Introduction to GIS

Smart Analytics for Big Data

Iragaël Joly
Grenoble INP - UMR GAEL

iragael.joly@grenoble-inp.fr

"2020-10-07"

Overview

Overview

Geovizualisation

Geovizualisation [1]

Faced with the abundance of data and tools *(multi-sensor, multi-source, multi-scale, more or less precise, more or less massive data)*, - need to visualize various geographic data (satellite imagery, maps, databases, GPS data, collaborative data, etc.), - and add quantitative / qualitative data, (also textual archives, photos old or historical image databases), - to navigate in this data, and to add / extract information yourself.

1. La géovisualisation, késako ? par Sidonie Christophe (LASTIG)

Geovisualization

- is the set of knowledge and techniques
- used to visualize a territory (or a spatial phenomenon)
- by interacting with geographic or geolocated data,
- using the user's perception and cognition capacities.

The objective of geovisualization

- is to give to see, perceive, understand and interpret,
- while preserving what has meaning, for the user,
- in the geographical space and in the spatio-temporal phenomenon represented
- (for example, flood simulation, weather prediction, climate scenarios, past urban dynamics, urban planning, etc.).

The challenge of geovisualization, as an interdisciplinary research field,

- is to design graphic representations and means of interaction with geographic data,
- to effectively help a user to see, perceive, understand, interpret, analyze or even take decisions on a spatial phenomenon.

- The use of the `ggmap` package is to geocode data and create maps using the `ggplot2` system.
- The `sp` and `sf` packages use different methodologies for integrating spatial data into R.
- The `sp` package introduced a coherent set of classes and methods for handling spatial data in 2005.
- The package remains the backbone of many packages that provide GIS capabilities in R.
- The `sf` package implements the simple features open standard for the representation of geographic vector data in R.
- `sf` package is meant to supersede `sp`, implementing ways to store spatial data in R that integrate with the `tidyverse` workflow of the packages developed by Hadley Wickham and others.

Mapping in R

Leaving point & click softs

Objectives 1 : access to a powerful (geo)statistical and visualization programming language and the benefits of a command-line approach (Sherman 2008) :

> *With the advent of 'modern' GIS software, most people want to point and click their way through life. That's good, but there is a tremendous amount of flexibility and power waiting for you with the command line.*

Some desktop Geographic Information System (GIS) are QGIS, ArcMap, GRASS or SAGA,

Using a command-line interface has the benefits of enabling both steps analysis and visualization in customizable, transparent and reproducible manner.

Objectives 2 : access a range of spatial skills, including :

- *reading, writing and manipulating geographic data ;*
- *making static and interactive maps ;*
- *applying geocomputation to solve real-world problems ;*
- *and modeling geographic phenomena.*

Using integrated reproducible ‘code chunks’ in the text, this lesson teaches a transparent and thus scientific workflow.

Table 1 – Differences in emphasis between software packages (Graphical User Interface (GUI) of Geographic Information Systems (GIS) and R).

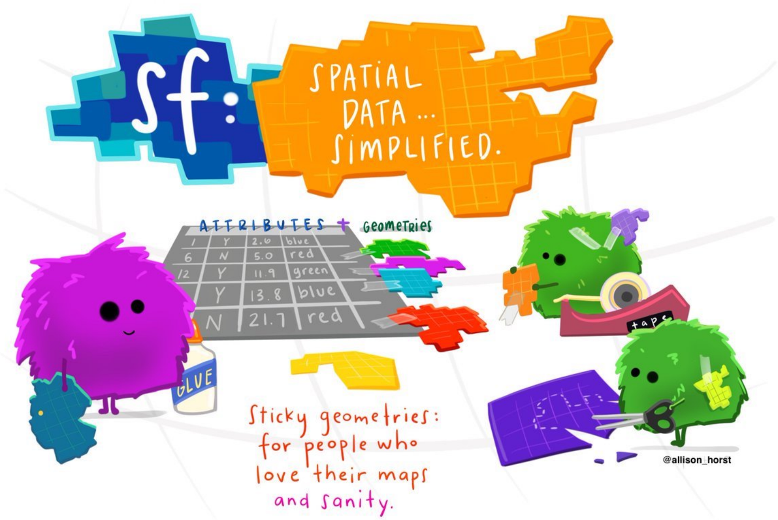
Attribute	Desktop GIS (GUI)	R
Home disciplines	Geography	Computing, Statistics
Software focus	Graphical User Interface	Command line
Reproducibility	Minimal	Maximal

All classical operations on spatialized data can be completely performed in :

- Reading and exploration of spatialized / geographic data
- Attributes manipulation (creation, selection)
- Geomatics processing (intersection, joint, surface calculation)
- Map creation (static, interactive)

To create a 2-dimensional map, a projection must be made. The areas you study will be more or less distorted by the projection you chose.

Simple Features for R : the sf package



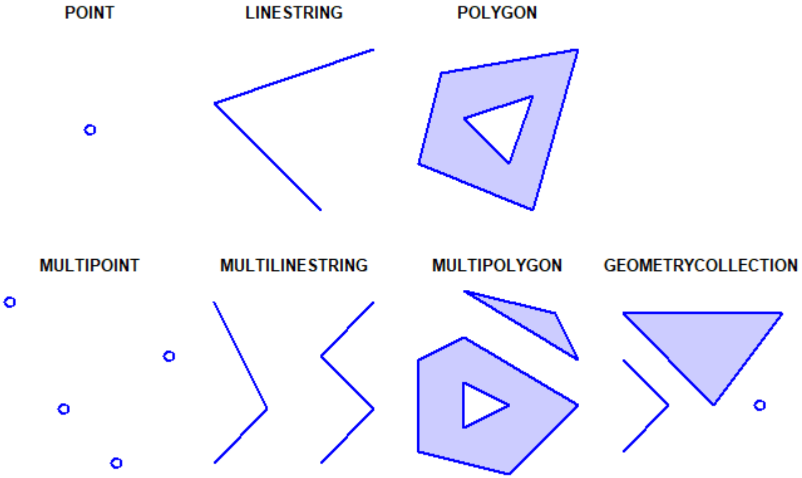
Vector layers in `sf`

- The `sf` class is a hierarchical structure composed of 3 classes
- **sf** - Vector layer object, `data.frame` with ≥ 1 attribute columns and 1 geometry column
- **sfc** - Geometric part of vector layer - geometry column
- **sfg** - Geometry of individual *simple feature*

Example of layer

X_CENTROID	Y_CENTROID	CODE_REG	NOM_REG	geometry
886172	6641548	27	BOURGOGNE-FRANCHE-COMTE	MULTIPOLYGON (((886244.2 66...
795655	6521581	84	AUVERGNE-RHONE-ALPES	MULTIPOLYGON (((764370.3 65...
550942	6952842	28	NORMANDIE	MULTIPOLYGON (((511688.8 69...
748211	6750855	27	BOURGOGNE-FRANCHE-COMTE	MULTIPOLYGON (((709449.1 67...
1016174	6763894	44	ALSACE-CHAMPAGNE-ARDENNE-LORRAINE	MULTIPOLYGON (((992779.1 67...
579506	6810114	24	CENTRE-VAL DE LOIRE	MULTIPOLYGON (((548948.9 68...

Simple feature geometry `sfg`



- spatial entities are called *features*, but for statisticians it is a record - but with a *geometry*
- column `geometry` :
    - It is where the feature's store its geometry. Each feature, in the example each department of France has a geometry, here it is Polygons.
- the *Simple Features* is made of *Points* known in coordinates (`lon` and `lat`).
- *Polygons summits* are Points and the *perimeter* can be view as a *LineString*.

Note that for the polygons, the first summit and the last summit needs to have exactly the same coordinates.

That's the way for the computer to know that it is a closed polygon and not an open LineString.

Projection Issue

Let start with the illustration of the problem.

Several CRS (*Coordinate Reference System*) exist per country [2]

Projection of Metropolitan France [3]

```
path <- 'G:/Ira_Lessons_with_Rmd/SABD_SmartAnalytics/GIS/data'

departements_L93 <- st_read(dsn   = path,
                            layer = "DEPARTEMENT",
                            quiet = TRUE) %>%
  st_transform(2154)
```

2. Example from : (https://statnmap.com/2018-07-14-introduction-to-mapping-with-sf-and-co/) Rochette (2018)
3. Example from : (https://statnmap.com/2018-07-14-introduction-to-mapping-with-sf-and-co/) Rochette (2018)

Difference between projections

```
ggplot(departements_L93) +
  aes(fill = CODE_REG) +
  scale_fill_viridis_d() +
  geom_sf() +
  coord_sf(crs = 4326) +
  guides(fill = FALSE) +
  ggtitle("Coord. géographiques") +
  theme(title = element_text(size = 16),
        plot.margin = unit(c(0,0.1,0,0.25), "inches"))
```

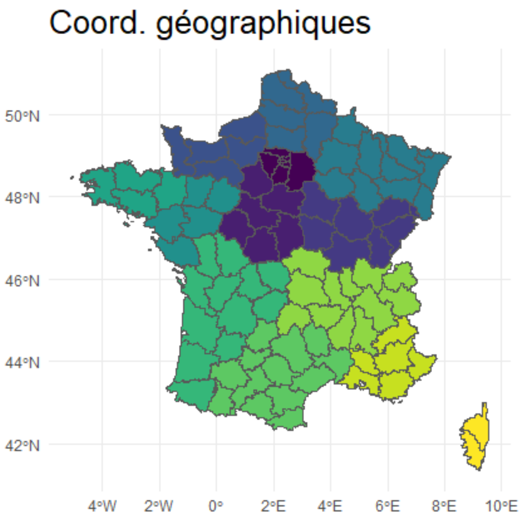
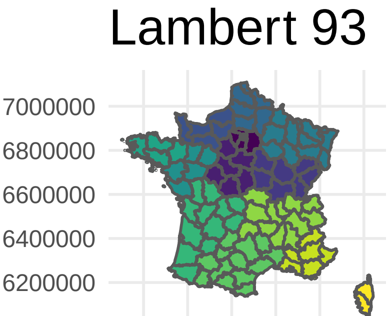


Figure 1 – France

```r
g_dept <- ggplot(departements_L93) +
  aes(fill = CODE_REG) +
  scale_fill_viridis_d() +
  geom_sf() +
  coord_sf(crs = 2154, datum = sf::st_crs(2154)) +
  guides(fill = FALSE) +
  ggtitle("Lambert 93") +
  theme(title = element_text(size = 16))
g_dept
```



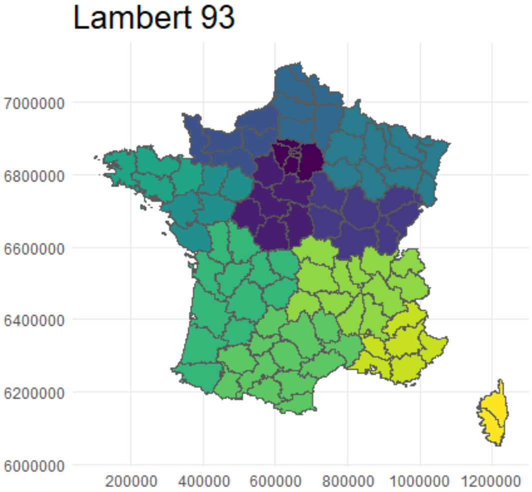
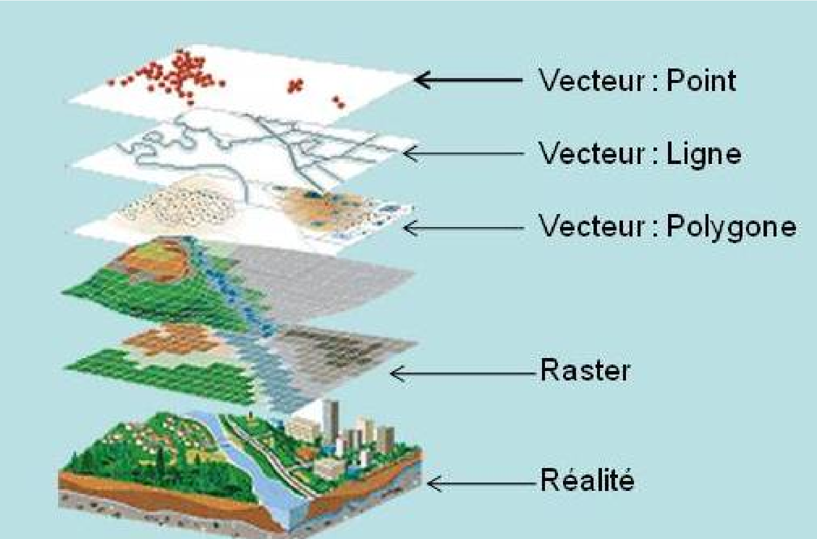


Figure 2 – France en Lambert 93

Figure 3 – Illustration of vector (point) data in which location of London (the red X) is represented with reference to an origin (the blue circle). The left plot represents a geographic CRS with an origin at 0° longitude and latitude. The right plot represents a projected CRS with an origin located in the sea west of the South West Peninsula.

Vector layer file format

We will focus only on vector data.



Vector layer file format

- `sf` reads with `st_read` all formats managed by GDAL (http://www.gdal.org/)
- ESRI shapfile format are 4 files minium `shp, shx, dbf, prj`
- with `sf` the shapefiles becomes a 'classic' dataframe

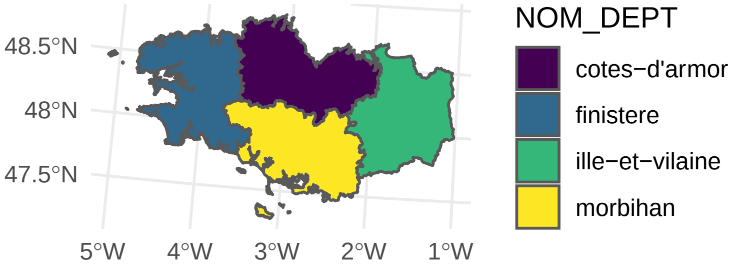
Example of manipulation

Mapping Bretagne region

```r
Bret_L93 <-
  departements_L93 %>%
  mutate_at(
    vars(NOM_DEPT, NOM_REG),
    tolower) %>%
  select(CODE_DEPT, NOM_DEPT, NOM_REG) %>%
  filter(NOM_REG == "bretagne")
Bret_L93
```

CODE_DEPT	NOM_DEPT	NOM_REG	geometry
35	ille-et-vilaine	bretagne	MULTIPOLYGON (((3304
22	cotes-d'armor	bretagne	MULTIPOLYGON (((2598
29	finistere	bretagne	MULTIPOLYGON (((1161
56	morbihan	bretagne	MULTIPOLYGON (((2562

```r
ggplot(Bret_L93) +
  geom_sf(aes(fill = NOM_DEPT)) +
  scale_fill_viridis_d()
```



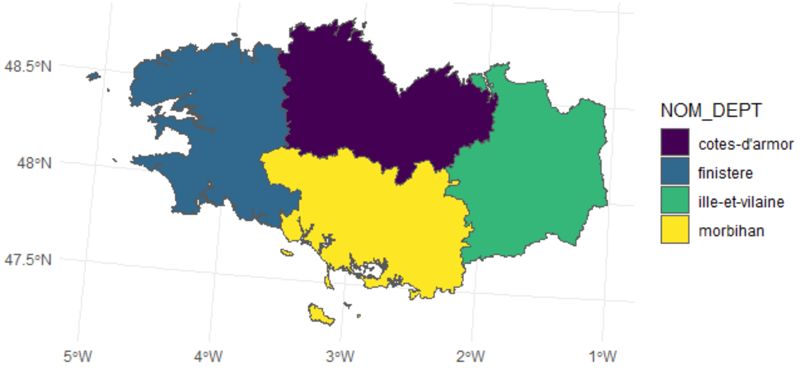


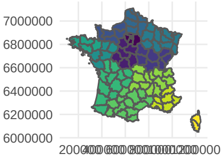
Figure 5 – Bretagne

Merge spatial features with `group_by` & `summarize`

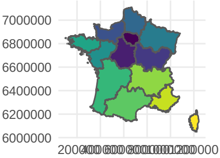
```
region_L93 <- departements_L93 %>%
  group_by(CODE_REG) %>%
  summarize()
g_region <- ggplot(region_L93)   +
   aes(fill = CODE_REG) +
   scale_fill_viridis_d() +
   geom_sf() +
   coord_sf(crs = 2154, datum = sf::st_crs(2154)) +
   guides(fill = FALSE) +
   ggtitle("Régions") +
   theme(title = element_text(size = 16))
```

```
g_region
```

Départements



Régions

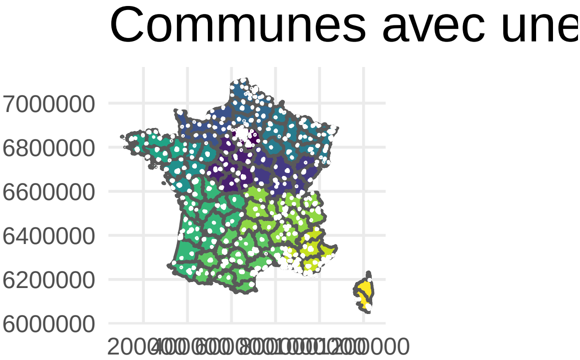


```r
# Read shapefile of French communes
communes <- st_read(dsn = path, layer = 'COMMUNE', quiet =
  select(NOM_COM, INSEE_COM)
# Read file of maternities for 2016
data.maternite <- readr::read_csv(file.path( path, "Materni
  filter(an == 2016)
```

```r
# Join database with shapefile by attributes
maternites_L93 <- communes %>%
  right_join(data.maternite,
             by = "INSEE_COM") %>%
  st_transform(2154)
```

```r
g_dept2 <- g_dept +
  geom_sf(data = maternites_L93, fill = "white", colour = '
  coord_sf(crs = 2154, datum = sf::st_crs(2154)) +
  ggtitle("Communes avec une maternité en 2016")
```

```
g_dept2
```


Communes avec une
7000000
6800000
6600000
6400000
6200000
6000000

Conclusion

Wrap up : Geometric calculations

Geometric operations on vector layers can conceptually be divided into **three groups** according to their output :

- **Numeric** values : Functions that summarize geometrical properties of :
  - A **single layer** (e.g. area, length)
  - A **pair of layers** (e.g. distance)
- **Logical** values : Functions that evaluate whether a certain condition holds true, regarding :
  - A **single layer** (e.g. geometry is valid)
  - A **pair of layers** (e.g. feature A intersects feature B)
- **Spatial** layers : Functions that create a new layer based on :
  - A **single layer** (e.g. centroids)
  - A **pair of layers** (e.g. intersection area)

Numeric

- Several functions to calculate **numeric geometric properties** of vector layers :
  - `st_length`
  - `st_area`
  - `st_distance`
  - `st_bbox`
  - ...

Logical

- Given two layers, `x` and `y`, the following **logical geometric functions** check whether each feature in `x` maintains the specified **relation** with each feature in `y` :
  - `st_intersects`
  - `st_disjoint`
  - `st_touches`
  - `st_crosses`
  - `st_within`
  - `st_contains`
  - `st_overlaps`
  - `st_covers`
  - `st_equals`
  - ...

Spatial

- Common **geometry-generating** functions applicable to **individual** geometries :
  - `st_centroid`
  - `st_buffer`
  - `st_union`
  - `st_sample`
  - `st_convex_hull`
  - `st_voronoi`
  - ...

All sf methods

```r
methods(class='sf')
```

```
##  [1] $<-                 [                   [[<-
##  [4] aggregate           anti_join           arrange
##  [7] as.data.frame       cbind               coerce
## [10] dbDataType          dbWriteTable        distinc
## [13] dplyr_reconstruct   filter              full_jo
## [16] gather              group_by            group_s
## [19] identify            initialize          inner_j
## [22] left_join           merge               mutate
## [25] nest                plot                print
## [28] rbind               rename              right_j
## [31] sample_frac         sample_n            select
## [34] semi_join           separate            separat
## [37] show                slice               slotsFr
## [40] spread              st_agr              st_agr
```

To dive even deeper into sf

- Detailed sf package vignettes
- Blog posts : here, here, here, here, here and there (in French)

- Video of Edzer Pebesma at rstudio::conf 2018
- wiki page describing sp-sf migration
- Awesome online book Geocomputation with R by Lovelace, Nowosad and Muenchow

Spatial manipulation with sf: : CHEAT SHEET

The sf package provides a set of tools for working with geospatial vectors, i.e. points, lines, polygons, etc.

Geometric confirmation

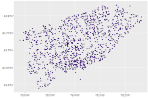
- st_contains(x, y, ...) Identifies if x is within y (i.e. point within polygon)
- st_covered_by(x, y, ...) Identifies if x is completely within y (i.e. polygon completely within polygon)
- st_covers(x, y, ...) Identifies if any point from x is outside of y (i.e. polygon outside polygon)
- st_crosses(x, y, ...) Identifies if any geometry of x have commonalities with y
- st_disjoint(x, y, ...) Identifies when geometries from x do not share space with y
- st_equals(x, y, ...) Identifies if x and y share the same geometry
- st_intersects(x, y, ...) Identifies if x and y geometry share any space
- st_overlaps(x, y, ...) Identifies if geometries of x and y share space, are of the same dimension, but are not completely contained by each other
- st_touches(x, y, ...) Identifies if geometries of x and y share a common point but their interiors do not intersect
- st_within(x, y, ...) Identifies if x is in a specified distance to y

Geometric operations

- st_boundary(x) Creates a polygon that encompasses the full extent of the geometry
- st_buffer(x, dist, nQuadSegs) Creates a polygon covering all points of the geometry within a given distance
- st_centroid(x, ..., of_largest_polygon) Creates a point at the geometric centre of the geometry
- st_convex_hull(x) Creates geometry that represents the minimum convex geometry of x
- st_line_merge(x) Creates linestring geometry from sewing multi linestring geometry together
- st_node(x) Creates nodes on overlapping geometry where nodes do not exist
- st_point_on_surface(x) Creates a point that is guarenteed to fall on the surface of the geometry
- st_polygonize(x) Creates polygon geometry from linestring geometry
- st_segmentize(x, dfMaxLength, ...) Creates linesting geometry from x based on a specified length
- st_simplify(x, preserveTopology, dTolerance) Creates a simplified version of the geometry based on a specified tolerance

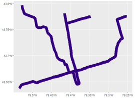
Geometry creation

- st_triangulate(x, dTolerance, bOnlyEdges) Creates polygon geometry as triangles from point geometry
- st_voronoi(x, envelope, dTolerance, bOnlyEdges) Creates polygon geometry covering all points of x, with x at the centre of the geometry
- st_point(x, c(numeric vector), dim = "XYZ") Creating point geometry from numeric values
- st_multipoint(x = matrix(numeric values in rows), dim = "XYZ") Creating multi point geometry from numeric values
- st_linestring(x = matrix(numeric values in rows), dim = "XYZ") Creating linestring geometry from numeric values
- st_multilinestring(x = list(numeric matricesin rows), dim = "XYZ") Creating multi linestring geometry from numeric values
- st_polygon(x = list(numeric matrices in rows), dim = "XYZ") Creating polygon geometry from numeric values
- st_multipolygon(x = list(numeric matrices in rows), dim = "XYZ") Creating multi polygon geometry from numeric values



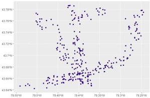
ggplot() +
geom_sf(data = schools)

+



ggplot() +
geom_sf(data = subway)

=>



ggplot() +
geom_sf(data = st_intersection(schools, st_buffer(subway, 1000)))

This cheatsheet presents the sf package [Edzer Pebesma 2018] in version 0.6.3. See https://github.com/r-spatial/sf for more details.

Spatial manipulation with sf: : CHEAT SHEET

The sf package provides a set of tools for working with geospatial vectors, i.e. points, lines, polygons, etc.

Geometry operations

st_contains(x, y, ...) Identifies if x is within y (i.e. point within polygon)

st_crop(x, y, ..., xmin, ymin, xmax, ymax) Creates geometry of x that intersects a specified rectangle

st_difference(x, y) Creates geometry from x that does not intersect with y

st_intersection(x, y) Creates geometry of the shared portion of x and y

st_sym_difference(x, y) Creates geometry representing portions of x and y that do not intersect

st_snap(x, y, tolerance) Snap nodes from geometry x to geometry y

st_union(x, y, ..., by_feature) Creates multiple geometries into a a single geometry, consisiting of all geometry elements

Geometric measurement

st_area(x) Calculate the surface area of a polygon geometry based on the current coordinate reference system

st_distance(x, y, ..., dist_fun, by_element, which) Calculates the 2D distance between x and y based on the current coordinate system

st_length(x) Calculates the 2D length of a geometry based on the current coordinate system

Misc operations

st_cast(x, to, ...) Change x geometry to a different geometry type

st_coordinates(x, ...) Creates a matrix of coordinate values from x

st_crs(x, ...) Identifies the coordinate reference system of x

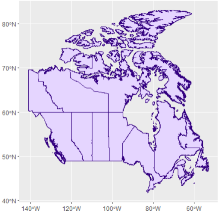
st_join(x, y, join, FUN, suffix, ...) Performs a spatial left or inner join between x and y

st_make_grid(x, cellsize, offset, n, crs, what) Creates rectangular grid geometry over the bounding box of x

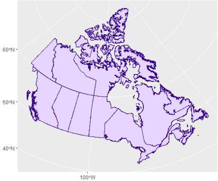
st_nearest_feature(x, y) Creates an index of the closest feature between x and y

st_nearest_points(x, y, ...) Returns the closest point between x and y

st_transform(x, crs, ...) Convert coordinates of x to a different coordinate reference system




```
ggplot() +
  geom_sf(data = cdn) +
  coord_sf(crs = st_crs(4326))
```



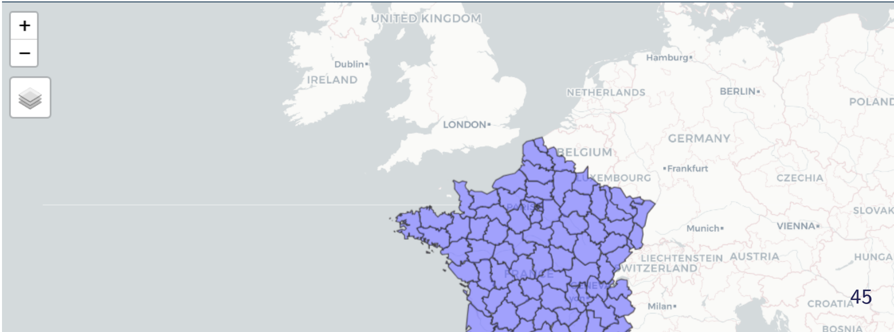

```
ggplot() +
  geom_sf(data = cdn) +
  coord_sf(crs = st_crs(3347))
```

This cheatsheet presents the sf package [Edzer Pebesma 2018] in version 0.6.3. See https://github.com/r-spatial/sf for more details.

Miscellaneous

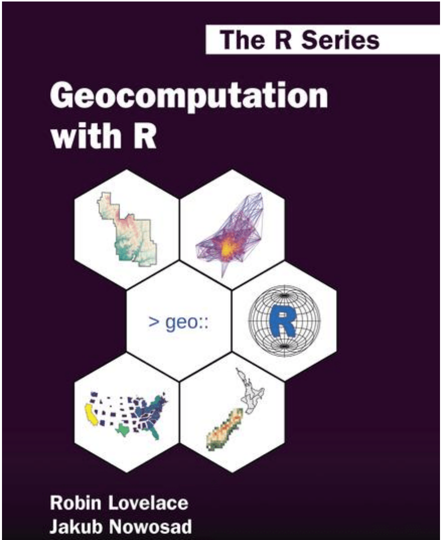
- What about raster data ? Check out package **star**
- `mapview::mapview()` creates interactive maps in html pages, using package `leaflet`, very useful to inspect spatial data (see also package `tmap`)

```
mapview::mapview(departements_2)
```



References

- Geocomputation with R by Robin Lovelace, Jakub Nowosad, Jannes Muenchow



- R Spatial by Edzer Pebesma
- Tidy spatial data analysis (video) by Edzer Pebesma at rstudio : :conf 2018
- Introduction to mapping with {sf} & Co. on spatial analysis with R by Sebastien Rochette
- Introduction to GIS and mapping in R using the sf package, olivier gimenez
- Nick Eubank's, GIS in R
- The package vignettes for sf are very helpful for providing an introduction to the package.
- (https://tender-curie-5b83bc.netlify.app/2019/03/01/mapping-sncf-stations/)

Bibliography

Rochette, Sébastien. 2018. "Introduction to Mapping with sf & Co." https://statnmap.com/2018-07-14-introduction-to-mapping-with-sf-and-co/.