AGENCE NATIONALE DE LA STATISTIQUE ET DE LA DEMOGRAPHIE

ECOLE NATIONALE DE LA STATISTIQUE ET DE L'ANALYSE ECONOMIQUE PIERRE NDIAYE

CARTOGRAPHIE AVEC R

Par

KPAKOU M'Mounéné
NDIAYE Saran
TRAORE Mohamed

Chargé du cours:

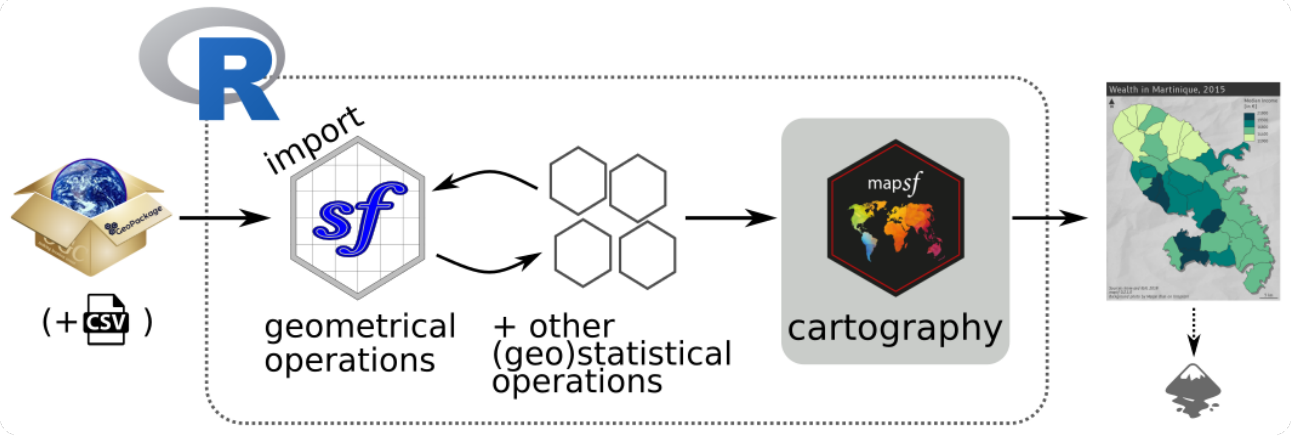
Mr. Aboubacar HEMA

Research Analyst

Libraries nécessaires

Introduction

Cet exposé nous enseigne une gamme de compétences spatiales, notamment : lire, écrire et manipuler des formats de fichiers géographiques ; créer des cartes statiques et interactives. il s'agit également de comprendre la structure des ensembles de données géographiques et des logiciels utilisés pour les traiter. il existe de nombreuses façons de gérer les données géographiques dans R, avec des dizaines de packages dans ce domaine.



I. Concepts de base en cartographie

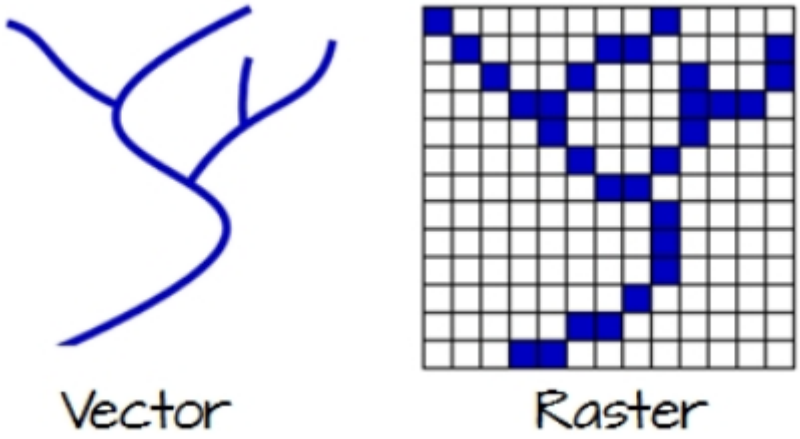
1. Données géographiques avec R

a. Présentation

Ce chapitre fournira des explications sur les modèles de données géographiques fondamentaux : vectoriels et raster.

-**Le modèle de données vectorielles** représente le monde à l'aide de points, de lignes et de polygones. Ceux-ci ont des frontières discrètes et bien définies, ce qui signifie que les ensembles de données vectorielles ont généralement un haut niveau de précision.

-**Le modèle de données raster** divise la surface en cellules de taille constante. Les rasters regroupent des caractéristiques spatialement spécifiques à une résolution donnée.



La figure ci dessus représente une même entité géographique dans l'un ou l'autre de ces formats.

b. Données vectorielles

Le modèle de données vectorielles géographiques est basé sur des points situés dans un système de référence de coordonnées (CRS). Les points peuvent représenter des éléments autonomes (par exemple, l'emplacement d'un arrêt de bus) ou ils peuvent être reliés entre eux pour former des géométries plus complexes telles que des lignes et des polygones. La figure ci dessous représente dans une même figure ces trois types d'objets.

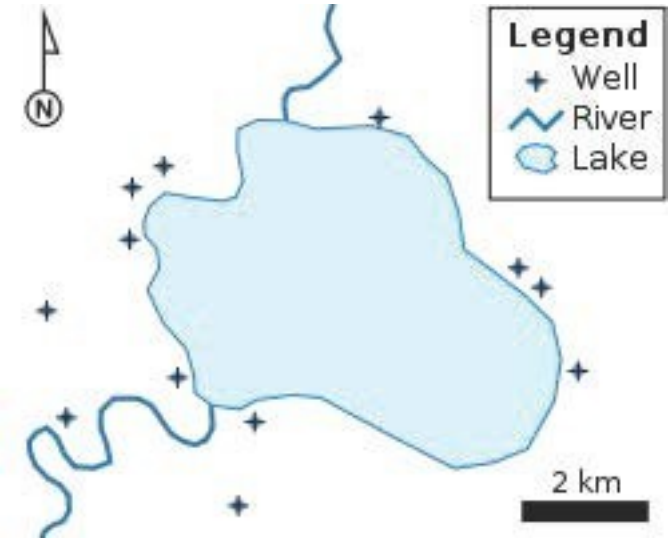


Figure 1: vector

Il existe plusieurs formats possibles pour stocker des données spatiales:

-**Format Shapefile**

Pour ce type de fichier, les données spatiales sont stockées dans plusieurs fichiers qui portent le même nom avec des extensions différentes. Le fichier qui porte l'extension **.shp** contient toute l'information liée à la géométrie des unités spatiales. Il doit être nécessairement accompagné de deux autres fichiers portant l'extension :

• **.dbf**, qui contient les données attributaires relatives aux données spatiales,

• **.shx**, qui contient les index des unités spatiales (fichier contenant des données sous forme d'octets.

Il existe parfois d'autres fichiers optionnels portant l'extension **.sbn**, **.sbx**, etc. qui sont également associés au fichier principal.

On notera que tous ces fichiers portent le même nom mais ont une extension différente.

Un fichier optionnel qui nous intéressera plus particulièrement est le fichier portant l'extension **.prj**, car ce fichier contient toute l'information relative sur le système de coordonnées.

-Format GeoJSON

Il s'afit d'un format de données spatiales de plus en plus utilisé dont on présente ci-dessous l'extrait d'un fichier. Il est issu de la syntaxe JSON. Il a l'avantage de contenir l'information (stockée dans un langage tout à fait compréhensible) dans un seul fichier : c'est-à-dire qu'on retrouve dans le même fichier l'information géographique sur les objets, l'information statistique observée sur ces objets et enfin le système de projection utilisé.

Nous utilisons le package sf pour l'analyse de données vectorielles. Le package sf est incontournable pour l'analyse des données vectorielles spatiales dans R. sf peut représenter tous les types de géométrie vectorielle courants (les classes de données raster ne sont pas prises en charge par sf) : points, lignes, polygones et leurs versions « multi » respectives (qui regroupent des entités du même type en une seule entité).

Une question plus spécifique du point de vue de R est « pourquoi utiliser le package sf » ? Les raisons sont nombreuses (liées aux avantages du modèle à fonctionnalités simples) :

• Lecture et écriture rapides des données

• Performances de traçage améliorées

• les objets sf peuvent être traités comme des trames de données dans la plupart des opérations

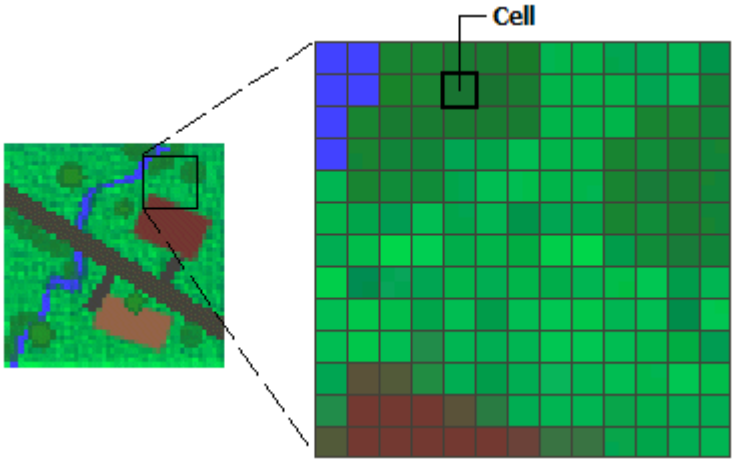
• les noms de fonctions sf sont relativement cohérents et intuitifs (ils commencent tous par st_)

• Les fonctions sf peuvent être combinées avec l' |>opérateur et fonctionnent bien avec la collection Tidyverse de packages R.

c. Données raster

Le modèle de données spatiales raster représente le monde avec une grille continue de cellules (souvent également appelées pixels). Ce modèle de données fait souvent référence à des grilles dites régulières, dans lesquelles chaque cellule a la même taille constante.
Contrairement aux données vectorielles, la cellule d'une couche raster ne peut contenir qu'une seule valeur.La valeur peut être continue ou catégorielle.
Les cartes raster représentent généralement des phénomènes continus tels que l'élévation, la température, la densité de population ou les données spectrales.

Le format le plus courant pour sauvegarder ces fichiers sont par exemple le format *GeoTIFF*. En général, ce type de données est volumineux, surtout losque la résolution d'une image (c'est-à-dire le nombre de pixels) est élevée.



d. Systèmes de coordonnées de référence

Les types de données spatiales vectorielles et raster partagent des concepts intrinsèques aux données spatiales. Le plus fondamental d'entre eux est peut-être le système de coordonnées de référence (CRS), qui définit la manière dont les éléments spatiaux des données sont liés à la surface de la Terre (ou à d'autres corps).
Les systèmes de référence de coordonnées géographiques identifient n'importe quel em-

placement sur la surface de la Terre à l'aide de deux valeurs : la longitude et la latitude

Sur R, pour créer un CRS on utilise la fonction CRS() du package sp. Par exemple :

```
library(sp)
crs <- CRS("+proj=utm +datum=WGS84 +ellps=WGS84")
```

•Le paramètre **+proj** spécifie la projection à utiliser, qui détermine comment les coordonnées géographiques sont transformées en coordonnées cartographiques.

• **+datum** spécifie le datum géodésique, qui définit l'origine, l'orientation et l'échelle du système de coordonnées par rapport à la Terre.

• **+ellps**, quant à lui, spécifie uniquement l'ellipsoïde de référence utilisé pour représenter la forme de la Terre.

Identification du système de coordonnées (CRS) d'un objet sf

-La fonction *st_crs()* permet d'identifier le système de coordonnées d'un objet sf.

```
st_crs(objetspa)
```

Modification du système de coordonnées (CRS) d'un objet sf

On peut modifier le « CRS » d'un objet sf dès lors que celui-ci est absent ou incorrect.

```
st_crs(objetspa)=newcrs
```

Pour convertir des données d'un CRS à un autre, on peut utiliser la fonction *st_transform()*

```
transformed_data <- st_transform(data, crs_new)
```

2. Opérations sur les données spatiales

a. Passage de données non spatiales à des données de classe "Spatial"

Nous allons utiliser dans la base de données *ACLED*

```
acled_df <- read.csv("Data/ACLED-Western_Africa.csv")
head(acled_df, 2)
```

```
        id      date annee                    type         pays latitude
1 BFO10160 30-Jun-23  2023 Strategic developments Burkina Faso  14.0400
```

```
2 BFO10164 30-Jun-23  2023 Strategic developments Burkina Faso  12.0299
  longitude
1   -0.0300
2    1.7719
```

Dans le cas de données de type vectoriel, on utilisera les packages *sp* et *sf*. Ces deux packages permettent de définir des classes d'objet "Spatial" (norme sp v.s. sf) et des fonctions associées qui permettent de faire du traitement de données adaptés et optimisés pour ce type de format.
Dans le cas de données de type raster, on utilisera essentiellement le package *raster* , qui devrait être remplacé petit à petit par le package *terra*.

b. Classe sp

Pour passer d'un objet de classe data.frame, à un objet de classe "Spatial" sp, il suffit d'utiliser la fonction **coordinates()** et de préciser avec le symbole ~ quelle sont les variables de géolocalisation. Par exemple :

```
acled_sp <- acled_df
coordinates(acled_sp) <- ~longitude + latitude
class(acled_sp)
```

```
[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

Dans ce cas, comme les objets géographiques correspondent à des points, la classe d'objet est la classe **SpatialPointsDataFrame**.
Les deux autres classes d'objets étant **SpatialLinesDataFrame** (pour les objets de type ligne brisée comme les routes) et **SpatialPolygonsDataFrame** (pour les objets de type polygone comme les contours administratifs). Nous appelerons par la suite la norme "Spatial" ces trois type d'objets.
Pour afficher les attributs de la classe sp (objet de classe S4), on utilise la fonction *str()*

```
str(acled_sp)
```

```
Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 54138 obs. of  5 variables:
  .. ..$ id   : chr [1:54138] "BFO10160" "BFO10164" "BFO10171" "MAA1633" ...
  .. ..$ date : chr [1:54138] "30-Jun-23" "30-Jun-23" "30-Jun-23" "30-Jun-23" ...
  .. ..$ annee: int [1:54138] 2023 2023 2023 2023 2023 2023 2023 2023 2023 2023 ...
  .. ..$ type : chr [1:54138] "Strategic developments" "Strategic developments" "Explosi
  .. ..$ pays : chr [1:54138] "Burkina Faso" "Burkina Faso" "Burkina Faso" "Mauritania"
  ..@ coords.nrs : int [1:2] 7 6
  ..@ coords     : num [1:54138, 1:2] -0.03 1.772 0.534 -17.038 -11.57 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:54138] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "longitude" "latitude"
  ..@ bbox       : num [1:2, 1:2] -25.16 2.86 14.65 25.99
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "longitude" "latitude"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr NA
```

Un objet SpatialPointsDataFrame est constitué des 5 attributs suivants : • **data** : le jeu de données au format data.frame,

• **coords.nrs** : indique le numéro des colonnes indiquant les localisations géographiques dans l'objet de départ ,

• **coords** : les variables de géolocalisation,

• **bbox** (pour "bounding box" en anglais) : une matrice qui indique les coordonnées de la fenêtre qui permet d'englober l'ensemble des objets spatiaux,

• **proj4string** : un objet de classe CRS qui détermine le CRS utilisé pour représenter les objets spatiaux.

Pour accéder aux différents élements qui constituent ce type d'objet, on utilise le symbole @. Par exemple, pour accéder uniquement au data.frame, on fait :

```
head(acled_sp@data, 2)
```

```
##         id      date annee                    type         pays
## 1 BFO10160 30-Jun-23  2023 Strategic developments Burkina Faso
## 2 BFO10164 30-Jun-23  2023 Strategic developments Burkina Faso
```

```
head(acled_sp@coords, 2)
```

```
##   longitude latitude
## 1   -0.0300  14.0400
## 2    1.7719  12.0299
```

c. Classe sf

Pour transformer un objet de classe data.frame (ou alors de type “Spatial”, c’est-à-dire **SpatialPointsDataFrame**, **SpatialLinesDataFrame** ou **SpatialPolygonsDataFrame**) en objet de classe sf, on utilise la fonction *st_as_sf()* de la manière suivante :

```
acled_sf <- st_as_sf(acled_df, coords = c("longitude", "latitude"))
class(acled_sf)
```

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

La structure de cet objet sf est comme celle d’un data.frame auquel on a ajouté une colonne geometry propre à l’information spatiale :

```
head(acled_sf,2)
```

```
Simple feature collection with 2 features and 5 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: -0.03 ymin: 12.0299 xmax: 1.7719 ymax: 14.04
CRS:           NA
        id      date annee                    type         pays
1 BFO10160 30-Jun-23  2023 Strategic developments Burkina Faso
2 BFO10164 30-Jun-23  2023 Strategic developments Burkina Faso
```

```
                 geometry
1    POINT (-0.03 14.04)
2 POINT (1.7719 12.0299)
```

On distingue les géometries suivantes:

•**POINT** : un point unique,

• **LINESTRING** : une séquence de points reliées par des traits,

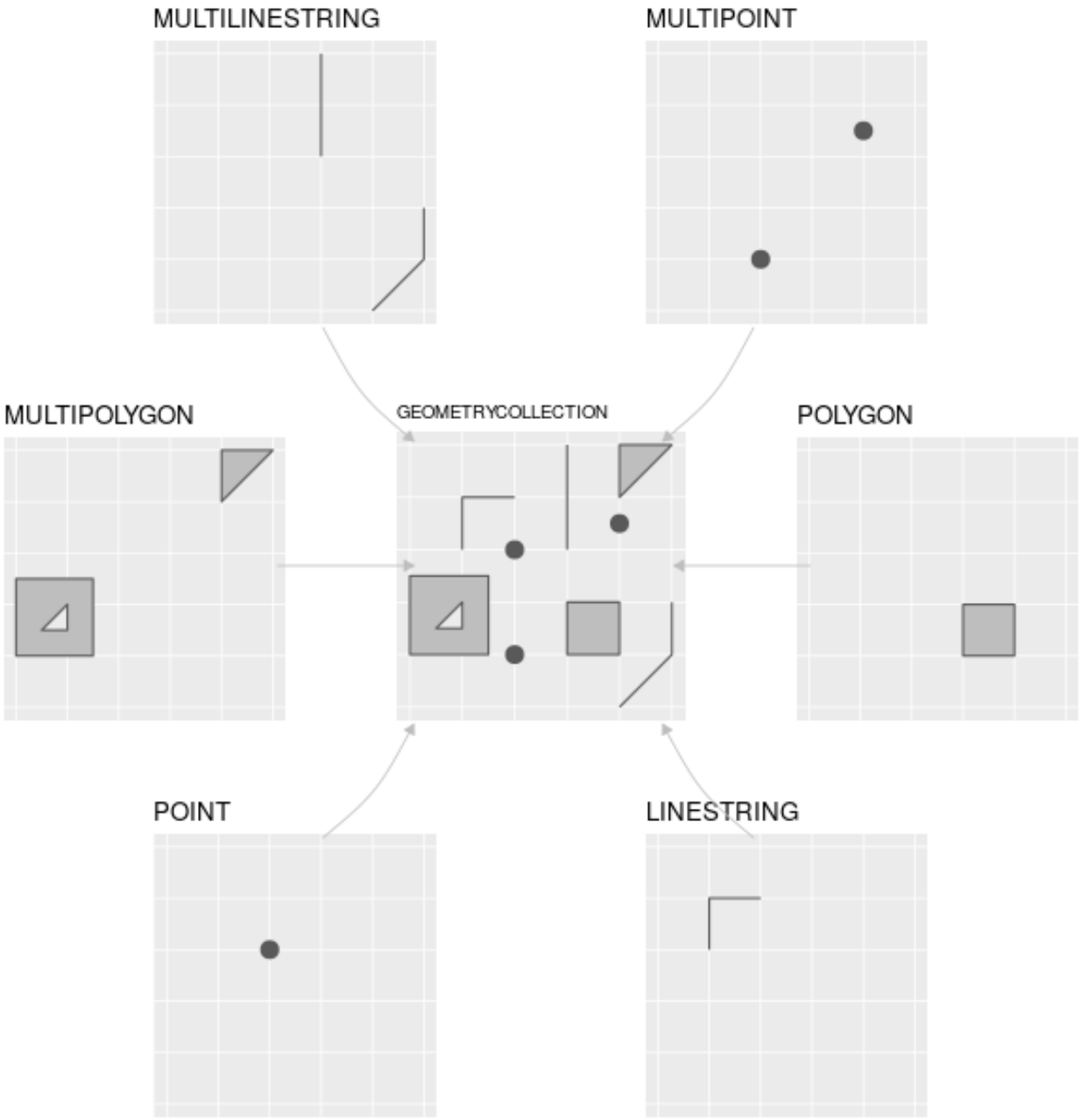
• **POLYGON** : une séquence de points définissant un polygone,

• **MULTIPOINT** : plusieurs points,

• **MULTILINESTRING** : plusieurs lignes,

• **MULTIPOLYGON** : plusieurs polygones

• **GEOMETRYCOLLECTION** : un mélange des géométries vues précédemment.



3. Manipulation des fichiers de données spatiales

Pour manipuler les objets spatiaux,il existe un certain nombre de fonctions qui peuvent y être appliquées. Pour connaître ces fonctions :

-**Objets sp**

```
methods(class = "Spatial")
```

```
 [1] $             $<-           [             [[            [[<-
 [6] [<-           aggregate     bbox          buffer        cbind
[11] coerce        coordinates<- couldBeLonLat crop          crs<-
[16] df_spatial    dimensions    distance      ext           extent
[21] fullgrid      geometry      geometry<-    geomtype      gridded
[26] head          is.projected  isLonLat      KML           mask
[31] merge         nlayers       over          plot          polygons
[36] print         proj4string   proj4string<- raster        rebuild_CRS
[41] select        shapefile     show          spChFIDs<-    spsample
[46] spTransform   st_as_sf      st_bbox       st_crs        subset
[51] summary       tail          vect          wkt           xmax
[56] xmin          ymax          ymin          zoom
see '?methods' for accessing help and source code
```

- **Objets sf**

```
methods(class = "sf")
```

```
 [1] $<-                      [
 [3] [[<-                     [<-
 [5] aggregate                anti_join
 [7] arrange                  as.data.frame
 [9] cbind                    coerce
[11] crs                      dbDataType
[13] dbWriteTable             df_spatial
[15] distance                 distinct
[17] dplyr_reconstruct        duplicated
[19] ext                      extent
[21] extract                  filter
[23] full_join                group_by
[25] group_split              identify
```

```
 [27] initialize                    inner_join
 [29] left_join                     lines
 [31] mask                          merge
 [33] mutate                        plot
 [35] points                        polys
 [37] print                         raster
 [39] rasterize                     rbind
 [41] rename                        rename_with
 [43] right_join                    rowwise
 [45] sample_frac                   sample_n
 [47] select                        semi_join
 [49] show                          slice
 [51] slotsFromS3                   st_agr
 [53] st_agr<-                      st_area
 [55] st_as_s2                      st_as_sf
 [57] st_as_sfc                     st_bbox
 [59] st_boundary                   st_break_antimeridian
 [61] st_buffer                     st_cast
 [63] st_centroid                   st_collection_extract
 [65] st_concave_hull               st_convex_hull
 [67] st_coordinates                st_crop
 [69] st_crs                        st_crs<-
 [71] st_difference                 st_drop_geometry
 [73] st_filter                     st_geometry
 [75] st_geometry<-                 st_inscribed_circle
 [77] st_interpolate_aw             st_intersection
 [79] st_intersects                 st_is
 [81] st_is_valid                   st_join
 [83] st_line_merge                 st_m_range
 [85] st_make_valid                 st_minimum_rotated_rectangle
 [87] st_nearest_points             st_node
```

```
 [89] st_normalize                st_point_on_surface
 [91] st_polygonize               st_precision
 [93] st_reverse                  st_sample
 [95] st_segmentize               st_set_precision
 [97] st_shift_longitude          st_simplify
 [99] st_snap                     st_sym_difference
[101] st_transform                st_triangulate
[103] st_triangulate_constrained  st_union
[105] st_voronoi                  st_wrap_dateline
[107] st_write                    st_z_range
[109] st_zm                       summarise
[111] svc                         transform
[113] transmute                   ungroup
[115] vect
see '?methods' for accessing help and source code
```

-Objets raster

```
methods(class = "raster")
```

```
## [1] [         [<-       anyNA     as.matrix as.raster is.na     Ops
## [8] plot      print
## see '?methods' for accessing help and source code
```

4. Union, intersection sur les géométries

Avec la norme sf, les fonctions qui peuvent être utiles pour travailler sur les géométries sont :

• st_intersection(A, B) : l'intersection des géométries A et B

• st_union(A, B) : l'union des géométries A et B

• st_difference(A, B) : la géométrie de A auquel on enlève la géométrie de B

• st_convex_hull(x) : détermine une enveloppe convexe de x

• st_area(x) : détermine l'aire de x

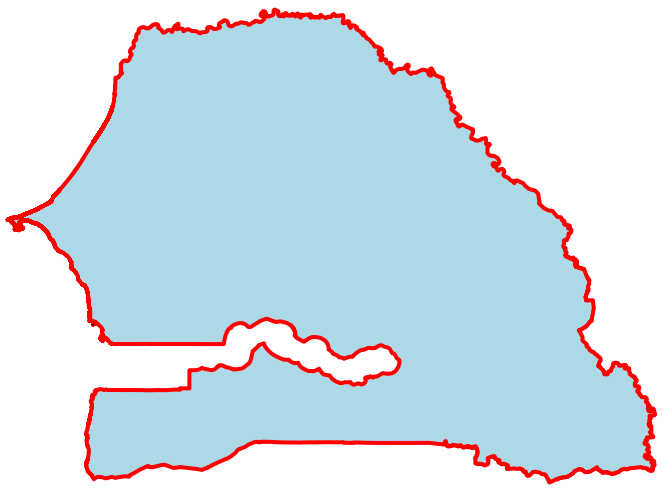
• st_buffer(x) : détermine un tampon de x.

a. Union simple

```
sen <- suppressMessages(st_read("Data/gadm41_SEN_shp/gadm41_SEN_2.shp"))
```

```
Reading layer `gadm41_SEN_2' from data source
  `E:\KPAKOU\KPAM ISEP 3\SEMESTRE 6\Projet sous R\Cartographie avec R\Data\gadm41_SEN_sh
  using driver `ESRI Shapefile'
Simple feature collection with 45 features and 13 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -17.54319 ymin: 12.30786 xmax: -11.34247 ymax: 16.69207
Geodetic CRS:  WGS 84
```

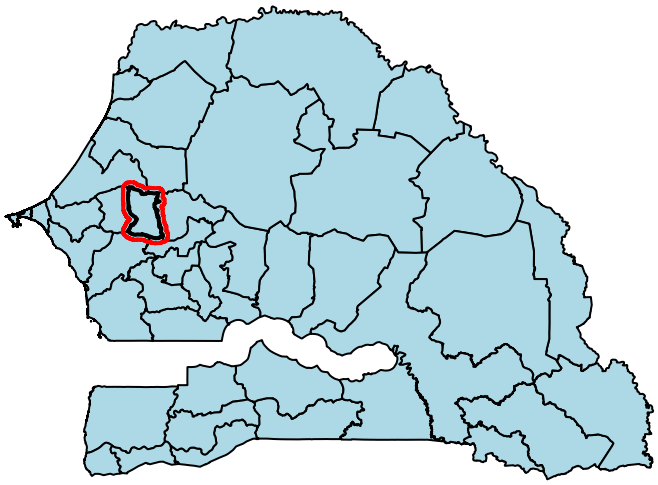
```
sen_u <- st_union(sen)
plot(st_geometry(sen_u), col="lightblue",main = "Carte du sénégal")
plot(st_geometry(sen_u), add=T, lwd=2, border = "red")
```



b. Zone tampon

```
Diourbel <- sen[sen$NAME_2 == "Diourbel", ]
Diourbel_b <- st_buffer(x = Diourbel, dist = 5000)
plot(st_geometry(sen), col = "lightblue",main = "Carte du sénégal/ Diourbel avec un buff
plot(st_geometry(Diourbel), add = TRUE, lwd = 2)
plot(st_geometry(Diourbel_b), add = TRUE, lwd = 2, border = "red")
```

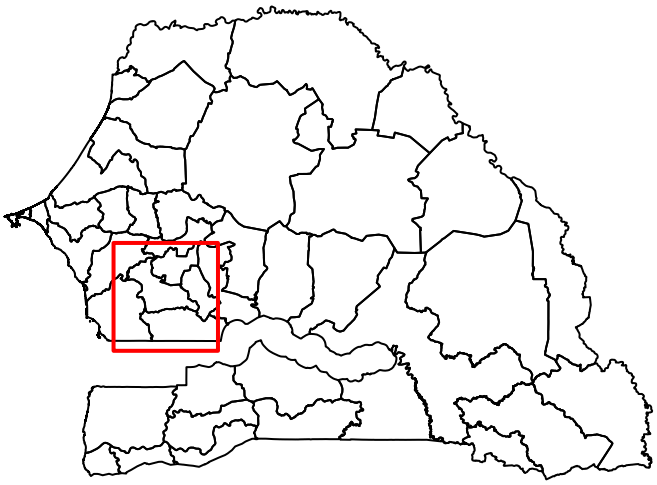
Carte du sénégal/ Diourbel avec un buffer de 5km



c. Intersection de polygones

Pour ce faire on crée un polygone

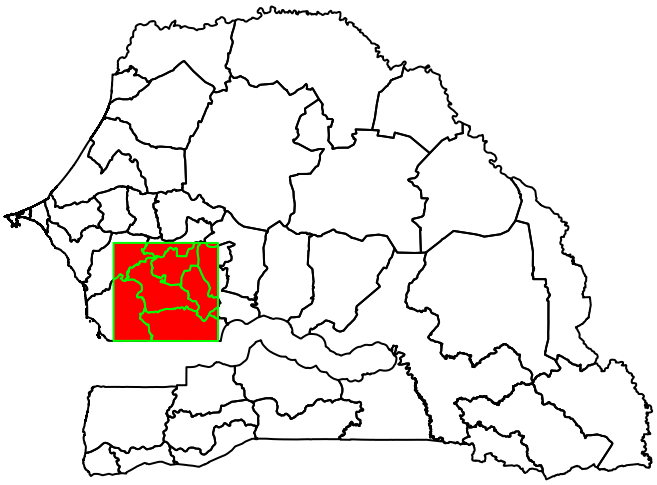
```
m <- rbind(c(-16.5, 13.5), c(-16.5, 14.5), c(-15.5, 14.5),
           c(-15.5, 13.5), c(-16.5, 13.5))
polygone <- st_sf(st_sfc(st_polygon(list(m))), crs = st_crs(sen))
plot(st_geometry(sen))
plot(polygone, border = "red", lwd = 2, add = TRUE)
```



- La fonction `st_intersection()` extrait la partie de mtq qui s'intersecte avec le polygone créé.

```
inter <- st_intersection(x = sen, y = polygone)
plot(st_geometry(sen),main = "Intersection du sénégal avec le polygone")
plot(st_geometry(inter), col = "red", border = "green", add = TRUE)
```

Intersection du sénégal avec le polygone



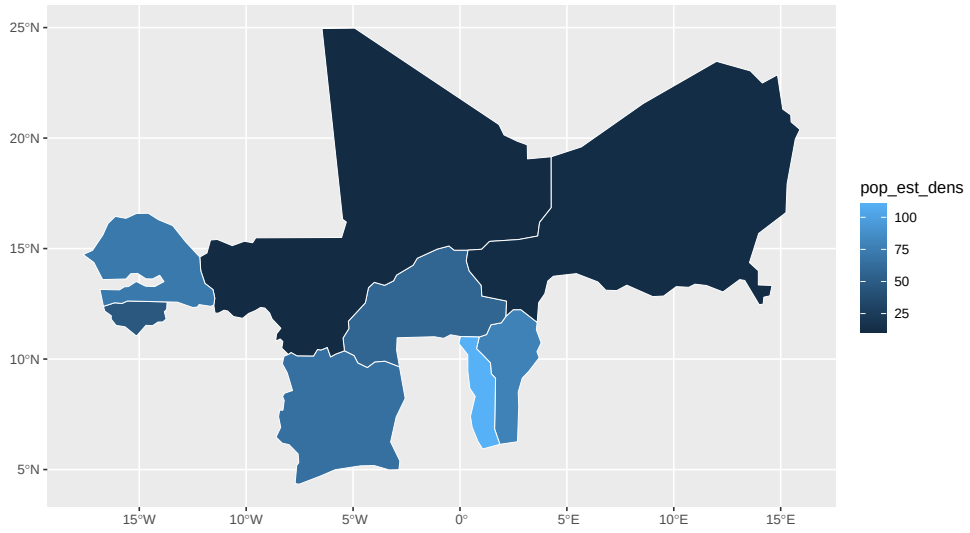
II. Cartes statiques

1 .Avec ‘ggplot 2‘

a. Les cartes choroplèthe

Le premier exemple que nous allons pouvoir voir, c’est une carte choroplèthe, càd en aplat de couleur, adaptée à la représentation d’indicateurs intensifs (%, densité…).

```
library(ggplot2)
ggplot(data = uemoa_sf) +
  geom_sf(mapping = aes(fill = pop_est_dens), color = "white")
```

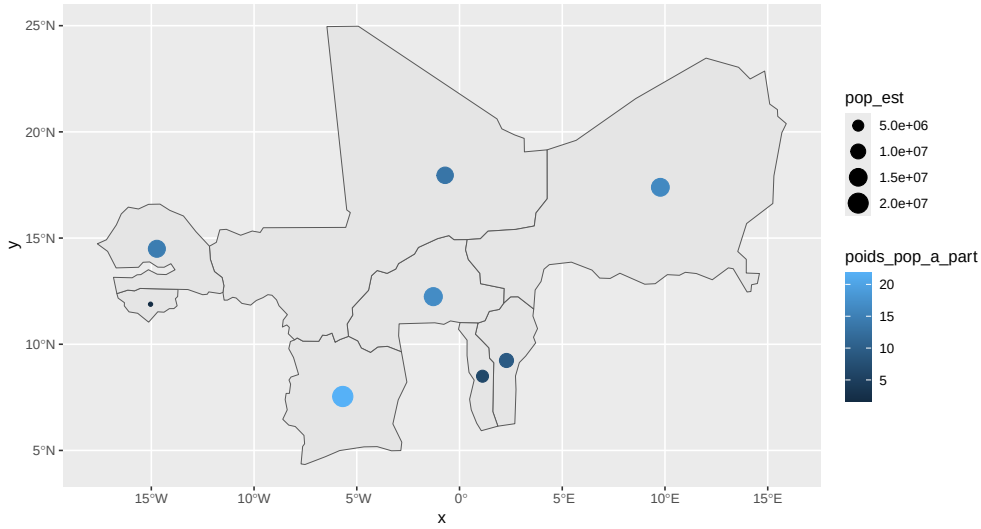


b. Les cartes à ronds proportionnels

- Utilisation de La fonction `stat_sf_coordinates()` permet d'extraire les coordonnées d'un objet 'sf' avant de produire le calque.

```
library(ggplot2)
carte_ronds_prop <- ggplot(data = uemoa_sf) +
  geom_sf() +
  stat_sf_coordinates(mapping = aes(size = pop_est, color = poids_pop_a_part))

carte_ronds_prop
```



Ajout du titre ,legende,flêche pole nord…

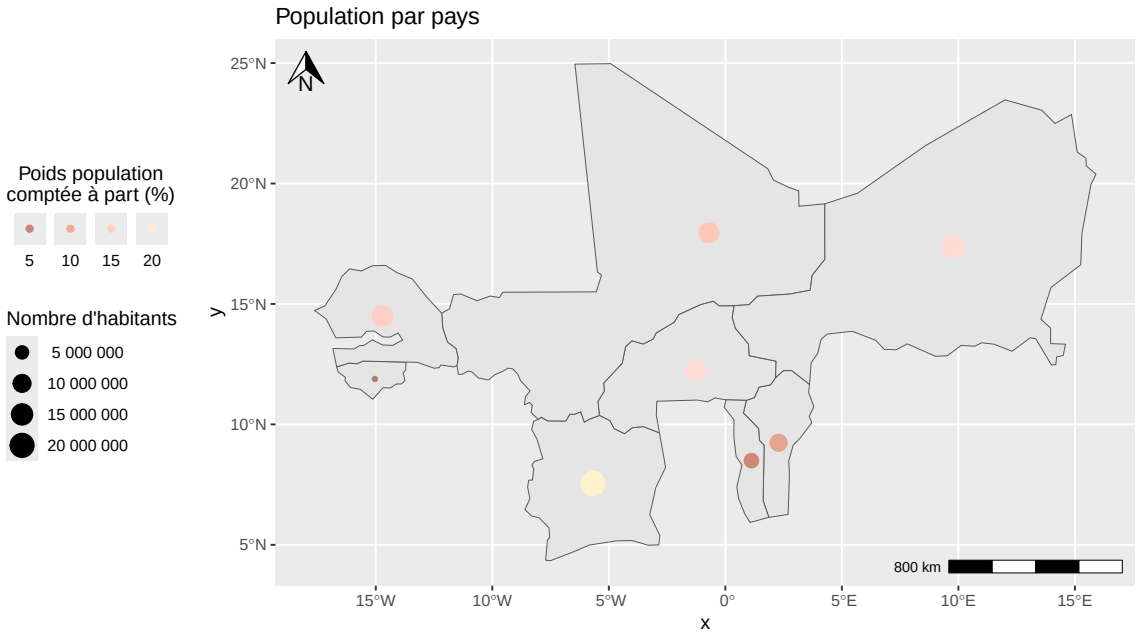
- *Utilisation des fonction* :

  -`labs()` pour les labels

  -`title` pour le tire, `subtitle` pour le sous-titre et `caption` le bas de page .

  -`guide_xx()` et `guides()` permettent de modifier finement la légende.

  -`annotation_scale()` et `annotation_north_arrow()` pour ajouter l' échelle et la flèche du nord.



2 .Avec mf_map

La fonction `mf_map()` est la fonction centrale du package `mapsf` . Elle permet de réaliser la plupart des représentations usuelles en cartographie. Ces arguments principaux sont :

• **x**, un objet sf ;

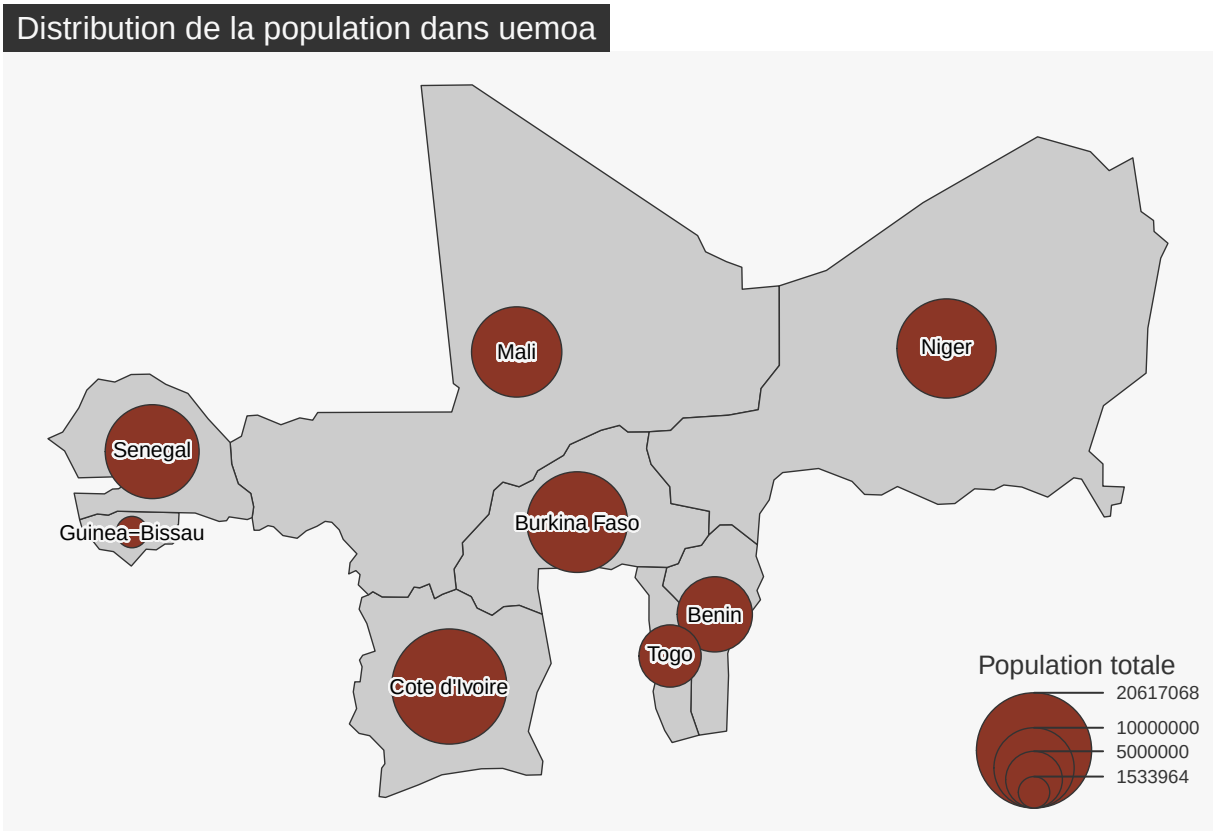
• **var**, le nom de la variable à représenter ;

• **type**, le type de représentation.

Utilisée sans précision de type, la fonction `mf_map()` affche simplement les couches spatiales.

a. Carte de symboles proportionnels

```
# uemoa
mf_map(x = uemoa_sf)
# Symboles proportionnels
mf_map(
x = uemoa_sf,
var = "pop_est",
type = "prop",
leg_title = "Population totale"
)
mf_label(
x = uemoa_sf,
var = "name",
col= "black",
halo = TRUE,
overlap = FALSE,
lines = FALSE
)
# Titre
mf_title("Distribution de la population dans uemoa")
```

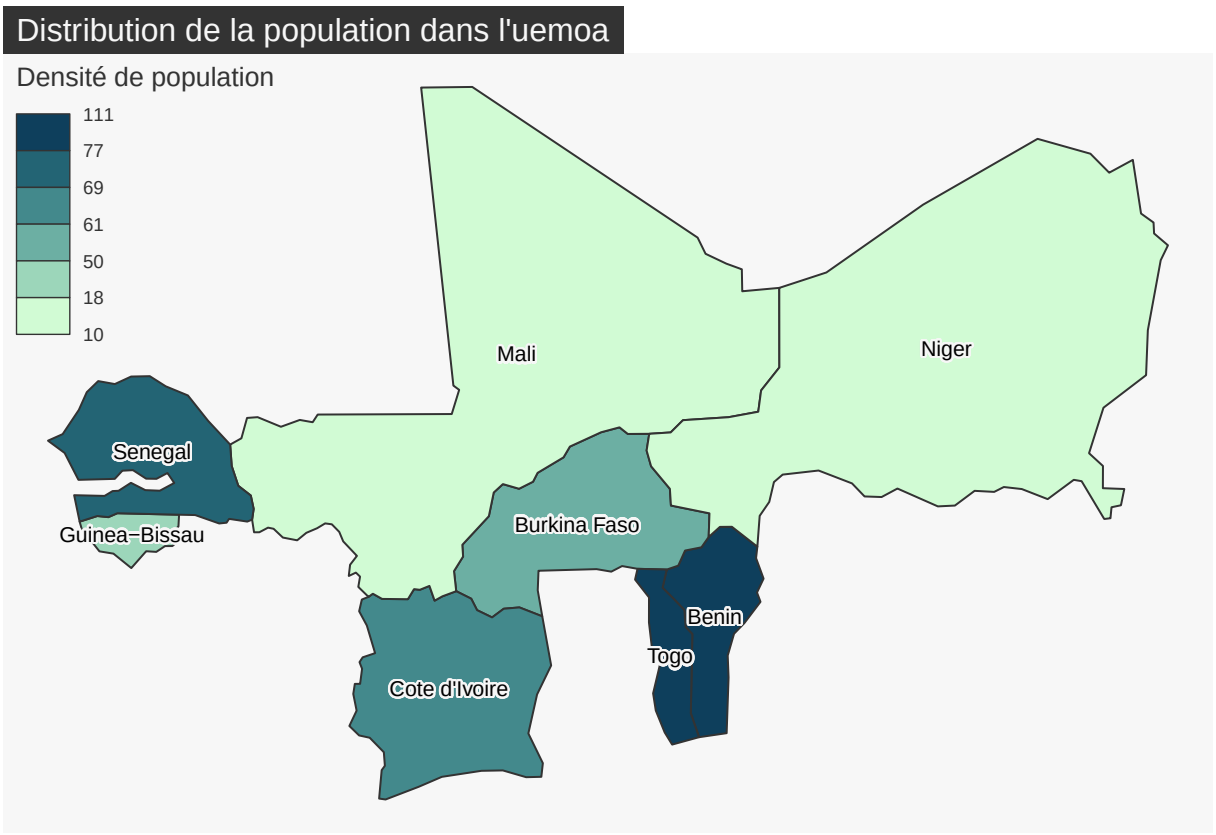


b. Les cartes choroplèthe

```
mf_map(x = uemoa_sf,
var = "pop_est_dens",
type = "choro",
breaks = "quantile",
nbreaks = 6,
pal = "Dark Mint",
lwd = 1,
leg_title = "Densité de population",
leg_val_rnd = 0,
leg_pos = "topleft"


)
mf_label(
```

```
x = uemoa_sf,
var = "name",
col= "black",
halo = TRUE,
overlap = FALSE,
lines = FALSE
)
mf_title("Distribution de la population dans l'uemoa")
```



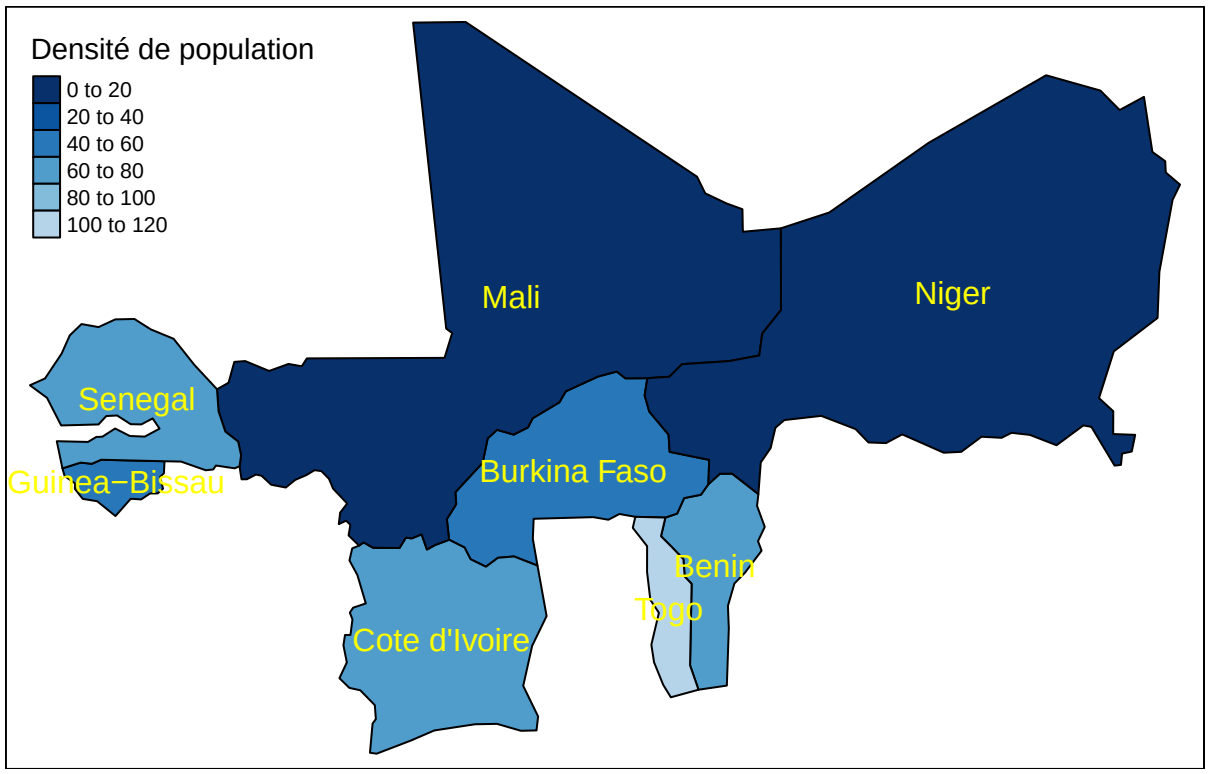
3.Avec 'tmap'

La syntaxe de `tmap` est similaire à celle de ggplot2. Cela inclut une séparation stricte entre les données et l'esthétique. Au début, `tm_shape()` est transmis un ensemble de données, suivi d'un ou plusieurs niveaux qui définissent le type d'affichage.

Les exemples sont `tm_fill()` et `tm_dots()` pour tracer des données sous forme de polygones ou de points.

Exemple de graphique avec `tmap`

```
library(tmap)
# plot
tm_shape(uemoa_sf) +
  tm_polygons("pop_est_dens", palette = "-Blues",
    title = "Densité de population", contrast = 0.7, border.col = "black", id = "name")
  tm_text("name", col = "yellow")
```



III- cartes interactives

(confère Presentation.Rmd)

Nous allons presentez certains packages permattant la representation interactive des cartes

1. ggiraph

ggiraph est un package qui permet de créer des graphes interactifs, notamment via l'ajout de tooltips, d'animations, et d'actions JavaScript. Il dispose notamment d'une fonction permettant de réaliser des cartes interactives.

2. cartes interactives avec leaflet

```
(confère Presentation.Rmd)
```

La représentation de données spatiales se veut de plus en plus interactive, et quoi de mieux qu'une carte pour illustrer son propos lorsque l'on souhaite transmettre une information géographique ?
Nous allons utiliser le package *leaflet*

IV- Raster

```
(confère Presentation.Rmd)
```