

# Cartographie avec R

*Timothée Giraud & Hugues Pécout*

*2019-01-15*



# Contents



# Introduction

Ce document se compose de trois parties permettant d'appréhender la création de cartes thématiques avec R.

- Les données spatiales
- Cartographie thématique
- Cartographie thématique avancée

Voici une partie des packages dédiés à l'import, la manipulation, la transformation et l'affichage de données spatiales que nous utiliserons : `sf`, `cartography`, `mapview`, `raster`, `SpatialPosition`, `spatstat`. D'autres pourront être nécessaires ponctuellement (`mapinsetr`, `osmdata`, `maptools`, `linemap`, `raster`, `rayshader`, `dplyr`, `photon`, `nominatim`, `banR`)

## Objectifs

- Savoir créer et manipuler des données spatiales
- Savoir créer des cartes thématiques conformes aux règles de la sémiologie graphique et de la cartographie
- Connaitre des modes de représentation plus complexes



La version en ligne de ce document est sous licence Creative Commons Attribution-NonCommercial-ShareAlike 4.0.



# Chapter 1

## Les données spatiales

Il est possible d'importer, de manipuler, de traiter, d'afficher et d'exporter des données spatiales avec R. La grande majorité des opérations de géotraitement sont disponibles dans R grâce au package **sf**. Il devient alors possible d'utiliser R comme un SIG.

### 1.1 Le package **sf**

#### Historique

Historiquement, trois packages permettent d'importer, de manipuler et de transformer les données spatiales :

- Le package **rgdal** qui est une interface entre R et les librairies GDAL (Geospatial Data Abstraction Library) et PROJ4 permet d'importer et d'exporter les données spatiales (les shapefiles par exemple) et aussi de gérer les projections cartographiques
- Le package **sp** fournit des classes et méthodes pour les données spatiales dans R. Il permet d'afficher des fonds de cartes, d'inspecter une table attributaire etc.
- Le package **rgeos** donne accès à la librairie d'opérations spatiales GEOS (Geometry Engine - Open Source) et rend donc disponible les opérations SIG classiques : calcul de surface ou de périmètre, calcul de distances, agrégations spatiales, zones tampons, intersections etc.

#### La suite

Le package **sf** ((?), (?)) a été publié fin 2016 par Edzer Pebesma (auteur de **sp**). Son objectif est de combiner les fonctionnalités de **sp**, **rgeos** et **rgdal** dans un package unique plus ergonomique. Ce package propose des objets plus simples (suivant le standard simple feature) dont la manipulation est plus aisée. Une attention particulière a été portée à la compatibilité du package avec la syntaxe *pipe* et les opérateurs du *tidyverse*.

Aujourd'hui, les principaux développements dans l'écosystème spatial de R se détachent progressivement des 3 anciens (**sp**, **rgdal**, **rgeos**) pour se reposer sur **sf**.

#### 1.1.1 Format des objets spatiaux **sf**

```

## Simple feature collection with 100 features
## geometry type: MULTIPOLYGON
## dimension: XY
## bbox: xmin: -84.32385 ymin: 33.25
## epsg (SRID): 4267
## proj4string: +proj=longlat +datum=NAD83
## precision: double (default; no precision)
## First 3 features:
##   BIR74 SID74 NWBIR74 BIR79 SID79 NWBIR79
## 1 1091 1 10 1364 0 19
## 2 487 0 10 542 3 11
## 3 3188 5 208 3616 6 26

```

Simple feature | Simple feature

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.

### 1.1.2 Construction d'un objet sf

#### Couche de points

```

library(sf)
pt1_sfg <- st_point(c(1,2))
pt2_sfg <- st_point(c(3,4))
pt3_sfg <- st_point(c(2,1))
(pt_sfc <- st_sf(pt1_sfg, pt2_sfg, pt3_sfg, crs = (4326)))

```

```

Geometry set for 3 features
geometry type: POINT
dimension: XY

```

```

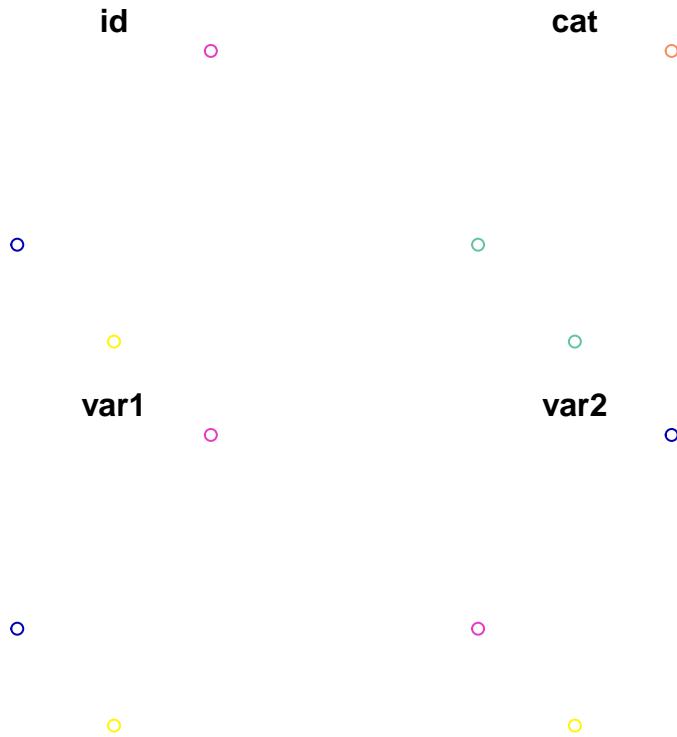
bbox:           xmin: 1 ymin: 1 xmax: 3 ymax: 4
epsg (SRID):   4326
proj4string:   +proj=longlat +datum=WGS84 +no_defs
pt_df <- data.frame(id= c(1,2,3), cat = c("A", "B", "A"),
                     var1 = c(10,20,30), var2 = c(2.3,1.9,4))
(pt_sf <- st_sf(pt_df,geometry = pt_sf))

```

Simple feature collection with 3 features and 4 fields

geometry type: POINT  
dimension: XY  
bbox: xmin: 1 ymin: 1 xmax: 3 ymax: 4  
epsg (SRID): 4326  
proj4string: +proj=longlat +datum=WGS84 +no\_defs  
id cat var1 var2 geometry  
1 1 A 10 2.3 POINT (1 2)  
2 2 B 20 1.9 POINT (3 4)  
3 3 A 30 4.0 POINT (2 1)

```
plot(pt_sf)
```



### Couche de polygones

```

p1 <- rbind(c(0,0), c(1,0), c(3,2), c(2,4), c(1,4), c(0,0))
p2 <- rbind(c(3,0), c(4,0), c(4,1), c(3,1), c(3,0))
p3 <- rbind(c(3,3), c(4,2), c(4,3), c(3,3))
pol1_sfg <-st_polygon(list(p1))
pol2_sfg <-st_polygon(list(p2))
pol3_sfg <-st_polygon(list(p3))
(pol_sfc <- st_sf(pol1_sfg, pol2_sfg, pol3_sfg, crs = 4326))

```

Geometry set for 3 features  
geometry type: POLYGON

```

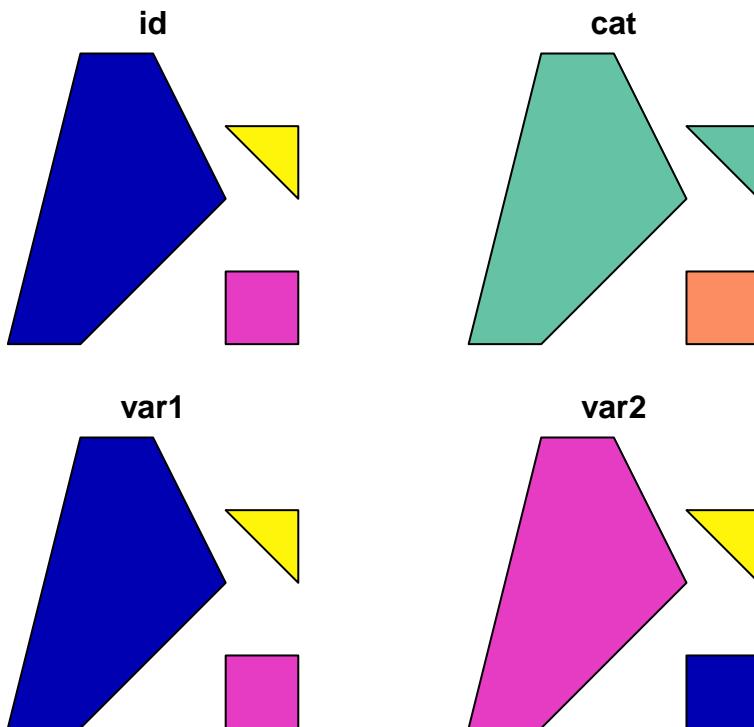
dimension:      XY
bbox:           xmin: 0 ymin: 0 xmax: 4 ymax: 4
epsg (SRID):   4326
proj4string:   +proj=longlat +datum=WGS84 +no_defs
pol_df <- data.frame(id= c(1,2,3), cat = c("A", "B", "A"),
                      var1 = c(10,20,30), var2 = c(2.3,1.9,4))
(pol_sf <- st_sf(pol_df,geometry = pol_sfc))

```

```

Simple feature collection with 3 features and 4 fields
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: 0 ymin: 0 xmax: 4 ymax: 4
epsg (SRID):   4326
proj4string:   +proj=longlat +datum=WGS84 +no_defs
  id cat var1 var2             geometry
1  1   A   10   2.3 POLYGON ((0 0, 1 0, 3 2, 2 ...
2  2   B   20   1.9 POLYGON ((3 0, 4 0, 4 1, 3 ...
3  3   A   30   4.0 POLYGON ((3 3, 4 2, 4 3, 3 3))
plot(pol_sf)

```



### Couche de linestring

```

p1 <- rbind(c(0,0), c(1,0), c(3,2), c(2,4), c(1,4))
p2 <- rbind(c(3,0), c(4,0), c(4,1), c(3,1))
p3 <- rbind(c(3,3), c(4,2), c(4,3))
ls1_sfg <-st_linestring(p1)
ls2_sfg <-st_linestring(p2)
ls3_sfg <-st_linestring(p3)
(ls_sfc <- st_sf(ls1_sfg, ls2_sfg, ls3_sfg, crs = 4326))

```

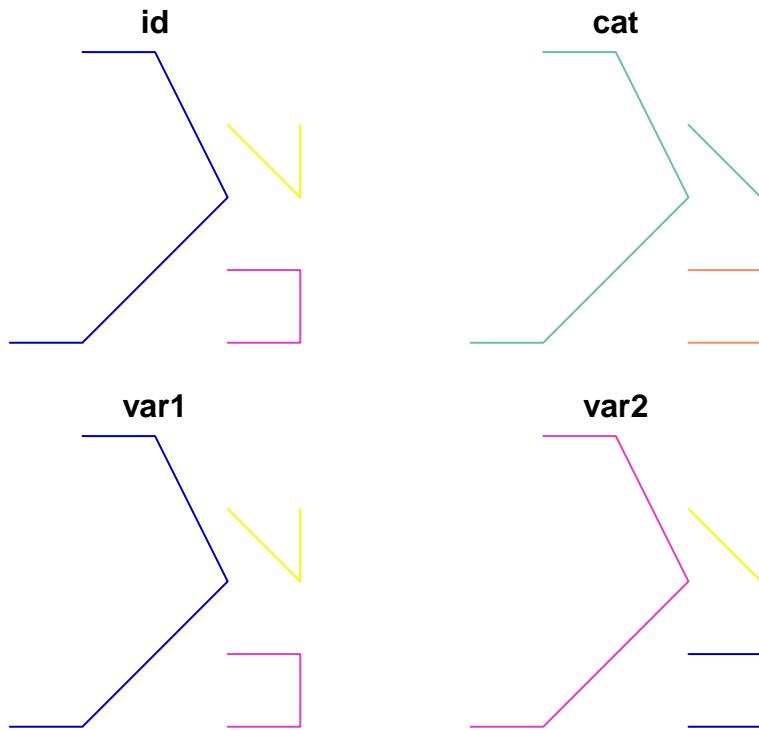
Geometry set for 3 features

```
geometry type:  LINESTRING
dimension:      XY
bbox:           xmin: 0 ymin: 0 xmax: 4 ymax: 4
epsg (SRID):   4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
ls_df <- data.frame(id= c(1,2,3), cat = c("A", "B", "A"),
                     var1 = c(10,20,30), var2 = c(2.3,1.9,4))
(ls_sf <- st_sf(ls_df,geometry = ls_sfc))
```

Simple feature collection with 3 features and 4 fields

```
geometry type:  LINESTRING
dimension:      XY
bbox:           xmin: 0 ymin: 0 xmax: 4 ymax: 4
epsg (SRID):   4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
  id cat var1 var2               geometry
1  1   A   10   2.3  LINESTRING (0 0, 1 0, 3 2, ...
2  2   B   20   1.9  LINESTRING (3 0, 4 0, 4 1, ...
3  3   A   30   4.0  LINESTRING (3 3, 4 2, 4 3)
```

```
plot(ls_sf)
```



### 1.1.3 Import / Export

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

```
library(sf)
mtq <- st_read("data/martinique.shp", quiet=TRUE)
```

```
st_write(obj = mtq, dsn = "data/mtq.gpkg", layer = "mtq", delete_layer = TRUE)
```

```
Deleting layer `mtq' using driver `GPKG'
Updating layer `mtq' to data source `data/mtq.gpkg' using driver `GPKG'
features:      34
fields:        23
geometry type: Polygon
st_write(obj = mtq, "data/mtq.shp", delete_layer = TRUE)
```

```
Deleting layer `mtq' using driver `ESRI Shapefile'
Writing layer `mtq' to data source `data/mtq.shp' using driver `ESRI Shapefile'
features:      34
fields:        23
geometry type: Polygon
```

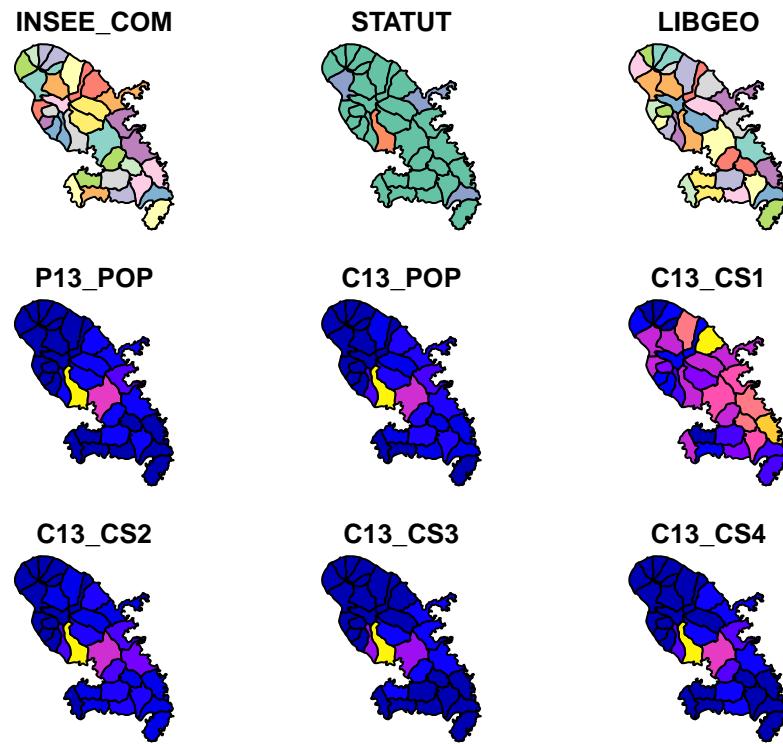
#### 1.1.4 Affichage de données

Aperçu des variables via les fonctions `head()` et `plot()`.

```
head(mtz)
```

```
Simple feature collection with 6 features and 23 fields
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: 695444.4 ymin: 1598817 xmax: 717731.2 ymax: 1645182
epsg (SRID):   32620
proj4string:    +proj=utm +zone=20 +datum=WGS84 +units=m +no_defs
  INSEE_COM      STATUT          LIBGEO P13_POP  C13_POP  C13_CS1  C13_CS2
1    97201 Commune simple L'Ajoupa-Bouillon    1830 1481.801  9.780866 48.90433
2    97202 Commune simple Les Anses-d'Arlet    3929 3190.115 97.433459 170.50855
3    97203 Commune simple     Basse-Pointe    3565 2983.215 39.510829 98.77707
  C13_CS3  C13_CS4  C13_CS5  C13_CS6  C13_CS7  C13_CS8 P08_POP  C08_POP  C08_CS1
1  9.780866 102.6991 273.8642 288.5355 430.3581 317.8781    1691 1346.519 31.40569
2 109.612642 239.5239 560.2424 385.6741 746.5743 880.5453    3826 3067.742 49.00453
3 43.461911 181.7498 568.9559 565.0048 940.5055 545.2494    3804 3054.108 44.51803
  C08_CS2  C08_CS3  C08_CS4  C08_CS5  C08_CS6  C08_CS7  C08_CS8
1 43.18282 11.77713 145.2513 223.7655 251.2455 380.7940 259.0969
2 143.95079 65.33937 216.3534 600.3054 459.4174 558.8020 974.5694
3 106.84327 27.70011 186.0292 448.1481 620.2845 881.5853 738.9993
  geometry
1 POLYGON ((699261.2 1637681, ...
2 POLYGON ((709840 1599026, 7...
3 POLYGON ((706092.8 1642964, ...
[ reached 'max' / getOption("max.print") -- omitted 3 rows ]
```

```
plot(mtz)
```



Affichage de la géométrie uniquement.

```
plot(st_geometry(mtq))
```



### 1.1.5 Joindre des données

On peut joindre un `data.frame` à un objet sf en utilisant la fonction `merge()`.

```
mtq2016 <- read.csv(file = "data/mtq2016.csv")
head(mtq2016)

  ID           NOM Population.totale
1 97201 L' Ajoupa-Bouillon          1964
2 97202 Les Anses-d'Arlet          3686
3 97203     Basse-Pointe          3238
4 97234    Bellefontaine          1760
5 97204      Le Carbet          3655
6 97205   Case-Pilote          4522

mtq <- merge(x = mtq, y = mtq2016, by.x = "INSEE_COM", by.y = "ID")
head(mtq)

Simple feature collection with 6 features and 25 fields
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: 695444.4 ymin: 1598817 xmax: 717731.2 ymax: 1645182
epsg (SRID):   32620
proj4string:   +proj=utm +zone=20 +datum=WGS84 +units=m +no_defs
  INSEE_COM      STATUT           LIBGEO P13_POP C13_POP C13_CS1  C13_CS2
1 97201 Commune simple L'Ajoupa-Bouillon    1830 1481.801 9.780866 48.90433
2 97202 Commune simple Les Anses-d'Arlet    3929 3190.115 97.433459 170.50855
3 97203 Commune simple     Basse-Pointe    3565 2983.215 39.510829 98.77707
  C13_CS3  C13_CS4  C13_CS5  C13_CS6  C13_CS7  C13_CS8 P08_POP C08_POP C08_CS1
1 9.780866 102.6991 273.8642 288.5355 430.3581 317.8781 1691 1346.519 31.40569
2 109.612642 239.5239 560.2424 385.6741 746.5743 880.5453 3826 3067.742 49.00453
3 43.461911 181.7498 568.9559 565.0048 940.5055 545.2494 3804 3054.108 44.51803
  C08_CS2  C08_CS3  C08_CS4  C08_CS5  C08_CS6  C08_CS7  C08_CS8           NOM
1 43.18282 11.77713 145.2513 223.7655 251.2455 380.7940 259.0969 L' Ajoupa-Bouillon
2 143.95079 65.33937 216.3534 600.3054 459.4174 558.8020 974.5694 Les Anses-d'Arlet
3 106.84327 27.70011 186.0292 448.1481 620.2845 881.5853 738.9993     Basse-Pointe
Population.totale                      geometry
1          1964 POLYGON ((699261.2 1637681, ...
2          3686 POLYGON ((709840 1599026, 7...
3          3238 POLYGON ((706092.8 1642964, ...
[ reached 'max' / getOption("max.print") -- omitted 3 rows ]
```

## 1.2 Les systèmes de projections

### 1.2.1 Consulter la projection d'un objet

La fonction `st_crs()` permet de consulter le système de projection utilisé par un objet sf et de la modifier (**sans reprojeter les données**).

```
st_crs(mtq)

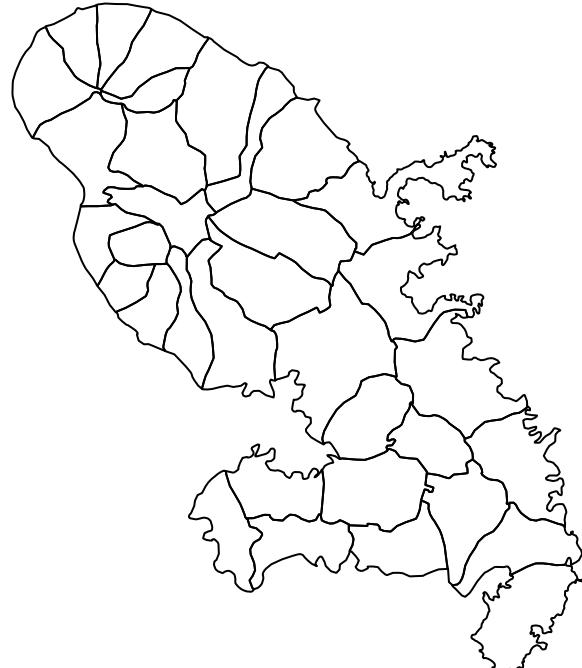
Coordinate Reference System:
EPSG: 32620
proj4string: "+proj=utm +zone=20 +datum=WGS84 +units=m +no_defs"
```

### 1.2.2 Modifier la projection d'un objet

La fonction `st_transform()` permet de reprojeter un objet sf.

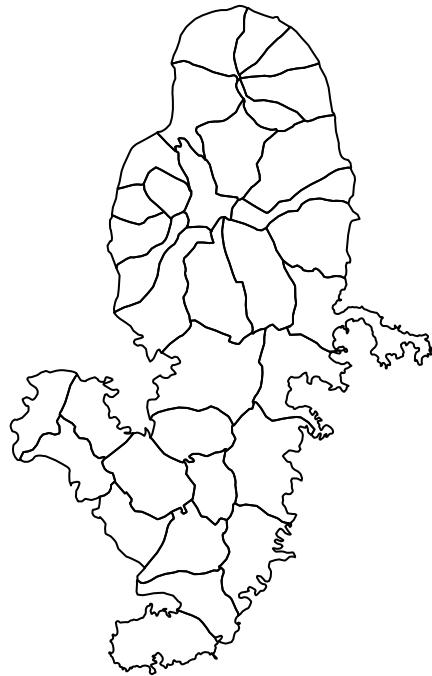
```
plot(st_geometry(mtq))
title("WGS 84 / UTM zone 20N")
```

**WGS 84 / UTM zone 20N**



```
mtq_reproj <- st_transform(mtq, 2154)
plot(st_geometry(mtq_reproj))
title("RGF93 / Lambert-93")
```

## RGF93 / Lambert-93



Le site Spatial Reference met à disposition les références de très nombreux systèmes de projection.

### 1.3 Opérations de géotraitements

#### 1.3.1 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`.

```
# selection de ligne
mtq[1:2, ]
```

```
Simple feature collection with 2 features and 25 fields
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: 697601.7 ymin: 1598817 xmax: 710461.9 ymax: 1640521
epsg (SRID):   32620
proj4string:   +proj=utm +zone=20 +datum=WGS84 +units=m +no_defs
  INSEE_COM      STATUT          LIBGEO P13_POP C13_POP C13_CS1  C13_CS2
1    97201 Commune simple L'Ajoupa-Bouillon    1830 1481.801  9.780866 48.90433
2    97202 Commune simple Les Anses-d'Arlet    3929 3190.115 97.433459 170.50855
      C13_CS3  C13_CS4  C13_CS5  C13_CS6  C13_CS7  C13_CS8 P08_POP C08_POP C08_CS1
1    9.780866 102.6991 273.8642 288.5355 430.3581 317.8781    1691 1346.519 31.40569
2 109.612642 239.5239 560.2424 385.6741 746.5743 880.5453    3826 3067.742 49.00453
      C08_CS2  C08_CS3  C08_CS4  C08_CS5  C08_CS6  C08_CS7  C08_CS8          NOM
1  43.18282 11.77713 145.2513 223.7655 251.2455 380.794 259.0969 L' Ajoupa-Bouillon
2 143.95079 65.33937 216.3534 600.3054 459.4174 558.802 974.5694  Les Anses-d'Arlet
  Population.totale          geometry
1            1964 POLYGON ((699261.2 1637681, ...
```

```
2           3686 POLYGON ((709840 1599026, 7...
mtq[mtq$LIBGEO=="Fort-de-France", ]
```

```
Simple feature collection with 1 feature and 25 fields
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: 704448.6 ymin: 1614283 xmax: 711650.9 ymax: 1626937
epsg (SRID):   32620
proj4string:    +proj=utm +zone=20 +datum=WGS84 +units=m +no_defs
                INSEE_COM          STATUT          LIBGEO P13_POP C13_POP C13_CS1 C13_CS2
9      97209 Préfecture de région Fort-de-France 84174 68712.33 86.57284 2720.489
                C13_CS3  C13_CS4  C13_CS5  C13_CS6  C13_CS7  C13_CS8 P08_POP C08_POP C08_CS1
9 4000.387 8407.424 13799.23 7309.136 16184.26 16204.84 89000 71566.82 119.0608
                C08_CS2  C08_CS3  C08_CS4  C08_CS5  C08_CS6  C08_CS7  C08_CS8           NOM
9 2480.105 3976.898 8630.605 15437.6 7964.513 15996.58 16961.46 Fort-de-France
                Population.totale      geometry
9           82030 POLYGON ((711183.1 1619627,...
```

# selection de colonnes

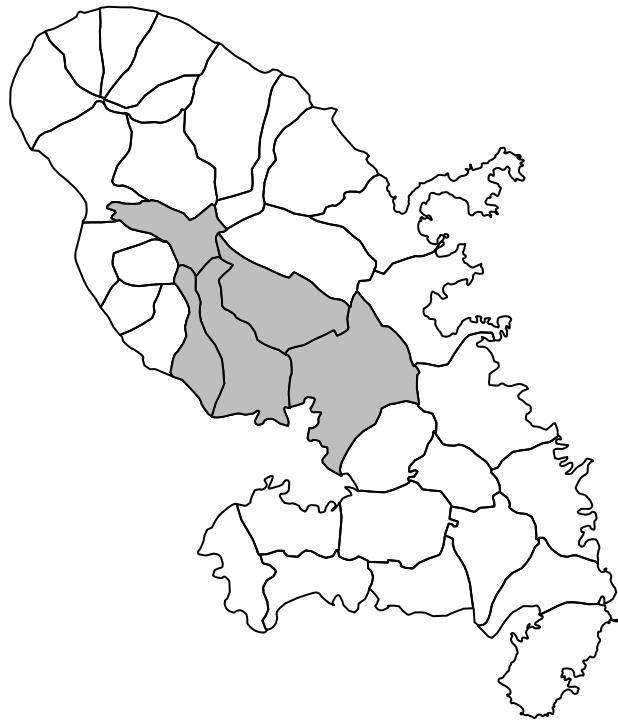
```
mtq[mtq$LIBGEO=="Fort-de-France", 1:4]
```

```
Simple feature collection with 1 feature and 4 fields
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: 704448.6 ymin: 1614283 xmax: 711650.9 ymax: 1626937
epsg (SRID):   32620
proj4string:    +proj=utm +zone=20 +datum=WGS84 +units=m +no_defs
                INSEE_COM          STATUT          LIBGEO P13_POP           geometry
9      97209 Préfecture de région Fort-de-France 84174 POLYGON ((711183.1 1619627,...
```

### 1.3.2 Sélection spatiale

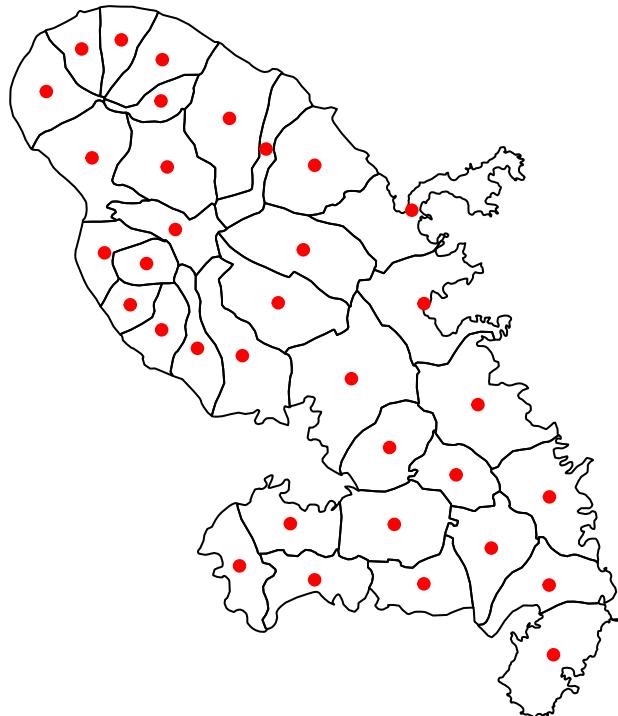
Sélection des communes intersectant Fort-de-France

```
fdf <- mtq[mtq$LIBGEO == "Fort-de-France", ]
mtq$fdf <- st_intersects(x = mtq, y = fdf, sparse = FALSE)
plot(st_geometry(mtq))
plot(st_geometry(mtq[mtq$fdf,]), col = "grey", add = TRUE)
```



### 1.3.3 Extraire des centroides

```
mtq_c <- st_centroid(mtq)
plot(st_geometry(mtq))
plot(st_geometry(mtq_c), add=TRUE, cex=1.2, col="red", pch=20)
```



### 1.3.4 Créer une matrice de distances

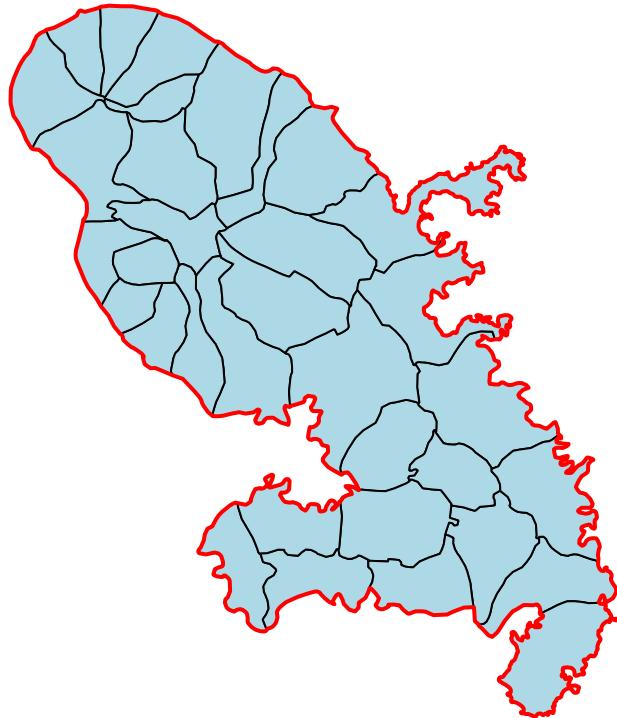
Si le système de projection du jeu de données est renseigné les distances sont exprimées dans l'unité de mesure de la projection (en mètres le plus souvent).

```
mat <- st_distance(x = mtq_c, y = mtq_c)
mat[1:5,1:5]
```

```
Units: [m]
 [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.000 35297.56 3091.501 12131.617 17136.310
[2,] 35297.557 0.000 38332.602 25518.913 18605.249
[3,] 3091.501 38332.600 0.000 15094.702 20226.198
[4,] 12131.617 25518.911 15094.702 0.000 7177.011
[5,] 17136.310 18605.250 20226.198 7177.011 0.000
```

### 1.3.5 Agréger des polygones

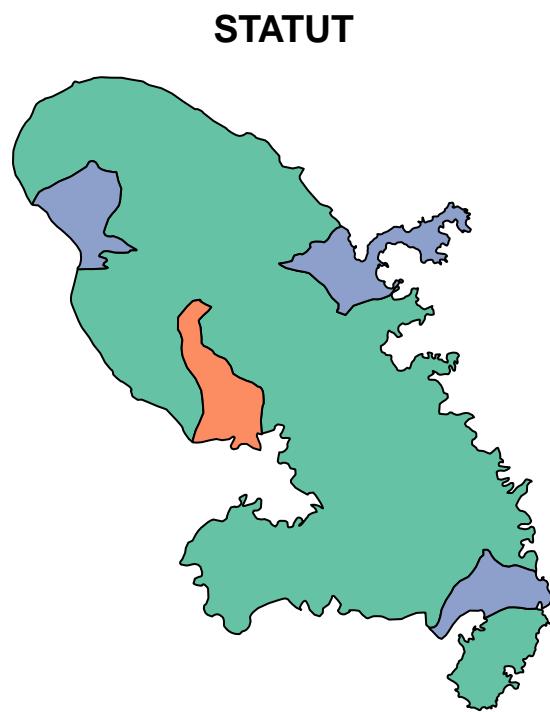
```
mtq_u <- st_union(mtz)
plot(st_geometry(mtz), col="lightblue")
plot(st_geometry(mtz_u), add=T, lwd=2, border = "red")
```



### 1.3.6 Agréger des polygones en fonction d'une variable

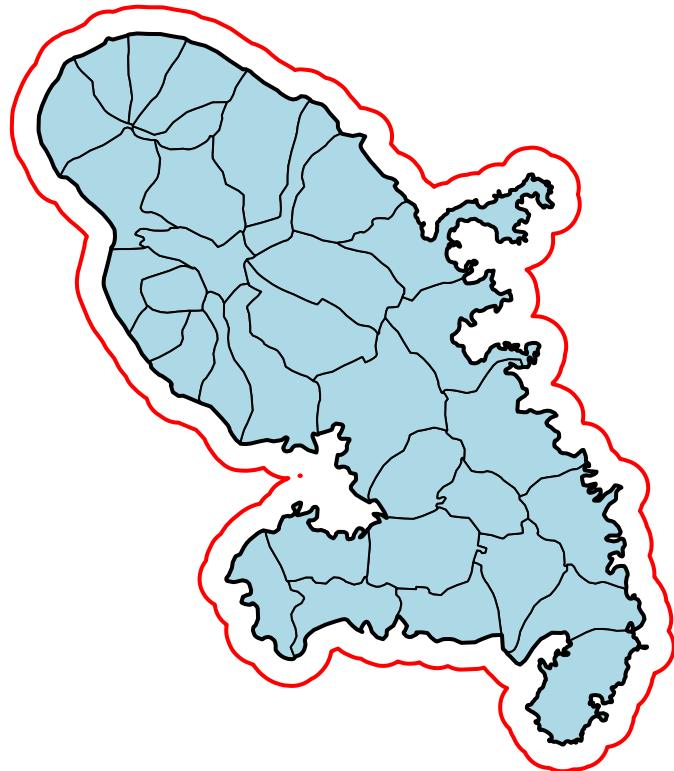
```
library(dplyr)
mtq_u2 <- mtq %>%
  group_by(STATUT) %>%
```

```
summarize(P13_POP=sum(P13_POP))
plot(mtq_u2["STATUT"], key.pos = NULL)
```



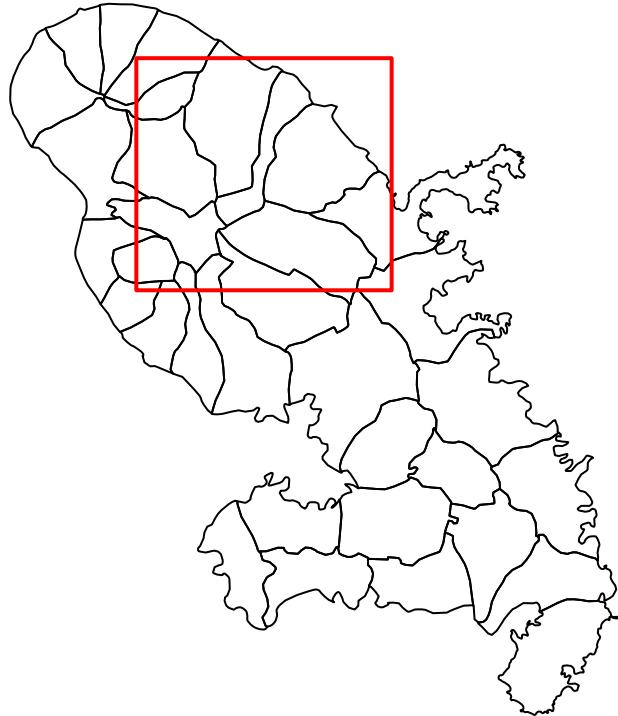
### 1.3.7 Construire une zone tampon

```
mtq_b <- st_buffer(x = mtq_u, dist = 2000)
plot(st_geometry(mtq), col="lightblue")
plot(st_geometry(mtq_u), add=T, lwd=2)
plot(st_geometry(mtq_b), add=T, lwd=2, border = "red")
```

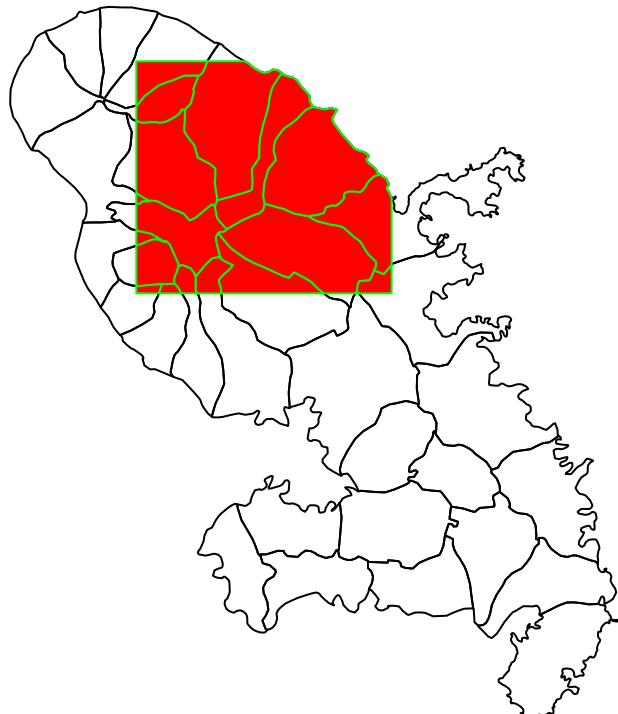


### 1.3.8 Réaliser une intersection

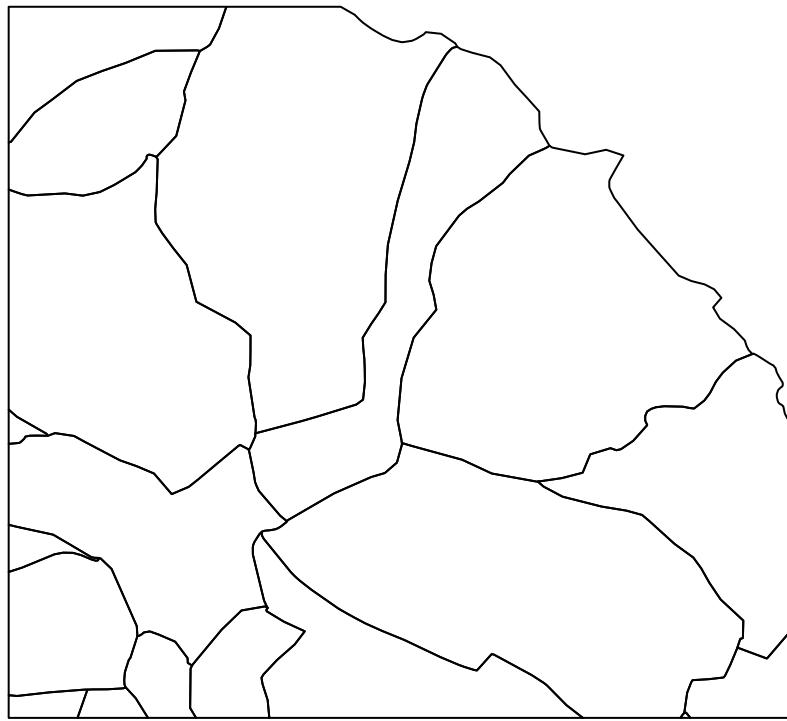
```
m <- rbind(c(700015,1624212), c(700015,1641586), c(719127,1641586),
            c(719127,1624212), c(700015,1624212))
p <- st_sf(st_sfc(st_polygon(list(m))), crs = st_crs(mtq))
plot(st_geometry(mtq))
plot(p, border="red", lwd=2, add=T)
```



```
mtq_z <- st_intersection(x = mtq, y = p)
plot(st_geometry(mtq))
plot(st_geometry(mtq_z), col="red", border="green", add=T)
```

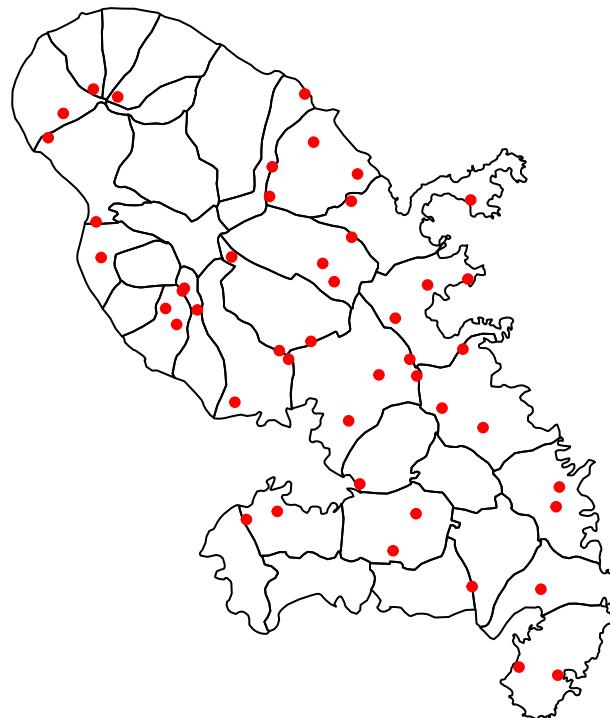


```
plot(st_geometry(mtq_z))
```



### 1.3.9 Compter des points dans un polygone

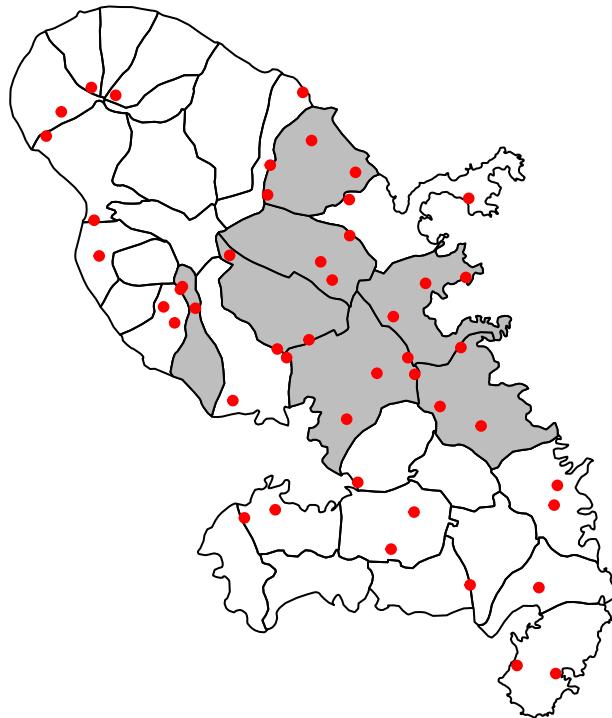
```
pts <- st_sample(x = mtq, size = 50)
plot(st_geometry(mtq))
plot(pts, pch = 20, col = "red", add=TRUE, cex = 1)
```



```

inter <- st_intersects(mtq, pts)
mtq$nbpts <- sapply(X = inter, FUN = length)
plot(st_geometry(mtq))
plot(st_geometry(mtq[mtq$nbpts>2,]), col = "grey", add=TRUE)
plot(pts, pch = 20, col = "red", add=TRUE, cex = 1)

```



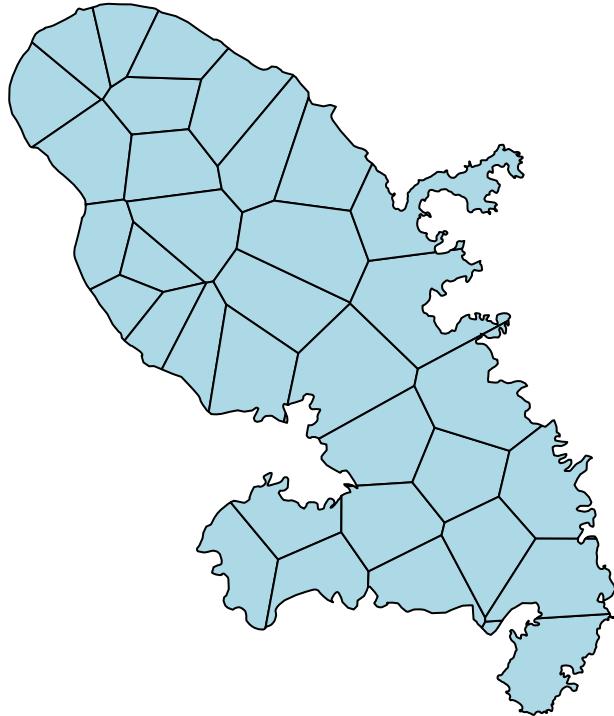
### 1.3.10 Construire des polygones de Voronoi

google: “st\_voronoi R sf” (<https://github.com/r-spatial/sf/issues/474> & <https://stackoverflow.com/questions/45719790/create-voronoi-polygon-with-simple-feature-in-r>)

```

mtq_v <- st_voronoi(x = st_union(mtq_c))
mtq_v <- st_intersection(st_cast(mtq_v), st_union(mtq))
mtq_v <- st_join(x = st_sf(mtq_v), y = mtq_c, join=st_intersects)
mtq_v <- st_cast(mtq_v, "MULTIPOLYGON")
plot(st_geometry(mtq_v), col='lightblue')

```



## 1.4 Géocodage d'adresses

Plusieurs packages permettent de géocoder des adresses.

- `photon` (?)

```
# remotes::install_github(repo = 'rCarto/photon')
library(photon)
address <- c("19 rue Michel Bakounine, 29600 Morlaix, France",
           "8 place Paul Ricoeur, 75013 Paris")
place <- photon::geocode(address, limit = 1, key = "place", lang = "fr")
place

location      osm_id osm_type name housenumber
1 19 rue Michel Bakounine, 29600 Morlaix, France 3241060871      N <NA>          19
2             8 place Paul Ricoeur, 75013 Paris 2608793979      N <NA>          8
            street postcode    city       state country osm_key osm_value      lon
1 Rue Michel Bakounine     29600 Morlaix      Bretagne France   place    house -3.816435
2 Place Paul Ricoeur     75013   Paris Île-de-France France   place    house  2.382483
            lat  msg
1 48.59041 <NA>
2 48.82670 <NA>
```

- `nominatim` (?)

```
# remotes::install_github("hrbrmstr/nominatim")
library(nominatim)
address <- c(URLencode("19 rue Michel Bakounine, 29600 Morlaix, France"),
           URLencode("8 place Paul Ricoeur, 75013 Paris"))
place <- osm_geocode(address,
                     country_codes = "FR",
```

```

key = "UneClefMapQuestValide")
place

place_id
1 44644129
2 27209988

licence

1 Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.org/copyright
2 Data © OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.org/copyright
osm_type    osm_id      lat      lon
1   node 3241060871 48.59041 -3.816435
2   node 2608793979 48.82670  2.382483

disposition

1 19, Rue Michel Bakounine, Ploujean, Kerozar, Morlaix, Finistère, Brittany, Metropolitan France, 29600
2 8, Place Paul Ricoeur, Gare, 13th Arrondissement, Paris, Ile-de-France, Metropolitan France, 75013
class type importance bbox_left bbox_top bbox_right bbox_bottom
1 place house      0.741 48.59036 48.59046 -3.816485 -3.816384
2 place house      0.631 48.82665 48.82675  2.382433  2.382533

```

- banR (?), pour des adresses en France uniquement.

```

library(banR)
address <- c("19 rue Michel Bakounine, 29600 Morlaix, France",
           "8 place Paul Ricoeur, 75013 Paris")
place <- geocode_tbl(tbl = data.frame(address), adresse = "address")
place

```

```

# A tibble: 2 x 14
  address latitude longitude result_label result_score result_type result_id
  <chr>     <dbl>     <dbl> <chr>          <dbl> <chr>       <chr>
1 19 rue~     48.6      -3.82 19 Rue Mich~        0.78 housenumber ADRNIVX_~
2 8 plac~     48.8       2.38  8 Place Pau~        0.93 housenumber ADRNIVX_~
# ... with 7 more variables: result_housenumber <int>, result_name <chr>,
#   result_street <chr>, result_postcode <int>, result_city <chr>, result_context <chr>,
#   result_citycode <chr>

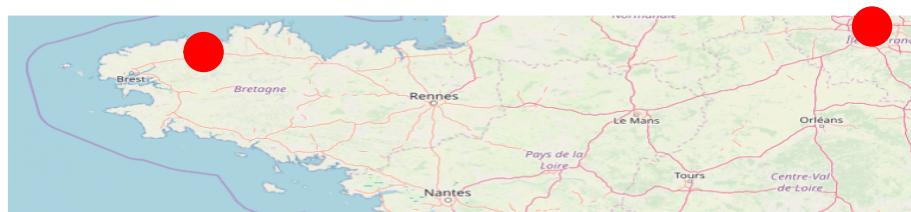
```

### Transformer les données en objet sf

```

library(sf)
library(cartography)
place_sf <- st_as_sf(place, coords = c("longitude", "latitude"), crs = 4326)
osm_fr <- getTiles(x = place_sf, zoom = 7 )
tilesLayer(osm_fr)
plot(st_geometry(place_sf), pch = 20, cex = 4, col = "red", add=T)

```



## 1.5 Importer des données OSM

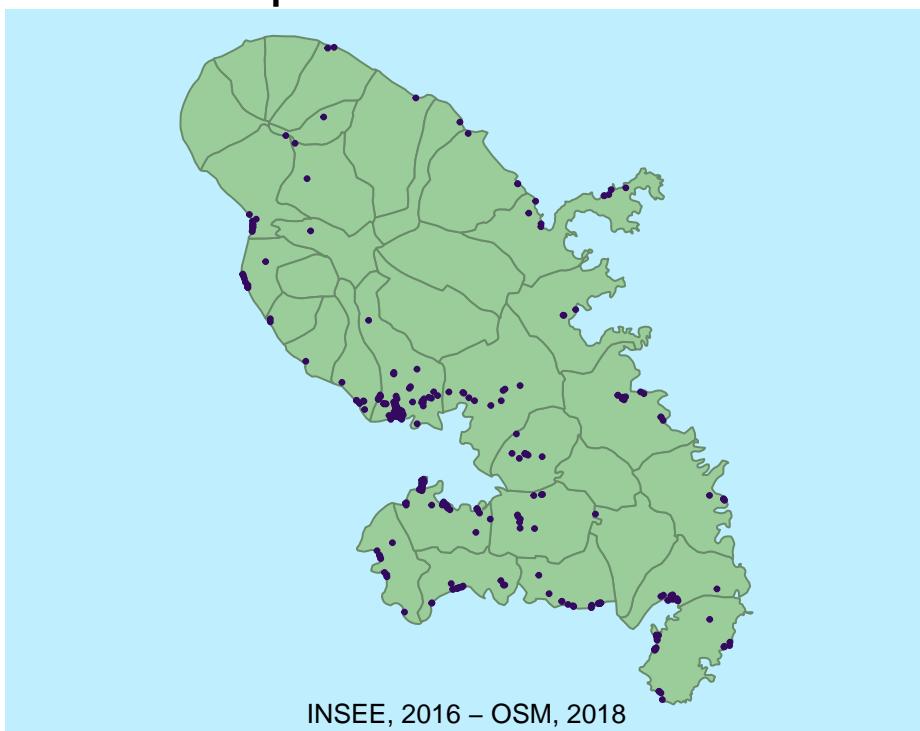
OpenStreetMap (OSM) est un projet de cartographie participative qui a pour but de constituer une base de données géographiques libre à l'échelle mondiale. OpenStreetMap vous permet de voir, modifier et utiliser des données géographiques dans le Monde entier.

Le package `osmdata` (?) permet d'extraire des données vectorielles depuis OSM.

```
library(sf)
library(osmdata)
mtq <- st_read("data/martinique.shp", quiet = TRUE)
# Définition d'une bounding box
q <- opq(bbox=st_bbox(st_transform(mtq,4326)))
# Extraction des restaurants
res <- add_osm_feature(opq = q, key = 'amenity', value = "restaurant")
res.sf <- osmdata_sf(res)
res.sf.pts <- res.sf$osm_points[!is.na(res.sf$osm_points$amenity),]
res.sf.pol <- res.sf$osm_polygons
st_geometry(res.sf.pol) <- st_centroid(st_geometry(res.sf.pol))
# Extraction des fast food
ff <- add_osm_feature(opq = q, key = 'amenity',value = "fast_food")
ff.sf <- osmdata_sf(ff)
ff.sf.pts <- ff.sf$osm_points[!is.na(ff.sf$osm_points$amenity),]
ff.sf.pol <- ff.sf$osm_polygons
st_geometry(ff.sf.pol) <- st_centroid(st_geometry(ff.sf.pol))
# Extraction des cafés
caf <- add_osm_feature(opq = q, key = 'amenity', value = "cafe" )
caf.sf <- osmdata_sf(caf)
caf.sf.pts <- caf.sf$osm_points[!is.na(caf.sf$osm_points$amenity),]
caf.sf.pol <- caf.sf$osm_polygons
st_geometry(caf.sf.pol) <- st_centroid(st_geometry(caf.sf.pol))
# regroupement des 3 types d'établissement
resto <- rbind(
  res.sf.pts[, c("osm_id", "name")], res.sf.pol[, c("osm_id", "name")],
  ff.sf.pts[, c("osm_id", "name")], ff.sf.pol[, c("osm_id", "name")],
  caf.sf.pts[, c("osm_id", "name")], caf.sf.pol[, c("osm_id", "name")]
)
resto <- st_transform(resto, st_crs(mtq))

# Affichage des restaurants
plot(st_geometry(mtq), col="darkseagreen3", border="darkseagreen4",
      bg = "lightblue1")
plot(st_geometry(resto), add=TRUE, pch=20, col = "#330A5FFF", cex = 0.5)
title("Répartition des restaurants")
mtext(text = "INSEE, 2016 - OSM, 2018", side = 1, line = -1, cex = 0.8)
```

## Répartition des restaurants



# Chapter 2

## Cartographie thématique

Nous ne détaillerons pas ici les règles de la cartographie thématique. Le lecteur pourra se référer à divers ouvrages de référence : ?, ?, ?

### 2.1 Le package `cartography`

Le package `cartography` (?) permet de créer et d'intégrer des cartes thématiques dans sa chaîne de traitements en R. Il permet des représentations cartographiques telles que les cartes de symboles proportionnels, des cartes choroplèthes, des typologies, des cartes de flux ou des cartes de discontinuités. Il offre également des fonctions qui permettent d'améliorer la réalisation de la carte, comme des palettes de couleur, des éléments d'habillage (échelle, flèche du nord, titre, légende...), d'y rattacher des labels ou d'accéder à des APIs cartographiques.

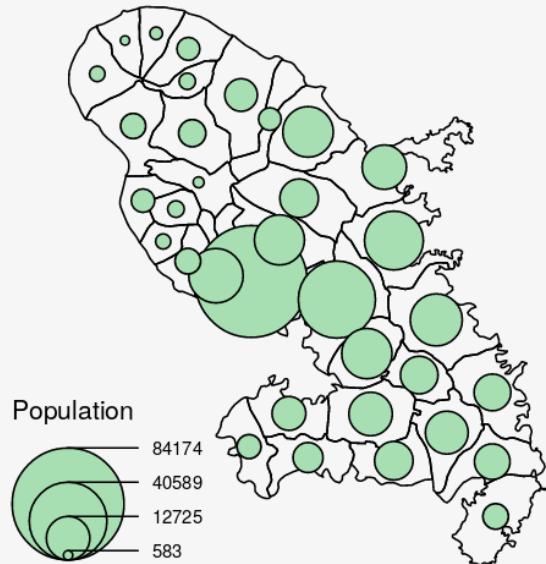
Pour utiliser ce package plusieurs sources peuvent être consultées :

- La documentation du package accessible sur internet ou directement dans R (`?cartography`),
- La vignette associée au package présente des exemples de scripts,
- Le blog R Géomatique qui met à disposition ressources et exemples liés au package et plus généralement à l'écosystème spatial de R,
- La cheat sheet de `cartography`, qui résume les principales fonctions du package de façon synthétique.

# Thematic maps with cartography :

Use cartography with spatial objects from sf or sp packages to create thematic maps.

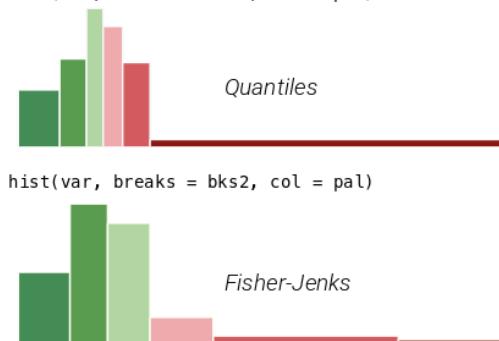
```
library(cartography)
library(sf)
mtq <- st_read("martinique.shp")
plot(st_geometry(mtq))
propSymbolsLayer(x = mtq, var = "P13_POP",
  legend.title.txt = "Population",
  col = "#a7dfb4")
```



## Classification

Available methods are: quantile, equal, q6, fisher-jenks, mean-sd, sd, geometric progression...

```
bks1 <- getBreaks(v = var, nclass = 6,
  method = "quantile")
bks2 <- getBreaks(v = var, nclass = 6,
  method = "fisher-jenks")
pal <- carto.pal("green.pal", 3, "wine.pal", 3)
hist(var, breaks = bks1, col = pal)
```



## Symbology

In most functions the x argument should be an sf object. sp objects are handled through spdf and df arguments.

Choropleth  
choroLayer(x = mtq, var = "myvar",  
method = "quantile", nclass = 8)

Typology  
typoLayer(x = mtq, var = "myvar")

Proportional Symbols  
propSymbolsLayer(x = mtq, var = "myvar",  
inches = 0.1, symbols = "circle")

Colorized Proportional Symbols (relative data)  
propSymbolsChoroLayer(x = mtq, var = "myvar",  
var2 = "myvar2")

Colorized Proportional Symbols (qualitative data)  
propSymbolsTypoLayer(x = mtq, var = "myvar",  
var2 = "myvar2")

Double Proportional Symbols  
propTrianglesLayer(x = mtq, var1 = "myvar",  
var2 = "myvar2")

OpenStreetMap Basemap (see rosm package)  
tiles <- getTiles(x = mtq, type = "osm")  
tilesLayer(tiles)

Isopleth (see SpatialPosition package)  
smoothLayer(x = mtq, var = "myvar",  
typefcf = "exponential", span = 500,  
beta = 2)

Discontinuities  
discLayer(x = mtq.borders, df = mtq\_df,  
var = "myvar", threshold = 0.5)

Flows  
propLinkLayer(x = mtq\_link, df = mtq\_df,  
var = "fij")

Dot Density  
dotDensityLayer(x = mtq, var = "myvar")

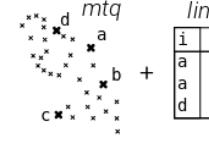
Labels  
labelLayer(x = mtq, txt = "myvar",  
halo = TRUE, overlap = FALSE)

## Transform

Polygons to Grid  
mtq\_grid <- getGrid(x = mtq,  
type = "hexagonal")



Points to Links  
mtq\_link <- getLink(x = mtq)



Polygons to Borders  
mtq\_border <- getBorder(x = mtq)



Polygons to Pencil  
mtq\_pen <- getPencil(x = mtq)

## Legends

legendChoro()  
 100  
80  
60  
40  
20  
0  
No Data

legendTypo()  
 type 1  
type 2  
type 3

legendCirclesSymbols()  
 100  
60  
30  
10

See also legendSquares(), legendGradLines(), legendCircles()

Les fonctions de `cartography` dédiées à la représentation utilisent le suffixe `Layer`. En général l'argument `x` est utilisé par un objet `sf` et l'argument `var` sert à renseigner la variable à représenter.

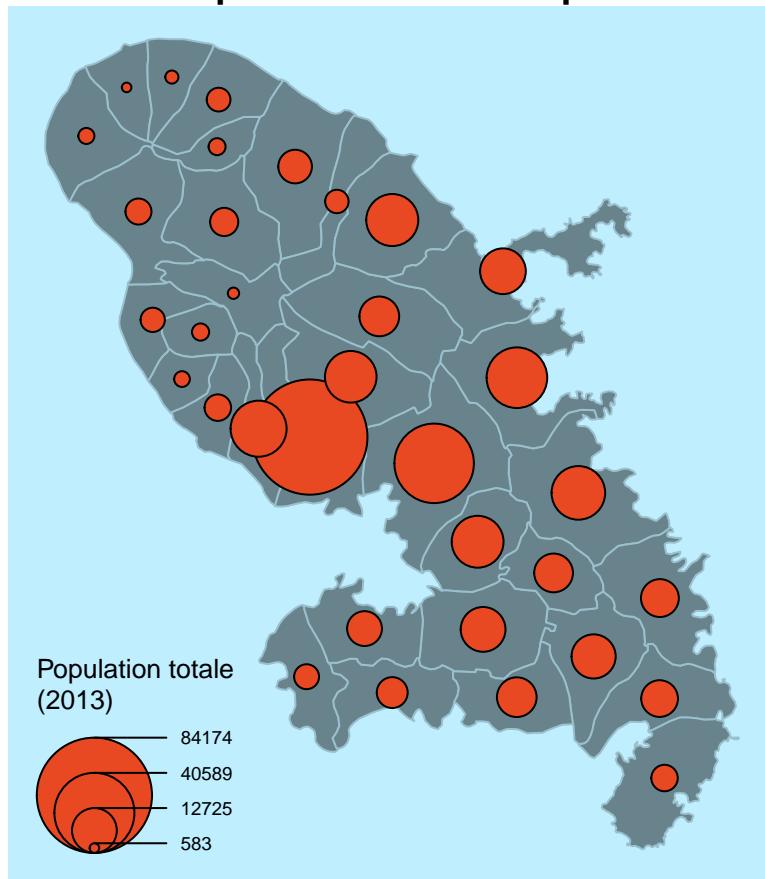
## 2.2 Représentations usuelles

### 2.2.1 Carte de symboles proportionnels

Les cartes de symboles proportionnels sont utilisées pour représenter les variables de stocks (variables quantitatives absolues, la somme et la moyenne ont un sens). La fonction `propSymbolsLayer()` propose cette représentation, plusieurs symboles sont disponibles : cercles, carrés et barres.

```
library(cartography)
library(sf)
# Import des données
mtq <- st_read("data/martinique.shp", quiet = TRUE)
# Communes
plot(
  st_geometry(mtq),
  col = "lightblue4",
  border = "lightblue3",
  bg = "lightblue1"
)
# Symboles proportionnels
propSymbolsLayer(
  x = mtq,
  var = "P13_POP",
  legend.title.txt = "Population totale\n(2013)"
)
# Titre
title(main = "Population en Martinique")
```

## Population en Martinique



### 2.2.2 Carte choroplète

Les cartes choroplèthes sont utilisées pour représenter les variables de ratios (variables quantitatives relatives, la moyenne a un sens, la somme n'a pas de sens).

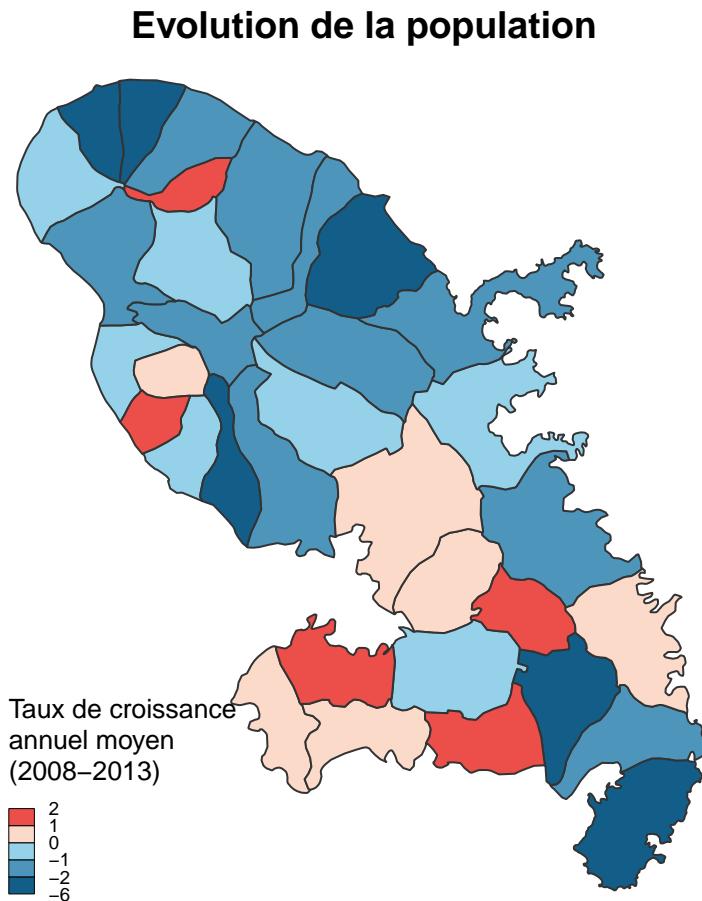
Pour ce genre de représentation il faut au préalable :

- choisir une méthode de discrétisation pour transformer une série statistique continue en classes définies par des intervalles,
- choisir un nombre de classes,
- choisir une palette de couleurs.

La fonction `choroLayer()` permet de créer des cartes choroplètes . Les arguments `nclass`, `method` et `breaks` servent à paramétrer les discrétisations et la fonction `getBreaks()` permet de travailler sur les discrétisations en dehors de la fonction `choroLayer()`. De même, l'argument `col` est utilisé pour renseigner une palette de couleur mais plusieurs fonctions peuvent être utilisées pour paramétrer les palettes en dehors de la fonction (`carto.pal()`...).

```
mtq$cagr <- (((mtq$P13_POP / mtq$P08_POP)^(1/4)) - 1) * 100
choroLayer(
  x = mtq,
  var = "cagr",
  breaks = c(-6.14,-2,-1,0,1,2),
  col = c("#135D89", "#4D95BA", "#96D1EA", "#FCDACA", "#EC4E49"),
  legend.title.txt = "Taux de croissance\nannuel moyen\n(2008-2013)"
```

```
)
title(main = "Evolution de la population")
```

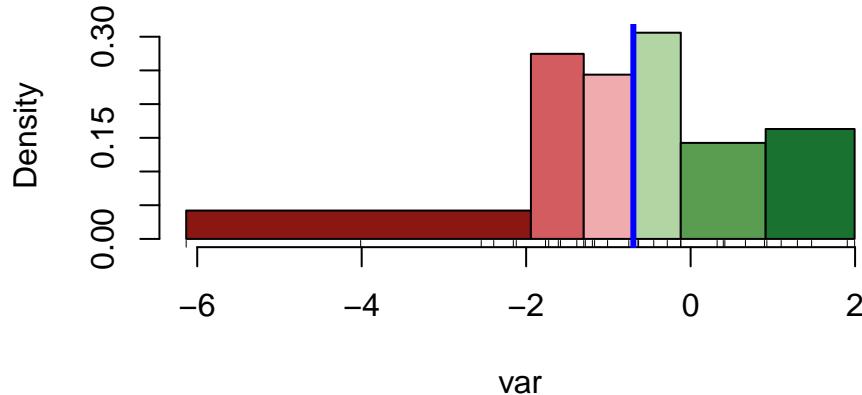


### 2.2.2.1 Discrétilisations

La fonction `getBreaks()` met à disposition les méthodes de discrétilisations de variables classique : quantiles, moyenn/écart-type, amplitudes égales, moyennes emboitées, Fisher-Jenks, géométrique ...

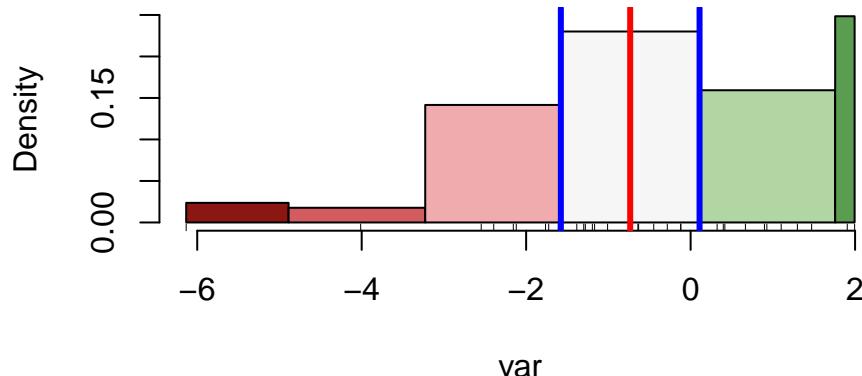
```
var <- mtq$cagr
moy <- mean(var)
med <- median(var)
std <- sd(var)
# Quantile intervals
breaks <- getBreaks(v = var, nclass = 6, method = "quantile")
hist(var, probability = TRUE, breaks = breaks, main="quantiles",
     col = carto.pal(pal1 = "wine.pal",3, "green.pal", 3))
rug(var)
abline(v = med, col = "blue", lwd = 3)
```

## quantiles



```
# Mean and standard deviation (msd)
breaks <- getBreaks(v = var, method = "msd", k = 1, middle = TRUE)
hist(var, probability = TRUE, breaks = breaks, main="moyenne / écart-type",
      col = carto.pal(pal1 = "wine.pal", 3, "green.pal", 2, middle = TRUE))
rug(var)
abline(v = moy, col = "red", lwd = 3)
abline(v = moy + 0.5 * std, col = "blue", lwd = 3)
abline(v = moy - 0.5 * std, col = "blue", lwd = 3)
```

## moyenne / écart-type



### 2.2.2.2 Palettes de couleurs

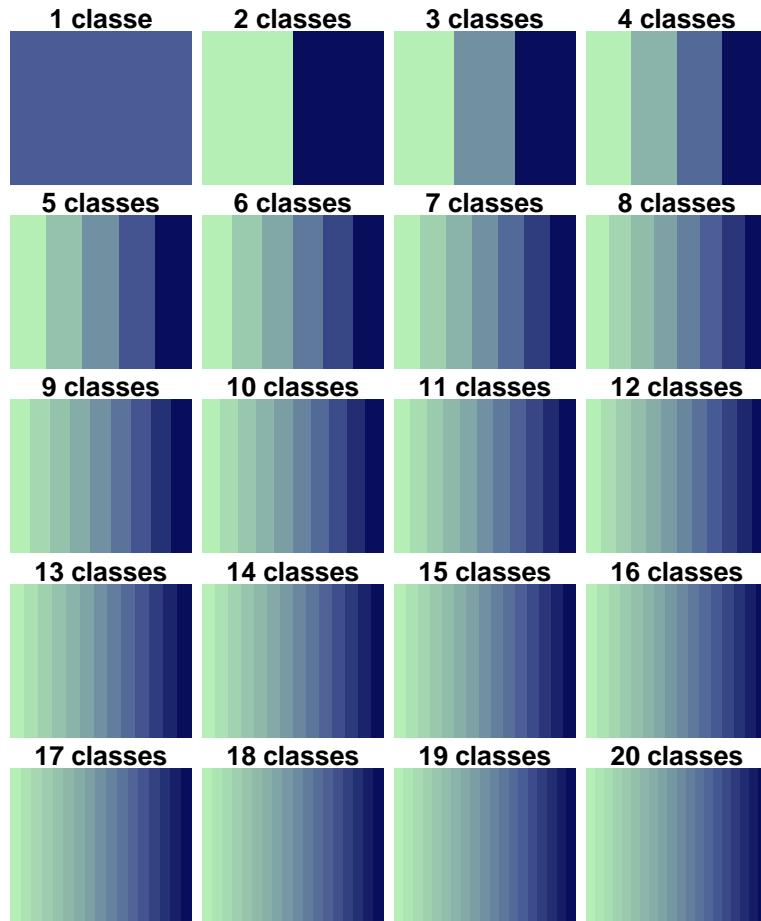
La fonction `display.carto.all()` permet d'afficher toutes palettes de couleurs disponibles dans `cartography`.

```
display.carto.all(20)
```



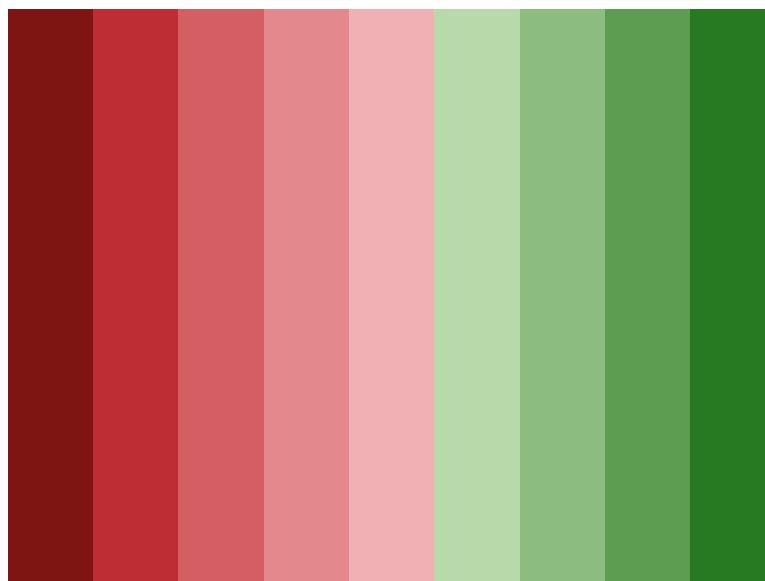
La fonction `display.carto.pal()` permet de détailler une palette de couleurs.

```
display.carto.pal("turquoise.pal")
```



La fonction `carto.pal()` permet de construire une palette de couleur. Il est possible de créer des palettes associant 2 couleurs.

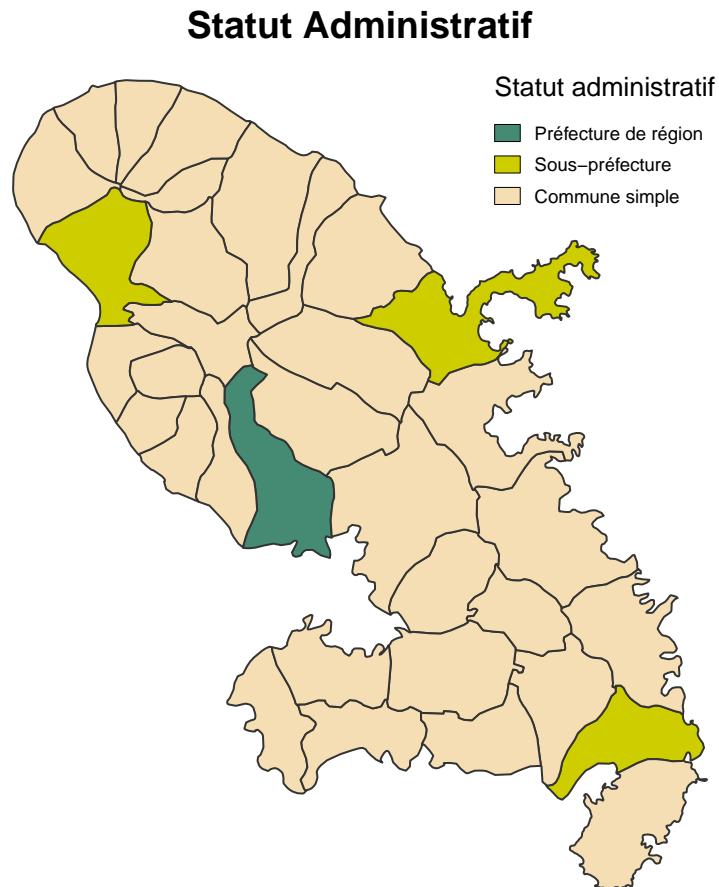
```
mypal <- carto.pal(pal1 = "wine.pal", n1 = 5, pal2 = "green.pal", n2 = 4)
image(1:9, 1, as.matrix(1:9), col=mypal, xlab = "", ylab = "", xaxt = "n",
      yaxt = "n", bty = "n")
```



### 2.2.3 Carte de typologie

Les cartes de typologies sont utilisées pour représenter les variables qualitatives. La fonction `typoLayer()` propose cette représentation. L'argument `legend.values.order` sert à ordonner les modalités dans la légende.

```
typoLayer(
  x = mtq,
  var="STATUT",
  legend.values.order = c("Préfecture de région",
                         "Sous-préfecture",
                         "Commune simple"),
  col = c("aquamarine4", "yellow3","wheat"),
  legend.pos = "topright",
  legend.title.txt = "Statut administratif"
)
title("Statut Administratif")
```



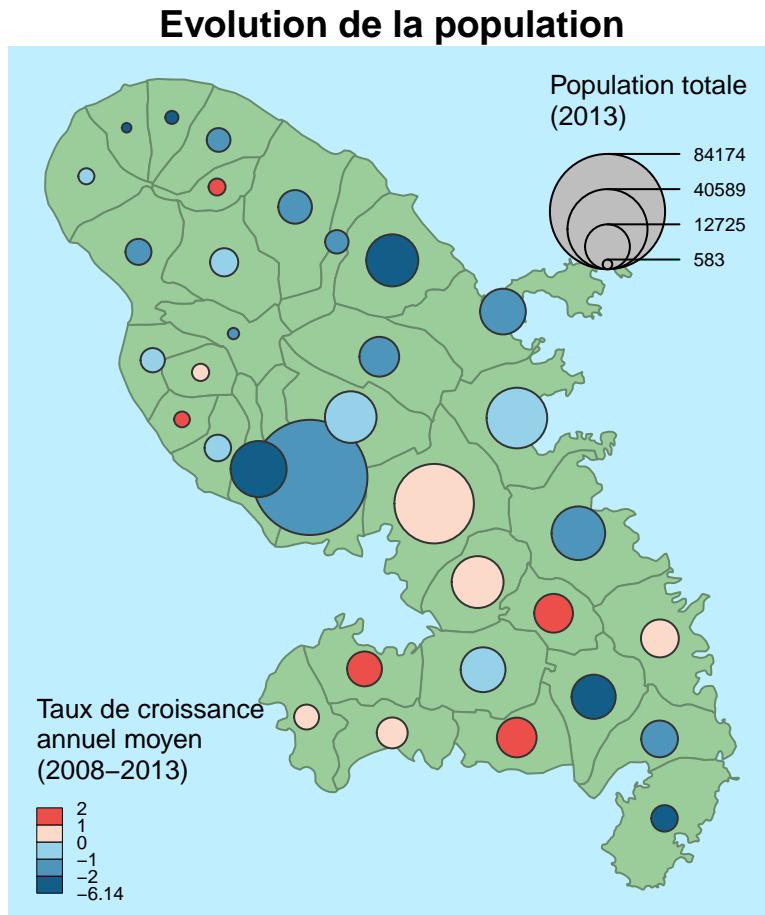
## 2.3 Combinaisons de représentations

Plusieurs fonctions sont dédiées à la représentation combinée de 2 variables.

### 2.3.1 Carte de stocks et de ratios

La fonction `propSymbolsChoroLayer()` représente des symboles proportionnels dont les surfaces sont proportionnelles aux valeurs d'une variable et dont la couleur repose sur la discréétisation d'une seconde variable. La fonction utilise les arguments des fonctions `propSymbolsLayer()` et `choroLayer()`.

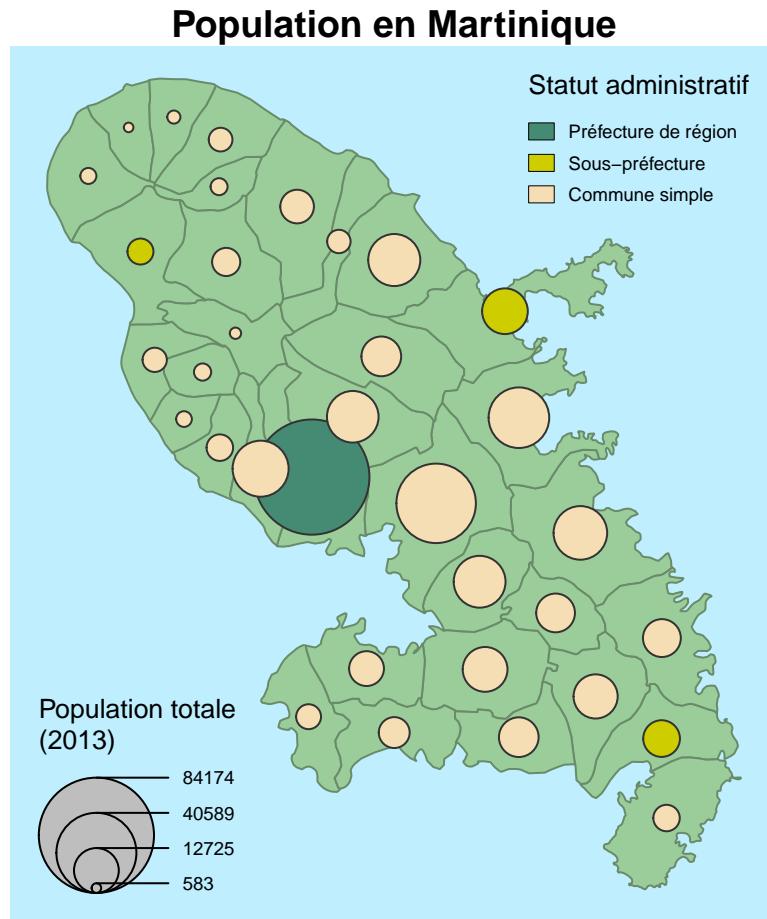
```
plot(
  st_geometry(mtq),
  col="darkseagreen3",
  border="darkseagreen4",
  bg = "lightblue1"
)
propSymbolsChoroLayer(
  x = mtq,
  var= "P13_POP",
  var2 = "cagr",
  breaks = c(-6.14,-2,-1,0,1,2),
  col = c("#135D89", "#4D95BA", "#96D1EA", "#FCDACA", "#EC4E49"),
  legend.var.pos = "topright",
  legend.var.title.txt = "Population totale\n(2013)",
  legend.var2.pos = "bottomleft",
  legend.var2.title.txt = "Taux de croissance\nannuel moyen\n(2008-2013)"
)
title("Evolution de la population")
```



### 2.3.2 Carte de stocks et de qualitative

La fonction `propSymbolsTypoLayer()` représente des symboles proportionnels dont les surfaces sont proportionnelles aux valeurs d'une variable et dont la couleur représente les modalités d'une variable qualitative. La fonction utilise les arguments des fonctions `propSymbolsLayer()` et `typoLayer()`.

```
plot(
  st_geometry(mtq),
  col="darkseagreen3",
  border="darkseagreen4",
  bg = "lightblue1"
)
propSymbolsTypoLayer(
  x = mtq,
  var = "P13_POP",
  symbols = "circle",
  var2 = "STATUT",
  col = c("aquamarine4", "yellow3","wheat"),
  legend.var.pos = "bottomleft",
  legend.var.title.txt = "Population totale\n(2013)",
  legend.var2.title.txt = "Statut administratif",
  legend.var2.values.order = c("Préfecture de région",
                               "Sous-préfecture",
                               "Commune simple")
)
title("Population en Martinique")
```



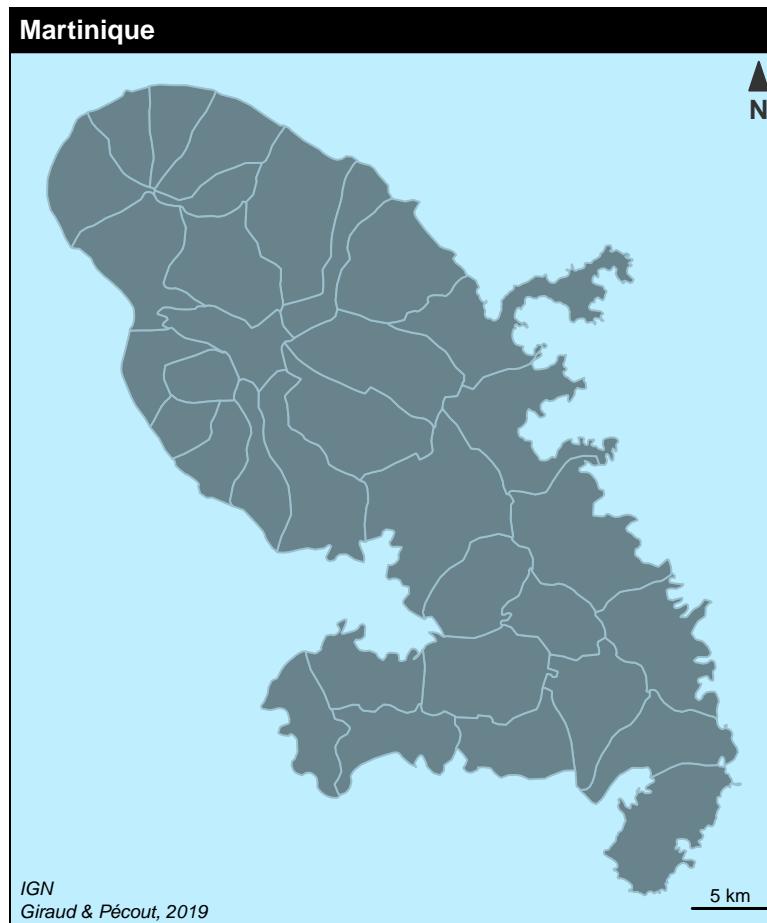
## 2.4 Éléments d'habillage

Pour être finalisée, une carte thématique doit contenir certains éléments additionnels tels que : le titre, l'auteur, la source, l'échelle, l'orientation...

### 2.4.1 Habillage complet

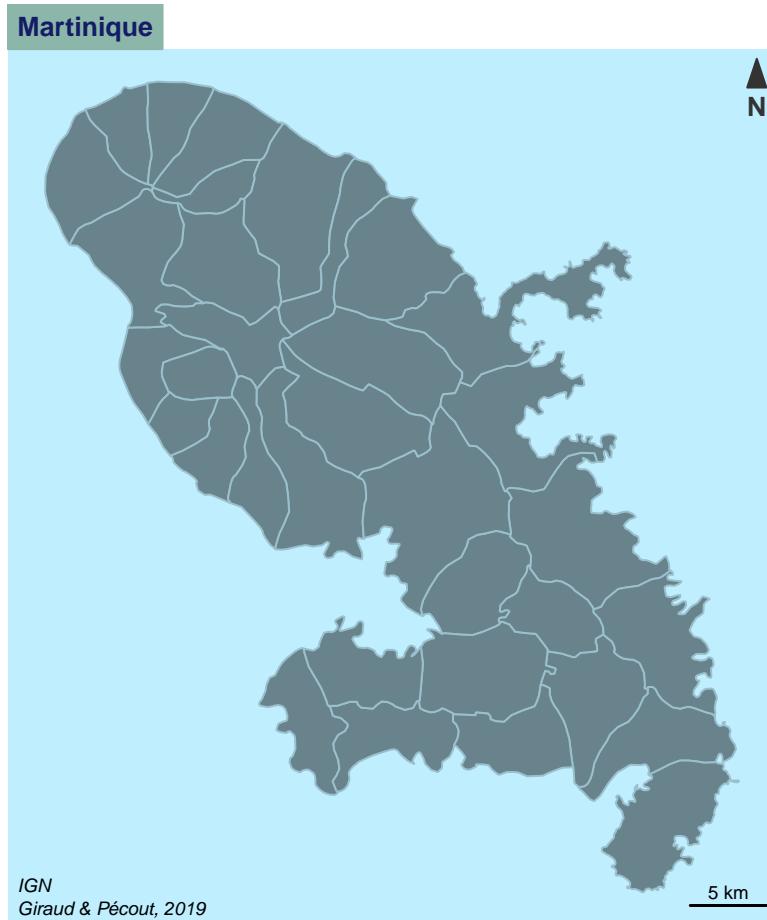
La fonction `layoutLayer()` permet d'afficher tous ces éléments.

```
plot(st_geometry(mtq), col = "lightblue4", border = "lightblue3",
     bg = "lightblue1")
layoutLayer(
  title = "Martinique",
  sources = "IGN",
  author = "Giraud & Pécout, 2019",
  north = TRUE
)
```



Plusieurs arguments permettent de paramétrer plus finement les éléments d'habillage pour aboutir à des cartes plus personnalisées (`tabtitle`, `col`, `coltitle`, `theme`...).

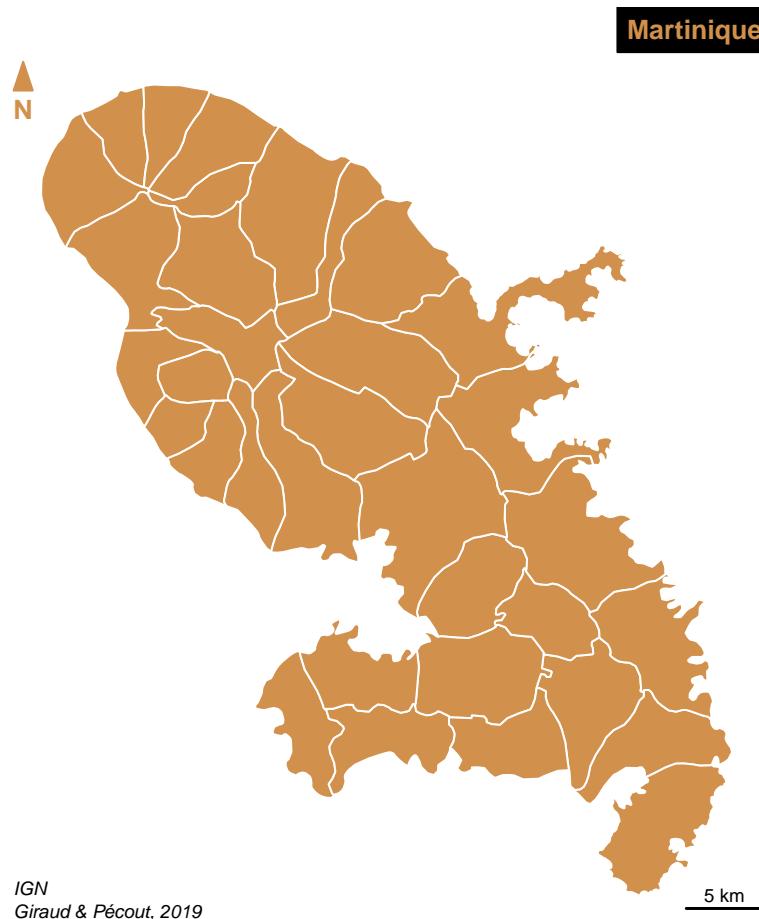
```
plot(st_geometry(mtq), col = "lightblue4", border = "lightblue3",
      bg = "lightblue1")
layoutLayer(
  title = "Martinique",
  sources = "IGN",
  author = "Giraud & Pécout, 2019",
  north = TRUE,
  scale = 5,
  frame = FALSE,
  tabtitle = TRUE,
  theme = "turquoise.pal"
)
```



#### 2.4.2 Flèche d'orientation

La fonction `north()` permet de mieux choisir la position et l'aspect de la flèche d'orientation.

```
plot(st_geometry(mtq), col = "#D1914D", border = "white")
north(pos = "topleft", col = "#D1914D")
layoutLayer(title = "Martinique", sources = "IGN",
           author = "Giraud & Pécout, 2019", frame = FALSE, scale = 5,
           coltitle = "#D1914D", tabtitle = TRUE, postitle = "right")
```



### 2.4.3 Échelle

La fonction `barscale()` permet de mieux choisir la position et l'aspect de l'échelle.

```
plot(st_geometry(mtq), col = "#D1914D", border = "white")
barscale(
  size = 5,
  lwd = 2,
  cex = 1.2,
  pos = c(713712.6,1594777)
)
layoutLayer(title = "Martinique", sources = "IGN",
           author = "Giraud & Pécout, 2019", frame = FALSE, scale = NULL,
           coltitle = "#D1914D", tabtitle = TRUE)
```



#### 2.4.4 Étiquettes

La fonction `labelLayer()` est dédiée à l'affichage d'étiquettes.

```
plot(st_geometry(mtq), col = "darkseagreen3", border = "darkseagreen4",
      bg = "#A6CAE0")
labelLayer(
  x = mtq,
  txt = "LIBGEO",
  col= "black",
  cex = 0.7,
  font = 4,
  halo = TRUE,
  bg = "white",
  r = 0.1,
  overlap = FALSE,
  show.lines = FALSE
)
layoutLayer(title = "Communes", tabtitle = TRUE, author = "INSEE, 2016",
            sources = "", north =TRUE, frame = FALSE, scale = 5)
```



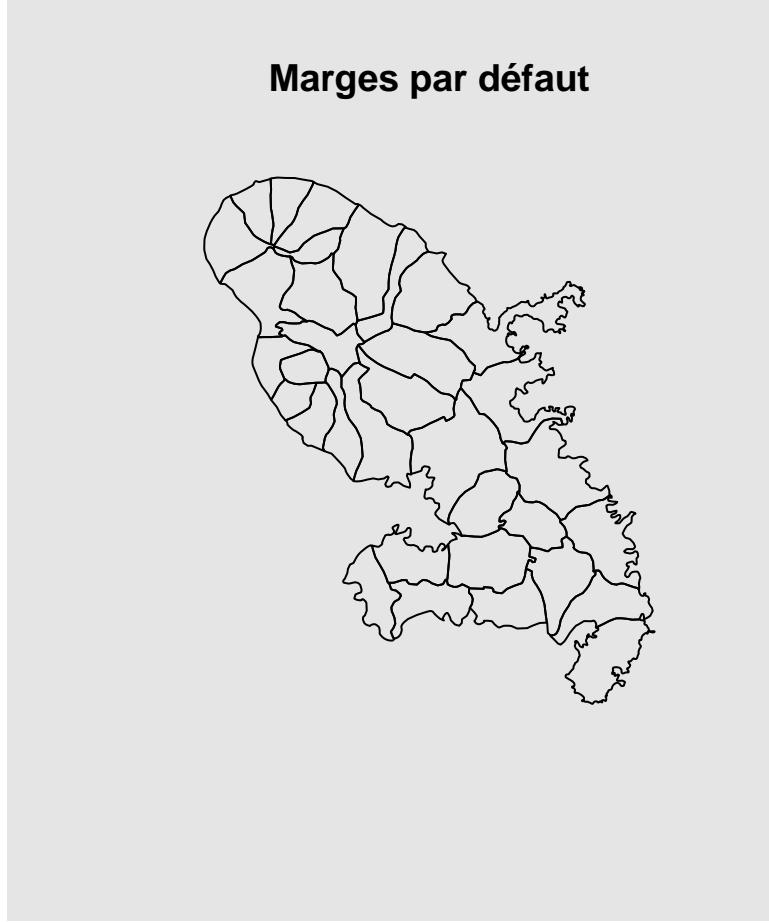
## 2.5 Autres fonctionnalités utiles

### 2.5.1 Mise en page

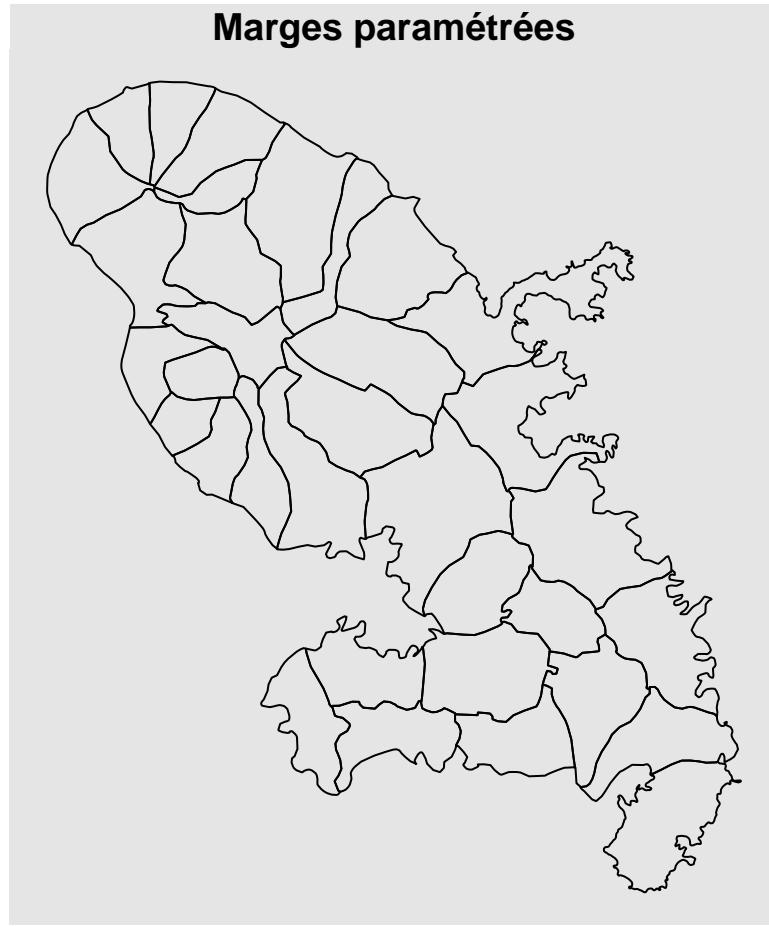
#### 2.5.1.1 Ajuster les marges d'une figure

Pour modifier les marges d'une figure (carte ou autre) il faut utiliser la fonction `par()` qui définit certains paramètres graphiques des figures et son argument `mar`. La fonction `dev.off()` efface tous les graphiques en mémoire et permet de réinitialiser les valeurs par défaut.

```
# Modification de la couleur de fond des graphiques
par(bg="grey90")
plot(st_geometry(mtq), main="Marges par défaut")
```



```
# Modification des marges
par(mar=c(0,0,1.2,0))
plot(st_geometry(mtq), main="Marges paramétrées")
```



### 2.5.1.2 Centrer la carte sur une région

Plusieurs solutions sont possibles :

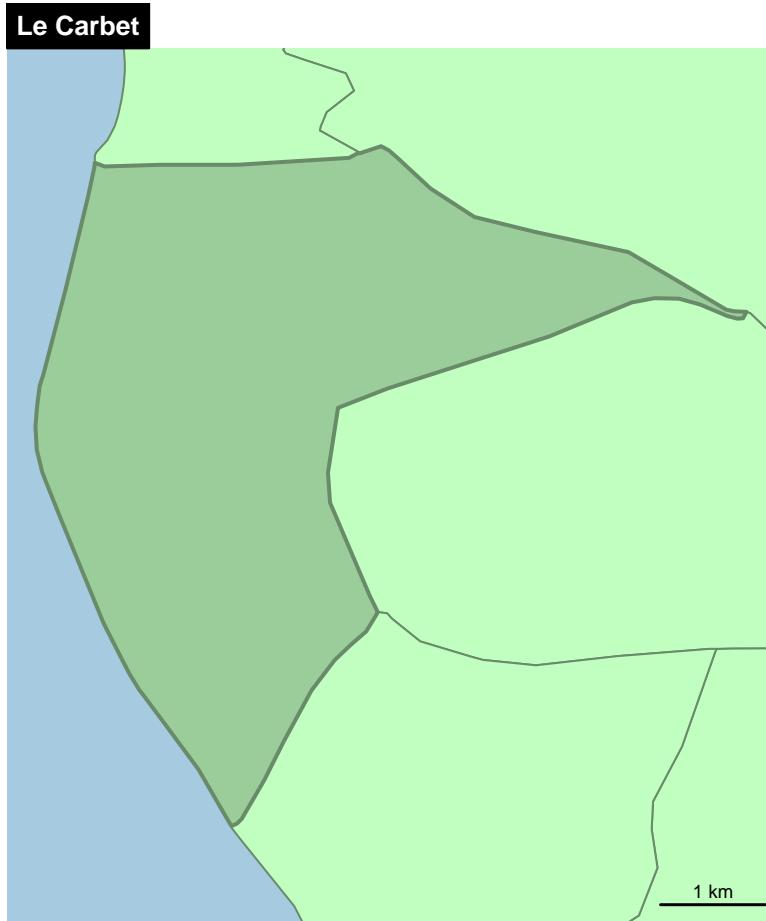
- Afficher une couche de la zone de zoom sans couleur pour le fond et les bordures puis afficher les couches que l'on souhaite afficher.

```
carbet <- mtq[mtq$LIBGEO=="Le Carbet",]
# affichage de la couche de zoom "invisible"
plot(
  st_geometry(carbet),
  col = NA,
  border = NA,
  bg = "#A6CAE0"
)
# affichage des communes
plot(
  st_geometry(mtq),
  col = "darkseagreen1",
  border = "darkseagreen4",
  add=TRUE
)
# affichage de la couche d'intérêt
plot(
```

```

    st_geometry(carbet),
    col = "darkseagreen3",
    border = "darkseagreen4",
    lwd = 2,
    add=TRUE
)
layoutLayer(
  title = "Le Carbet",
  sources = "",
  author = "",
  scale = 1,
  tabtitle = TRUE,
  frame=FALSE
)

```



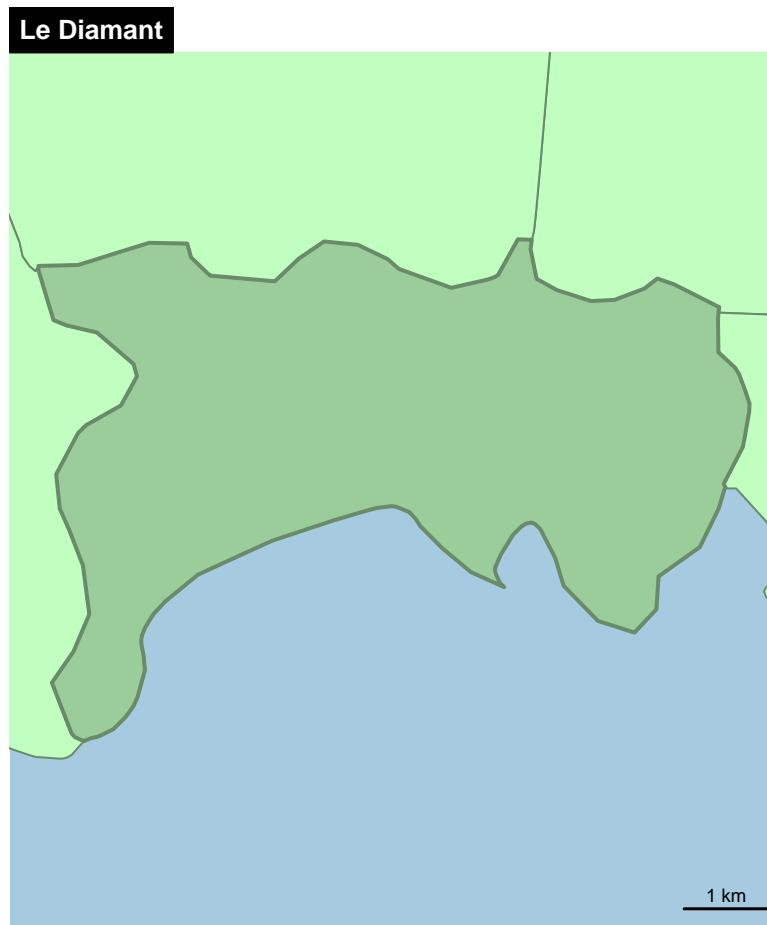
- Utiliser les paramètres `xlim` et `ylim` de la fonction `plot()` avec les valeurs fournies par la fonction `st_bbox()`

```

diams <- mtq[mtq$LIBGEO=="Le Diamant",]
diams_bb <- st_bbox(diams)
# affichage des communes
plot(
  st_geometry(mtq),
  col = "darkseagreen1",
  border = "darkseagreen4",

```

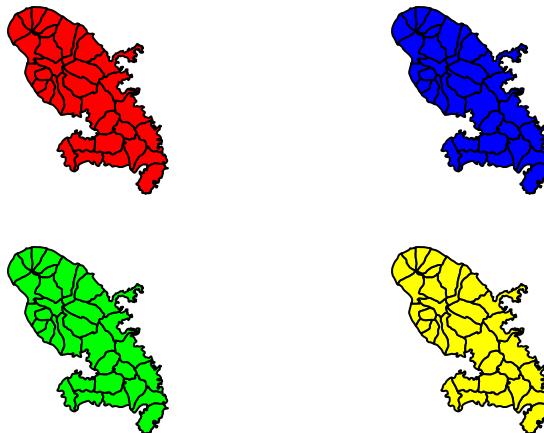
```
  xlim = diams_bb[c(1,3)],
  ylim = diams_bb[c(2,4)],
  bg = "#A6CAE0"
)
# affichage de la couche d'intérêt
plot(
  st_geometry(diams),
  col = "darkseagreen3",
  border = "darkseagreen4",
  lwd = 2,
  add=TRUE
)
layoutLayer(
  title = "Le Diamant",
  sources = "",
  author = "",
  scale = 1,
  tabtitle = TRUE,
  frame=FALSE
)
)
```



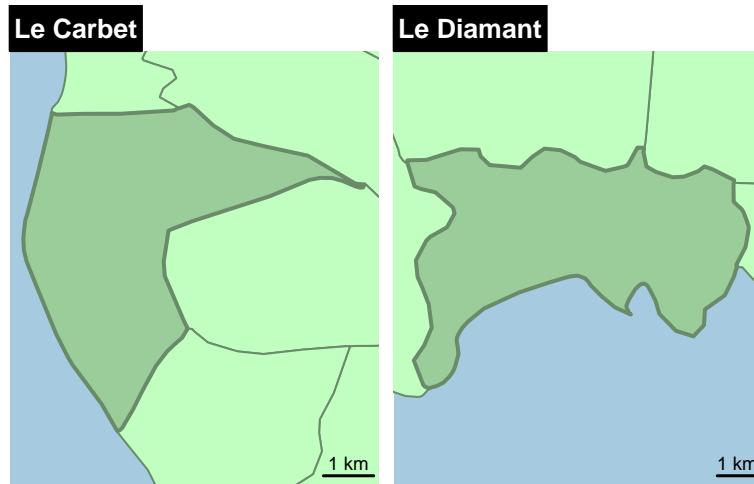
### 2.5.1.3 Afficher plusieurs cartes sur la même figure

Il faut ici utiliser l'argument `mfrow` de la fonction `par()`. Le premier chiffre représente le nombre lignes et le deuxième le nombre de colonnes.

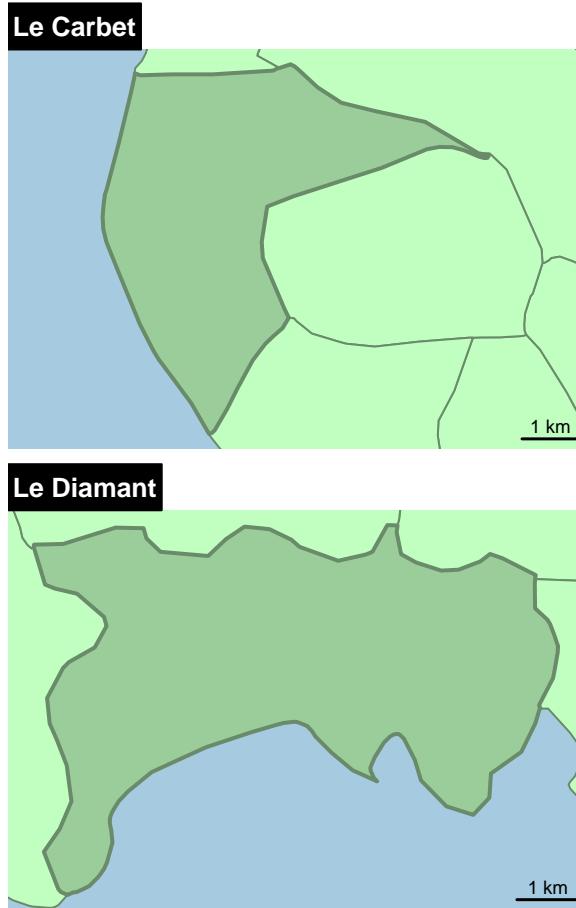
```
# deux lignes et deux colonnes
par(mfrow=c(2,2))
plot(st_geometry(mtq), col="red")
plot(st_geometry(mtq), col="blue")
plot(st_geometry(mtq), col="green")
plot(st_geometry(mtq), col="yellow")
```



```
# une ligne et deux colonnes
par(mfrow=c(1,2), mar = c(0,.2,1.2,.2))
# 1ere carte
carbet_bb <- st_bbox(carbet)
plot(st_geometry(mtq), col = "darkseagreen1", border = "darkseagreen4",
      xlim = carbet_bb[c(1,3)], ylim = carbet_bb[c(2,4)], bg = "#A6CAE0")
plot(st_geometry(carbet), col = "darkseagreen3", border = "darkseagreen4",
      lwd = 2, add=TRUE)
layoutLayer(title = "Le Carbet", sources = "", author = "", scale = 1,
            tabtitle = TRUE, frame=FALSE)
# 2eme carte
plot(st_geometry(mtq), col = "darkseagreen1", border = "darkseagreen4",
      xlim = diams_bb[c(1,3)], ylim = diams_bb[c(2,4)], bg = "#A6CAE0")
plot(st_geometry(diams), col = "darkseagreen3", border = "darkseagreen4",
      lwd = 2, add=TRUE)
layoutLayer(title = "Le Diamant", sources = "", author = "", scale = 1,
            tabtitle = TRUE, frame=FALSE)
```



```
# une ligne et deux colonnes
par(mfrow=c(2,1), mar = c(0.2,0,1.4,0))
# 1ere carte
carbet_bb <- st_bbox(carbet)
plot(st_geometry(mtq), col = "darkseagreen1", border = "darkseagreen4",
      xlim = carbet_bb[c(1,3)], ylim = carbet_bb[c(2,4)], bg = "#A6CAE0")
plot(st_geometry(carbet), col = "darkseagreen3", border = "darkseagreen4",
      lwd = 2, add=TRUE)
layoutLayer(title = "Le Carbet", sources = "", author = "", scale = 1,
            tabtitle = TRUE, frame=FALSE)
# 2eme carte
plot(st_geometry(mtq), col = "darkseagreen1", border = "darkseagreen4",
      xlim = diams_bb[c(1,3)], ylim = diams_bb[c(2,4)], bg = "#A6CAE0")
plot(st_geometry(diams), col = "darkseagreen3", border = "darkseagreen4",
      lwd = 2, add=TRUE)
layoutLayer(title = "Le Diamant", sources = "", author = "", scale = 1,
            tabtitle = TRUE, frame=FALSE)
```



#### 2.5.1.4 Obtenir un ratio de figure adapté

Il est assez difficile d'exporter des figures (cartes ou autres) dont le ratio hauteur/largeur soit satisfaisant. Le ratio par défaut des figure au format png est de 1 (480x480 pixels) :

```
png(filename = "img/martinique1.png", res = 96)
par(mar = c(0,0,1.2,0), bg = "grey90")
plot(st_geometry(mtq), bg = "#A6CAE0", col = "#D1914D", border = "white")
layoutLayer(title = "Martinique", sources = "", author = "", scale = NULL)
dev.off()
```

Sur cette carte beaucoup d'espace est perdu à l'est et à l'ouest de l'île.

La fonction `getFigDim()` de `cartography` permet de choisir un ratio hauteur/largeur correspondant à l'emprise d'un objet `sf` en prenant en compte une largeur (ou hauteur) fixée, les paramètres de marges et la résolution souhaitée.

```
getFigDim(x = mtq, width = 480, mar = c(0,0,1.2,0), res = 96)
```

[1] 480 583

```
png(filename = "img/martinique2.png", width = 480, height = 583, res = 96)
par(mar = c(0,0,1.2,0), bg = "grey90")
plot(st_geometry(mtq), bg = "#A6CAE0", col = "#D1914D", border = "white")
layoutLayer(title = "Martinique", sources = "", author = "", scale = NULL)
dev.off()
```

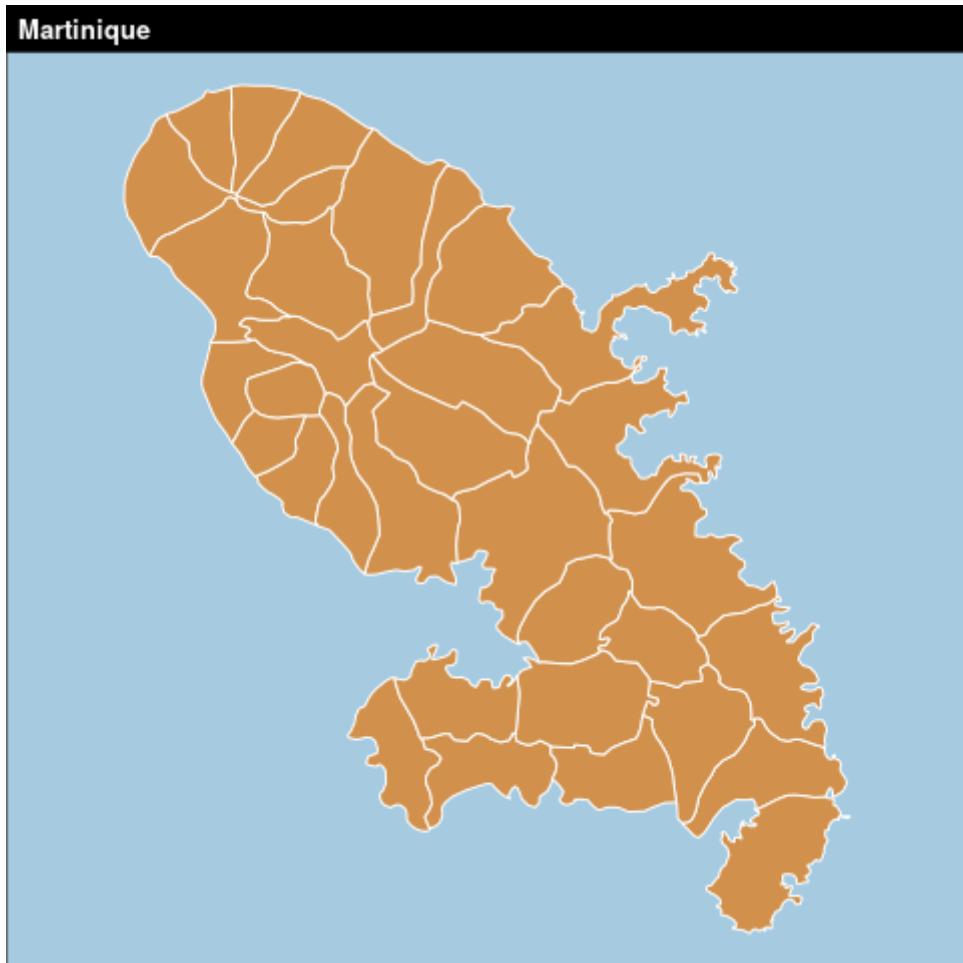


Figure 2.1:

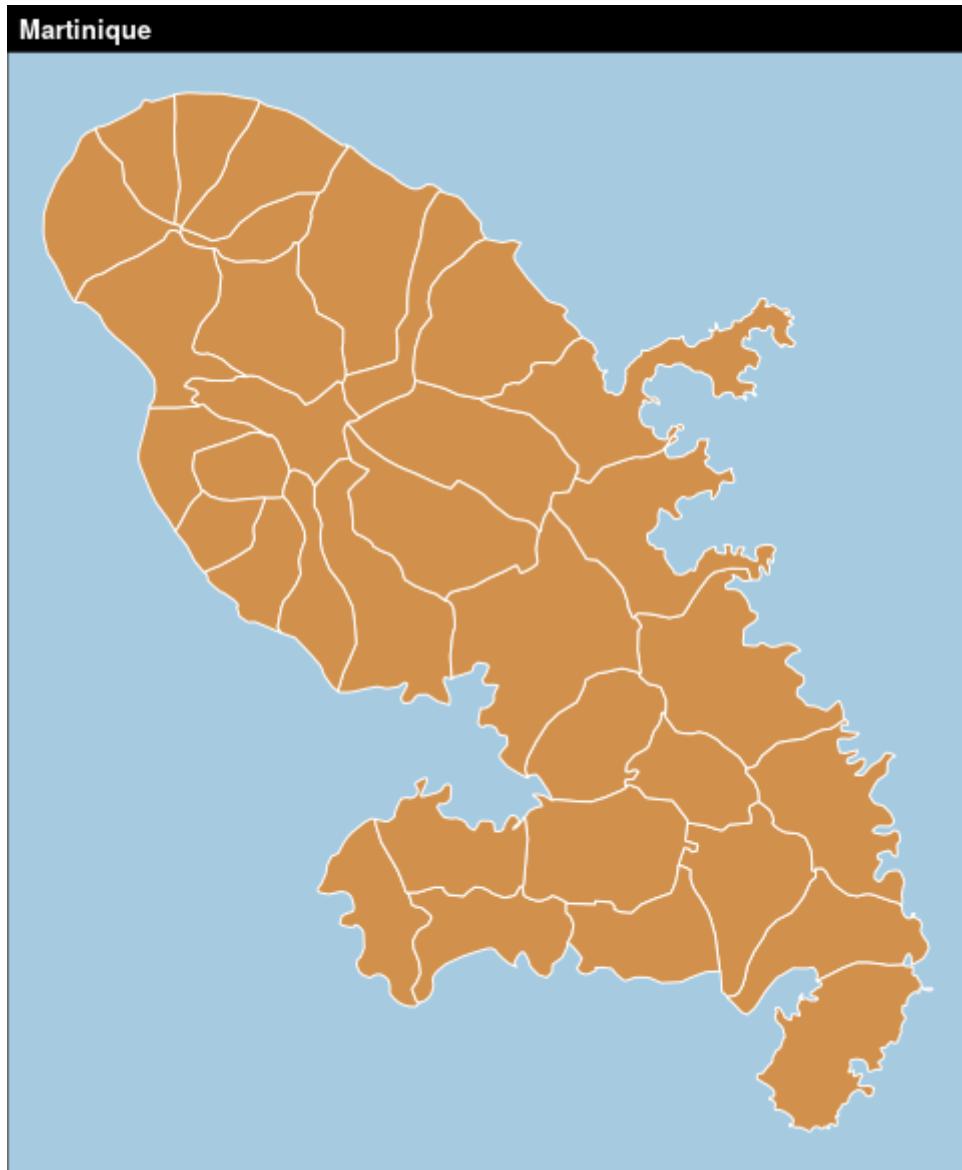


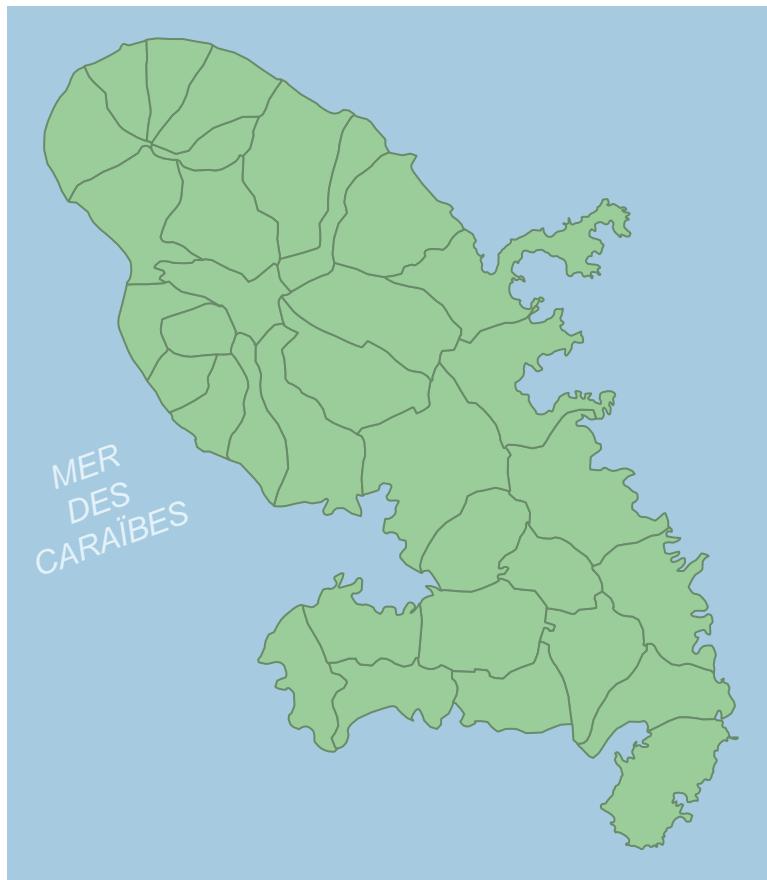
Figure 2.2:

L'emprise de cette carte est exactement celle de l'île.

#### 2.5.1.5 Placer précisément un élément sur la carte

La fonction `locator()` permet de cliquer sur une figure et d'obtenir les coordonnées d'un point dans le système de coordonnées de la figure (de la carte).

```
plot(st_geometry(mtq), col = "darkseagreen3", border = "darkseagreen4",
      bg = "#A6CAE0")
text(x = 694019, y = 1615161,
     labels = "MER\nDES\nCARAÏBES",
     col = "#e3f1f9", font = 3, srt=20 )
```



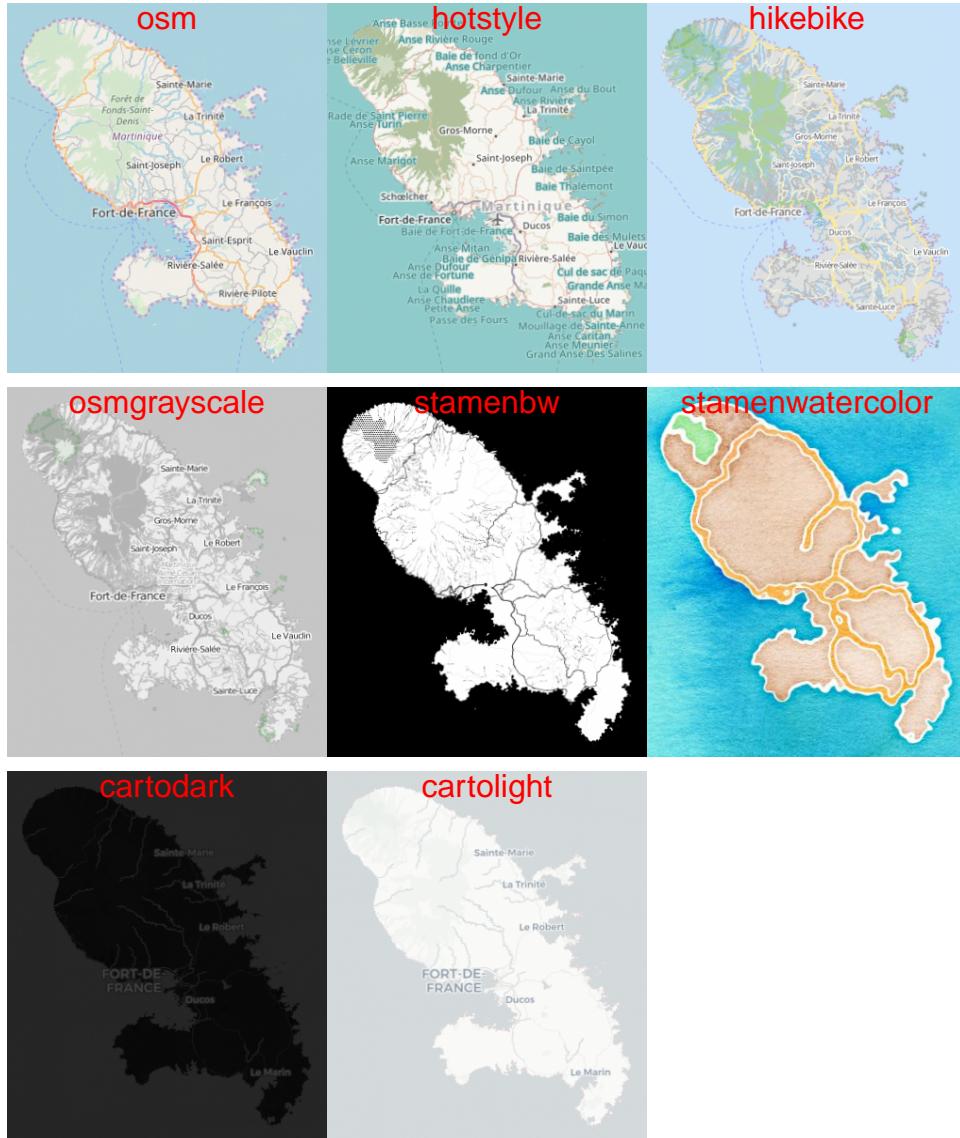
`locator()` peut être utilisée sur la plupart des graphiques (pas ceux produits avec `ggplot2`).

#### 2.5.2 Utiliser un fond de carte OSM

La fonction `getTiles()` permet de télécharger des fonds de cartes OSM et la fonction `tilesLayer()` permet de les afficher.

```
type <- c( "osm", "hotstyle", "hikebike", "osmgrayscale", "stamenbw",
         "stamenwatercolor", "cartodark", "cartolight")
par(mar = c(0,0,0,0), mfrow = c(3,3))
for (i in type){
  tilesLayer(getTiles(x = mtq, type = i, crop=TRUE))
```

```
mtext(side = 3, line = -1.5, text = i, col="red")
}
```



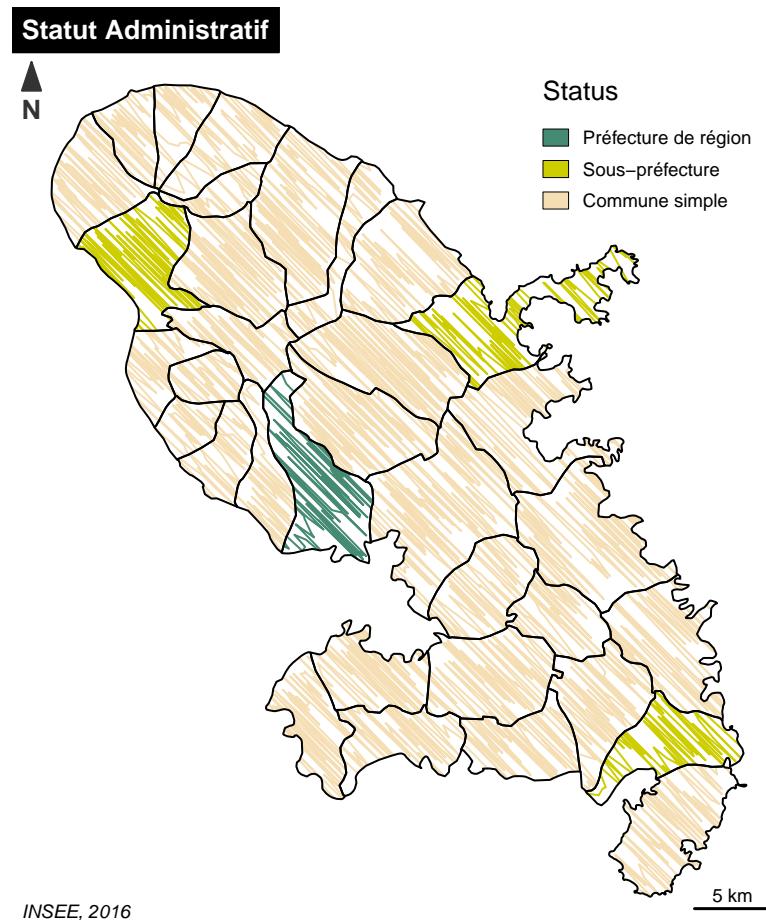
### 2.5.3 Créer un effet crayonné

```
library(sf)
mtq_pencil <- getPencilLayer(x = mtq)
typoLayer(
  x = mtq_pencil,
  var="STATUT",
  col = c("aquamarine4", "yellow3","wheat"),
  legend.values.order = c("Préfecture de région",
                        "Sous-préfecture",
                        "Commune simple"),
  legend.pos = "topright",
```

```

    legend.title.txt = "Status"
)
plot(st_geometry(mtq), add = TRUE, ldy=2)
layoutLayer(title = "Statut Administratif", tabtitle=TRUE,
            author= "INSEE, 2016", sources="",
            frame=FALSE, scale = 5)
north(pos = "topleft")

```

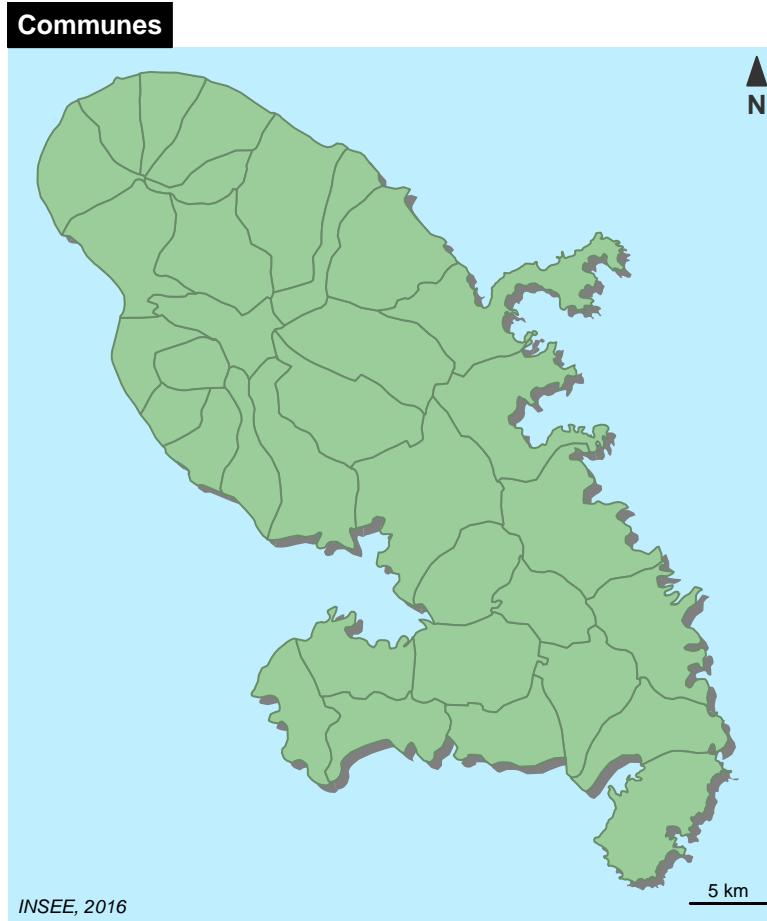


#### 2.5.4 Ajouter un ombrage à une couche

```

plot(st_geometry(mtq) + c(500, -500),
      col = "grey50", border = NA, bg = "lightblue1")
plot(st_geometry(mtq), col="darkseagreen3", border="darkseagreen4", add=TRUE)
layoutLayer(title = "Communes", tabtitle=TRUE,
            author= "INSEE, 2016", sources="", north=TRUE,
            frame=FALSE, scale = 5)

```



### 2.5.5 Crédit de cartons

Le package `mapinsetr`(?) est dédié à la création de cartons cartographiques. Il n'est pas sur le CRAN pour l'instant, mais on peut l'installer via le package `remotes`.

```
remotes::install_github("riatelab/mapinsetr")
```

`mapinsetr` permet de découper, redimensionner et déplacer une zone d'un fond de carte.

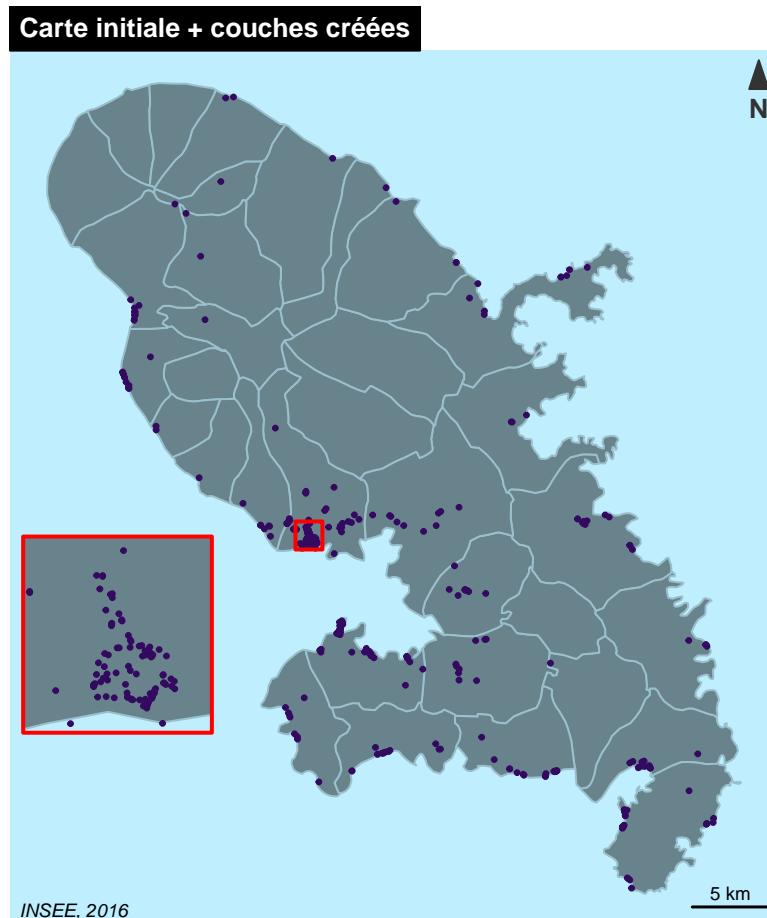
```
library(mapinsetr)
library(cartography)
library(sf)

mtq <- st_read("data/martinique.shp", quiet = TRUE)
resto <- st_read("data/resto.gpkg", quiet = TRUE)
# Création d'un masque
box_FDF <- create_mask(bb = c(706880, 1615030, 708650, 1616870),
                        prj = st_crs(mtq))
# Découpage, déplacement et redimensionnement des couches sous le masque
zbox_FDF <- move_and_resize(
  x = box_FDF,
  mask = box_FDF,
  xy = c(689000, 1603000),
  k = 7
)
```

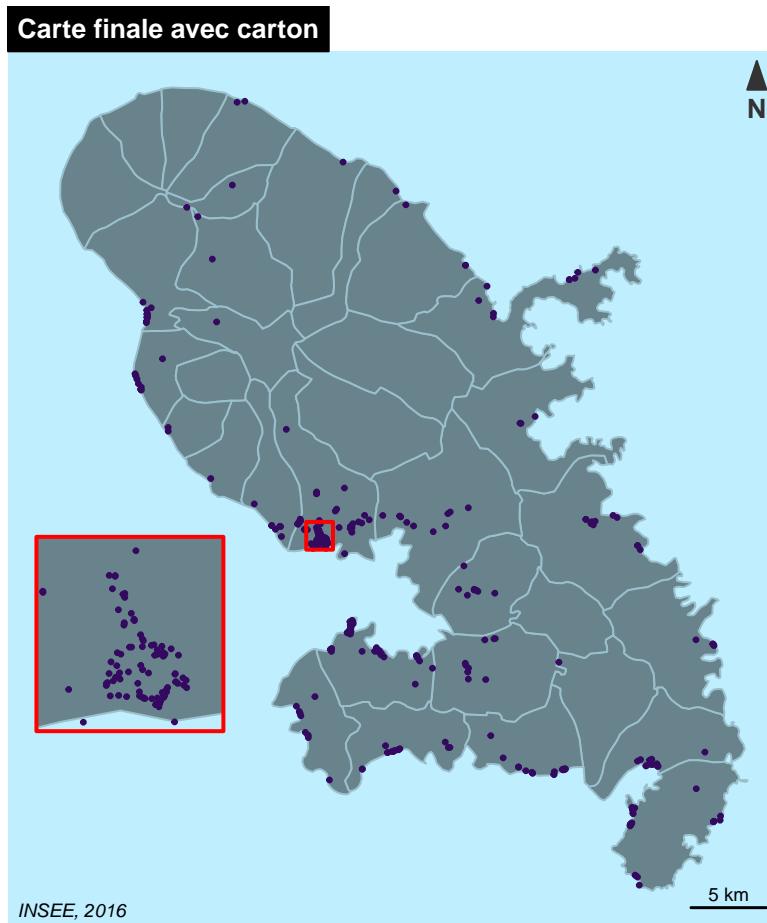
```

zmtq_FDF <- move_and_resize(
  x = mtq,
  mask = box_FDF,
  xy = c(689000, 1603000),
  k = 7
)
zresto_FDF <- move_and_resize(
  x = resto,
  mask = box_FDF,
  xy = c(689000, 1603000),
  k = 7
)
# Affichage de la carte et des couches créées
plot(st_geometry(mtq), col = "lightblue4", border = "lightblue3",
      bg = "lightblue1")
plot(st_geometry(resto), add=T, pch=20, col = "#330A5FFF", cex = 0.5)
plot(st_geometry(box_FDF), border = "red", add = T, lwd = 2)
plot(st_geometry(zmtq_FDF), col = "lightblue4", border = "lightblue3", add=TRUE)
plot(st_geometry(zresto_FDF), add=TRUE, pch=20, col = "#330A5FFF", cex = 0.5)
plot(st_geometry(zbox_FDF), border = "red", add = T, lwd = 2)
layoutLayer(title = "Carte initiale + couches créées", tabtitle=TRUE,
            author= "INSEE, 2016", sources="", north=TRUE,
            frame=FALSE, scale = 5)

```



```
# Cr ation de couches uniques comprenant le zoom
resto <- inset_rbinder(l = list(resto, zresto_FDF))
mtq <- inset_rbinder(l = list(mtq, zmtq_FDF))
box <- inset_rbinder(l = list(box_FDF, zbox_FDF))
plot(st_geometry(mtq), col = "lightblue4", border = "lightblue3",
     bg = "lightblue1")
plot(st_geometry(resto), add=T, pch=20, col = "#330A5FFF", cex = 0.5)
plot(st_geometry(box), border = "red", add = T, lwd = 2)
layoutLayer(title = "Carte finale avec carton", tabtitle=TRUE,
            author= "INSEE, 2016", sources="", north=TRUE,
            frame=FALSE, scale = 5)
```



# Chapter 3

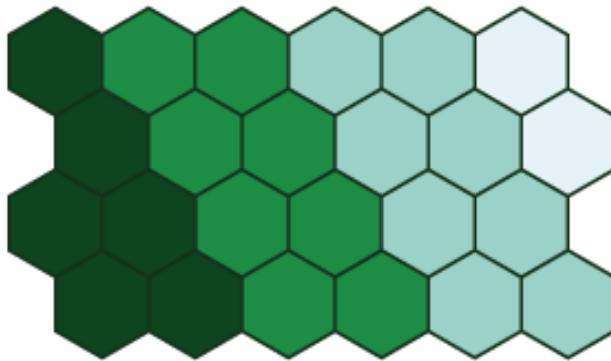
## Cartographie thématique avancée

### 3.1 Les cartes de discontinuités

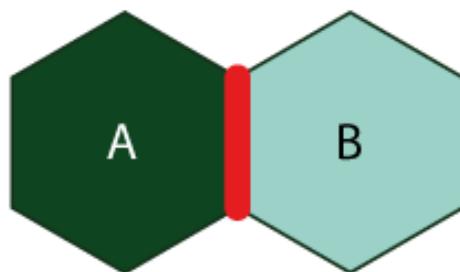
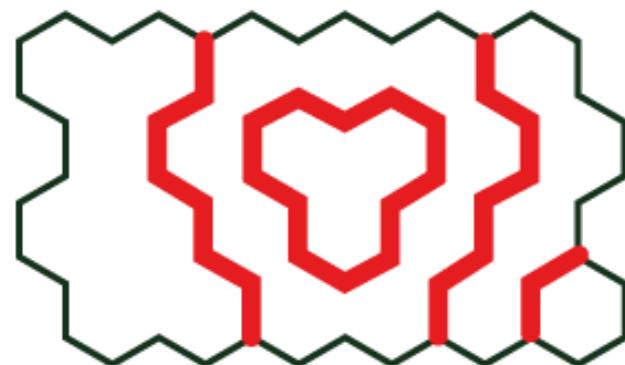
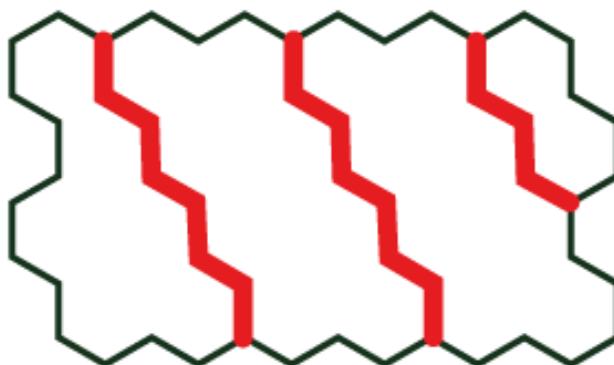
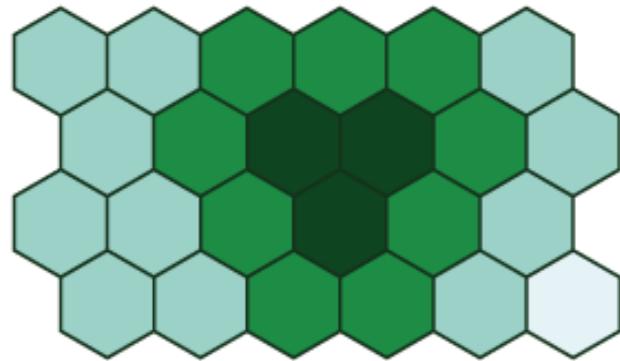
Ce type de représentation permet de souligner cartographiquement les discontinuités territoriales d'un phénomène. L'accent est porté sur ce qui distingue des territoires. Pour chaque frontière nous calculons le rapport ou la différence des valeurs des polygones de part et d'autre. Puis nous représentons la frontière par un figuré d'autant plus épais que la différence est forte. Il est souvent bénéfique de coupler ce type de représentation à une représentation choroplèthe (pour comprendre le sens des discontinuités).

## Les discontinuités

Gradation régulière



Centre-péphérie



**Discontinuité absolue**  
 $\text{disc}(A,B) = \max(A,B) - \min(A,B)$



