Lab 2: R for spatial data analysis

Joaquin Cavieres

1. Introduction to the packages sf, tidyverse and starts

1.1. The package sf

It was developed to replace the package **sp**, **rgeos** and some parts of the **rgdal** packages. The main characteristic of the package **sf** is represent modern visions of the open source geospatial sotware stack, and to allow for the integration of R spatial software with the **tidyverse** package. In particular **sp** package provides an interface to several **tidyverse** packages, for example: **ggplot2**, **dplyr** and **tidyr**. Also, we can read and write data by GDAL, make geometrical operations (GEOS) for projected coordinates, or use s2geometry for ellipsoidal coordinates.

When we created an sf object, it contains the following meta-data

- The name of the (active) geometry column, held in attribute `sf_column`
- For each non-geometry variable, the attribute-geometry relationship, held in attribute `agr`

Additionally, the geometry is extracted from the sf object using the function `st_geomtry` and contains the following meta-data:

- The coordinate reference system held in attribute `crs`
- The bounding box held in attribute `bbox`
- The precision held in attribute `precision`
- The number of empty geometries held in attribute `n_empty`

We can acces to those attributes using the functions `st_bbox`, `st_crs`, `st_set_crs`, `st_precision`, and `st_set_precision`.

If we want to create an sf object, from any type of spatial data, we can do:

```r
library(sf)
# Linking to GEOS 3.10.2, GDAL 3.4.3, PROJ 8.2.1; sf_use_s2() is TRUE
p1 <- st_point(c(7.35, 52.42))
p2 <- st_point(c(7.22, 52.18))
p3 <- st_point(c(7.44, 52.19))
sfc <- st_sfc(list(p1, p2, p3), crs = 'OGC:CRS84')
st_sf(elev = c(33.2, 52.1, 81.2),
      marker = c("Id01", "Id02", "Id03"), geom = sfc)
```

```
## Simple feature collection with 3 features and 2 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 7.22 ymin: 52.18 xmax: 7.44 ymax: 52.42
## Geodetic CRS:  WGS 84
##   elev marker               geom
## 1 33.2   Id01 POINT (7.35 52.42)
## 2 52.1   Id02 POINT (7.22 52.18)
## 3 81.2   Id03 POINT (7.44 52.19)
```

where "3 features" indicates the rows (records) and "2 fields" indicates the attribute variables. The column "geom" indicates the geometry and every row of that indicates a single geometry (for example, `POINT (7.22 52.18)`).

That example was created from scratch, but commonly we read spatial data from external sources, for example:

- An external file
- A table (or set of tables) in a database
- A request to a web service
- A dataset held in some form in another R package

1.2.1. Reading external files

We can read external files using the `st_read` function as follow:

```r
file <- system.file("gpkg/nc.gpkg", package = "sf")
nc <- st_read(file)
```

```
## Reading layer ‘nc.gpkg’ from data source
##   ‘C:\Users\joaquin\AppData\Local\Programs\R\R-4.2.2\library\sf\gpkg\nc.gpkg’
##   using driver ‘GPKG’
## Simple feature collection with 100 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## Geodetic CRS:  NAD27
```

The command `st_read` has two arguments:

- The data source name (dsn)
- The layer.

In the example above, the geopackage (GPKG) file contains only a single layer that is being read. If it had contained multiple layers, then the first layer would have been read and a warning would have been emitted. The available layers of a data set can be queried by

```
st_layers(file)
```

```
## Driver: GPKG
## Available layers:
##   layer_name geometry_type features fields crs_name
## 1    nc.gpkg Multi Polygon      100     14    NAD27
```

Others simple charactersitics can be written by using `st_write` function:

```
file = tempfile(fileext = ".gpkg")
st_write(nc, file, layer = "layer_nc")
```

```
## Writing layer ‘layer_nc’ to data source
##   ‘C:\Users\joaquin\AppData\Local\Temp\RtmpYLs3UR\file357842ec4359.gpkg’ using driver ‘GPKG
## Writing 100 features with 14 fields and geometry type Multi Polygon.
```

where the file format (GPKG) is derived from the file name extension. Layers can also be deleted, e.g. from a database, using `st_delete` function.

1.2.2. Subsetting

Frequently we only use some features of an object, for example, a subset of it. Here we can use the same rules that applied for a `data.frame` object, for example if we want to see the first 2-5 records from the column 3-7, we do:

```
nc[2:5, 3:7]
```

```
## Simple feature collection with 4 features and 5 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -81.34754 ymin: 36.07282 xmax: -75.77316 ymax: 36.57286
## Geodetic CRS:  NAD27
##   CNTY_ CNTY_ID        NAME  FIPS FIPSNO                           geom
## 2  1827    1827   Alleghany 37005  37005 MULTIPOLYGON (((-81.23989 3...
## 3  1828    1828       Surry 37171  37171 MULTIPOLYGON (((-80.45634 3...
## 4  1831    1831   Currituck 37053  37053 MULTIPOLYGON (((-76.00897 3...
## 5  1832    1832 Northampton 37131  37131 MULTIPOLYGON (((-77.21767 3...
```

however, it has additional features:

- The `drop` argument is by default `FALSE` meaning that the geometry column is always selected, and an sf object is returned, instead if we select `TRUE` and the geometry column is not selected, it is dropped and a `data.frame` is returned
- Selecting a spatial object (`sf, sfc or sfg`), the first argument leads to selection of the features that spatially intersect with that object.

1.2. The package tidyverse

tidyverse is a collection of different packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures. It contains for example, **ggplot2**, **tidyr**, **readr**, and **dplyr**.

Name	Description	Functions
ggplot2	data visualisation	`ggplot`
dplyr	data manipulation	`mutate, select, filter, arrange`
tidyr	col=var, row=obs, cell=value	`pivot_longer, pivot_wider, separate, extract, unite, complete, fill, replace_na and drop_na`
readr	import data	`read_csv, read_tsv, read_delim, read_fwf, read_table`, and `read_log`

Cuadro 1: Source: Study.com

The package **sf** has tidyverse-style read and write functions, as `read_sf` and `write_sf`, and:

- Return a tibble rather than a `data.frame`
- Don't print any output
- Overwrite existing data by default.

Besides, for `sf` objects, tidyverse allows us to use: `filter, select, group_by, ungroup, mutate, transmute,` etc..

For example.

```
library(tidyverse) |> suppressPackageStartupMessages()
nc |> as_tibble() |> select(BIR74) |> head(3)
```

```
## # A tibble: 3 x 1
##   BIR74
##   <dbl>
## 1  1091
## 2   487
## 3  3188
```

Here the `summarise` function for the `sf` object has two special arguments:

- `do_union` (default TRUE) determines whether grouped geometries are unioned on return, so that they form a valid geometry
- `is_coverage` (default FALSE) in case the geometries grouped form a coverage (do not have overlaps), setting this to TRUE speeds up the unioning

The function `distinct` select distinct records (rows), where `st_equals` is used to evaluate distinctness of geometries.

The function `filter` can be used with the usual predicates; when one wants to use it with a spatial predicate, for instance to select all counties less than 50 km away from Orange County, one could use

```
orange <- nc |> dplyr::filter(NAME == "Orange")
wd <- st_is_within_distance(nc, orange,
                            units::set_units(50, km))
o50 <- nc |> dplyr::filter(lengths(wd) > 0)
nrow(o50)
```

```
## [1] 17
```

1.3. The package stars

The package **sp** has limitations to support raster data, for the same the package **raster** (Hijmans 2022a) has clearly been dominant as the prime package for powerful, flexible and scalable raster analysis. The raster data model of package **raster** (and its successor, **terra** (Hijmans 2022b)) uses a 2D regular raster, or a set of raster layers (a "raster stack"). This aligns with the classical static "GIS view", where the world is modelled as a set of layers, each representing a different theme. However, the data today is dynamic, and perfectly could comes as time series of rasters or raster stacks. The **raster** and **terra** packages allows us an efficient computation when the data sizes are no larger the local storage, but some data sets include satellite imagery, climate model or weather forecasting data, often no longer fit in local storage. For the same we will use the package **starts** which allows us:

- Represent dinamyc raster stacks
- It can be sacalable, beyond of the local disk size
- Provides an integration with raster functions of the GDAL library
- Allow a tight integration with package **sf**
- Follows the **tidyverse** design principles

1.2.1. Reading and writing raster data

Using the package "dplyr" we can aggregate and summaries specifying the following functions: `group_by` (specifies the group membership of records) and `summarise` that computes data (such as `sum` or `mean`) for each of the groups.

```
library(tidyverse)
library(sf)
nc <- read_sf(system.file("gpkg/nc.gpkg", package = "sf"))
# encode quadrant by two logicals:
nc$lng <- st_coordinates(st_centroid(st_geometry(nc)))[,1] > -79
nc$lat <- st_coordinates(st_centroid(st_geometry(nc)))[,2] > 35.5
nc.grp <- aggregate(nc["SID74"], list(nc$lng, nc$lat), sum)
nc.grp["SID74"] |> st_transform('EPSG:32119') |>
plot(graticule = TRUE, axes = TRUE)
```

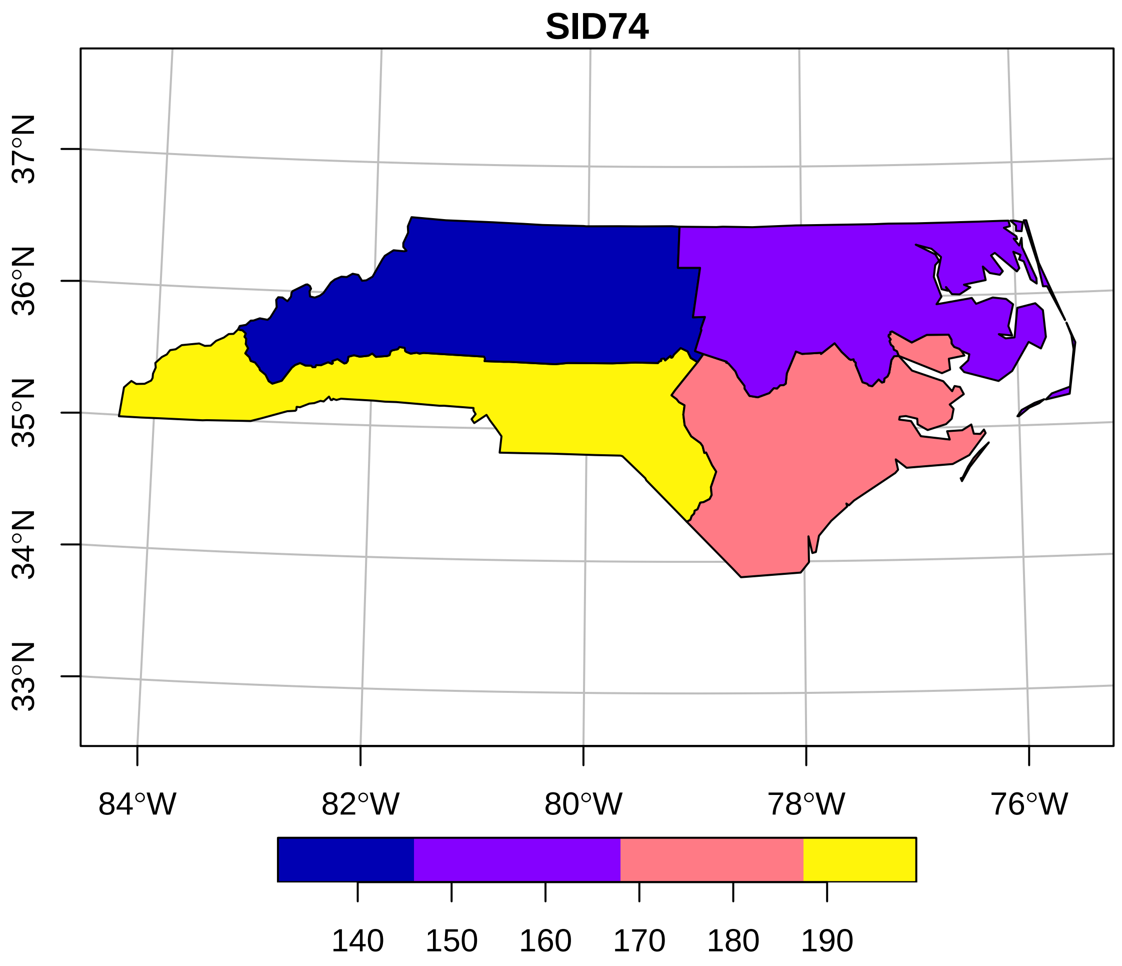

Figura 1: SID74 total incidences aggregated to four areas

The main property of the aggregation is that each record is assigned to a single group, this has the advantage that the sum of the group-wise sums equals the sum of the un-grouped data. The new geometry is the result of the contributing records.

```
nc <- st_transform(nc, 2264)
gr <- st_sf(label = apply(expand.grid(1:10, LETTERS[10:1])[,2:1], 1, paste0, collapse = " "),
            geom = st_make_grid(nc))
plot(st_geometry(nc), reset = FALSE, border = 'grey')
plot(st_geometry(gr), add = TRUE)
```

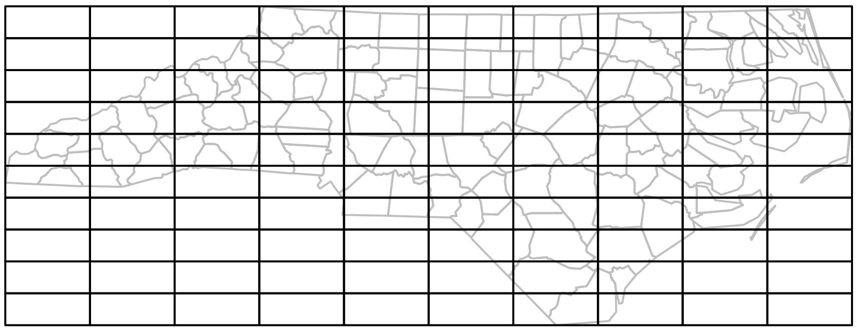

Figura 2: Example target blocks plotted over North Carolina counties

When we need an aggregate for a new area that is not a union of the geometries for a group of records, and we use a spatial predicate then single records may be matched to multiple groups. If we take the rectangles of the previous figure as target areas, and summing for every rectangle the desases cases of the counties intersected with each reactangle, the sum should be much larger:

```
a <- aggregate(nc["SID74"], gr, sum)
c(sid74_sum_counties = sum(nc$SID74),
  sid74_sum_rectangles = sum(a$SID74, na.rm = TRUE))
```

```
##   sid74_sum_counties sid74_sum_rectangles
##                  667                 2621
```

References

Pebesma, E., & Bivand, R. (2023). Spatial Data Science: With Applications in R. CRC Press.

Pebesma, E., (2018). Simple Features for R: Standardized Support for Spatial Vector Data. The R Journal 10 (1): 439–46. https://doi.org/10.32614/RJ-2018-009.