Espace et Temps avec R

1 - Manipulation de données spatiales

Robin Cura, Hélène Mathian & Lise Vaudor

16/10/2018

École Thématique GeoViz 2018

Sommaire

Les données spatiales dans R

Deux formats de données spatiales coexistent :

- `sp`, le format le plus ancien et répandu
- `sf`, un format plus récent, puissant, mais pas encore universel

```
library(rgdal)
library(sp)
iris_Paris_sp <- readOGR("data/ParisIris2017.shp",
                         stringsAsFactors = FALSE)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/data/user/c/rcura/geoviz2018/data/ParisIris2017.shp", layer: "ParisIris2017"
## with 992 features
## It has 77 fields
```

```
iris_Paris_sp                Large SpatialPolygonsDataFrame (987 elements, 3.3 Mb)
  ..@ data :'data.frame': 987 obs. of 5 variables:
  ..  ..$ Dept : chr [1:987] "75" "75" "75" "75" ...
  ..  ..$ DepCom : chr [1:987] "75101" "75101" "75101" "75101" ...
  ..  ..$ Nom_Com : chr [1:987] "PARIS 1ER" "PARIS 1ER" "PARIS 1ER" "PARIS 1ER" ...
  ..  ..$ Iris : chr [1:987] "0101" "0102" "0103" "0104" ...
  ..  ..$ DComIris: chr [1:987] "751010101" "751010102" "751010103" "751010104" ...
  ..@ polygons :List of 987
  ..  ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  ..  ..  ..  ..@ Polygons :List of 1
  ..  ..  ..  ..  ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  ..  ..  ..  ..  ..  ..  ..@ labpt : num [1:2] 651976 6862246
  ..  ..  ..  ..  ..  ..  ..@ area : num 64278
  ..  ..  ..  ..  ..  ..  ..@ hole : logi FALSE
  ..  ..  ..  ..  ..  ..  ..@ ringDir: int 1
  ..  ..  ..  ..  ..  ..  ..@ coords : num [1:21, 1:2] 652145 652121 652096 652078 651998 ...
  ..  ..  ..  ..@ plotOrder: int 1
  ..  ..  ..  ..@ labpt : num [1:2] 651976 6862246
  ..  ..  ..  ..@ ID : chr "0"
  ..  ..  ..  ..@ area : num 64278
  ..  ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  ..  ..  ..  ..@ Polygons :List of 1
  ..  ..  ..  ..  ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  ..  ..  ..  ..  ..  ..  ..@ labpt : num [1:2] 651867 6861957
```

```
library(sf)
iris_Paris_sf <- st_read("data/ParisIris2017.shp",
                         stringsAsFactors = FALSE)
```

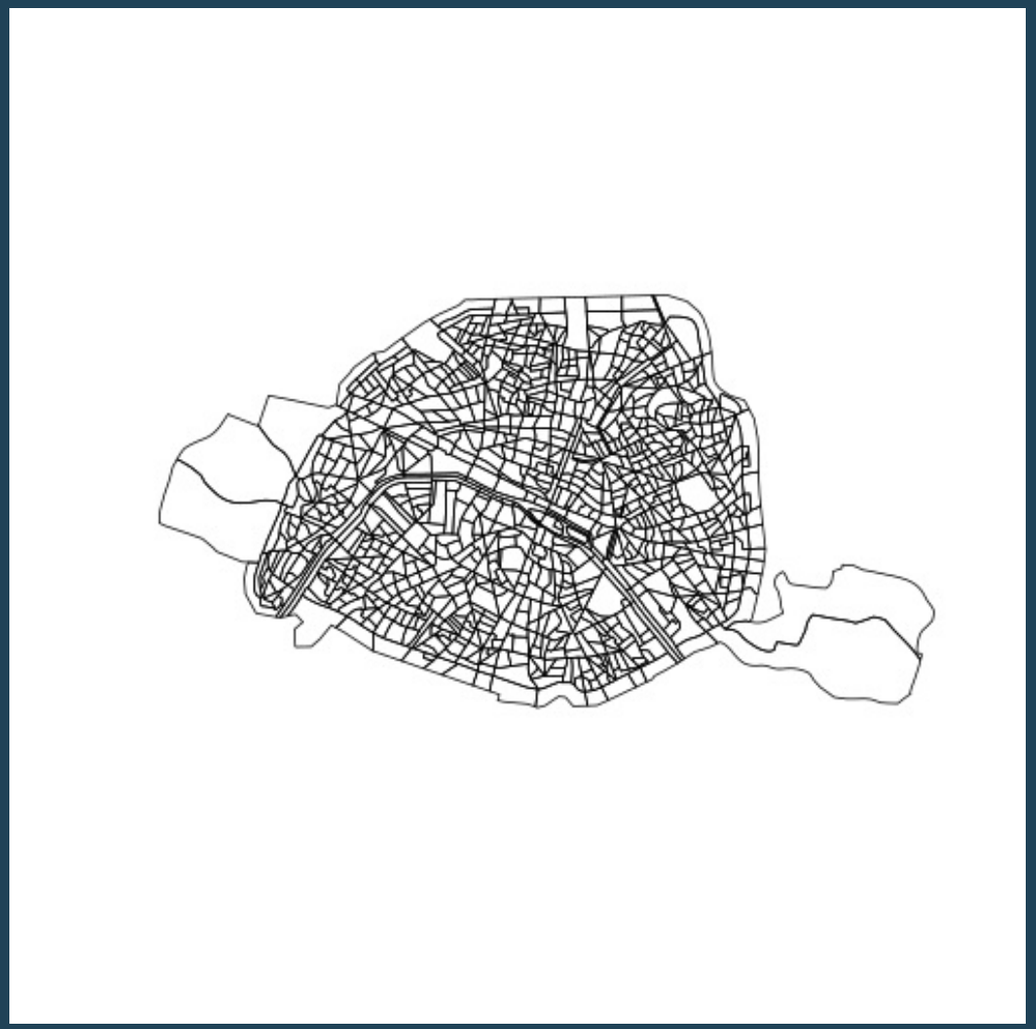
```
## Reading layer `ParisIris2017' from data source `/data/user/c/rcura/geoviz201
## Simple feature collection with 992 features and 77 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 643075.6 ymin: 6857477 xmax: 661086.2 ymax: 6867081
## epsg (SRID):    NA
## proj4string:    +proj=lcc +lat_1=44 +lat_2=49 +lat_0=46.5 +lon_0=3 +x_0=700
```

```
str(iris_Paris_sf)
```

```
## Classes 'sf' and 'data.frame':    992 obs. of  78 variables:
##  $ INSEE     : chr  "75119" "75117" "75112" "75110" ...
##  $ NOM_C     : chr  "Paris 19e Arrondissement" "Paris 17e Arrondissement" "P
##  $ IRIS      : chr  "7316" "6716" "4502" "3703" ...
##  $ CODE_     : chr  "751197316" "751176716" "751124502" "751103703" ...
##  $ NOM_I     : chr  "Villette 16" "Batignolles 16" "Bel Air 2" "Saint-Vincer
##  $ TYP_IRIS_x: chr  "H" "H" "H" "H" ...
##  $ REG       : chr  "11" "11" "11" "11" ...
##  $ REG20     : chr  "11" "11" "11" "11" ...
##  $ DEP       : chr  "75" "75" "75" "75" ...
##  $ UU201     : chr  "00851" "00851" "00851" "00851" ...
##  $ COM_x     : chr  "75119" "75117" "75112" "75110" ...
##  $ LIBCOM_x  : chr  "Paris 19e Arrondissement" "Paris 17e Arrondissement" "P
##  $ TRIRI     : chr  "752471" "752011" "750921" "750631" ...
##  $ GRD_Q     : chr  "7511973" "7511767" "7511245" "7511037" ...
##  $ LIBIRIS_x : chr  "Villette 16" "Batignolles 16" "Bel Air 2" "Saint-Vincer
##  $ TYP_IRIS_y: chr  "H" "H" "H" "H" ...
##  $ MODIF     : chr  "00" "00" "00" "00" ...
##  $ LAB_I     : chr  "1" "2" "1" "1" ...
##  $ P12_POP156: num  2773 1608 3299 2073 1828 ...
##  $ P12_POP152: num  404 339 473 233 223 ...
##  $ P12_POP2  : num  2028 999 2439 1556 1337 ...
##  $ P12_POP5  : num  341 270 386 284 268 ...
##  $ P12_ACTOCC: num  1896 1044 2442 1634 1269 ...
```

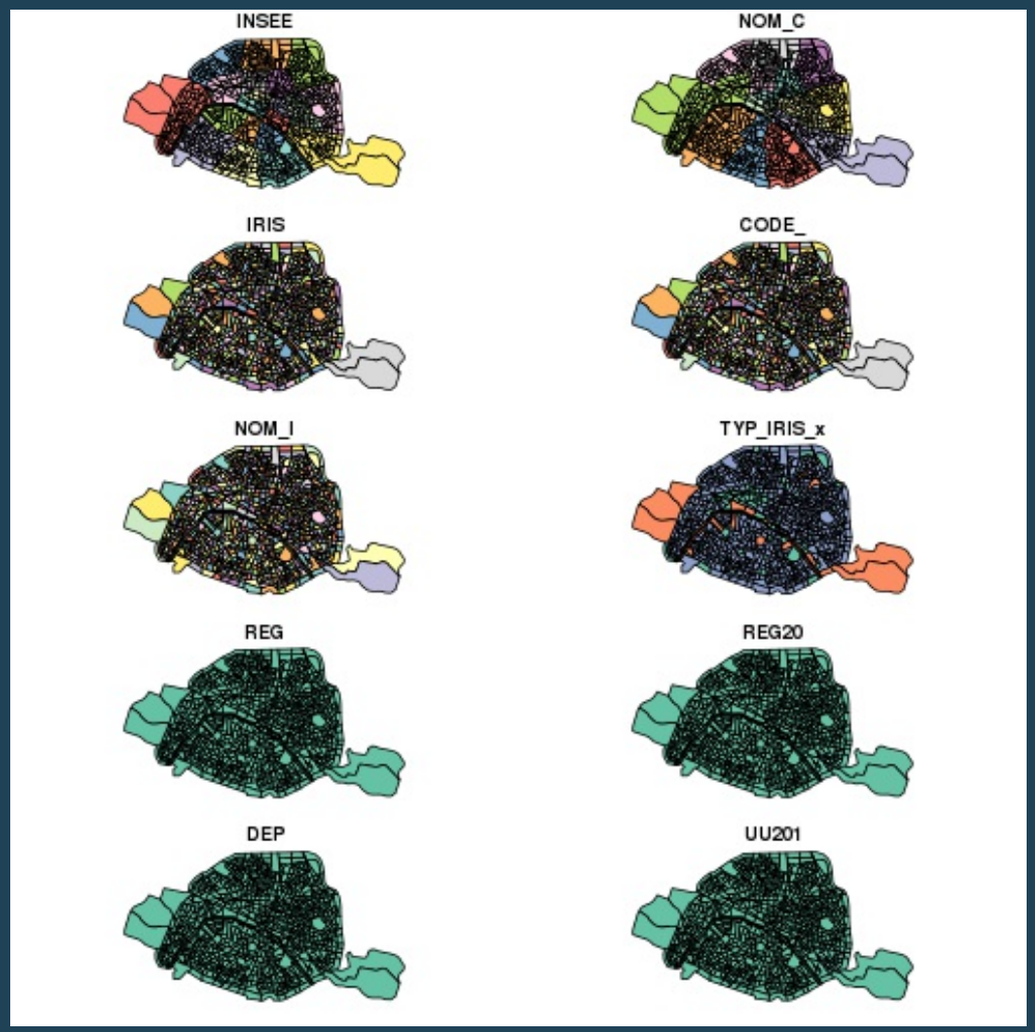
Les données spatiales dans R

```
plot(iris_Paris_sp)
```



```
plot(iris_Paris_sf)
```

```
## Warning: plotting the first 10 out of 77 attributes; use max.plot = 77 to
## plot all
```

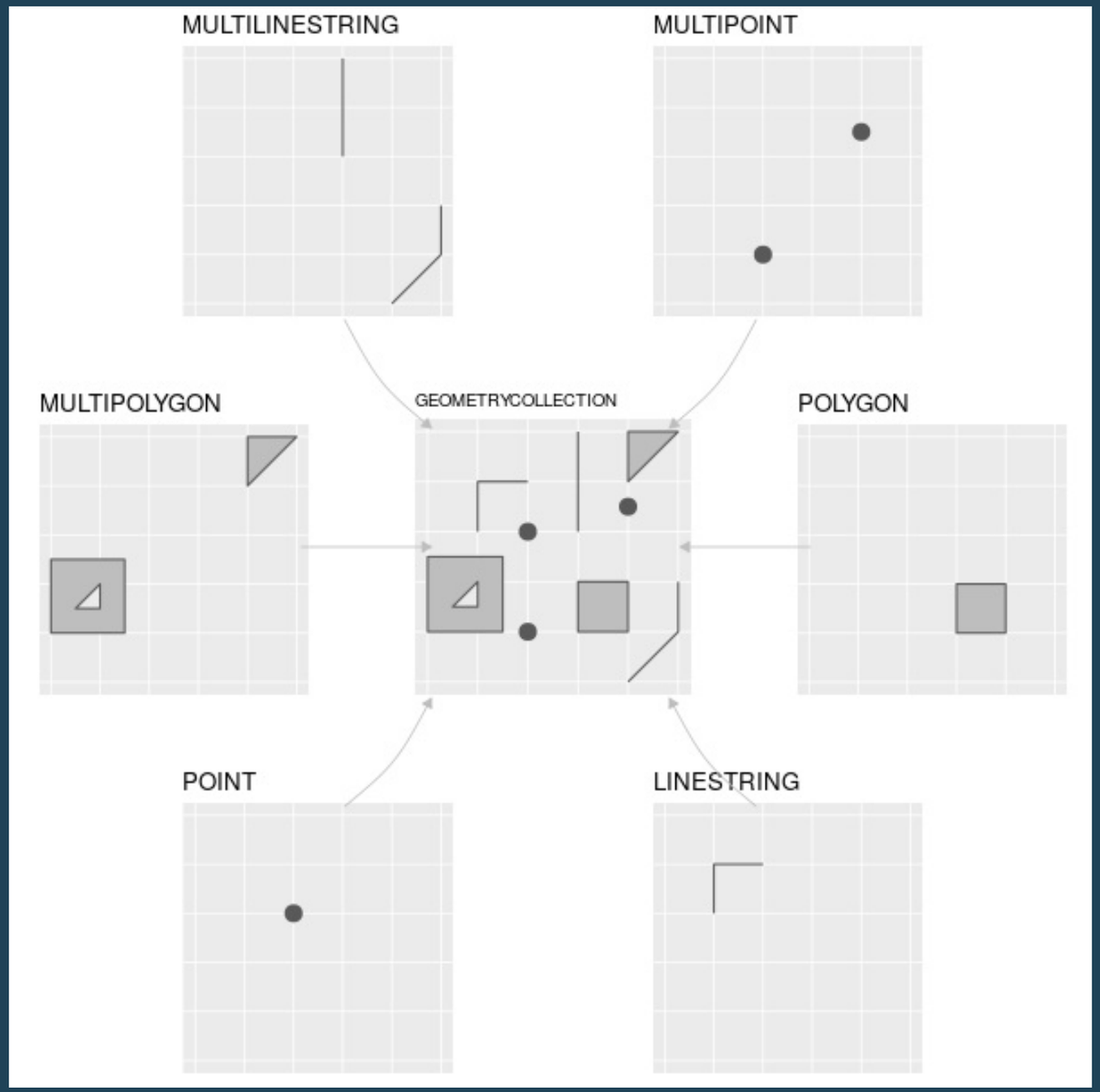


Les données spatiales dans R

On va se contenter de mobiliser le `sf`, qui permet une manipulation d'objets simples, semblables à des `data.frame`/`tibbles`.

N.B. : Pour une très grande majorité d'opérations, `sf` suffit. Il peut toutefois être nécessaire de passer de `sf` à `sp` dès lors qu'on souhaite utiliser des fonctions spatiales avancées, par exemple pour mobiliser des méthodes d'analyse spatiale.

sf : Les Simples Features



R. Lovelace, J. Nowosad & J. Muenchow (2018), Geocomputation with R - https://geocompr.robinlovelace.net/

sf : Structure d'un objet sf

```
iris_Paris_sf
```

```
## Simple feature collection with 992 features and 77 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 643075.6 ymin: 6857477 xmax: 661086.2 ymax: 6867081
## epsg (SRID):    NA
## proj4string:    +proj=lcc +lat_1=44 +lat_2=49 +lat_0=46.5 +lon_0=3 +x_0=700000 +y_0=6600000 +ellps=GRS80 +units
## First 10 features:
##    INSEE                   NOM_C IRIS     CODE_                  NOM_I
## 1  75119 Paris 19e Arrondissement 7316 751197316            Villette 16
## 2  75117 Paris 17e Arrondissement 6716 751176716          Batignolles 16
## 3  75112 Paris 12e Arrondissement 4502 751124502               Bel Air 2
## 4  75110 Paris 10e Arrondissement 3703 751103703 Saint-Vincent de Paul 3
## 5  75118 Paris 18e Arrondissement 7104 751187104           Goutte d'Or 4
## 6  75111 Paris 11e Arrondissement 4314 751114314             Roquette 14
## 7  75110 Paris 10e Arrondissement 3707 751103707 Saint-Vincent de Paul 7
## 8  75120 Paris 20e Arrondissement 7702 751207702            Belleville 2
## 9  75113 Paris 13e Arrondissement 5004 751135004                  Gare 4
## 10 75112 Paris 12e Arrondissement 4623 751124623              Picpus 23
##    TYP_IRIS_x REG REG20 DEP UU201 COM_x                 LIBCOM_x  TRIRI
## 1           H  11    11  75 00851 75119 Paris 19e Arrondissement 752471
## 2           H  11    11  75 00851 75117 Paris 17e Arrondissement 752011
## 3           H  11    11  75 00851 75112 Paris 12e Arrondissement 750921
## 4           H  11    11  75 00851 75110 Paris 10e Arrondissement 750631
## 5           H  11    11  75 00851 75118 Paris 18e Arrondissement 752241
## 6           H  11    11  75 00851 75111 Paris 11e Arrondissement 750761
## 7           H  11    11  75 00851 75110 Paris 10e Arrondissement 750521
## 8           H  11    11  75 00851 75120 Paris 20e Arrondissement 752671
## 9           H  11    11  75 00851 75113 Paris 13e Arrondissement 751171
## 10          H  11    11  75 00851 75112 Paris 12e Arrondissement 750871
##      GRD_Q                LIBIRIS_x TYP_IRIS_y MODIF LAB_I P12_POP156
## 1  7511973             Villette 16          H    00     1   2773.295
## 2  7511767           Batignolles 16          H    00     2   1607.938
## 3  7511245                Bel Air 2          H    00     1   3298.589
## 4  7511037 Saint-Vincent de Paul 3          H    00     1   2073.318
## 5  7511871           Goutte d'Or 4          H    00     1   1827.740
## 6  7511143             Roquette 14          H    00     1   2119.054
```

Lecture de fichiers géographiques

Toutes les fonctions de **sf** s'inspirent de la syntaxe des fonctions **PostGIS** : **st_OPERATION** :

- Lecture d'un shapefile

```
irisParis <- st_read(dsn = "data/ParisIris2017.shp",
                     stringsAsFactors = FALSE) # Comme pour la lecture des data.frame
```

```
## Reading layer `ParisIris2017' from data source `/data/user/c/rcura/geoviz2018/data/ParisIris2017.shp' usin
## Simple feature collection with 992 features and 77 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 643075.6 ymin: 6857477 xmax: 661086.2 ymax: 6867081
## epsg (SRID):    NA
## proj4string:    +proj=lcc +lat_1=44 +lat_2=49 +lat_0=46.5 +lon_0=3 +x_0=700000 +y_0=6600000 +ellps=GRS80 +
```

Projections / transformations

Comme dans tout SIG, le système de projection (**`st_crs`**) est récuperé à la lecture d'un objet :

```
st_crs(irisParis)
```

```
## Coordinate Reference System:
##   No EPSG code
##   proj4string: "+proj=lcc +lat_1=44 +lat_2=49 +lat_0=46.5 +lon_0=3 +x_0=700000 +y_0=6600000 +ellps=GRS80 +units
```

Ici, le "code de projection" est bien reconnu, mais le code EPSG n'est pas renseigné.
On va le spécifier pour clarifier (même si ce n'est pas utile) :

```
irisParis <- irisParis %>%
  st_set_crs(2154) # SRID/EPSG du Lambert 93
```

```
## Warning: st_crs<- : replacing crs does not reproject data; use st_transform
## for that
```

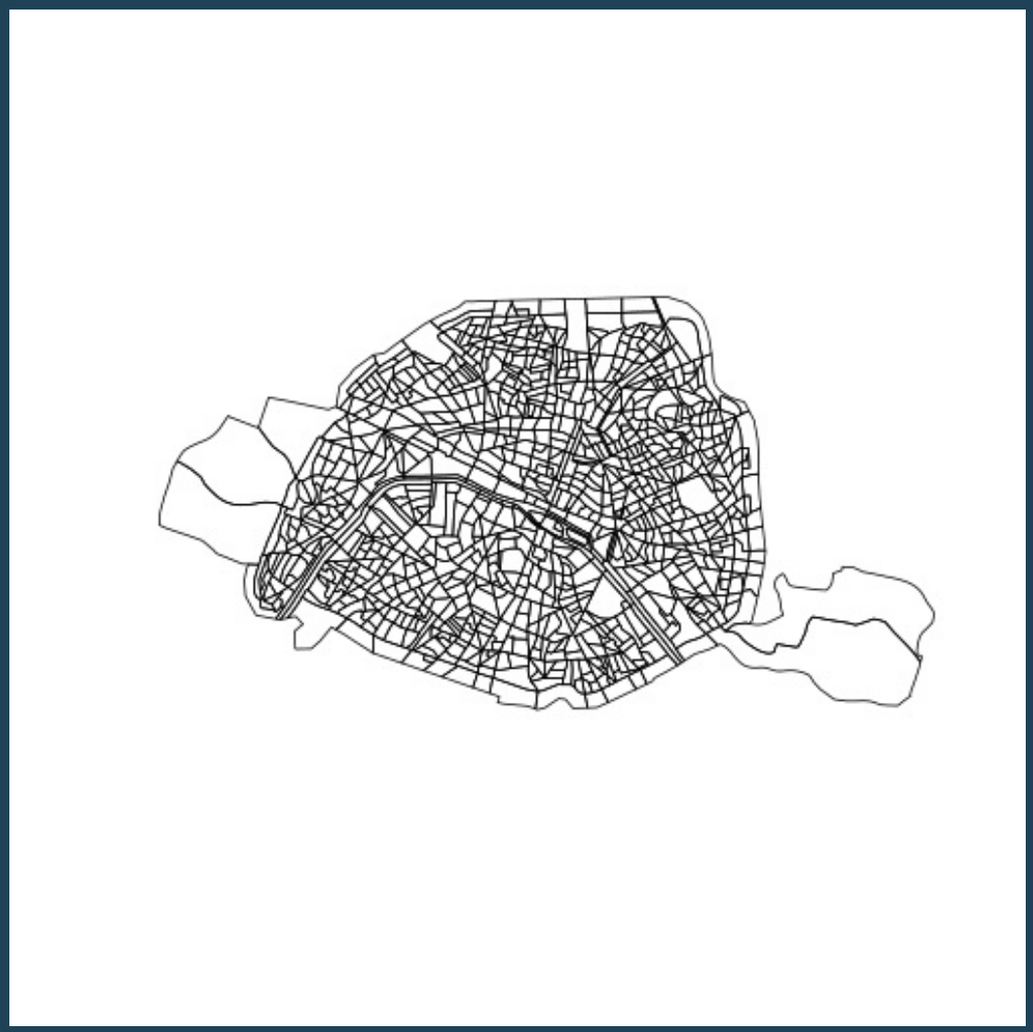
```
head(irisParis)
```

```
## Simple feature collection with 6 features and 77 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 648979.6 ymin: 6859518 xmax: 656589.8 ymax: 6866159
## epsg (SRID):    2154
## proj4string:    +proj=lcc +lat_1=49 +lat_2=44 +lat_0=46.5 +lon_0=3 +x_0=700000 +y_0=6600000 +ellps=GRS80 +towgs
##   INSEE                    NOM_C IRIS     CODE_                  NOM_I
## 1 75119 Paris 19e Arrondissement 7316 751197316             Villette 16
## 2 75117 Paris 17e Arrondissement 6716 751176716          Batignolles 16
## 3 75112 Paris 12e Arrondissement 4502 751124502               Bel Air 2
## 4 75110 Paris 10e Arrondissement 3703 751103703 Saint-Vincent de Paul 3
## 5 75118 Paris 18e Arrondissement 7104 751187104           Goutte d'Or 4
## 6 75111 Paris 11e Arrondissement 4314 751114314             Roquette 14
```

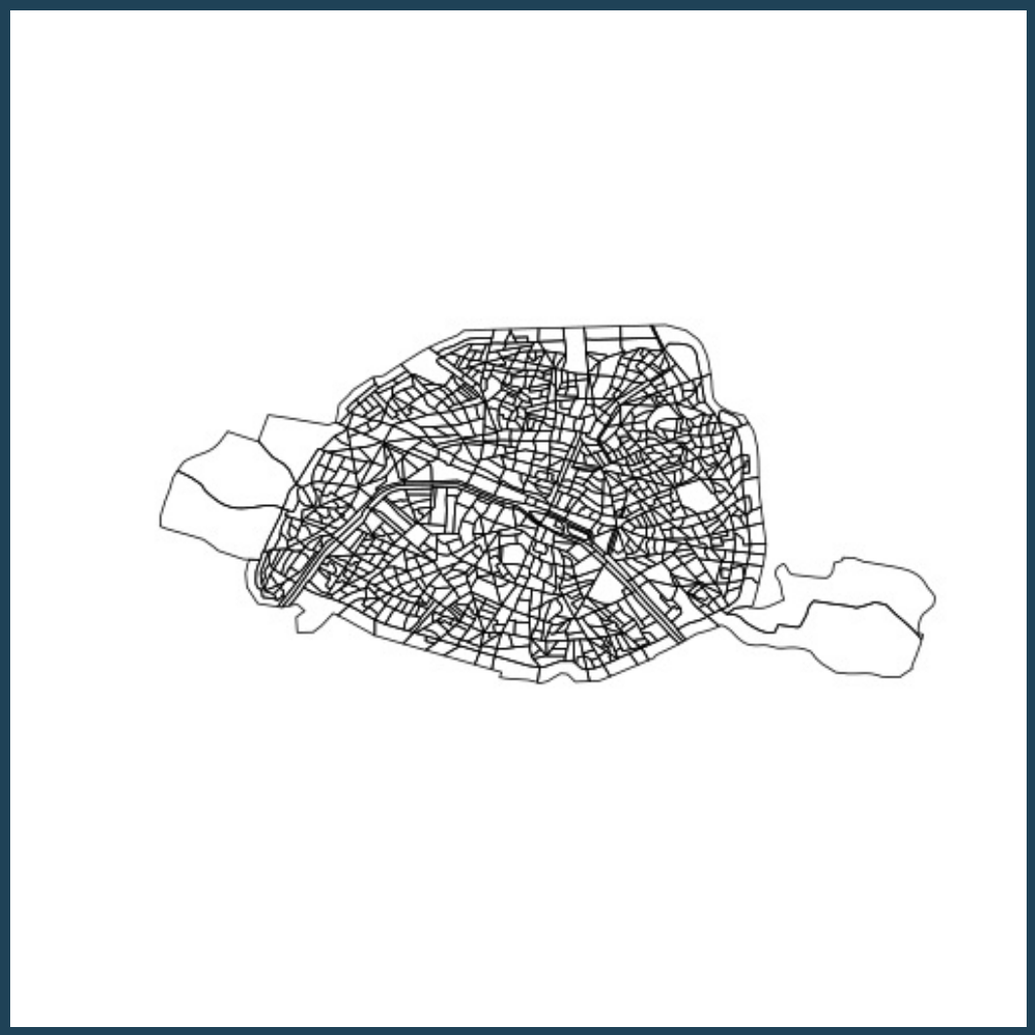
Projections / transformations

On peut re-projeter un objet **`sf`** avec la fonction **`st_transform(SRID/CRS)`** :

```
irisParis %>%
  select(geometry) %>% # Lambert 93
  plot()
```



```
GallPeters <- "+proj=cea +lon_0=0 +lat_ts=45 \
  +x_0=0 +y_0=0 +ellps=WGS84 +units=m +no_defs"
irisParis %>%
  select(geometry) %>%
  st_transform(crs = GallPeters) %>%
  plot()
```

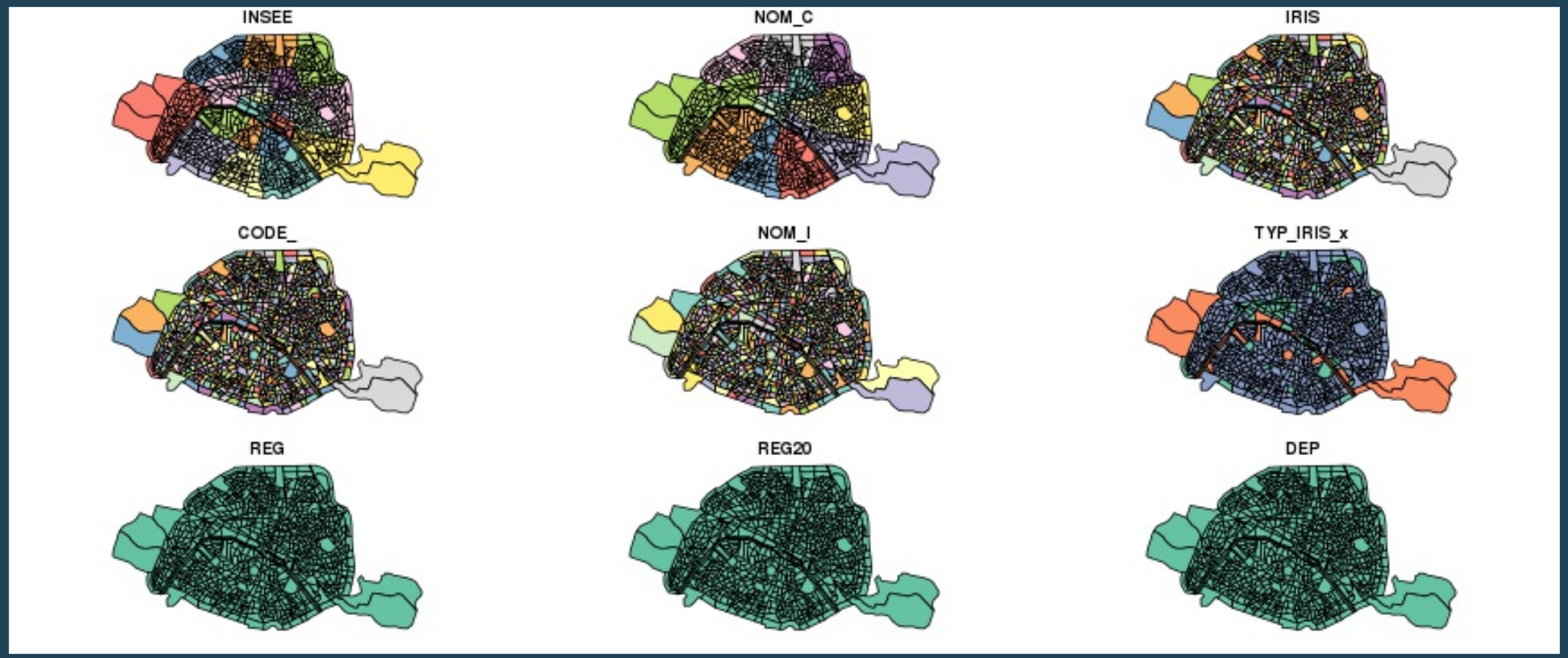


Visualisation exploratoire : `plot()`

On peut utiliser la fonction de base `plot()`, qui affiche alors l'ensemble des attributs du jeu de données :

```
plot(irisParis)
```

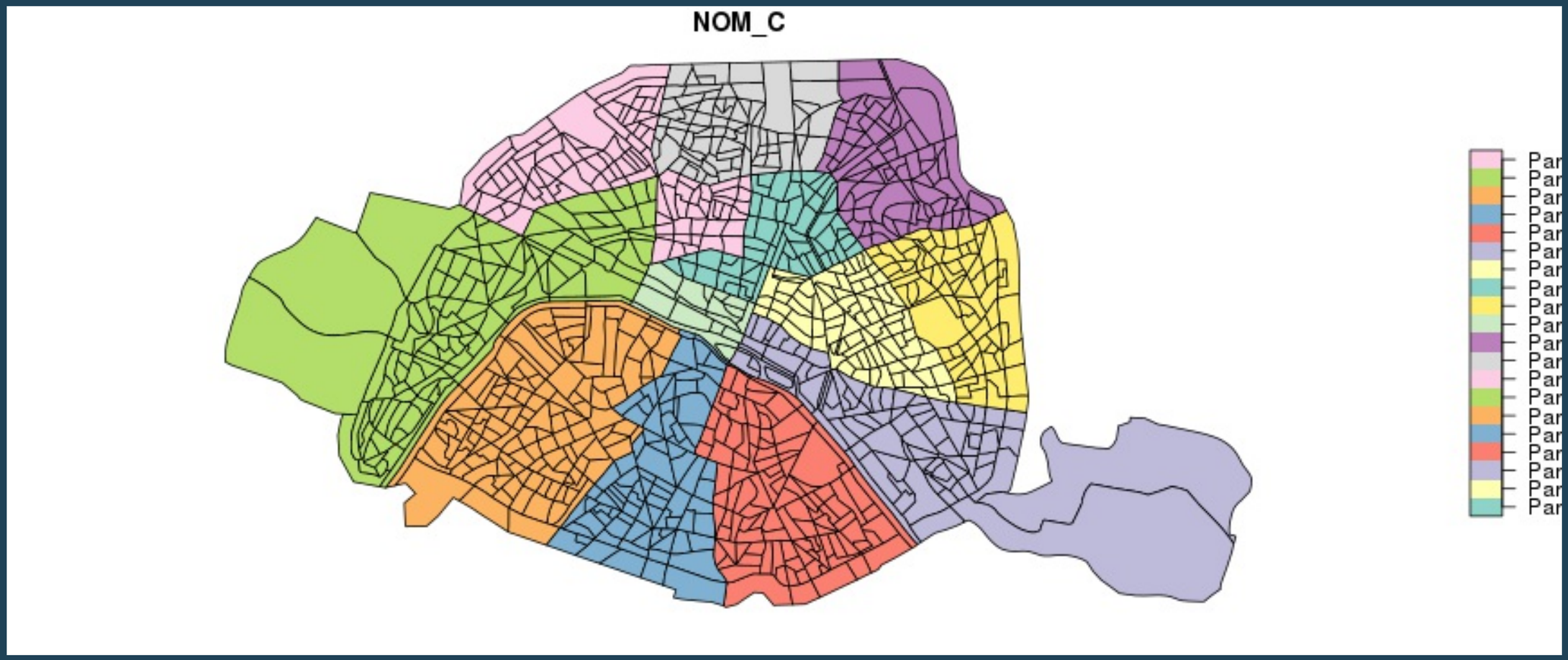
```
## Warning: plotting the first 9 out of 77 attributes; use max.plot = 77 to
## plot all
```



Visualisation exploratoire : `plot()`

Pour "cartographier" le contenu d'une seule variable :

```
plot(irisParis["NOM_C"])
```



Visualisation exploratoire : `mapview`

Le `mapview` permet de mener rapidement une exploration des données spatiales et attributaires :

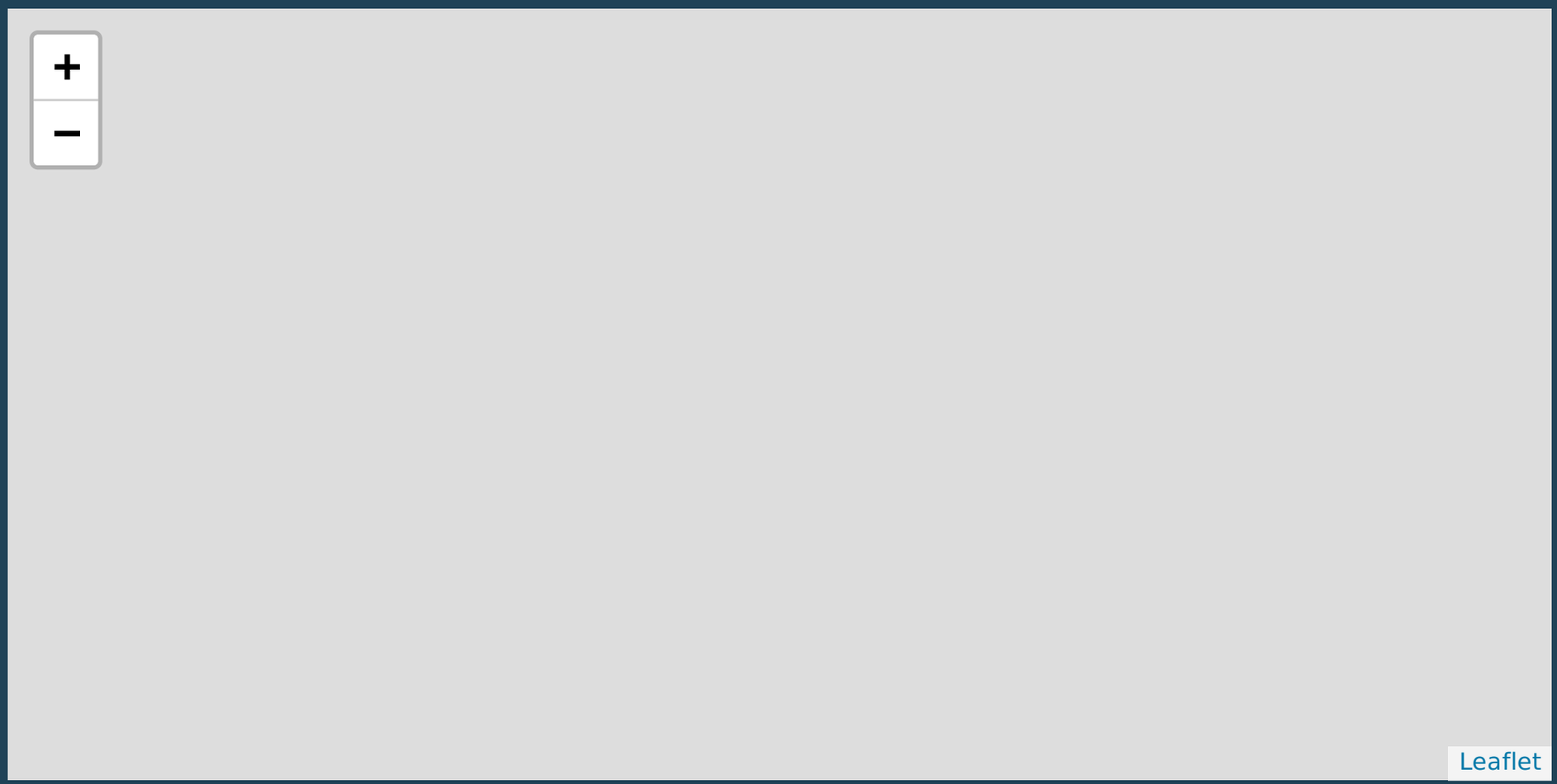
```
library(mapview) # install.packages("mapview")
mapview(irisParis)
```

Visualisation exploratoire : `mapview`

Le `mapview` permet de mener rapidement une exploration des données spatiales et attributaires.

On peut aussi spécifier un attribut à observer avec l'instruction `zcol` :

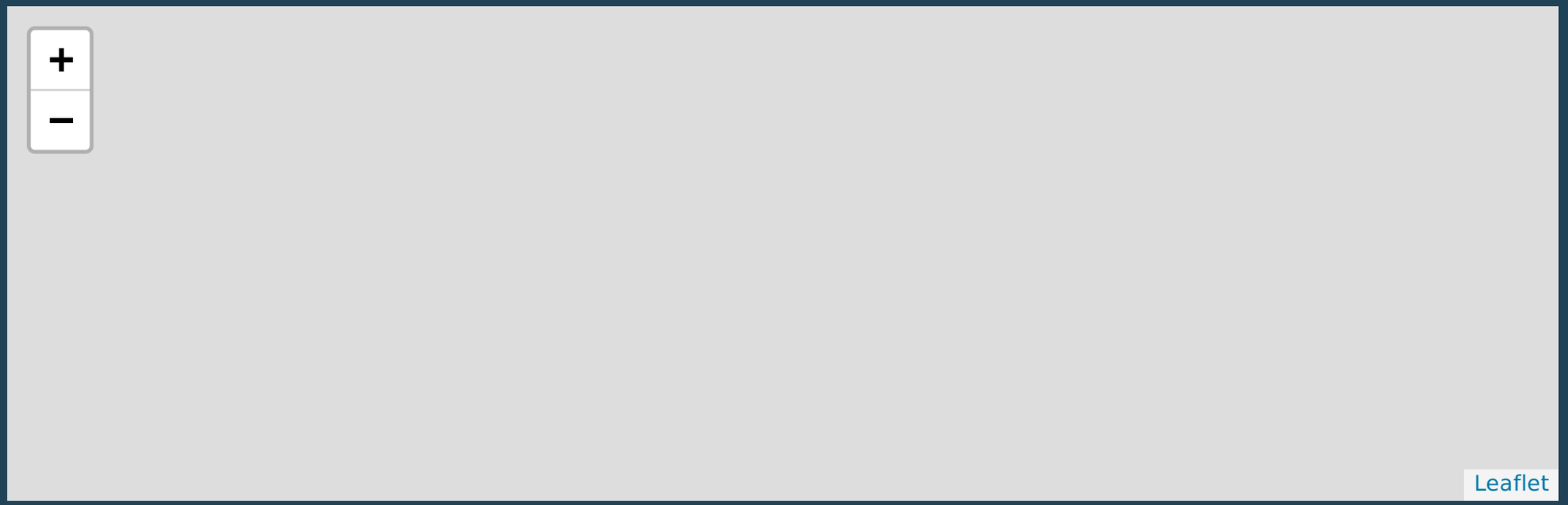
```
mapview(irisParis, zcol = "NOM_C")
```



Opérations attributaires

Comme l'objet `sf` est fondamentalement un `data.frame`, on peut lui appliquer les manipulations vues hier : filtrage (`filter`), réductions de variables (`select`), renommage de variables (`rename`) et création/modification de variables (`mutate`) :

```
irisParis %>%
  select(INSEE, CODE_) %>% # La sélection de la colonne `geometry` est implicite
  mutate(ARRONDISSEMENT = as.numeric(INSEE) - 100) %>%
  filter(ARRONDISSEMENT >= 75001, ARRONDISSEMENT <= 75007) %>%
  mapview(zcol = "ARRONDISSEMENT")
```



Agrégations

Comme avec un `tibble`, on peut réaliser des opérations d'agrégation. La géométrie est alors, elle-aussi, agrégée en conséquence :

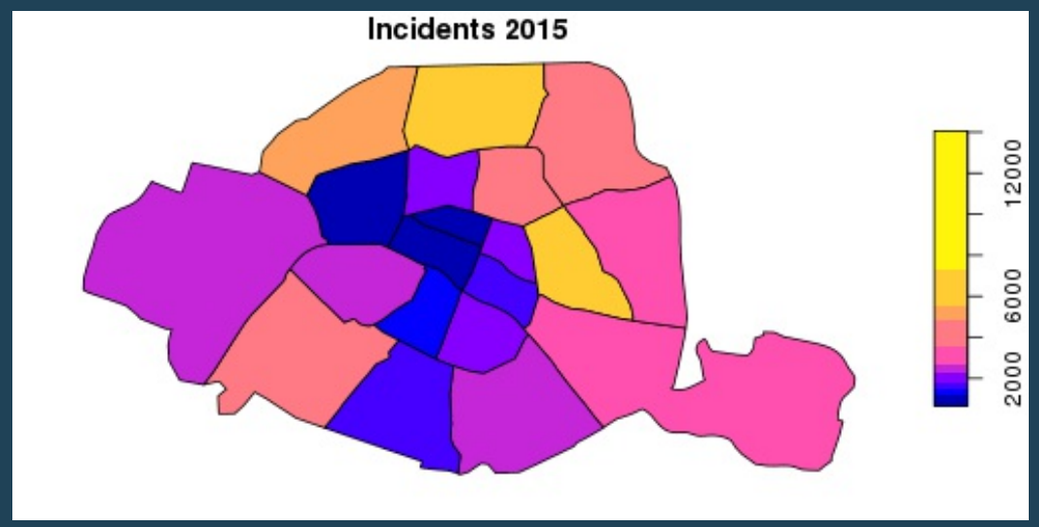
```
arrondissements_Paris <- irisParis %>%
  group_by(INSEE) %>%
  summarise()

plot(arrondissements_Paris)
```

Jointures attributaires

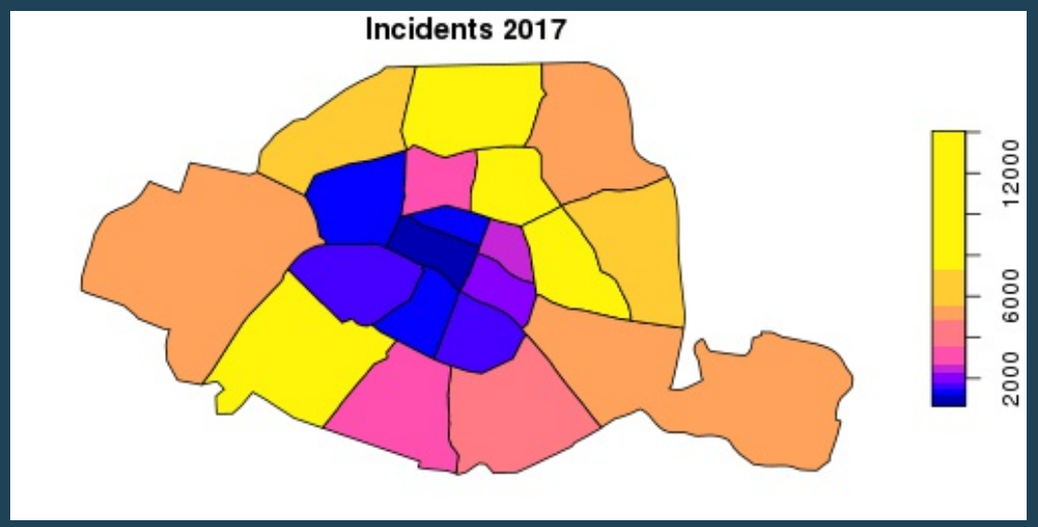
Comme pour tous les `data.frame`, on peut réaliser des jointures attributaires :

```
df_dmr  <- readRDS("dans_ma_rue_clean.RDS")
joinData <-  df_dmr %>%
  group_by(CODE_POSTAL, ANNEE_DECLARATION) %>%
  summarise(NbIncidents = n()) %>%
  ungroup()

donnees_incidents <- arrondissements_Paris %>%
  mutate(CODGEO = as.character(as.numeric(INSEE) - 100)) %>%
  left_join(joinData, by = c("CODGEO" = "CODE_POSTAL"))

incidents2015 <- filter(donnees_incidents, ANNEE_DECLARATION == 2015)
incidents2017 <- filter(donnees_incidents, ANNEE_DECLARATION == 2017)

commonBreaks <- joinData %>%
  filter(ANNEE_DECLARATION %in% c(2015, 2017)) %>%
  pull(NbIncidents) %>%
  quantile(probs = seq(from = 0, to = 1, by = 0.1))
```

```
plot(incidents2015[,"NbIncidents"],
     main = "Incidents 2015",
     breaks = commonBreaks)
```




```
plot(incidents2017[,"NbIncidents"],
     main = "Incidents 2017",
     breaks = commonBreaks)
```



Conversion en `sf` et jointures spatiales

Les objets `sf` sont des objets spatiaux, on peut donc aussi effectuer des jointures spatiales (entre deux objets `sf`) :

- Conversion du jeu de données "Dans ma Rue" en objet spatial avec la fonction `st_as_sf` :

```
dmr_spatial <- df_dmr %>%
  filter(ANNEE_DECLARATION %in% c(2015, 2017)) %>%
  st_as_sf(coords = c("Long", "Lat"), crs = 4326) %>%
  st_transform(2154) # Pour une opération géométrique, les objets doivent avoir le même SRID
```

Jointures spatiales

- Jointure spatiale pour récuperer les IRIS contenant les points :

```
dmr_augmente <- dmr_spatial %>%
  st_join(irisParis %>% select(INSEE), join = st_within)
```

```
dmr_augmente %>%
  group_by(INSEE, ANNEE_DECLARATION) %>%
  summarise(NbIncidents = n()) %>%
  ungroup() %>%
  head()
```

```
## Simple feature collection with 6 features and 3 fields
## geometry type:  MULTIPOINT
## dimension:      XY
## bbox:           xmin: 650249.7 ymin: 6861822 xmax: 653644 ymax: 6863729
## epsg (SRID):    2154
## proj4string:    +proj=lcc +lat_1=49 +lat_2=44 +lat_0=46.5 +lon_0=3 +x_0=7000
## # A tibble: 6 x 4
##   INSEE ANNEE_DECLARATION NbIncidents                          geometry
##   <chr>             <int>       <int>                <MULTIPOINT [m]>
## 1 75101              2015         653 (650305.8 6862792, 650334.4 6862783…
## 2 75101              2017         931 (650249.7 6862872, 650305.9 6862792…
## 3 75102              2015        1028 (650926.9 6863572, 650981.8 6863589…
## 4 75102              2017        1492 (650871.4 6863497, 650915.6 6863514…
## 5 75103              2015        2316 (652345.8 6862636, 652346.3 6862627…
## 6 75103              2017        2466 (652343.7 6862637, 652343.9 6862646…
```

Opérations spatiales

name: operations-spatiales

`sf` permet de réaliser la quasi-totalité des opérations attendues d'un SIG, avec des fonctions dédiées reprenant le vocabulaire classique (inspiré par `PostGIS`) :

`st_agr`, `st_area`, `st_bbox`, `st_bind_cols`, `st_boundary`, `st_buffer`, `st_cast`, `st_centroid`, `st_combine`, `st_contains`, `st_contains_properly`, `st_convex_hull`, `st_covered_by`, `st_covers`, `st_crop`, `st_crosses`, `st_difference`, `st_disjoint`, `st_distance`, `st_equals`, `st_equals_exact`, `st_graticule`, `st_interpolate_aw`, `st_intersection`, `st_intersects`, `st_is_simple`, `st_is_valid`, `st_is_within_distance`, `st_jitter`, `st_join`, `st_layers`, `st_length`, `st_line_merge`, `st_line_sample`, `st_make_grid`, `st_overlaps`, `st_point_on_surface`, `st_polygonize`, `st_relate`, `st_sample`, `st_segmentize`, `st_simplify`, `st_snap`, `st_sym_difference`, `st_touches`, `st_transform`, `st_triangulate`, `st_union`, `st_viewport`, `st_voronoi`, `st_within`, `st_wrap_dateline`, `st_write`, `st_write_db`, `st_zm`, `write_sf`

Voir les exemples illustrées dans les du `sf` : Manipulating Simple Feature Geometries et Manipulating Simple Features

Opérations spatiales : un exemple de traitement

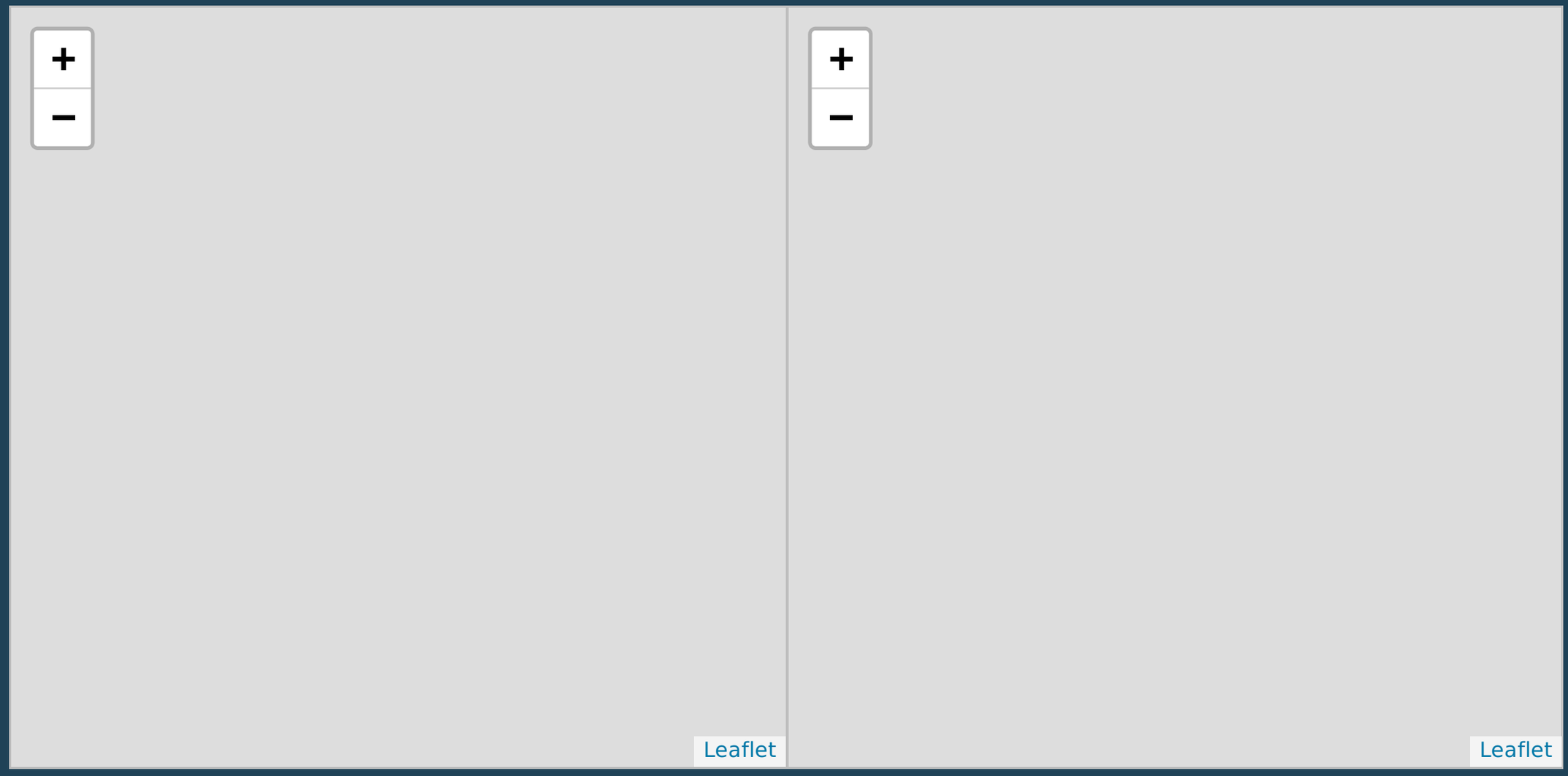
-> On peut chercher à créer une carte des densités d'incidents déclarés par IRIS, potentiellement selon les années :

```
nbIncidentsIris <- dmr_spatial %>%
  st_join(irisParis %>% select(INSEE), join = st_within) %>%
  st_set_geometry(NULL) %>% # On n'a plus besoin que ce soit un objet spatial
  group_by(INSEE, ANNEE_DECLARATION) %>%
  summarise(NbIncidents = n()) %>%
  ungroup()

irisParis_Incidents <- irisParis %>%
  mutate(surface = st_area(.)) %>%
  left_join(nbIncidentsIris, by = "INSEE")

densite_incidents <- irisParis_Incidents %>%
  mutate(densiteIncidents = NbIncidents / (surface / 1E6)) %>% # Pour avoir une densité / km²
  select(INSEE, CODE_, ANNEE_DECLARATION, NbIncidents, densiteIncidents)

# Les densités sont exprimées avec des "unités", qui complexifient le traitement.
# On les enlève donc :

densite_incidents <- densite_incidents %>%
  mutate(densiteIncidents = units::drop_units(densiteIncidents))
```

Cartographie exploratoire : mapview

```
densite2015 <- filter(densite_incidents, ANNEE_DECLARATION == 2015)
densite2017 <- filter(densite_incidents, ANNEE_DECLARATION == 2017)

map2015 <- mapview(densite2015, zcol = "densiteIncidents", legend = TRUE)
map2017 <- mapview(densite2017, zcol = "densiteIncidents", legend = TRUE)

latticeView(map2013, map2017, ncol = 2)
```

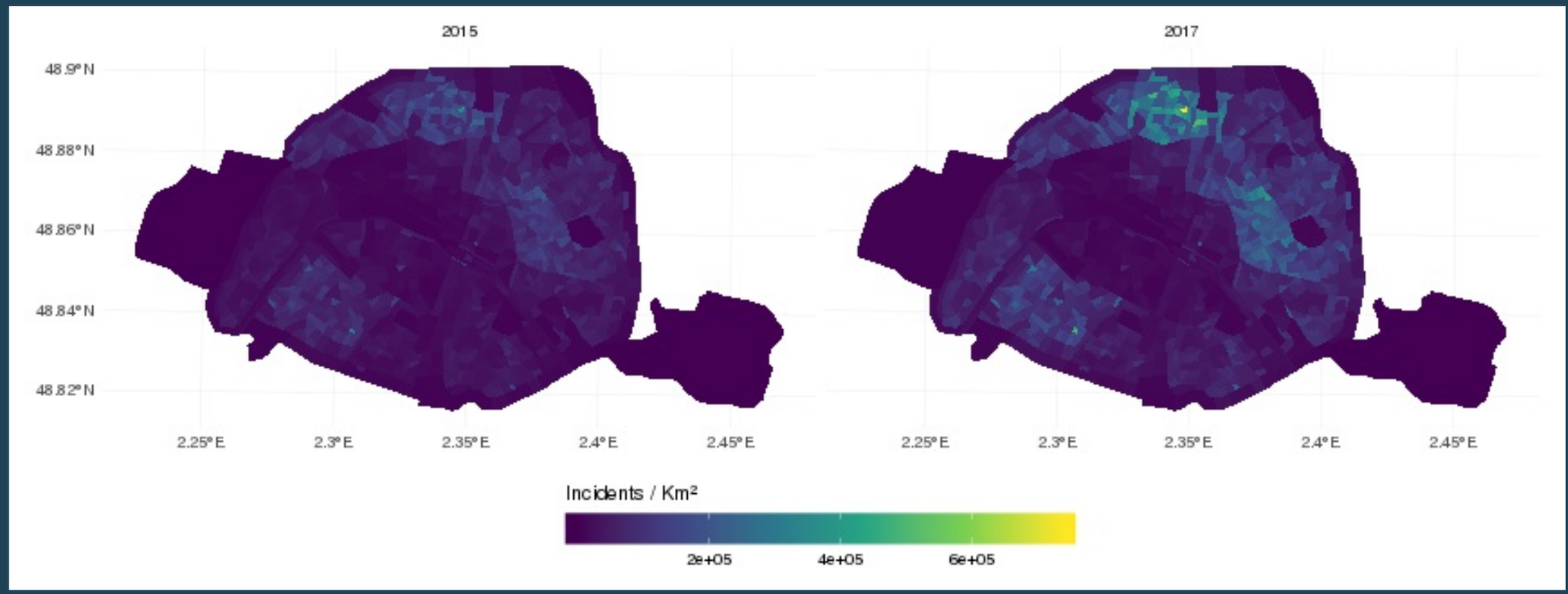
Cartographie exploratoire : `mapview`



Cartographie statique : ggplot2

On peut bien sûr afficher ces éléments avec le `ggplot2`, en faisant appel à la géométrie dédiée aux objets de type `sf` : `geom_sf`

```
ggplot(densite_incidents) +
  geom_sf(data = densite_incidents, aes(fill = densiteIncidents), lwd = 0) +
  # On est obligé de remettre la couche sf en data
  facet_wrap(~ANNEE_DECLARATION) +
  scale_fill_viridis_c(name = "Incidents / Km²") +
  guides(fill = guide_colourbar(title.position = "top")) +
  coord_sf() + # On s'assure que les coordonnées du graphique respectent le CRS
  theme_minimal() +
  theme(legend.position = "bottom", legend.key.width = unit(2,"cm"))
```



Cartographie statique : ggplot2

- Certains (`ggsn`, `ggmap`) proposent des foncitonnalités permettant d'améliorer le rendu cartographique (échelles graphiques, flèches d'orientations, fonds de cartes...) ;

- mais `ggplot` est avant tout un outil de visualisation générique :

- pour créer des cartographies de qualité, mieux vaut se tourner vers des dédiés : **`tmap`** (qu'on ne verra pas ici) et **`cartography`**

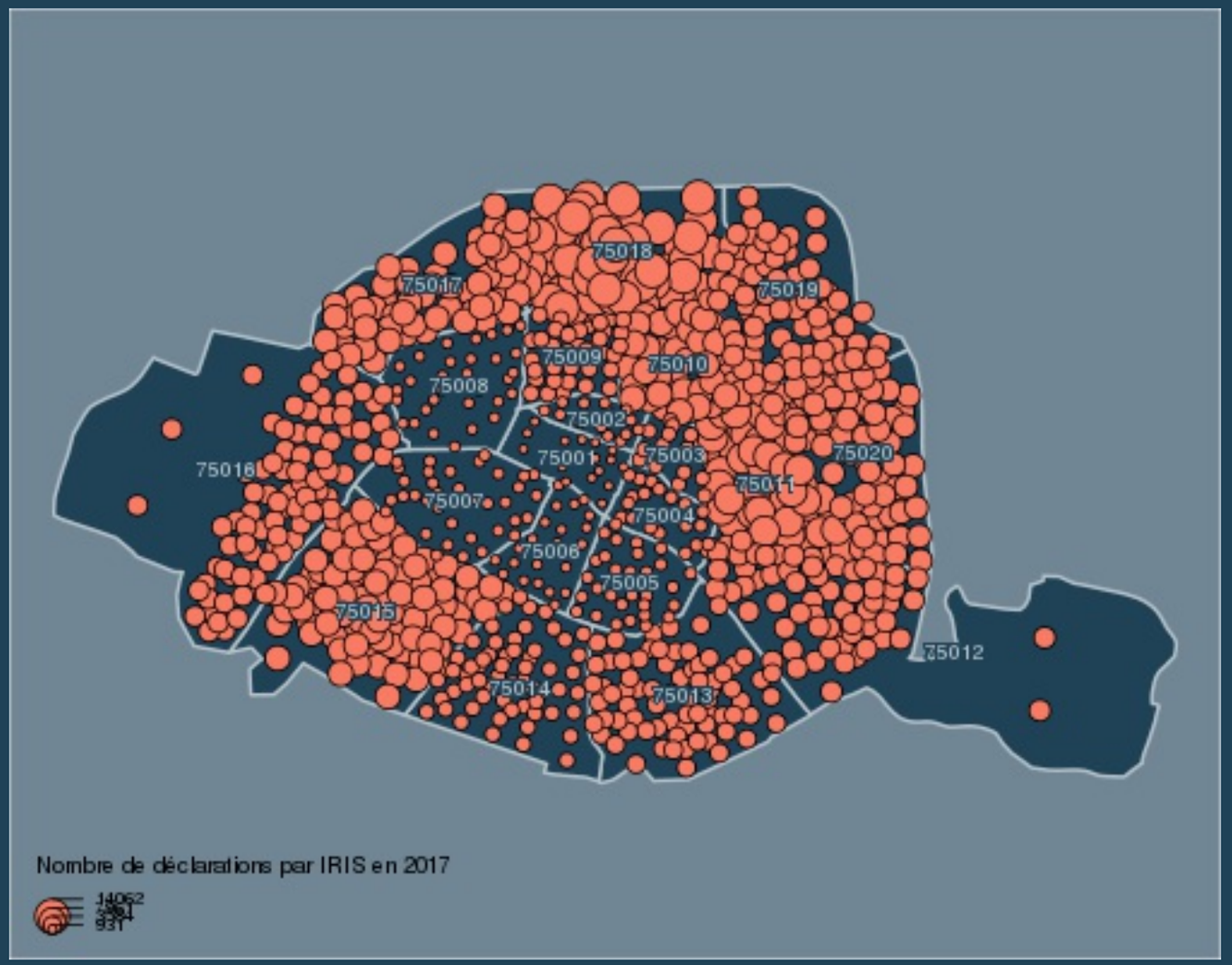
Cartographie statique : cartography

Exemple d'une carte en symboles proportionnels

```
library(cartography)
# On commence par afficher un fond de carte
par(mar=c(0,0,0,0))
plot(st_geometry(arrondissements_Paris),
     col = "#1F4257",
     border = "#B6C1C8",
     bg = "#708694",
     lwd = 2)

# On ajoute des symboles proportionnels pour les incidents
propSymbolsLayer(x = densite_incidents %>% filter(ANNEE_DECLARATION == 2017),
                 var = "NbIncidents",
                 col = "#F97B64",
                 inches = 0.1,
                 legend.title.txt = "Nombre de déclarations par IRIS en 2017")

# On ajoute les labels des arrondissements
labelLayer(x = arrondissements_Paris %>% mutate(CP = as.numeric(INSEE) - 100),
           txt = "CP",
           col = "#B6C1C8",
           bg = "#1F4257",
           halo = TRUE,
           overlap = FALSE, show.lines = FALSE)
```

Cartographie statique : cartography

Exemple d'une carte en symboles proportionnels



Cartographie dynamique : leaflet

- Pour créer des cartes "dynamiques", en plus de **mapview** (qui sert surtout à l'exploration de données), on peut utiliser le **leaflet**, qui génère une carte en HTML+JavaScript que l'on pourra placer sur une page web quelconque :

```
library(leaflet)
densite_incidents2017 <- densite_incidents %>%
  filter(ANNEE_DECLARATION == 2017) %>%
  st_transform(4326) # Leaflet requiert une couche en WGS84

seuils <- quantile(densite_incidents2017$densiteIncidents,
                   probs = seq(from = 0, to = 1, by = 0.2))

colorPalette <- colorBin("YlOrRd",
                         domain = densite_incidents2017$densiteIncidents,
                         bins = seuils)

infosPopup <- sprintf("<strong>%s</strong><br/>%s incidents  (%.1f / km²)",
                      densite_incidents2017$CODE_,
                      densite_incidents2017$NbIncidents,
                      densite_incidents2017$densiteIncidents) %>% lapply(htmltools::HTML)

leaflet(data = densite_incidents2017) %>%
  addProviderTiles(providers$CartoDB.DarkMatterNoLabels) %>%
  addPolygons(
    fillColor = ~colorPalette(densiteIncidents), fillOpacity = 0.7,
    color = "white", weight = .5, opacity = 1, dashArray = "3",
    label = infosPopup) %>%
  addLegend(pal = colorPalette, values = ~densiteIncidents, opacity = 0.7,
            title = "Densités d'incidents<br/>[incident/km²]", position = "topright")
```

Cartographie dynamique : `leaflet`

- Pour créer des cartes "dynamiques", en plus de **`mapview`** (qui sert surtout à l'exploration de données), on peut utiliser le **`leaflet`**, qui génère une carte en HTML+JavaScript que l'on pourra placer sur une page web quelconque :