

```
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.1 --
## v ggplot2 3.3.5.9000     v purrr    0.3.4
## v tibble   3.1.6        v dplyr    1.0.7
## v tidyverse 1.2.0       v stringr   1.4.0
## v readr    2.1.2        v forcats  0.5.1

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()

library(units)

## udunits database from D:/OneDriveUSC/OneDrive - Universidade de Santiago de Compostela/Documentos/R/
paises2 <- mutate(paises, area = st_area(paises) %>%
  set_units(km2), pop_est = parse_number(pop_est), densidad = pop_est/area)

dplyr::select(paises2, name, area:densidad)

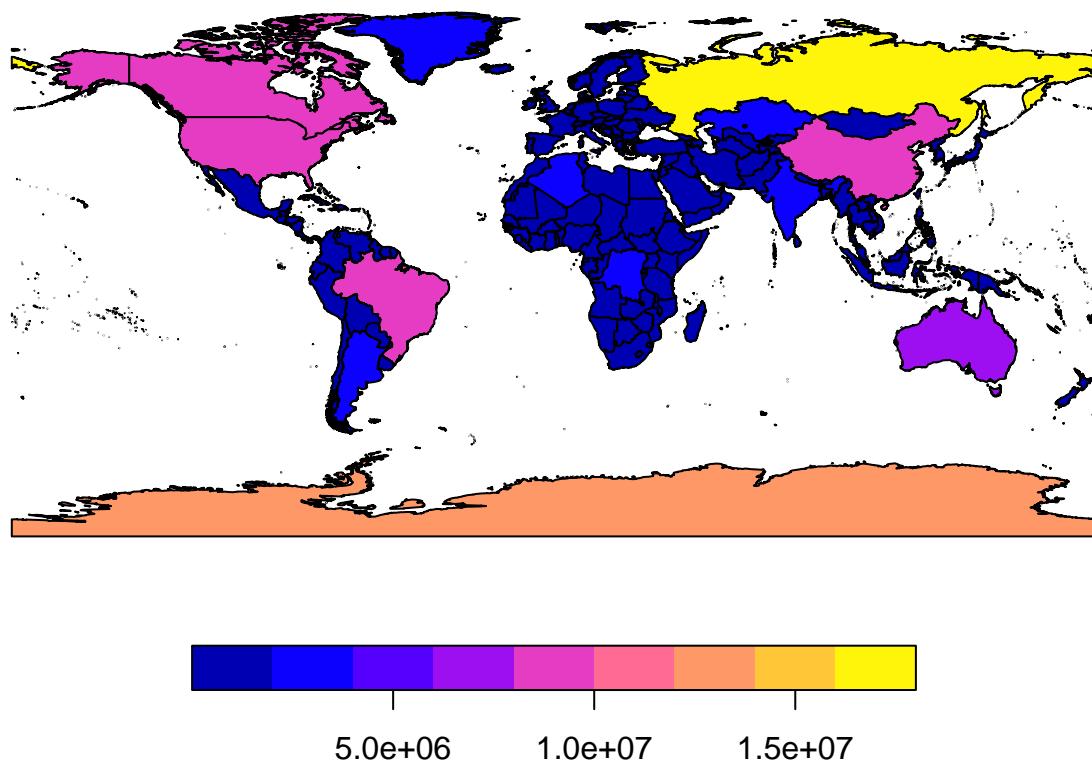
## Simple feature collection with 255 features and 3 fields
## Geometry type: MULTIPOLYGON
## Dimension:      XY
## Bounding box:  xmin: -180 ymin: -90 xmax: 180 ymax: 83.6341
## Geodetic CRS:  WGS 84
## First 10 features:
##           name          area        densidad
## 1  Indonesia 1888148.2872 [km2] 138.00862 [1/km2]
## 2  Malaysia  329335.2100 [km2]  95.28891 [1/km2]
## 3    Chile    736362.4129 [km2]  24.15830 [1/km2]
## 4  Bolivia   1090457.3131 [km2] 10.21428 [1/km2]
## 5    Peru    1295127.0777 [km2] 23.96418 [1/km2]
## 6 Argentina  2784527.7479 [km2] 15.90693 [1/km2]
## 7 Dhekelia     127.4314 [km2] 61.60179 [1/km2]
## 8    Cyprus    5395.5059 [km2] 226.40120 [1/km2]
## 9    India    3159027.2623 [km2] 405.80084 [1/km2]
## 10   China    9372788.7404 [km2] 147.16034 [1/km2]
##           geometry
## 1 MULTIPOLYGON (((117.7036 4...
## 2 MULTIPOLYGON (((117.7036 4...
## 3 MULTIPOLYGON (((-69.51009 -...
## 4 MULTIPOLYGON (((-69.51009 -...
## 5 MULTIPOLYGON (((-69.51009 -...
## 6 MULTIPOLYGON (((-67.28475 -...
## 7 MULTIPOLYGON (((33.78094 34...
## 8 MULTIPOLYGON (((33.78183 34...
## 9 MULTIPOLYGON (((77.80035 35...
## 10 MULTIPOLYGON (((78.91595 33...

dplyr::select(paises2, name, area:densidad) %>%
  slice(5)
```

```
## Simple feature collection with 1 feature and 3 fields
## Geometry type: MULTIPOLYGON
## Dimension: XY
## Bounding box: xmin: -81.33756 ymin: -18.33775 xmax: -68.68425 ymax: -0.02909271
## Geodetic CRS: WGS 84
##   name      area      densidad      geometry
## 1 Peru 1295127 [km2] 23.96418 [1/km2] MULTIPOLYGON ((((-69.51009 -...
```

```
plot(paises2[["area"]])
```

area [km2]

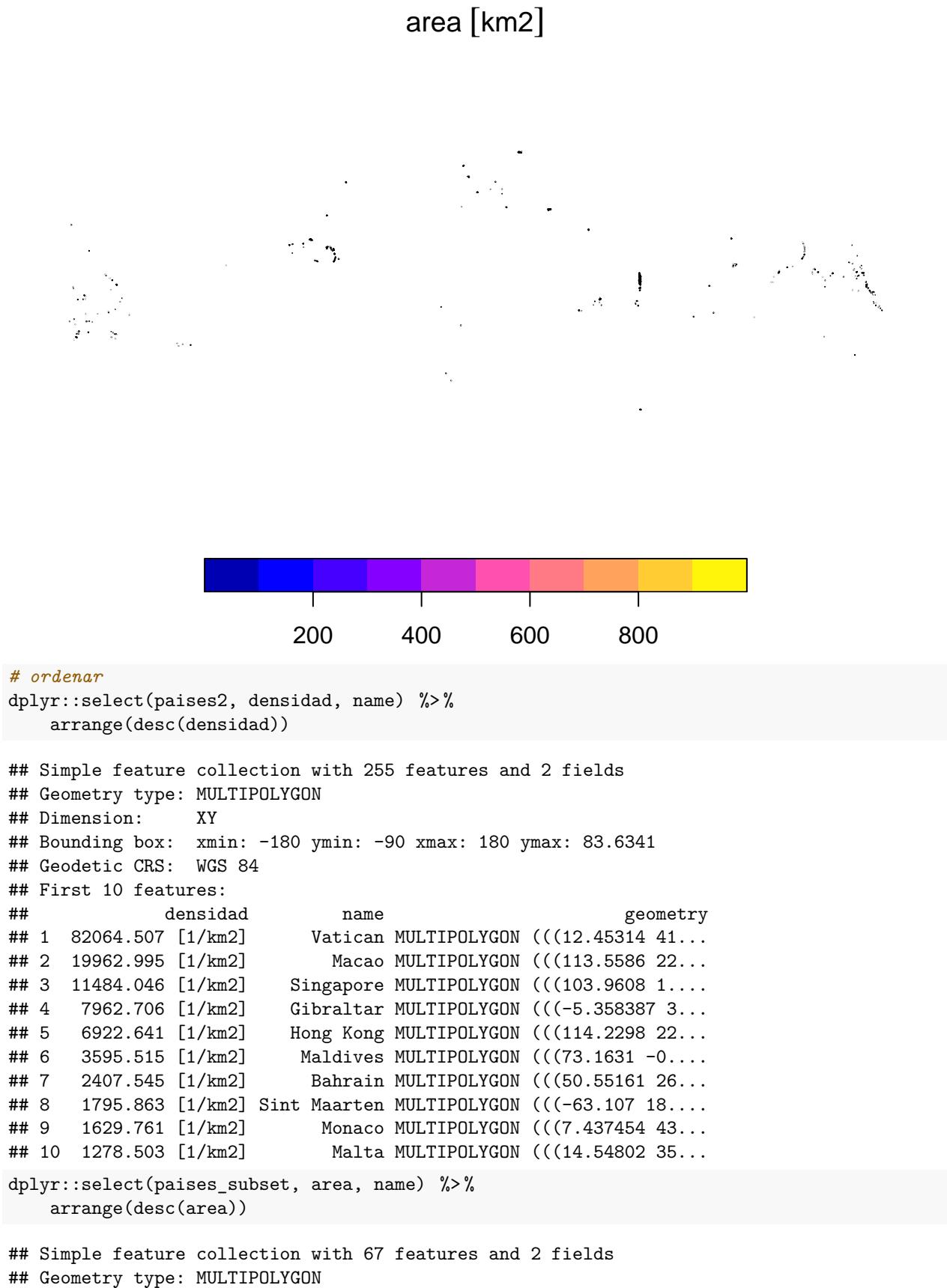


1.4.2. Filtrar y ordenar

La separación de elementos se realiza con la función `filter()`, mediante la que podemos hacer una (o varias) selección(es) por atributos.

```
# filtrar
paises_subset <- filter(paises2, area < set_units(1000, km2))

plot(paises_subset[["area"]])
```



```

## Dimension: XY
## Bounding box: xmin: -178.1857 ymin: -53.19256 xmax: 179.9067 ymax: 60.48078
## Geodetic CRS: WGS 84
## First 10 features:
##           area      name          geometry
## 1  954.0956 [km2] Kiribati MULTIPOLYGON (((173.0383 1...
## 2  936.7706 [km2] Åland MULTIPOLYGON (((20.2776 60....
## 3  732.8966 [km2] Dominica MULTIPOLYGON (((-61.36286 1...
## 4  636.5617 [km2] Micronesia MULTIPOLYGON (((163.0261 5...
## 5  607.3785 [km2] Saint Lucia MULTIPOLYGON (((-60.88679 1...
## 6  604.8635 [km2] Tonga MULTIPOLYGON (((-173.9564 -...
## 7  586.0502 [km2] Bahrain MULTIPOLYGON (((50.55161 26...
## 8  581.2906 [km2] N. Mariana Is. MULTIPOLYGON (((145.2057 14...
## 9  572.3209 [km2] Isle of Man MULTIPOLYGON (((-4.612131 5...
## 10 566.3271 [km2] Guam MULTIPOLYGON (((144.8864 13...

```

1.4.3. Agrupar y disolver

En algunos objetos espaciales, por ejemplo los formados por múltiples polígonos, se permiten operaciones adicionales sobre los atributos a través de la función `group_by()`, equivalente a disolver polígonos según una variable. En este ejemplo, las funciones `st_area()` y `set_units()` sirven para calcular el área en las unidades definidas, respectivamente. En secciones sucesivas se explican de manera más detallada.

```

# s2 es geometría esférica
sf_use_s2(FALSE)

```

```

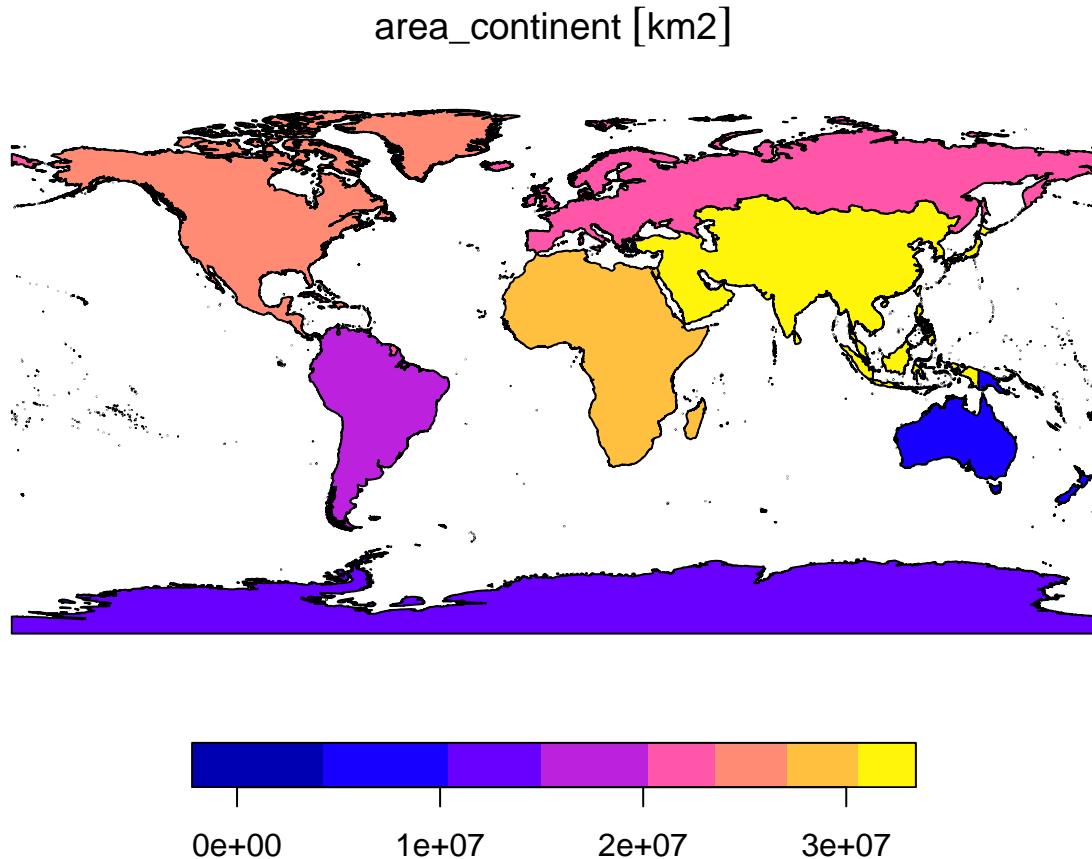
## Spherical geometry (s2) switched off
# disolver según continentes
paises_group <- group_by(paises2, continent) %>%
  summarise(area_continente = sum(area))

## although coordinates are longitude/latitude, st_union assumes that they are planar
## although coordinates are longitude/latitude, st_union assumes that they are planar
## although coordinates are longitude/latitude, st_union assumes that they are planar
## although coordinates are longitude/latitude, st_union assumes that they are planar
## although coordinates are longitude/latitude, st_union assumes that they are planar
## although coordinates are longitude/latitude, st_union assumes that they are planar
## although coordinates are longitude/latitude, st_union assumes that they are planar
## although coordinates are longitude/latitude, st_union assumes that they are planar
## although coordinates are longitude/latitude, st_union assumes that they are planar
## although coordinates are longitude/latitude, st_union assumes that they are planar
head(paises_group)

## Simple feature collection with 6 features and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension: XY
## Bounding box: xmin: -180 ymin: -90 xmax: 180 ymax: 83.6341
## Geodetic CRS: WGS 84
## # A tibble: 6 x 3
##   continent    area_continente          geometry
##   <chr>        [km2]            <MULTIPOLYGON [°]>
## 1 Africa       29987031. (((37.86378 -46.94085, 37.83644 -46.95867, 37.80...
## 2 Antarctica  12258325. ((((-51.73064 -82.06256, -51.2469 -82.01067, -50.~...
## 3 Asia         31201575. (((115.4888 -8.740655, 115.4746 -8.736586, 115.4~...
## 4 Europe       22864962. (((3.361176 -54.45843, 3.357921 -54.45721, 3.354~...
## 5 North America 24192164. ((((-87.99226 15.78461, -87.93391 15.81879, -87.9~...
## 6 Oceania      8529737. ((((-176.2212 -44.32399, -176.2275 -44.31618, -17...

```

```
# Lo que hicimos es disolver los polígonos por continente
plot(paises_group["area_continent"])
```



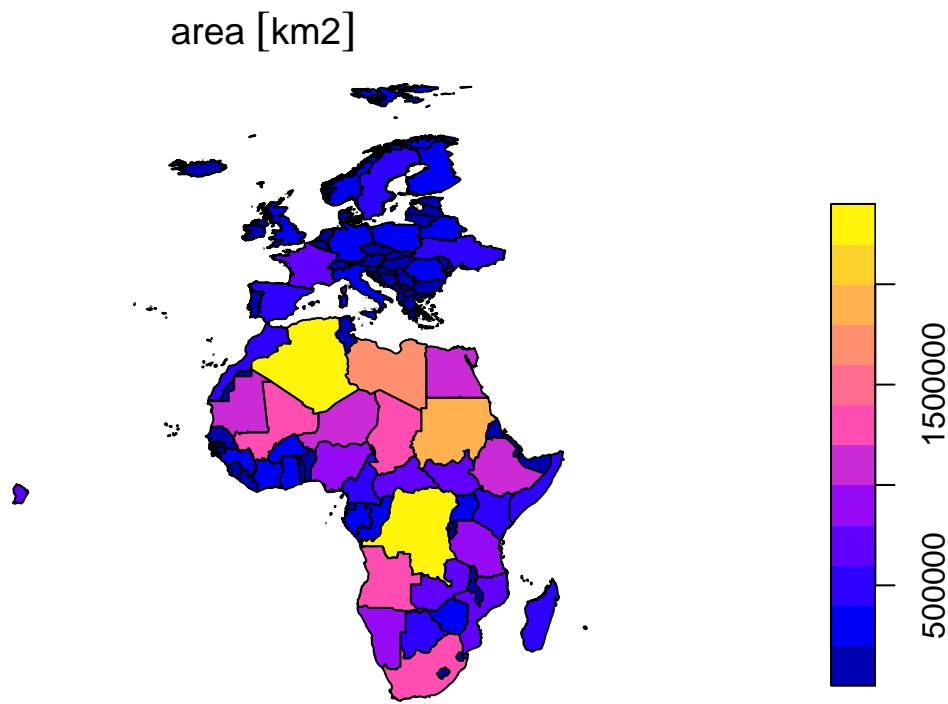
1.4.4. Unión (Append)

La disolución (*dissolve*), a diferencia de la unión (*append*), crea un nuevo y único polígono a partir de todos los polígonos que lo componen. La función `summarise()` define qué variables y con qué datos agregados se obtendrán como resultado.

```
paises_Eu <- filter(paises2, continent == "Europe", name != "Russia")
paises_Afr <- filter(paises2, continent == "Africa")
plot(paises_Eu["area"])
```



```
EuroAsia <- bind_rows(paises_Eu, paises_Afr)  
plot(EuroAsia[\"area\"])
```



También se puede usar la combinación entre la función `group_by()` y `summarise()` sin que esta última tenga argumentos, lo que provocaría la disolución sin la creación de nuevas variables.

Si queremos disolver todos los polígonos de nuestro objeto espacial, podemos usar la función `st_union()`. No obstante, esta disolución provoca que perdamos el formato `data.frame` y pasemos a una clase `sfc`, que contiene únicamente la geometría.

1.4.5. Unir tablas (Join)

Unir un objeto `sf` con otra tabla con nuevos atributos es fácil. Se usan las mismas funciones que aplicamos en el capítulo sobre `tidyverse`.

```
df <- data.frame(name = paises$name, var_dummy = rnorm(255))
head(df)

##          name    var_dummy
## 1 Indonesia -0.68774399
## 2 Malaysia -0.00453654
## 3 Chile     0.13517151
## 4 Bolivia   -2.73335374
## 5 Peru      1.50713364
## 6 Argentina -0.49077956

paises_df <- left_join(paises, df)

## Joining, by = "name"
paises_df <- left_join(paises, df, by = "name")
```

```
names(paises_df) [90:96]
```

```
## [1] "name_ru"    "name_sv"    "name_tr"    "name_vi"    "name_zh"    "var_dummy"
## [7] "geometry"
```

1.5. Funciones de análisis espacial

1.5.1. Intersección

Unión de atributos (*join*) con `st_intersection()`:

```
int <- st_intersection(ciudades, paises)
```

```
## although coordinates are longitude/latitude, st_intersection assumes that they are planar
## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries
head(int[, 1:5])
```

```
## Simple feature collection with 6 features and 5 fields
## Geometry type: POINT
## Dimension:      XY
## Bounding box:  xmin: 100.352 ymin: -7.796997 xmax: 110.426 ymax: -0.02300395
## Geodetic CRS:  WGS 84
##   FID ObjectID CITY_NAME GMI_ADMIN      ADMIN_NAME
## 162 2162     2458 Padang IDN-SBA Sumatera Barat
## 168 2168     2459 Bandung IDN-JBA Jawa Barat
## 173 2173     2460 Semarang IDN-JTN Jawa Tengah
## 179 2179     2461 Yogyakarta IDN-YGY Yogyakarta
## 238 2238     2467 Pontianak IDN-KBA Kalimantan Barat
## 240 2240     2468 Jambi IDN-JAM Jambi
##   geometry
## 162     POINT (100.352 -0.954)
## 168     POINT (107.607 -6.912)
## 173     POINT (110.426 -6.971004)
## 179     POINT (110.369 -7.796997)
## 238 POINT (109.339 -0.02300395)
## 240 POINT (103.6167 -1.600002)
```

`st_intersects()` devuelve un listado. Lista con los índices de los países conteniendo cada uno de ellos los índices de las ciudades que intersectan con ellos.

```
int_spr <- st_intersects(paises, ciudades)
```

```
## although coordinates are longitude/latitude, st_intersects assumes that they are planar
int_spr
```

```
## Sparse geometry binary predicate list of length 255, where the
## predicate was `intersects'
## first 10 elements:
## 1: 162, 168, 173, 179, 238, 240, 245, 256, 263, 271, ...
## 2: 361, 367, 374, 378, 384, 391, 394, 399, 405, 411, ...
## 3: 591, 602, 608, 612, 615, 618, 624, 627, 630, 634, ...
## 4: 571, 590, 673, 821, 825, 828, 831, 836, 840
## 5: 560, 562, 564, 566, 568, 686, 694, 698, 701, 704, ...
## 6: 558, 594, 596, 605, 609, 611, 614, 617, 621, 628, ...
```

```
## 7: (empty)
## 8: 1942, 1950
## 9: 69, 77, 84, 90, 96, 101, 104, 2172, 2175, 2181, ...
## 10: 53, 95, 111, 120, 124, 128, 133, 138, 144, 150, ...
```

Para unir diferentes objetos espaciales manteniendo los polígonos originales (*append*) usamos simplemente la función `rbind()`. Si quisieramos unir diferentes geometrías ó atributos, es necesario usar la función `st_union()`, lo que provocaría en caso de distintos tipos de geometrías una *GEOMETRYCOLLECTION*.

En el siguiente ejemplo generamos un objeto con todos los países de Europa excepto Rusia y otro con todos los países de África.

```
# Creamos un polígono con la dimensión de Francia
```

```
spain <- filter(paises, name == "Spain")
```

```
poly_sp <- st_bbox(spain) %>%
  st_as_sfc() %>%
  st_transform(4326)
```

```
# Calculamos la diferencia entre las dos geometrías
```

```
diff_pol <- st_difference(paises, poly_sp)
```

```
## although coordinates are longitude/latitude, st_difference assumes that they are planar
## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries
diff_pol[, 1:5]
```

```
## Simple feature collection with 252 features and 5 fields
## Geometry type: GEOMETRY
## Dimension: XY
## Bounding box: xmin: -180 ymin: -90 xmax: 180 ymax: 83.6341
## Geodetic CRS: WGS 84
## First 10 features:
##           featurecla scalerank labelrank    sovereignt sov_a3
## 1 Admin-0 country      5        2 Indonesia   IDN
## 2 Admin-0 country      5        3 Malaysia    MYS
## 3 Admin-0 country      6        2 Chile       CHL
## 4 Admin-0 country      0        3 Bolivia     BOL
## 5 Admin-0 country      0        2 Peru        PER
## 6 Admin-0 country      0        2 Argentina   ARG
## 7 Admin-0 country      3        3 United Kingdom GB1
## 8 Admin-0 country      6        5 Cyprus      CYP
## 9 Admin-0 country      0        2 India       IND
## 10 Admin-0 country     0        2 China       CH1
##           geometry
## 1 MULTIPOLYGON (((117.7036 4.....
## 2 MULTIPOLYGON (((117.6971 4.....
## 3 MULTIPOLYGON ((((-69.50611 -...
## 4 POLYGON (((-69.51009 -17.505...
## 5 MULTIPOLYGON ((((-69.63832 -...
## 6 MULTIPOLYGON ((((-67.25133 -...
## 7 POLYGON ((33.76043 34.97968...
## 8 MULTIPOLYGON (((33.78094 34...
## 9 MULTIPOLYGON (((77.81533 35...
```

```
## 10 MULTIPOLYGON (((78.82426 33...
```

```
plot(diff_pol["admin"], xlim = c(-20, 10), ylim = c(30, 55))
```

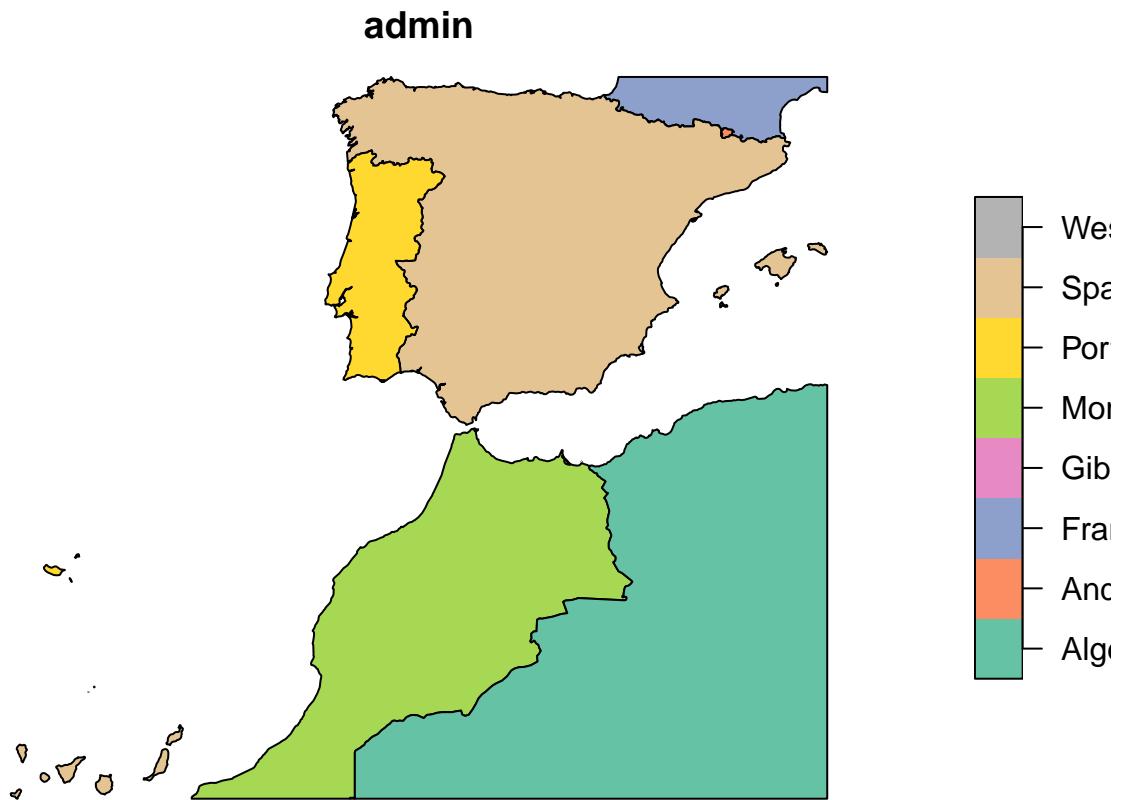


Podemos extraer o recortar la geometría de una capa a partir de la de otra.

```
recort_sp <- st_crop(paises, poly_sp)
```

```
## although coordinates are longitude/latitude, st_intersection assumes that they are planar
## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries
```

```
plot(recort_sp["admin"])
```



1.5.2. Manejo de área, longitud y distancia

Una de las ventajas de trabajar con `sf` es que, en combinación con el paquete `units`, se pueden obtener y manipular las métricas espaciales sin tener que hacer cálculos adicionales. De manera muy intuitiva, la función `st_area()` calcula el área de un polígono o conjunto de polígonos de un objeto espacial en las unidades de la proyección, las que podemos reconvertir con la función `set_units()`:

```
st_area(paises) %>%
  head()

## # Units: [m^2]
## [1] 1.879827e+12 3.278849e+11 7.365927e+11 1.086811e+12 1.289866e+12
## [6] 2.784306e+12

area <- st_area(paises) %>%
  set_units(km2)
head(area)

## # Units: [km2]
## [1] 1879827.4 327884.9 736592.7 1086810.6 1289866.4 2784305.9
```

La distancia entre puntos de un objeto espacial se calcula con `st_distance()`, dando como resultado un objeto de la clase `units` que contiene una matriz de distancias con el mismo número de filas y columnas que el número de puntos y una serie de atributos espaciales que identifican las unidades de medida. Además, podemos calcular la distancia entre un par de coordenadas y todos los puntos de un objeto espacial. Esto puede ser útil, como en el ejemplo, para calcular la distancia (en línea recta) entre un punto concreto y todas las ciudades de nuestro objeto espacial.

```
# matriz de distancia
dist_ciudades <- st_distance(ciudades) %>%
  set_units(km)
dim(dist_ciudades) #matrix

## [1] 2540 2540

# head(dist_ciudades)

# distancia entre un único punto y otros
p <- data.frame(lon = 0, lat = 40) %>%
  st_as_sf(coords = c("lon", "lat")) %>%
  st_set_crs(4326)

p_dist <- st_distance(ciudades, p)
dim(p_dist)

## [1] 2540     1
```

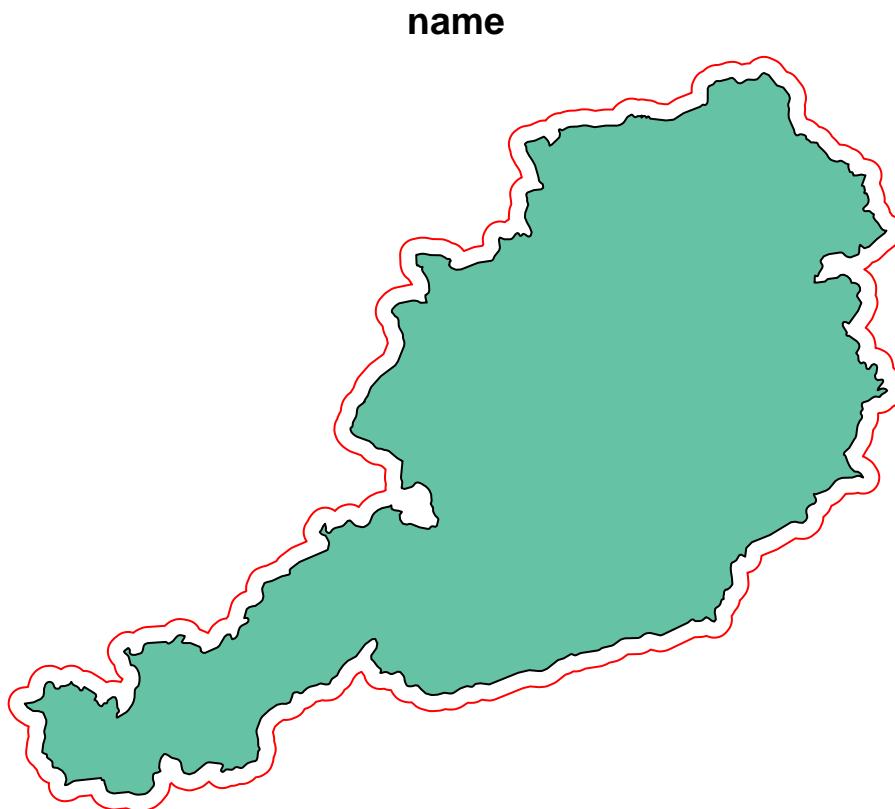
1.5.3. Áreas de influencia (buffer)

También es posible calcular áreas de influencia (*buffers*) alrededor de una geometría espacial (utiliza las unidades del sistema de coordenadas del objeto espacial). Si queremos calcular un área de influencia de 10000 metros alrededor de un polígono concreto del objeto países, usaremos la función `st_buffer()`.

```
# buffer de 10 kilómetros alrededor de Austria
buf_ic <- paises %>%
  dplyr::filter(name == "Austria") %>%
  st_transform(3055) %>%
  st_buffer(10000)

austria <- paises %>%
  dplyr::filter(name == "Austria") %>%
  st_transform(3055)

plot(austria$name, reset = FALSE) # usamos reset para poder añadir el otro mapa encima
plot(buf_ic$name, col = NA, border = "red", add = T) # add=T añade el buffer
```



Cuando se usa un sistema de coordenadas decimales se obtiene un aviso de que el cálculo puede ser menos exacto o correcto. Por eso, proyectamos en este caso a un sistema UTM con la función `st_transform()`. Por defecto, la unidad usada en la distancia aquí también es la del sistema de coordenadas.

1.5.4. Cambiar el tipo de geometría

En objetos vectoriales es muy fácil cambiar entre distintos tipos de geometría (polígono a línea, línea a puntos, etc.) con `st_cast()`. Esta función, combinada con las de métricas espaciales, pueden dar resultados interesantes para el análisis espacial.

```
# Convertimos los polígonos de los países a líneas
pol_to_line <- st_cast(paises, "MULTILINESTRING")

# Perímetro de todos los polígonos
perim <- st_cast(paises, "MULTILINESTRING") %>%
  st_length()
head(perim)

## Units: [m]
## [1] 54827173 7584941 37353542 5901387 8069398 14598644
```

2. Cambiar entre formatos: sp, sf, data.frame

Es posible que nos haga falta convertir en ocasiones entre la clase de `sp`, `sf` ó `data.frame`. Veamos algunos ejemplos: podemos eliminar la geometría original de nuestro objeto `paises`, que es de polígonos, lo cual lo convierte en un simple `data.frame` que ya no podemos tratar como un objeto espacial porque hemos perdido las coordenadas (lon, lat), así que este objeto es irreversible.

```

# eliminar geometría
paises_df <- dplyr::select(paises2, area, name) %>%
  st_set_geometry(NULL)

head(paises_df) #data.frame (es irreversible sin lon, lat)

##           area      name
## 1 1888148.3 [km2] Indonesia
## 2 329335.2 [km2] Malaysia
## 3 736362.4 [km2] Chile
## 4 1090457.3 [km2] Bolivia
## 5 1295127.1 [km2] Peru
## 6 2784527.7 [km2] Argentina

# para exportar debemos eliminar la clase unit
paises <- mutate(paises2, across(c(pop_est, area, densidad),
  as.numeric))

# sf a sp
paises_sp <- as(paises, "Spatial")
class(paises_sp)

## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"

# sp a sf
paises_sf <- st_as_sf(paises_sp)
class(paises_sf)

## [1] "sf"        "data.frame"

```

3. Exportación de datos

En `sf` se utiliza la función `st_write()` y los tipos de formatos posibles son los mismos. En este caso no hace falta definir el `driver` si añadimos la extensión que tendrá el archivo.

```

# Exportación a shapefile de ESRI
st_write(paises_sf, "paises_sf.shp")

# Exportación a GeoJSON
st_write(paises_sf, "paises_sf.geojson")

```

4. Visualización vectorial con ggplot2

Hemos visto las posibilidades y funcionalidad del paquete `ggplot2`. Aquí ampliaremos los aspectos relacionados con la representación cartográfica utilizando algunas de esas funciones y otras nuevas. La combinación de la gramática de `ggplot2` con la gestión de la información espacial que hace `sf`, produce un lenguaje de visualización muy intuitivo y escalable (podemos añadir funciones de representación sin modificar lo anterior). Esta interacción se lleva a cabo principalmente con la función `geom_sf()`, que es aplicable a todos los tipos de geometrías en el paquete de `ggplot2`.

Para ilustrarlo con un ejemplo sencillo, cargaremos los paquetes `tidyverse` y `sf` y leemos un dataset que contiene información sobre los husos horarios. Además, cargamos otros paquetes que usaremos en esta sección.

Un paquete de gran ayuda es `{rnaturalearth}` (<https://github.com/ropensci/rnaturalearth>) que nos da acceso a datos administrativos del mundo con diferentes resoluciones.

```
install.packages("rnaturalearth")
devtools::install_github("ropensci/rnaturalearthdata") # resolución media
devtools::install_github("ropensci/rnaturalearthhires") # resolución alta

library(rnaturalearth)

# límites administrativos
wld <- ne_countries(scale = 50, returnclass = "sf")

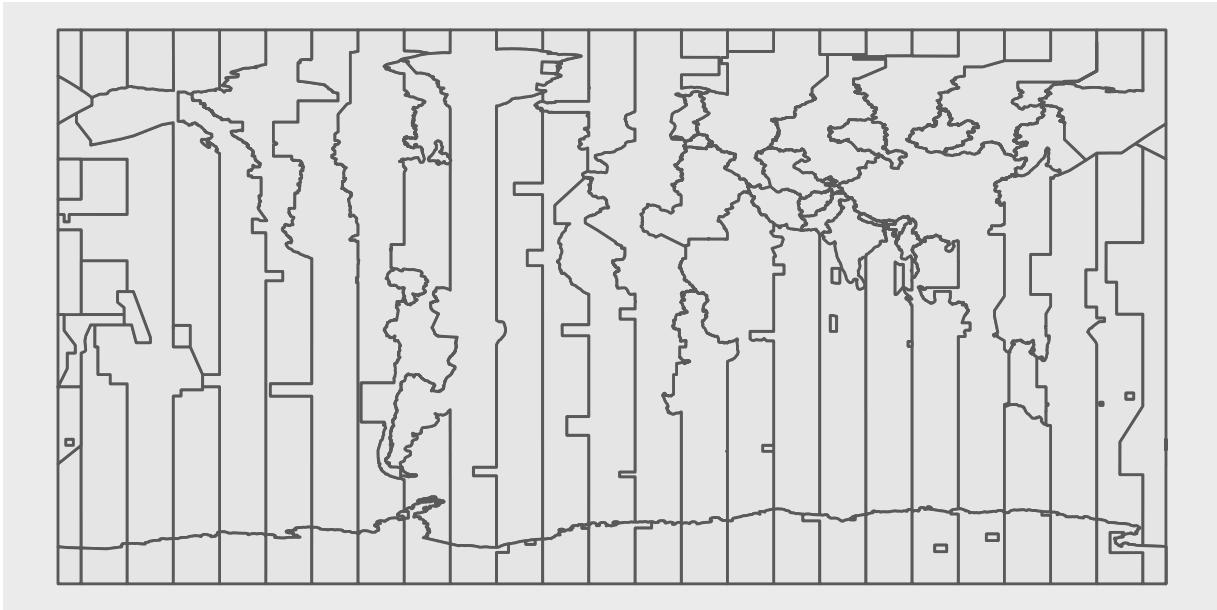
# mapamundi
ggplot(wld) + geom_sf(fill = "grey90", colour = "white") + coord_sf(crs = "ESRI:54009") +
  theme_void()
```



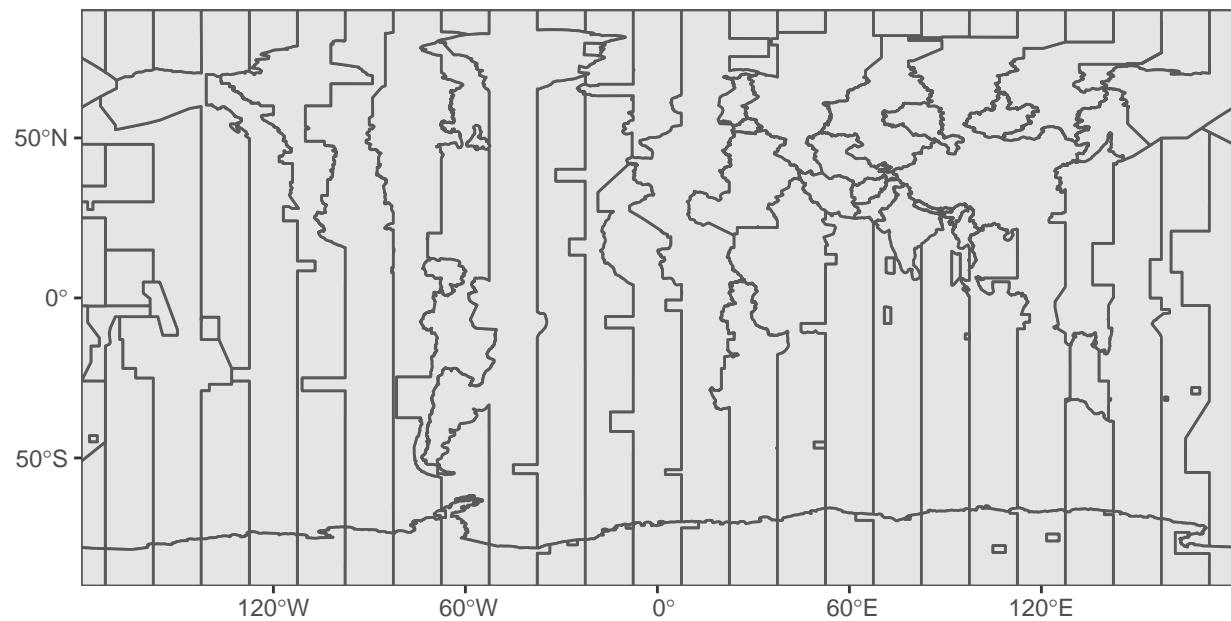
```
# importamos los husos horarios
tz <- st_read("./data/ne_10m_time_zones.shp")

## Reading layer 'ne_10m_time_zones' from data source
##   `D:\OneDriveUSC\OneDrive - Universidade de Santiago de Compostela\Escritorio\GIS con R_Madrid\01_v...
##   using driver 'ESRI Shapefile'
## Simple feature collection with 120 features and 15 fields
## Geometry type: MULTIPOLYGON
## Dimension:      XY
## Bounding box:  xmin: -180 ymin: -90 xmax: 180 ymax: 90.00021
## Geodetic CRS:  WGS 84
```

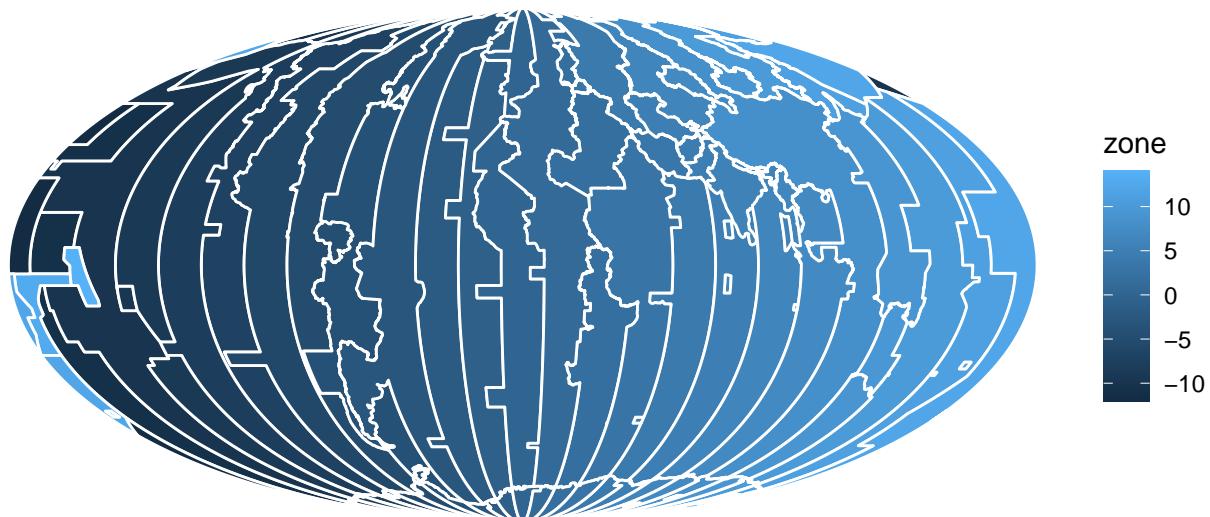
```
# primer mapa  
ggplot(tz) + geom_sf()
```



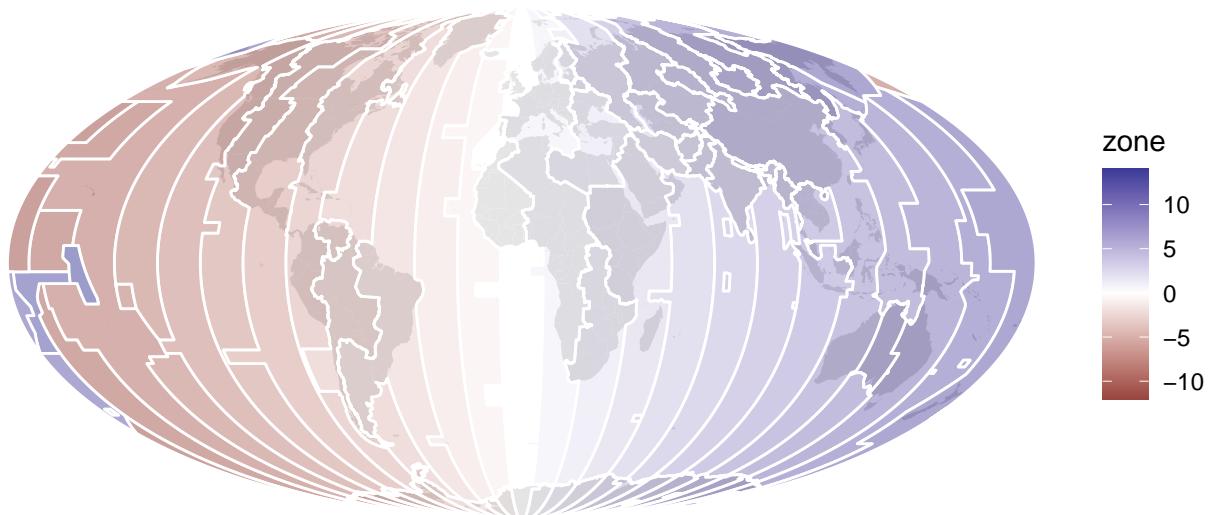
```
# mapamundi sin 'expansión'  
ggplot(tz) + geom_sf() + coord_sf(expand = FALSE)
```



```
# con otra proyección y colorado por el huso
ggplot(tz) + geom_sf(aes(fill = zone), colour = "white") + coord_sf(crs = "ESRI:54009") +
  theme_void()
```



```
# con más ajustes
ggplot() + geom_sf(data = wld, fill = "grey80", colour = NA) +
  geom_sf(data = tz, aes(fill = zone), colour = "white", alpha = 0.5) +
  scale_fill_gradient2() + coord_sf(crs = "ESRI:54009") + theme_void()
```



Recomiendo la instalación del paquete *{mapview}* que ayuda en visualizar información espacial en forma interactiva.

5. Paquetes útiles

- Escala, Norte, y más utilidades. *{ggspatial}* <https://paleolimbot.github.io/ggspatial/>
- Datos de Eurostat. *{eurostat}*
- Datos específicos de España. *{mapSpain}*
- Manejo de datos espacio-temporales. *{stars}*
- Mapas dinámicos. *{leaflet}*
- Alternativa a ggplot2 para mapas. *{tmap}*
- Datos de OSM. *{osmdata}*
- Cartogramas. *{cartogram}*
- Geoestadística. *{geoR}*, *{gstat}*, etc.

Importante también: <https://cran.r-project.org/web/views/Spatial.html>