Agence Nationale de Statistique et de la Démographie ¶

ANSD

¶

Ecole Nationale de la Statistique et de l'Analyse Economique

¶

ENSAE Pierre NDIAYE

¶

CARTOGRAPHIE SUR R

¶

Rédigé par :
Yatoute MINTOAMA
Richard GOZAN
Isabelle Danielle MOSSE

Elèves ingénieurs statisticiens économistes

Sous la supervision de :
HEMA Aboubacar
Research Analyst

24 Juin 2023

Contents

Introduction

```
install.packages(c(
"cartography", # réaliser des cartes
"classInt", # discrétisation de variables quantitatives
"ggspatial", # syntaxe complémentaires à la ggplot
"GISTools", # outils pour faire de la carto
"leaflet", # interractivité avec JavaScript
"maptools", # manipulation de données "spatial",
"OpenStreetMap", # OSM
"osrm", # openstreetmap avec R
```

```
"popcircle", # représentation style bubble plot
"raster", # manipulation de données raster
"RColorBrewer", # palette de couleurs pour carto
"rgdal", # import de données spatiales
"rgeos", # manipulation de données spatiales
"sf", # nouvelle classe d'objets spatials
"sp", # ancienne classe d'objets spatials
"tidyverse", # ggplot, dplyr, etc
"tmaptools" # pour la carto
),
dependencies = TRUE)
devtools::install_github(repo = 'rCarto/photon')
```

I. Conceptes de base de la cartographie

I.1. Système de coordonnées de référence (CRS)

Le système de coordonnées de référence (CRS) est un élément essentiel de la cartographie. Il définit comment les coordonnées spatiales sont représentées dans un système de référence donné, permettant ainsi de situer les objets géographiques sur la Terre.Si on possède un fichier de données spatial sans cette information, il sera difficile, voire impossible de travailler avec plusieurs sources de données.

Pour construire un CRS, il faut définir essentiellement les deux critères suivants :

- choisir une forme géométrique pour représenter la terre ;
- choisir une projection pour représenter la forme de la terre, initialement en 3D, en deux dimensions.

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.

Pour les CRS les plus connus, ceux adoptés en général par des organismes officiels, il existe un code *EPSG*. Par exemple, pour le Référentiel Géodésique Français 93, le code EPSG correspondant est le 2154. Ainsi, plutôt que d'appeler le CRS par tous les éléments qui le décrivent, on pourra utiliser son code *EPSG* :

```
CRS("+epsg=2154")
```

Pour attribuer un CRS à un objet spatial existant, on peut utiliser la fonction st_crs() du package *sf* :

```
library(sf)
data <- st_read("data.shp")
st_crs(data) <- crs
```

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

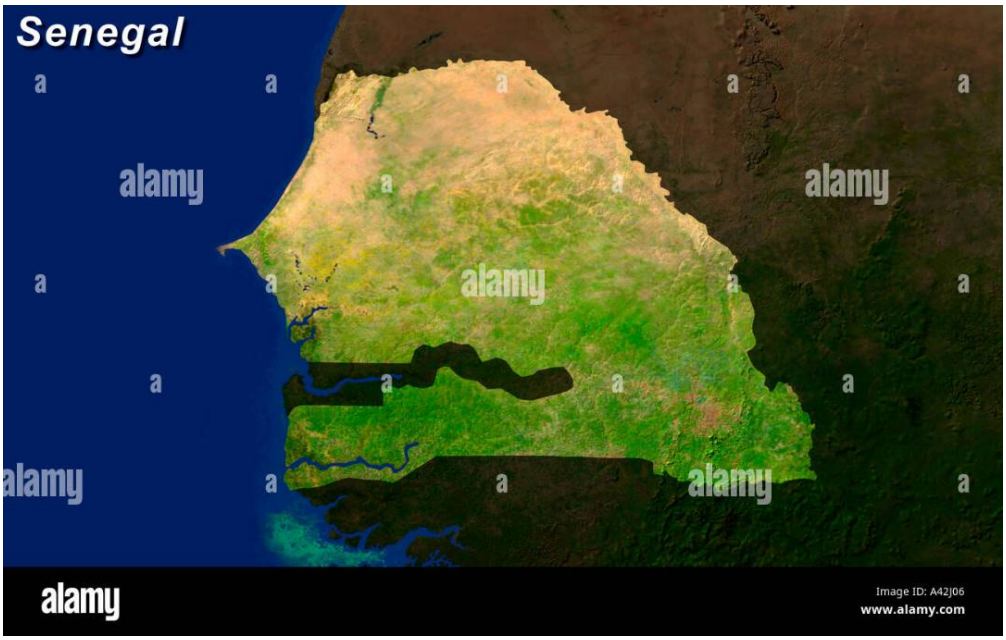
```
transformed_data <- st_transform(data, crs_new)
```

I.2. Fichiers de données spaciales

D'un point de vue informatique, il y a deux modes fondamentaux qui permettent de distinguer les données spatiales : le mode raster et le mode vecteur. Selon le mode, les données sont stockées dans des formats différents.

I.2.1. Le mode raster

Dans sa forme la plus simple, un raster est une image, autrement dit une matrice composée de pixels de même taille. A chaque pixel, on peut observer une information qualitative (par exemple, la valeur oui ou non, le type de sol urbain, forêts, prairies, etc.) ou alors une information quantitative (par exemple une altitude et dans ce cas on représente la valeur par un dégré de coloration plus ou moins fort selon le code couleur utilisé).



Dans ce type de données on y trouve des fichiers d'extention :

- .png ou .jpg ou .tif: représentant généralement une visualisation des données raster ;

• .asc (ASCII Grid) : stocke sous forme de grille régulière, les informations sur les coordonnées spatiales des cellules, les limites du raster, la résolution et les valeurs associées à chaque cellule.

• .db : une table attributaire qui associe des informations non spatiales aux cellules du raster, telles que des noms, des valeurs, des descriptions ou d'autres attributs spécifiques.

I.2.2. Le mode vecteur

Le format vectoriel utilise le concept d'objets géométriques (points, lignes, polygones) ou "Spatial Features" pour représenter les entités géographiques.Il existe plusieurs formats vectoriels possibles pour stocker des données spatiales. Nous allons décrire dans cette section les deux principaux : Shapefile et GeoJSON.

Format shapefile

Pour ce type de fichier, les données spatiales sont stockés 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 (dBase File): une table attributaire qui contient les attributs associés à chaque entité géographique du fichier .shp ;

• .shx (Shape Index File): un index spatial qui permet d'accéder rapidement aux données géométriques dans le fichier .shp.

Format GSON

Il s'afit d'un format de données spatiales issu de la syntaxe JSON. Il 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. Ci-dessous un extrait d'un fichier de ce format :


```
{"type": "FeatureCollection",
"crs": {"type": "name", "properties": {"name": "EPSG:26904"}},
"features": [
{"type": "Feature",
"properties": {
"name": "Van Dorn Street",
"marker-color": "#0000ff",
"marker-symbol": "rail-metro",
"line": "blue"},
"geometry": {
"type": "Point",
"coordinates": [
-77.12911152370515,
```

```
38.79930767201779
]}
}]
}
```

I.2.3. Le format de données spatiales Geopackage

Le format de données spatiales Geopackage est un format de base de données géospatiales qui combine à la fois les données vecteur et raster. Il a été développé comme un standard ouvert par l'Open Geospatial Consortium (OGC) et est largement utilisé dans le domaine de la géomatique.

Geopackage est conçu pour stocker des ensembles de données spatiales avec une structure organisationnelle hiérarchique. Il utilise une seule base de données SQLite pour stocker à la fois les données géospatiales et les attributs associés. Cela permet de stocker plusieurs couches de données (point, ligne, polygone, raster, etc.) dans un seul fichier, ce qui facilite le partage et la distribution des données.

I.2.4. Où trouver des fichiers de données spatiales "Open Data" ?

Ci-dessous une liste non exhaustive de liens où il est possible de télécharger des fichiers de données spatiales "Open Data" dont on utilisera une partie dans la suite de cet eexposé :

• Les contours administratifs par pays : le site https://www.gadm.org/ présente une liste exhaustive de fonds de cartes présentant les contours administratifs de la plupart des pays du globe. Le découpage est fait en plusieurs niveaux allant du contours du pays jusqu'aux contours des communes.

• l'IGN propose de télécharger gratuitement un certain nombre de données françaises portant sur les axes routiers, l'hydrographie, l'altitude, la localisation d'exploitation agricole, le découpage de la France en iris (qui est un sous-découpage des communes en France), etc. Pour accéder à ces données : https://geoservices.ign.fr/documentation/diffusion/telechargement-donnees-libres.html

• Le site de la "Natural Earth" (http://www.naturalearthdata.com/) propose de télécharger un certain nombre de données écologiques à l'échelle planétaire.

• Les données libres américaines diffusées par le gouvernement américain (www.data.gov), la version européenne (data.europa.eu/euodp) et la version française (https://www.data.gouv.fr/fr/) dont de nombreuses bases sont géoréférencées.

• La SNCF diffuse de nombreuses bases de données en libre accès : https://data.sncf.com/explore/?sort=modified

II. Lecture et manipulations des données spaciales sur R

II.1. Passage des données non spaciale à des données de la classe "Spaciale"

Commençons par importer des données classiques, c'est-à-dire dans un format non spatial, par exemple auformat texte .csv. Pour importer ces données sous R :

```
seisme_df <- read.csv2("Donnees/earthquake/earthquakes.csv")
head(seisme_df, 2)
```

```
##   Year Month   YYMM Day Time.hhmmss.mm.UTC Latitude Longitude Magnitude Depth
## 1 1973     1 197301   1            34609.8    -9.21    150.63       5.3    41
## 2 1973     1 197301   1            52229.8   -15.01   -173.96       5.0    33
```

Dans la suite, nous présenterons les deux solutions (*sp* et *sf*) simultatément

II.1.1. La 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 :

```
library(sp)
seisme_sp <- seisme_df
coordinates(seisme_sp) <- ~Longitude + Latitude
class(seisme_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)

Pour afficher quelles sont ces attributs, on peut utiliser la fonction *str()*

```
str(seisme_sp)
```

```
## Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
##   ..@ data       :'data.frame':  57230 obs. of  7 variables:
##   .. ..$ Year              : int [1:57230] 1973 1973 1973 1973 1973 1973 1973 1973 19
##   .. ..$ Month             : int [1:57230] 1 1 1 1 1 1 1 1 1 ...
##   .. ..$ YYMM              : int [1:57230] 197301 197301 197301 197301 197301 197301
##   .. ..$ Day               : int [1:57230] 1 1 1 2 2 2 2 3 3 3 ...
##   .. ..$ Time.hhmmss.mm.UTC: num [1:57230] 34610 52230 114238 5320 22709 ...
```

```
##   .. ..$ Magnitude         : num [1:57230] 5.3 5 6 5.5 5.4 5.2 5.2 5.6 5.5 5.3 ...
##   .. ..$ Depth             : int [1:57230] 41 33 33 66 61 30 33 563 33 18 ...
##   ..@ coords.nrs : int [1:2] 7 6
##   ..@ coords     : num [1:57230, 1:2] 150.6 -174 -16.2 117.4 126.2 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:57230] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:2] "Longitude" "Latitude"
##   ..@ bbox       : num [1:2, 1:2] -180 -72.5 180 87
##   .. ..- 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
```

Pour accéder aux différents élements qui constituent ce type d'objet, on utilise le symbole **@**. Par exemple:

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

```
##   Year Month   YYMM Day Time.hhmmss.mm.UTC Magnitude Depth
## 1 1973     1 197301   1            34609.8       5.3    41
## 2 1973     1 197301   1            52229.8       5.0    33
```

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

```
##   Longitude Latitude
## 1    150.63    -9.21
## 2   -173.96   -15.01
```

II.1.2 La classe sf

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

```
library(sf)
seisme_sf <- st_as_sf(seisme_df, coords = c("Longitude", "Latitude"))
class(seisme_sf)
```

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

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

```
head(seisme_sf)
```

```
## Simple feature collection with 6 features and 7 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -173.96 ymin: -35.51 xmax: 150.63 ymax: 5.4
```

```
## CRS:           NA
##   Year Month   YYMM Day Time.hhmmss.mm.UTC Magnitude Depth
## 1 1973     1 197301   1            34609.8       5.3    41
## 2 1973     1 197301   1            52229.8       5.0    33
## 3 1973     1 197301   1           114237.5       6.0    33
## 4 1973     1 197301   2             5320.3       5.5    66
## 5 1973     1 197301   2            22709.2       5.4    61
## 6 1973     1 197301   2            34752.5       5.2    30
##                  geometry
## 1   POINT (150.63 -9.21)
## 2 POINT (-173.96 -15.01)
## 3  POINT (-16.21 -35.51)
## 4   POINT (117.43 -9.85)
## 5    POINT (126.21 1.03)
## 6     POINT (-82.54 5.4)
```

Pour travailler uniquement sur le jeu de données et exclure les géométries, on utilise la fonction *st_drop_geometry()*

```
head(seisme_sf %>% st_drop_geometry(), 2)
```

```
##   Year Month   YYMM Day Time.hhmmss.mm.UTC Magnitude Depth
## 1 1973     1 197301   1            34609.8       5.3    41
## 2 1973     1 197301   1            52229.8       5.0    33
```

Pour changer le CRS, on utilise la fonction *st_crs()* :

```
st_crs(seisme_sf) <- 4326
```

II.2. Importation et exportation des données spaciales

II.2.1. Importation

Mode vecteur

On va importer le jeu de données qui contient les contours administratifs des pays (donc des polygones) connus sur Terre. Il s'agit d'un fichier Shapefile. Nous allons voir les deux façons d'importer ces données selon qu'on choisit la classe sp ou bien la classe sf

a) Classe sp

Nous utilisons la fonction "readOGR()" issu du package "rgdal" qui permet d'importer les types de données spatiales:

```
library(rgdal)
world_sp <- readOGR(dsn = "Donnees/World WGS84",
layer = "Pays_WGS84")
```

```
## Warning: OGR support is provided by the sf and terra packages among others
```

```
## Warning: OGR support is provided by the sf and terra packages among others
```

```
## Warning: OGR support is provided by the sf and terra packages among others

## Warning: OGR support is provided by the sf and terra packages among others

## Warning: OGR support is provided by the sf and terra packages among others

## Warning: OGR support is provided by the sf and terra packages among others

## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\NIMBUZ\Desktop\Donnees\World WGS84", layer: "Pays_WGS84"
## with 251 features
## It has 1 fields
```

Dans le cas de polygones, l'objet spatial est un SpatialPolygonsDataFrame. Pour analyser sa structure,on extrait ici une seule observation (on procède de la même façon qu'on fait avec un data.frame) :

```
str(world_sp[53, ])
```

```
## Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
##   ..@ data       :'data.frame':  1 obs. of  1 variable:
##   .. ..$ NOM: chr "France"
##   ..@ polygons   :List of 1
##   .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
##   .. .. .. ..@ Polygons :List of 2
##   .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
##   .. .. .. .. .. .. ..@ labpt  : num [1:2] 2.46 46.63
##   .. .. .. .. .. .. ..@ area   : num 63.3
##   .. .. .. .. .. .. ..@ hole   : logi FALSE
##   .. .. .. .. .. .. ..@ ringDir: int 1
##   .. .. .. .. .. .. ..@ coords : num [1:718, 1:2] -1.78 -1.73 -1.67 -1.59 -1.53 ...
##   .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
##   .. .. .. .. .. .. ..@ labpt  : num [1:2] 9.1 42.2
##   .. .. .. .. .. .. ..@ area   : num 1
##   .. .. .. .. .. .. ..@ hole   : logi FALSE
##   .. .. .. .. .. .. ..@ ringDir: int 1
##   .. .. .. .. .. .. ..@ coords : num [1:45, 1:2] 9.45 9.43 9.41 9.4 9.4 ...
##   .. .. .. ..@ plotOrder: int [1:2] 1 2
##   .. .. .. ..@ labpt    : num [1:2] 2.46 46.63
##   .. .. .. ..@ ID       : chr "52"
##   .. .. .. ..@ area     : num 64.3
##   .. .. .. ..$ comment: chr "0 0"
##   ..@ plotOrder  : int 1
##   ..@ bbox       : num [1:2, 1:2] -4.79 41.36 9.56 51.09
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2] "x" "y"
##   .. .. ..$ : chr [1:2] "min" "max"
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
```

```
##   .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +no_defs"
##   .. .. ..$ comment: chr "GEOGCRS[\"WGS 84\",\n    DATUM[\"World Geodetic System 1984
##   ..$ comment: chr "TRUE"
```

On retrouve pratiquement les mêmes attributs que pour un objet SpatialPointsDataFrame exceptés les attributs polygons et plotOrder

On va essayer d'aller un peu plus loin dans l'analyse d'un tel objet

```
str(world_sp[53, ]@polygons[[1]])
```

```
## Formal class 'Polygons' [package "sp"] with 5 slots
##   ..@ Polygons :List of 2
##   .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
##   .. .. .. ..@ labpt  : num [1:2] 2.46 46.63
##   .. .. .. ..@ area   : num 63.3
##   .. .. .. ..@ hole   : logi FALSE
##   .. .. .. ..@ ringDir: int 1
##   .. .. .. ..@ coords : num [1:718, 1:2] -1.78 -1.73 -1.67 -1.59 -1.53 ...
##   .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
##   .. .. .. ..@ labpt  : num [1:2] 9.1 42.2
##   .. .. .. ..@ area   : num 1
##   .. .. .. ..@ hole   : logi FALSE
##   .. .. .. ..@ ringDir: int 1
##   .. .. .. ..@ coords : num [1:45, 1:2] 9.45 9.43 9.41 9.4 9.4 ...
##   ..@ plotOrder: int [1:2] 1 2
##   ..@ labpt    : num [1:2] 2.46 46.63
##   ..@ ID       : chr "52"
##   ..@ area     : num 64.3
##   ..$ comment: chr "0 0"
```

On constate qu'il s'agit d'un objet de classe Polygons et que celui est constitué de 5 attributs.

b) Classe sf Nous pouvons utiliser la fonction st_read() qui permet de lire de nombreux types de fichiers spatiales. Nous allons importer des fichiers shapefile et geopackage

- Fichier shp

```
world_sf <- st_read("Donnees/World WGS84/Pays_WGS84.shp")
```

```
## Reading layer `Pays_WGS84' from data source
##   `C:\Users\NIMBUZ\Desktop\Donnees\World WGS84\Pays_WGS84.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 251 features and 1 field
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -180 ymin: -89.9 xmax: 180 ymax: 83.6236
## Geodetic CRS:  WGS 84
```

Les coordonnées des polygones sont stockées dans la colonne geometry :

```
head(world_sf)
```

```
## Simple feature collection with 6 features and 1 field
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -176.6445 ymin: 0.2152777 xmax: 112.7472 ymax: 80.50166
## Geodetic CRS:  WGS 84
##                NOM                          geometry
## 1    Baker Island MULTIPOLYGON (((-176.4614 0...
## 2  Howland Island MULTIPOLYGON (((-176.6362 0...
## 3  Johnston Atoll MULTIPOLYGON (((-169.5389 1...
## 4 Paracel Islands MULTIPOLYGON (((112.2714 16...
## 5        Svalbard MULTIPOLYGON (((27.145 80.0...
## 6       Jan Mayen MULTIPOLYGON (((-9.043091 7...
```

Pour accéder à la géométrie :

```
str(st_geometry(world_sf[53, ]))
```

```
## sfc_MULTIPOLYGON of length 1; first list element: List of 2
##  $ :List of 1
##   ..$ : num [1:718, 1:2] -1.78 -1.73 -1.67 -1.59 -1.53 ...
##  $ :List of 1
##   ..$ : num [1:45, 1:2] 9.45 9.43 9.41 9.4 9.4 ...
##  - attr(*, "class")= chr [1:3] "XY" "MULTIPOLYGON" "sfg"
```

- fichier gpkp

Les lignes suivantes importent 2 couches dans le fichier geopackage lot46.gpkg

```
dep <- st_read("Donnees/data/lot46.gpkg", layer = "departement", quiet = TRUE)
route <- st_read("Donnees/data/lot46.gpkg", layer = "route", quiet = TRUE)
com <- st_read("Donnees/data/lot46.gpkg", layer = "commune")
```

```
## Reading layer `commune' from data source
##   `C:\Users\NIMBUZ\Desktop\Donnees\data\lot46.gpkg' using driver `GPKG'
## Simple feature collection with 313 features and 12 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 539668.5 ymin: 6346290 xmax: 637380.9 ymax: 6439668
## Projected CRS: RGF93 v1 / Lambert-93
```

Mode raster

Pour importer les données raster sous R, nous pouvons utliser la fonction *raster()* issu du package *"raster"*

```
library(raster)
wind <- raster("Donnees/images/FRA_wind-speed_200m.tif")
```

II.2.2. Exportation

La fonction *st_write()* d'exporter de nombreux types de fichiers.

La ligne suivante exporte l'objet dep dans un dossier data au format shapefile.

```
st_write(obj = dep, "Donnees/data/dep.shp",
         layer_options = "ENCODING=UTF-8",
         delete_layer = TRUE)
```

```
## Deleting layer `dep' using driver `ESRI Shapefile'
## Writing layer `dep' to data source
##   `Donnees/data/dep.shp' using driver `ESRI Shapefile'
## options:        ENCODING=UTF-8
## Writing 96 features with 5 fields and geometry type Multi Polygon.
```

La ligne suivante exporte l'objet world_sf dans un dossier data au format GeoJSON

```
st_write(world_sf, "Donnees/data/world_json.json",
         driver = "GeoJSON")
```

II.3. Manipulations d'objets de type "Spatial"

Pour chacune des trois classes d'objets vues ci-dessus, il existe un certain nombre de fonctions qui peuvent y être appliquées. Pour connaître ces fonctions :

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

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

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

```
##  [1] $<-                       [
##  [3] [[<-                      aggregate
##  [5] as.data.frame             cbind
##  [7] coerce                    crs
##  [9] dbDataType                dbWriteTable
## [11] distance                  duplicated
## [13] extent                    extract
```

```
## [15] filter                          identify
## [17] initialize                      lines
## [19] mask                            merge
## [21] plot                            print
## [23] raster                          rasterize
## [25] rbind                           select
## [27] show                            slotsFromS3
## [29] st_agr                          st_agr<-
## [31] st_area                         st_as_s2
## [33] st_as_sf                        st_as_sfc
## [35] st_bbox                         st_boundary
## [37] st_break_antimeridian           st_buffer
## [39] st_cast                         st_centroid
## [41] st_collection_extract           st_concave_hull
## [43] st_convex_hull                  st_coordinates
## [45] st_crop                         st_crs
## [47] st_crs<-                        st_difference
## [49] st_drop_geometry                st_filter
## [51] st_geometry                     st_geometry<-
## [53] st_inscribed_circle             st_interpolate_aw
## [55] st_intersection                 st_intersects
## [57] st_is                           st_is_valid
## [59] st_join                         st_line_merge
## [61] st_m_range                      st_make_valid
## [63] st_minimum_rotated_rectangle st_nearest_points
## [65] st_node                         st_normalize
## [67] st_point_on_surface             st_polygonize
## [69] st_precision                    st_reverse
## [71] st_sample                       st_segmentize
## [73] st_set_precision                st_shift_longitude
## [75] st_simplify                     st_snap
## [77] st_sym_difference               st_transform
## [79] st_triangulate                  st_triangulate_constrained
## [81] st_union                        st_voronoi
## [83] st_wrap_dateline                st_write
## [85] st_z_range                      st_zm
## [87] transform
## see '?methods' for accessing help and source code
```

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

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

II.3.1. Fonctions de base

Pour connaître le nombre d'observations et le nombre de variables, on utilise la fonction dim() (dans le cas de la norme sf, la géométrie compte pour une variable) :

```
dim(world_sp)
```

```
## [1] 251   1
```

```
dim(world_sf)
```

```
## [1] 251   2
```

Pour changer le nom des observations, on utilise row.names() :

```
row.names(world_sp) <- as.character(world_sp@data$NOM)
```

II.3.2. Sélection d'observations

Pour sélectionner un sous-échantillon, on utilise la même syntaxe que pour les data.frame dans le cas de la norme sp :

```
uemoa_sp <- world_sp[c("Benin", "Burkina Faso",
                       "Guinea-Bissau","Ivory Coast",
                     "Mali", "Niger", "Senegal", "Togo"),]
```

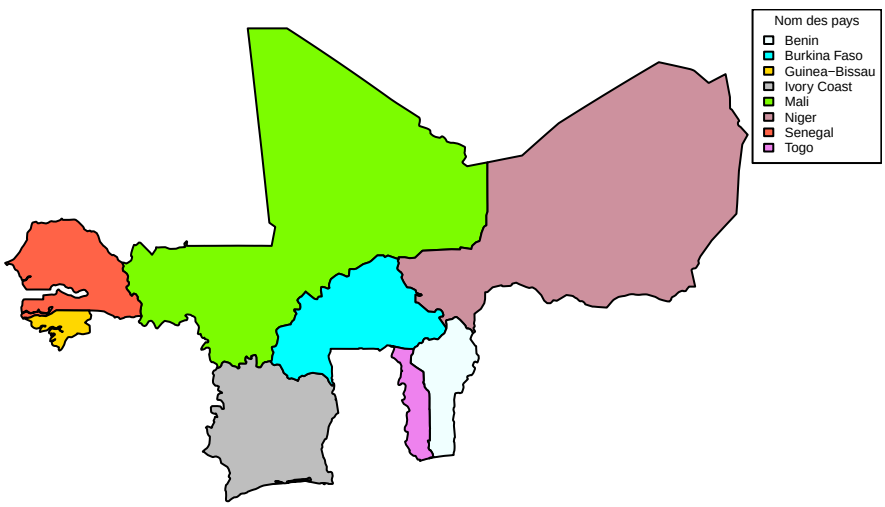
Pour la classe sf, on peut utiliser la même syntaxe que pour sp, mais en plus, on peut utiliser la syntaxe à la mode dplyr.

```
library(tidyverse)
uemoa_sf <- world_sf %>%
  filter(NOM %in% c("Benin", "Burkina Faso", "Guinea-Bissau",
                    "Ivory Coast", "Mali",
                    "Niger", "Senegal","Togo"))
```

II.3.3. Représentations de cartes de base

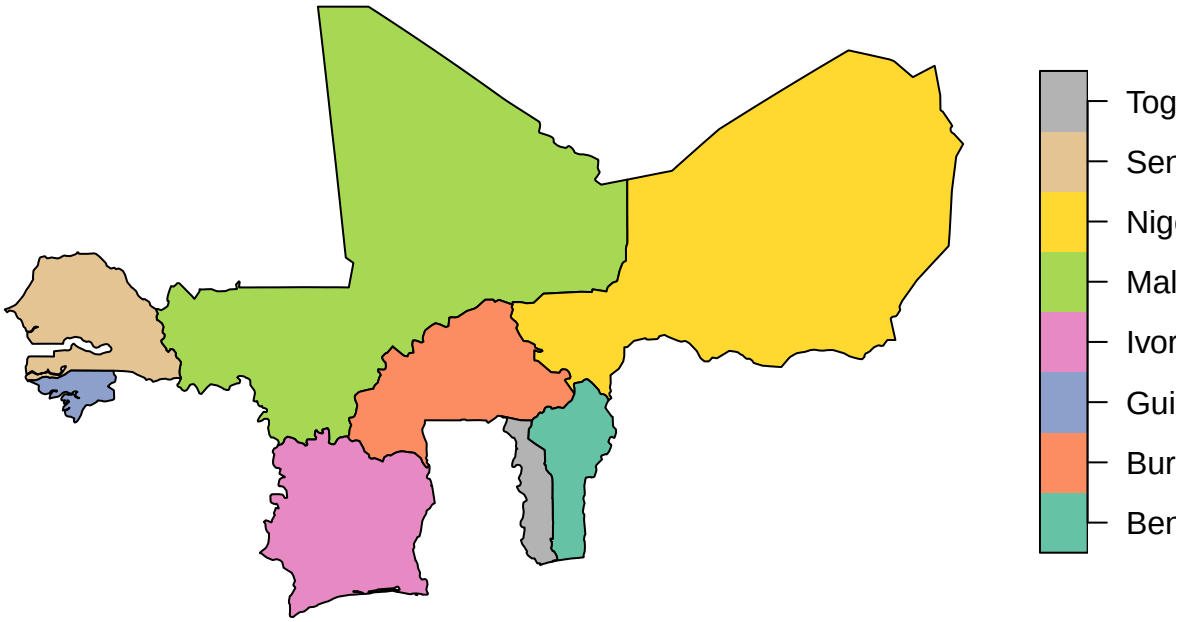
On utilise la fonction plot() qui appliquée à un objet "Spatial" va seulement représenter la géométrie de l'objet. On peut ensuite utiliser les fonctions graphiques de base (title(), legend(), etc.) pour ornementer le graphique

```
plot(uemoa_sp, col = c("azure","cyan","gold","gray",
                       "lawngreen", "pink3","tomato","violet"))
title("Pays de l'UEMOA")
legend("topright",
       legend = row.names(uemoa_sp),
       cex = 0.4, title = "Nom des pays",
       fill = c("azure","cyan","gold","gray",
      "lawngreen","pink3","tomato","violet"))
```



Pour la norme sf, le principe est différent. En effet, une carte par variable sera automatiquement représentée.Dans cet exemple, le jeu de données ne contient qu'une seule variable et par conséquent une seule figure est représentée. Pour ne sélectionner que la géométrie, il aurait fallu faire appel à la fonction st_geometry().

```
plot(uemoa_sf, main = "Pays de l'UEMOA")
```



II.3.4. Aggrégation des données

Pour aggréger des données spatiales, il y a deux contraintes :

1. faire l'aggrégation sur les objets spatiaux : par exemple deux polygones contigus vont fusioner pour n'en former plus qu'un seul.
2. faire l'aggrégation sur les variables. Dans ce cas-là, il est important de tenir compte de la nature des variables à aggréger

On va ici réaliser ces deux étapes avec la norme "Spatial" à la suite. Pour fusionner les objets spatiaux, on utilisera la fonction *unionSpatialPolygons()* incluse dans le package maptools. L'argument IDs contient un vecteur de la même taille que la table initiale où chaque élément correspond au nom de l'observation dans la nouvelle table. Par exemple, pour fusionner le "Benin" et le "Togo", sachant que le Benin et le Togo sont les éléments 1 et 8 de la table initiale, on fera :

```
library(maptools)
uemoa_sp_new <- unionSpatialPolygons(uemoa_sp,
                IDs = c("Benin", "Burkina Faso",                    "Guinea
                        "Niger", "Senegal", "Benin"))
```

L'objet créé, de classe SpatialPolygons ne contient que la géométrie des observations et pas de data.frame. On va donc créer un nouveau jeu de données avec les mêmes observations.

```
uemoa2.df <- data.frame(NOM = c("Benin", "Burkina Faso",
                        "Guinea-Bissau", "Ivory Coast",
                        "Mali", "Niger", "Senegal"),
  pib = c(14954,16686,1458,59221,16183,12609,23354),
  pop = c(12996,22100,2060,27478,21904,25252,16876),
  region = rep("O", 7))
row.names(uemoa2.df) <- uemoa2.df$NOM
```

Enfin, on associe les géométries au jeu de données en utilisant la fonction *SpatialPolygonsDataFrame()*

```
uemoa_sp_new <- SpatialPolygonsDataFrame(
  uemoa_sp_new, uemoa2.df)
class(uemoa_sp_new)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

Pour aggéger les données spatiales de type sf, on peut aussi utiliser la syntaxe dplyr. Ici, on aggrége d'abord les entités spatiales et on applique ensuite la fonction merge() :

```
uemoa_sf_new <- uemoa_sf %>% group_by(
  NOM = c("Benin", "Burkina Faso", "Guinea-Bissau",
  "Ivory Coast", "Mali","Niger", "Senegal", "Benin")) %>%
  summarise() %>%
  merge(uemoa2.df, by = "NOM")
class(uemoa_sf_new)
```

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

II.3.5. Ajout de données spatiales

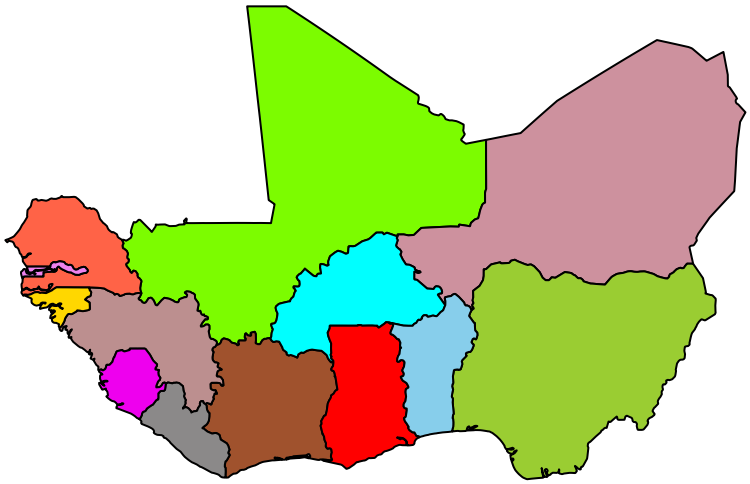
Pour pouvoir ajouter une unité spatiale à un objet "Spatial", il faut que les deux objets aient les mêmes attributs (i.e. les même variables). Par exemple, pour ajouter les pays de la ZMOA aux données précédentes, on créée d'abord un objet "Spatial" :

```
zmoa_sp <- world_sp[c("Gambia, The", "Ghana", "Guinea",
    "Liberia", "Nigeria", "Sierra Leone"), ]

zmoa_df <- data.frame(NOM = c("Gambia, The", "Ghana",
  "Guinea", "Liberia", "Nigeria", "Sierra Leone"),
   pib = c(1719,66882,13606,2965,452971,3505),
   pop = c(2639,31394,13531,5193,213401,8141),
  region = rep("O", 6))
zmoa_sp <- merge(zmoa_sp, zmoa_df, by = "NOM")
row.names(zmoa_sp) = c("Gambia", "Ghana",
  "Guinea", " Liberia", " Nigeria", "Sierra Leone")
```

On utilise ensuite la fonction *spRbind()* (analogue de la fonction rbind())

```
westAf_sp <- spRbind(uemoa_sp_new, zmoa_sp)
plot(westAf_sp, col =
        c("skyblue","cyan","gold","sienna","lawngreen",
        "pink3","tomato","violet","red1","rosybrown",
        "snow4","yellowgreen","magenta2"))
```



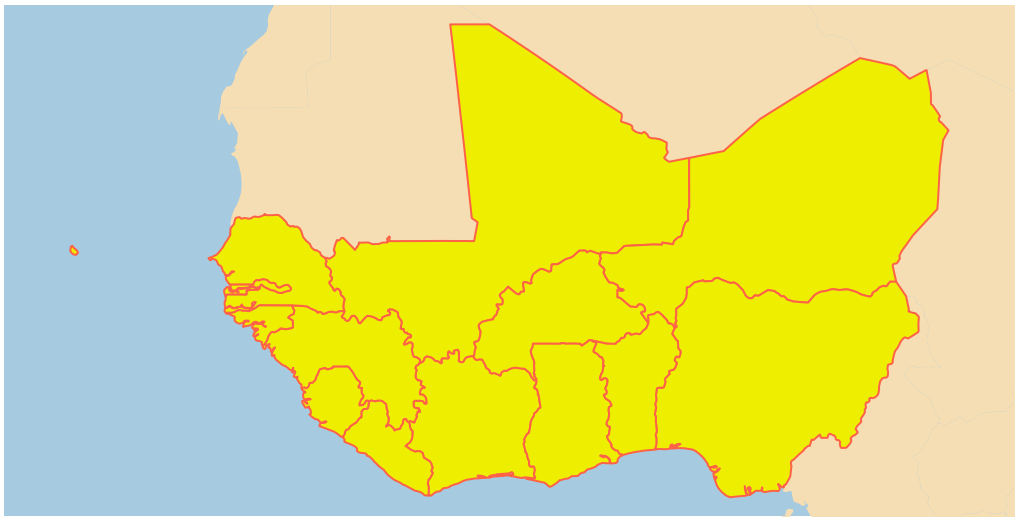
Nous ajoutons le cap vert pour former la CEDEAO:

```
capvert_sp <- world_sp["Cape Verde", ]
capvert_df <- data.frame(NOM = "Cape Verde", pib = 1784,
                         pop = 587925, region = "0")
capvert_sp <- merge(capvert_sp, capvert_df, by = "NOM")
row.names(capvert_sp) = "Cape Verde"
cedeao_sp <- spRbind(westAf_sp, capvert_sp)
```

Nous affichons la carte de la CEDEAO:

```
plot(cedeao_sp)
```

```
plot(cedeao_sp, border = NA, col = "NA", bg = "#A6CAE0")
plot(world_sp, col = "wheat", border = NA, add = T)
plot(cedeao_sp, col = "yellow2", border = "tomato", add = T)
```



Le principe est le même avec la classe sf sauf que la fonction s'appelle rbind() et qu'on peut continuer à utiliser la syntaxe dplyr :

```
zmoa_sf <- world_sf %>% filter(
        NOM %in% c("Gambia, The", "Ghana", "Guinea",
                   "Liberia", "Nigeria", "Sierra Leone"))

zmoa_sf <- merge(zmoa_sf, zmoa_df)
westAf_sf <- uemoa_sf_new %>% rbind(zmoa_sf)
```

Ajout du cap vert:

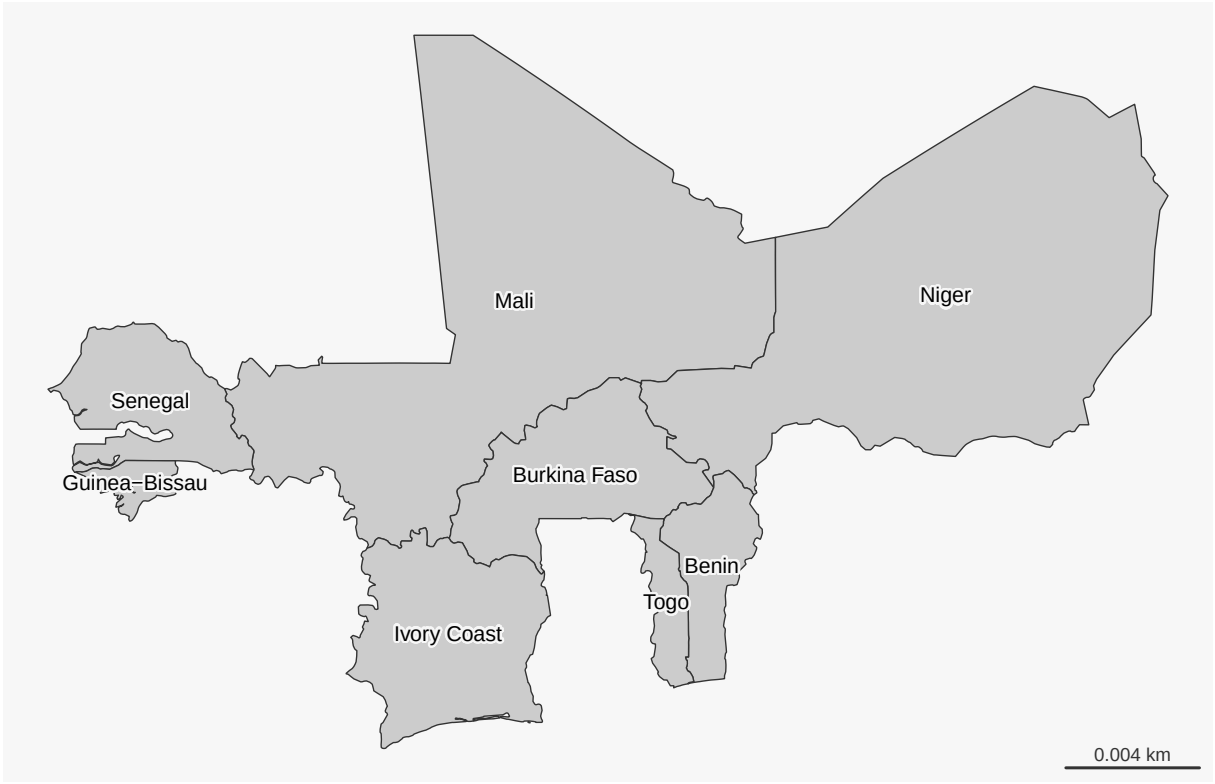
```
capvert_sf <- world_sf[world_sf$NOM == "Cape Verde", ]
capvert_sf <- merge(capvert_sf, capvert_df)
cedeao_sf <- westAf_sf%>%
  rbind(capvert_sf)
```

II.3.6. Représenter une étiquette

La fonction *mf_label()* du package *mapsf* est dédiée à l'afichage d'étiquettes

```
library(mapsf)
mf_map(uemoa_sf)
mf_label(
```

```
  x = uemoa_sf,
  var = "NOM",
  col= "black",
  halo = TRUE,
  overlap = FALSE,
  lines = FALSE
)
mf_scale()
```

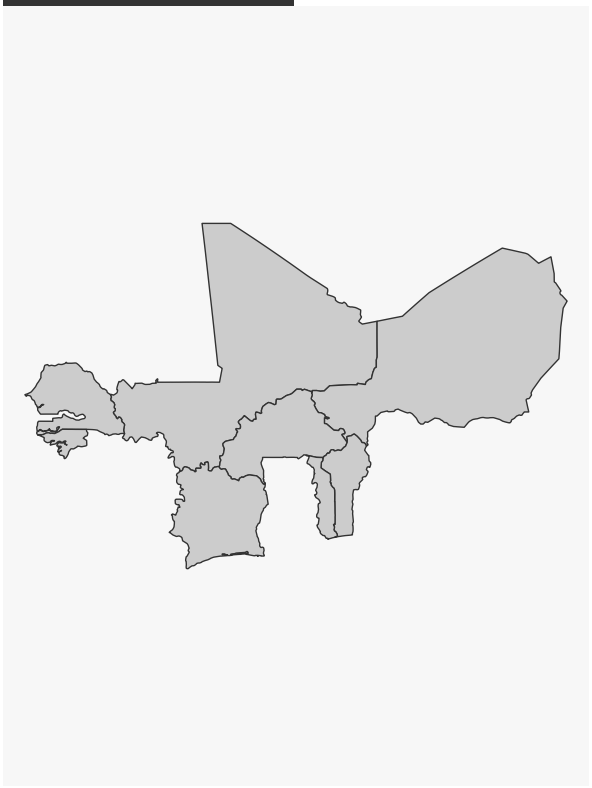


II.3.6. Afficher plusieurs cartes sur la même figure

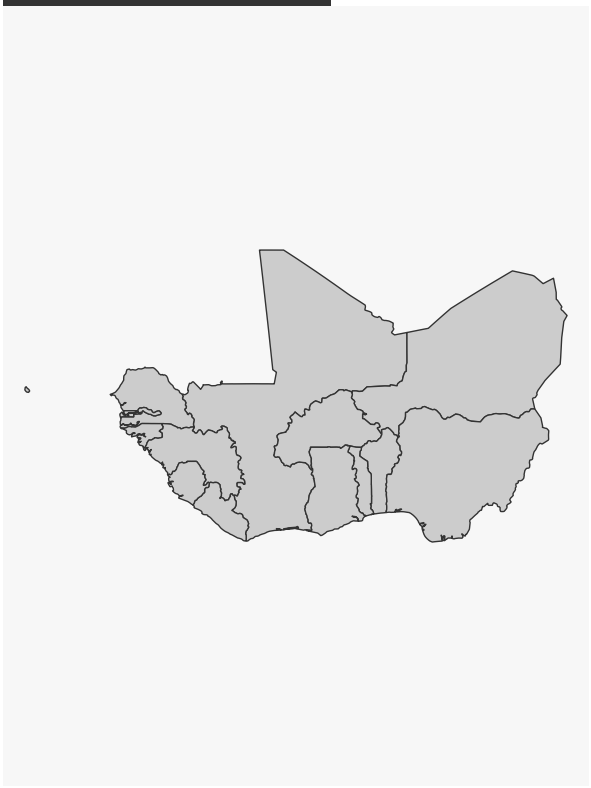
Il faut ici utiliser l'argument mfrow de la fonction *par()*. Le premier chiffre représente le nombre lignes et le deuxième le nombre de colonnes

```
par(mfrow = c(1, 2))
# first map
mf_map(uemoa_sf)
mf_title("Carte de l'UEMOA")
# second map
mf_map(cedeao_sf)
mf_title("Carte de la CEDEAO")
```

Carte de l'UEMOA



Carte de la CEDEAO



II.4. Manipulations des données de type raster

Voir script R

III. Etude de cas

Voir script R