

Géomatique avec R

Timothée Giraud

Hugues Pecout

8 oct. 2024

Table des matières

Préambule	6
I Les données vectorielles : le package sf	8
1 Le package sf	9
1.1 Présentation	9
1.2 Format des objets <code>sf</code>	11
1.3 Ressources	11
Exercice	12
2 Import et export	13
2.1 Import	13
2.2 Export	14
Exercice	15
3 Exploration et affichage	16
3.1 Aperçu des variables	16
3.2 Affichage	18
Exercice	21
4 Les systèmes de coordonnées	22
4.1 Consulter le système de coordonnées d'un objet	22
4.2 Modifier le système de coordonnées d'un objet	23
5 Sélection et jointure attributaire	25
5.1 Sélection par attributs	25
5.2 Jointure attributaire	26
Exercice	28
6 Sélection et jointure spatiale	29
6.1 Sélection spatiale	29
6.2 Jointure spatiale	30
Exercice	31

7 Opérations sur les géométries	32
7.1 Extraire des centroïdes	32
7.2 Agréger des polygones	33
7.3 Agréger des polygones en fonction d'une variable	34
7.4 Construire une zone tampon	36
7.5 Réaliser une intersection	38
7.6 Créer une grille régulière	40
7.7 Compter des points dans un polygone	42
7.8 Agréger les valeurs de points dans des polygones	44
7.9 Simplifier des géométries	47
Exercice	48
8 Mesures	49
8.1 Créer une matrice de distances	49
8.2 Calcul de superficies	49
8.3 Convertir des unités	50
II Les données raster : le package terra	52
9 Le package terra	53
9.1 Présentation	53
9.2 Format des objets <code>SpatRaster</code>	53
10 Import et export	55
10.1 Import	55
10.2 Export	55
11 Affichage	57
12 Modifications de la zone d'étude	62
12.1 Projections	62
12.2 Crop	63
12.3 Mask	65
12.4 Agrégation & désagrégation	67
12.5 Fusion de raster	69
12.6 Segregate	72
13 Algèbre spatiale	74
13.1 Opérations locales	75
13.1.1 Remplacement de valeur	75
13.1.2 Opération sur chaque cellule	75
13.1.3 Reclassification	76
13.1.4 Opération sur plusieurs couches (ex: NDVI)	78

13.2 Opérations focales	82
13.2.1 Opération focales pour rasters d'élévation	83
13.3 Opérations globales	85
13.4 Opérations zonales	89
13.4.1 Opération zonale à partir d'une couche vectorielle	89
13.4.2 Opération zonale à partir d'un raster	90
14 Conversions	92
14.1 Rasterisation	92
14.2 Vectorisation	95
III Focus sur OpenStreetMap	99
15 OpenStreetMap	100
16 Cartes interactives	101
16.1 leaflet	101
16.2 mapview	102
17 Import de fonds de carte	104
18 Import de données	108
18.1 osmdata	108
18.2 osmextract	110
19 Matrices de temps et itinéraires	114
19.1 Calcul d'un itinéraire	114
19.2 Calcul d'une matrice de temps	115
IV Acquisition de données	118
20 Géocodage	119
20.1 Géocodage d'adresse avec tidygeocoder	119
20.2 Transformer des données long/lat en objet sf	119
20.3 Affichage sur un fond OpenStreetMap	120
21 Digitalisation	122
22 Packages de données spatiales	123
22.1 À l'échelle mondiale	123
22.2 À l'échelle européenne	124
22.3 À l'échelle nationale	124

References	126
Appendices	130
A Les données du projet	130

Préambule

Ce manuel est destiné tant aux utilisateurs de R souhaitant mettre en place des traitements de données spatiales qu'aux utilisateurs souhaitant utiliser R pour réaliser les tâches qu'ils réalisent habituellement avec un SIG.

Les principales étapes du traitement de l'information géographiques y sont abordées. L'accent est porté sur le traitement des données vectorielles mais une partie est tout de même dédiée aux données raster.

Comment utiliser le manuel

Les données utilisées dans ce document sont stockées dans un projet RStudio. Vous devez le télécharger puis le décompresser sur votre machine. Il vous sera ensuite possible de tester l'ensemble des manipulations proposées dans ce document au sein du projet **geodata**.

[Télécharger le projet](#)

Contribution et feedback

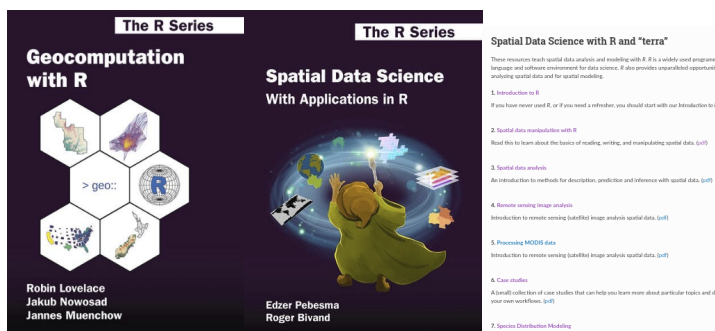
Vous pouvez nous envoyer vos remarques et suggestions en [postant une issue](#) sur le [dépôt GitHub](#) de ce document.

Contexte

Ce manuel a été initialement conçu pour accompagner le cours "Cartographie avec R" du Master 2 Géomatique, géodécisionnel, géomarketing et multimédia (G2M) de l'Université Paris 8 Vincennes - Saint-Denis. Une version PDF est disponible ici : [📄](#).

Un deuxième manuel centré sur la cartographie est disponible ici : [Cartographie avec R](#).

Voir aussi en anglais :



(a) Lovelace et al (2019) (a) Pebesma et Bivand (2023) (a) Hijmans (2023a)

Pour citer le document :

Giraud, T. et Pecout, H. (2024). Géomatique avec R. <https://doi.org/10.5281/zenodo.5906212>



partie I

Les données vectorielles : le package sf

1 Le package sf

1.1 Présentation

Le package `sf` (Pebesma, 2018) a été publié fin 2016 par Edzer Pebesma. Ce package permet l'import, l'export, la manipulation et l'affichage de données spatiales vectorielles. Pour cela `sf` s'appuie sur une série de bibliothèques spatiales : GDAL (GDAL/OGR contributors, 2022) et PROJ (PROJ contributors, 2021) pour les opérations d'import, d'export et de projection, et GEOS (GEOS contributors, 2021) pour les opérations de géotraitement (buffer, intersection...). Ce package propose des objets simples (suivant le standard *simple feature*) dont la manipulation est assez aisée. Une attention particulière a été portée à la compatibilité du package avec la syntaxe *pipe* (`|>` ou `%>%`) et les opérateurs du `tidyverse` (Wickham et al., 2019).

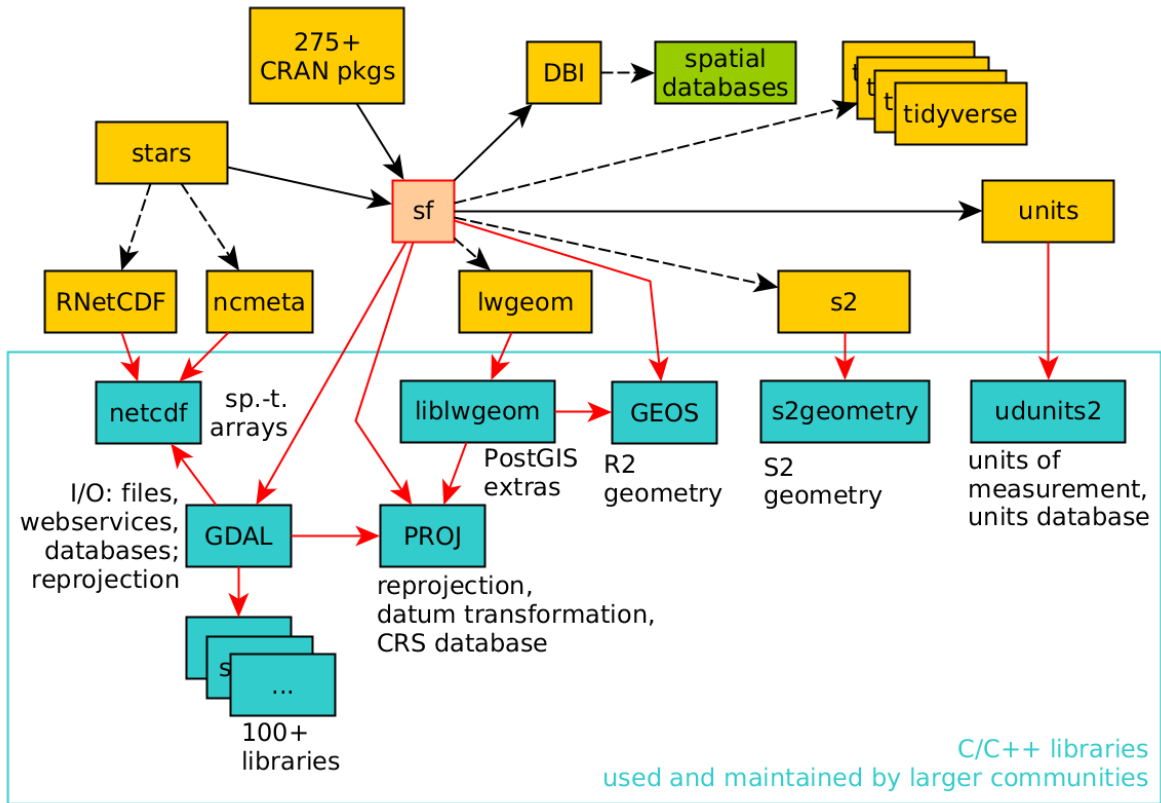


Figure 1.1: Pebesma et Bivand (2023)

Historique

Le package `sf` est venu remplacer les packages `sp` (Pebesma et Bivand, 2005), `rgeos` (Bivand et Rundel, 2023) et `rgdal` (Bivand et al., 2023) en combinant leurs fonctionnalités dans un package unique plus ergonomique. Sur ce sujet on peut lire avec profit l'article de Bivand (2021) qui évoque l'évolution de l'écosystème spatial de R.

1.2 Format des objets 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 objets `sf` sont des `data.frame` dont l'une des colonnes contient des géométries. Cette colonne est de la classe `sfc` (*simple feature column*) et chaque individu de la colonne est un `sfg` (*simple feature geometry*).

Ce format est très pratique dans la mesure où les données et les géométries sont intrinsèquement liées dans un même objet.

1.3 Ressources

sf 1.0-15 Reference Articles Changelog Search

Simple Features for R

A package that provides [simple features access](#) for R.

[Blogs](#) · [links](#) · [cheatsheet](#) · [installing](#) · [contributing](#) · [acknowledgment](#) · [how to cite](#)

Package sf:

- represents simple features as records in a `data.frame` or `tibble` with a geometry list-column
- represents natively in R all 17 simple feature types for all dimensions (XY, XYZ, XYM, XYZM)
- interfaces to [GEOS](#) for geometrical operations on projected coordinates, and (through R package [s2](#)) to [s2geometry](#) for geometrical operations on ellipsoidal coordinates
- interfaces to [GDAL](#), supporting all driver options, `date` and `POSIXct` and list-columns
- interfaces to [PROJ](#) for coordinate reference system conversion and transformation
- uses [well-known-binary](#) serialisations written in C++/Rcpp for fast I/O with GDAL and GEOS
- reads from and writes to spatial databases such as [PostGIS](#) using [DBI](#)
- is extended by
 - [lwgeom](#) for selected liblwgeom/PostGIS functions
 - [stars](#) for raster data, and faster or vector data cubes (spatial time series)
 - [sfnetworks](#) for geospatial network data

Links

[View on CRAN](#)
[Browse source code](#)
[Report a bug](#)

License

GPL-2 | MIT + file LICENSE

Citation

[Citing sf](#)

Developers

Edzer Pebesma
 Author, maintainer
[More about authors...](#)

Dev status

CRAN checks: [passed](#)
 R for Windows: [passed](#)
 R for Linux: [passed](#)
 R for macOS: [passed](#)
 R for Solaris: [passed](#)
 R for FreeBSD: [passed](#)
 R for OpenBSD: [passed](#)
 R for NetBSD: [passed](#)
 R for DragonFly: [passed](#)
 R for AIX: [passed](#)
 R for Hurd: [passed](#)
 R for IRIX: [passed](#)
 R for OS/2: [passed](#)
 R for Symbian: [passed](#)
 R for VxWorks: [passed](#)

(a) Site web de sf

Spatial manipulation with sf: CHEAT SHEET

Geometric confirmation

- `sf_is_valid()` - Checks if a geometry is valid
- `sf_is_valid_reason()` - Returns the reason for a geometry being invalid
- `sf_is_valid_geometry()` - Checks if a geometry is valid and returns the reason if not
- `sf_is_valid_geometry_reason()` - Returns the reason for a geometry being invalid
- `sf_is_valid_geometry_reason_geometry()` - Returns the reason for a geometry being invalid and the geometry
- `sf_is_valid_geometry_reason_geometry_geometry()` - Returns the reason for a geometry being invalid, the geometry, and the reason
- `sf_is_valid_geometry_reason_geometry_geometry_geometry()` - Returns the reason for a geometry being invalid, the geometry, the reason, and the geometry
- `sf_is_valid_geometry_reason_geometry_geometry_geometry_geometry()` - Returns the reason for a geometry being invalid, the geometry, the reason, the geometry, and the reason
- `sf_is_valid_geometry_reason_geometry_geometry_geometry_geometry_geometry()` - Returns the reason for a geometry being invalid, the geometry, the reason, the geometry, the reason, and the geometry

Geometric operations

- `sf_buffer()` - Buffer a geometry
- `sf_buffer_by_dist()` - Buffer a geometry by distance
- `sf_buffer_by_dist_geometry()` - Buffer a geometry by distance and returns the geometry
- `sf_buffer_by_dist_geometry_geometry()` - Buffer a geometry by distance and returns the geometry and the reason
- `sf_buffer_by_dist_geometry_geometry_geometry()` - Buffer a geometry by distance and returns the geometry, the reason, and the geometry
- `sf_buffer_by_dist_geometry_geometry_geometry_geometry()` - Buffer a geometry by distance and returns the geometry, the reason, the geometry, and the reason
- `sf_buffer_by_dist_geometry_geometry_geometry_geometry_geometry()` - Buffer a geometry by distance and returns the geometry, the reason, the geometry, the reason, and the geometry
- `sf_buffer_by_dist_geometry_geometry_geometry_geometry_geometry_geometry()` - Buffer a geometry by distance and returns the geometry, the reason, the geometry, the reason, the geometry, and the reason
- `sf_buffer_by_dist_geometry_geometry_geometry_geometry_geometry_geometry_geometry()` - Buffer a geometry by distance and returns the geometry, the reason, the geometry, the reason, the geometry, the reason, and the geometry

Geometry creation

- `sf_from_wkt()` - Create a geometry from WKT
- `sf_from_wkt_geometry()` - Create a geometry from WKT and returns the geometry
- `sf_from_wkt_geometry_geometry()` - Create a geometry from WKT and returns the geometry and the reason
- `sf_from_wkt_geometry_geometry_geometry()` - Create a geometry from WKT and returns the geometry, the reason, and the geometry
- `sf_from_wkt_geometry_geometry_geometry_geometry()` - Create a geometry from WKT and returns the geometry, the reason, the geometry, and the reason
- `sf_from_wkt_geometry_geometry_geometry_geometry_geometry()` - Create a geometry from WKT and returns the geometry, the reason, the geometry, the reason, and the geometry
- `sf_from_wkt_geometry_geometry_geometry_geometry_geometry_geometry()` - Create a geometry from WKT and returns the geometry, the reason, the geometry, the reason, the geometry, and the reason
- `sf_from_wkt_geometry_geometry_geometry_geometry_geometry_geometry_geometry()` - Create a geometry from WKT and returns the geometry, the reason, the geometry, the reason, the geometry, the reason, and the geometry

(a) Cheat sheet

Exercice

1. Les données utilisées dans ce document sont stockées dans un projet RStudio. Vous devez le télécharger puis le décompresser sur votre machine. Il vous sera ensuite possible de tester l'ensemble des manipulations proposées dans ce document au sein du projet **geodata**.
[Télécharger le projet](#)
2. Nous utiliserons principalement le package `sf` dans la suite de ce document. Installez le en utilisant la fonction `install.packages()`.

2 Import et export

Les fonctions `st_read()` et `st_write()` permettent d'importer et d'exporter de nombreux types de fichiers.

2.1 Import

Les lignes suivantes importent la couche des communes du département du Lot mise à disposition dans le fichier [geopackage lot.gpkg](#).

```
library(sf)
```

```
#> Linking to GEOS 3.11.1, GDAL 3.6.2, PROJ 9.1.1; sf_use_s2() is TRUE
```

```
com <- st_read("data/lot.gpkg", layer = "communes")
```

```
#> Reading layer `communes' from data source  
#>   `/home/tim/Documents/prj/geomatique_avec_r/data/lot.gpkg' using driver `GPKG'  
#> Simple feature collection with 313 features and 12 fields  
#> Geometry type: MULTIPOLYGON  
#> Dimension:      XY  
#> Bounding box:  xmin: 539668.5 ymin: 6346290 xmax: 637380.9 ymax: 6439668  
#> Projected CRS: RGF93 v1 / Lambert-93
```

```
class(com)
```

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

La fonction `st_read()` importe les couches géographiques au format `sf`.

Geopackage

Le format [geopackage](#) permet de stocker plusieurs couches dans un même fichier.

La fonction `st_layers()` permet d'avoir un aperçu des couches présentes dans un fichier geopackage.

```
st_layers("data/lot.gpkg")
```

```
#> Driver: GPKG
#> Available layers:
#>   layer_name geometry_type features fields      crs_name
#> 1 communes Multi Polygon    313    12 RGF93 v1 / Lambert-93
#> 2 departements Multi Polygon    96     5 RGF93 v1 / Lambert-93
#> 3 restaurants      Point    694     2 RGF93 v1 / Lambert-93
#> 4 elevations      Point   5228     1 RGF93 v1 / Lambert-93
#> 5 routes      Line String  1054     2 RGF93 v1 / Lambert-93
```

2.2 Export

Les lignes suivantes exportent l'objet `com` dans la couche `communes` du geopackage `com.gpkg` dans le dossier `data`.

```
st_write(obj = com, dsn = "data/com.gpkg", layer = "communes")
```

```
#> Writing layer `communes` to data source `data/com.gpkg` using driver `GPKG`
#> Writing 313 features with 12 fields and geometry type Multi Polygon.
```

Conversion pour le package terra

La fonction `vect()` du package `terra` permet de transformer un objet `sf` en objet `SpatVector`.

```
library(terra)
```

```
#> terra 1.7.78
```

```
# conversion
com2 <- vect(com)
class(com2)
```

```
#> [1] "SpatVector"
#> attr(,"package")
#> [1] "terra"
```

Exercice

Importez la couche des restaurants depuis le fichier **lot.gpkg** dans un objet nommé "resto".

3 Exploration et affichage

3.1 Aperçu des variables

Les objets `sf` sont des `data.frame`.

Nous pouvons utiliser les fonctions `head()` ou `summary()`.

```
library(sf)
com <- st_read("data/lot.gpkg", layer = "communes", quiet = TRUE)
head(com, n = 3)
```

```
#> Simple feature collection with 3 features and 12 fields
#> Geometry type: MULTIPOLYGON
#> Dimension:      XY
#> Bounding box:  xmin: 557759.2 ymin: 6371852 xmax: 607179 ymax: 6418606
#> Projected CRS: RGF93 v1 / Lambert-93
#>   INSEE_COM  NOM_COM          STATUT POPULATION    AGR_H    AGR_F    IND_H
#> 1    46001    Albas Commune simple      522  4.978581 0.000000  4.936153
#> 2    46002    Albiac Commune simple      67  0.000000 9.589041  0.000000
#> 3    46003    Alvignac Commune simple    706 10.419682 0.000000 10.419682
#>   IND_F    BTP_H BTP_F    TER_H    TER_F                                geom
#> 1 0.000000  9.957527    0 44.917145 34.681799 MULTIPOLYGON (((559262 6371...
#> 2 0.000000  4.794521    0  4.794521  9.589041 MULTIPOLYGON (((605540.7 64...
#> 3 5.209841 10.419682    0 57.308249 78.147612 MULTIPOLYGON (((593707.7 64...
```

```
summary(com)
```

```
#>   INSEE_COM          NOM_COM          STATUT          POPULATION
#> Length:313      Length:313      Length:313      Min.   :   49.0
#> Class :character  Class :character  Class :character  1st Qu.:  172.0
#> Mode  :character  Mode  :character  Mode  :character  Median :  300.0
#>                                     Mean   :  555.7
#>                                     3rd Qu.:  529.0
#>                                     Max.   :19907.0
#>   AGR_H          AGR_F          IND_H          IND_F
```



```

#> Min. : 0.000 Min. : 0.000 Min. : 0.000 Min. : 0.000
#> 1st Qu.: 0.000 1st Qu.: 0.000 1st Qu.: 4.843 1st Qu.: 0.000
#> Median : 5.000 Median : 0.000 Median : 5.516 Median : 4.943
#> Mean : 6.935 Mean : 2.594 Mean : 16.395 Mean : 7.635
#> 3rd Qu.:10.013 3rd Qu.: 5.000 3rd Qu.: 19.715 3rd Qu.: 9.905
#> Max. :56.179 Max. :24.641 Max. :602.867 Max. :184.016
#> BTP_H BTP_F TER_H TER_F
#> Min. : 0.000 Min. : 0.0000 Min. : 0.00 Min. : 0.00
#> 1st Qu.: 0.000 1st Qu.: 0.0000 1st Qu.: 10.00 1st Qu.: 15.15
#> Median : 5.000 Median : 0.0000 Median : 20.00 Median : 30.26
#> Mean : 9.572 Mean : 0.9723 Mean : 42.17 Mean : 60.77
#> 3rd Qu.: 10.329 3rd Qu.: 0.0000 3rd Qu.: 44.69 3rd Qu.: 63.95
#> Max. :203.122 Max. :16.9238 Max. :1778.87 Max. :2397.18
#> geom
#> MULTIPOLYGON :313
#> epsg:2154 : 0
#> +proj=lcc ...: 0
#>
#>
#>

```

Supprimer la colonne de géométrie d'un objet sf

Pour transformer un objet sf en simple data.frame (sans géométries), nous pouvons utiliser les fonctions `st_set_geometry()` ou `st_drop_geometry()`.

```

com_df1 <- st_set_geometry(com, NULL)
com_df2 <- st_drop_geometry(com)
identical(com_df1, com_df2)

```

```

#> [1] TRUE

```

```

head(com_df1, n = 3)

```

```

#> INSEE_COM NOM_COM STATUT POPULATION AGR_H AGR_F IND_H
#> 1 46001 Albas Commune simple 522 4.978581 0.000000 4.936153
#> 2 46002 Albiac Commune simple 67 0.000000 9.589041 0.000000
#> 3 46003 Alvignac Commune simple 706 10.419682 0.000000 10.419682
#> IND_F BTP_H BTP_F TER_H TER_F
#> 1 0.000000 9.957527 0 44.917145 34.681799
#> 2 0.000000 4.794521 0 4.794521 9.589041
#> 3 5.209841 10.419682 0 57.308249 78.147612

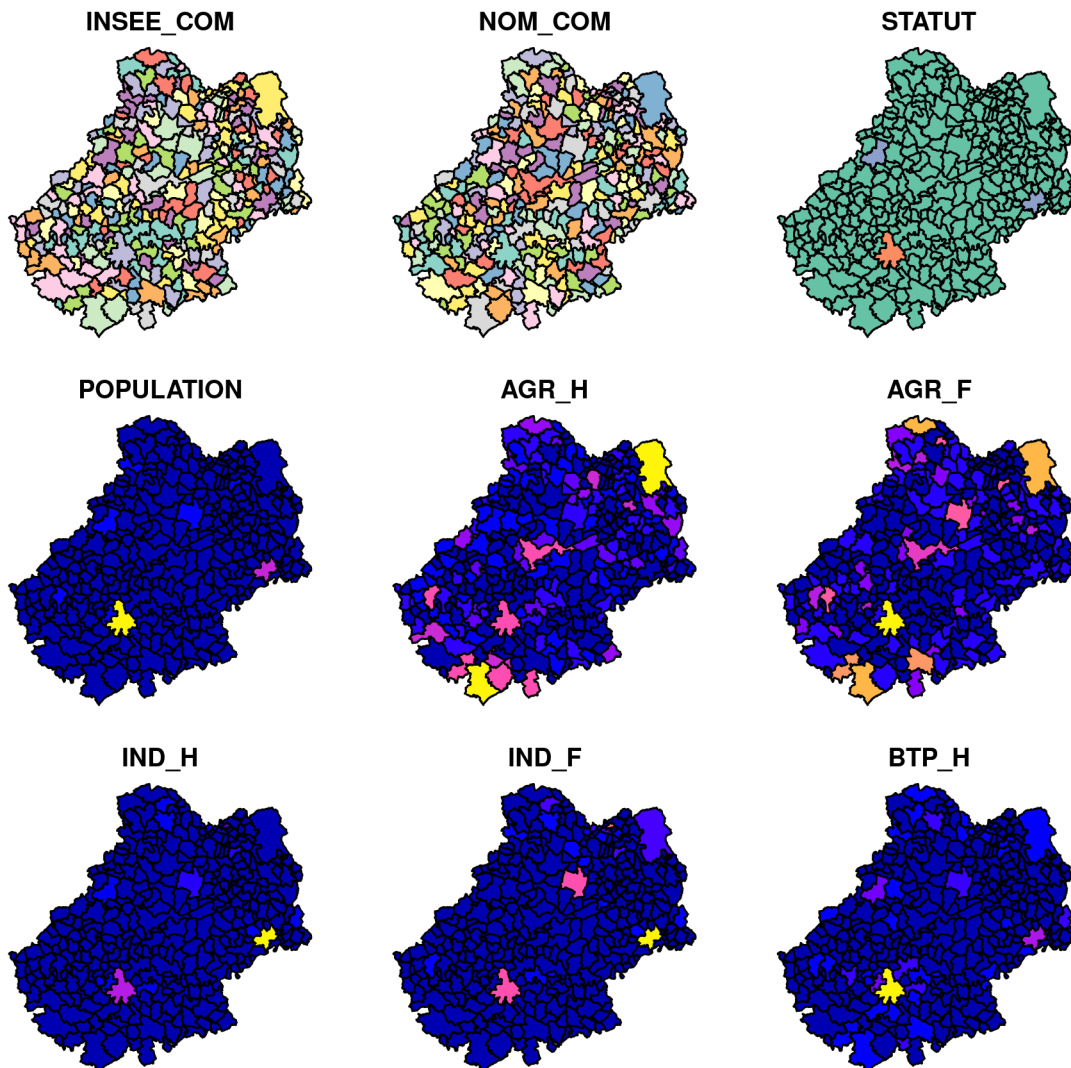
```

3.2 Affichage

Aperçu des variables avec `plot()` :

```
plot(com)
```

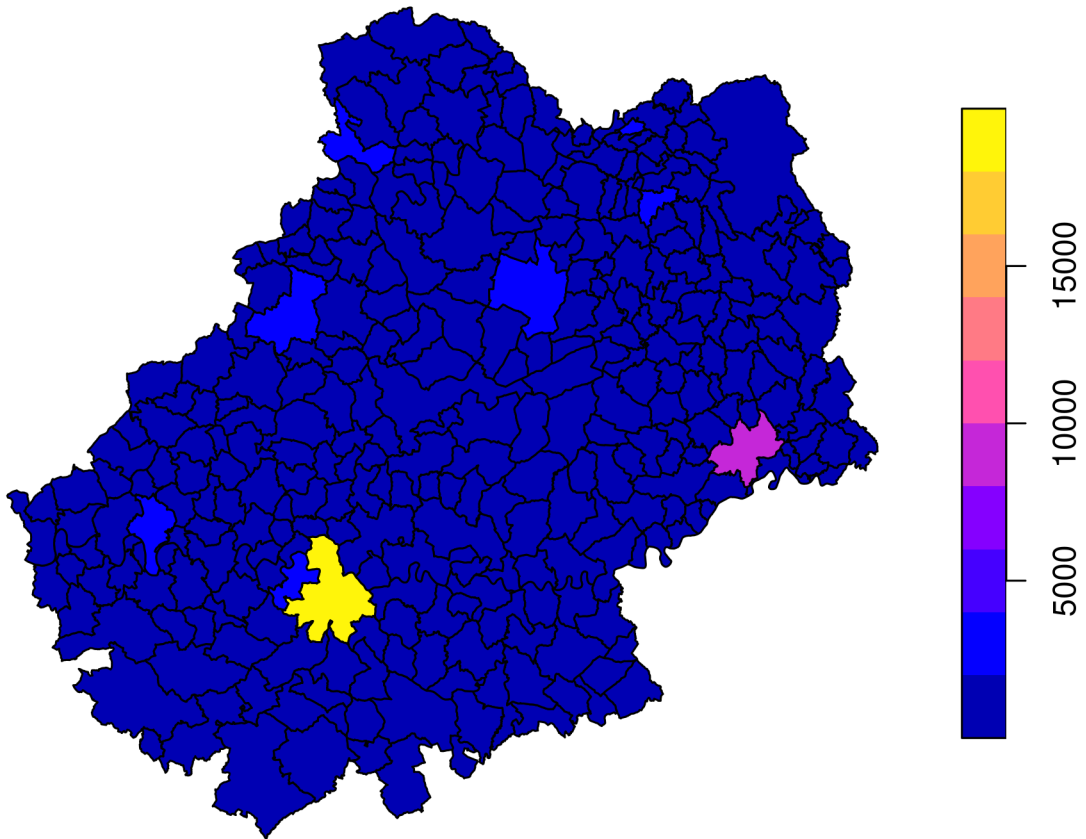
```
#> Warning: plotting the first 9 out of 12 attributes; use max.plot = 12 to plot  
#> all
```



Affichage d'une seule variable :

```
plot(com["POPULATION"])
```

POPULATION



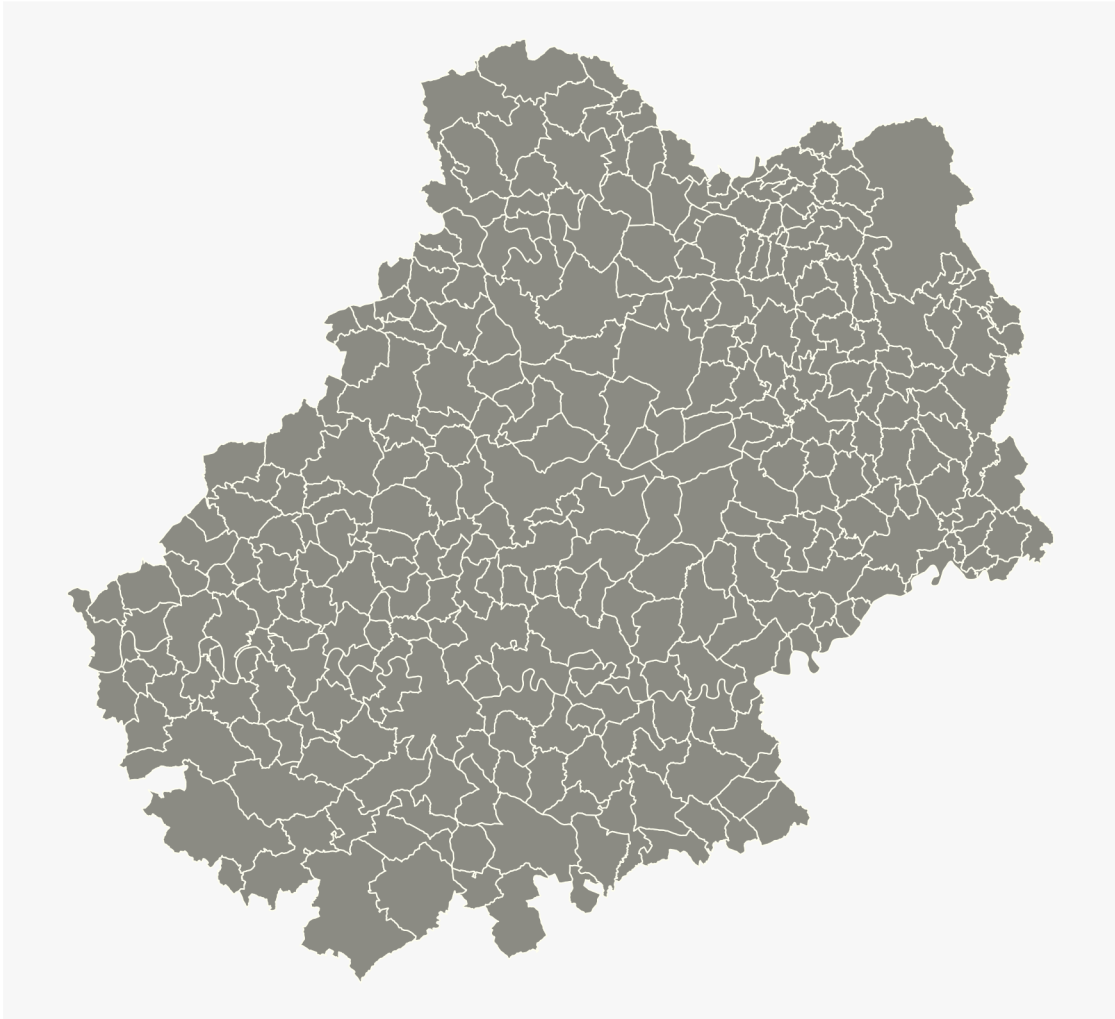
Affichage de la géométrie seule :

```
plot(st_geometry(com), col = "ivory4", border = "ivory")
```



Il est aussi possible d'utiliser le package `mapsf` (Giraud, 2023a) pour afficher les objets `sf`.

```
library(mapsf)
mf_map(com, col = "ivory4", border = "ivory")
```



Exercice

Affichez les communes et les restaurants sur une même carte.

4 Les systèmes de coordonnées

4.1 Consulter le système de coordonnées d'un objet

La fonction `st_crs()` permet de consulter le système de coordonnées utilisé par un objet `sf`.

```
library(sf)
st_crs(x = com)
```

```
#> Coordinate Reference System:
#>   User input: RGF93 v1 / Lambert-93
#>   wkt:
#> PROJCRS["RGF93 v1 / Lambert-93",
#>   BASEGEOGCRS["RGF93 v1",
#>     DATUM["Réseau Géodésique Français 1993 v1",
#>       ELLIPSOID["GRS 1980",6378137,298.257222101,
#>         LENGTHUNIT["metre",1]]],
#>     PRIMEM["Greenwich",0,
#>       ANGLEUNIT["degree",0.0174532925199433]],
#>     ID["EPSG",4171]],
#>   CONVERSION["Lambert-93",
#>     METHOD["Lambert Conic Conformal (2SP)",
#>       ID["EPSG",9802]],
#>     PARAMETER["Latitude of false origin",46.5,
#>       ANGLEUNIT["degree",0.0174532925199433],
#>       ID["EPSG",8821]],
#>     PARAMETER["Longitude of false origin",3,
#>       ANGLEUNIT["degree",0.0174532925199433],
#>       ID["EPSG",8822]],
#>     PARAMETER["Latitude of 1st standard parallel",49,
#>       ANGLEUNIT["degree",0.0174532925199433],
#>       ID["EPSG",8823]],
#>     PARAMETER["Latitude of 2nd standard parallel",44,
#>       ANGLEUNIT["degree",0.0174532925199433],
#>       ID["EPSG",8824]],
#>     PARAMETER["Easting at false origin",700000,
```

```

#>           LENGTHUNIT["metre",1],
#>           ID["EPSG",8826]],
#>     PARAMETER["Northing at false origin",6600000,
#>           LENGTHUNIT["metre",1],
#>           ID["EPSG",8827]]],
#>   CS[Cartesian,2],
#>     AXIS["easting (X)",east,
#>       ORDER[1],
#>       LENGTHUNIT["metre",1]],
#>     AXIS["northing (Y)",north,
#>       ORDER[2],
#>       LENGTHUNIT["metre",1]],
#>   USAGE[
#>     SCOPE["Engineering survey, topographic mapping."],
#>     AREA["France - onshore and offshore, mainland and Corsica (France métropolitaine :
#>     BBOX[41.15,-9.86,51.56,10.38]],
#>     ID["EPSG",2154]]

```

4.2 Modifier le système de coordonnées d'un objet

La fonction `st_transform()` permet de changer le système de coordonnées d'un objet `sf`, de le reprojeter.

```

mf_map(com, expandBB = c(0, .12, 0, 0))
mf_graticule(x = com)
mf_title("RGF93 / Lambert-93")
# changement de projection
com_reproj <- st_transform(x = com, crs = "EPSG:3035")

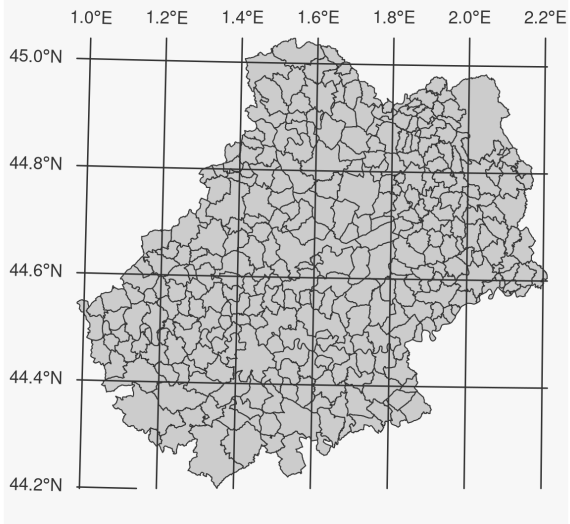
mf_map(com_reproj, expandBB = c(0, .12, .0, 0))
mf_graticule(x = com_reproj)
mf_title("ETRS89-extended / LAEA Europe")

```

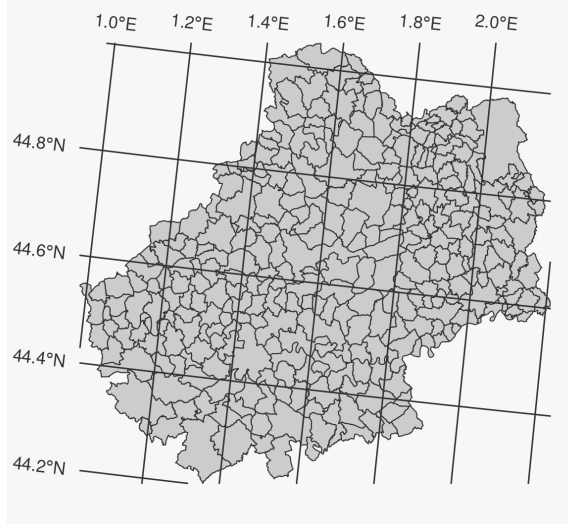
Si l'on souhaite projeter un objet spatial utilisant des coordonnées géographiques (lon/lat), le package `crsuggest` (Walker, 2022) propose des projections adaptées à l'emprise de l'objet.

Le site [CRS Explorer](#) met à disposition les références de très nombreux systèmes de coordonnées.

RGF93 / Lambert-93



ETRS89-extended / LAEA Europe



5 Sélection et jointure attributaire

5.1 Sélection par attributs

Les objets `sf` sont des `data.frame`, on peut donc sélectionner leur lignes et leur colonnes de la même manière que les `data.frame`.

```
# sélection de lignes
com[1:2, ]
```

```
#> Simple feature collection with 2 features and 12 fields
#> Geometry type: MULTIPOLYGON
#> Dimension: XY
#> Bounding box: xmin: 557759.2 ymin: 6371852 xmax: 607179 ymax: 6410204
#> Projected CRS: RGF93 v1 / Lambert-93
#>   INSEE_COM NOM_COM          STATUT POPULATION   AGR_H   AGR_F   IND_H IND_F
#> 1    46001   Albas Commune simple         522 4.978581 0.000000 4.936153    0
#> 2    46002  Albiac Commune simple          67 0.000000 9.589041 0.000000    0
#>   BTP_H BTP_F   TER_H   TER_F                                geom
#> 1 9.957527    0 44.917145 34.681799 MULTIPOLYGON (((559262 6371...
#> 2 4.794521    0  4.794521  9.589041 MULTIPOLYGON (((605540.7 64...
```

```
com[com$NOM_COM == "Gramat", ]
```

```
#> Simple feature collection with 1 feature and 12 fields
#> Geometry type: MULTIPOLYGON
#> Dimension: XY
#> Bounding box: xmin: 593605.6 ymin: 6402330 xmax: 602624.6 ymax: 6413784
#> Projected CRS: RGF93 v1 / Lambert-93
#>   INSEE_COM NOM_COM          STATUT POPULATION   AGR_H   AGR_F   IND_H
#> 119    46128  Gramat Commune simple         3468 10.19868 15.29802 122.3842
#>   IND_F   BTP_H BTP_F   TER_H   TER_F                                geom
#> 119 107.0862 56.09275    0 260.0664 304.1941 MULTIPOLYGON (((594713.1 64...
```

```
# sélection de colonnes
com[, "POPULATION"]
```

```
#> Simple feature collection with 313 features and 1 field
#> Geometry type: MULTIPOLYGON
#> Dimension: XY
#> Bounding box: xmin: 539668.5 ymin: 6346290 xmax: 637380.9 ymax: 6439668
#> Projected CRS: RGF93 v1 / Lambert-93
#> First 10 features:
#>   POPULATION          geom
#> 1      522 MULTIPOLYGON (((559262 6371...
#> 2       67 MULTIPOLYGON (((605540.7 64...
#> 3      706 MULTIPOLYGON (((593707.7 64...
#> 4      219 MULTIPOLYGON (((613211.3 64...
#> 5      329 MULTIPOLYGON (((556744.9 63...
#> 6      377 MULTIPOLYGON (((576667.2 64...
#> 7      988 MULTIPOLYGON (((581404 6370...
#> 8      203 MULTIPOLYGON (((558216 6389...
#> 9      642 MULTIPOLYGON (((612729.6 63...
#> 10     367 MULTIPOLYGON (((581404 6370...
```

```
com[com$NOM_COM == "Gramat", 1:4]
```

```
#> Simple feature collection with 1 feature and 4 fields
#> Geometry type: MULTIPOLYGON
#> Dimension: XY
#> Bounding box: xmin: 593605.6 ymin: 6402330 xmax: 602624.6 ymax: 6413784
#> Projected CRS: RGF93 v1 / Lambert-93
#>   INSEE_COM NOM_COM          STATUT POPULATION          geom
#> 119    46128  Gramat Commune simple      3468 MULTIPOLYGON (((594713.1 64...
```

5.2 Jointure attributaire

Nous pouvons joindre un `data.frame` à un objet `sf` en utilisant la fonction `merge()` et en s'appuyant sur des identifiants communs aux deux objets.

Attention à l'ordre des arguments, l'objet retourné sera du même type que `x`. Il n'est pas possible de faire une jointure attributaire en utilisant deux objets `sf`.

```
# import de données supplémentaires
com_df <- read.csv(file = "data/com.csv")

# des identifiants en commun?
names(com_df)
```

```
#> [1] "INSEE_COM" "ACT"          "IND"          "SACT"          "SACT_IND"
```

```
names(com)
```

```
#> [1] "INSEE_COM" "NOM_COM"    "STATUT"     "POPULATION" "AGR_H"
#> [6] "AGR_F"     "IND_H"     "IND_F"      "BTP_H"      "BTP_F"
#> [11] "TER_H"     "TER_F"     "geom"
```

```
# jointure attributaire
com_final <- merge(
  x = com,          # l'objet sf
  y = com_df,      # le data.frame
  by.x = "INSEE_COM", # identifiant dans x
  by.y = "INSEE_COM", # identifiant dans y
  all.x = TRUE      # conserver toutes les lignes
)
```

```
# Les deux objets ont bien été joints
head(com_final, 3)
```

```
#> Simple feature collection with 3 features and 16 fields
#> Geometry type: MULTIPOLYGON
#> Dimension: XY
#> Bounding box: xmin: 557759.2 ymin: 6371852 xmax: 607179 ymax: 6418606
#> Projected CRS: RGF93 v1 / Lambert-93
#>   INSEE_COM NOM_COM      STATUT POPULATION    AGR_H    AGR_F    IND_H
#> 1    46001   Albas Commune simple      522  4.978581 0.000000  4.936153
#> 2    46002   Albiac Commune simple      67  0.000000 9.589041  0.000000
#> 3    46003 Alvignac Commune simple     706 10.419682 0.000000 10.419682
#>   IND_F    BTP_H BTP_F    TER_H    TER_F    ACT    IND    SACT
#> 1 0.000000  9.957527    0 44.917145 34.681799 99.47120  4.936153 19.05579
#> 2 0.000000  4.794521    0  4.794521  9.589041 28.76712  0.000000 42.93600
#> 3 5.209841 10.419682    0 57.308249 78.147612 171.92475 15.629522 24.35195
#>   SACT_IND
#>          geom
```

```
#> 1 4.962393 MULTIPOLYGON (((559262 6371...
#> 2 0.000000 MULTIPOLYGON (((605540.7 64...
#> 3 9.090909 MULTIPOLYGON (((593707.7 64...
```

Exercice

1. Importer la couche des communes du département du Lot à partir du fichier geopackage **lot.gpkg**.
2. Importer le fichier **com.csv**.
Ce jeu de données porte sur les communes du Lot et contient plusieurs variables supplémentaires:
 - le nombre d'actifs (**ACT**).
 - le nombre d'actifs dans l'industrie (**IND**)
 - la part des actifs dans la population totale (**SACT**)
 - la part des actifs dans l'industrie dans le total des actifs (**SACT_IND**)
3. Joindre le jeu de données et la couche des communes.
4. Sélectionnez les communes du Lot ayant plus de 500 actifs et dont la part des actifs dans la population totale est supérieure à 30%.
5. Affichez toutes les communes en gris et les communes sélectionnées en rouge.

6 Sélection et jointure spatiale

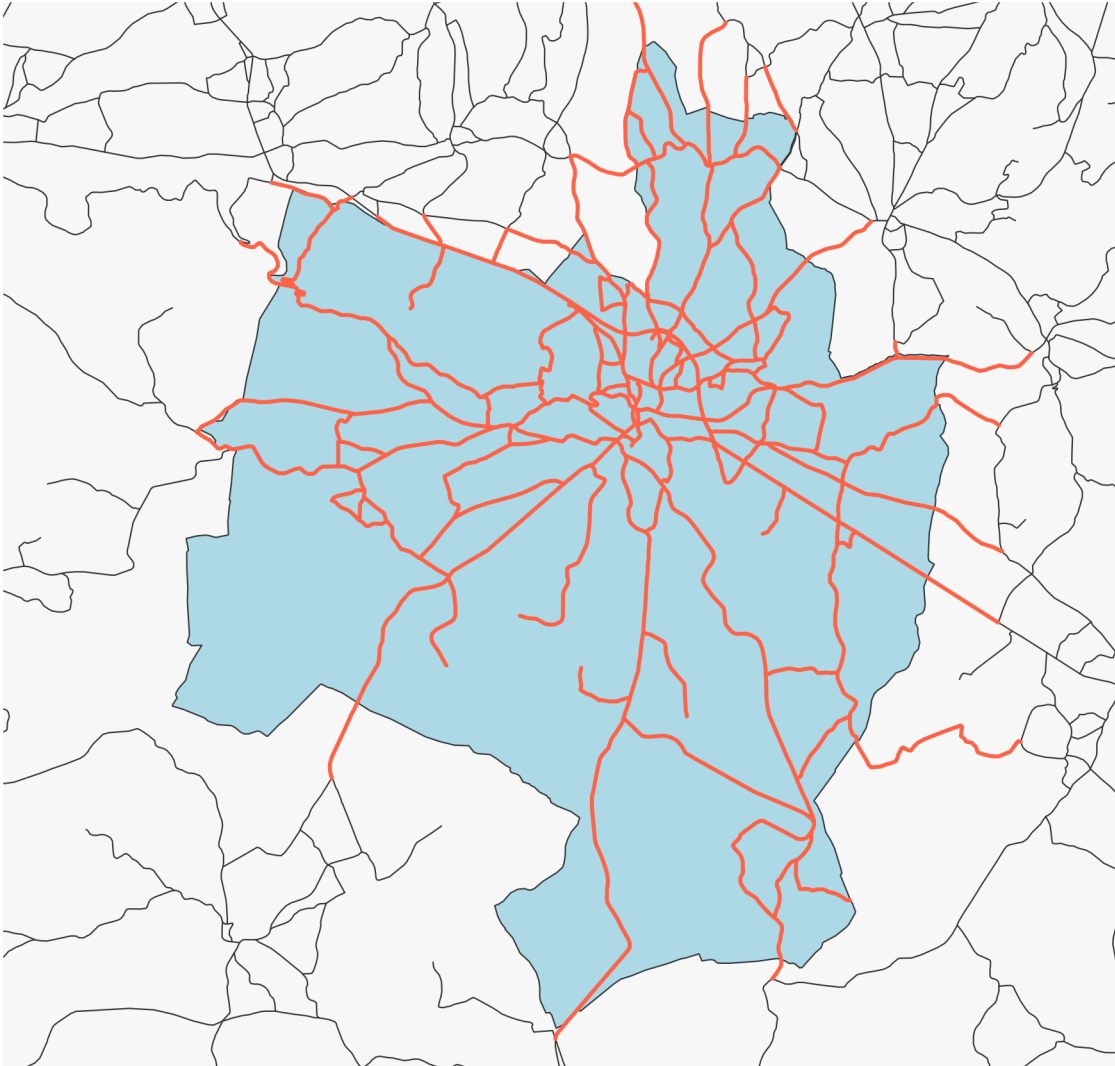
6.1 Sélection spatiale

La fonction `st_filter()` permet d'effectuer des sélections spatiales. L'argument `.predicate` permet de choisir sur quel critère se fait la sélection en utilisant l'une des fonctions de "prédicat géométrique" (par exemple `st_intersects()`, `st_within()`, `st_crosses()`...). Nous allons ici sélectionner les routes qui intersectent la commune de Gramat

```
route <- st_read("data/lot.gpkg", layer = "routes", quiet = TRUE)
gramat <- com[com$NOM_COM == "Gramat", ]

route_gramat <- st_filter(x = route,
                          y = gramat,
                          .predicate = st_intersects)

# Affichage
mf_map(gramat, col = "lightblue")
mf_map(route, add = TRUE)
mf_map(route_gramat, col = "tomato", lwd = 2, add = TRUE)
```



6.2 Jointure spatiale

La fonction `st_join()` permet de réaliser des jointures spatiales. Cette fois-ci c'est l'argument `join` qui utilise une fonction de prédicat géométrique.

```
route_gramat <- st_join(x = route,  
                        y = com[, "INSEE_COM"],  
                        join = st_intersects,  
                        left = FALSE)  
route_gramat
```

```

#> Simple feature collection with 1247 features and 3 fields
#> Geometry type: LINESTRING
#> Dimension: XY
#> Bounding box: xmin: 587147.6 ymin: 6394844 xmax: 608194.7 ymax: 6420006
#> Projected CRS: RGF93 v1 / Lambert-93
#> First 10 features:
#>   ID      CLASS_ADM INSEE_COM      geom
#> 1   1 Départementale  46240 LINESTRING (590557.5 641181...
#> 2   2 Départementale  46240 LINESTRING (593733.2 641429...
#> 3   3 Départementale  46240 LINESTRING (590665 6412381,...
#> 4   4 Départementale  46128 LINESTRING (598940.9 640909...
#> 5   5 Départementale  46104 LINESTRING (603201.9 640181...
#> 6   6      Sans objet  46235 LINESTRING (598162.3 640108...
#> 7   7 Départementale  46090 LINESTRING (598887.3 639763...
#> 7.1 7 Départementale  46138 LINESTRING (598887.3 639763...
#> 7.2 7 Départementale  46233 LINESTRING (598887.3 639763...
#> 8   8 Départementale  46090 LINESTRING (601184.3 639697...

```

Exercice

1. Importez la couche des communes et celle des restaurants du Lot.
2. Réaliser une jointure spatiale pour connaître pour chaque restaurant le nom et l'identifiant de la commune dans laquelle il se trouve.

7 Opérations sur les géométries

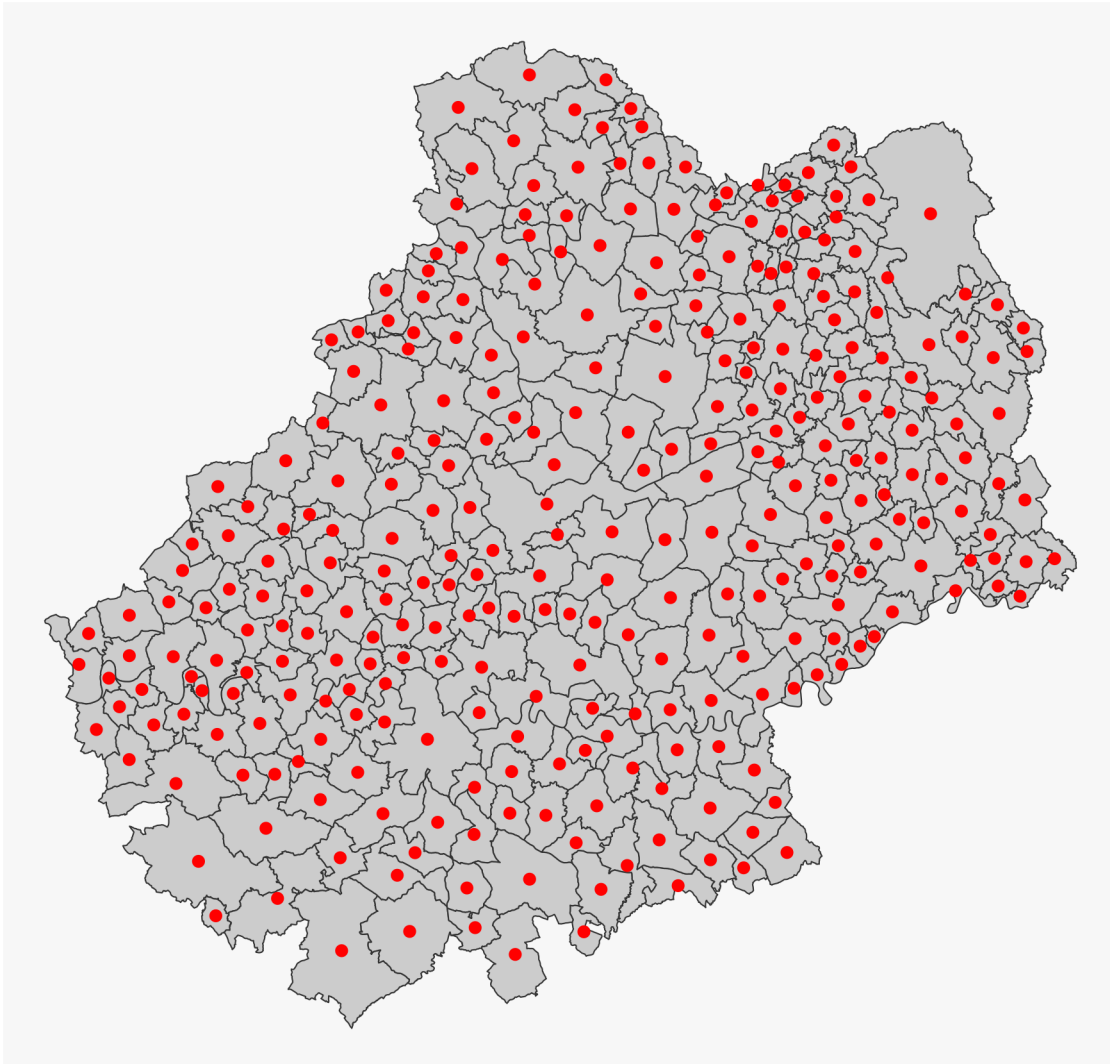
7.1 Extraire des centroïdes

```
com_c <- st_centroid(com)
```

```
#> Warning: st_centroid assumes attributes are constant over geometries
```

```
mf_map(com)
```

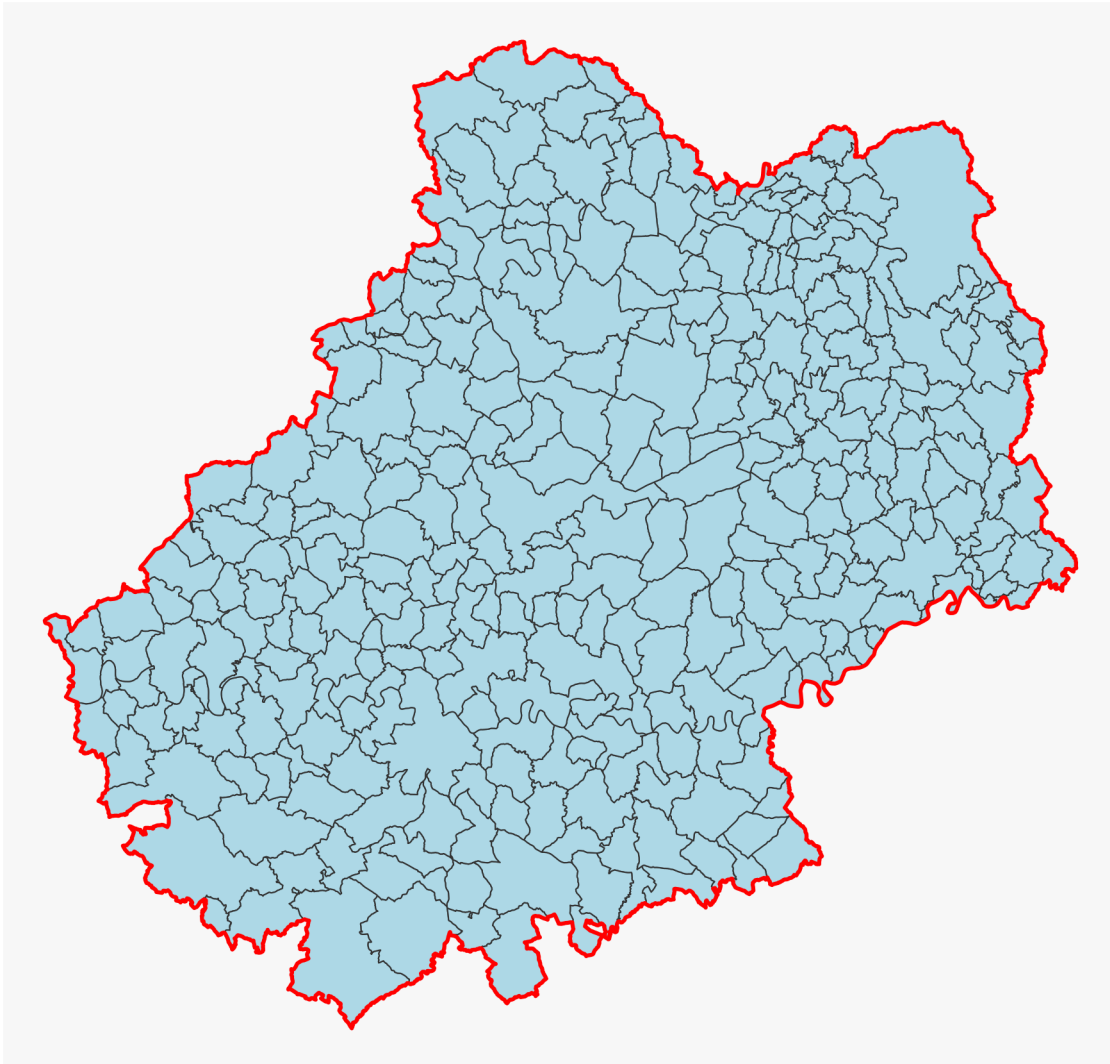
```
mf_map(com_c, add = TRUE, cex = 1.2, col = "red", pch = 20)
```

7.2 Agréger des polygones

```
dep_46 <- st_union(com)

mf_map(com, col = "lightblue")
mf_map(dep_46, col = NA, border = "red", lwd = 2, add = TRUE)
```



7.3 Agréger des polygones en fonction d'une variable

- Avec la fonction `tapply()`

```
# variable servant à agréger les polygones
i <- com$STATUT

com_u <- st_sf(
  STATUT = tapply(X = com$STATUT, INDEX = i, FUN = head, 1),
```

```
POPULATION = tapply(X = com$POPULATION , INDEX = i, FUN = sum),
geometry    = tapply(X = com          , INDEX = i, FUN = st_union),
crs         = st_crs(com)
)
```

Explication de la méthode d'agrégation

tapply(X, INDEX, FUN) permet d'aggréger une variable en fonction d'une autre. Il faut indiquer la variable à agréger **X**, la variable servant à agréger **INDEX** et la manière d'agréger (la fonction d'agrégation) **FUN**.

Ici par exemple nous calculons la somme des population des communes en fonction de leur statut :

```
tapply(X = com$POPULATION, INDEX = com$STATUT, FUN = sum)
```

```
#> Commune simple      Préfecture Sous-préfecture
#>           140259           19907           13763
```

tapply() fonctionne également avec les objets sf et sfc:

```
st_sf(geometry = st_sfc(tapply(com, com$STATUT, st_union)))
```

```
#> Simple feature collection with 3 features and 0 fields
#> Geometry type: GEOMETRY
#> Dimension: XY
#> Bounding box: xmin: 539668.5 ymin: 6346290 xmax: 637380.9 ymax: 6439668
#> CRS: NA
#> geometry
#> 1 POLYGON ((554732.7 6356281,...
#> 2 MULTIPOLYGON (((580994.5 63...
#> 3 MULTIPOLYGON (((626296.4 63...
```

Nous pouvons ensuite combiner plusieurs appels tapply() à l'intérieur d'un appel à st_sf() en ajoutant également les informations sur le système de coordonnées.

```
st_sf(
  STATUT      = tapply(com$STATUT      , com$STATUT, head, 1), # identifiants
  POPULATION  = tapply(com$POPULATION , com$STATUT, sum),      # somme des populations
  geometry    = tapply(com            , com$STATUT, st_union), # union des géométries
  crs         = st_crs(com)           # information sur le CRS
)
```

```
#> Simple feature collection with 3 features and 2 fields
#> Geometry type: GEOMETRY
#> Dimension:      XY
#> Bounding box:  xmin: 539668.5 ymin: 6346290 xmax: 637380.9 ymax: 6439668
#> Projected CRS: RGF93 v1 / Lambert-93
#>
#>      STATUT POPULATION      geometry
#> 1 Commune simple    140259 POLYGON ((554732.7 6356281,...
#> 2  Préfecture      19907 MULTIPOLYGON (((580994.5 63...
#> 3 Sous-préfecture  13763 MULTIPOLYGON (((626296.4 63...
```

L'avantage de cette solution est qu'elle permet d'agrégier les variables attributaires avec des fonctions d'agrégation différentes. Nous pouvons par exemple utiliser la somme pour une population (un stock) et la moyenne pour un taux de chômage (un ratio).

- Avec la fonction `aggregate()`

```
com_u <- aggregate(
  x = com["POPULATION"],
  by = list(STATUT = com$STATUT),
  FUN = sum
)
```

Cette solution ne permettra pas d'agrégier les variables attributaires avec des fonctions d'agrégation différentes. Nous devons donc choisir avec précaution en amont les variables que l'on souhaite agréger et leur fonction d'agrégation.

- Avec la bibliothèque `dplyr`

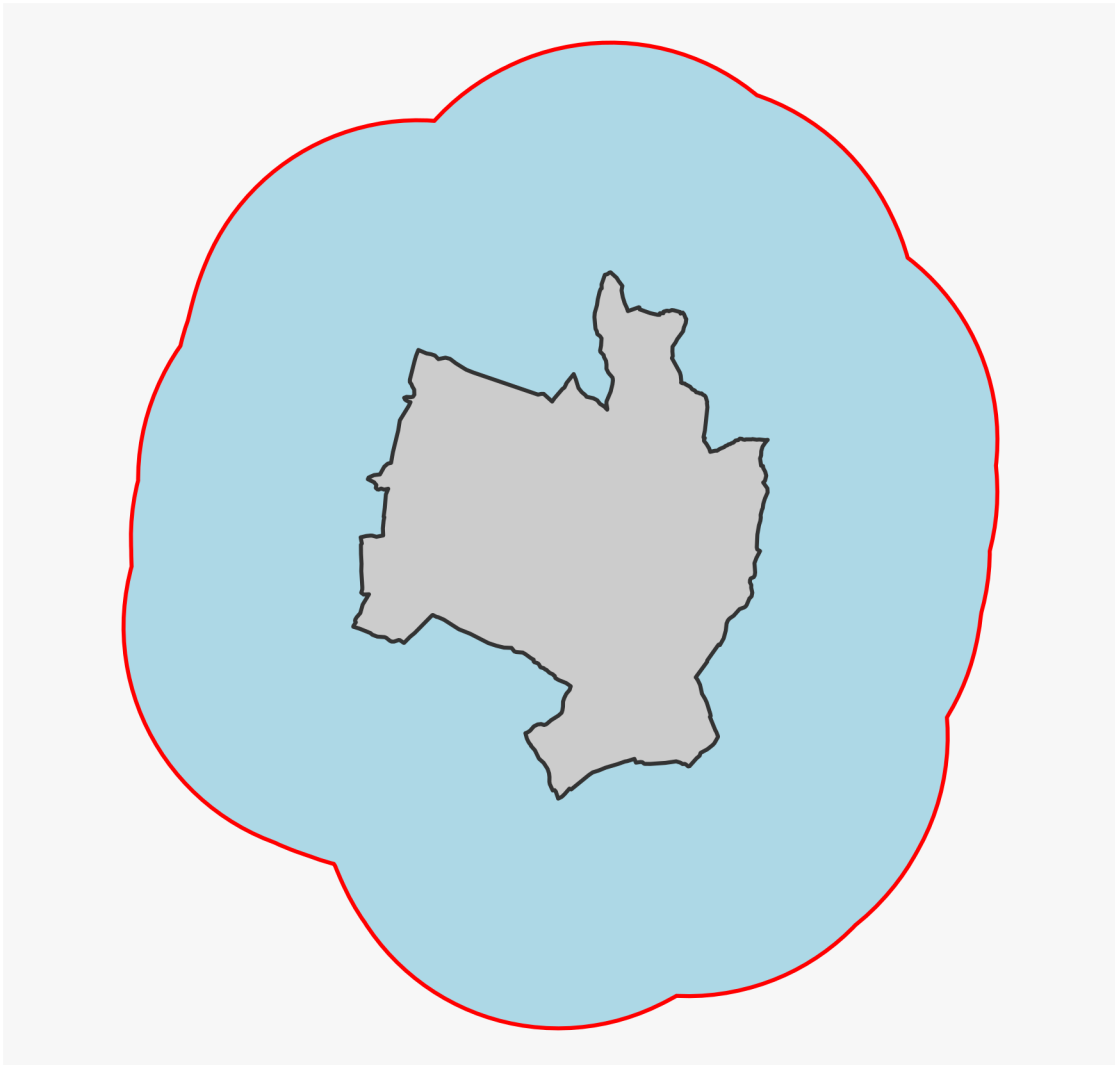
```
library(dplyr)

com_u <- com |>
  group_by(STATUT) |>
  summarise(POPULATION = sum(POPULATION))
```

7.4 Construire une zone tampon

La fonction `st_buffer()` permet de construire des zones tampons. La distance est exprimée en unité de la projection utilisée (`st_crs(x)$units`).

```
gramat <- com[com$NOM_COM == "Gramat", ]  
  
gramat_b <- st_buffer(x = gramat, dist = 5000)  
  
mf_map(gramat_b, col = "lightblue", lwd=2, border = "red")  
mf_map(gramat, add = TRUE, lwd = 2)
```

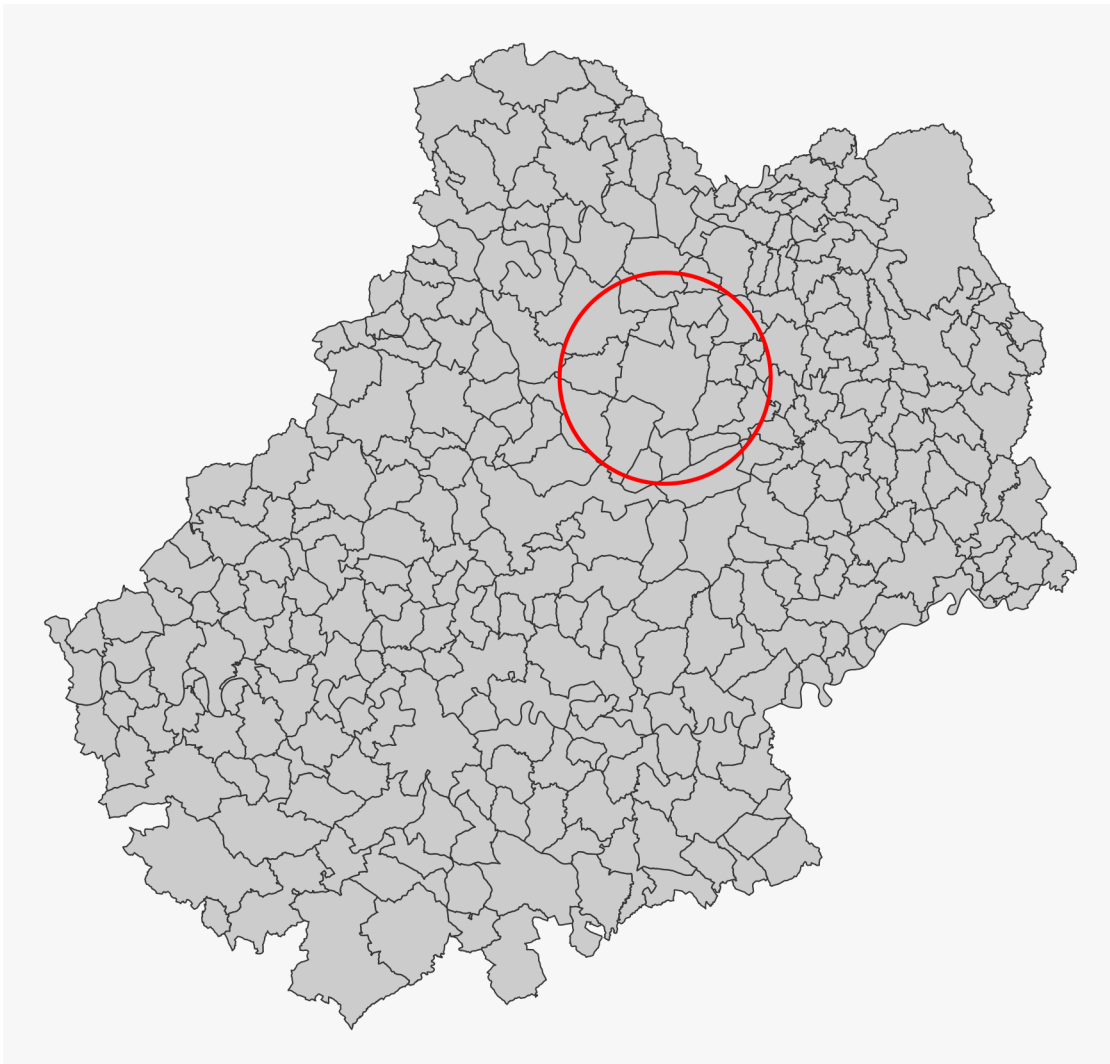


7.5 Réaliser une intersection

En utilisant la fonction `st_intersection()`, on peut découper une couche par une autre.

```
# création d'une zone tampon autour du centroïde de la commune de Gramat
zone <- st_geometry(gramat) |>
  st_centroid() |>
  st_buffer(10000)

mf_map(com)
mf_map(zone, border = "red", col = NA, lwd = 2, add = TRUE)
```

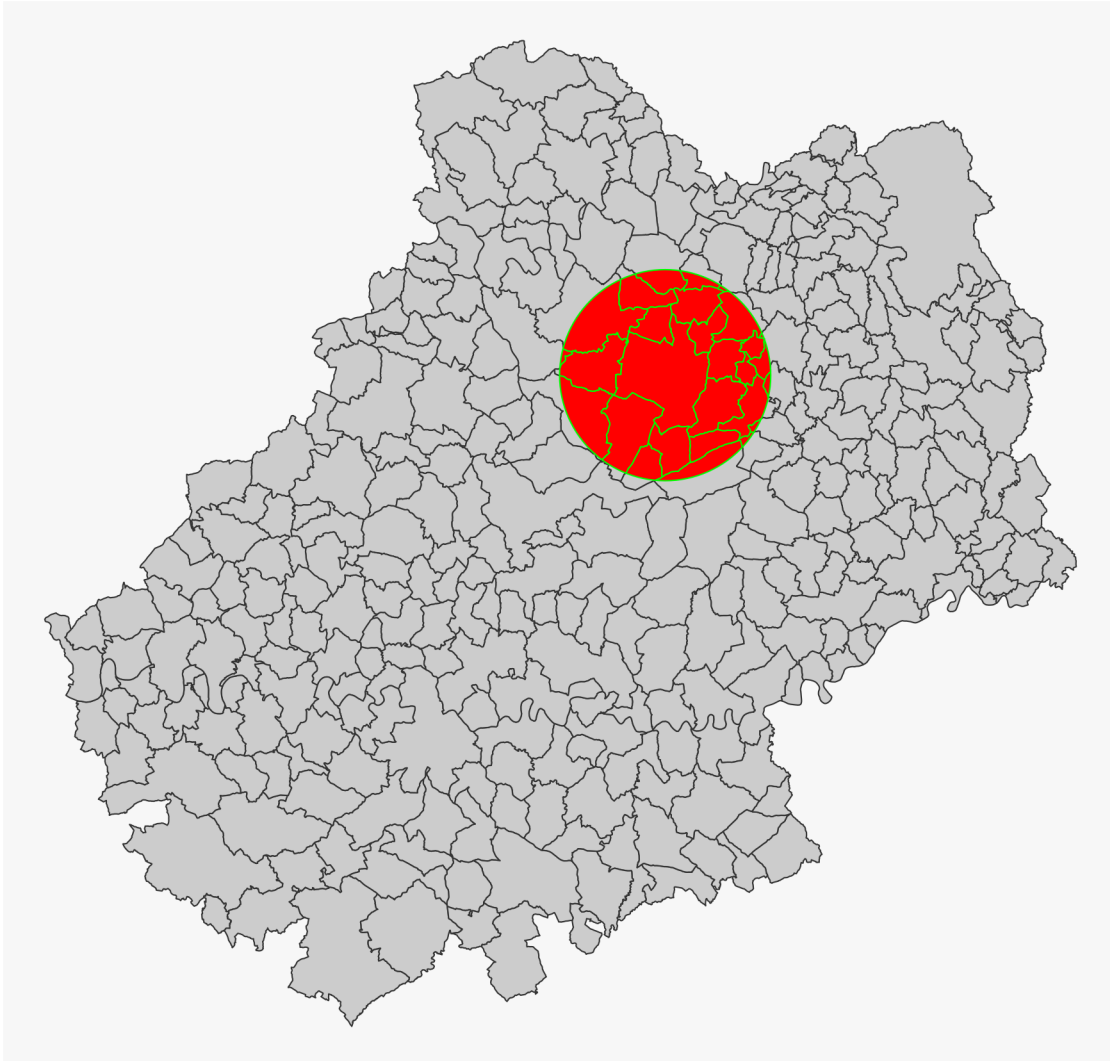


```
com_z <- st_intersection(x = com, y = zone)
```

```
#> Warning: attribute variables are assumed to be spatially constant throughout  
#> all geometries
```

```
mf_map(com)
```

```
mf_map(com_z, col = "red", border = "green", add = TRUE)
```



```
mf_map(com_z)
```



Dans cet exemple nous avons utilisé les pipes (`|>`). Les pipes permettent d'enchaîner une suite d'instructions.

7.6 Créer une grille régulière

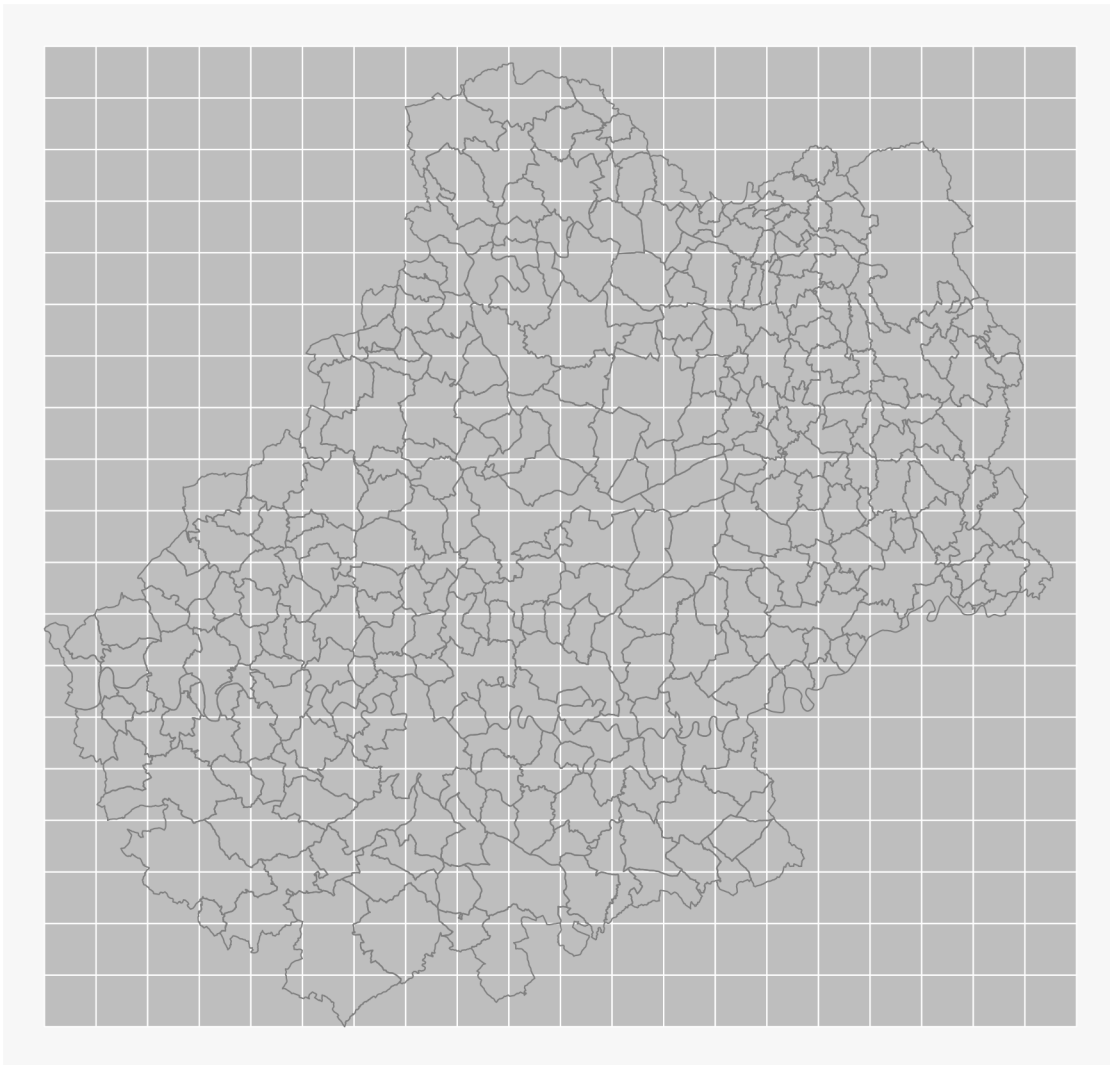
La fonction `st_make_grid()` permet de créer une grille régulière. La fonction produit un objet `sfc`, il faut ensuite utiliser la fonction `st_sf()` pour transformer cet objet `sfc` en objet `sf`.

Lors de cette transformation nous rajoutons ici une colonne d'identifiants uniques.

```
# Création de la grille
grid <- st_make_grid(x = com, cellsize = 5000)

# Ajout d'un identifiant unique
grid <- st_sf(ID = 1:length(grid), geom = grid)

mf_map(grid, col = "grey", border = "white")
mf_map(com, col = NA, border = "grey50", add = TRUE)
```



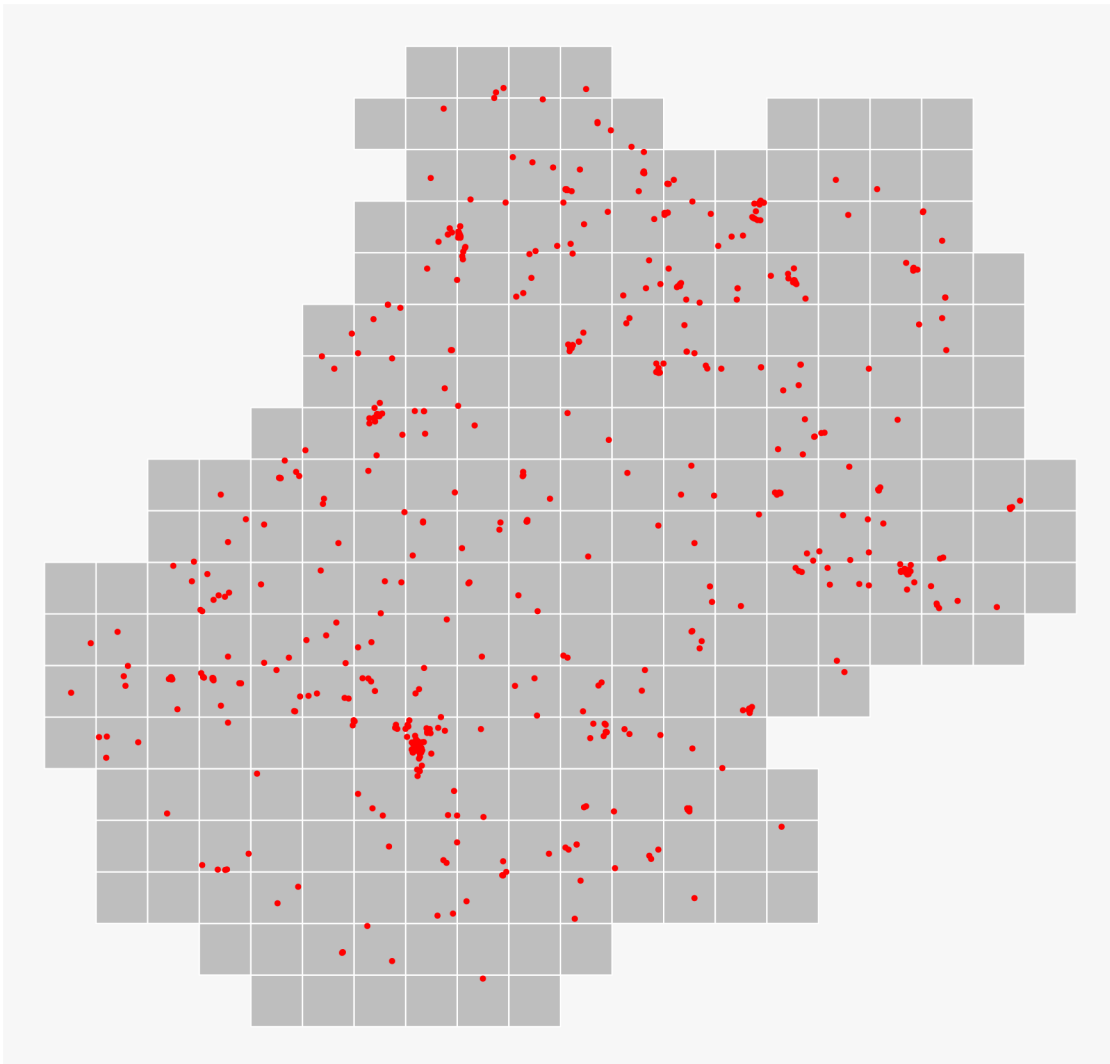
7.7 Compter des points dans un polygone

Sélection des carreaux de la grille qui intersectent le département avec `st_filter()`.

```
grid <- st_filter(grid, dep_46, .predicate = st_intersects)

# Import d'une couche géographique ponctuelle des restaurants du Lot
restaurant <- st_read("data/lot.gpkg", layer = "restaurants", quiet = TRUE)

mf_map(grid, col = "grey", border = "white")
mf_map(restaurant, pch = 20, col = "red", cex = .5, add = TRUE)
```



Nous utilisons ensuite la fonction `st_intersects(..., sparse = TRUE)` qui nous permettra d'avoir pour chaque élément de l'objet `grid` la liste des éléments (via leurs indexes) de l'objet `restaurant` qui se trouvent à l'intérieur.

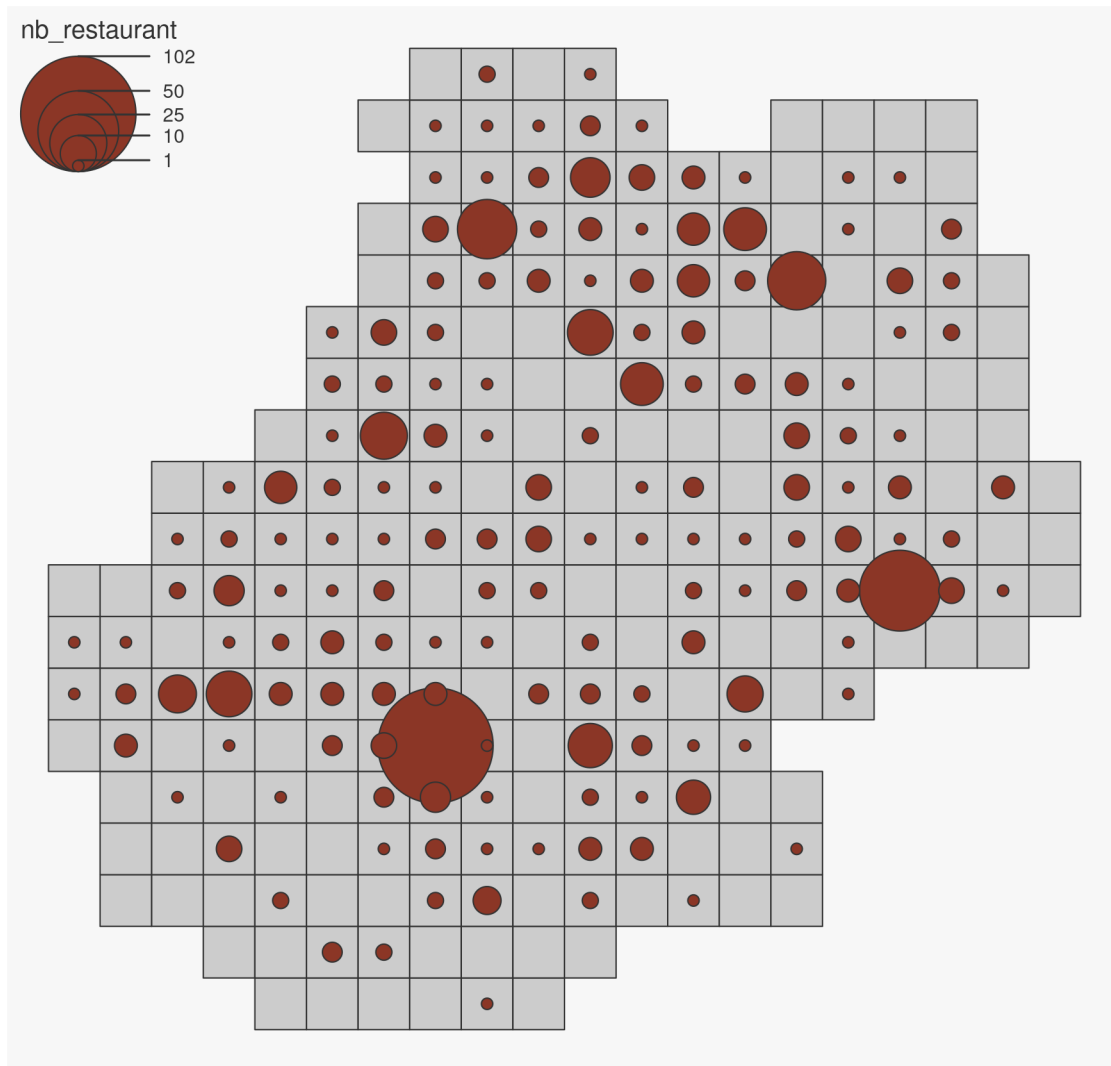
```
inter <- st_intersects(grid, restaurant, sparse = TRUE)
length(inter) == nrow(grid)
```

```
#> [1] TRUE
```

Pour compter le nombre de restaurants il suffit donc de reporter la longueur de chacun des éléments de cette liste.

```
grid$nb_restaurant <- lengths(inter)
mf_map(grid)
mf_map(grid, var = "nb_restaurant", type = "prop")
```

```
#> 94 '0' values are not plotted on the map.
```

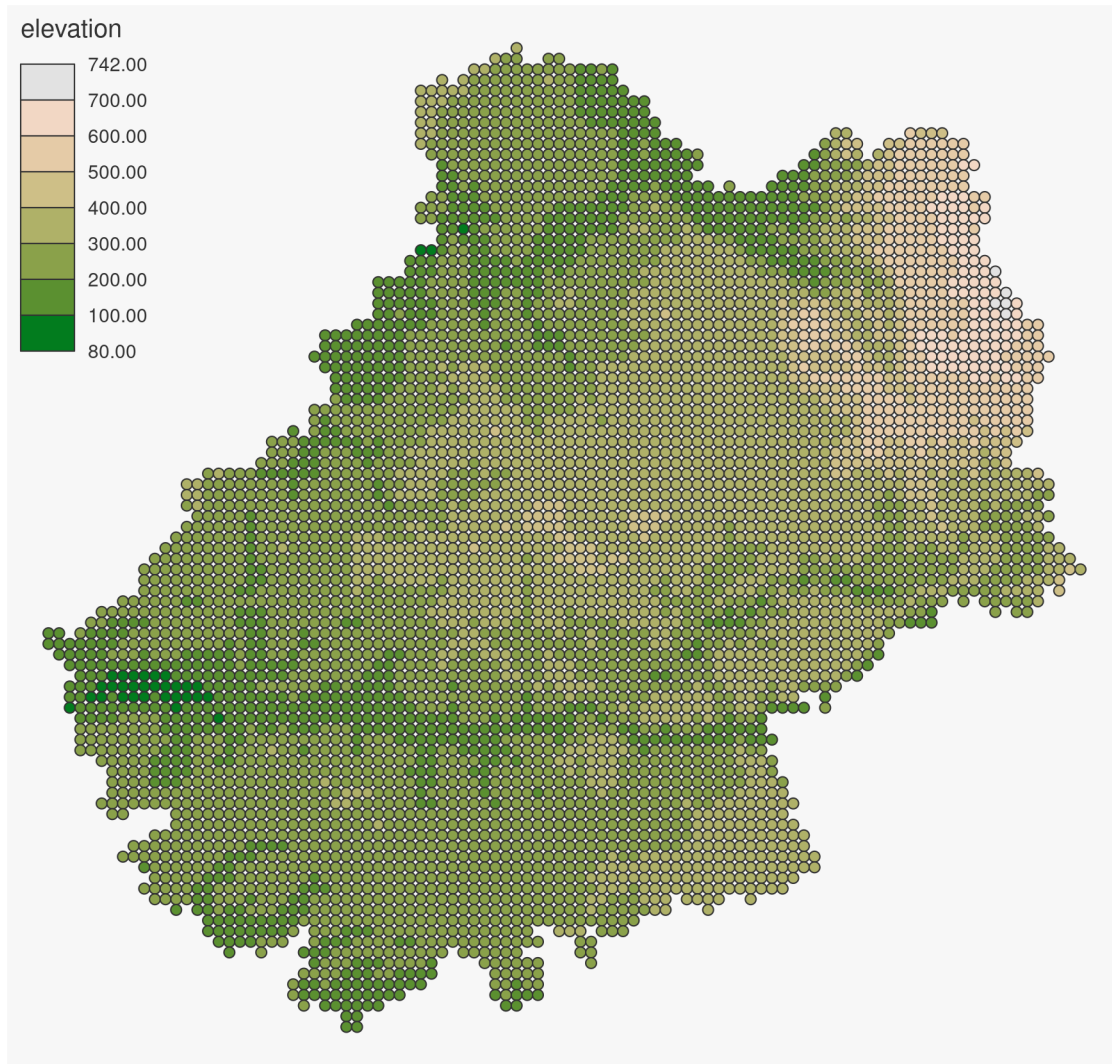


7.8 Agréger les valeurs de points dans des polygones

Ici nous voulons résumer l'information contenue dans une couche de points dans des polygones. Nous voulons connaître l'altitude minimale et maximale de chaque communes. Nous commençons par importer une couche de points d'altitude, la couche **elevations** du fichier **lot.gpkg**.

```
elev <- st_read("data/lot.gpkg", "elevations", quiet = TRUE)
```

```
mf_map(elev, "elevation", "choro",  
       breaks = c(80, seq(100, 700, by = 100), 742),  
       pal = hcl.colors(8, "Terrain2"),  
       pch = 21, leg_pos = "topleft", cex = .75)
```



L'objectif est d'agréger les valeurs de ces points (les altitudes contenues dans le champ **elevation**) dans les communes du Lot.

En utilisant la fonction `st_join()` nous pouvons récupérer les attributs des communes dans lesquelles se trouvent les points.

```
inter <- st_join(x = elev, y = com[, "INSEE_COM"])
inter
```

```
#> Simple feature collection with 5228 features and 2 fields
#> Geometry type: POINT
#> Dimension:      XY
#> Bounding box:  xmin: 540333.3 ymin: 6347372 xmax: 637333.3 ymax: 6439372
#> Projected CRS: RGF93 v1 / Lambert-93
#> First 10 features:
#>   elevation INSEE_COM          geom
#> 1   308.8546    46083 POINT (584333.3 6439372)
#> 2   304.6855    46083 POINT (582333.3 6438372)
#> 3   290.6638    46083 POINT (583333.3 6438372)
#> 4   295.0353    46083 POINT (584333.3 6438372)
#> 5   297.6773    46083 POINT (587333.3 6438372)
#> 6   257.7393    46083 POINT (588333.3 6438372)
#> 7   310.1883    46083 POINT (580333.3 6437372)
#> 8   305.0571    46083 POINT (581333.3 6437372)
#> 9   298.5876    46083 POINT (582333.3 6437372)
#> 10  287.6990    46083 POINT (583333.3 6437372)
```

Nous pouvons ensuite utiliser la fonction `aggregate()` pour agréger les altitudes par communes, d'abord l'altitude minimale, puis l'altitude maximale.

```
# x   : la variable que l'on veut agréger
# by  : la variable qui servira à agréger
# FUN : la fonction à utiliser lors de l'agrégation
alti_min <- aggregate(x = list(alt_min = inter$elevation),
                      by = list(INSEE_COM = inter$INSEE_COM),
                      FUN = "min")

alti_max <- aggregate(x = list(alt_max = inter$elevation),
                      by = list(INSEE_COM = inter$INSEE_COM),
                      FUN = "max")

head(alti_max, n = 3)
```

```
#>   INSEE_COM alt_max
#> 1     46001 302.4913
#> 2     46002 393.9218
#> 3     46003 376.6632
```

On peut ensuite combiner ces résultats à la couche des communes (avec la fonction `merge()`).

```
com <- merge(com, alti_min, by = "INSEE_COM", all.x = TRUE)
com <- merge(com, alti_max, by = "INSEE_COM", all.x = TRUE)
head(com, n = 3)
```

```
#> Simple feature collection with 3 features and 14 fields
#> Geometry type: MULTIPOLYGON
#> Dimension:      XY
#> Bounding box:  xmin: 557759.2 ymin: 6371852 xmax: 607179 ymax: 6418606
#> Projected CRS: RGF93 v1 / Lambert-93
#>   INSEE_COM NOM_COM      STATUT POPULATION   AGR_H   AGR_F   IND_H
#> 1    46001   Albas Commune simple      522  4.978581 0.000000  4.936153
#> 2    46002   Albiac Commune simple      67  0.000000 9.589041  0.000000
#> 3    46003 Alvignac Commune simple     706 10.419682 0.000000 10.419682
#>   IND_F   BTP_H BTP_F   TER_H   TER_F alt_min alt_max
#> 1 0.000000  9.957527    0 44.917145 34.681799 109.5772 302.4913
#> 2 0.000000  4.794521    0  4.794521  9.589041 363.4579 393.9218
#> 3 5.209841 10.419682    0 57.308249 78.147612 258.8378 376.6632
#>
#>      geom
#> 1 MULTIPOLYGON (((559262 6371...
#> 2 MULTIPOLYGON (((605540.7 64...
#> 3 MULTIPOLYGON (((593707.7 64...
```

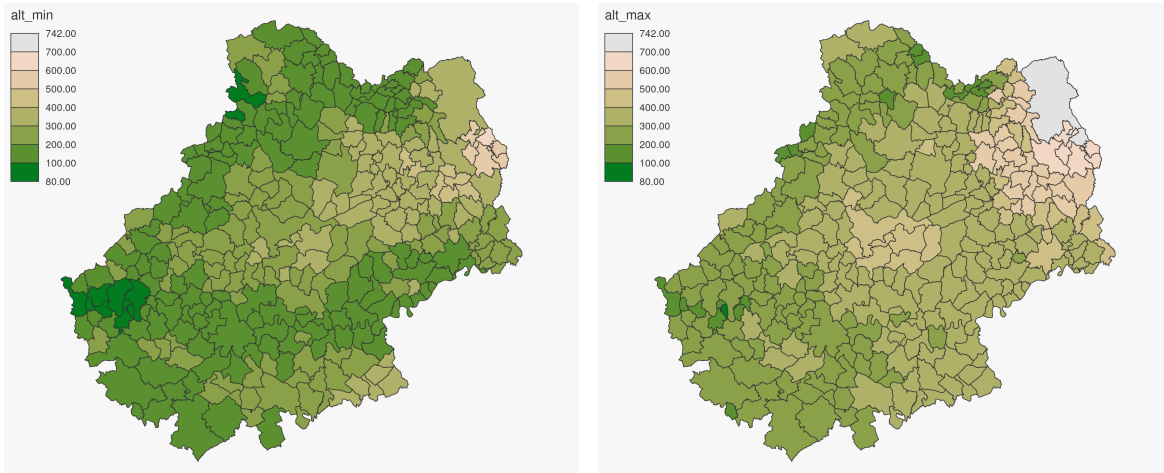
```
bks <- c(80, seq(100, 700, by = 100), 742)
cols <- hcl.colors(8, "Terrain2")

mf_map(com, "alt_min", "choro", breaks = bks, pal = cols)
mf_map(com, "alt_max", "choro", breaks = bks, pal = cols)
```

7.9 Simplifier des géométries

Le package `rmapshaper` (Teucher et Russell, 2023) s'appuie sur la bibliothèque JavaScript Mapshaper (Bloch, 2013) pour proposer plusieurs méthodes de simplification des géométries qui respectent la topologie.

L'argument `keep` permet d'indiquer le niveau de simplification. L'argument `keep_shapes` permet de conserver tous les polygones quand le niveau de simplification est élevé.



```
library("rmapshaper")
com_simp1 <- ms_simplify(com, keep = 0.01 , keep_shapes = TRUE)
com_simp2 <- ms_simplify(com, keep = 0.001, keep_shapes = TRUE)
mf_map(com)
mf_map(com_simp1)
mf_map(com_simp2)
```



Exercice

1. Calculez le nombre de restaurants par commune.
2. Quelles communes ont plus de 10 restaurants et moins de 1000 habitants ?
3. Créez une carte où vous afficherez toutes les communes en gris et les communes sélectionnées plus haut en rouge.

8 Mesures

8.1 Créer une matrice de distances

La fonction `st_distance()` permet de calculer une matrice de distance entre deux couches de points.

Si le système de projection du jeu de données est renseigné, les distances sont exprimées dans l'unité de mesure de la projection (le plus souvent en mètres).

```
library(sf)
com <- st_read("data/lot.gpkg", layer = "communes", quiet = TRUE, agr = "constant")
# transformation de la couche com en couche de points
com_c <- st_centroid(com)
mat <- st_distance(x = com_c, y = com_c)
mat[1:5,1:5]
```

```
#> Units: [m]
#>      [,1]      [,2]      [,3]      [,4]      [,5]
#> [1,]  0.000 56784.77 54353.94 61166.42 3790.688
#> [2,] 56784.770  0.00 12454.29 7146.11 57288.103
#> [3,] 54353.942 12454.29  0.00 19388.52 54030.811
#> [4,] 61166.418 7146.11 19388.52  0.00 62016.141
#> [5,] 3790.688 57288.10 54030.81 62016.14  0.000
```

8.2 Calcul de superficies

La fonction `st_area()` permet de calculer des superficies.

```
st_area(com[1:5, ])
```

```
#> Units: [m^2]
#> [1] 21721665 3813205 13024216 9993074 5540367
```

8.3 Convertir des unités

Le package `units` (Pebesma et al., 2016) permet de définir et convertir facilement des unités de mesure.

Le package peut se révéler assez utile quand nous manipulons différentes unités de mesures régulièrement.

```
library(units)

distances <- c(1, 2, 3, 5, 0.5)
surfaces <- c(500, 1000, 10000, 20000)

# définition des unités de mesure initiales
distances <- set_units(distances, "km")
distances
```

```
#> Units: [km]
#> [1] 1.0 2.0 3.0 5.0 0.5
```

```
surfaces <- set_units(surfaces, "m2")
surfaces
```

```
#> Units: [m^2]
#> [1] 500 1000 10000 20000
```

```
# transformation des unités de mesure
distances <- set_units(distances, "m")
distances
```

```
#> Units: [m]
#> [1] 1000 2000 3000 5000 500
```

```
surfaces <- set_units(surfaces, "ha")
surfaces
```

```
#> Units: [ha]
#> [1] 0.05 0.10 1.00 2.00
```

```
## Suppression des unités  
distances <- set_units(distances, NULL)  
distances
```

```
#> [1] 1000 2000 3000 5000 500
```

```
surfaces <- set_units(surfaces, NULL)  
surfaces
```

```
#> [1] 0.05 0.10 1.00 2.00
```

partie II

Les données raster : le package terra

9 Le package terra

9.1 Présentation

L'objectif du package `terra` (Hijmans, 2023b) est de proposer des méthodes de traitement et d'analyse de données raster. Ce package est très similaire au package `raster`, mais il propose plus de fonctionnalités. Il est plus rapide et plus facile à utiliser.

Ce chapitre est largement inspiré de deux présentations (Madelin, 2021; Nowosad, 2021) réalisées dans le cadre de l'école thématique "Science de l'Information Géographique Reproductibles 2021" (SIGR2021).

Site web du package `terra` :

[Spatial Data Science with R and "terra"](#)

9.2 Format des objets `SpatRaster`

Le package `terra` (Hijmans, 2023b) permet de gérer des données vectorielles et raster. Pour manipuler ces données spatiales, `terra` les stockent dans des objets de type `SpatVector` et `SpatRaster`. Dans ce document, nous nous focalisons sur la manipulation de données raster (`SpatRaster`) à partir de fonctions proposées par ce package.

Un objet `SpatRaster` représente des données matricielles, en une ou plusieurs couches (variables). Cet objet stocke également un certain nombre de paramètres fondamentaux qui le décrivent (nombre de colonnes, de lignes, étendue spatiale, système de référence des coordonnées...).

4	8	7	4	4	1	8	3	7	6
1	7	9	3	4	4	8	9	7	9
7	7	2	2	7	7	6	6	7	7
6	9	6	2	3	7	9	8	9	2
3	5	9	5	4	5	8	8	1	3
3	6	8	9	3	8	7	3	9	8
6	8	1	2	1	7	1	1	3	4
4	3	4	8	9	5	3	3	4	3
6	4	4	1	7	7	8	7	2	1
5	1	6	9	1	6	9	9	1	1

Figure 9.1: Racine (2016)

10 Import et export

Le package `terra` permet d'importer et d'exporter des fichiers raster. Il repose sur la bibliothèque GDAL (GDAL/OGR contributors, 2022) qui permet de lire et de traiter un très grand nombre de format d'images géographiques.

```
library(terra)
```

10.1 Import

La fonction `rast()` permet de créer et/ou d'importer des données raster. Les lignes suivantes importent le fichier raster `elevation.tif` (*Tagged Image File Format*) au format d'objet `SpatRaster`.

```
elev <- rast("data/elevation.tif")
elev
```

```
#> class      : SpatRaster
#> dimensions : 987, 1300, 1  (nrow, ncol, nlyr)
#> resolution : 0.0002972796, 0.0002972796  (x, y)
#> extent     : 1.245749, 1.632213, 44.30927, 44.60269  (xmin, xmax, ymin, ymax)
#> coord. ref.: lon/lat WGS 84 (EPSG:4326)
#> source     : elevation.tif
#> name       : altitude
#> min value  :      91
#> max value  :     421
```

10.2 Export

La fonction `writeRaster()` permet d'enregistrer un objet `SpatRaster` sur votre machine, dans le format de votre choix.

```
writeRaster(x = elev, filename = "data/new_elevation_Lot.tif")
```

Conversion pour le package sf

Le package `terra` permet de manipuler des objets vectoriels en utilisant des objets de type `SpatVector`.

La fonction `st_as_sf()` du package `sf` permet de transformer un objet `SpatVector` en objet `sf`.

```
# adresse du fichier d'exemple
f <- system.file("ex/lux.shp", package="terra")
# import au format SpatVector
v <- vect(f)
library(sf)
# conversion
v2 <- st_as_sf(v)
class(v2)
```

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

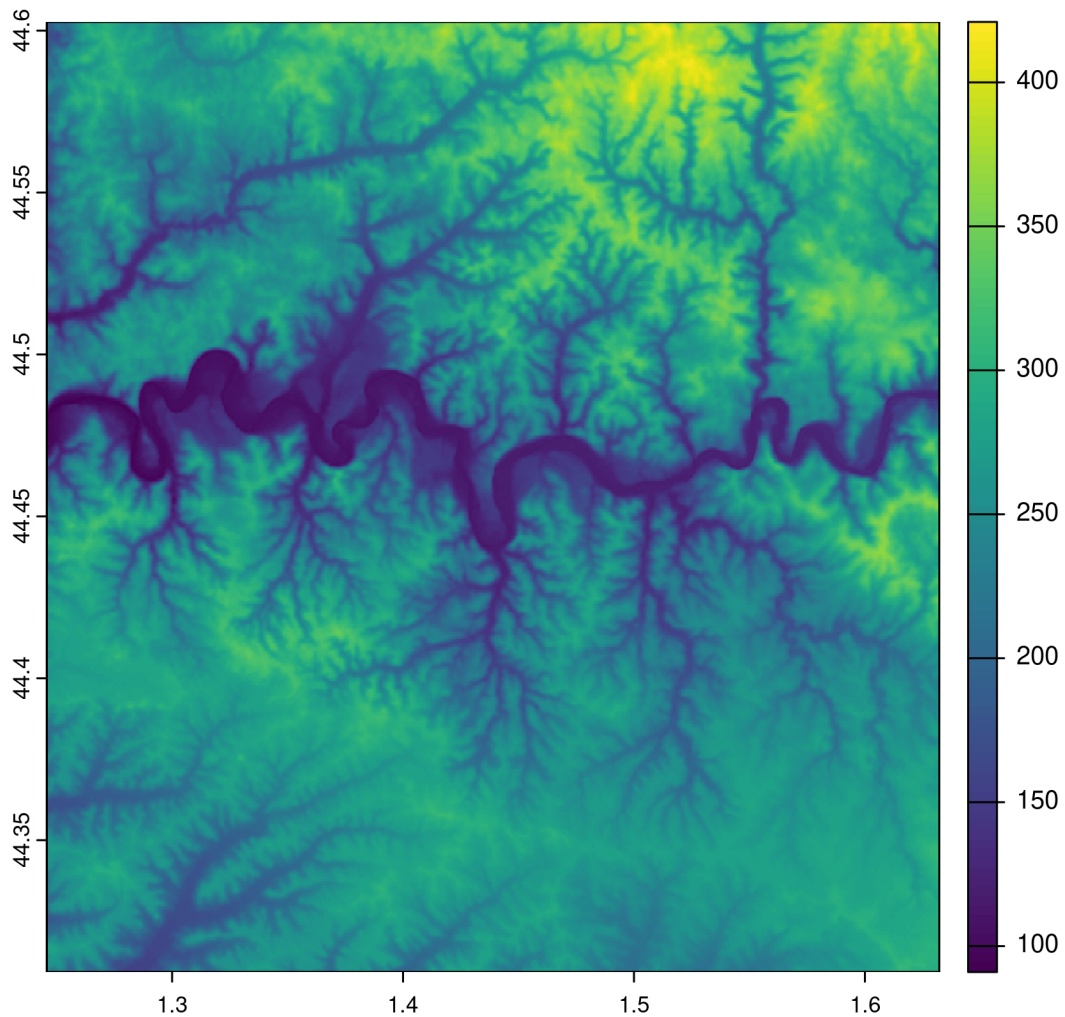

11 Affichage

La fonction `plot()` permet d'afficher un objet `SpatRaster`.

```
library(terra)
```

```
#> terra 1.7.78
```

```
elev <- rast("data/elevation.tif")  
plot(elev)
```



Un raster contient toujours des données numériques, mais il peut aussi bien s'agir de données quantitatives que de données qualitatives (catégorielles) codées numériquement.

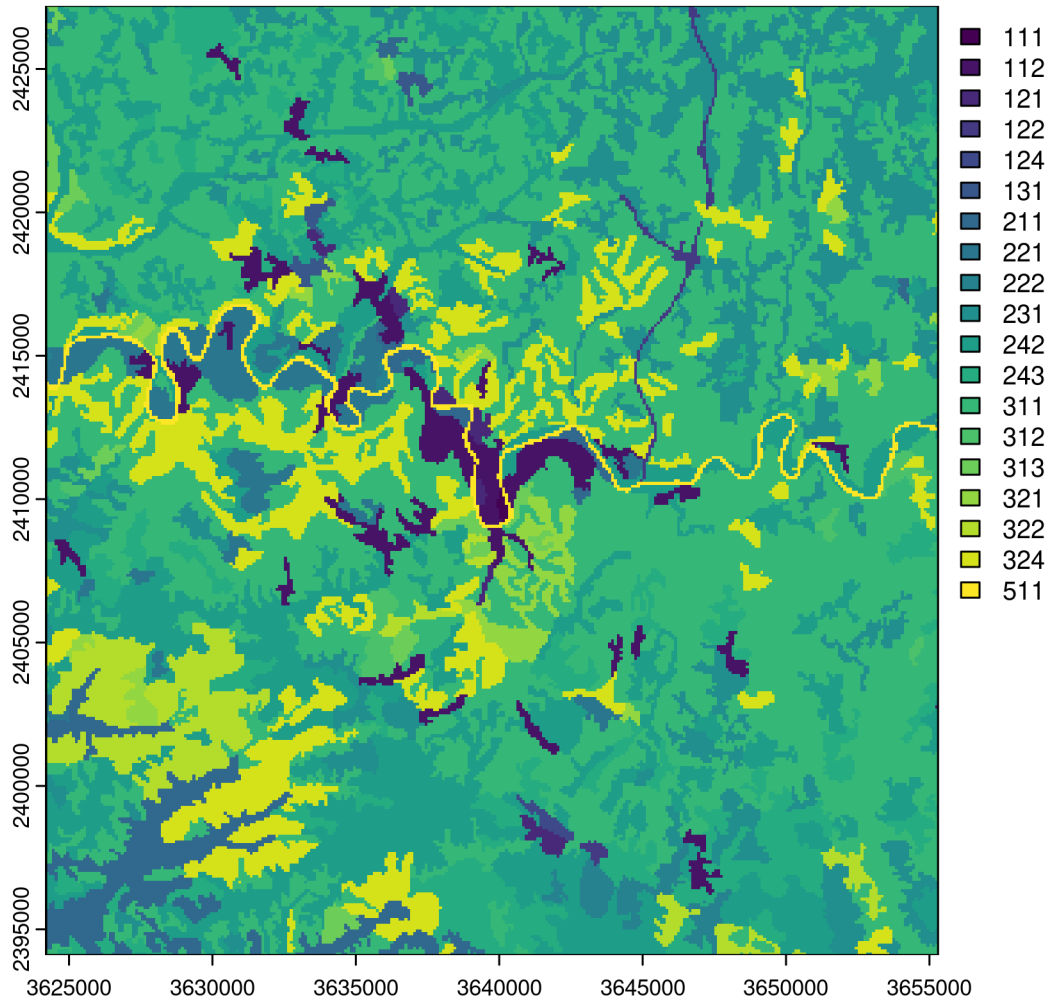
Précisez le type de données stockées avec l'argument `type` (`type = "continuous"` par défaut), pour les afficher correctement.

Import et affichage d'un raster contenant des données catégorielles : CORINE Land Cover 2018 (type d'occupation du sol) avec une résolution de 100m. Ces données ont été récupérées sur le [site de Copernicus](#), le programme européen de surveillance de la Terre qui collecte et met à disposition des données issues de ses propres satellites (*Sentinelles*) d'observation. Une extraction centrée sur la commune de Cahors a ensuite été réalisée.

```
clc <- rast("data/clc_2018.tif")
```

```
# Affichage
```

```
plot(clc, type="classes")
```

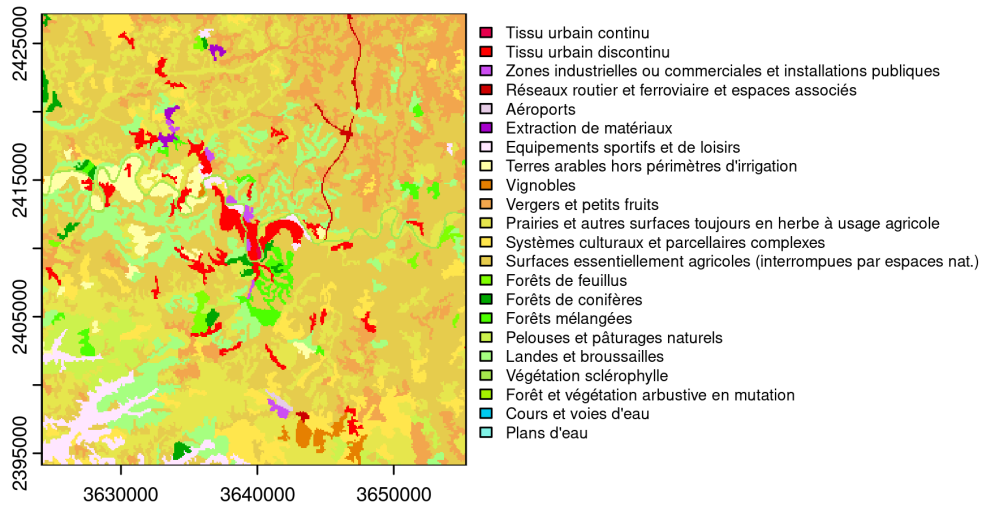


Pour afficher les intitulés réels des types d'occupation du sol, ainsi que les couleurs officielles de la nomenclature CORINE Land Cover (consultables [ici](#)), vous pouvez procéder de la manière suivante :

```

intitule_poste <- c(
  "Tissu urbain continu", "Tissu urbain discontinu",
  "Zones industrielles ou commerciales et installations publiques",
  "Réseaux routier et ferroviaire et espaces associés",
  "Aéroports","Extraction de matériaux",
  "Equipements sportifs et de loisirs",
  "Terres arables hors périmètres d'irrigation", "Vignobles",
  "Vergers et petits fruits",
  "Prairies et autres surfaces toujours en herbe à usage agricole",
  "Systèmes culturaux et parcellaires complexes",
  "Surfaces essentiellement agricoles (interrompues par espaces nat.)",
  "Forêts de feuillus", "Forêts de conifères", "Forêts mélangées",
  "Pelouses et pâturages naturels",
  "Landes et broussailles", "Végétation sclérophylle",
  "Forêt et végétation arbustive en mutation",
  "Cours et voies d'eau", "Plans d'eau"
)
couleur_off <- c("#E6004D", "#FF0000", "#CC4DF2", "#CC0000", "#E6CCE6", "#A600CC",
  "#FFE6FF", "#FFFA8", "#E68000", "#F2A64D", "#E6E64D", "#FFE64D",
  "#E6CC4D", "#80FF00", "#00A600", "#4DFF00", "#CCF24D", "#A6FF80",
  "#A6E64D", "#A6F200", "#00CCF2", "#80F2E6")
plot(clc,
  type = "classes",
  levels = intitule_poste,
  col = couleur_off,
  plg = list(cex = 0.7),
  mar = c(3.1, 1.1, 1.1, 10)
)

```



12 Modifications de la zone d'étude

12.1 Projections

Pour modifier le système de projection d'un raster, nous pouvons utiliser la fonction `project()`. Il est alors nécessaire de **fournir un modèle** et d'indiquer la **méthode d'estimation** des nouvelles valeurs des cellules.

Le modèle est un nouveau raster sur lequel aligner/projeter les données. Pour construire un modèle, nous utilisons dans un premier temps la fonction `project(x, crs)`. Cette fonction va produire un raster avec une résolution choisie automatiquement. Nous utilisons ensuite la fonction `res()` (ou `dim()`) pour ajuster la résolution de ce raster modèle (voir `?project`). La fonction `project()` peut ensuite être réutilisée pour projeter les valeurs dans le modèle, en précisant la méthode d'estimation à utiliser.

Quatre méthodes d'estimation sont disponibles :

- *near* : plus proche voisin. Méthode rapide et par défaut pour les données qualitatives;
- *bilinear* : interpolation bilinéaire. Méthode par défaut pour les données quantitatives;
- *cubic* : interpolation cubique;
- *cubicspline* : interpolation cubique spline.

```
library(terra)
```

```
#> terra 1.7.78
```

```
elev_raw <- rast("data/elevation.tif")
clc_raw <- rast("data/clc_2018.tif")

# Création d'un modèle de raster en 2154
model_proj <- project(x = elev_raw, y = "EPSG:2154")

# Ajustement de la résolution du modèle (100 m)
```

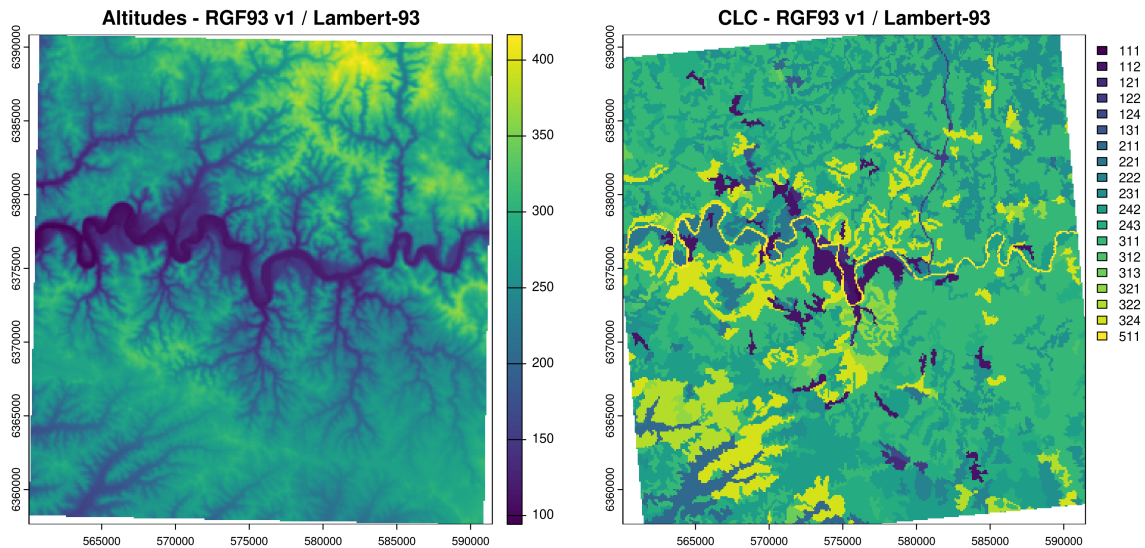
```

res(model_proj) <- 100

# Projection dans le modèle
elev <- project(x = elev_raw, y = model_proj, method = "bilinear")
clc <- project(x = clc_raw, y = model_proj, method = "near")

plot(elev, main = "Altitudes - RGF93 v1 / Lambert-93" )
plot(clc, type = "classes", main = "CLC - RGF93 v1 / Lambert-93")

```



Pour sauvegarder les rasters reprojetés :

```

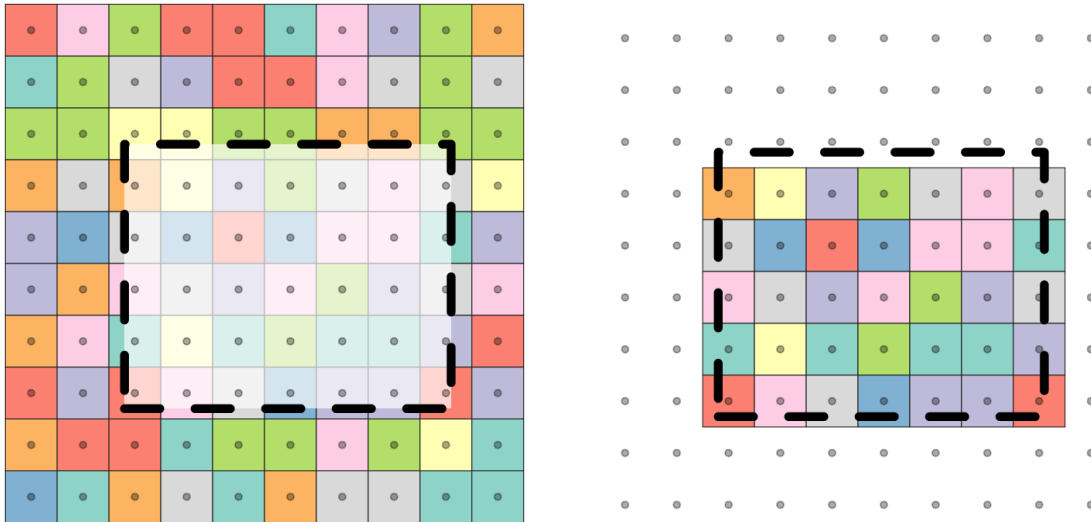
writeRaster(elev, filename = "data/elev.tif")
writeRaster(clc, filename = "data/clc.tif")

```

12.2 Crop

Le découpage d'un raster en fonction de l'étendue d'un autre objet, `SpatVector` ou `SpatRaster`, est réalisable avec la fonction `crop()`.

Import de données vectorielles (découpages communaux) avec la fonction `vect` du package `terra`. Ces données seront stockées dans un objet `SpatVector`.



(a) Racine (2016)

```
commune <- vect("data/lot.gpkg", layer="communes")
```

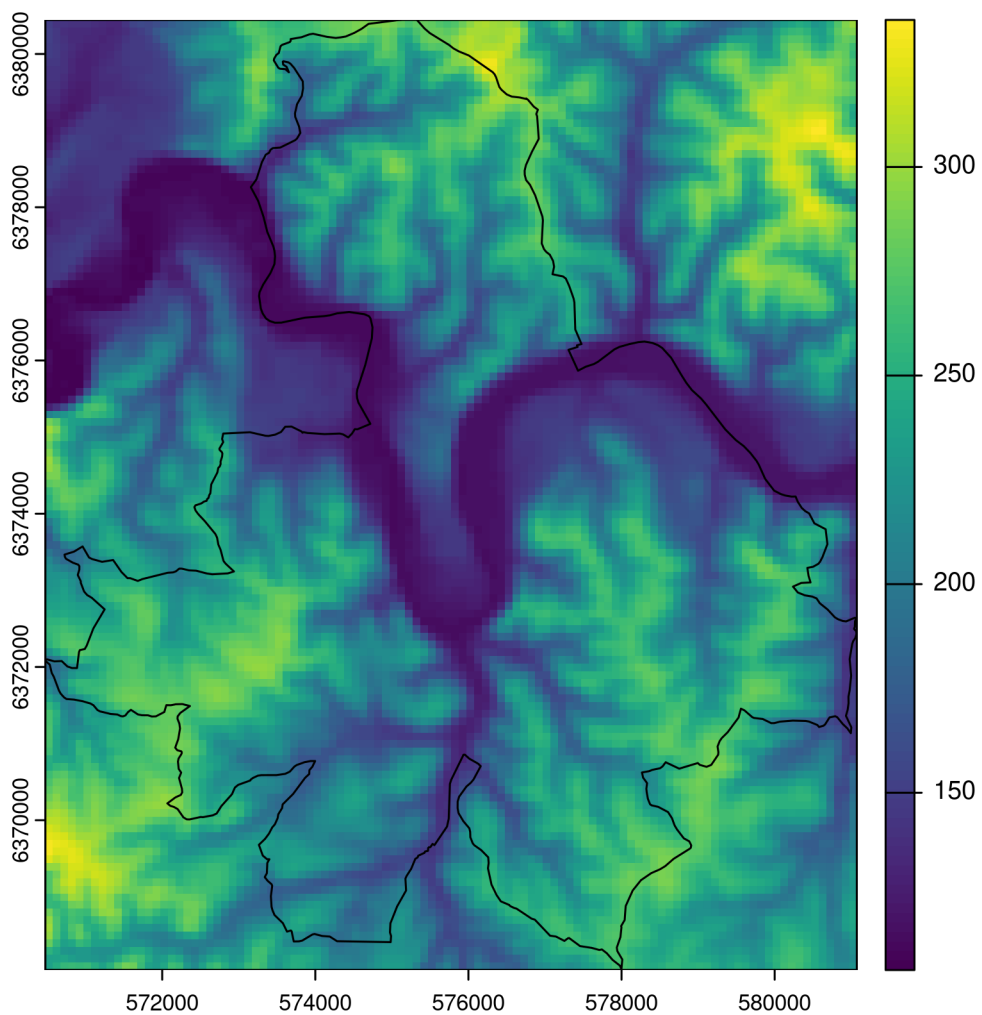
Extraction des limites communales de Cahors (code INSEE = 46042).

```
cahors <- subset(commune, commune$INSEE_COM == "46042")
```

Pour utiliser la fonction `crop()`, les deux couches de données doivent être dans la même projection.

```
crop_cahors <- crop(elev, cahors)
```

```
plot(crop_cahors)
plot(cahors, add = TRUE)
```

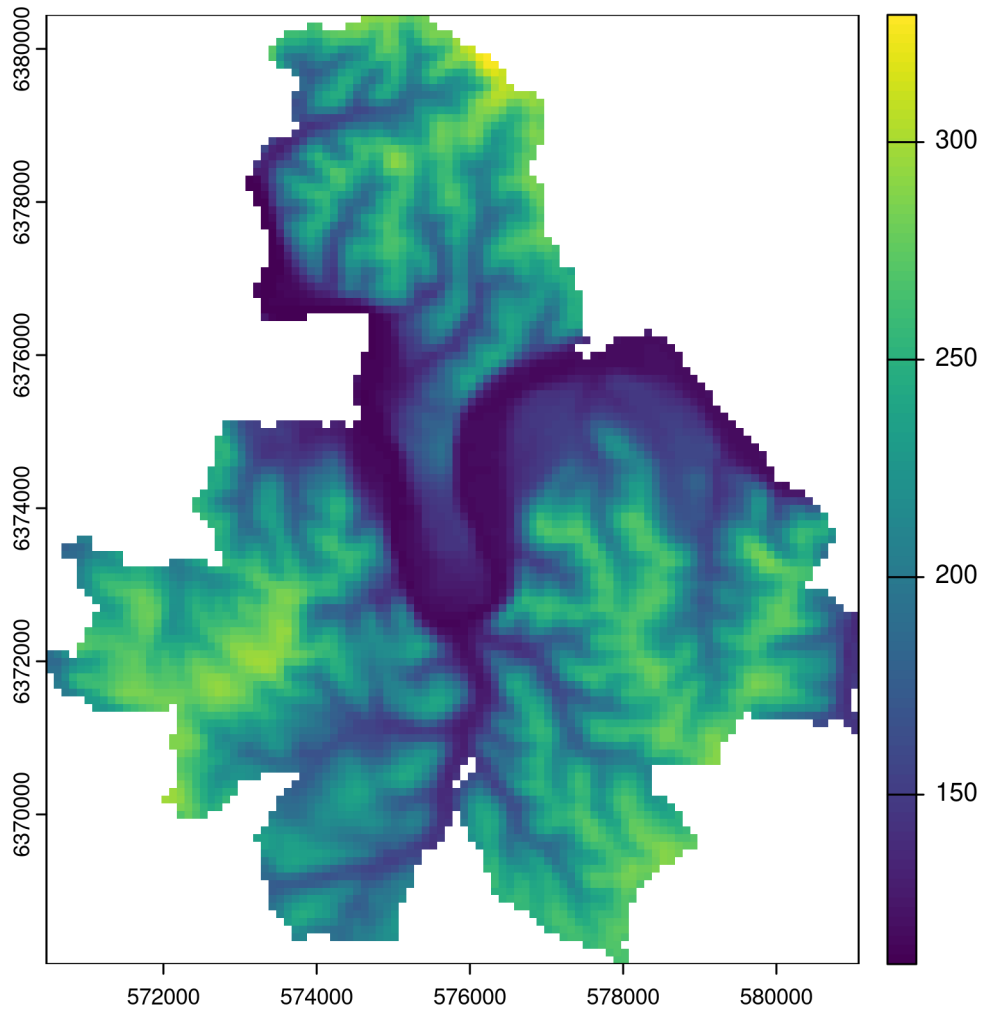
12.3 Mask

Pour afficher uniquement les valeurs d'un raster contenu dans un polygone, utilisez la fonction `mask()`.

Création d'un masque sur le raster `crop_cahors` en fonction des limites communales (polygone) de `cahors` :

```
mask_cahors <- mask(crop_cahors, cahors)
```

```
plot(mask_cahors)
```



Mask vs. Crop

Masquer un raster (**mask**) signifie remplacer les valeurs de pixels en dehors d'une zone d'intérêt en NA. Les dimensions du raster ne sont pas modifiées.

Recadrer un raster (**crop**) signifie supprimer les lignes et/ou les colonnes qui se trouvent en dehors d'une zone d'intérêt. Les dimensions du raster sont modifiées.

12.4 Agrégation & désagrégation

Le ré-échantillonnage d'un raster dans une résolution différente se fait en plusieurs étapes.

1. Afficher la résolution d'un raster avec la fonction `res()`.

```
res(elev)
```

```
#> [1] 100 100
```

2. Créer une grille de même étendue, puis en diminuer la résolution spatiale (plus grosses cellules).

```
elev_lower_model <- elev
```

```
# Tailles des cellules = 1000 mètres
```

```
res(elev_lower_model) <- 1000
```

```
elev_lower_model
```

```
#> class      : SpatRaster
```

```
#> dimensions : 33, 31, 1 (nrow, ncol, nlyr)
```

```
#> resolution  : 1000, 1000 (x, y)
```

```
#> extent      : 560073.1, 591073.1, 6357644, 6390644 (xmin, xmax, ymin, ymax)
```

```
#> coord. ref. : RGF93 v1 / Lambert-93 (EPSG:2154)
```

3. La fonction `resample()` permet de ré-échantillonner les valeurs de départ dans la nouvelle résolution spatiale. Plusieurs méthodes de ré-échantillonnage sont disponibles (voir Section 12.1).

```
elev_lower <- resample(x = elev,  
                      y = elev_lower_model,  
                      method = "bilinear")
```

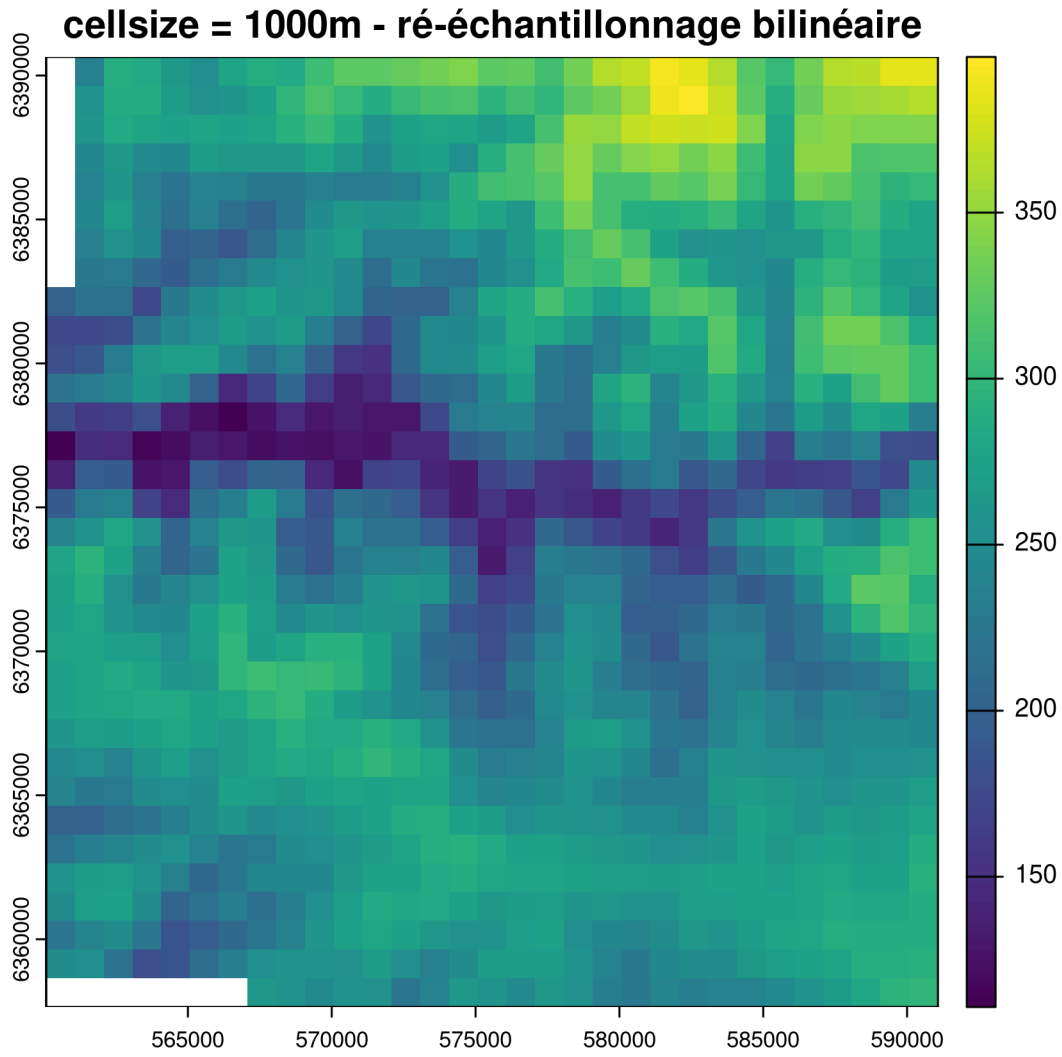
```
plot(elev_lower,  
     main = "cellsize = 1000m - ré-échantillonnage bilinéaire")
```

4	8	7	4	4	1	8	3	7	6
1	7	9	3	4	4	8	9	7	9
7	7	2	2	7	7	6	6	7	7
6	9	6	2	3	7	9	8	9	2
3	5	9	5	4	5	8	8	1	3
3	6	8	9	3	8	7	3	9	8
6	8	1	2	1	7	1	1	3	4
4	3	4	8	9	5	3	3	4	3
6	4	4	1	7	7	8	7	2	1
5	1	6	9	1	6	9	9	1	1

4	4	8	8	7	7	4	4	4	4	1	1	8	8	3	3	7	7	6	6
4	4	8	8	7	7	4	4	4	4	1	1	8	8	3	3	7	7	6	6
1	1	7	7	9	9	3	3	4	4	4	4	8	8	9	9	7	7	9	9
1	1	7	7	9	9	3	3	4	4	4	4	8	8	9	9	7	7	9	9
7	7	7	7	2	2	2	2	7	7	7	7	6	6	6	6	7	7	7	7
7	7	7	7	2	2	2	2	7	7	7	7	6	6	6	6	7	7	7	7
6	6	9	9	6	6	2	2	3	3	7	7	9	9	8	8	9	9	2	2
6	6	9	9	6	6	2	2	3	3	7	7	9	9	8	8	9	9	2	2
3	3	5	5	9	9	5	5	4	4	5	5	8	8	8	8	1	1	3	3
3	3	5	5	9	9	5	5	4	4	5	5	8	8	8	8	1	1	3	3
3	3	6	6	8	8	9	9	3	3	8	8	7	7	3	3	9	9	8	8
3	3	6	6	8	8	9	9	3	3	8	8	7	7	3	3	9	9	8	8
6	6	8	8	1	1	2	2	1	1	7	7	1	1	1	1	3	3	4	4
6	6	8	8	1	1	2	2	1	1	7	7	1	1	1	1	3	3	4	4
4	4	3	3	4	4	8	8	9	9	5	5	3	3	3	3	4	4	3	3
4	4	3	3	4	4	8	8	9	9	5	5	3	3	3	3	4	4	3	3
6	6	4	4	4	4	1	1	7	7	7	7	8	8	7	7	2	2	1	1
6	6	4	4	4	4	1	1	7	7	7	7	8	8	7	7	2	2	1	1
5	5	1	1	6	6	9	9	1	1	6	6	9	9	9	9	1	1	1	1
5	5	1	1	6	6	9	9	1	1	6	6	9	9	9	9	1	1	1	1

5	5.8	3.2	7	7.2
7.2	3	6	7.2	6.2
4.2	7.8	5	6.5	5.2
5.2	3.8	5.5	2	3.5
4	5	5.2	8.2	1.2

(a) Racine (2016)



12.5 Fusion de raster

Il est possible de fusionner plusieurs objets `SpatRaster` en un seul avec `merge()` ou `mosaic()`.

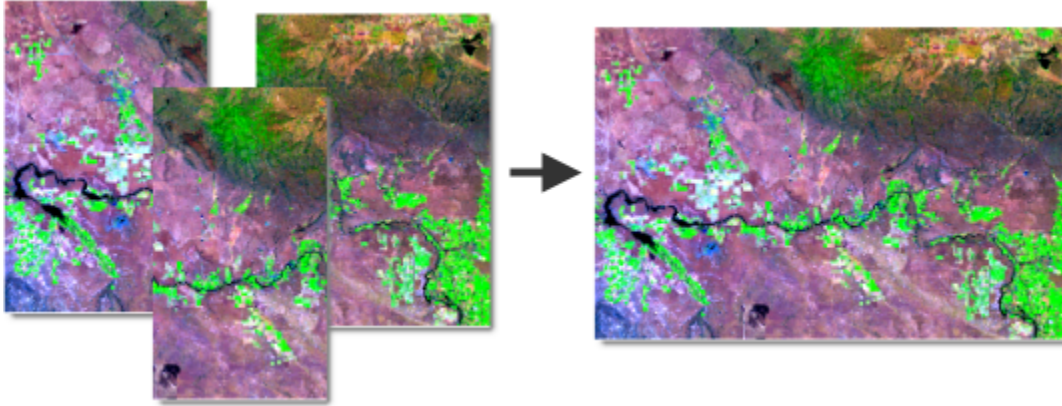


Figure 12.3: [Site web ESRI](#)

Après un découpage du raster d'élévation par la limite communale de Cahors (voir Section 12.2), nous réalisons la même chose pour la commune limitrophe de Bellefont-La Rauze.

```
# Extraction des limites communales de Bellefont-La Rauze
bellefont <- subset(commune, commune$INSEE_COM == "46156")

# Découpage du raster d'élévation en fonction des limites de Bellefont-La Rauze
crop_bellefont <- crop(elev, bellefont)
```

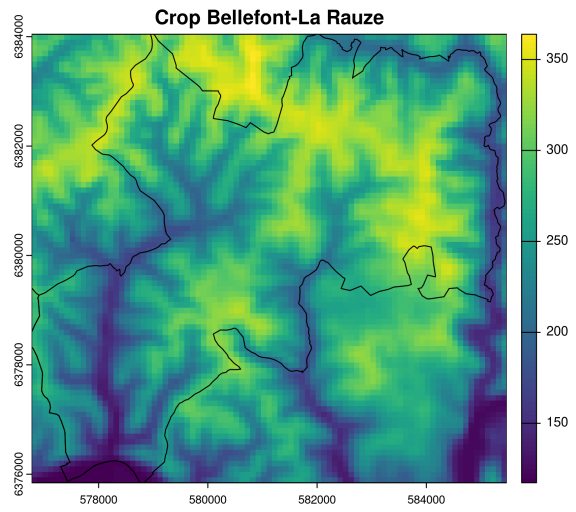
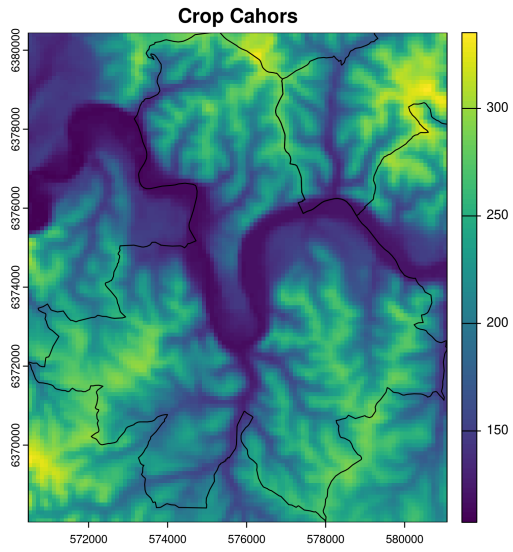
Les rasters d'élévation `crop_cahors` et `crop_bellefont` se chevauchent spatialement :

```
plot(crop_cahors, main = "Crop Cahors")
plot(cahors, add = TRUE)
plot(bellefont, add = TRUE)
plot(crop_bellefont, main = "Crop Bellefont-La Rauze")
plot(bellefont, add = TRUE)
plot(cahors, add = TRUE)
```

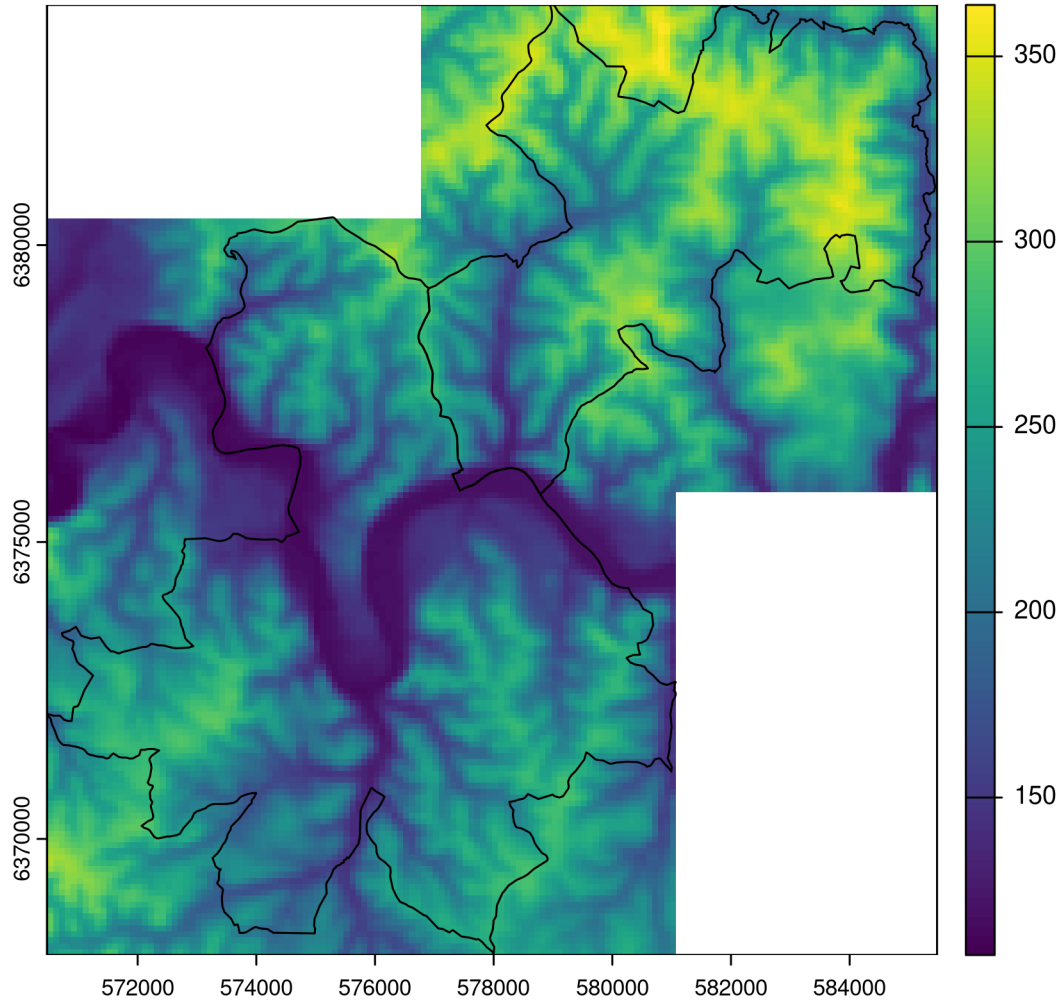
La différence entre les fonctions `merge()` ou `mosaic()` concerne les valeurs des cellules qui se superposent. La fonction `mosaic()` calcule la valeur moyenne tandis que `merge()` conserve la valeur du premier objet `SpatRaster` appelé dans la fonction.

```
# Dans cet exemple, merge() et mosaic() donnent le même résultat
merge_raster <- merge(crop_cahors, crop_bellefont)
mosaic_raster <- mosaic(crop_cahors, crop_bellefont)

plot(merge_raster)
```



```
plot(bellefont, add = TRUE)  
plot(cahors, add = TRUE)
```

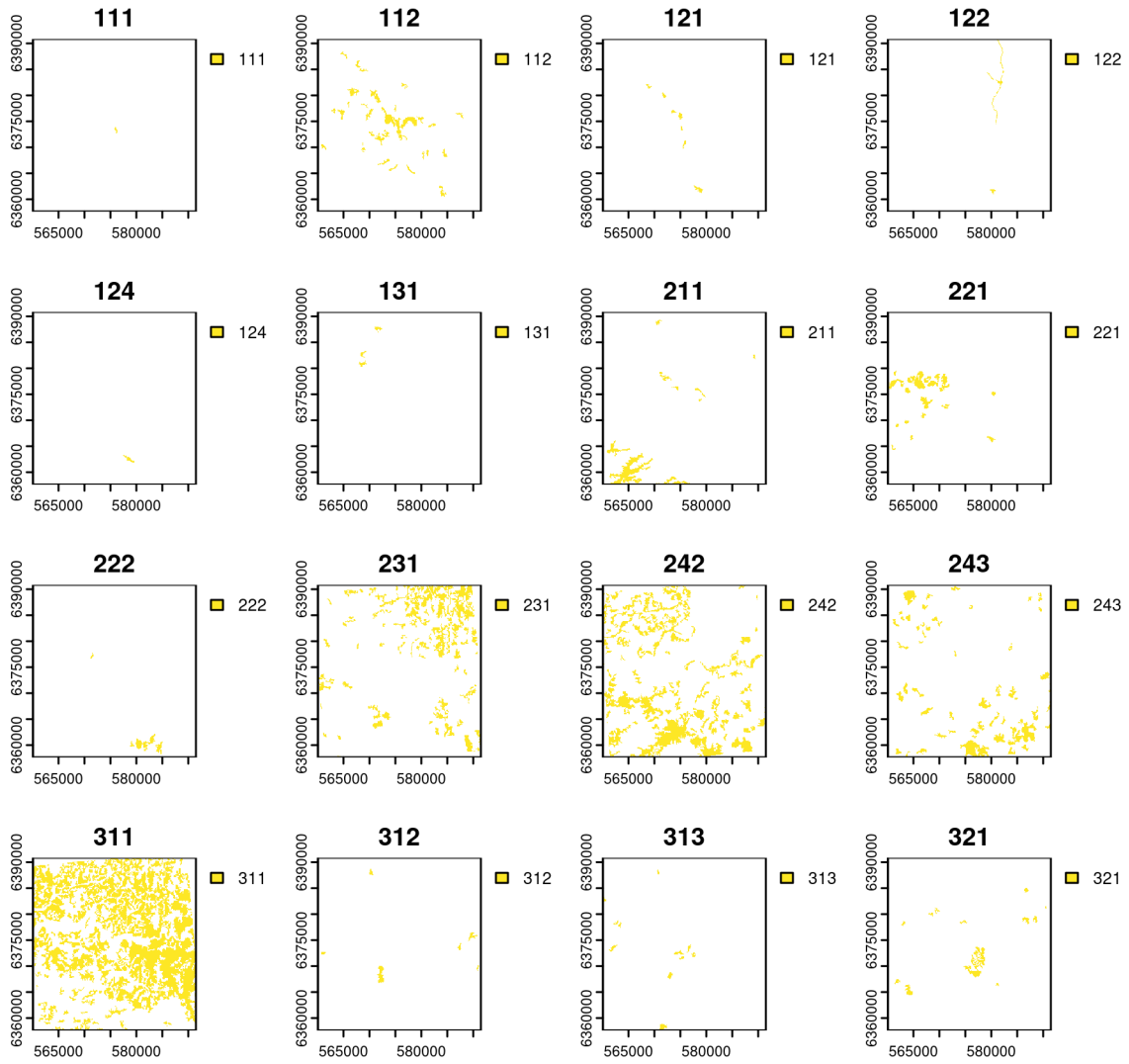


12.6 Segregate

La fonction `segregate()` permet de décomposer un raster par valeur (ou modalité) en différentes couches matricielles.

```
clc_by_class <- segregate(clc, keep = TRUE, other = NA)

plot(clc_by_class)
```

13 Algèbre spatiale

L'algèbre spatiale se classe en quatre groupes d'opération (Tomlin, 1990) :

- **Local** : opération par cellule, sur une ou plusieurs couches;
- **Focal** : opération de voisinage (cellules environnantes);
- **Zonal** : pour résumer les valeurs matricielles pour certaines zones, généralement irrégulières;
- **Global** : pour résumer les valeurs matricielles d'une ou plusieurs matrices.

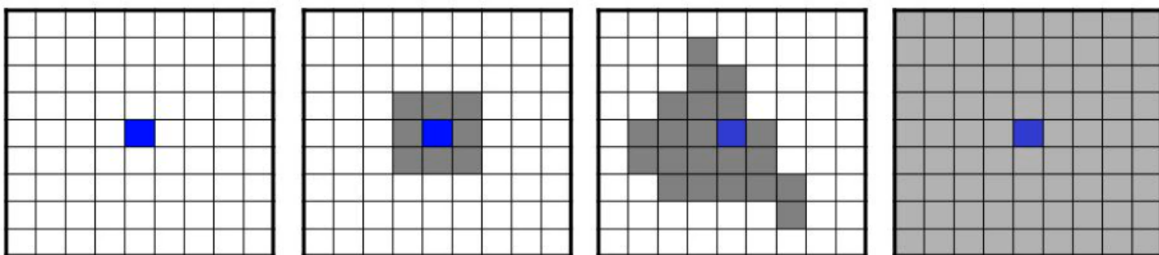


Figure 13.1: X. Li (2009)

13.1 Opérations locales

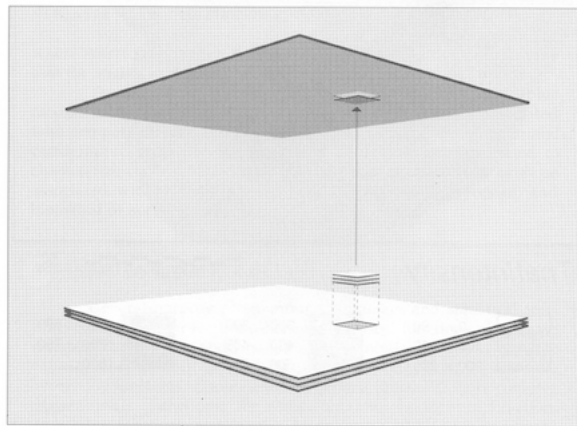


Figure 4-5 Functions of multiple values associated with individual locations. Operations *LocalArcTangent*, *LocalCombination*, *LocalDifference*, *LocalMajority*, *LocalMaximum*, *LocalMinimum*, *LocalMinority*, *LocalMean*, *LocalProduct*, *LocalRating*, *LocalRatio*, *LocalRoot*, *LocalSum*, and *LocalVariety* can all be used to compute a new value for each location as a specified function of that location's values on two or more existing map layers.

Figure 13.2: Mennis (2015)

Les opérations locales concernent les calculs réalisés indépendamment sur une cellule, à partir d'une ou plusieurs couches (matrices).

13.1.1 Remplacement de valeur

```
# Remplace les valeurs -9999 par NA
elev[elev[[1]]== -9999] <- NA

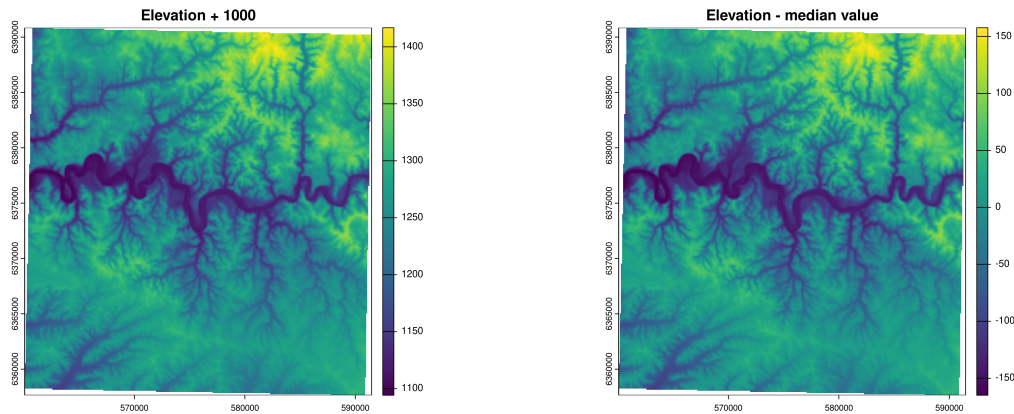
# Remplace les valeurs < 1500 par NA
elev[elev < 1500] <- NA

# Remplace les valeurs NA par 0
elev[is.na(elev )] <- 0
```

13.1.2 Opération sur chaque cellule

```
# Ajout de 1000 à la valeur de chaque cellule
elev_1000 <- elev + 1000
```

```
# Suppression de l'altitude médiane à la valeur de chaque cellule
elev_med <- elev - global(x = elev, fun = median, na.rm = TRUE)[[1]]
```



13.1.3 Reclassification

La reclassification des valeurs d'un raster peut aussi bien être utilisée pour discrétiser des données quantitatives que pour catégoriser des modalités qualitatives.

Cela permet par exemple de répartir les [44 postes de la nomenclature CLC](#) selon les 5 grands types d'occupation du territoire : territoires artificialisés, agricoles, forêts et milieux semi-naturels, zones humides et surfaces en eau. Pour cela, il est d'abord nécessaire de construire une table de correspondance avec la fonction `matrix()`.

```
reclassif <- matrix(c(100, 199, 1,
                     200, 299, 2,
                     300, 399, 3,
                     400, 499, 4,
                     500, 599, 5),
                   ncol = 3,
                   byrow = TRUE)
```

```
reclassif
```

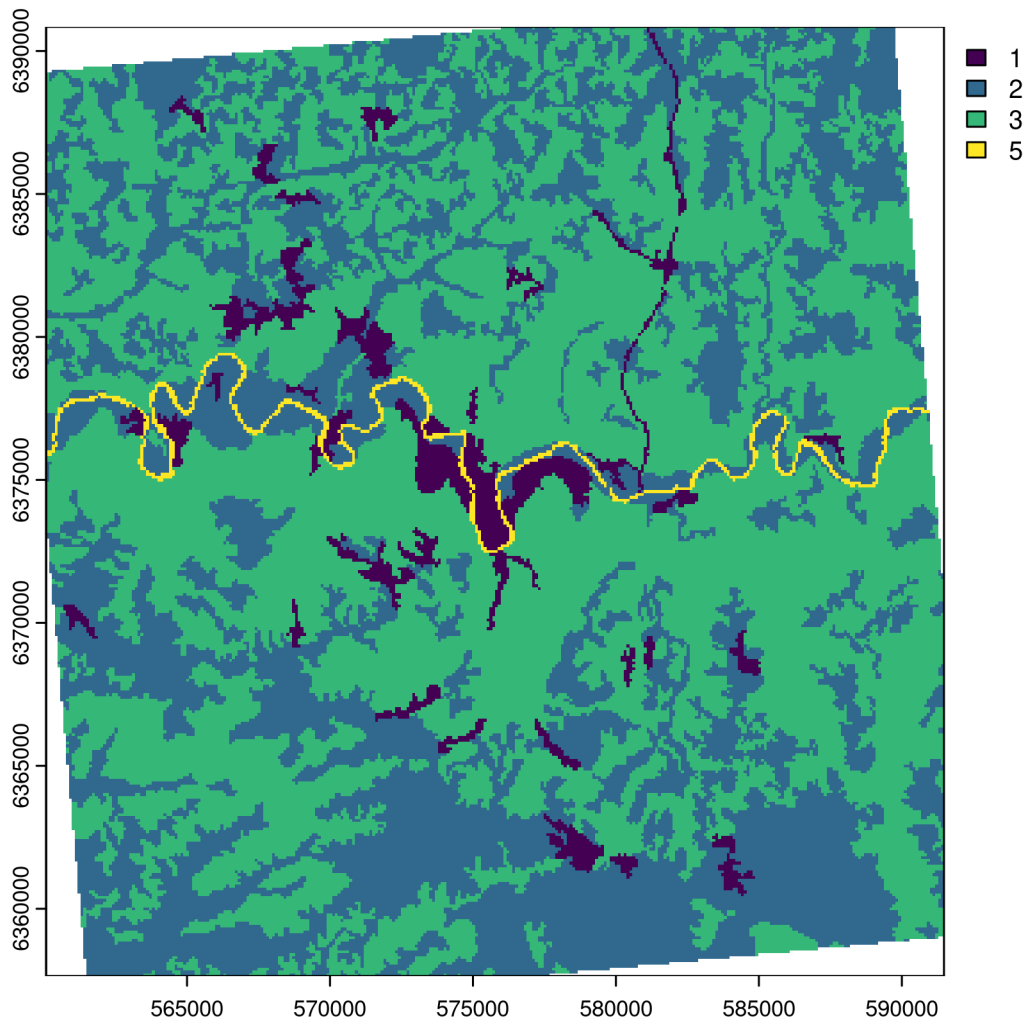
```
#>      [,1] [,2] [,3]
#> [1,] 100 199   1
#> [2,] 200 299   2
#> [3,] 300 399   3
#> [4,] 400 499   4
#> [5,] 500 599   5
```

Les valeurs comprises entre 100 et 199 seront remplacées par la valeur 1. Les valeurs comprises entre 200 et 299 seront remplacées par la valeur 2. Les valeurs comprises entre 300 et 399 seront remplacées par la valeur 3. ...

La fonction `classify()` permet de réaliser la reclassification.

```
clc_5 <- classify(clc, rcl = reclassif)

plot(clc_5, type = "classes")
```

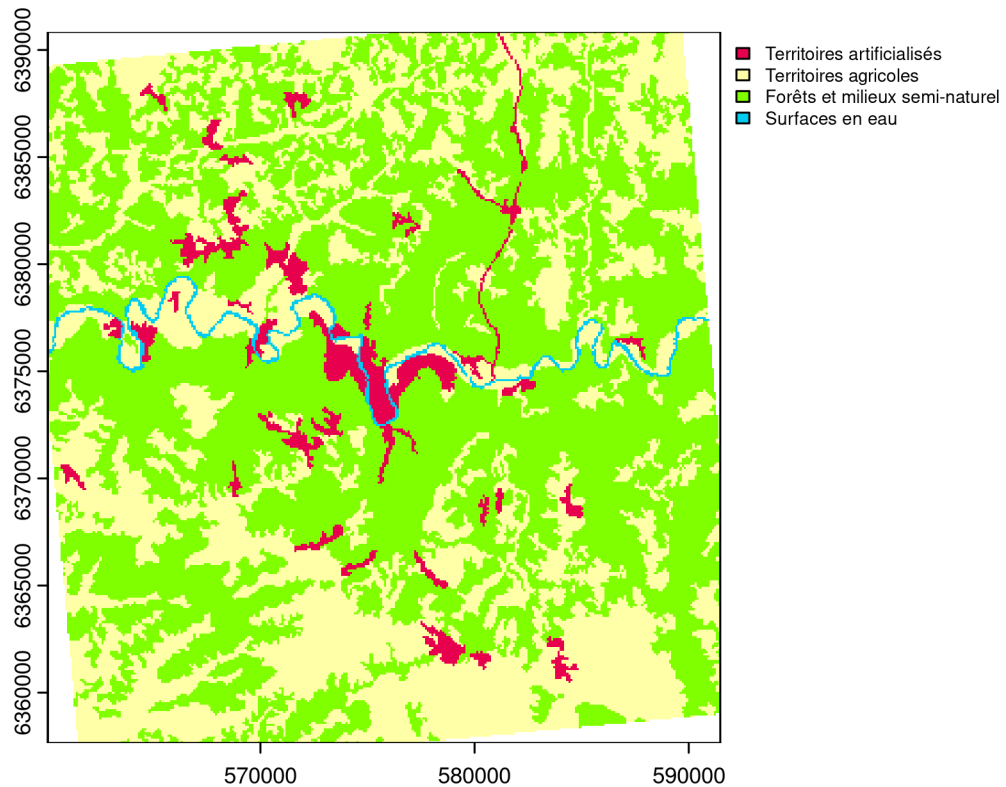


Affichage avec les intitulés et couleurs officiels des différentes catégories.

```

plot(clc_5,
     type = "classes",
     levels = c("Territoires artificialisés",
                "Territoires agricoles",
                "Forêts et milieux semi-naturels",
                "Surfaces en eau"),
     col = c("#E6004D", "#FFFA8", "#80FF00", "#00CCF2"),
     plg = list(cex = 0.7))

```



13.1.4 Opération sur plusieurs couches (ex: NDVI)

Il est possible de calculer une valeur de cellule à partir de différentes valeurs stockées dans plusieurs couches d'un objet `SpatRaster`.

L'exemple le plus courant est sans doute le calcul de l'[indice de végétation normalisé \(NDVI\)](#). Pour chaque cellule, on calcule une valeur à partir de deux couches de données matricielles d'une image satellite multispectrale.

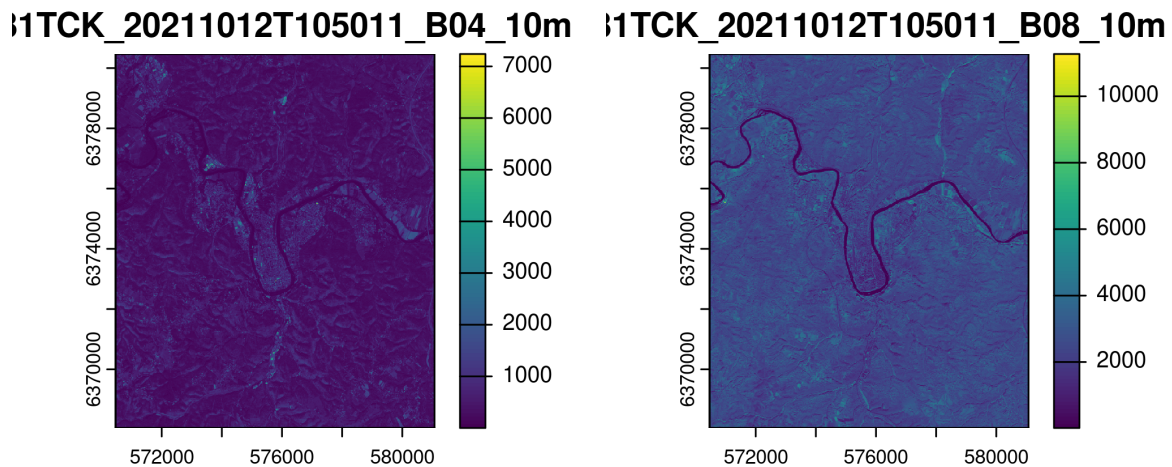
```
# Import d'une image satellite multispectrale
Sentinel2a <- rast("data/Sentinel2A.tif")
```

Sentinel2a

```
#> class      : SpatRaster
#> dimensions  : 1242, 1061, 2 (nrow, ncol, nlyr)
#> resolution  : 9.997187, 9.997187 (x, y)
#> extent      : 570465.7, 581072.7, 6368052, 6380468 (xmin, xmax, ymin, ymax)
#> coord. ref. : RGF93 v1 / Lambert-93 (EPSG:2154)
#> source      : Sentinel2A.tif
#> names       : T31TCK_20211012T105011_B04_10m, T31TCK_20211012T105011_B08_10m
#> min values  : 1, 4
#> max values  : 16112, 11273
```

Cette image satellite multispectrale (résolution de 10m) datée du 12/10/2021, a été produite par le satellite *Sentinel-2* et a été récupérée sur la [plateforme Copernicus Open Access Hub](#). Une extraction des bandes spectrales Rouge et proche infrarouge, centrée sur le département du Lot a ensuite été réalisée.

```
plot(Sentinel2a)
```



Pour alléger le code, on assigne les couches matricielles dans deux objets `SpatRaster` différents.

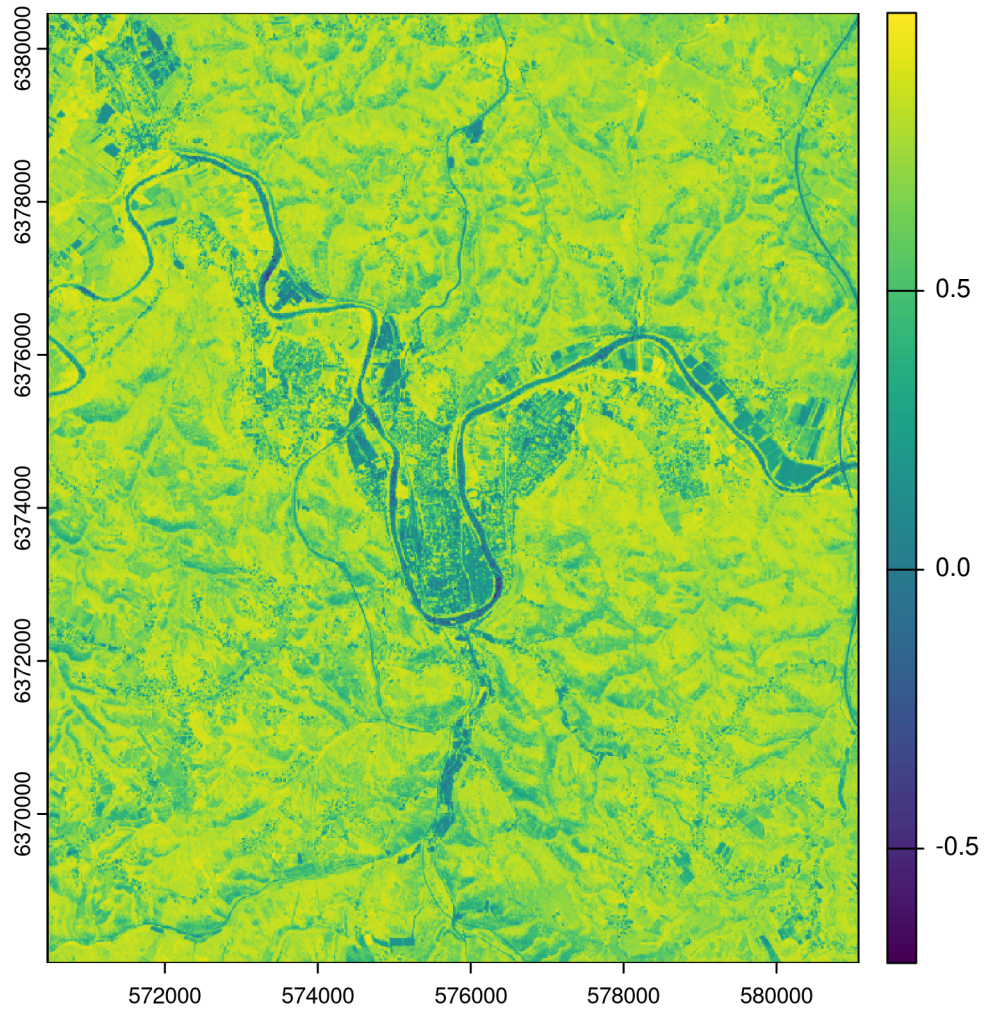
```
# Bande spectrale rouge
B04_Red <- Sentinel2a[[1]]

# Bande spectrale proche infrarouge
B08_NIR <- Sentinel2a[[2]]
```

À partir de ces deux rasters, nous pouvons calculer l'indice de végétation normalisé :

$$NDVI = \frac{NIR - Red}{NIR + Red}$$

```
raster_NDVI <- (B08_NIR - B04_Red ) / (B08_NIR + B04_Red )
plot(raster_NDVI)
```

Plus les valeurs sont importantes (proche de 1), plus la végétation est dense.

13.2 Opérations focales

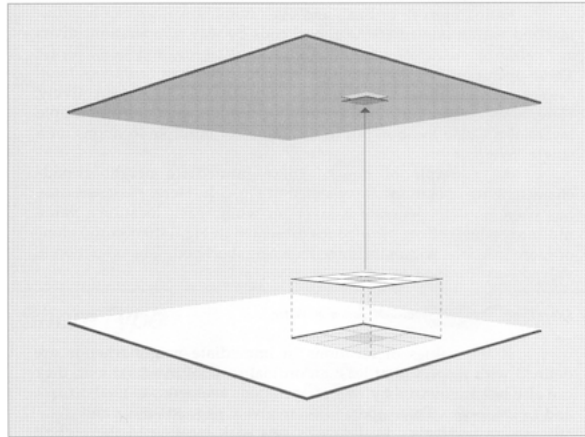


Figure 5-1 Functions of immediate neighborhoods. Operations *FocalCombination*, *FocalInsularity*, *FocalMajority*, *FocalMaximum*, *FocalMean*, *FocalMinimum*, *FocalMinority*, *FocalPercentage*, *FocalPercentile*, *FocalProduct*, *FocalRanking*, *FocalRating*, *FocalSum*, *FocalVariety*, *IncrementalArea*, *IncrementalAspect*, *IncrementalPartition*, *IncrementalDrainage*, *IncrementalFrontage*, *IncrementalGradient*, *IncrementalLength*, *IncrementalLinkage*, and *IncrementalVolume* can all be used to compute a new value (above) for each location as a function of its immediate neighbors on an existing layer (below).

Figure 13.3: Mennis (2015)

L'analyse focale considère une cellule plus ses voisins directs de manière contiguë et symétrique (opérations de voisinage). Le plus souvent, la valeur de la cellule de sortie est le résultat d'un bloc de cellules d'entrée 3 x 3 (nombre impair).

La première étape consiste à construire une matrice qui détermine le bloc de cellules qui sera pris en compte autour de chaque cellule.

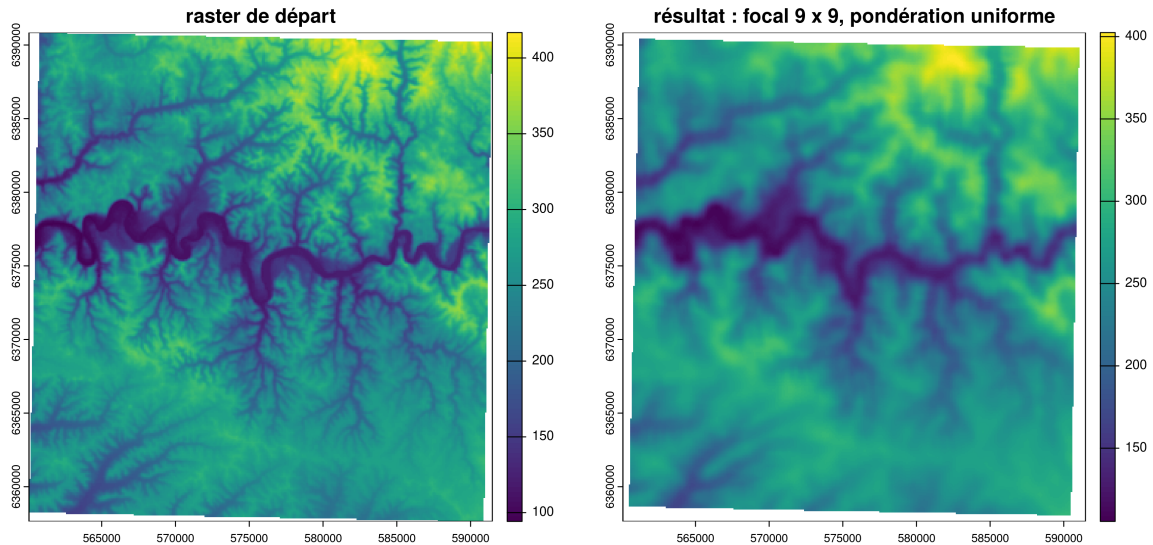
```
# Matrice 9 x 9, où chaque cellule présente la même pondération (1)
mon_focal <- matrix(1, nrow = 9, ncol = 9)
mon_focal
```

```
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
#> [1,]    1    1    1    1    1    1    1    1    1
#> [2,]    1    1    1    1    1    1    1    1    1
#> [3,]    1    1    1    1    1    1    1    1    1
#> [4,]    1    1    1    1    1    1    1    1    1
#> [5,]    1    1    1    1    1    1    1    1    1
#> [6,]    1    1    1    1    1    1    1    1    1
#> [7,]    1    1    1    1    1    1    1    1    1
#> [8,]    1    1    1    1    1    1    1    1    1
#> [9,]    1    1    1    1    1    1    1    1    1
```

La fonction `focal()` permet ensuite de réaliser l'analyse souhaitée. Par exemple : le calcul de la moyenne des valeurs dans une fenêtre spatiale déterminée, pour chaque cellule du raster.

```
elev_focal_mean <- focal(elev, w = mon_focal, fun = mean)
```

```
plot(elev, main = "raster de départ")  
plot(elev_focal_mean, main="résultat : focal 9 x 9, pondération uniforme")
```



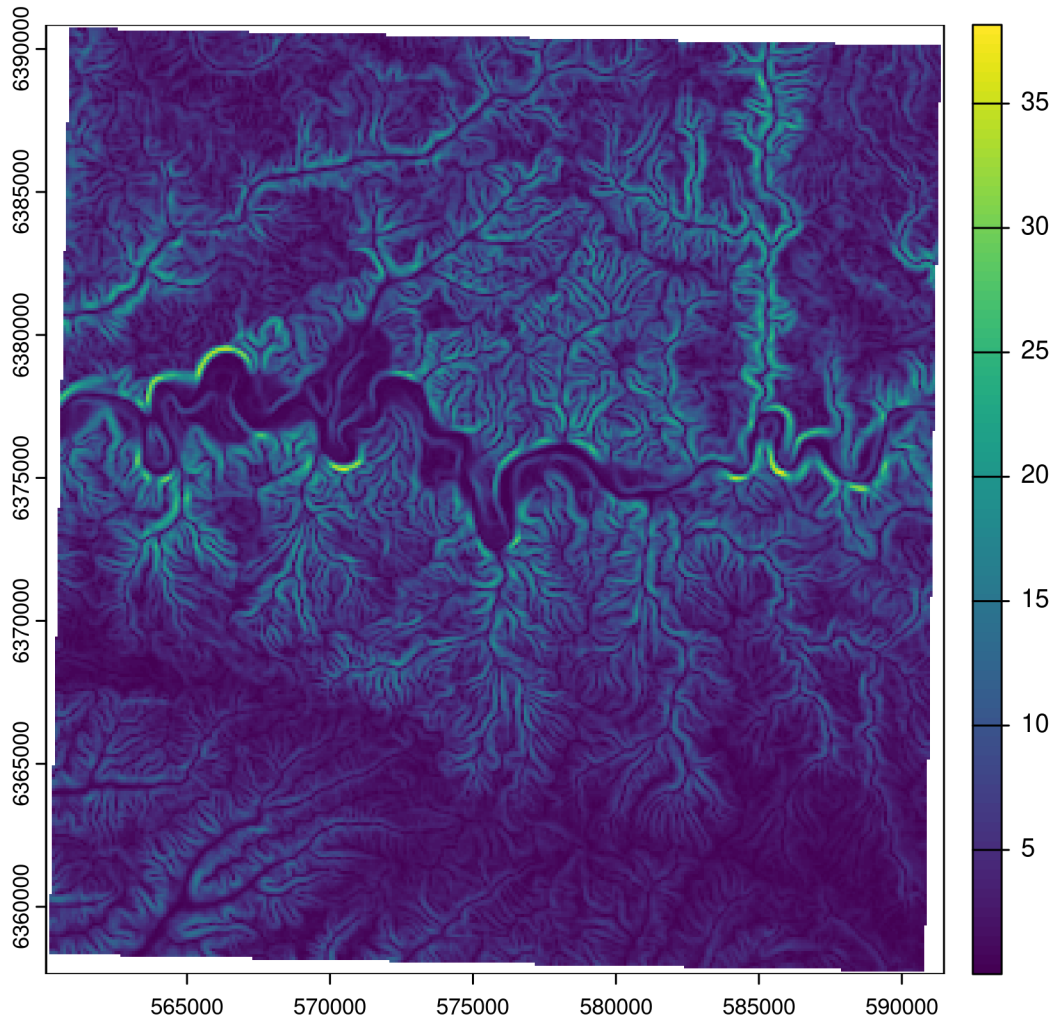
13.2.1 Opération focales pour rasters d'élévation

La fonction `terrain()` permet de réaliser des analyses focales spécifiques au rasters d'élévation. Six opérations sont disponibles :

- *slope* = calcul de la pente ou degré d'inclinaison de la surface;
- *aspect* = calcul de l'orientation de la pente;
- *roughness* = calcul de la variabilité ou l'irrégularité de l'élévation;
- *TPI* = calcul de l'indice des positions topographiques;
- *TRI* = calcul de l'indice de la variabilité de l'élévation;
- *flowdir* = calcul du sens d'écoulement de l'eau.

Exemples avec le calcul des pentes (*slope*), c'est à dire leur inclinaison en degrés.

```
slope <- terrain(elev, "slope",  
                neighbors = 8, # 8 (ou 4) cellules autour pris en compte  
                unit = "degrees") # Unité en sortie  
  
plot(slope)
```



13.3 Opérations globales

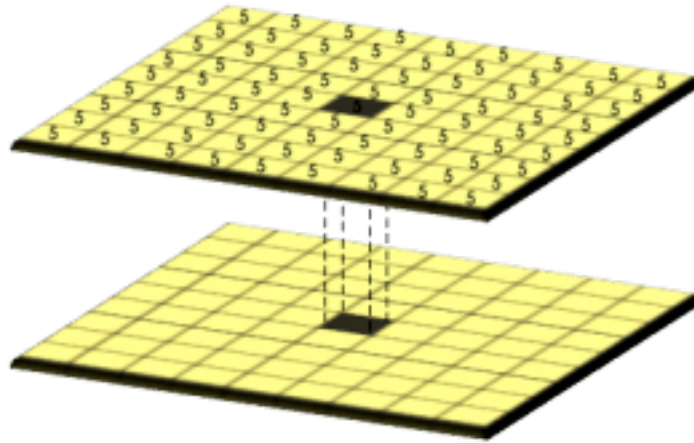


Figure 13.4: gisgeography.com

Les opérations globales permettent de résumer les valeurs matricielles d'une ou plusieurs matrices.

```
# Valeur moyenne  
global(elev, fun = "mean", na.rm = TRUE)
```

```
#>          mean  
#> altitude 251.3601
```

```
# Écart-type  
global(elev, fun = "sd", na.rm = TRUE)
```

```
#>          sd  
#> altitude 54.58627
```

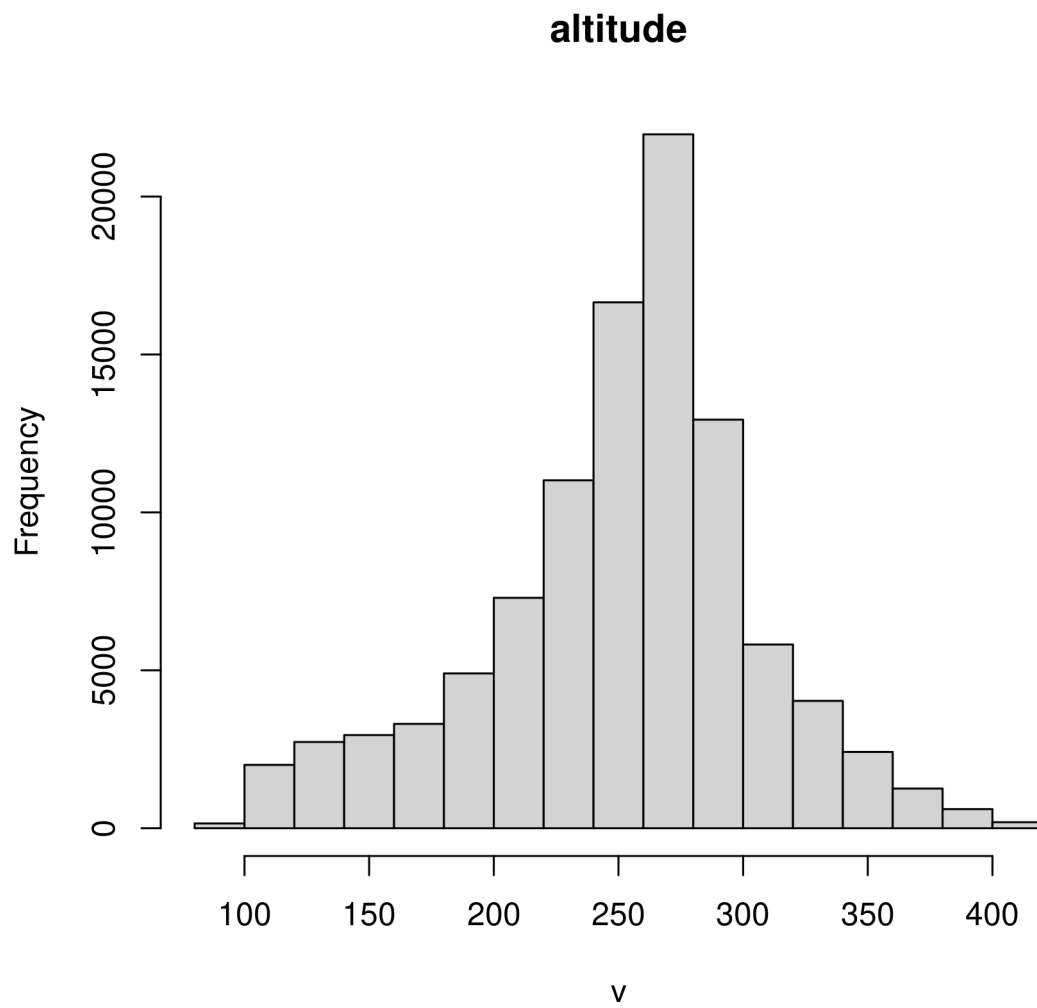
```
# Fréquence  
freq(clc_5)
```

```
#> layer value count
```

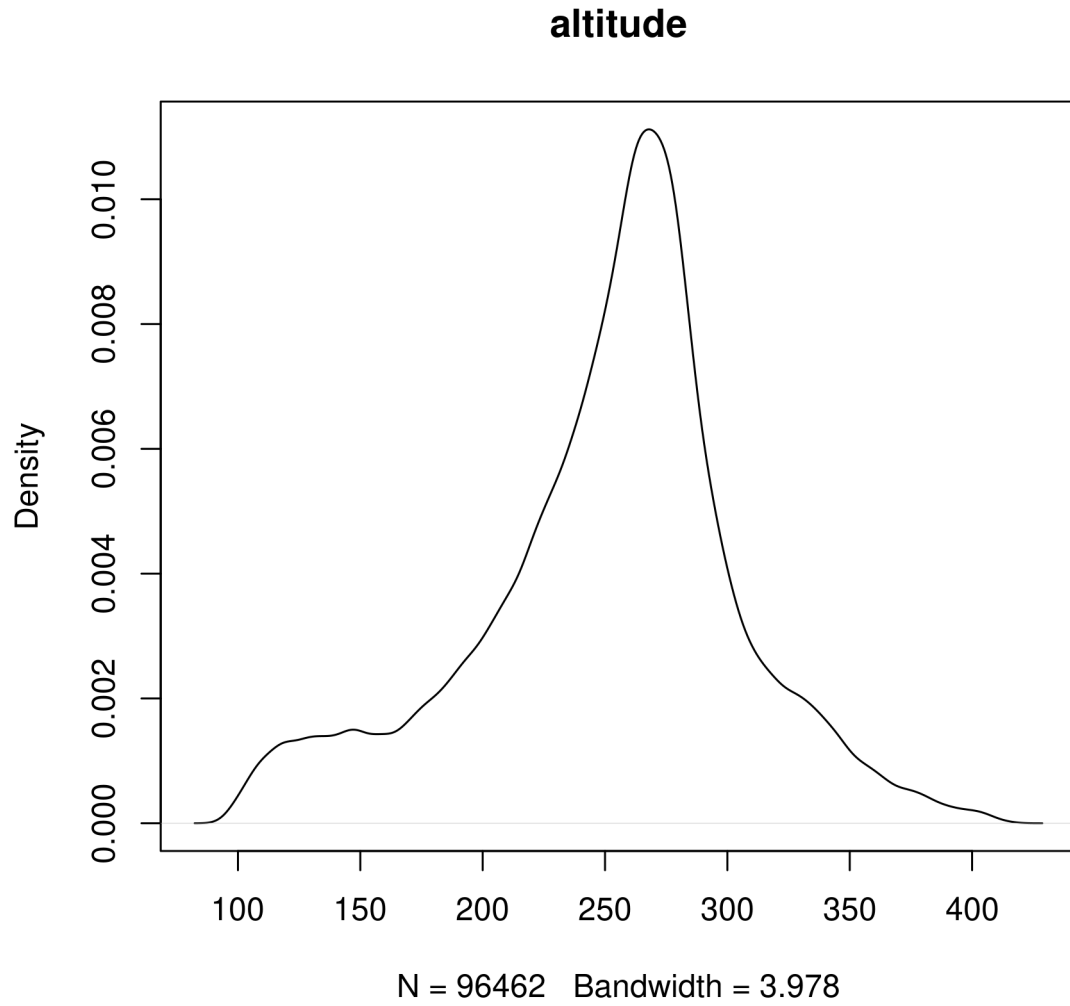
```
#> 1 1 1 3775
#> 2 1 2 38118
#> 3 1 3 56331
#> 4 1 5 928
```

Représentations statistiques qui résument les informations matricielles.

```
# Histogramme
hist(elev)
```

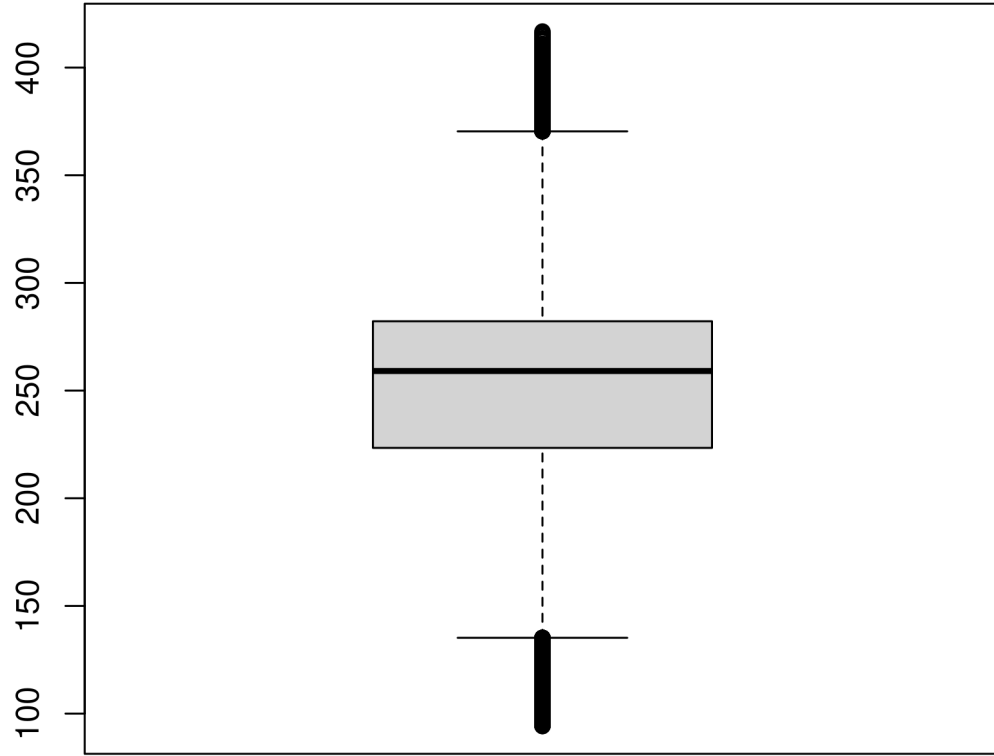


```
# Densité  
density(elev)
```



```
# boxplot  
boxplot(elev)
```

```
#> Warning: [boxplot] taking a sample of 1e+05 cells
```



13.4 Opérations zonales

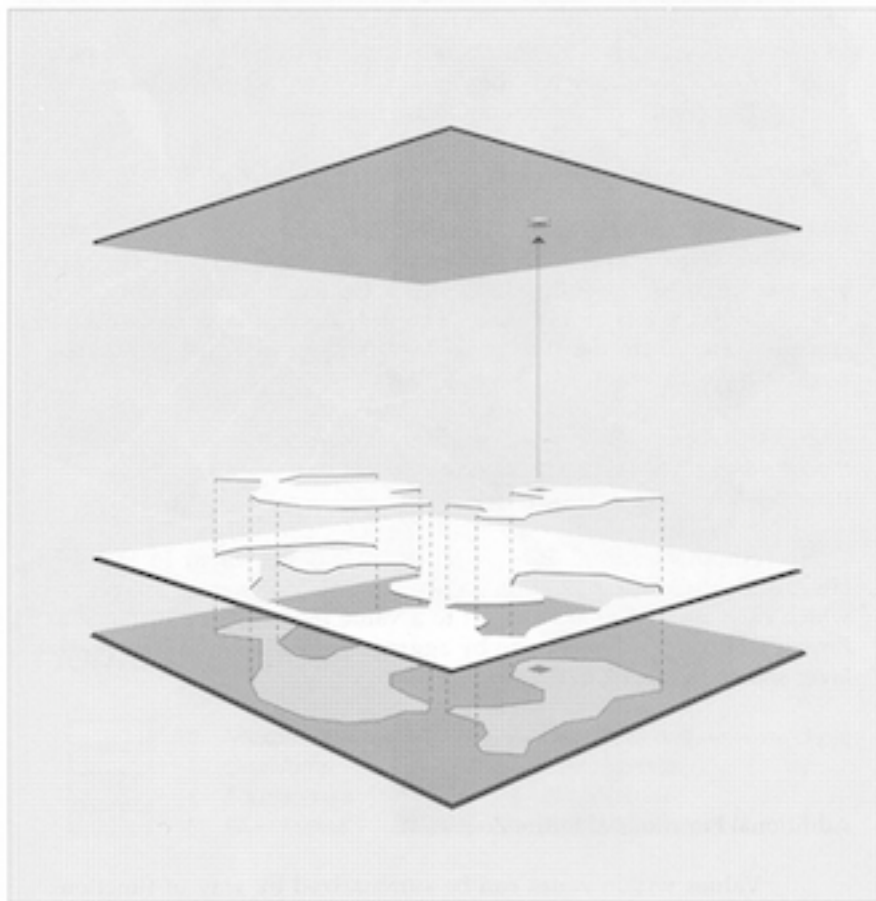


Figure 6-1 Functions of entire zones. Operations *ZonalCombination*, *ZonalMajority*, *ZonalMaximum*, *ZonalMean*, *ZonalMinimum*, *ZonalMinority*, *ZonalProduct*, *ZonalRating*, *ZonalSum*, and *ZonalVariety* can all be used to compute a new value (above) for each location as a specified function of existing values within a common zone (below).

Figure 13.5: Mennis (2015)

Les opérations zonales permettent de résumer les valeurs matricielles de certaines zones (groupe de cellules contiguës dans l'espace ou en valeur).

13.4.1 Opération zonale à partir d'une couche vectorielle

La fonction `extract()` permet d'extraire et de manipuler les valeurs des cellules qui intersectent des données vectorielles.

Exemple à partir des limites communales :

```
commune <- vect("data/lot.gpkg", layer = "communes")

# Moyenne d'élévation pour chaque polygone (commune)
elev_by_com <- extract(elev, commune, fun = mean, na.rm = FALSE)
head(elev_by_com, n = 3)
```

```
#>   ID altitude
#> 1  1      NaN
#> 2  2      NaN
#> 3  3      NaN
```

```
# Suppression des valeurs NaN
elev_by_com <- elev_by_com[!is.nan(elev_by_com$altitude),]

# Remplacement des identifiants uniques par le nom des communes
elev_by_com$ID <- commune[elev_by_com$ID]$NOM_COM
head(elev_by_com, n = 6)
```

```
#>           ID altitude
#> 7           Arcambal 213.0009
#> 10          Aujols 216.7961
#> 23 Belmont-Sainte-Foi 302.4436
#> 29          Boissières 252.0305
#> 39          Cahors 205.1988
#> 41          Caillac 145.9868
```

13.4.2 Opération zonale à partir d'un raster

Les opérations zonales peuvent être réalisées par zone délimitée par les valeurs catégorielles d'un second raster avec la fonction `zonal()`. **Pour cela, les deux rasters doivent avoir exactement le même étendue et la même résolution.**

```
# Élévation moyenne pour chaque zone de clc
zonal(elev, clc_5, "mean", na.rm = TRUE)
```

```
#>   clc altitude
#> 1  1 209.7177
#> 2  2 259.3624
```

```
#> 3 3 249.4070  
#> 4 5 115.8435
```

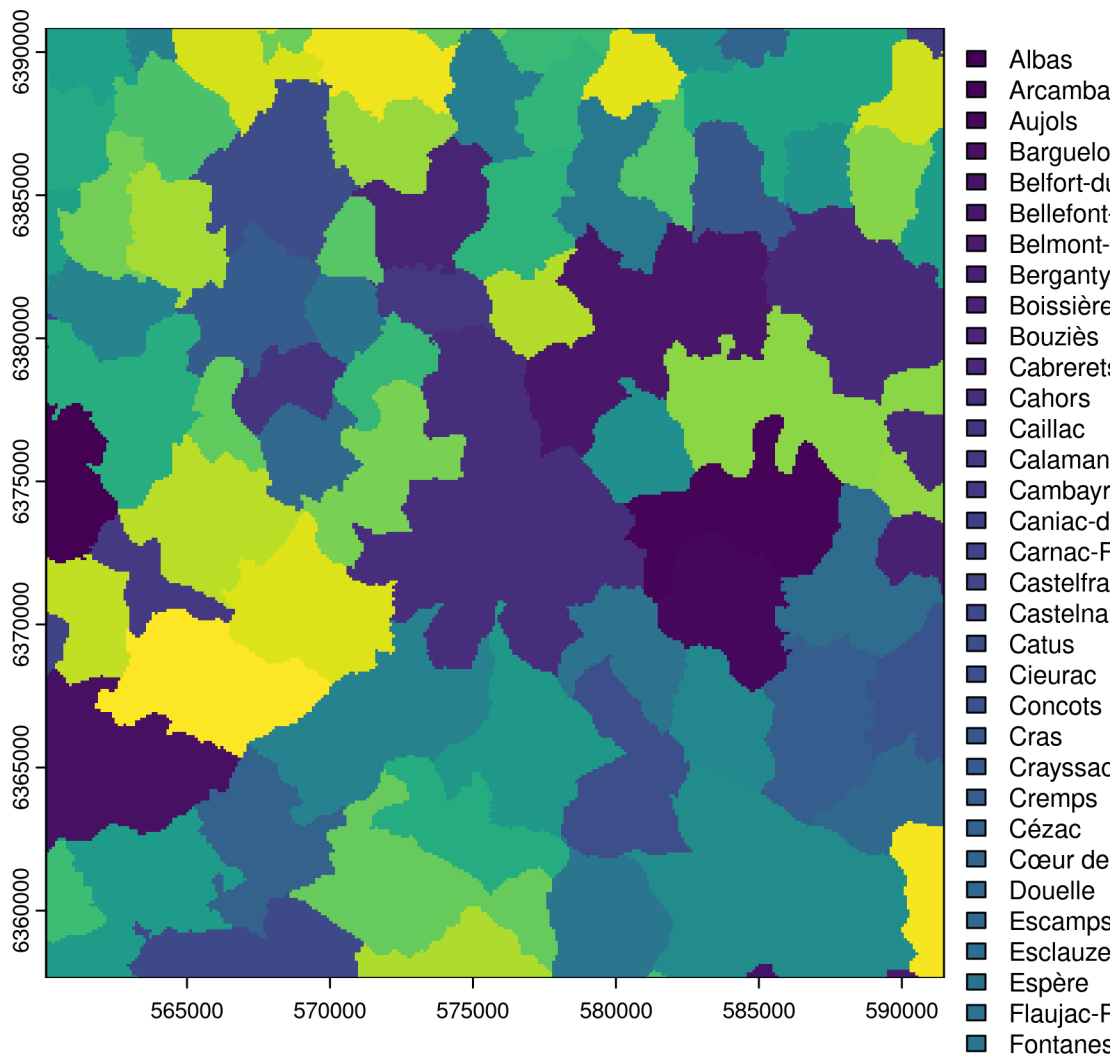
14 Conversions

14.1 Rasterisation

Transformer des polygones en format raster avec la fonction `rasterize()`.

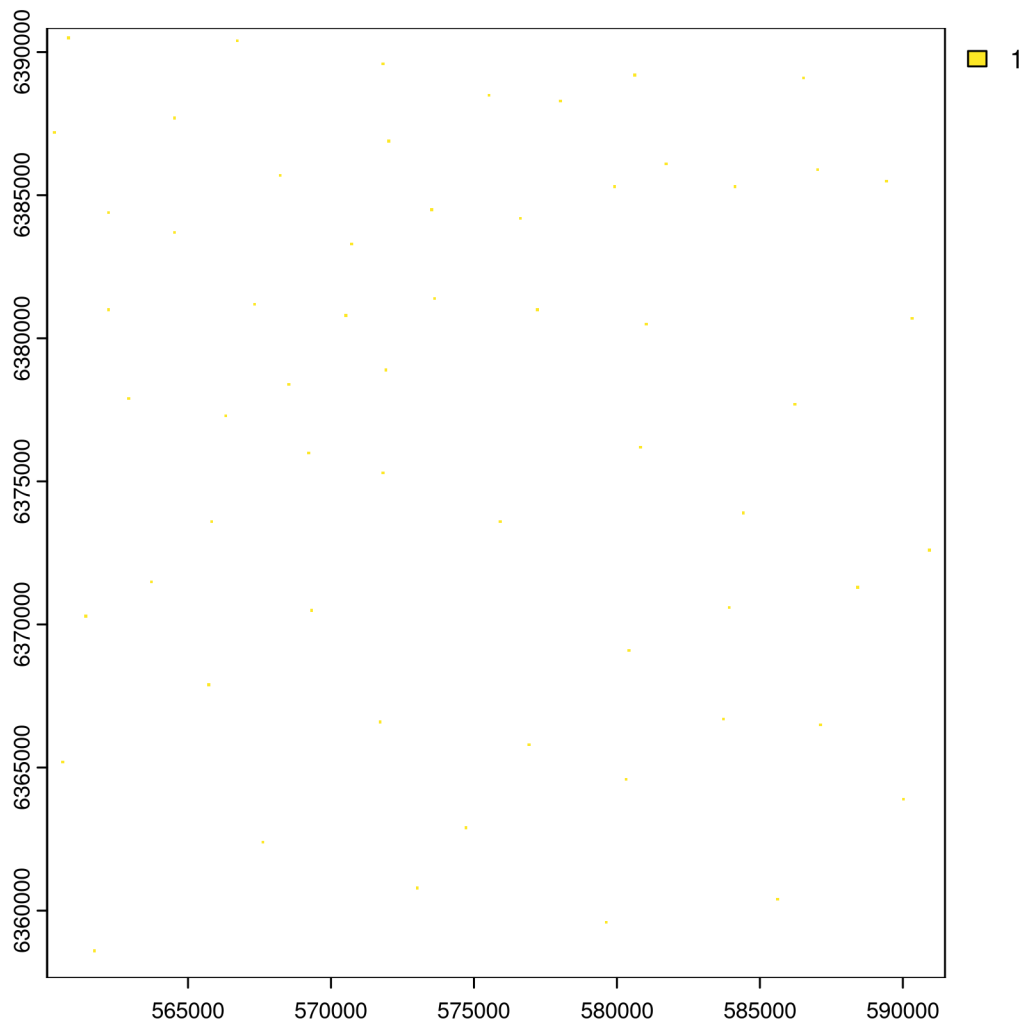
```
commune <- vect("data/lot.gpkg", layer = "communes")
elev <- rast("data/elev.tif")

raster_commune <- rasterize(x = commune, y = elev , field = "NOM_COM")
plot(raster_commune)
```



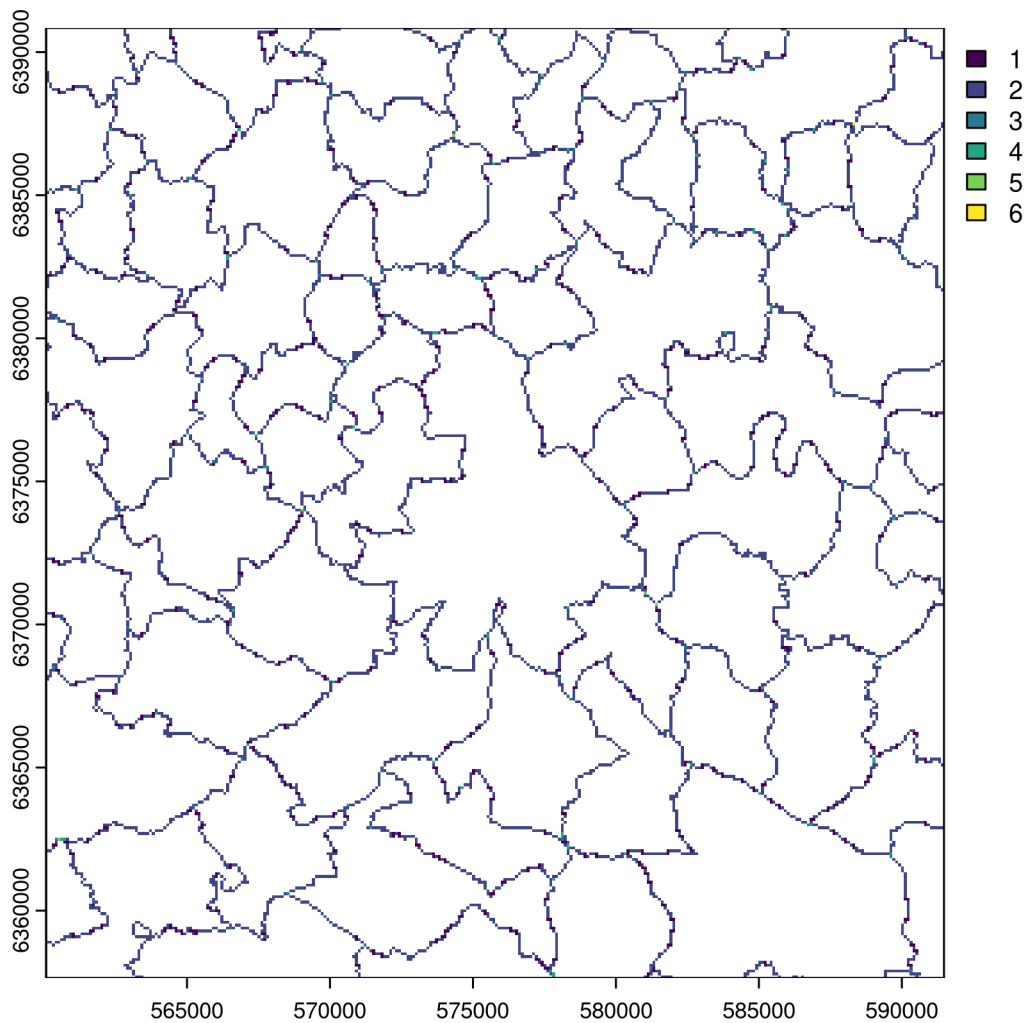
Transformer des points en format raster :

```
# Rasterisation des centroïdes des communes
raster_com_centroide <- rasterize(x = centroids(commune),
                                y = elev, fun = sum)
plot(raster_com_centroide)
```



Transformer des lignes format raster :

```
# Rasterisation des limites communales  
raster_com_line <- rasterize(x = as.lines(commune), y = elev, fun=sum)  
plot(raster_com_line)
```



14.2 Vectorisation

Transformer un raster en polygones ou en points avec les fonctions `as.polygons()` et `as.points()`. Les objets créés sont dans le format `SpatVector` de `terra`. Il est ensuite possible de les transformer en objets `sf` avec la fonction `st_as_sf()`.

```
library(terra)
library(sf)
clc <- rast(x = "data/clc.tif")
```

```

# Reclassifions le raster CLC
reclassif <- matrix(c(100, 199, 1,
                     200, 299, 2,
                     300, 399, 3,
                     400, 499, 4,
                     500, 599, 5),
                   ncol = 3,
                   byrow = TRUE)
clc <- classify(clc, rcl = reclassif)

# Changeons d'abord la résolution du raster CLC
clc_lower_model <- clc
res(clc_lower_model) <- 1000

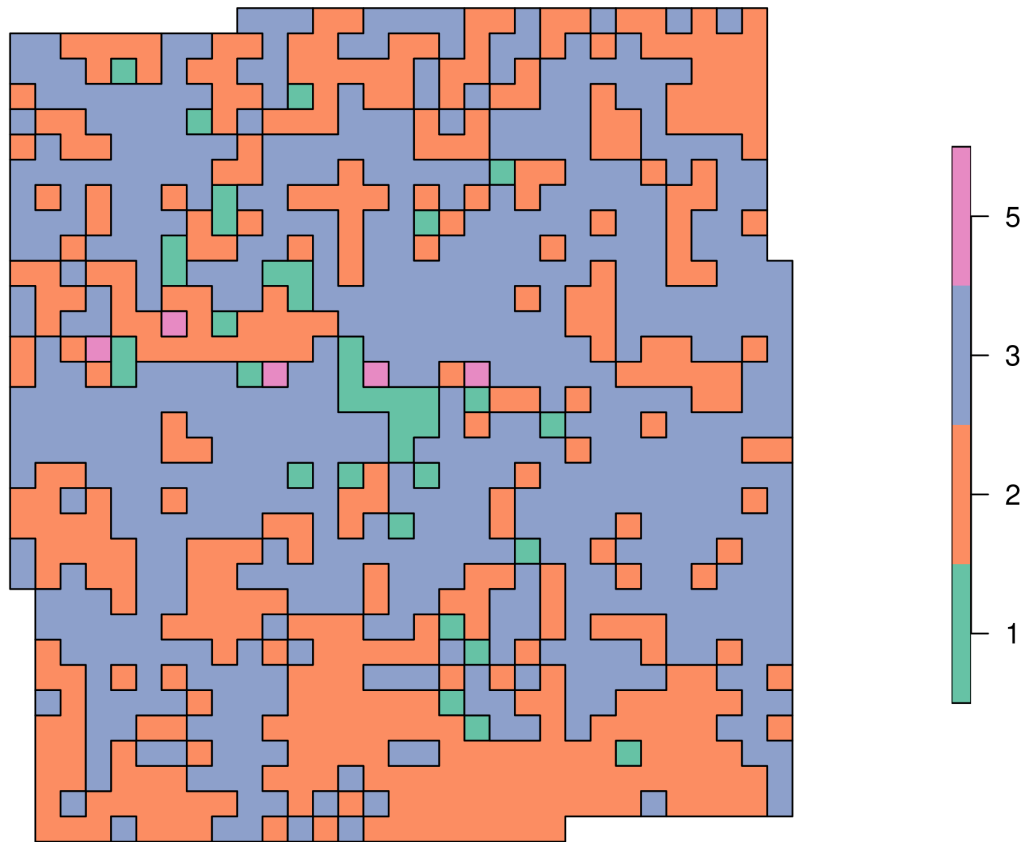
clc_lower <- resample(x = clc, y = clc_lower_model, method = "near")

# Transformation en polygones
clc_poly <- as.polygons(clc_lower)
clc_poly <- st_as_sf(clc_poly)

# Affichage
clc_poly$clc <- as.factor(clc_poly$clc)
plot(clc_poly["clc"])

```

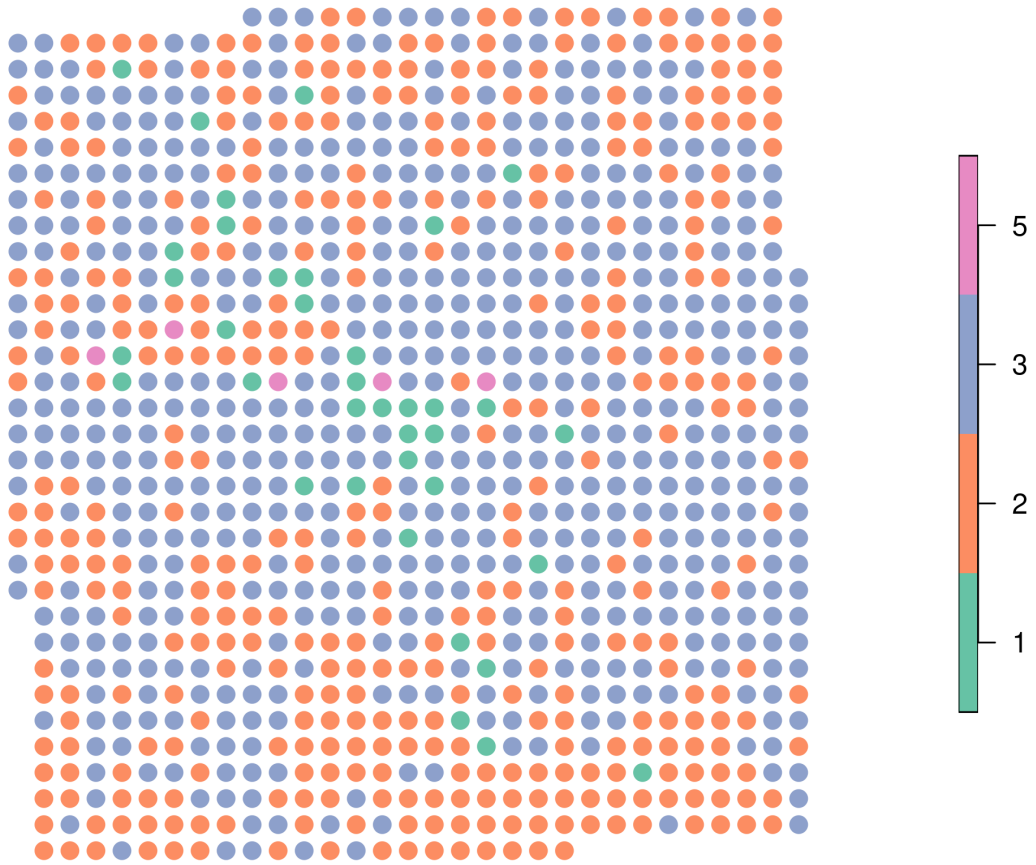

clc



Transformer un raster en points vectoriels avec la fonction `as.points()`:

```
clc_point <- as.points(clc_lower)
clc_point <- st_as_sf(clc_point)
clc_point$clc <- as.factor(clc_point$clc)
plot(clc_point["clc"], pch = 20, cex = 2)
```

clc



partie III

Focus sur OpenStreetMap

15 OpenStreetMap

[OpenStreetMap](#) (OSM) est un projet de cartographie participative qui a pour but de constituer une base de données géographiques libre à l'échelle mondiale. OpenStreetMap vous permet de voir, modifier et utiliser des données géographiques dans le monde entier.

Conditions d'utilisation

OpenStreetMap est en données ouvertes : vous êtes libre de l'utiliser pour n'importe quel but tant que vous créditez OpenStreetMap et ses contributeurs. Si vous modifiez ou vous appuyez sur les données d'une façon quelconque, vous pouvez distribuer le résultat seulement suivant la même licence. (...)

Contributeurs

(...) Nos contributeurs incluent des cartographes enthousiastes, des professionnels du SIG, des ingénieurs qui font fonctionner les serveurs d'OSM, des humanitaires cartographiant les zones dévastées par une catastrophe et beaucoup d'autres. (...)

[A propos d'OpenStreetMap](#)

16 Cartes interactives

Les deux principaux packages qui permettent d'afficher une carte interactive basée sur OSM sont `leaflet` (Cheng et al., 2023) et `mapview` (Appelhans et al., 2023).

16.1 leaflet

`leaflet` utilise la librairie javascript Leaflet (Agafonkin, 2015) pour créer des cartes interactives.

```
library(sf)
```

```
#> Linking to GEOS 3.11.1, GDAL 3.6.2, PROJ 9.1.1; sf_use_s2() is TRUE
```

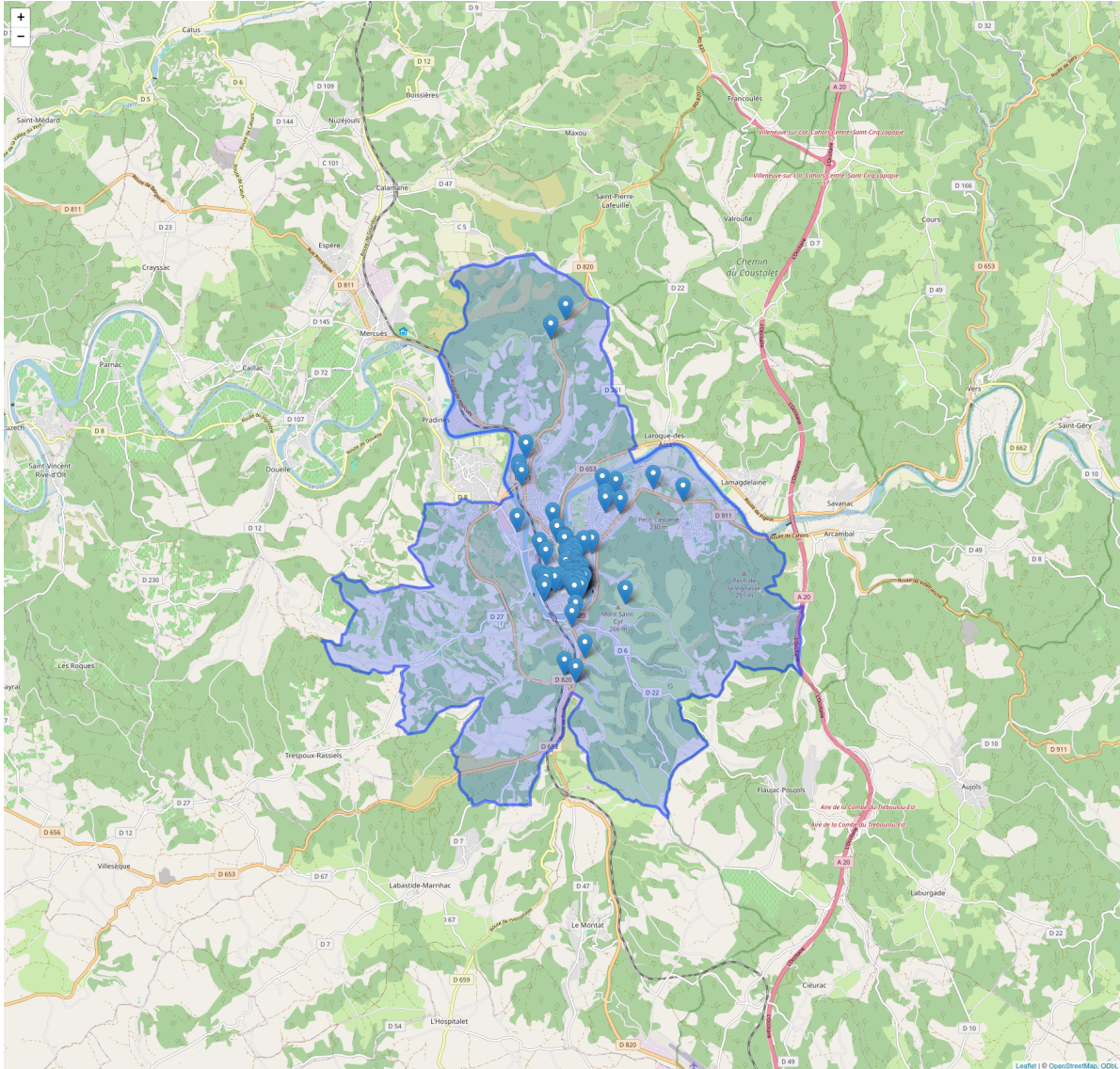
```
library(leaflet)
com <- st_read("data/lot.gpkg", layer = "communes", quiet = TRUE)
restaurant <- st_read("data/lot.gpkg", layer = "restaurants", quiet = TRUE)

# Sélection de la commune de Cahors
cahors <- com[com$INSEE_COM == "46042", ]

# Sélection des restaurants de Cahors
restaurant_cahors <- st_filter(restaurant, cahors)

# transformation du système de coordonnées en WGS84
cahors <- st_transform(cahors, 4326)
restaurant_cahors <- st_transform(restaurant_cahors, 4326)

# Création de la carte interactive
m <- leaflet(cahors) %>%
  addTiles() %>%
  addPolygons() %>%
  addMarkers(data = restaurant_cahors)
m
```



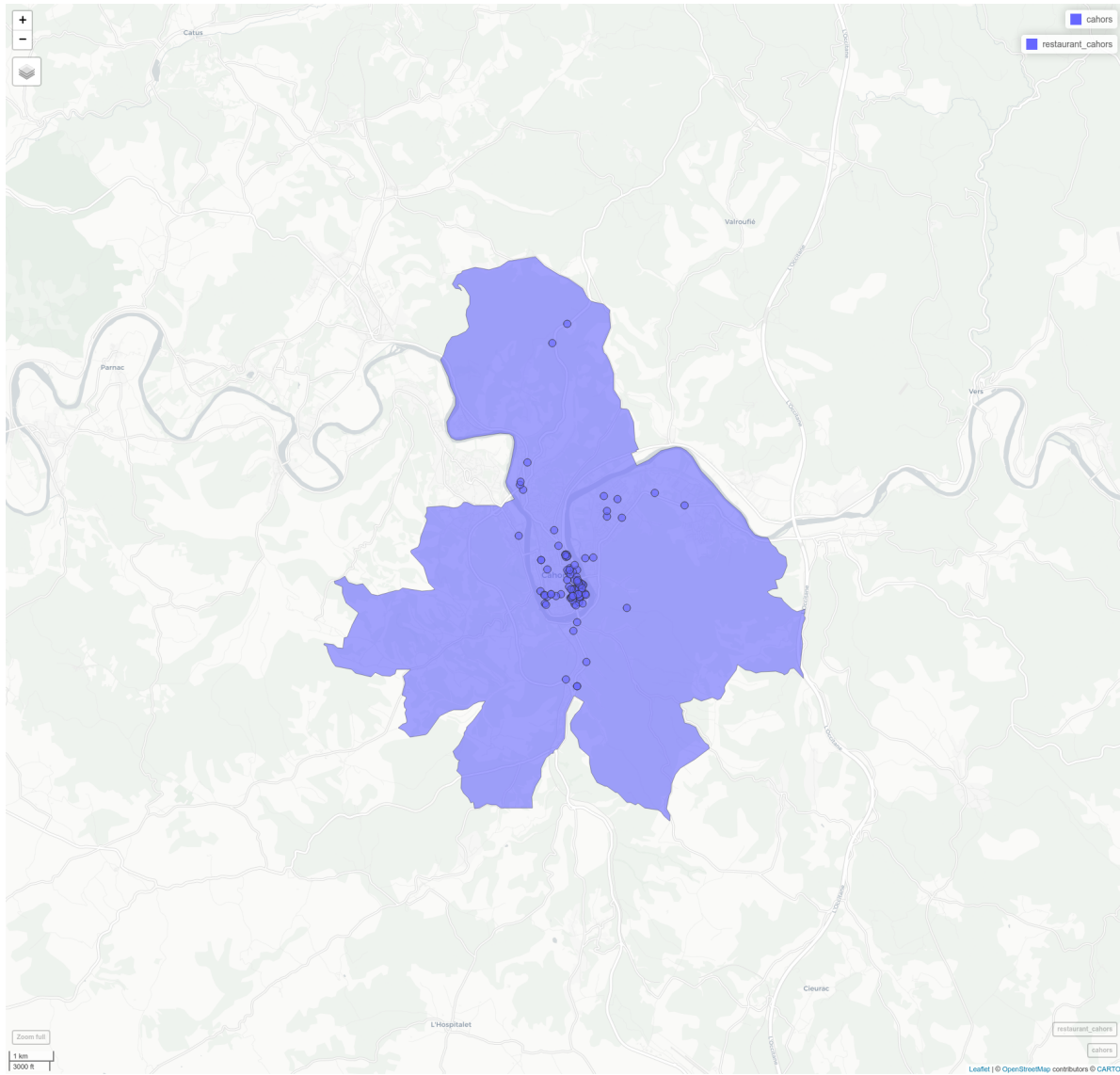
Site web de leaflet

[Leaflet for R](#)

16.2 mapview

mapview s'appuie sur leaflet pour créer des cartes interactives, son utilisation basique est plus simple bien que sa documentation ne soit pas vraiment facile à aborder.

```
library(mapview)  
mapview(cahors) + mapview(restaurant_cahors)
```



Site web de mapview [mapview](https://mapview.tidyverse.org/)

17 Import de fonds de carte

Le package `maptiles` (Giraud, 2023b) permet de télécharger et d'afficher des fonds de cartes raster. La fonction `get_tiles()` permet de télécharger des fonds de cartes OSM au format `SpatRaster` du package `terra`.

Dans cette exemple nous utilisons le package `mapsf` pour afficher la carte.

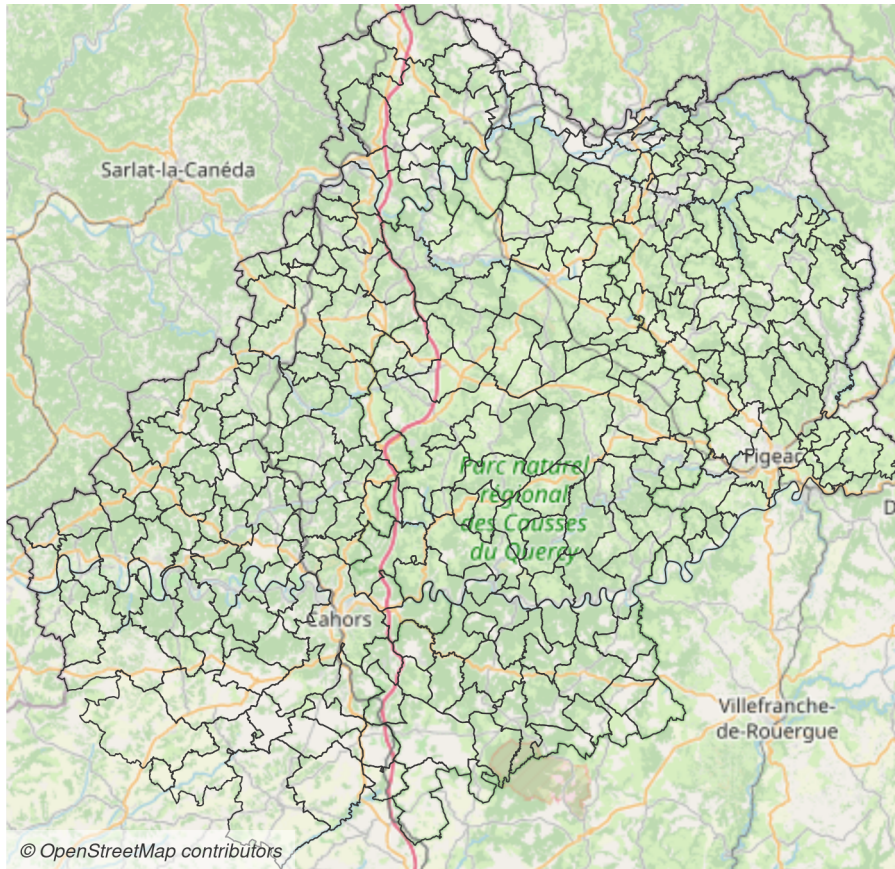
Les rendus sont meilleurs si les données en entrée utilisent le même système de coordonnées que les tuiles ([EPSG:3857](#)).

```
library(sf)
library(maptiles)
library(mapsf)

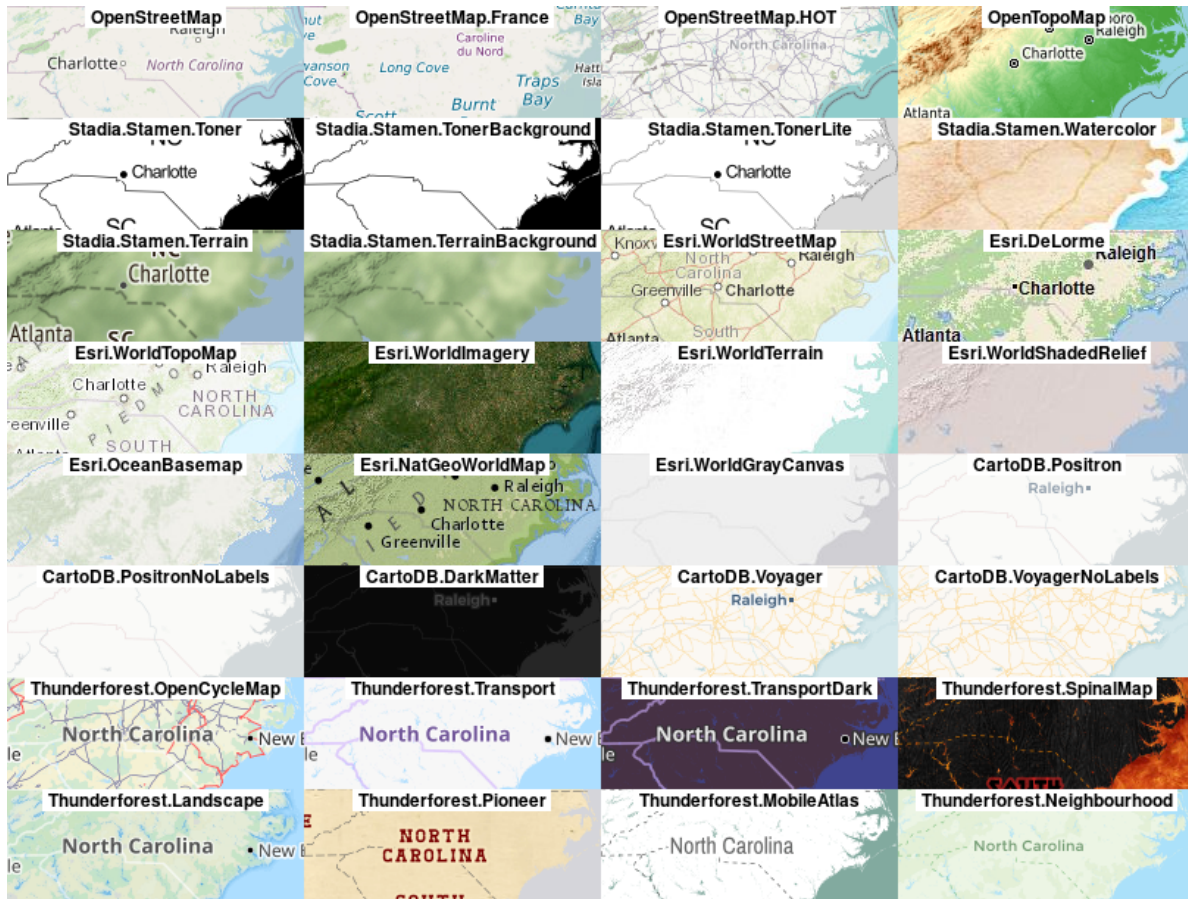
com <- st_read("data/lot.gpkg", layer = "communes", quiet = TRUE)
com <- st_transform(com, 3857)

# Récupération d'un fond de carte OSM
osm_tiles <- get_tiles(x = com, zoom = 9, crop = TRUE)

mf_theme(mar = c(0,0,0,0))
mf_raster(osm_tiles)
mf_map(com, border = "grey20", col = NA, lwd = .7, add = TRUE)
mf_credits(get_credit("OpenStreetMap"), bg = "#ffffff80")
```

De nombreux styles de tuiles sont disponibles avec le package. En voici quelque uns:



Certains styles ne contiennent que des labels et peuvent être utilisés en complément de données vectorielles:

```
# Récupération d'un fond de carte OSM des labels
mf_theme(mar = c(0,0,0,0))
osm_labels <- get_tiles(x = com, provider = "CartoDB.PositronOnlyLabels")
mf_map(com, col = 'ivory', border = 'ivory3')
mf_raster(osm_labels, add = T)
mf_credits(get_credit("CartoDB.PositronOnlyLabels"), bg = "#ffffff80")
```



Pour certains styles, ceux fournis par Stadia ou Thunderforest par exemple, vous aurez besoin d'une clef d'API. Vous devez vous inscrire sur le site Web de ces fournisseurs pour obtenir une clef.

18 Import de données

18.1 osmdata

Le package `osmdata` (Mark Padgham et al., 2017) permet d'extraire des données vectorielles depuis OSM en utilisant l'API [Overpass turbo](#).

```
library(sf)
library(osmdata)
library(sf)
com <- st_read("data/lot.gpkg", layer = "communes", quiet = TRUE)

# Sélection de la commune de Cahors
cahors <- com[com$INSEE_COM == "46042", ]

# Définition d'une bounding box
q <- opq(bbox = st_bbox(st_transform(cahors, 4326)))

# Extraction des restaurants
req <- add_osm_feature(opq = q, key = 'amenity', value = "restaurant")
res <- osmdata_sf(req)

# Réduction du resultats :
# les points composant les polygones sont supprimés
res <- unique_osmdata(res)
```

Le résultat contient une couche de points et une couche de polygones. Cela signifie que certains restaurants (la très grande majorité) sont enregistrés sous forme de points dans OSM et d'autres sous forme de polygones. Pour obtenir une couche de points cohérente nous pouvons utiliser les centroïdes des polygones.

```
resto_point <- res$osm_points

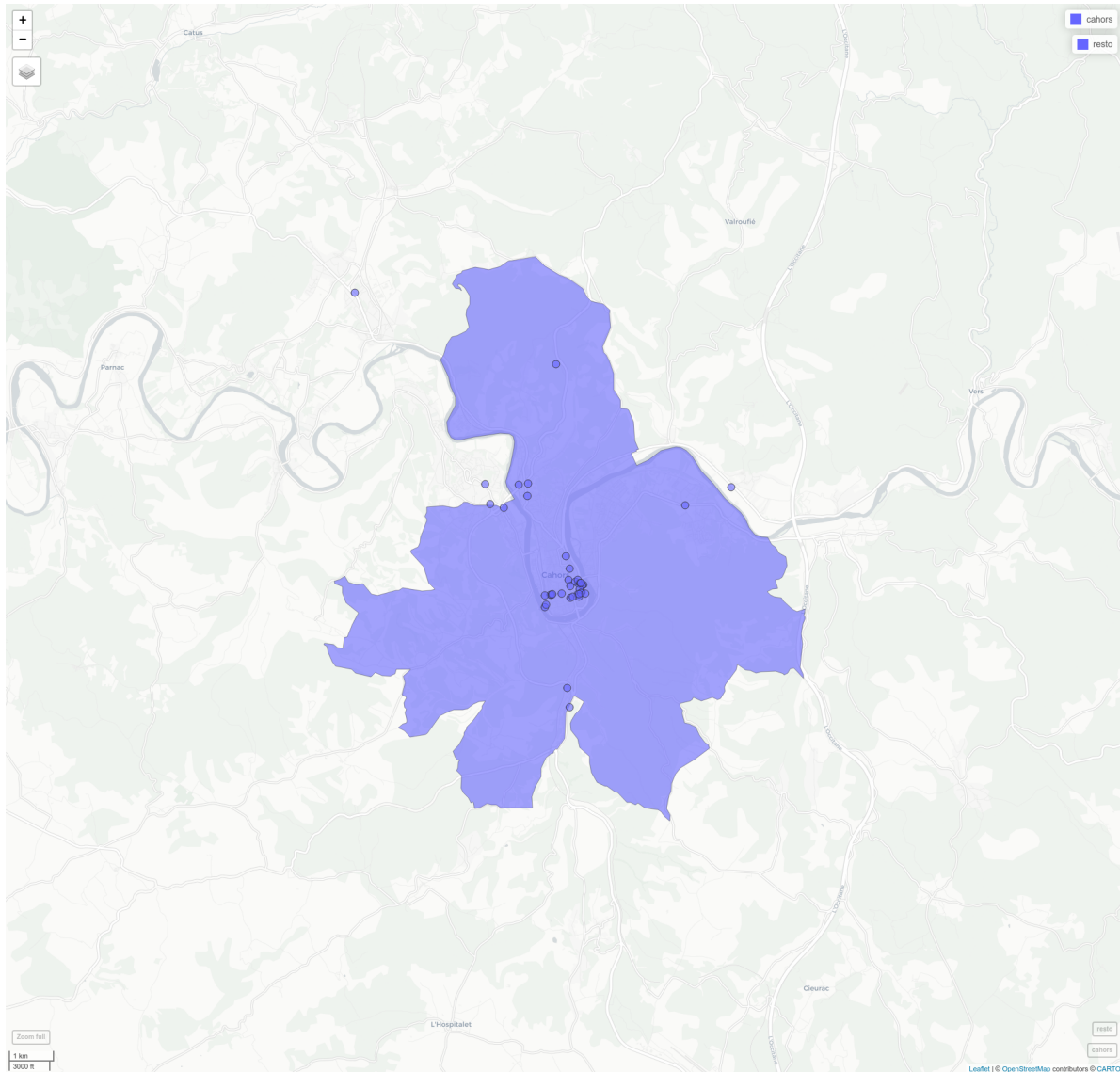
# extraire les centroïdes des polygones
resto_poly_point <- st_centroid(res$osm_polygons)
```

```
#> Warning: st_centroid assumes attributes are constant over geometries
```

```
# identifier les champs en commun  
chps <- intersect(names(resto_point), names(resto_poly_point))  
  
# Union des deux couches  
resto <- rbind(resto_point[, chps], resto_poly_point[, chps])
```

Affichage des résultats

```
library(mapview)  
mapview(cahors) + mapview(resto)
```



Site web d'osmdata

[osmdata](https://osmdata.org/)

18.2 osmextract

Le package `osmextract` (Gilardi et Lovelace, 2023) permet d'extraire des données depuis une base de données OSM directement. Ce package permet de travailler en local sur des volumes de données très importants et ainsi d'éviter de surcharger un serveur Overpass turbo.

La fonction `oe_get()` permet de télécharger un extrait de la base de données OSM sur une zone particulière et de sélectionner un type d'objet à importer.

L'argument `place` correspond au nom du fichier `*.pbf` accessible sur le site [Geofabrik](#). L'argument `extra_tag` permet de sélectionner les objets de la base OSM correspondant à une clef particulière (se référer à la [documentation d'OSM](#) pour choisir les clefs).

Nous nous intéressons ici à l'ensemble des équipements (clef `amenity`) enregistrés dans la base OSM sous forme de points en Andorre.

```
library(osmextract)
osm_pt <- oe_get(place = "Andorra",
                 layer = "points",
                 extra_tags = "amenity",
                 quiet = TRUE)
```

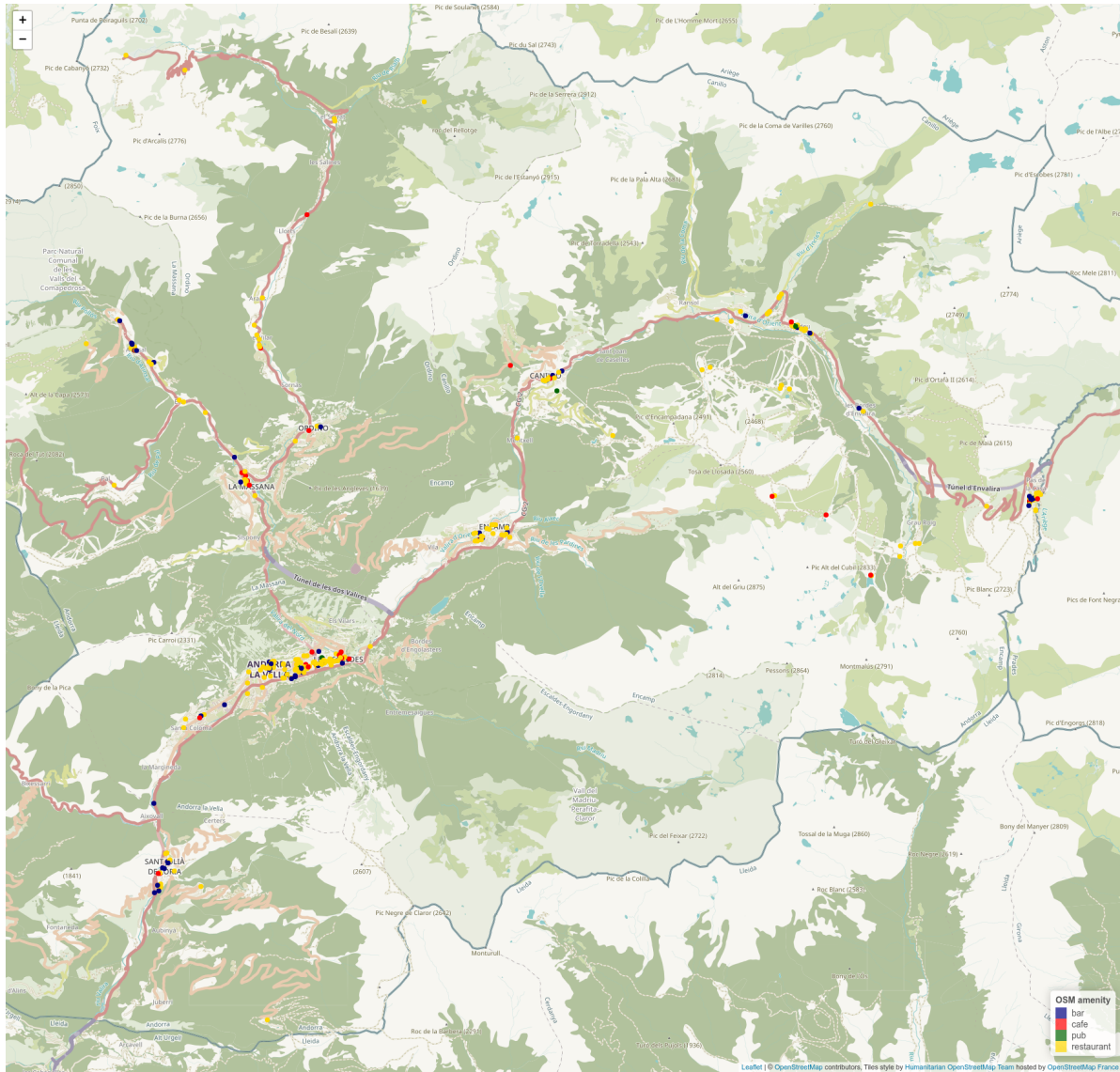
Des équipements de nature très variés sont saisis dans OSM:

```
unique(osm_pt$amenity)
```

```
#> [1] NA "parking_entrance" "fuel"
#> [4] "arts_centre" "restaurant" "parking"
#> [7] "theatre" "bar" "hospital"
#> [10] "atm" "school" "pharmacy"
#> [13] "ski_rental" "fast_food" "toilets"
#> [16] "bank" "veterinary" "post_box"
#> [19] "pub" "telephone" "cafe"
#> [22] "public_building" "shelter" "car_wash"
#> [25] "fountain" "post_office" "bench"
#> [28] "place_of_worship" "drinking_water" "recycling"
#> [31] "cinema" "library" "nightclub"
#> [34] "police" "kindergarten" "mini_storage"
#> [37] "bbq" "courthouse" "car_rental"
#> [40] "dojo" "waste_basket" "water_point"
#> [43] "dentist" "doctors" "townhall"
#> [46] "bus_station" "bicycle_parking" "sanitary_dump_station"
#> [49] "charging_station" "community_centre" "waste_disposal"
#> [52] "watering_place" "internet_cafe" "motorcycle_parking"
#> [55] "photo_booth" "locker" "shower"
#> [58] "events_venue" "public_bookcase" "animal_shelter"
#> [61] "biergarten" "childcare" "clinic"
#> [64] "lavoir" "vending_machine"
```

Nous allons maintenant sélectionner les bars, cafés, pubs et restaurants et les visualiser sur une carte interactive.

```
poi <- c("bar", "cafe", "pub", "restaurant")
osm_pt <- osm_pt[osm_pt$amenity %in% poi, ]
library(leaflet)
pal <- colorFactor(palette = c("navy", "red", "darkgreen", "gold"),
                  domain = poi)
leaflet(osm_pt) |>
  addProviderTiles("OpenStreetMap.HOT") |>
  addCircleMarkers(radius = 4,
                  stroke = FALSE,
                  color = ~ pal(amenity),
                  fillOpacity = 1,
                  popup = osm_pt$name) |>
  addLegend(pal = pal,
            values = poi,
            opacity = 0.7,
            title = "OSM amenity",
            position = "bottomright"
  )
```

Site web d'osmextract

[osmextract](https://osmextract.org/)

19 Matrices de temps et itinéraires

Le package `osrm` (Giraud, 2022) sert d'interface entre R et le service de calcul d'itinéraire [OSRM](#) (Luxen et Vetter, 2011). Ce package permet de calculer des matrices de temps et de distances, des itinéraires routiers, des isochrones. Le package utilise par défaut le serveur de démo d'OSRM. En cas d'utilisation intensive il est fortement recommandé d'[utiliser sa propre instance d'OSRM avec Docker](#).

19.1 Calcul d'un itinéraire

La fonction `osrmRoute()` permet de calculer des itinéraires.

```
library(sf)
library(osrm)
library(maptiles)

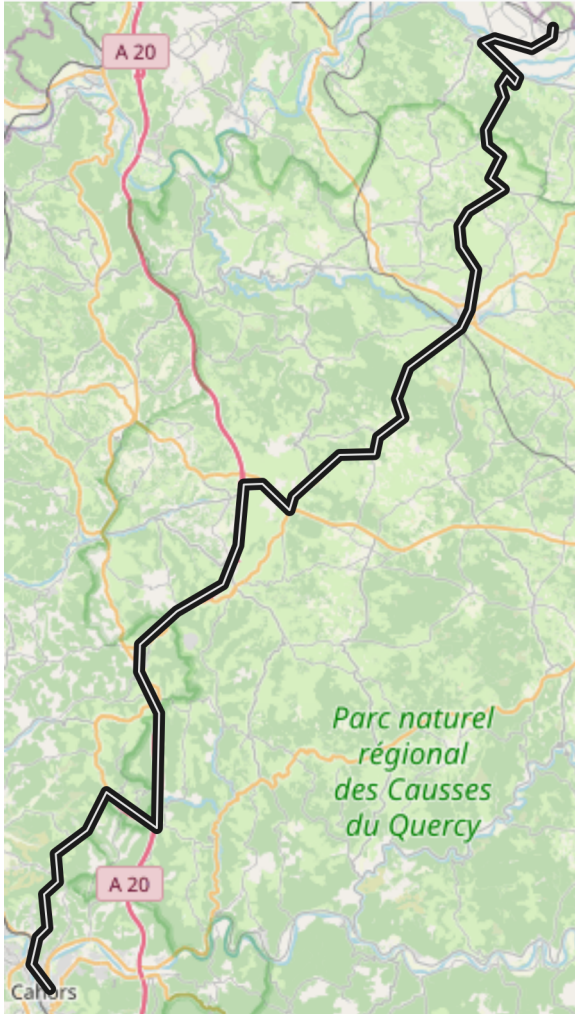
com_raw <- st_read("data/lot.gpkg", layer = "communes", quiet = TRUE)
com <- st_transform(com_raw, 3857)

# Itinéraire entre les centroïdes de Cahors et de Puybrun
cahors <- st_centroid(com[com$INSEE_COM == "46042", ])
puybrun <- st_centroid(com[com$INSEE_COM == "46229", ])

route <- osrmRoute(src = cahors,
                   dst = puybrun)

# Récupération d'un fond de carte OSM
osm <- get_tiles(st_buffer(route, 2000), crop = TRUE)

# Affichage
mf_theme(mar = c(0,0,0,0))
mf_raster(osm)
mf_map(route, col = "grey10", lwd = 6, add = T)
mf_map(route, col = "grey90", lwd = 1, add = T)
```



19.2 Calcul d'une matrice de temps

La fonction `osrmTable()` permet de calculer des matrices de distances ou de temps par la route.

Géoréférencement d'adresses ?

Voir le point [Géoréférencement](#) pour le géocodage d'adresse avec R.

Dans cet exemple nous calculons une matrice de temps entre 2 adresses et les restaurants de Cahors à pied.

```

library(sf)
library(tidygeocoder)

restaurant <- st_read("data/lot.gpkg", layer = "restaurants", quiet = TRUE)

# Sélection des restaurants de Cahors
restaurant_cahors <- st_filter(restaurant, com_raw[com_raw$INSEE_COM == "46042", ])

# Construction d'un data.frame contenant deux adresses
adresses <- data.frame(ad = c("3 rue Montaudié, Cahors, France",
                              "5 rue Albert Camus, Cahors, France"))

# Geocodage de 2 adresses à Cahors
places <- geocode(.tbl = adresses, address = ad, quiet = TRUE)
places <- as.data.frame(places)
row.names(places) <- c("Rue Montaudié", "Rue Albert Camus")

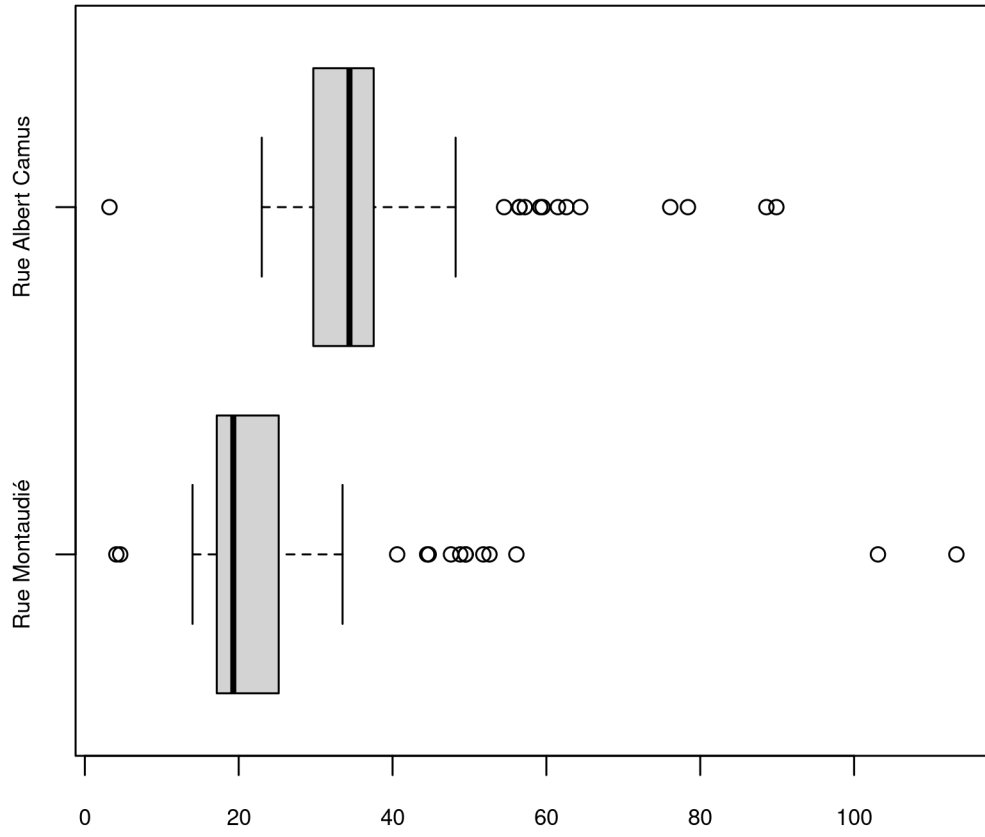
# Calcul de la matrice de distance entre les 2 adresses et les restaurants de Cahors
mat <- osrmTable(src = places[c(3, 2)],
                 dst = restaurant_cahors,
                 osrm.profile = "foot")

mat$durations[, 1:5]

#>
#> Rue Montaudié      17.8 21.3 22.1 22.6 14.4
#> Rue Albert Camus 33.0 64.4 37.0 24.6 34.2

# Quelle adresse possède une meilleure accessibilité aux restaurants ?
boxplot(t(mat$durations), cex.axis = .7, horizontal = TRUE)

```



partie IV

Acquisition de données

20 Géocodage

20.1 Géocodage d'adresse avec tidygeocoder

Plusieurs packages permettent de géocoder des adresses. Le package `tidygeocoder` (Cambon et al., 2021) permet d'utiliser un grand nombre de [services de géocodage en ligne](#).

```
library(tidygeocoder)
address_df <- data.frame(
  address = c("10 Emma Goldmanweg, 5032MN Tilburg, Netherlands",
             "19 rue Michel Bakounine, 29600 Morlaix, France")
)

places <- geocode(.tbl = address_df, address = "address", quiet = TRUE)
places
```

```
#> # A tibble: 2 x 3
#>   address                                lat long
#>   <chr>                                <dbl> <dbl>
#> 1 10 Emma Goldmanweg, 5032MN Tilburg, Netherlands  51.5  5.04
#> 2 19 rue Michel Bakounine, 29600 Morlaix, France   48.6 -3.82
```

20.2 Transformer des données long/lat en objet sf

La fonction `st_as_sf()` permet de créer un objet `sf` à partir d'un `data.frame` contenant des coordonnées géographiques.

Ici nous utilisons le `data.frame` `places` créé précédemment :

```
library(sf)
place_sf <- st_as_sf(places,
  coords = c("long", "lat"),
  crs = 'EPSG:4326')
place_sf
```

```

#> Simple feature collection with 2 features and 1 field
#> Geometry type: POINT
#> Dimension:      XY
#> Bounding box:  xmin: -3.816434 ymin: 48.59041 xmax: 5.038699 ymax: 51.53649
#> Geodetic CRS:  WGS 84
#> # A tibble: 2 x 2
#>   address                                geometry
#> * <chr>                                <POINT [°]>
#> 1 10 Emma Goldmanweg, 5032MN Tilburg, Netherlands (5.038699 51.53649)
#> 2 19 rue Michel Bakounine, 29600 Morlaix, France (-3.816435 48.59041)

```

Pour créer un objet `sf` de type `POINT` à partir d'une paire de coordonnées, ici le point de longitude 0.5 et de latitude 45.5 en WGS84 (EPSG:4326), il est nécessaire de créer le `data.frame` au préalable :

```

library(sf)
df_pt <- data.frame(x = 0.5, y = 45.5)
mon_point <- st_as_sf(df_pt, coords = c("x", "y"), crs = 'EPSG:4326')
mon_point

```

```

#> Simple feature collection with 1 feature and 0 fields
#> Geometry type: POINT
#> Dimension:      XY
#> Bounding box:  xmin: 0.5 ymin: 45.5 xmax: 0.5 ymax: 45.5
#> Geodetic CRS:  WGS 84
#>           geometry
#> 1 POINT (0.5 45.5)

```

20.3 Affichage sur un fond OpenStreetMap

Nous pouvons afficher cet objet `sf` sur un fond de carte [OpenStreetMap](#) avec le package `maptiles` (Giraud, 2023b).

```

library(mapsf)
library(maptiles)

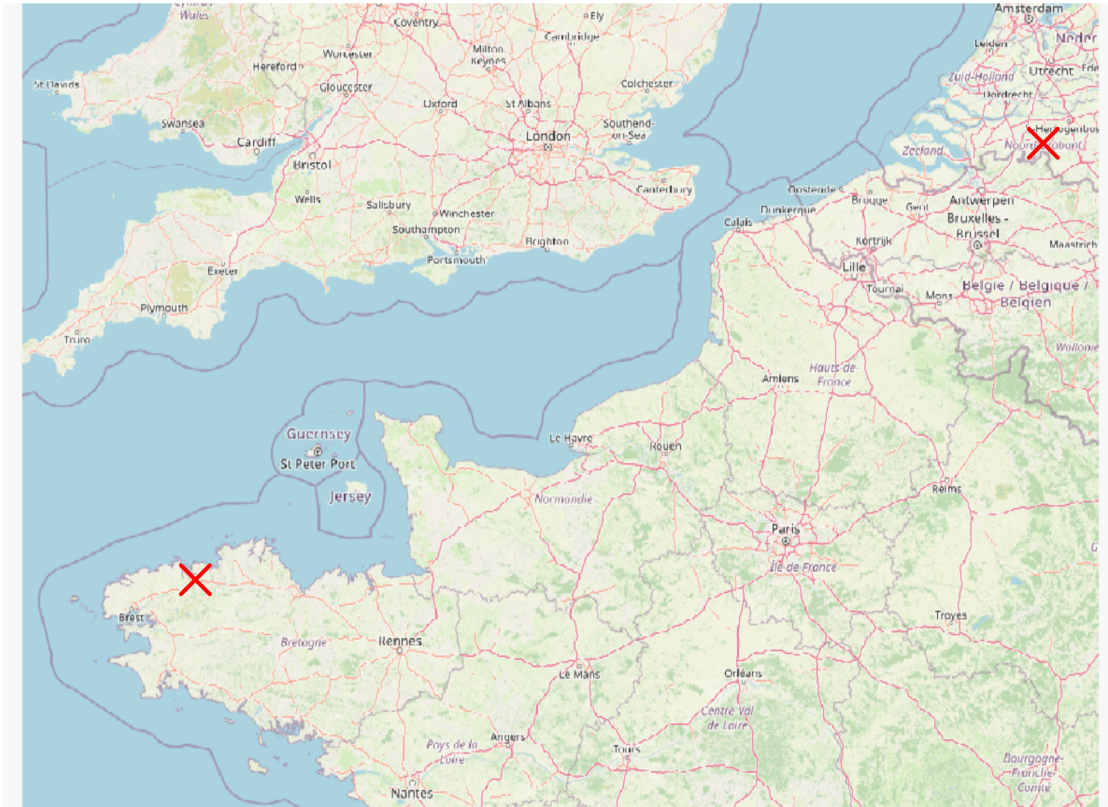
# Récupération d'un fond de carte OSM
osm <- get_tiles(x = place_sf, zoom = 7)

# Affichage

```



```
mf_raster(osm)
mf_map(place_sf, pch = 4, cex = 2,
       lwd = 2, col = "red", add = TRUE)
```



21 Digitalisation






Le package `mapedit` (Appelhans et al., 2020) permet de digitaliser des fonds de carte directement dans R. Mais disons le franchement, bien que pouvant se révéler pratique dans certains cas, ce package ne saurait se substituer aux fonctionnalités d'un SIG pour les tâches de digitalisation importantes.

22 Packages de données spatiales

De nombreux packages de mise à disposition de données géographiques (géométries et/ou attributs) ont été développés. Il s'agit le plus souvent de packages interfaçant des API qui permettent d'interroger des données mises à disposition sur le Web, directement avec R.

Ce chapitre en présente une liste non exhaustive.

22.1 À l'échelle mondiale

-  `rnaturalearth` (Massicotte et South, 2023) : permet de récupérer les données cartographiques [Natural Earth](#).
-  `cshapes` (Weidmann et al., 2021) : met à disposition les frontières nationales, de 1886 à aujourd'hui.
-  `osmextract` (Gilardi et Lovelace, 2023) : permet d'importer des données [OpenStreetMap](#).
-  `osmdata` (Mark Padgham et al., 2017) : pour télécharger et utiliser les données d'OpenStreetMap.
-  `maptiles` (Giraud, 2023b) : Ce package télécharge, compose et affiche des tuiles à partir d'un grand nombre de fournisseurs (*OpenStreetMap, Stadia, Esri, CARTO* ou *Thunderforest...*).
-  `geonames` (Rowlingson, 2019) : permet d'interroger la BD [geonames](#), qui fournit notamment des localisations.
-  `wbstats` (Piburn, 2020) et `WDI` (Arel-Bundock, 2022) : donnent accès aux données et statistiques de la Banque mondiale.
-  `sen2r` (Ranghetti et al., 2020) : permet de télécharger et prétraiter automatiquement les données du satellite Sentinel-2.
-  `MODISrsp` (Busetto et Ranghetti, 2016) : permet de trouver, télécharger et traiter des images *MODIS*.

- 🌐 `geodata` (Hijmans et al., 2023) : fournit un accès à des [données](#) sur le climat, l'altitude, le sol, la présence d'espèces et les limites administratives.
- 🌐 `elevatr` (Hollister et al., 2023) : donne accès à des données d'élévation mises à disposition par [Amazon Web Services Terrain Tiles](#), l'[Open Topography Global Datasets API](#) et l'[USGS Elevation Point Query Service](#).
- 🌐 `rgee` (Aybar, 2023) : permet d'utiliser l'API de [Google Earth Engine](#), catalogue de données publiques et infrastructure de calcul pour les images satellites.
- 🌐 `nasapower` (Sparks, 2018) : API client *NASA* (prévision des ressources énergétiques mondiales, météorologie, énergie solaire de surface et climatologie).
- 🌐 `geoknife` (Read et al., 2015) : permet de traiter (en ligne) des données matricielles volumineuses issues du *Geo Data Portal* de l'*U.S. Geological Survey*.
- 🌐 🏠 `rdhs` (Watson et al., 2019) : API client et gestions de données de l'[enquête démographique et de santé \(DHS\)](#).

22.2 À l'échelle européenne

- 🌐 `giscoR` (Hernangómez, 2023a) : permet de télécharger des données cartographiques mondiales et européennes de la BD [GISCO](#) d'Eurostat (système d'information géographique de la Commission).
- 🏠 `eurostat` (Lahti et al., 2017) : permet de télécharger des données de la BD [Eurostat](#).

22.3 À l'échelle nationale

- **Brésil**
 - 🌐 `geobr` (Pereira et Goncalves, 2023) : fournit un accès facile aux séries de données spatiales officielles du Brésil pour différentes années et découpages administratifs.
- **Chili**
 - 🌐 `chilemapas` (Vargas, 2022) : donne accès aux divisions politiques et administratives du Chili.
- **Espagne**
 - 🌐 `mapSpain` (Hernangómez, 2023b) : propose les limites administratives de l'Espagne à plusieurs niveaux (Communautés autonomes, Provinces, Municipalités), ainsi que des tuiles.
- **États-Unis**
 - 🌐 🏠 `tidycensus` (Walker et Herman, 2023) : permet de charger des données et géométries du recensement américain en format `sf` et `tidyverse`

- 🌐 **tigris** (Walker, 2023) : donne accès aux éléments cartographiques fournis par le US Census Bureau TIGER, y compris les limites cartographiques, les routes et l'eau.
- 🌐 🗄️ **FedData** (Bocinsky, 2023) : automatise le téléchargement de données géospatiales disponibles à partir de plusieurs sources de données fédérées.
- 🗄️ **acs** (Glenn, 2019) : permet de télécharger et manipuler les données de l'*American Community Survey* et les données décennales du recensement des États-Unis.
- 🗄️ **censusapi** (Recht, 2022) : wrapper pour les API du *Census Bureau* des États-Unis.
- 🗄️ **idbr** (Walker, 2021) : interface avec l'API de la base de données internationale du US Census Bureau.

- 🗄️ **ipumsr** (Greg Freedman Ellis et al., 2023) : Permet d'importer des données de recensement, d'enquête et géographiques fournies par l'[IPUMS](#).
- 🗄️ **totalcensus** (G. Li, 2021) : permet d'extraire les données du recensement décennal et de l'*American Community Survey* au niveaux des *block*, *block group* et *tract*.

- **Finland**
 - 🌐 **mapsFinland** (Haukka, 2023) : donne un accès à des cartes et données concernant la Finlande.

- **France**
 - 🌐 **happign** (Carteron, 2023) : accès à certaines données de l'IGN.
 - 🌐 **insee** (Leclerc, 2022) : pour télécharger facilement les données de la base BDM ([Banque de Données Macroéconomiques](#)) de l'INSEE.

- **Pologne**
 - 🌐 **rgugik** (Dyba et Nowosad, 2021) : permet l'acquisition automatique de données ouvertes à partir des ressources du Bureau central polonais de la géodésie et de la cartographie ([Główny Urząd Geodezji i Kartografii](#)).

- **Uruguay**
 - 🌐 **geouy** (Detomasi, 2023) : permet le chargement d'informations géographiques sur l'Uruguay.

- ...

References

- Agafonkin, V. (2015). JavaScript library for mobile-friendly interactive maps. <https://github.com/Leaflet/Leaflet>
- Appelhans, T., Detsch, F., Reudenbach, C. et Woellauer, S. (2023). *mapview: Interactive Viewing of Spatial Data in R*. <https://CRAN.R-project.org/package=mapview>
- Appelhans, T., Russell, K. et Busetto, L. (2020). *mapedit: Interactive Editing of Spatial Data in R*. <https://CRAN.R-project.org/package=mapedit>
- Arel-Bundock, V. (2022). *WDI: World Development Indicators and Other World Bank Data*. <https://CRAN.R-project.org/package=WDI>
- Aybar, C. (2023). *rgee: R Bindings for Calling the 'Earth Engine' API*. <https://CRAN.R-project.org/package=rgee>
- Bivand, R. (2021). Progress in the R ecosystem for representing and handling spatial data. *Journal of Geographical Systems*, 23(4), 515-546. <https://doi.org/10.1007/s10109-020-00336-0>
- Bivand, R., Keitt, T. et Rowlingson, B. (2023). *rgdal: Bindings for the 'Geospatial' Data Abstraction Library*. <https://CRAN.R-project.org/package=rgdal>
- Bivand, R. et Rundel, C. (2023). *rgeos: Interface to Geometry Engine - Open Source ('GEOS')*. <https://CRAN.R-project.org/package=rgeos>
- Bloch, M. (2013). Mapshaper: Tools for editing Shapefile, GeoJSON, TopoJSON and CSV files JavaScript library. <https://github.com/mbloch/mapshaper>
- Bocinsky, R. K. (2023). *FedData: Functions to Automate Downloading Geospatial Data Available from Several Federated Data Sources*. <https://CRAN.R-project.org/package=FedData>
- Busetto, L. et Ranghetti, L. (2016). MODISstp: an R package for preprocessing of MODIS Land Products time series. *Computers & Geosciences*, 97, 40-48. <https://doi.org/10.1016/j.cageo.2016.08.020>
- Cambon, J., Hernangómez, D., Belanger, C. et Possenriede, D. (2021). tidygeocoder: An R package for geocoding. *Journal of Open Source Software*, 6(65), 3544. <https://doi.org/10.21105/joss.03544>
- Carteron, P. (2023). *happign: R Interface to 'IGN' Web Services*. <https://CRAN.R-project.org/package=happign>
- Cheng, J., Schloerke, B., Karambelkar, B. et Xie, Y. (2023). *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*. <https://CRAN.R-project.org/package=leaflet>
- Detomasi, R. (2023). *geouy: Geographic Information of Uruguay*. <https://github.com/RichDeto/geouy>

- Dyba, K. et Nowosad, J. (2021). rgugik: Search and Retrieve Spatial Data from the Polish Head Office of Geodesy and Cartography in R. *Journal of Open Source Software*, 6(59), 2948. <https://doi.org/10.21105/joss.02948>
- GDAL/OGR contributors. (2022). *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation. <https://doi.org/10.5281/zenodo.5884351>
- GEOS contributors. (2021). *GEOS coordinate transformation software library*. Open Source Geospatial Foundation. <https://libgeos.org/>
- Gilardi, A. et Lovelace, R. (2023). *osmextract: Download and Import Open Street Map Data Extracts*. <https://CRAN.R-project.org/package=osmextract>
- Giraud, T. (2022). osrm: Interface Between R and the OpenStreetMap-Based Routing Service OSRM. *Journal of Open Source Software*, 7(78), 4574. <https://doi.org/10.21105/joss.04574>
- Giraud, T. (2023a). *mapsf: Thematic Cartography*. <https://CRAN.R-project.org/package=mapsf>
- Giraud, T. (2023b). *maptiles: Download and Display Map Tiles*. <https://CRAN.R-project.org/package=maptiles>
- Glenn, E. H. (2019). *acs: Download, Manipulate, and Present American Community Survey and Decennial Data from the US Census*. <https://CRAN.R-project.org/package=acs>
- Greg Freedman Ellis, Derek Burk et Finn Roberts. (2023). *ipumsr: An R Interface for Downloading, Reading, and Handling IPUMS Data*. <https://CRAN.R-project.org/package=ipumsr>
- Haukka, J. (2023). *mapsFinland: Maps of Finland*. <https://CRAN.R-project.org/package=mapsFinland>
- Hernangómez, D. (2023a). *giscoR: Download Map Data from GISCO API - Eurostat* (version 0.4.0). <https://doi.org/10.5281/zenodo.4317946>
- Hernangómez, D. (2023b). *mapSpain: Administrative Boundaries of Spain* (version 0.8.0). <https://doi.org/10.5281/zenodo.5366622>
- Hijmans, R. J. (2023a). *Spatial Data Science with R and "terra"*. <https://rspatial.org>
- Hijmans, R. J. (2023b). *terra: Spatial Data Analysis*. <https://CRAN.R-project.org/package=terra>
- Hijmans, R. J., Barbosa, M., Ghosh, A. et Mandel, A. (2023). *geodata: Download Geographic Data*. <https://CRAN.R-project.org/package=geodata>
- Hollister, J., Shah, T., Nowosad, J., Robitaille, A. L., Beck, M. W. et Johnson, M. (2023). *elevatr: Access Elevation Data from Various APIs*. <https://doi.org/10.5281/zenodo.8335450>
- Lahti, L., Huovari, J., Kainu, M. et Biecek, P. (2017). Retrieval and Analysis of Eurostat Open Data with the eurostat Package. *The R Journal*, 9(1), 385-392. <https://doi.org/10.32614/RJ-2017-019>
- Leclerc, H. (2022). *insee: Tools to Easily Download Data from INSEE BDM Database*. <https://CRAN.R-project.org/package=insee>
- Li, G. (2021). *totalcensus: Extract Decennial Census and American Community Survey Data*. <https://CRAN.R-project.org/package=totalcensus>
- Li, X. (2009). Map algebra and beyond : 1. Map algebra for scalar fields. <https://slideplayer.com/slide/5822638/>.
- Lovelace, R., Nowosad, J. et Muenchow, J. (2019). *Geocomputation with R*. CRC Press. <https://r.geocompx.org/>

- Luxen, D. et Vetter, C. (2011). *Real-time routing with OpenStreetMap data*. New York, NY, USA (p. 513-516). <https://doi.org/10.1145/2093973.2094062>
- Madelin, M. (2021). Analyse d'images raster (et télédétection). https://mmadelin.github.io/sigr2021/SIGR2021_raster_MM.html.
- Mark Padgham, Bob Rudis, Robin Lovelace et Maëlle Salmon. (2017). osmdata. *Journal of Open Source Software*, 2(14), 305. <https://doi.org/10.21105/joss.00305>
- Massicotte, P. et South, A. (2023). *rnaturalearth: World Map Data from Natural Earth*. <https://CRAN.R-project.org/package=rnaturalearth>
- Mennis, J. (2015). Fundamentals of GIS : raster operations. <https://cupdf.com/document/gus-0262-fundamentals-of-gis-lecture-presentation-7-raster-operations-jeremy.html>.
- Nowosad, J. (2021). Image processing and all things raster. <https://nowosad.github.io/SIGR2021/workshop2/workshop2.html>.
- Pebesma, E. (2018). Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, 10(1), 439-446. <https://doi.org/10.32614/RJ-2018-009>
- Pebesma, E. et Bivand, R. (2005). Classes and methods for spatial data in R. *R News*, 5(2), 9-13. <https://CRAN.R-project.org/doc/Rnews/>
- Pebesma, E. et Bivand, R. (2023). *Spatial Data Science: With applications in R* (p. 352). Chapman and Hall/CRC. <https://r-spatial.org/book/>
- Pebesma, E., Mailund, T. et Hiebert, J. (2016). Measurement Units in R. *R Journal*, 8(2), 486-494. <https://doi.org/10.32614/RJ-2016-061>
- Pereira, R. H. M. et Goncalves, C. N. (2023). *geobr: Download Official Spatial Data Sets of Brazil*. <https://CRAN.R-project.org/package=geobr>
- Piburn, J. (2020). *wbstats: Programmatic Access to the World Bank API*. Oak Ridge National Laboratory. <https://doi.org/10.11578/dc.20171025.1827>
- PROJ contributors. (2021). *PROJ coordinate transformation software library*. Open Source Geospatial Foundation. <https://proj.org/>
- Racine, E. B. (2016). The Visual Raster Cheat Sheet. <https://rpubs.com/etiennebr/visualraster>.
- Ranghetti, L., Boschetti, M., Nutini, F. et Busetto, L. (2020). sen2r: An R toolbox for automatically downloading and preprocessing Sentinel-2 satellite data. *Computers & Geosciences*, 139, 104473. <https://doi.org/10.1016/j.cageo.2020.104473>
- Read, J. S., Walker, J. I., Appling, A., Blodgett, D. L., Read, E. K. et Winslow, L. A. (2015). geoknife: Reproducible web-processing of large gridded datasets. *Ecography*. <https://doi.org/10.1111/ecog.01880>
- Recht, H. (2022). *censusapi: Retrieve Data from the Census APIs*. <https://CRAN.R-project.org/package=censusapi>
- Rowlingson, B. (2019). *geonames: Interface to the "Geonames" Spatial Query Web Service*. <https://CRAN.R-project.org/package=geonames>
- Sparks, A. H. (2018). nasapower: A NASA POWER Global Meteorology, Surface Solar Energy and Climatology Data Client for R. *The Journal of Open Source Software*, 3(30), 1035. <https://doi.org/10.21105/joss.01035>
- Teucher, A. et Russell, K. (2023). *rmapshaper: Client for 'mapshaper' for 'Geospatial' Operations*. <https://CRAN.R-project.org/package=rmapshaper>

- Tomlin, C. D. (1990). *Geographic information systems and cartographic modeling*. Prentice Hall.
- Vargas, M. (2022). *chilemapas: Mapas de las Divisiones Politicas y Administrativas de Chile (Maps of the Political and Administrative Divisions of Chile)*. <https://CRAN.R-project.org/package=chilemapas>
- Walker, K. (2021). *idbr: R Interface to the US Census Bureau International Data Base API*. <https://CRAN.R-project.org/package=idbr>
- Walker, K. (2022). *crsuggest: Obtain Suggested Coordinate Reference System Information for Spatial Data*. <https://CRAN.R-project.org/package=crsuggest>
- Walker, K. (2023). *tigris: Load Census TIGER/Line Shapefiles*. <https://CRAN.R-project.org/package=tigris>
- Walker, K. et Herman, M. (2023). *tidycensus: Load US Census Boundary and Attribute Data as 'tidyverse' and 'sf'-Ready Data Frames*. <https://CRAN.R-project.org/package=tidycensus>
- Watson, O. J., FitzJohn, R. et Eaton, J. W. (2019). rdhs: an R package to interact with The Demographic and Health Surveys (DHS) Program datasets. *Wellcome Open Research*, 4, 103. <https://doi.org/10.12688/wellcomeopenres.15311.1>
- Weidmann, N. B., Schvitz, G. et Girardin, L. (2021). *cshapes: The CShapes 2.0 Dataset and Utilities*. <https://CRAN.R-project.org/package=cshapes>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., Takahashi, K., et al. Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. <https://doi.org/10.21105/joss.01686>

A Les données du projet

Les données utilisées dans ce document sont stockées dans un projet RStudio. Vous devez le télécharger puis le décompresser sur votre machine. Il vous sera ensuite possible de tester l'ensemble des manipulations proposées dans ce document au sein du projet **geodata**.

[Télécharger le projet](#)

lot.gpkg

Ce fichier contient plusieurs couches d'informations.

- **departements** : les départements français métropolitains, [Admin Express COG Carto 3.0, IGN - 2021](#);
- **communes** : les communes du département du Lot (46) avec des données sur la population active occupée âgée de 25 à 54 ans, par secteur d'activité et sexe, au lieu de résidence, en 2017, [BD CARTO® 4.0, IGN - 2021 & Recensements harmonisés - Séries départementales et communales, INSEE - 2020](#);
- **routes** : les routes de la commune de Gramat et alentours (46128), [BD CARTO® 4.0, IGN - 2021](#);
- **restaurants** : les restaurants du Lot, [Base permanente des équipements \(BPE\), INSEE - 2021](#);
- **elevations** : une grille régulière de points d'altitude (pas d'1 km), [Jarvis A., H.I. Reuter, A. Nelson, E. Guevara, 2008, Hole-filled seamless SRTM data V4, International Centre for Tropical Agriculture \(CIAT\)](#).

com.csv

Ce fichier tabulaire contient des informations complémentaires sur la population active occupée âgée de 25 à 54 ans, par secteur d'activité et sexe, au lieu de résidence, en 2017, [Recensements harmonisés - Séries départementales et communales, INSEE - 2020](#).

- le nombre d'actifs (ACT);
- le nombre d'actifs dans l'industrie (IND);
- la part des actifs dans la population totale (SACT);
- la part des actifs dans l'industrie dans le total des actifs (SACT_IND).

elevation.tif

Une grille régulière de points d'altitude (pas de 30 mètres environ), [Jarvis A., H.I. Reuter, A. Nelson, E. Guevara, 2008, Hole-filled seamless SRTM data V4, International Centre for](#)

Tropical Agriculture (CIAT).

elev.tif est une version reprojétée en Lambert 93 de **elevation.tif**

✂ **clc_2018.tif**

Données CORINE Land Cover, [Corine Land Cover \(CLC\) 2018, Version 2020_20u1 - Copernicus Programme](#).

clc.tif est une version reprojétée en Lambert 93 de **clc_2018.tif**

✂ **Sentinel2A.tif**

Données Sentinel, [Sentinel, Sentinel-2A, S2A_OPER_MSI_L2A_DS_VGS2_20211012T140548_S20211012T140548_12 Octobre 2021 - Copernicus Programme](#), téléchargées le 28 décembre 2021.