



CESAB
CENTRE DE SYNTHÈSE ET D'ANALYSE
SUR LA BIODIVERSITÉ



Les données spatiales

{sf} {ggplot2}

Nina SCHIETTEKATTE
(Nicolas CASAJUS)

Mardi 3 novembre 2020

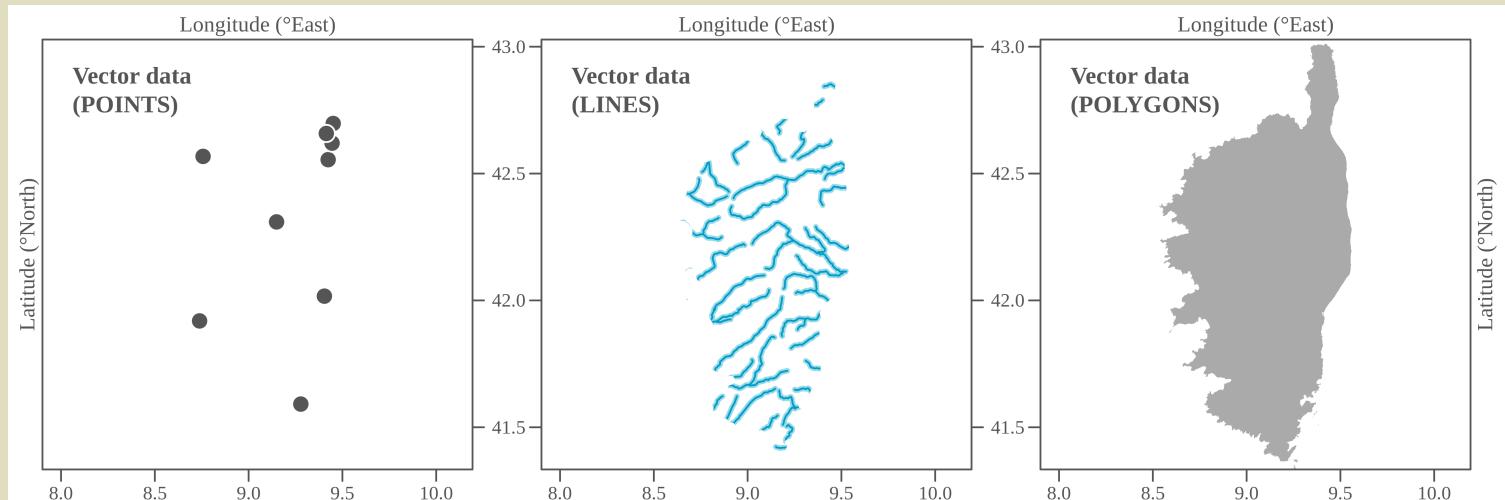


Introduction

Qu'est-ce qu'un objet spatial ?

👉 Deux catégories :

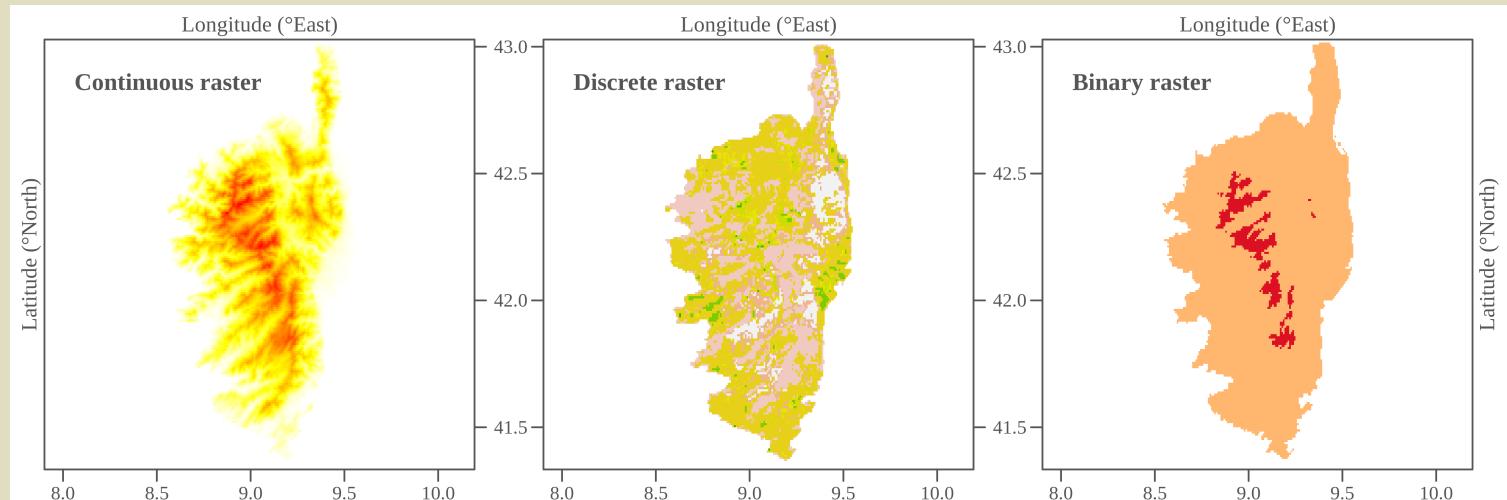
1) Les données vectorielles



Qu'est-ce qu'un objet spatial ?

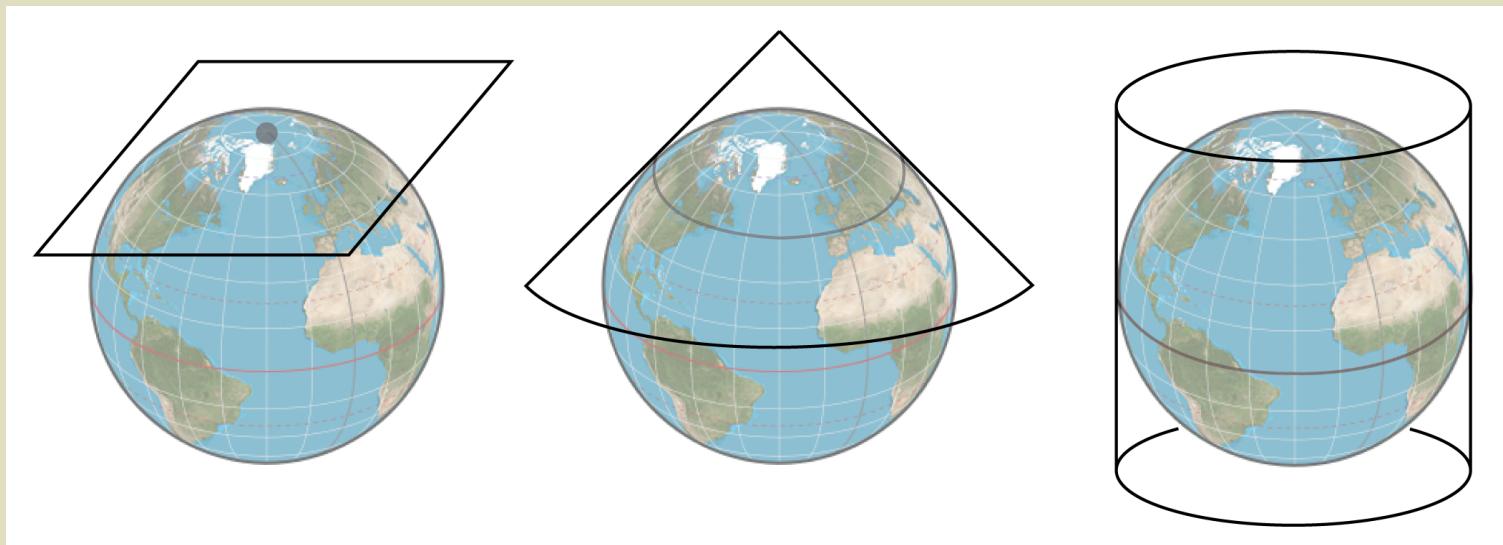
👉 Deux catégories :

2) Les données matricielles



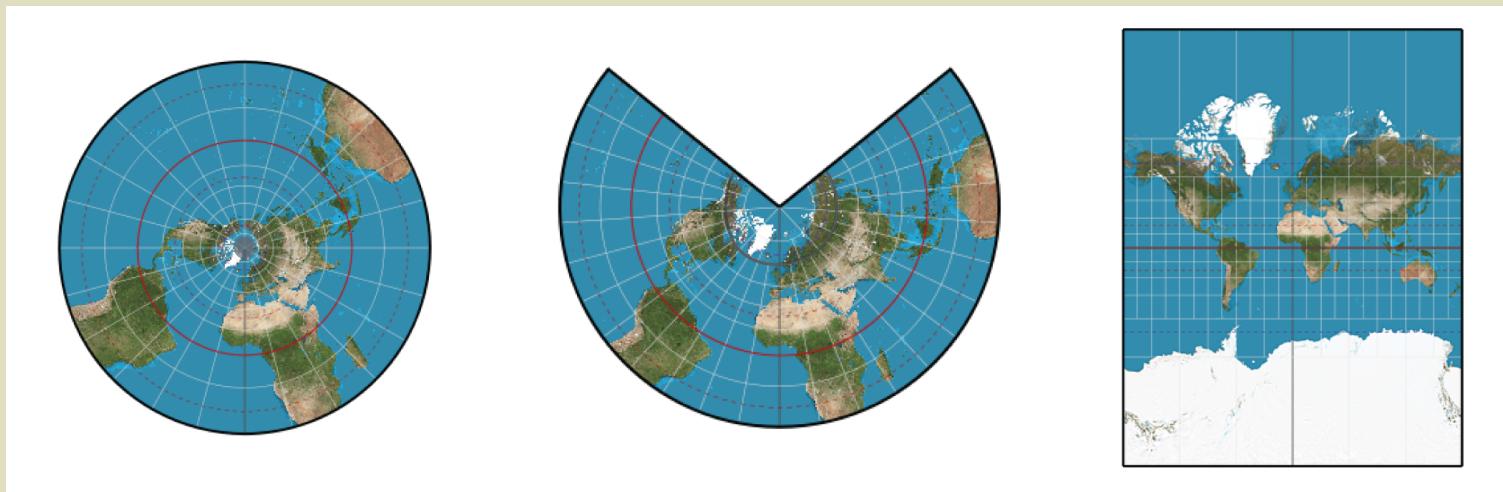
Le système de coordonnées

- 👉 Un objet spatial se représente dans l'espace selon un référentiel spatial : le **système de coordonnées (CRS)** pour *Coordinates Reference System*)
 - 👉 Deux types de systèmes de coordonnées existent :
- 1) Les systèmes géographiques (ou non projetés) [en degrés]



Le système de coordonnées

- 👉 Un objet spatial se représente dans l'espace selon un référentiel spatial : le **système de coordonnées (CRS)** pour *Coordinates Reference System*)
 - 👉 Deux types de systèmes de coordonnées existent :
- 2) Les systèmes projetés [en mètres]**



- 👉 Le choix du CRS peut être crucial et dépend souvent de l'information que l'on souhaite représenter

Le système de coordonnées

Sous  le CRS s'exprime selon le standard **proj4string** défini par le projet **PROJ**

```
+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
```

Chaque CRS possède un **SRID** (*Spatial Reference Identifier*) ou **EPSG** (*European Petroleum Survey Group*) : un identifiant unique qui nous évite de devoir écrire à la main le CRS au format **proj4string**

+init=epsg:4326 est l'identifiant du CRS ci-dessus

```
+init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
```

👉 Où trouver le bon CRS ? Réponse : **Spatial Reference**

Objets vectoriels

Les packages R - Vecteurs

Package	Fonctionnalités
{sp}	Classes et méthodes pour couches vectorielles
{rgdal}	Lecture et écriture de couches spatiales
{rgeos}	Opérations spatiales sur couches vectorielles
{maptools}	Opérations spatiales sur couches vectorielles
{sf}	Le petit dernier - Une vraie bête !

👉 {sf} est devenu la "norme": {sf} = {sp} + {rgdal} + {rgeos}

Pourquoi {sf} ?

- Implémentation sous  des *Simple Features*
 - Standard reposant sur la norme ISO 19125 (stockage et accès)
 - Remplace le format d'ESRI™, vous savez le shapefile !?
- 3 packages en 1
- Syntaxe (*snake_case*) des fonctions simples (`{st_()^*}`)
- Objets stockés dans des `data.frame` ou des `tibble`
 - avec la géométrie stockée dans des `list-column`
- Très grandes performances comparées à `{sp}`
- Pipe-friendly et intégration au tidyverse (dont `{ggplot2}`)

👉 Edzer Pebesma le recommande 😊

Structure d'un objet sf

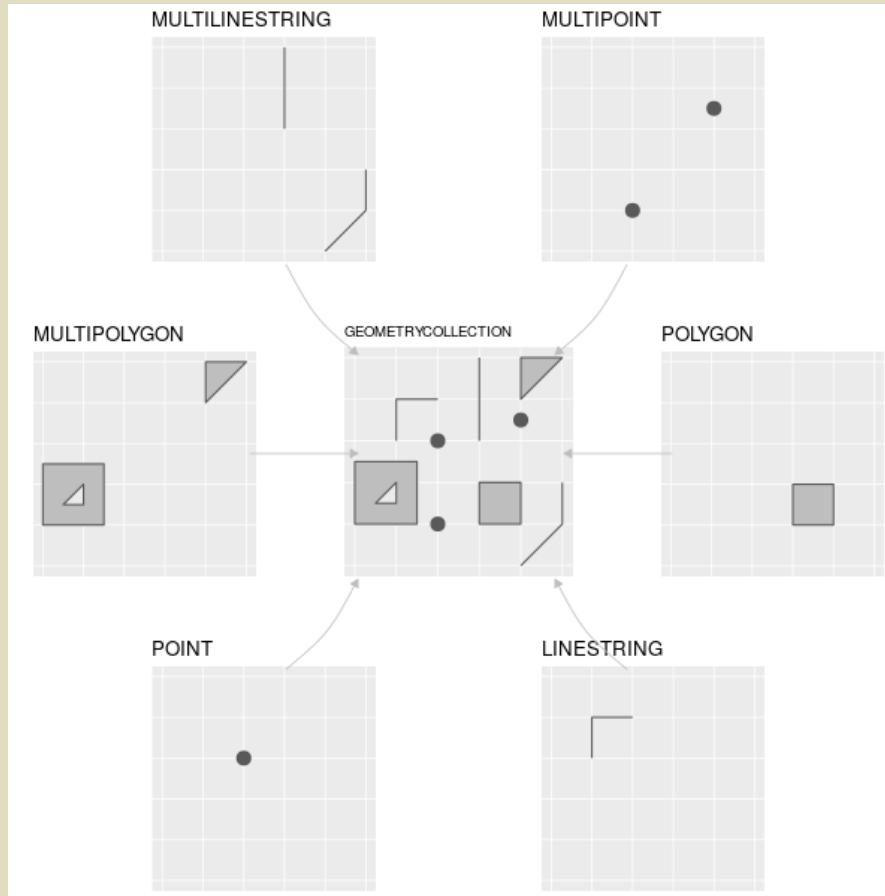
```
## Simple feature collection with 100 features and 6 fields
## geometry type: MULTIPOLYGON
## dimension: XY
## bbox: xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## epsg (SRID): 4267
## proj4string: +proj=longlat +datum=NAD27 +no_defs
## precision: double (default; no precision model)
## First 3 features:
##   BIR74 SID74 NWBIR74 BIR79 SID79 NWBIR79      geom
## 1  1091     1      10   1364     0    19 MULTIPOLYGON((( -81.47275543...
## 2   487     0      10    542     3    12 MULTIPOLYGON((( -81.23989105...
## 3  3188     5     208   3616     6   260 MULTIPOLYGON((( -80.45634460...
```

Simple feature

Simple feature geometry list-column (sfc)

Simple feature geometry (sfg)

Les *Simple feature geometry* sfg



Installation

```
## install.packages("sf")
```

Création de points spatiaux

Soit les villes avec les coordonnées suivantes :

```
foix      <- c(1.6053807, 42.9638998)
carcassonne <- c(2.3491069, 43.2130358)
ronez      <- c(2.5728419, 44.3516207)
```

Convertissons ces vecteurs en points spatiaux :

```
foix      <- sf::st_point(foix)
carcassonne <- sf::st_point(carcassonne)
ronez      <- sf::st_point(ronez)
```

Et regroupons-les en un seul *spatial column* en définissant le CRS :

```
villes <- sf::st_sf(
  list(foix, carcassonne, ronez),
  crs = 4326
)
class(villes)

## [1] "sf<POINT" "sf"
```

Création de points spatiaux

Ajoutons une table d'attributs et convertissons l'objet en *simple feature* :

```
datas <- data.frame(  
  ville      = c("Foix", "Carcassonne", "Rodez"),  
  population = c(9613, 45895, 23739)  
)  
  
villes <- sf:::st_sf(datas, geom = villes)  
class(villes)  
  
## [1] "sf"           "data.frame"
```

👉 Quid des lignes et polygones ?

Voir les fonctions `st_linestring()` et `st_polygon()`

Création de points spatiaux

Bien souvent, vous importerez directement un `data.frame` que vous souhaiterez convertir en *simple feature*

```
## URL et nom du fichier
url      <- "https://raw.githubusercontent.com/FRBCesab/datatoolbox/master/data/"
filename <- "occitanie_prefectures.csv"

## Téléchargement du fichier
download.file(
  url      = paste0(url, filename),
  destfile = filename
)

## Extraction du ZIP
unzip(zipfile = filename)
```

Création de points spatiaux

```
## Ouverture du csv  
(prefectures <- readr::read_csv("occitanie_prefectures.csv"))  
  
## # A tibble: 13 x 5  
##   departement    prefecture  population longitude latitude  
##   <chr>          <chr>        <dbl>      <dbl>     <dbl>  
## 1 Ariège         Foix           9613       1.61     43.0  
## 2 Aude           Carcassonne   45895      2.35     43.2  
## 3 Aveyron        Rodez          23739      2.57     44.4  
## 4 Gard            Nîmes          151001     4.36     43.8  
## 5 Haute-Garonne Toulouse      475438     1.44     43.6  
## 6 Gers            Auch           21618      0.585    43.6  
## 7 Hérault        Montpellier  281613     3.88     43.6  
## 8 Lot              Cahors         19405      1.44     44.4  
## 9 Lozère          Mende          11860      3.50     44.5  
## 10 Hautes-Pyrénées Tarbes        40318      0.0781   43.2  
## 11 Pyrénées-Orientales Perpignan  121875     2.90     42.7  
## 12 Tarn            Albi           49024      2.15     43.9  
## 13 Tarn-et-Garonne Montauban    60444      1.35     44.0
```

Création de points spatiaux

```
prefectures <- sf::st_as_sf(  
  x      = prefectures,  
  coords = c("longitude", "latitude"),  
  crs    = 4326  
)  
prefectures %>% head(6)  
  
## Simple feature collection with 6 features and 3 fields  
## geometry type:  POINT  
## dimension:      XY  
## bbox:            xmin: 0.5850507 ymin: 42.9639 xmax: 4.360069 ymax: 44.35162  
## geographic CRS: WGS 84  
## # A tibble: 6 x 4  
##   departement  prefecture  population           geometry  
##   <chr>        <chr>       <dbl>          <POINT [°]>  
## 1 Ariège       Foix         9613  (1.605381 42.9639)  
## 2 Aude         Carcassonne  45895 (2.349107 43.21304)  
## 3 Aveyron      Rodez        23739 (2.572842 44.35162)  
## 4 Gard          Nîmes        151001 (4.360069 43.83742)  
## 5 Haute-Garonne Toulouse  475438 (1.444247 43.60446)  
## 6 Gers          Auch         21618 (0.5850507 43.64636)
```

Une première carte

Les packages `{rnatural-earth}` et `{rnatural-earth-data}` proposent différentes couches vectorielles, dont les limites administratives des pays du monde.

Installons ces packages :

```
## install.packages("rnatural-earth")
## install.packages("rnatural-earth-data")
```

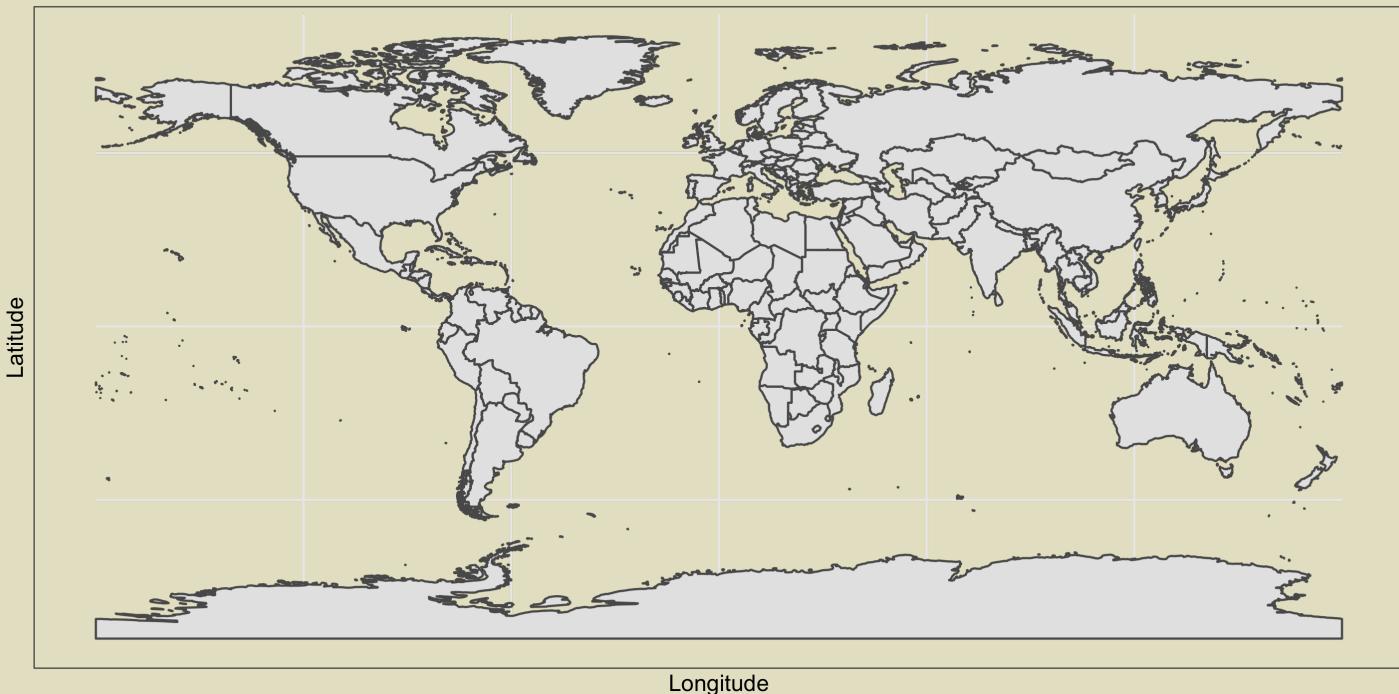
Et chargeons le fond de carte du monde entier :

```
world <- rnatural-earth::ne_countries(scale = "medium", returnclass = "sf")
class(world)
```

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

Une première carte

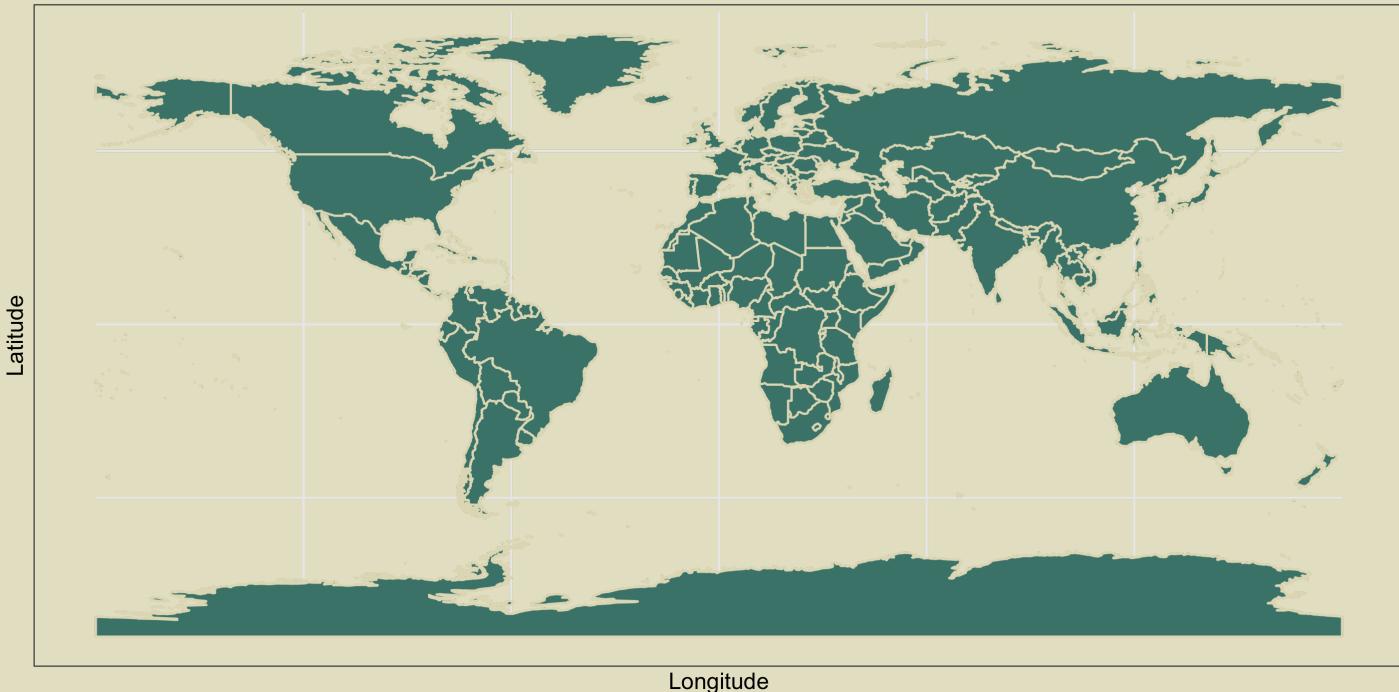
Carte du monde



```
ggplot(data = world) +  
  geom_sf() +  
  xlab("Longitude") + ylab("Latitude") + ggtitle("Carte du monde")
```

Une première carte

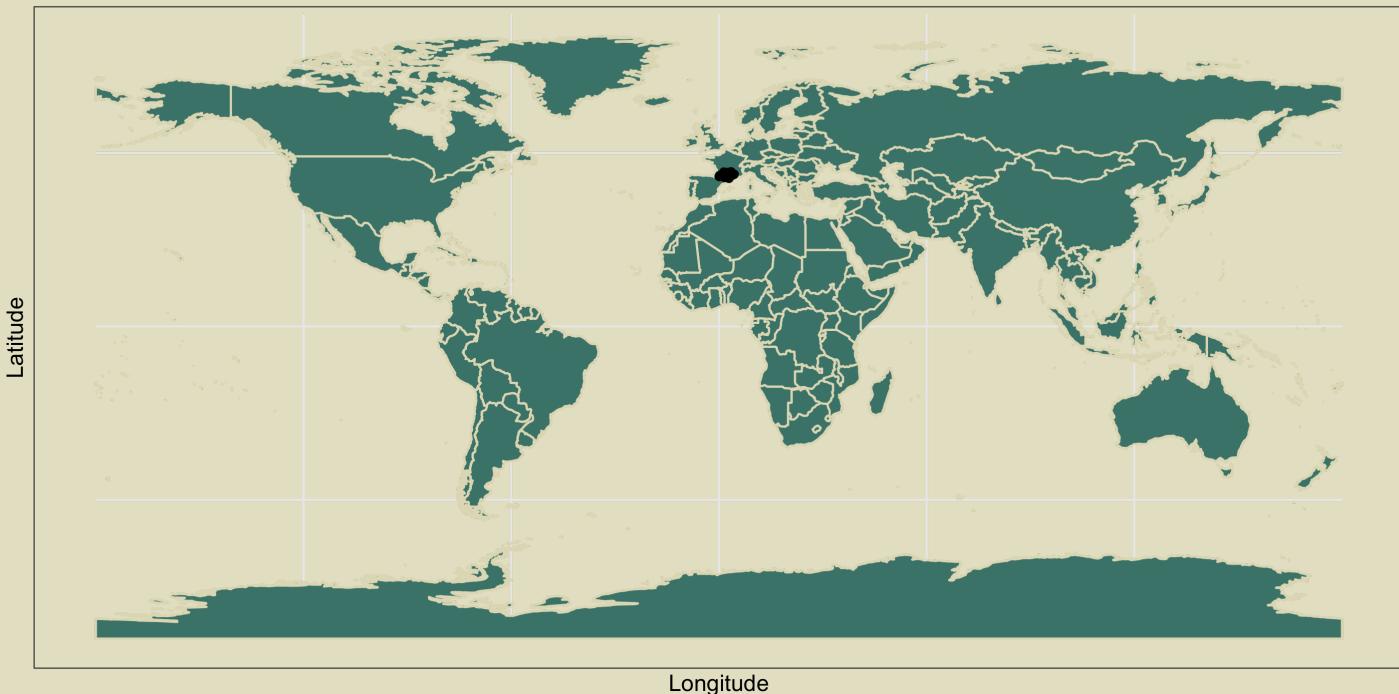
Carte du monde



```
ggplot(data = world) +  
  geom_sf(fill = "#49847b", colour = "#e1ddc0") +  
  xlab("Longitude") + ylab("Latitude") + ggtitle("Carte du monde")
```

Une première carte

Carte du monde



```
ggplot(data = world) +  
  geom_sf(fill = "#49847b", colour = "#e1ddc0") +  
  xlab("Longitude") + ylab("Latitude") + ggtitle("Carte du monde") +  
  geom_sf(data = prefectures, colour = "black")
```

Importation d'une couche vectorielle

Dans la plupart des cas, vous devrez importer une couche spatiale déjà prête.

Téléchargeons les départements de la région Occitanie

```
## URL et nom du fichier
url      <- "https://raw.githubusercontent.com/FRBCesab/datatoolbox/master/data/"
filename <- "occitanie.zip"

## Téléchargement du fichier
download.file(
  url      = paste0(url, filename),
  destfile = filename
)

## Extraction du ZIP
unzip(zipfile = filename)
```

Importation d'une couche vectorielle

Que contient la couche ?

```
sf::st_layers("occitanie.shp")  
  
## Driver: ESRI Shapefile  
## Available layers:  
##   layer_name geometry_type features fields  
## 1  occitanie      Polygon       13     13
```

Importons-la :

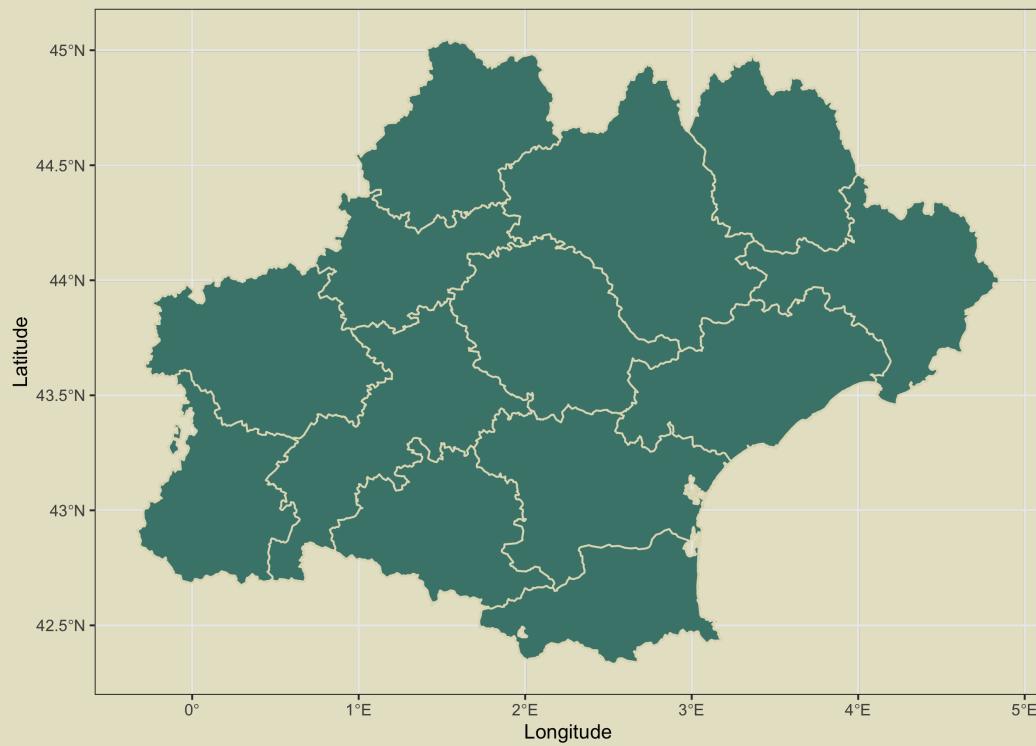
```
occitanie <- sf::st_read(dsn = ".", layer = "occitanie")  
  
## Reading layer 'occitanie' from data source '/Users/nicolascasajus/Desktop/...'  
## Simple feature collection with 13 features and 13 fields  
## geometry type: MULTIPOLYGON  
## dimension: XY  
## bbox: xmin: -0.326875 ymin: 42.33344 xmax: 4.845344 ymax: 45.04467  
## epsg (SRID): 4326  
## proj4string: +proj=longlat +datum=WGS84 +no_defs
```

Importation d'une couche vectorielle

```
occitanie %>% head(3)
```

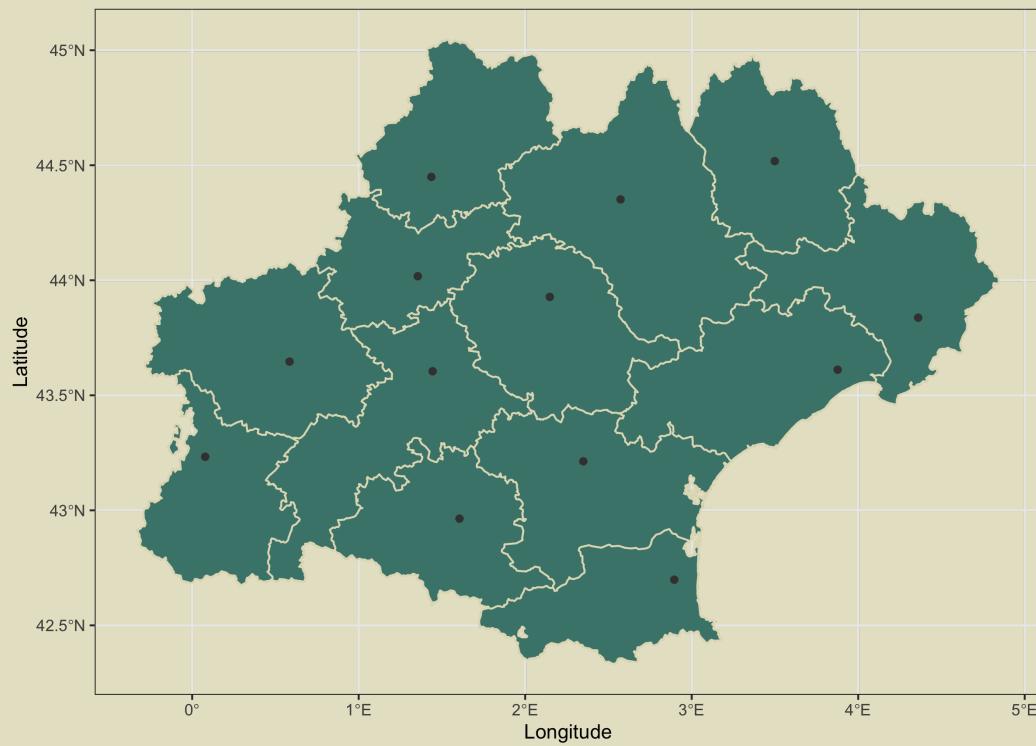
```
## Simple feature collection with 3 features and 13 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:            xmin: 0.8266816 ymin: 42.57232 xmax: 3.450981 ymax: 44.94122
## geographic CRS: WGS 84
##   GID_0 NAME_0    GID_1    NAME_1 NL_NAME_1      GID_2 NAME_2 VARNAME_2
## 1 FRA France FRA.11_1 Occitanie <NA> FRA.11.1_1 Ariège <NA>
## 2 FRA France FRA.11_1 Occitanie <NA> FRA.11.2_1 Aude <NA>
## 3 FRA France FRA.11_1 Occitanie <NA> FRA.11.3_1 Aveyron <NA>
##   NL_NAME_2      TYPE_2 ENGTYPE_2 CC_2 HASC_2                               geometry
## 1 <NA> Département Department 09 FR.AG MULTIPOLYGON (((1.494104 42...
## 2 <NA> Département Department 11 FR.AD MULTIPOLYGON (((3.027638 42...
## 3 <NA> Département Department 12 FR.AV MULTIPOLYGON (((3.236248 43...
```

OCCITANIE

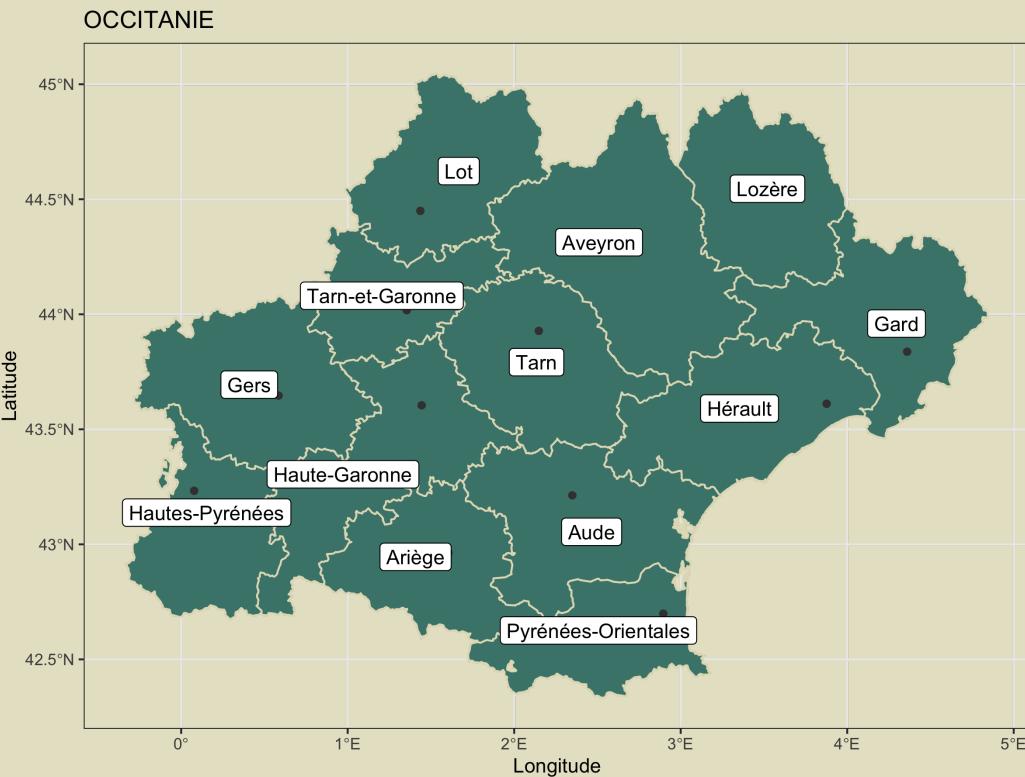


```
ggplot(data = occitanie) +  
  geom_sf(fill = "#49847b", colour = "#e1ddc0") +  
  xlab("Longitude") + ylab("Latitude") + ggtitle("OCCITANIE")
```

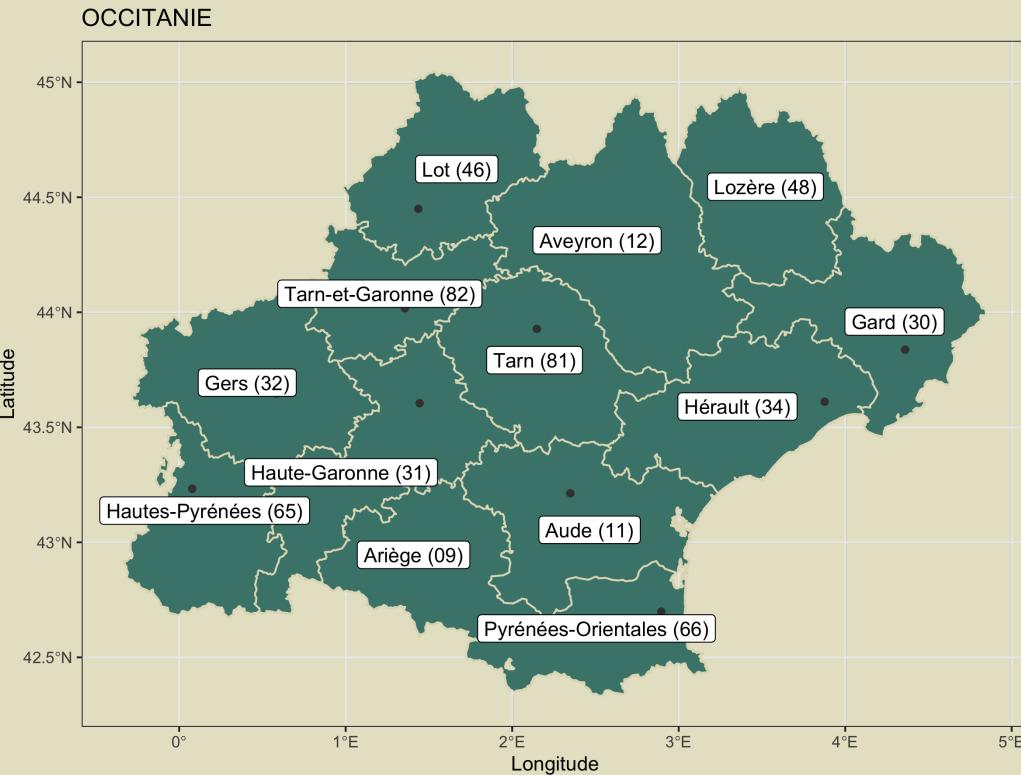
OCCITANIE



```
ggplot(data = occitanie) +  
  geom_sf(fill = "#49847b", colour = "#e1ddc0") +  
  xlab("Longitude") + ylab("Latitude") + ggtitle("OCCITANIE") +  
  geom_sf(data = prefectures, colour = "#3f3f3f")
```



```
ggplot(data = occitanie) +  
  geom_sf(fill = "#49847b", colour = "#e1ddc0") +  
  xlab("Longitude") + ylab("Latitude") + ggtitle("OCCITANIE") +  
  geom_sf(data = prefectures, colour = "#3f3f3f") +  
  geom_sf_label(aes(label = NAME_2))
```



```
ggplot(data = occitanie) +  
  geom_sf(fill = "#49847b", colour = "#e1ddc0") +  
  xlab("Longitude") + ylab("Latitude") + ggtitle("OCCITANIE") +  
  geom_sf(data = prefectures, colour = "#3f3f3f") +  
  geom_sf_label(aes(label = paste0(NAME_2, " (", CC_2, ")")))
```

Exportation d'une couche vectorielle

```
sf::st_drivers()
```

		name	long_name	write	copy	is_raster	is_vector	vsi
		PCIDSK	PCIDSK Database File	TRUE	FALSE	TRUE	TRUE	TRUE
		netCDF	Network Common Data Format	TRUE	TRUE	TRUE	TRUE	FALSE
		PDF	Geospatial PDF	TRUE	TRUE	TRUE	TRUE	FALSE
		MBTiles	MBTiles	TRUE	TRUE	TRUE	TRUE	TRUE
		EEDA	Earth Engine Data API	FALSE	FALSE	FALSE	TRUE	FALSE
ESRI	Shapefile		ESRI Shapefile	TRUE	FALSE	FALSE	TRUE	TRUE
	MapInfo File		MapInfo File	TRUE	FALSE	FALSE	TRUE	TRUE
	UK .NTF		UK .NTF	FALSE	FALSE	FALSE	TRUE	TRUE
	OGR_SDTS		SDTS	FALSE	FALSE	FALSE	TRUE	TRUE
	S57		IHO S-57 (ENC)	TRUE	FALSE	FALSE	TRUE	TRUE
	DGN		Microstation DGN	TRUE	FALSE	FALSE	TRUE	TRUE
	OGR_VRT		VRT - Virtual Datasource	FALSE	FALSE	FALSE	TRUE	TRUE
	REC		EPIInfo .REC	FALSE	FALSE	FALSE	TRUE	FALSE
	Memory		Memory	TRUE	FALSE	FALSE	TRUE	FALSE
	BNA		Atlas BNA	TRUE	FALSE	FALSE	TRUE	TRUE
	CSV	Comma Separated Value (.csv)		TRUE	FALSE	FALSE	TRUE	TRUE
	GML	Geography Markup Language (GML)		TRUE	FALSE	FALSE	TRUE	TRUE
	GPX		GPX	TRUE	FALSE	FALSE	TRUE	TRUE
	KML	Keyhole Markup Language (KML)		TRUE	FALSE	FALSE	TRUE	TRUE
	GeoJSON		GeoJSON	TRUE	FALSE	FALSE	TRUE	TRUE

Exportation d'une couche vectorielle

Au format ESRI™ :

```
sf::st_write(  
  obj    = prefectures,  
  dsn    = ".",  
  layer   = "prefectures",  
  driver = "ESRI Shapefile"  
)
```

Le système de coordonnées

Le CRS de notre couche vectorielle (monde) est-il défini ?

```
world
```

```
Simple feature collection with 241 features and 63 fields
geometry type:  MULTIPOLYGON
dimension:      XY
bbox:           xmin: -180 ymin: -89.99893 xmax: 180 ymax: 83.59961
CRS:            +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
...
```

Supprimons-le :

```
world <- world %>% sf::st_set_crs(NA)
```

Le système de coordonnées

Et redéfinissons-le :

```
world <- world %>% sf::st_set_crs(4326)
```

```
Simple feature collection with 241 features and 63 fields
geometry type:  MULTIPOLYGON
dimension:      XY
bbox:           xmin: -180 ymin: -89.99893 xmax: 180 ymax: 83.59961
geographic CRS: WGS 84
...
...
```

Projetons notre couche dans un autre système (par .ex, la projection Lambert azimuthal equal area centrée sur la France)

```
prj <- paste(
  "+proj=laea +lat_0=48 +lon_0=5",
  "+x_0=4321000 +y_0=3210000",
  "+ellps=GRS80 +units=m +no_defs"
)
world_ortho <- world %>% sf::st_transform(crs = prj)
```



```
ggplot(data = world_ortho) +  
  geom_sf(fill = "#49847b", colour = "#e1ddc0")
```



```
ggplot(data = world) +  
  geom_sf(fill = "#49847b", colour = "#e1ddc0") +  
  coord_sf(crs = prj)
```

Manipulations spatiales : sf cheat sheet

	<code>st_contains(x, y, ...)</code> Identifies if x is within y (i.e. point within polygon)		<code>st_boundary(x)</code> Creates a polygon that encompasses the full extent of the geometry
	<code>st_covered_by(x, y, ...)</code> Identifies if x is completely within y (i.e. polygon completely within polygon)		<code>st_buffer(x, dist, nQuadSegs)</code> Creates a polygon covering all points of the geometry within a given distance
	<code>st_covers(x, y, ...)</code> Identifies if any point from x is outside of y (i.e. polygon outside polygon)		<code>st_centroid(x, ..., of_largest_polygon)</code> Creates a point at the geometric centre of the geometry
	<code>st_crosses(x, y, ...)</code> Identifies if any geometry of x have commonalities with y		<code>st_convex_hull(x)</code> Creates geometry that represents the minimum convex geometry of x
	<code>st_disjoint(x, y, ...)</code> Identifies when geometries from x do not share space with y		<code>st_line_merge(x)</code> Creates linestring geometry from sewing multi linestring geometry together
	<code>st_equals(x, y, ...)</code> Identifies if x and y share the same geometry		<code>st_node(x)</code> Creates nodes on overlapping geometry where nodes do not exist
	<code>st_intersects(x, y, ...)</code> Identifies if x and y geometry share any space		<code>st_point_on_surface(x)</code> Creates a point that is guaranteed to fall on the surface of the geometry
	<code>st_overlaps(x, y, ...)</code> Identifies if geometries of x and y share space, are of the same dimension, but are not completely contained by each other		<code>st_polygonize(x)</code> Creates polygon geometry from linestring geometry
	<code>st_touches(x, y, ...)</code> Identifies if geometries of x and y share a common point but their interiors do not intersect		<code>st_segmentize(x, dfMaxLength, ...)</code> Creates linestring geometry from x based on a specified length
	<code>st_within(x, y, ...)</code> Identifies if x is in a specified distance to y		<code>st_simplify(x, preserveTopology, dTolerance)</code> Creates a simplified version of the geometry based on a specified tolerance

Manipulations spatiales : sf cheat sheet

Mesures géométriques

Fonction R	Détails
<code>sf::st_length(x)</code>	Longueur de lignes spatiales
<code>sf::st_area(x)</code>	Superficie de polygones spatiaux
<code>sf::st_distance(x, y)</code>	Distance entre deux couches
<code>sf::st_bbox(x)</code>	Etentue spatiale (emprise) d'une couche

Opérations géométriques unitaires

Fonction R	Détails
<code>sf::st_buffer(x)</code>	Buffers autour d'une géométrie
<code>sf::st_boundary(x)</code>	Limites d'une géométrie
<code>sf::st_convex_hull(x)</code>	Convex hull d'un ensemble de points
<code>sf::st_centroid(x)</code>	Centroïde d'une géométrie

Manipulations spatiales : sf cheat sheet

Opérations géométriques binaires

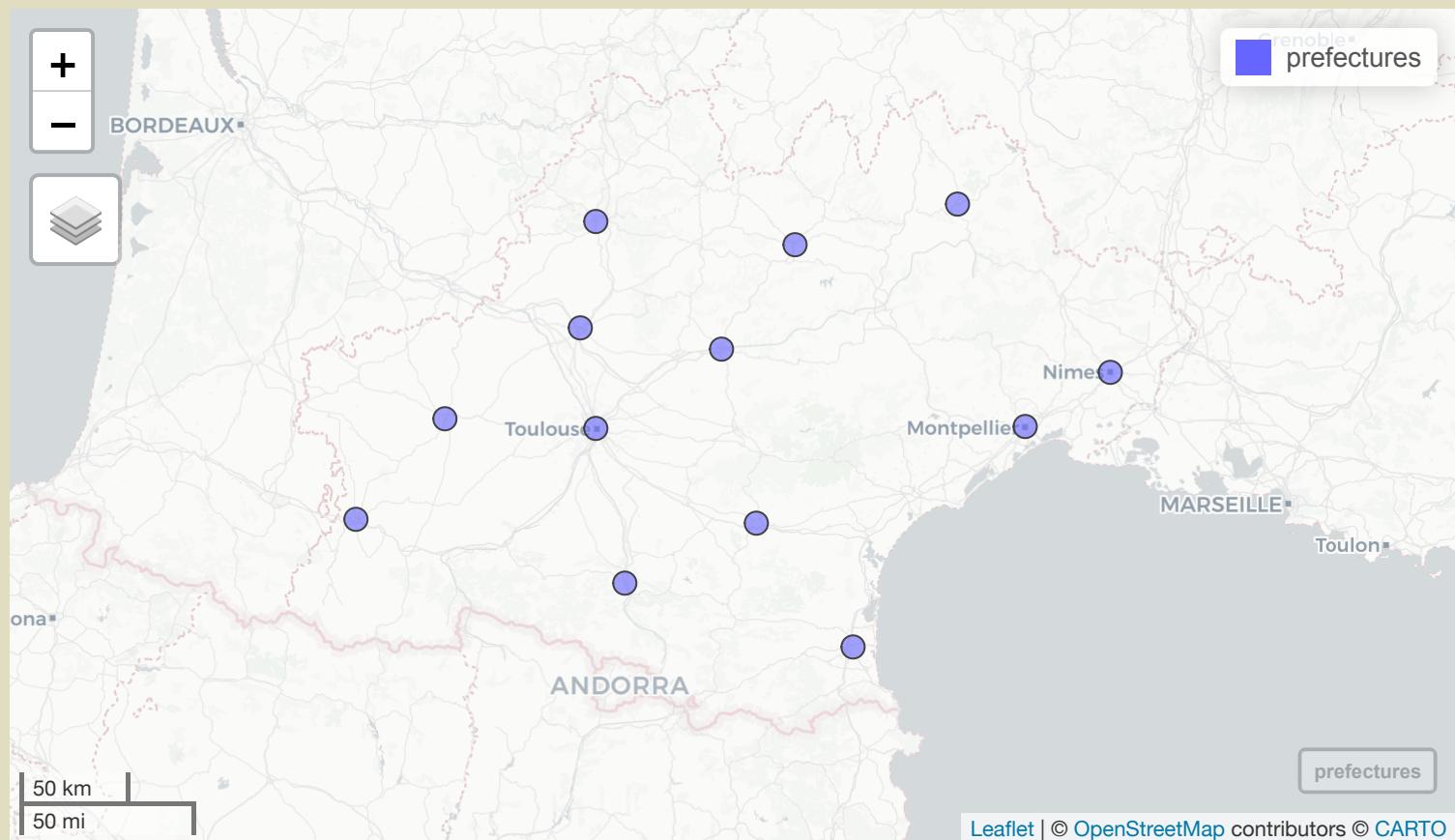
- `sf::st_intersects(x, y)`
- `sf::st_disjoint(x, y)`
- `sf::st_contains(x, y)`
- `sf::st_overlaps(x, y)`
- ...

Autres opérations

- `sf::st_crop()`
- `sf::st_intersection()`
- `sf::st_difference()`
- `sf::st_contains()`
- `sf::st_union()`

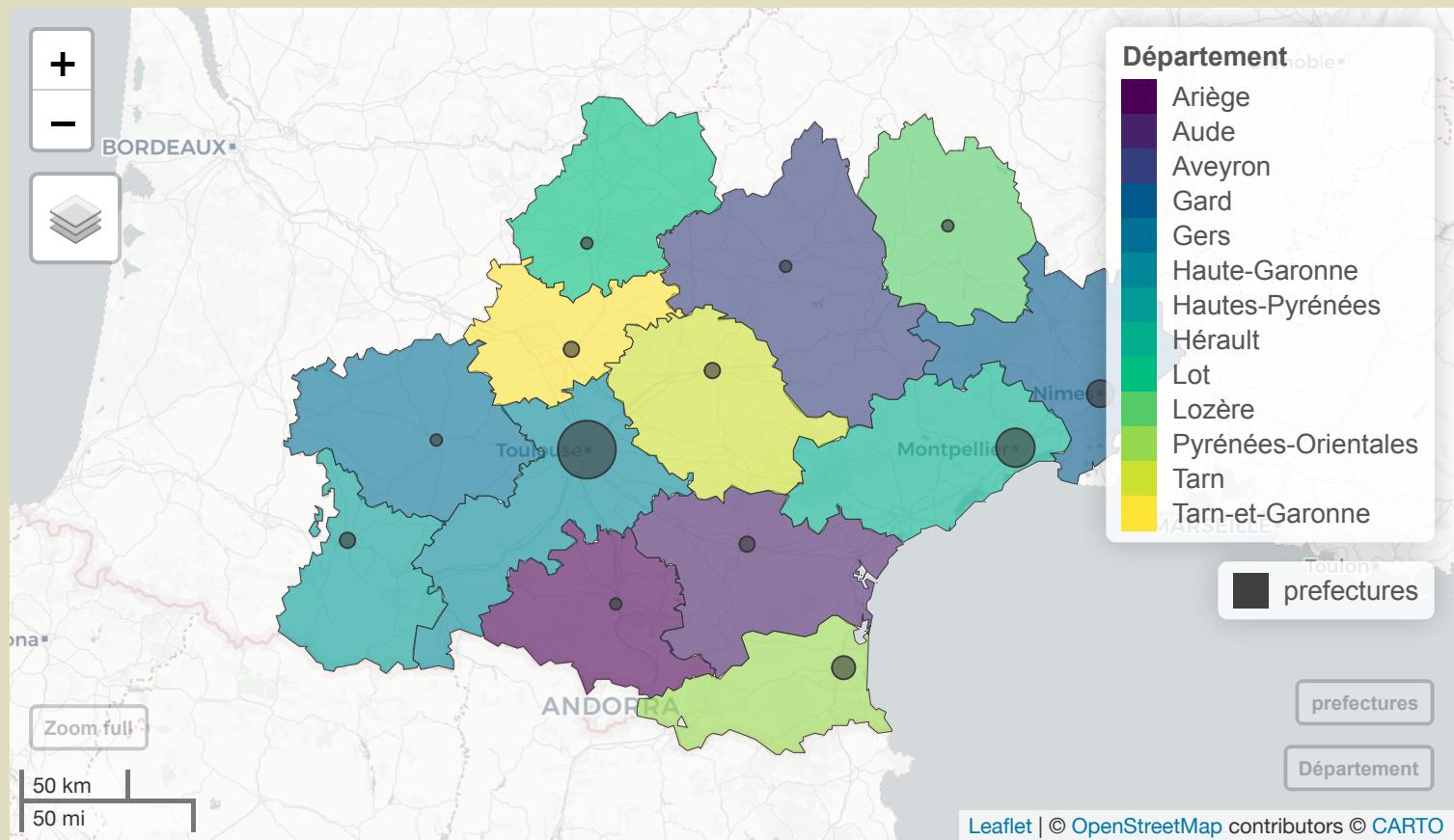
```
ls("package:sf")
```

Interactivité



```
mapview::mapview(prefectures)
```

Interactivité



```
mapview::mapview(occitanie, zcol = "NAME_2", layer.name = "Département") +  
mapview::mapview(prefectures, col.regions = "#3f3f3f", cex = "population")
```

Quelques bonnes ressources

- **Spatial Data Science with R**
- **sf vignettes**
- **Introduction to Geospatial Raster and Vector Data with R**
- **StatnMap**
- **Raster analysis in R**
- **Geocomputation with R**
- **ggplot2 et sf**
- **Très bonne introduction d'Olivier**