Introduction à la cartographie avec **R**

Thibault Laurent

Mise à jour : 31 décembre 2021

Contents

Ce document a été généré directement depuis **RStudio** en utilisant l’outil Markdown. La version *.pdf* se trouve ici.

Packages à installer :

```
install.packages(c(
  "cartography", # réaliser des cartes
  "maptiles", # importer des fonds de carte
  "classInt", # discrétisation de variables quantitatives
  "geogrid", # permet de transformer un polygone en hexagone
  "ggspatial", # syntaxe complémentaires à la ggplot
  "GISTools", # outils pour faire de la carto
  "leaflet", # interractivité avec JavaScript
  "mapview", # interractivité avec JavaScript
  "maptools", # manipulation de données "spatial",
  "OpenStreetMap", # OSM
  "osrm", # openstreetmap avec R
  "raster", # manipulation de données raster
  "terra", # nouvelle version du package raster
  "RColorBrewer", # palette de couleurs pour carto
  "readxls", # import de données XLS
```

```
  "rgdal", # import de données spatiales
  "rgeos", # manipulation de données spatiales
  "sf", # nouvelle classe d'objets spatials
  "sp", # ancienne classe d'objets spatials
  "tidygeocoder", # localiser une adresse
  "tidyverse", # ggplot, dplyr, etc
  "tmaptools" # pour la carto
  ),
                 dependencies = TRUE)
devtools::install_github(repo = 'rCarto/photon') # calcul de distance routière
devtools::install_github("rCarto/popcircle")
```

Données à récupérer : les donnnées peuvent être téléchargées sur ce lien, ou bien directement en faisant :

```
folder <- getwd()
download.file("http://www.thibault.laurent.free.fr/cours/spatial/Donnees.zip",
  destfile = paste0(folder, "/Donnees.zip"))
unzip("Donnees.zip", exdir = paste0(folder, "/Donnees"))
```

1 Introduction

1.1 Objectifs

En analyse de données, on représente la plupart du temps les données sous forme d'un tableau à deux dimensions : la première dimension, de taille n représente l'espace des individus, la seconde de taille p représente l'espace des variables.

Ind. VS Var.	x_1	...	x_p
1	x_{11}	...	x_{1p}
2	x_{21}	...	x_{2p}
⋮	⋮	...	⋮
n	x_{n1}	...	x_{np}

Il arrive de plus en plus que les observations soient géolocalisées. Dans ce cas-là, on associe d'une part une forme géographique (un point, une ligne brisée, un polygonne, un pixel) aux observations et d'autre part des coordonnées géographiques qui permettent de positionner ces formes géographiques sur une carte.

L'analyse statistique de données spatiales est un champ disciplinaire qui a permis de créer des méthodes statistiques propres à ce type de données. Les données spatiales pouvant être de différentes natures, Cressie (1991) a proposé un découpage de la statistique spatiale en trois champs : la géostatistique, l'économétrie spatiale et les processus ponctuels spatiaux.

La cartographie intervient en amont et en parallèle de l'analyse statistique de données spatiales. Elle permet de représenter sur une carte une information statistique (une variable quantitative ou qualitative, les résidus d'une régression, etc.). Une définition est "la cartographie est un art, une science et une technologie pour réaliser des cartes' '. Selon Nicolas Lambert, elle est est à l'intersection de la géomatique (alliance des technologies de l'informatique aux sciences de la Terre) et de l'infographie (pour plus d'informations, voir le post intitulé Géomatique + Infographie = Cartographie https://neocarto.hypotheses.org/2068).

L'objectif de ce chapitre est de permettre de comprendre quel type d'informations est donné dans un fichier de données spatiales (de type shapefile ou GeoJSON), faire des cartes et visualiser sur celles-ci une ou plusieurs informations statistiques.

1.2 Logiciels

Les logiciels qui permettent de faire de la cartographie sont appelés Système d'Information Géographique (SIG). Ils permettent de représenter des cartes, mais aussi de faire du traitement de données spatiales : sélection, aggrégation, fusion, création de nouvelles entités spatiales, passage d'un référentiel géographique à un autre, etc.

On distingue les logiciels libres des logiciels payants dont on présente ici une liste non exhaustive :

- logiciels libres : QGIS (http://www.qgis.org/fr/site/), gvSIG, Kartoza.
- logiciels propriétaires : ArcGIS, MapInfo.

Les SIG présentent de nombreux avantages comme par exemple le traitement de données massives et la possibilité de visualiser celles-ci facilement grâce à des outils de visualisation performants (voir figure 1).

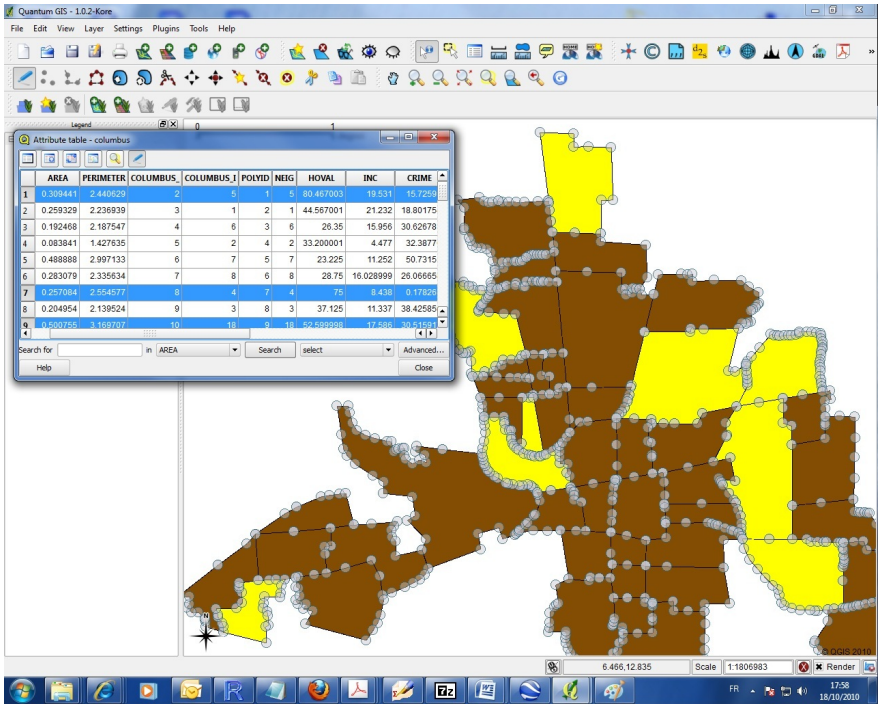


Figure 1: Exemple de SIG.

Par ailleurs, il est très facile de représenter plusieurs sources de données à la fois sous formes de couches ("layer" en anglais) qu'il est facile de faire apparaître ou disparaître en faisant du clique-bouton. Cela ne sera pas aussi simple avec **R**.

En revanche, pour faire de l'analyse statistique de données spatiales, les SIG sont limités et ne présentent pas autant d'outils statistiques que ceux disponibles sur **R**.

Le logiciel **R** permet de faire de la cartographie de base. Bien évidemment, pour de gros volumes de données, il est peut-être plus limité que les SIG mais certains packages développés ces dernières années permettent d'interagir avec des outils externes (telles que Leaflet, https://leafletjs.com/) pour visualiser les données directement à partir d'un explorateur internet plutôt que sur la fenêtre graphique de base.

Par ailleurs, il existe plus d'une centaine de packages dédiés à l'analyse de données spatiales (voir la page web http://cran.r-project.org/web/views/Spatial.html) maintenue par Roger Bivand sur le site du CRAN). **R** est donc un parfait compromis (un mini-SIG et un puissant logiciel d'analyse de données) pour réaliser les différentes étapes de l'analyse statistique de données spatiales.

2 Concepts de base en cartographie

Lorsqu'on importe des fichiers de données spatiales, on peut être confronté au problème présenté dans la figure 2. On possède deux informations : d'une part la localisation d'hypermarchés représententés par des points et d'autre part le potentiel de clients par départements. Ces deux informations sont géolocalisées sur la même zone (la France métropolitaine), mais on constate qu'elles n'ont pas été représentées dans le même référentiel, ceci pouvant s'observer en regardant les échelles sur les axes des abscisses et des ordonnées.

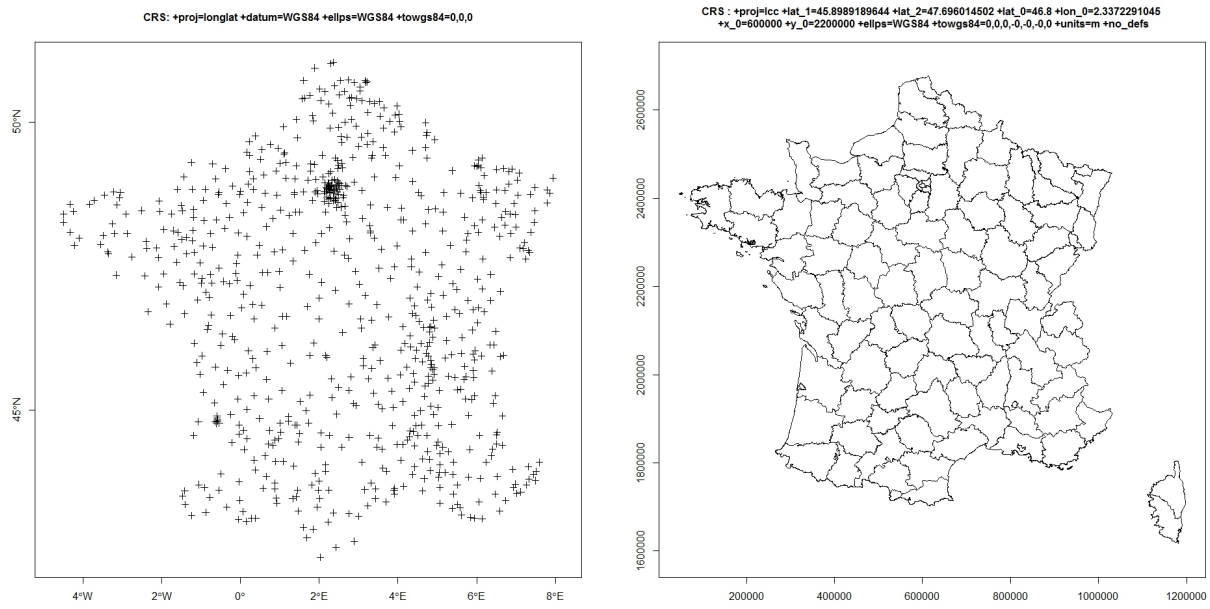


Figure 2: Deux fichiers, deux représentations.

Un des enjeux de la cartographie est de comprendre comment a été choisi le référentiel permettant de dessiner des cartes et comment il est possible de passer d'un référentiel à un autre.

2.1 Système de coodonnées (CRS)

Au-dessus des cartes de la figure 2, on a représenté une légende qui commence par CRS, qui signifie "Coordinate Reference System" ou système de coordonnées en français. Il s'agit d'un ensemble de paramètres (dont nous allons en voir un certain nombres) qui permettent de représenter une information géographique dans un référentiel. Ces paramètres sont résumés à l'aide de plusieurs nomenclatures dont nous nommerons ici trois exemples (voir ce lien pour plus d'informations):

- codage de type PROJ.4

```
+proj=longlat +datum=WGS84 +no_defs
```

- codage de type EPSG

```
"EPSG:4326"
```

- codage de type WKT

```
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
```

```
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    USAGE[
        SCOPE["Horizontal component of 3D system."],
        AREA["World."],
        BBOX[-90,-180,90,180]],
    ID["EPSG",4326]]
```

Depuis 2019, le codage de type PROJ.4 a évolué entraînant des approximations dans les conversions (voir https://rsbivand.github.io/ECS530_h19/ECS530_III.html et https://link.springer.com/content/pdf/10.1007/s10109-020-00336-0.pdf pour plus d'informations), ce qui fait que les deux systèmes à priviligier seront l'EPSG et le WKT.

Si on possède un fichier de données spatial sans une de ces informations, il sera difficile, voire impossible de travailler avec plusieurs sources de données. En effet, il est possible de passer d'un CRS à un autre, faut-il encore savoir comment ils ont été initialement construits. Ceci explique pourquoi beaucoup de nations utilisaient leur propre CRS afin de ne pas partager des informations géographiques secrètes avec les nations ennemies.

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.

Au cours des siècles, de nombreuses représentations de la terre et systèmes de projection ont été proposés. Nous allons définir dans ce paragraphe quels en sont les principes de base et quels sont les CRS les plus utilisés en France et autour.

2.1.1 Datum ou quelle forme géométrique choisir pour la terre

La vraie forme de la terre s'appele le géoïde : il ne s'agit pas d'une sphère car celle-ci est plus aplatie au niveau des deux pôles. Il s'agirait donc plutôt d'un ellipsoïde, mais ce n'est pas non plus le cas car la Terre contient de nombreuses déformations. Le lecteur pourra regarder cette vidéo (https://www.youtube.com/watch?v=H1bzV942LxM) produite par l'IGN (institut national de l'information géographique) afin de se faire une idée d'une bonne représentation de la Terre. La figure 3 représente un modèle de géoïde : les écarts à l'ellipsoïde ont été exagérés à but pédagogique, mais elle permet de comprendre que le géoïde n'est pas une forme géographique simple.

Pour le représenter de façon simplifiée, les cartographes ont utilisé essentiellement les deux formes mathématiques suivantes : la sphère et l'ellipsoïde. Ce dernier est le modèle le plus utilisé car il s'ajuste mieux au géoïde du fait des applatissements au niveau des pôles. Pour définir un ellipsoïde, il s'agit de déterminer le demi petit-axe b et le demi grand-axe a (voir figure 4). Le géoïde diffère de l'ellipsoïde le plus représentatif d'un écart d'une centaine de mètres au maximum.

Le choix d'une forme géographique (associé à un centre) définit le datum ou système géodésique. La plupart des datum utilisé aujourd'hui sont globaux, c'est-à-dire que l'ellipsoïde de référence choisi permet une représentation de la totalité de la surface terrestre.

Pour obtenir la liste des ellipsoïdes connus, on peut faire :

```
rgdal::projInfo("ellps")
```

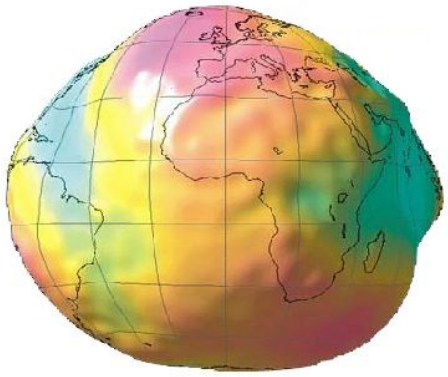


Figure 3: Le géoïde ou la véritable forme de la terre.

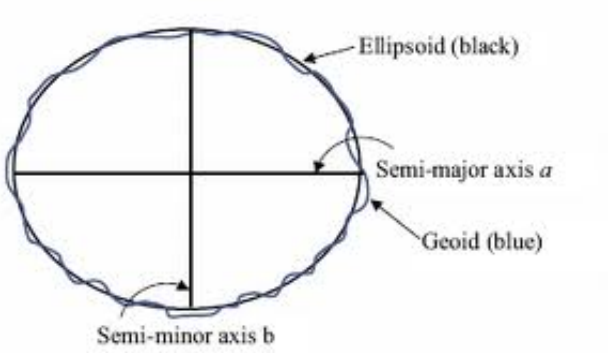


Figure 4: Ajustement d'un ellipsoïde au géoïde.

```
##         name          major                 ell
## 1      MERIT    a=6378137.0           rf=298.257
## 2      SGS85    a=6378136.0           rf=298.257
## 3      GRS80    a=6378137.0     rf=298.257222101
## 4      IAU76    a=6378140.0           rf=298.257
## 5       airy  a=6377563.396       rf=299.3249646
## 6     APL4.9    a=6378137.0            rf=298.25
## 7      NWL9D    a=6378145.0            rf=298.25
## 8   mod_airy  a=6377340.189        b=6356034.446
## 9     andrae   a=6377104.43             rf=300.0
## 10    danish a=6377019.2563             rf=300.0
## 11   aust_SA    a=6378160.0            rf=298.25
## 12     GRS67    a=6378160.0    rf=298.2471674270
## 13   GSK2011    a=6378136.5       rf=298.2564151
## 14    bessel  a=6377397.155       rf=299.1528128
## 15  bess_nam  a=6377483.865       rf=299.1528128
## 16    clrk66    a=6378206.4          b=6356583.8
## 17    clrk80  a=6378249.145          rf=293.4663
## 18 clrk80ign    a=6378249.2 rf=293.4660212936269
## 19       CPM    a=6375738.7            rf=334.29
## 20    delmbr     a=6376428.             rf=311.5
## 21   engelis   a=6378136.05          rf=298.2566
## 22   evrst30  a=6377276.345          rf=300.8017
## 23   evrst48  a=6377304.063          rf=300.8017
## 24   evrst56  a=6377301.243          rf=300.8017
## 25   evrst69  a=6377295.664          rf=300.8017
## 26   evrstSS  a=6377298.556          rf=300.8017
## 27   fschr60     a=6378166.             rf=298.3
## 28  fschr60m     a=6378155.             rf=298.3
## 29   fschr68     a=6378150.             rf=298.3
## 30   helmert     a=6378200.             rf=298.3
## 31     hough    a=6378270.0              rf=297.
## 32      intl    a=6378388.0              rf=297.
## 33     krass    a=6378245.0            rf=298.3
## 34     kaula     a=6378163.           rf=298.24
## 35     lerch     a=6378139.          rf=298.257
## 36     mprts     a=6397300.             rf=191.
## 37  new_intl    a=6378157.5          b=6356772.2
## 38   plessis     a=6376523.           b=6355863.
## 39      PZ90    a=6378136.0         rf=298.25784
## 40    SEasia    a=6378155.0       b=6356773.3205
## 41   walbeck    a=6376896.0       b=6355834.8467
## 42     WGS60    a=6378165.0             rf=298.3
## 43     WGS66    a=6378145.0            rf=298.25
## 44     WGS72    a=6378135.0            rf=298.26
## 45     WGS84    a=6378137.0     rf=298.257223563
## 46    sphere    a=6370997.0          b=6370997.0
##                         description
## 1                        MERIT 1983
## 2         Soviet Geodetic System 85
## 3              GRS 1980(IUGG, 1980)
## 4                          IAU 1976
## 5                         Airy 1830
## 6               Appl. Physics. 1965
```

```
## 7         Naval Weapons Lab., 1965
## 8                   Modified Airy
## 9       Andrae 1876 (Den., Iclnd.)
## 10  Andrae 1876 (Denmark, Iceland)
## 11 Australian Natl & S. Amer. 1969
## 12               GRS 67(IUGG 1967)
## 13                       GSK-2011
## 14                     Bessel 1841
## 15           Bessel 1841 (Namibia)
## 16                     Clarke 1866
## 17                Clarke 1880 mod.
## 18              Clarke 1880 (IGN).
## 19 Comm. des Poids et Mesures 1799
## 20         Delambre 1810 (Belgium)
## 21                     Engelis 1985
## 22                     Everest 1830
## 23                     Everest 1948
## 24                     Everest 1956
## 25                     Everest 1969
## 26       Everest (Sabah & Sarawak)
## 27    Fischer (Mercury Datum) 1960
## 28           Modified Fischer 1960
## 29                     Fischer 1968
## 30                     Helmert 1906
## 31                           Hough
## 32    International 1909 (Hayford)
## 33                Krassovsky, 1942
## 34                      Kaula 1961
## 35                      Lerch 1979
## 36                Maupertius 1738
## 37          New International 1967
## 38           Plessis 1817 (France)
## 39                          PZ-90
## 40                 Southeast Asia
## 41                        Walbeck
## 42                         WGS 60
## 43                         WGS 66
## 44                         WGS 72
## 45                         WGS 84
## 46       Normal Sphere (r=6370997)
```

Remarque : dans le tableau ci-dessus, b n'est pas représenté, mais c'est l'inverse de f qui est représenté à la place où f vaut $f = \frac{a-b}{a}$

2.1.1.1 Datum global Dans le cas d'un datum global, le centre de l'ellipsoïde correspond au centre du géoïde, soit le centre de masse de la terre. Avec des choix particuliers de a et b, on caractérise des datum. Dans la figure 5, on a représenté les deux ellipsoïdes les plus utilisés : l'ellipsoïde Clarke 1880 et l'ellipsoïde WGS 84 (World Geodesic System 84). Ce dernier est utilisé dans le GPS, système de positionnement à couverture mondiale, ainsi que dans Google Maps.

2.1.1.2 Datum local Dans le cas d'un datum local, la surface de l'ellipsoïde est tangeante à celle du géoïde en un point fondamental qui permet de donner une représentation fidèle de cette partie de la surface de la terre. La précision diminue au fur et à mesure que l'on s'éloigne du point fondamental. Le système géodésique local est défini par l'ellipsoïde d'une part, et son positionnement par rapport au WGS84 (voir

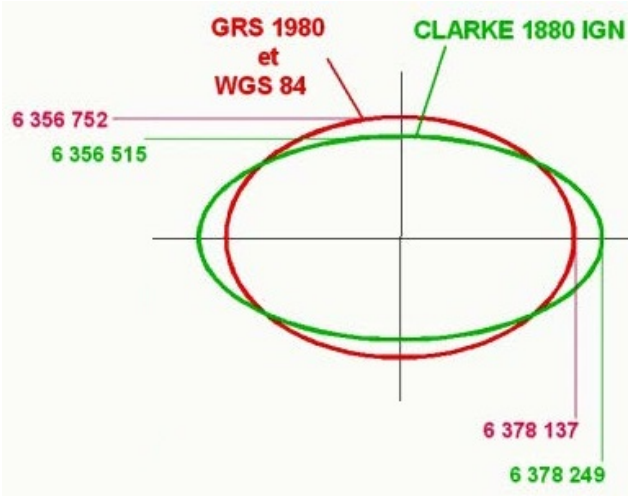


Figure 5: Les datum Clarke 1880 et WGS 84.

figure 6). Dans ce cas, il s'agira de déterminer également des paramètres tels que Dx, Dy, Dz, Rx, Ry et Rz.

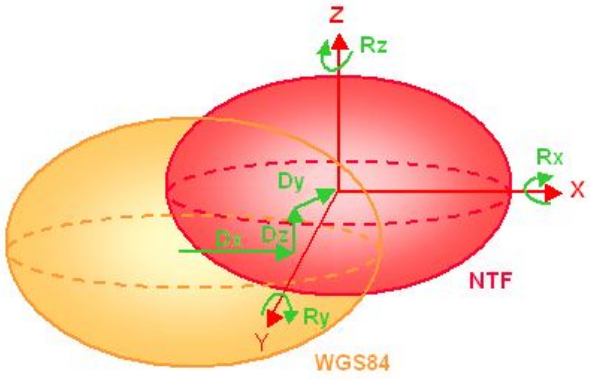


Figure 6: Exemple de datum local.

Par exemple, le système géodésique NTF (Nouvelle Triangulation de France, ancien système officiel en France) utilise l'ellipsoïde Clarke 1880 ($a = 6\ 378\ 249$m et $f = 1/293.45$) dont le centre est décalé du centre du géoïde avec les mesures $Dx = +168m$, $Dy = +60m$, $Dz = -320m$, $Rx = 0°$, $Ry = 0°$, $Rz = 0°$. Dans un système géodésique local, il est également d'usage d'indiquer le point fondamental où l'ellipsoïde tangente le géoïde. Dans le NTF, il s'agit de la croix du Panthéon.

2.1.1.3 Aujourd'hui en France Le datum de référence en France est le RGF93 (même si le NTF n'a pas été complètement abandonné) dont l'ellipsoïde est appelé IAG GRS80. Ce datum est quasiment identique au datum WGS 84. En effet, le RGF93 est compatible avec le WGS 84 pour des précisions égales ou supérieures à 10m.

Il s'agit donc d'un système géodésique global, compatible avec le système géodésique ETRS 89 (European Terrestrial Reference System 1989), qui est le système officiel européen.

2.1.1.4 Nomenclature Avec les éléments que nous venons de définir, il est déjà possible de comprendre la nomenclature utilisée pour définir un CRS. Par exemple, dans un fichier de données spatiales qui a utilisé le datum WGS 84, on devrait retrouver les informations suivantes dans la nomenclature WKT :

```
DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]]
```

ou alors dans la nomenclature PROJ.4 :

```
+datum=WGS84 +ellps=WGS84
```

Dans le cas du datum NTF, on devrait retrouver les informations suivantes qui définissent les valeurs de a et $1/f$, ainsi que les paramètres qui permettent de translater le centre par rapport au centre de référence :

```
DATUM["Nouvelle_Triangulation_Francaise_Paris",
            SPHEROID["Clarke 1880 (IGN)",6378249.2,293.4660212936265,
                AUTHORITY["EPSG","7011"]],
            TOWGS84[-168,-60,320,0,0,0,0],
                AUTHORITY["EPSG","6807"]]
```

ou alors en PROJ.4

```
+a=6378249.2 +b=6356515.000000472 +towgs84=-168,-60,320,-0,-0,-0,0
```

Il existe de nombreux autres paramètres qui permettent de définir un CRS (pour plus de renseignements, le lecteur pourra consulter cette documentation https://inbo.github.io/tutorials/tutorials/spatial_crs_coding/).

A présent qu'on a défini une forme géométrique pour la Terre, il faut définir un système de coordonnées géographiques qui permettent de localiser un objet sur celle-ci.

2.1.2 Système de coordonnées géographiques

La localisation d'un élément à la surface de la terre peut s'exprimer sous la forme de coordonnées géographiques (voir figure 7). Les coordonnées sont alors déclinées à l'aide de deux valeurs angulaires : λ (longitude) et ϕ (latitude).

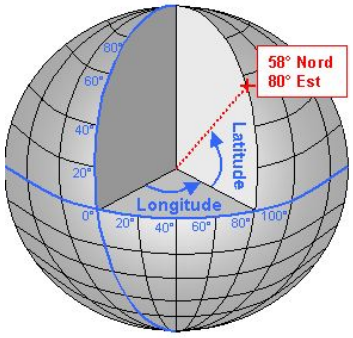


Figure 7: Exemple de datum local.

Le plus souvent, le méridien de Greenwich est choisi comme origine, vers l'Est on attribue les valeurs positives (de 0 à 180°) et vers l'Ouest, les valeurs négatives (de 0 à -180°). L'équateur est pris comme latitude d'origine, avec des valeurs positives vers le Nord (de 0 à 90°) et des valeurs négatives vers le Sud (de 0 à -90°). Ces deux angles peuvent être exprimés dans différentes unités : degrés-minutes-secondes, degrès-décimaux, radians, grades. L'unité de base est le degré d'angle (1 tour complet = 360°), puis la minute d'angle (1° = 60'), puis la seconde d'angle (1° = 3 600'').

Comme tous les outils informatiques, les SIG travaillent de préférence avec des coordonnées en degrés décimaux. On rappelle que la conversion de degrés-minutes-secondes à degrès-décimaux se fait ainsi : $10°50'59''$ est égal à 10.849722°. En effet, 59 secondes est l'équivalent de $\frac{59}{60}$ minute et $50 + \frac{59}{60}$ minutes est égal à $(50 + \frac{59}{60})/60 = 0.8497222$ degré décimal.

Remarque 1 : des coordonnées géographiques n'ont aucun sens si on ne précise pas dans quel Datum ils sont exprimés. Par exemple, un même point peut être repéré par les coordonnées (7°44′16″ Est, 48°36′00″ Nord) dans le Datum NTF et (7°44′12″ Est, 48°35′59″ Nord) dans le Datum WGS84.

Remarque 2 : pour calculer des distances kilométriques entre deux points exprimés en coordonnées angulaires, il faut faire appel à des formules trigonométriques. Pour simplifier les calculs, on fait très souvent l'hypothèse que la forme géométrique choisie pour représenter la Terre est une sphère de rayon $R = 6378$Km. La distance entre A (λ_A, ϕ_A) et B (λ_A, ϕ_A) est alors égale à :

$$d_{A-B} = R \arccos(\sin\phi_A \sin\phi_B + \cos\phi_A \cos\phi_B \cos(\lambda_A - \lambda_B))$$

Exercice 1

Calculer la distance kilométrique entre Paris ($\lambda_A = 2°21'07''$, $\phi_A = 48°51'24''$) et Toulouse ($\lambda_B = 1°26'03''$, $\phi_B = 43°36'16''$) en utilisant l'approximation ci-dessus. Attention, les fonctions *cos()* et *sin()* prennent des angles en radians.

2.1.2.1 Récupérer des coordonnées angulaires à partir d'adresse postale

Il est possible de récupérer sous **R** des coordonnées angulaires à partir d'adresse postales. C'est ce que fait la fonction *geocode()* du package **photon** (Giraud, 2017) disponible depuis Github. Cette fonction permet d'interroger la base de données OpenStreetMap (OSM). Sur Windows, il semble que JAVA soit nécessaire (voir ce lien pour installer JAVA)

```
require("photon")
```

```
## Le chargement a nécessité le package : photon
```

```
address <- c("21 allee de Brienne, 31000 Toulouse, France",
             "2 Rue du Doyen Gabriel Marty, 31042 Toulouse")
locgeo.full <- photon::geocode(address, limit = 1, key = "place")
locgeo.full
```

```
##                                      location     osm_id osm_type name
## 1  21 allee de Brienne, 31000 Toulouse, France 1328054192        N <NA>
## 2 2 Rue du Doyen Gabriel Marty, 31042 Toulouse 1454237612        N <NA>
##   housenumber                      street postcode     city     state country
## 1          21           Allée de Brienne    31000 Toulouse Occitanie  France
## 2           2 Rue du Doyen Gabriel Marty    31000 Toulouse Occitanie  France
##   osm_key osm_value      lon      lat  msg
## 1   place     house 1.431661 43.60560 <NA>
## 2   place     house 1.438197 43.60614 <NA>
```

On peut vérifier avec l'application **leaflet** que les coordonnées trouvées sont les bons :

```
library(leaflet)
m <- leaflet() %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addMarkers(lng = locgeo.full$lon, lat = locgeo.full$lat, popup = "MT")
m
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pleas
```

Si l'installation de **photon** pose problème, une alternative est le package **tidygeocoder** (Cambon et al. 2021) qui propose également de trouver une adresse à partir de coordonnées géographiques (voir https://cran.r-project.org/web/packages/tidygeocoder/vignettes/tidygeocoder.html pour plus d'informations)

```
osm_s1 <- tidygeocoder::geo(
     address = address, method = "osm",
```

```
    lat = latitude, long = longitude
)
```

```
## Passing 2 addresses to the Nominatim single address geocoder
```

```
## Query completed in: 2.3 seconds
```

Exercice 2

Retrouver les coordonnées angulaires de la place Compans Cafarelli à Toulouse et du centre INRAE à Auzeville.

2.1.2.2 Représentation des coordonnées angulaires

Beaucoup de cartes représentent directement les coordonnées angulaires dans un plan à deux dimensions. Autrement dit, l'axe des abscisses représente les longitudes et l'axe des ordonnées les latitudes (voir figure 8). Cette représentation, bien que souvent utilisée pour des raisons de simplification, présente énormément d'inconvénients : la principale est la déformation des formes et des aires, surtout pour les zones proches des pôles.

Par exemple, le Groënland, qui paraît aussi vaste que l'Amérique du Sud a en réalité approximativement la même taille que le Mexique. Avec une superficie avoisinant les 2.000.0000 km^2 pour les deux pays. Pour des pays proches de l'Equateur (et même pour la France qui est pourtant sur le 45ème parallèle Nord), les déformations sont minimes, ce qui explique que ce type de réprésentation est courante. Cependant, tous les instituts géographiques officiels utilisent des représentations beaucoup plus "justes" et pour cela, il s'agit de définir des systèmes de projection afin de transformer les coordonnées angulaires en coordonnées planes. Certaines projections permettent ainsi de conserver les angles (pour la navigation), de lire correctement les distances kilométriques (pour les randonnées), etc.

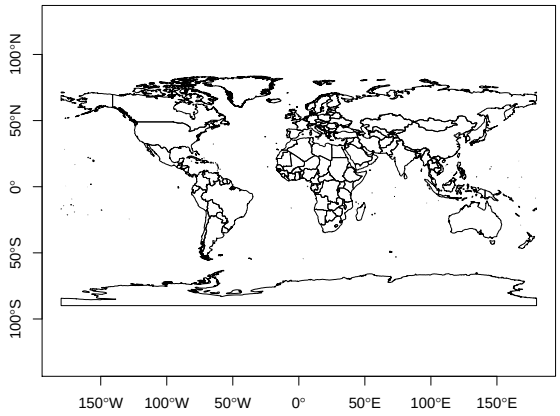


Figure 8: Exemple de représentation globale

2.1.3 Système de projection

Pour représenter des objets en 3D sur un plan, chaque point de la surface terrestre est d'abord projeté sur l'ellipsoïde selon la direction normale. Puis l'ellipsoïde est transformé en surface plane, en faisant des projections de l'ellipsoïde sur des formes géométriques qui peuvent être facilement représentées en 2D.

Une projection est donc un procédé permettant de représenter à plat un ellipsoïde. Il en existe plus de 200 qui portent des noms différents et aucune n'est absolument exacte car il n'est pas possible de cartographier la terre sans la déformer. Le lecteur peut consulter cette vidéo (https://www.youtube.com/watch?v=kIID5FDi2JQ) qui explique pourquoi aucune représentation 2D de la Terre n'est correcte.

Aussi, le choix d'une projection dépend de ce que l'on veut représenter. Par exemple, est-ce qu'on souhaite représenter une carte avec une étendue grande ? La région est-elle polaire ? Souhaite-t-on conserver les angles pour utiliser la carte à des fins de navigation ? Souhaite-on conserver les aires des pays ? Pour ces raisons, dans un même pays, on peut avoir recours à plusieurs projections différentes selon l'utilisation de la carte que l'on souhaite faire.

Mathématiquement, une projection revient à convertir les coordonnées géographiques (λ, ϕ) par des coordonnées cartésiennes ($x = f(\lambda, \phi)$, $y = g(\lambda, \phi)$), où f et g varient selon la projection utilisée.

Géométriquement, nous allons voir qu'il s'agit de projeter le globe sur des surfaces particulières (des cylindres, des cônes, un plan, etc.). Nous présentons ici différents types de projections et montrons comment il est possible de passer d'un système de projection à un autre avec **R**.

2.1.3.1 Projection cylindrique L'idée de ce type de projection est représentée dans la figure 9 à gauche. On suppose qu'on a représenté sur l'ellipsoïde les objets qui nous intéressent, par exemple les contours administratifs des pays. On inscrit ensuite dans un cylindre l'ellipsoïde tangent à une ellipse de telle sorte que les zones proches de la tangente auront une représentation plus précise. On projette ensuite à partir du centre de la Terre les objets localisées sur l'ellipsoïde (ici le point P_1 et le cercle C) sur le cylindre (ici P_1' et C'). Il est ensuite facile de déplier le cylindre de telle sorte qu'on obtienne une surface plane.

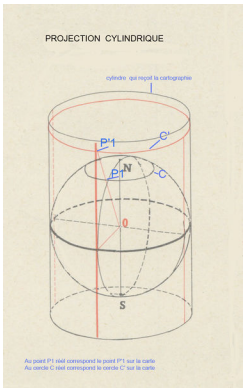


Figure 9: Projection cylindrique (vue globale)

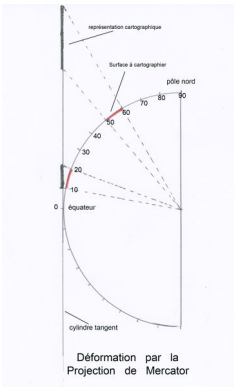


Figure 10: Projection cylindrique (vue globale)

Dans la figure ci-dessus, on constate que la tangente correspond à l'Equateur. Par ailleurs, on voit que ce type de projection déforme les distances; en effet, sur la figure 10, les deux segments rouges sur l'ellipsoïde ont la même longueur, mais leurs projections sur le cylindre ne sont pas de la même taille.

Si au lieu de prendre comme tangente un parallèle, on choisit un méridien, on obtient alors une projection

dite UTM. En fonction de la position sur l'ellipsoïde de la zone qu'on souhaite représenter, on va considérer comme tangente un méridien plutôt qu'un autre. Le monde a ainsi été découpé en 60 zones UTM et la France se trouve sur les zones 30, 31 et 32 (voir figure 11). Ce type de projection est utilisée par l'IGN sur les cartes de randonnée notamment. Dans ce cas-là, la nomenclature utilisée pour définir le CRS aura la forme suivante :

```
sf::st_crs("+proj=utm +zone=31 +datum=WGS84 +units=m +no_defs")
```

```
## Coordinate Reference System:
##   User input: +proj=utm +zone=31 +datum=WGS84 +units=m +no_defs
##   wkt:
## PROJCRS["unknown",
##     BASEGEOGCRS["unknown",
##         DATUM["World Geodetic System 1984",
##             ELLIPSOID["WGS 84",6378137,298.257223563,
##                 LENGTHUNIT["metre",1]],
##             ID["EPSG",6326]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8901]]],
##     CONVERSION["UTM zone 31N",
##         METHOD["Transverse Mercator",
##             ID["EPSG",9807]],
##         PARAMETER["Latitude of natural origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",3,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8802]],
##         PARAMETER["Scale factor at natural origin",0.9996,
##             SCALEUNIT["unity",1],
##             ID["EPSG",8805]],
##         PARAMETER["False easting",500000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8807]],
##         ID["EPSG",16031]],
##     CS[Cartesian,2],
##         AXIS["(E)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1,
##                 ID["EPSG",9001]]],
##         AXIS["(N)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1,
##                 ID["EPSG",9001]]]]
```

D'un point de vue mathématique, les transformations qui permettent de passer de coordonnées angulaires à des coordonnées planes sont données par des formules dont des exemples sont trouvés sur la page Wikipedia (https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system).

Remarque : Googlemaps utilise par défaut une projection de Mercator qui est une projection cylindrique conforme, c'est-à-dire qu'elle conserve les angles (plus précisément les angles conformes, autrement dit les parallèles et les méridiens se coupent à angle droit). En revanche, cette projection ne respecte pas du tout

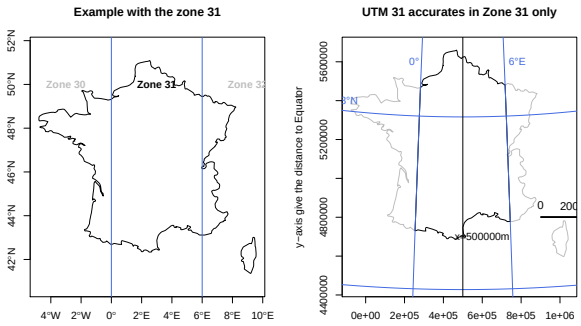


Figure 11: Projection UTM en France utilisée sur les cartes de l'IGN

les distances et donc la superficie des états (voir https://sciencepost.fr/google-maps-abandonne-enfin-la-projection-cartographique-de-mercator/), ce qui explique pourquoi depuis 2018, Google Maps propose une autre projection (cliquer sur la forme de la terre en bas à droite de la page Google Maps).

2.1.3.2 Projection conique En France, on associe généralement au datum RGF93, la projection conique appelée "Lambert 93' '. Cette projection a l'intérêt de concilier les formes, les dimensions et les distances réelles. La figure 11 illustre son mécanisme. On inscrit dans un cône l'ellipsoïde tangent à une ellipse (ici le parallèle 45) de telle sorte que les zones proches de cette tangente auront une représentation plus précise. On projette ensuite à partir du centre de la Terre les objets localisées sur l'ellipsoïde (ici le point P_1 et les cercles C_1 et C_2) sur le cône (ici P_1', C_1' et C_2'). On déplie ensuie le cône pour obtenir une surface plane.

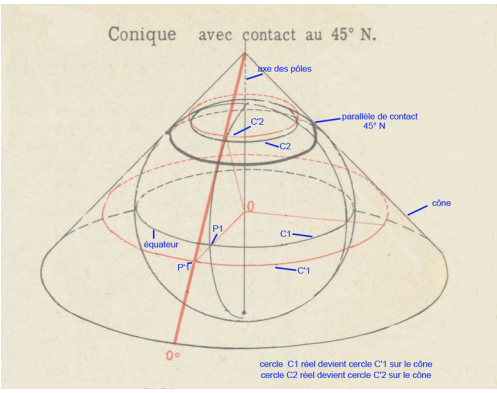


Figure 12: Projection conique (vue globale et tronquée)

Un exemple de projection conique est la projection dite de Albers représentée dans la figure ci-dessous. Il s'agit d'une alternative à la projection Mercator dans le but d'une représentation globale de la Terre. Dans le cas particulier de cette projection, le cône sectionne l'ellipsoïde; il y a donc deux tangentes. Cette projection préserve globalement très bien les formes et les aires au niveau des zones proches des tangentes. La nomenclature utilisée pour cette projection renseigne les latitudes hautes, basses et médiane (**+lat_1**, **+lat_2**, **+lat_0**) et se code de la façon suivante :

```
sf::st_crs("+proj=aea +lat_1=29.83 +lat_2=45.83 +lat_0=37.5
     +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs")
```

```
## Coordinate Reference System:
##   User input: +proj=aea +lat_1=29.83 +lat_2=45.83 +lat_0=37.5
##      +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs
```

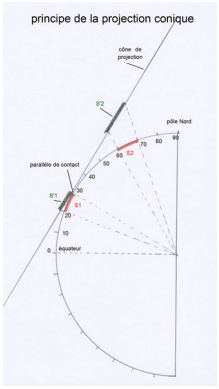


Figure 13: Projection conique (vue globale et tronquée)

```
##   wkt:
## PROJCRS["unknown",
##     BASEGEOGCRS["unknown",
##         DATUM["World Geodetic System 1984",
##             ELLIPSOID["WGS 84",6378137,298.257223563,
##                 LENGTHUNIT["metre",1]],
##             ID["EPSG",6326]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8901]]],
##     CONVERSION["unknown",
##         METHOD["Albers Equal Area",
##             ID["EPSG",9822]],
##         PARAMETER["Latitude of false origin",37.5,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8821]],
##         PARAMETER["Longitude of false origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8822]],
##         PARAMETER["Latitude of 1st standard parallel",29.83,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8823]],
##         PARAMETER["Latitude of 2nd standard parallel",45.83,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8824]],
##         PARAMETER["Easting at false origin",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8826]],
##         PARAMETER["Northing at false origin",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8827]]],
##     CS[Cartesian,2],
##         AXIS["(E)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1,
##                 ID["EPSG",9001]]],
##         AXIS["(N)",north,
##             ORDER[2],
```

```
##             LENGTHUNIT["metre",1,
##                 ID["EPSG",9001]]]]
```

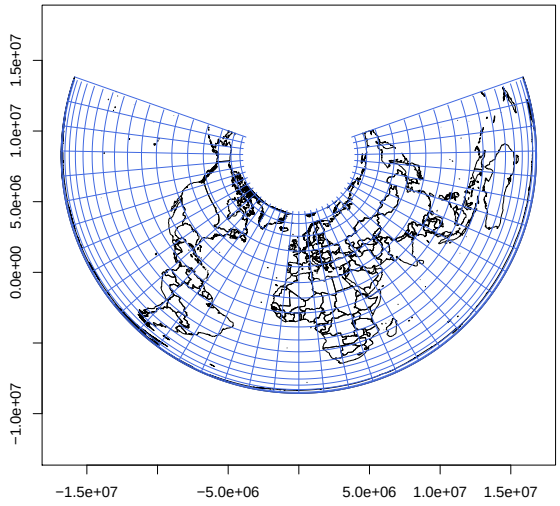


Figure 14: Projection Albers

En France, on utilise également une projection conique conforme (conservation des angles localement) appelée projection conique conforme de Lambert. Elle a pour parallèles de référence les parallèles 44 et 49, la latitude de référence étant prise égale à 44. La longitude de référence est le 3ème méridien ouest (**+lon_0**). Les coordonnées sur la surface plane de la longitude et latitude de référence seront égales à **+x_0=700000** et **+y_0=6600000**.

```
sf::st_crs("+proj=lcc +lat_1=49 +lat_2=44 +lat_0=46.5
     +lon_0=3 +x_0=700000 +y_0=6600000 +ellps=GRS80 +units=m +no_defs")
```

```
## Coordinate Reference System:
##   User input: +proj=lcc +lat_1=49 +lat_2=44 +lat_0=46.5
##      +lon_0=3 +x_0=700000 +y_0=6600000 +ellps=GRS80 +units=m +no_defs
##   wkt:
## PROJCRS["unknown",
##     BASEGEOGCRS["unknown",
##         DATUM["Unknown based on GRS80 ellipsoid",
##             ELLIPSOID["GRS 1980",6378137,298.257222101,
##                 LENGTHUNIT["metre",1],
##                 ID["EPSG",7019]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8901]]],
##     CONVERSION["unknown",
##         METHOD["Lambert Conic Conformal (2SP)",
##             ID["EPSG",9802]],
##         PARAMETER["Latitude of false origin",46.5,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8821]],
##         PARAMETER["Longitude of false origin",3,
```

```
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8822]],
##         PARAMETER["Latitude of 1st standard parallel",49,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8823]],
##         PARAMETER["Latitude of 2nd standard parallel",44,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8824]],
##         PARAMETER["Easting at false origin",700000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8826]],
##         PARAMETER["Northing at false origin",6600000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8827]]],
##     CS[Cartesian,2],
##         AXIS["(E)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1,
##                 ID["EPSG",9001]]],
##         AXIS["(N)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1,
##                 ID["EPSG",9001]]]]
```

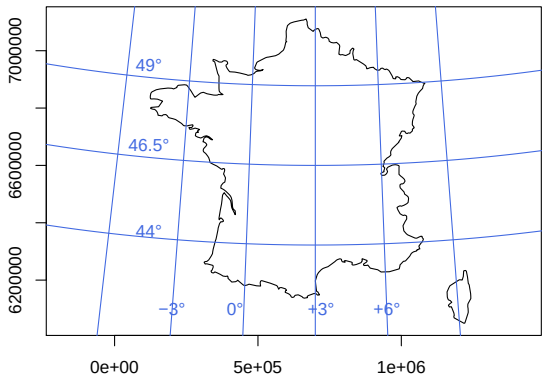


Figure 15: Projection Lambert 93

2.1.3.3 Projection azimuthale Pour limiter la déformation au niveau des pôles, les projections azimuthales sont très appréciées des cartographes. Par exemple, pour représenter l'Antarctique, on utilise une projection azimuthale de type stéréographique (voir https://fr.wikipedia.org/wiki/Projection_st%C3%A9r%C3%A9ographique pour plus de renseignements.) dont la figure 14 illustre le mécanisme pour 4 points A', B', C' et D' projetés depuis un point d'origine localisé ici sur la surface (mais pouvant aussi être au-dessus) de la Terre, sur un plan situé en-dessous de la Terre.

La figure 15 montre un exemple de projection azimuthale globale de la Terre, qui fait penser au type de représentation utilisée par Google Earth et Google maps en mode sphère.

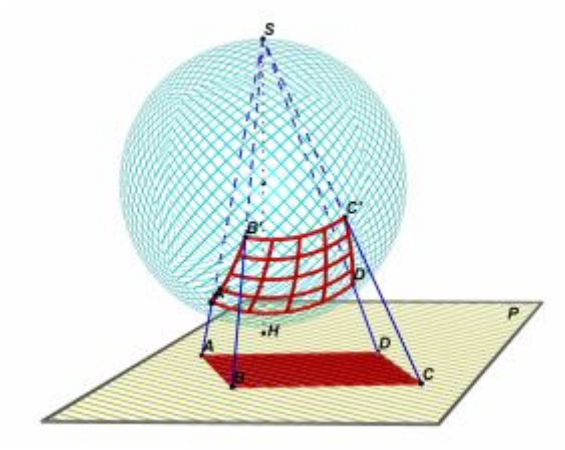


Figure 16: Projection azimuthale

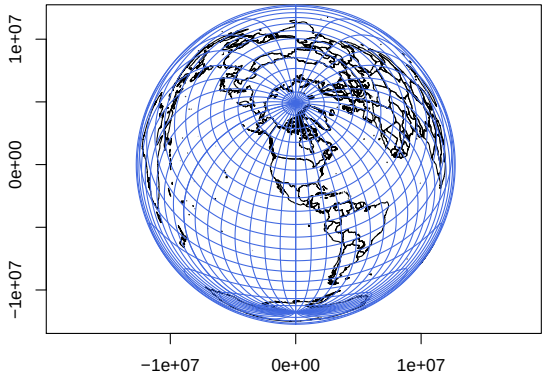


Figure 17: Exemple de projection azimuthale : US National Atlas Equal Area

2.1.4 Comment choisir un CRS ?

Revenons au problème posé dans la figure 2, où deux fichiers de données provenant de deux sources différentes utilisaient deux CRS différents (on rappelle que le CRS contient l'information sur le datum d'une part et sur la projection utilisée d'autre part). Dans ce cas-là, quel CRS devrait-on choisir ? Il n'y a pas de vérités générales, mais l'idée est de faire en sorte que les cartes que l'on souhaite créer puissent si possible être reproduites par le plus grand nombre. Dans ce cas, il vaut mieux utiliser celui en vigueur dans la zone étudiée. Par exemple, pour savoir quel CRS est utilisé en France, on peut consulter le site suivant : http://magrit.cnrs.fr/docs/projection_list_fr.html qui a recensé par pays les CRS adaptés à un pays donné (pour une liste plus complète de CRS, le lecteur pourra consulter cette page : http://www.spatialreference.org/).

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 :

```
sf::st_crs("EPSG:2154")
```

```
## Coordinate Reference System:
##   User input: EPSG:2154
##   wkt:
## PROJCRS["RGF93 / Lambert-93",
##     BASEGEOGCRS["RGF93",
##         DATUM["Reseau Geodesique Francais 1993",
##             ELLIPSOID["GRS 1980",6378137,298.257222101,
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4171]],
##     CONVERSION["Lambert-93",
##         METHOD["Lambert Conic Conformal (2SP)",
##             ID["EPSG",9802]],
##         PARAMETER["Latitude of false origin",46.5,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8821]],
##         PARAMETER["Longitude of false origin",3,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8822]],
##         PARAMETER["Latitude of 1st standard parallel",49,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8823]],
##         PARAMETER["Latitude of 2nd standard parallel",44,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8824]],
##         PARAMETER["Easting at false origin",700000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8826]],
##         PARAMETER["Northing at false origin",6600000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8827]]],
##     CS[Cartesian,2],
##         AXIS["easting (X)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["northing (Y)",north,
##             ORDER[2],
```

```
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["Engineering survey, topographic mapping."],
##         AREA["France - onshore and offshore, mainland and Corsica."],
##         BBOX[41.15,-9.86,51.56,10.38]],
##     ID["EPSG",2154]]
```

En France, c'est donc le CRS dont le code EPSG vaut 2154 qu'on aura tendance à adopter. Le site *http://www.spatialreference.org/* répertorie la plupart des CRS qui sont utilisés dans le monde, en fonction de la région étudiée. Cette information est directement accessible sur **R** avec la fonction *make_EPSG()* du package **rgdal**.

Voici un exemple d'utilisation. On stocke d'abord l'ensemble des CRS disponibles :

```
EPSG <- rgdal::make_EPSG()
```

Il faut ensuite faire une recherche par mots-clés pour trouver la liste des CRS disponibles par pays :

```
EPSG[grep("RGF93", EPSG$note), 1:2]
```

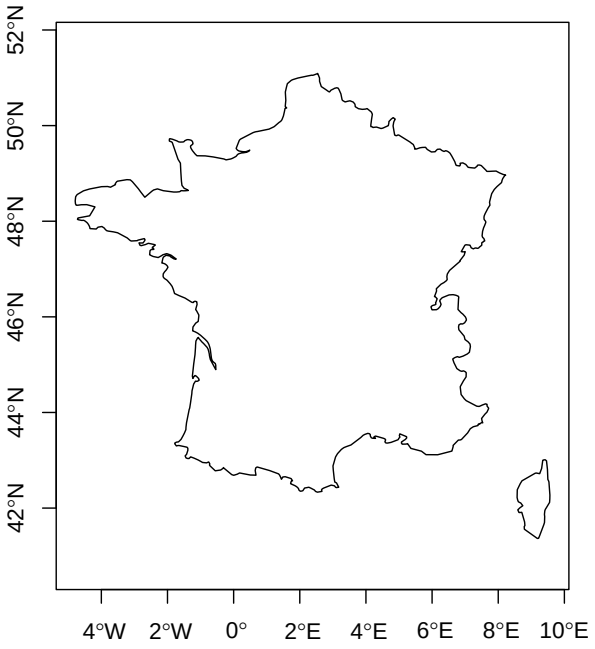
```
##      code                                note
## 116  4171                               RGF93
## 304  4369                        RGF93 (3D)
## 305  4370                 RGF93 (geocentric)
## 603  4964                               RGF93
## 604  4965                               RGF93
## 760  7042                   RGF93 (lon-lat)
## 764  7084                   RGF93 (lon-lat)
## 1355 2154                 RGF93 / Lambert-93
## 4374 3942                       RGF93 / CC42
## 4375 3943                       RGF93 / CC43
## 4376 3944                       RGF93 / CC44
## 4377 3945                       RGF93 / CC45
## 4378 3946                       RGF93 / CC46
## 4379 3947                       RGF93 / CC47
## 4380 3948                       RGF93 / CC48
## 4381 3949                       RGF93 / CC49
## 4382 3950                       RGF93 / CC50
## 6437 5698 RGF93 / Lambert-93 + NGF-IGN69 height
## 6438 5699 RGF93 / Lambert-93 + NGF-IGN78 height
```

Exercice 3

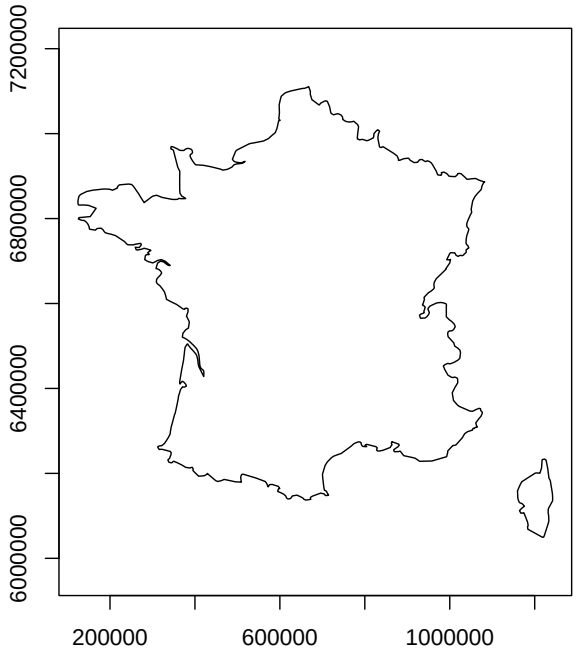
Quelle représentation vous semble la mieux appropriée et pourquoi ?

Proposition 1

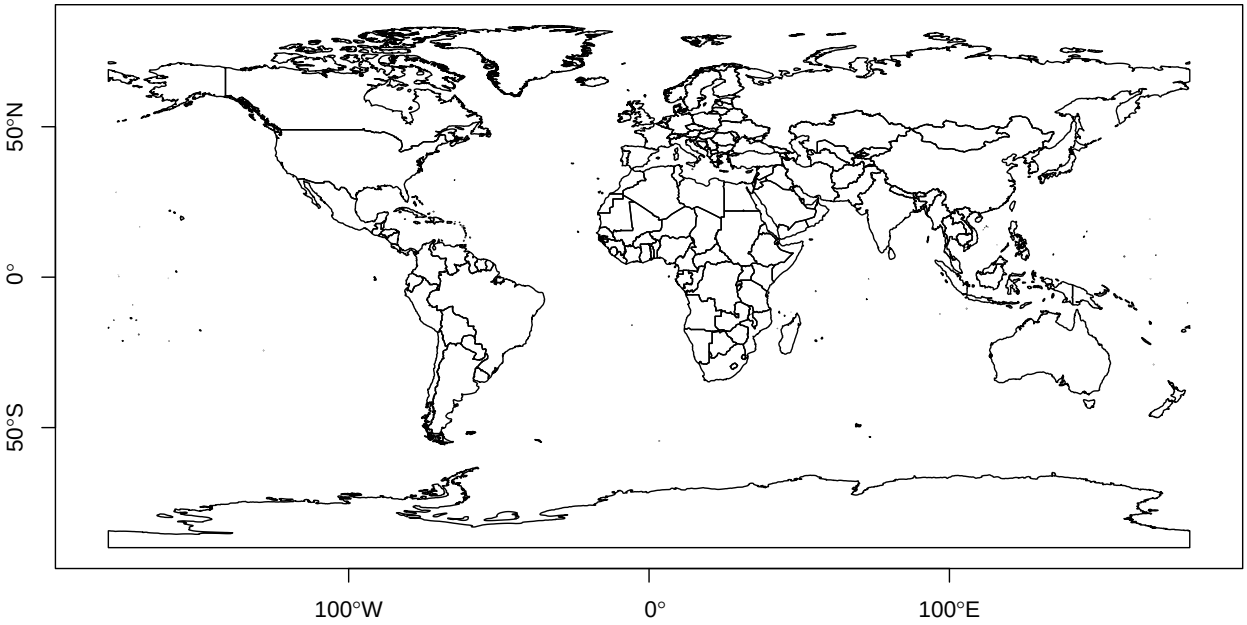
EPSG:4326



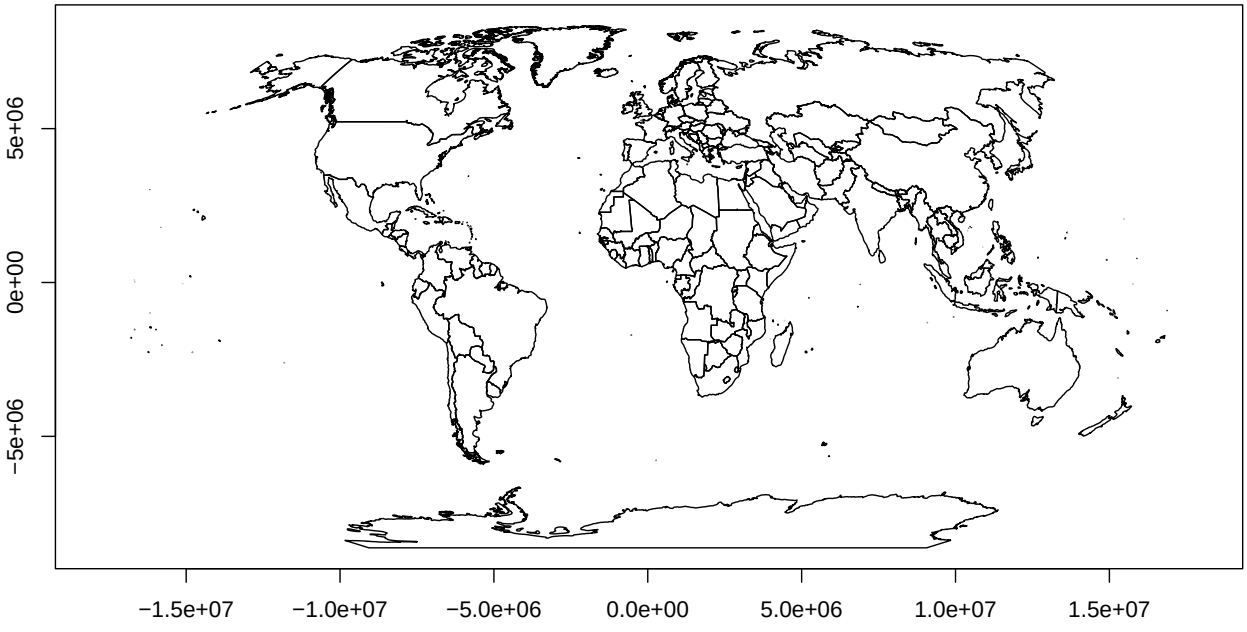
EPSG:2154

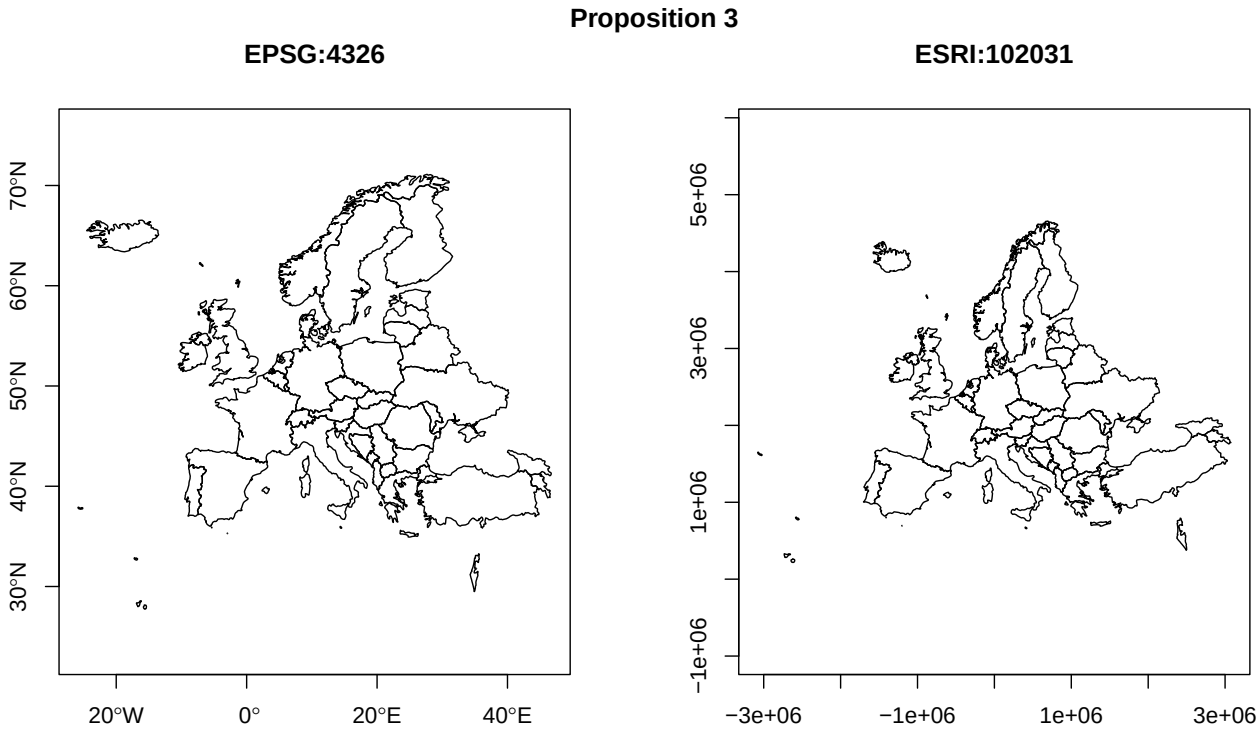


Proposition 2
EPSG:4326



ESRI:54030





2.2 Fichiers de données spatiales

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. La figure 16 représente une même entité géographique dans l'un ou l'autre de ces formats.

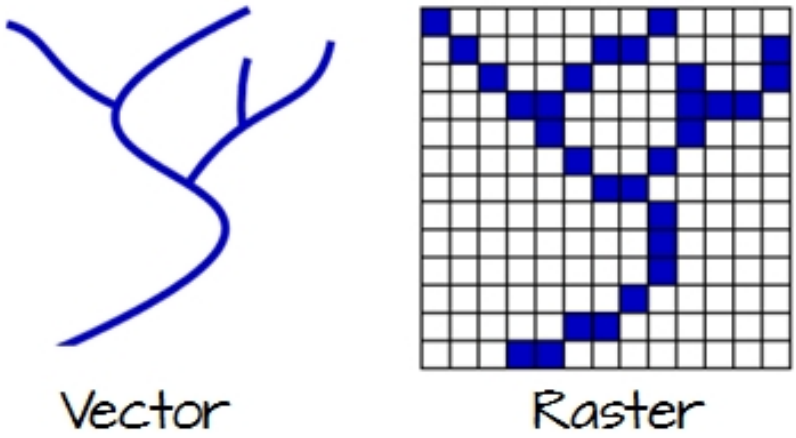


Figure 18: Vecteur VS raster

Selon le mode, les données sont stockées dans des formats différents. Nous allons décrire ici succinctement ces deux modes. Pour une revue plus complète de ces fichiers, on renvoit le lecteur au livre numérique écrit par Gimont (2019).

2.2.1 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 la figure 16, le code couleur associé aux pixels correspond au degré d'ensoleillement moyen observé sur un certain laps de temps.

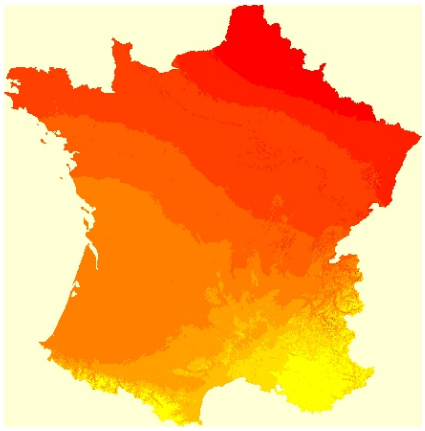
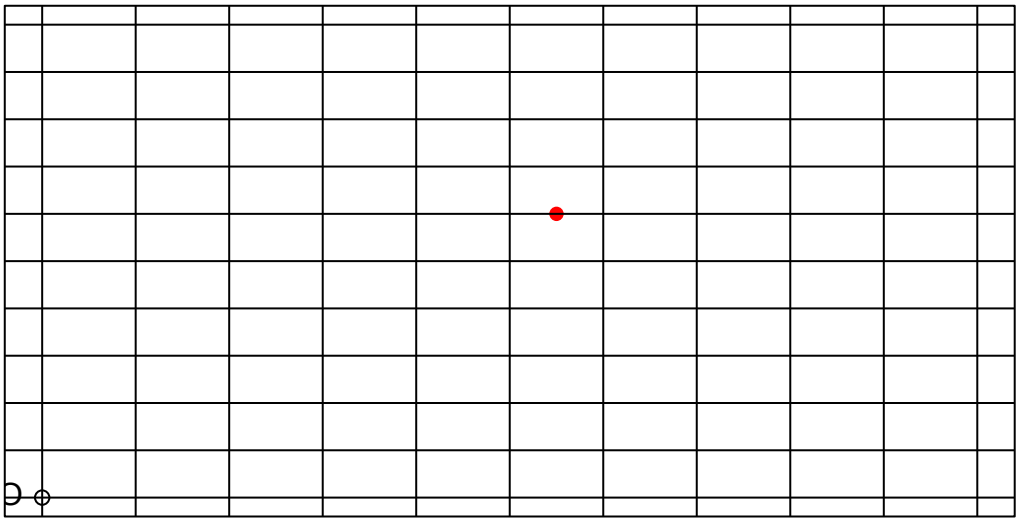


Figure 19: Exemple de fichier raster

Comme il s'agit de données spatiales, les pixels sont associés à des coordonnées géographiques. Le fichier doit donc contenir les informations concernant le géoréférencement (comme le CRS). 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. Le traitement d'images est également souvent contraint à des temps de calcul long. C'est pourquoi une zone géographique peut-être découpée en plusieurs sous-images, pour diminuer la taille des fichiers et essayer de gagner en temps calcul dans le traitement de l'information.

Exercice 4

Quelle est la localisation géographique du point rouge sachant que les coordonnées du point O sont $(10000, 15000)$ et que la taille d'un carreau vaut $500 \times 500m$?



2.2.2 Mode vecteur

Le format vectoriel utilise le concept d'objets géométriques (points, lignes, polygones) ou "Spatial Features'' pour représenter les entités géographiques. Ces objets géométriques sont définis par leurs coordonnées dans un système de projection. La figure 17 représente dans une même figure ces trois types d'objets.

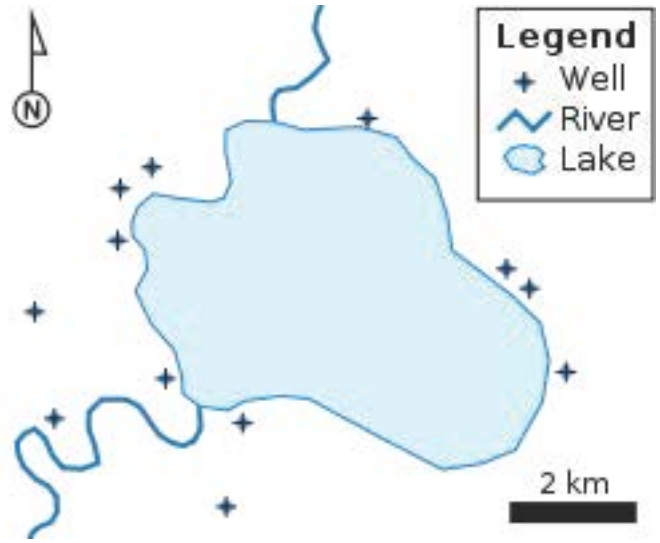


Figure 20: Exemple de fichier vecteur

Il existe plusieurs formats possibles pour stocker des données spatiales. Nous allons décrire ici les deux principaux : **Shapefile** et **GeoJSON**, sachant que ces deux formats sont a priori reconnus par tous les SIG et peuvent être importés sous **R**.

2.2.2.1 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.

2.2.2.2 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 dont on pourra trouver une description à ce lien. 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.

```
{"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
        ]}
    }]
}
```

Exemple de fichiers geojson : on va récupérer deux fonds de carte des pays du monde sur la page Github de l'équipe RIATE.

```
download.file(url = "https://raw.githubusercontent.com/riatelab/basemaps/master/World/countries.geojson
              destfile = "Donnees/country.geojson")
download.file(url = "https://raw.githubusercontent.com/riatelab/basemaps/master/World/graticule30.geojs
              destfile = "Donnees/graticule.geojson")
```

2.2.3 Ou trouver des fichiers de données spatiales "Open Data"

On présente ici 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 du cours.

- 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
- La version "données spatiales" de data.gouv.fr : https://geo.data.gouv.fr/fr/
- Open Spatial Demographic Data and Research : https://www.worldpop.org/
- données d'usage des sols au niveau européen : https://www.eea.europa.eu/data-and-maps/data/corine-land-cover-accounting-layers

3 Lecture et manipulations de données spatiales avec R

3.1 Passage de données non spatiales à des données de classe "Spatial"

Commençons par importer des données classiques, c'est-à-dire dans un format non spatial, par exemple au format texte *.csv*. Les données sont téléchargeables à ce lien. Il s'agit de données sur les tremblements de terre observés entre 1973 et 2009 sur Terre (source : https://earthquake.usgs.gov/earthquakes/search/). Pour chaque tremblement de terre, on observe la localisation géographique (variables **Latitude** et **Longitude**) et différentes mesures telles que la **Magnitude**. Notez que la localisation géographique d'une observation est représentée par un point. 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 le cas de données de type vectoriel, on utilisera les packages **sp** (Pebesma et Bivand, 2005) et **sf** (Pebesma, 2018). 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** (Hijmans, 2019), qui devrait être remplacé petit à petit par le package **terra** (Hijmans, 2021).

Jusqu'en 2018, la classe **sp** qui était de type **S4** (pour plus d'informations sur la classe **S4**, on renvoit le lecteur au chapitre 2 de Déjean et Laurent, 2019) était la plus utilisée. Depuis, l'univers Tidyverse initié par H. Wickham a connu un succès immense et la classe **sf** permet d'intégrer les concepts de données bien présentées ("tidy") ainsi que la syntaxe du piping %>% (Wickham et al. 2019).

Dans la suite, nous présenterons les deux solutions (**sp** et **sf**) simultatément, mais en pratique s'il fallait n'en retenir qu'une seule des deux, on recommanderait certainement de conserver la norme **sf**, plus récente et plus adaptée pour être utilisée avec les outils du **tidyverse**.

3.1.1 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). Nous appelerons par la suite la norme "Spatial" ces trois type d'objets.

Il s'agit d'un objet de classe **S4** et 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 1973 1973 ...
##   .. ..$ Month             : int [1:57230] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..$ YYMM              : int [1:57230] 197301 197301 197301 197301 197301 197301 197301 197301 19
##   .. ..$ 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
```

On va essayer de comprendre comment est structuré un tel objet de classe **S4**. 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 (ici **seisme_df**),
- **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 (cf ci-dessous comment on créée un tel objet).

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(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
```

Remarque : parmi les attributs de cette classe d'objets, il y en a un qui s'appelle **proj4string**. Il s'agit du CRS dans lequel sont exprimées les variables de géolocalisation. Il est fortement recommandé de renseigner cette information quand cela est possible. Pour cela, on utilise la fonction *proj4string()* ainsi que la fonction

CRS() dans laquelle on va renseigner la nomenclature standard qui permet de définir un CRS. Dans le cas de ces données de tremblements de terre, les coordonnées sont représentées en coordonnées géographiques (longitude/latitude) et le datum utilisé est le WGS 84. Ce CRS correspond au numéro 4326 dans la nomenclature EPSG. L'option recommandé est donc d'utiliser cette information de la façon suivante :

```
proj4string(seisme_sp) <- CRS(SRS_string = "EPSG:4326")
```

Dans les autres façons de faire, on aurait pu écrire directement la nomenclature en PROJ.4 mais comme expliqué dans la section précédente, cette façon de procéder peut conduire à des messages d'avertissement; on montre néanmoins comment faire dans le cas où on n'aurait pas d'autres possibilités de définir un CRS :

```
proj4string(seisme_sp) <- CRS("+proj=longlat +datum=WGS84 +no_defs
                              +ellps=WGS84 +towgs84=0,0,0")
# équivalent à cette ligne de commande
proj4string(seisme_sp) <- CRS("+proj=longlat +datum=WGS84 +no_defs")
```

3.1.2 La 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 :

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

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

Remarque : en chargeant la librairie **sf**, on constate que celle-ci fait appel à trois bibliothèques de codes **GEOS**, **GDAL** et **PROJ**, qui sont intégrées dans la plupart des SIG et incluent des fonctions pour manipuler des objets spatiaux, lire des données géospatiales, passer d'un CRS à un autre, etc. Pour installer le package **sf** sous Linux ou Mac, il faut au préalable installer des bibliothèques de code (voir la page suivante : https://github.com/r-spatial/sf et/ou https://rtask.thinkr.fr/fr/installation-de-r-4-0-sur-ubuntu-20-04-lts-et-astuces-pour-les-packages-de-cartographie/).

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)
```

Dans le cas où les données sont des points, les géométries sont caractérisées par des classes d'objets directement inspirées de ce qu'il se fait avec les fichiers **GeoJSON**, aussi connues sont le nom de "Simple feature geometry". Les différentes géométries ou "simple feature geometry" sont représentées dans la figure 19. Il s'agit des géométries 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.

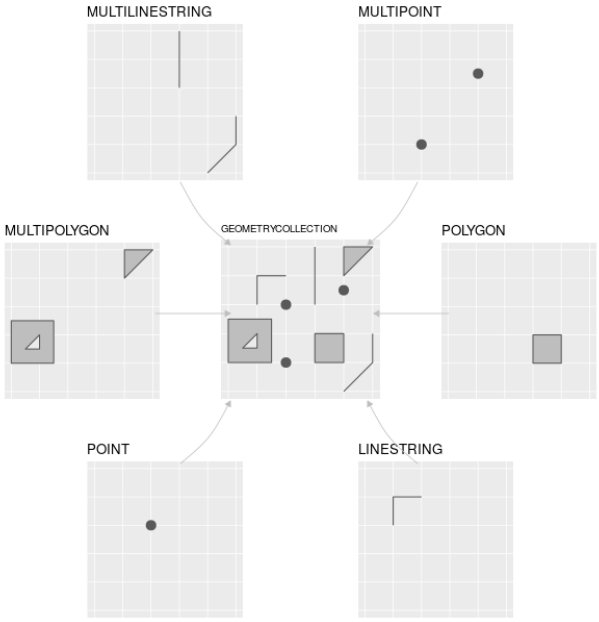


Figure 21: Simple feature geometry

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

```
head(st_drop_geometry(seisme_sf), 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
```

soit en utilisant la syntaxe **dplyr** :

```
seisme_sf %>%
  st_drop_geometry() %>%
  head(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()* et on affecte le numéro d'EPSG sous forme d'**integer** :

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

Le package **leaflet** (Cheng et al., 2019) permet d'ouvrir les fonds de carte OpenStreetMap sur une interface web en JavaScript et d'y superposer des données spatiales. Ca peut être une très bonne occasion pour vérifier que l'import s'est bien passé.

```
library(leaflet)
m <- leaflet(seisme_sf) %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addCircles()
m
```

Exercice 5

Télécharger les données suivantes issues de la World Urbanization Prospects des Nations Unies qui correspondent aux villes avec plus de 3 millions d'habitants dans le monde :

```
# Population data base
download.file(url = "https://population.un.org/wup/Download/Files/WUP2018-F12-Cities_Over_300K.xls",
              destfile = "Donnees/population_mondiale/citypop.xls")
# Growth rate data base
download.file(url = "https://population.un.org/wup/Download/Files/WUP2018-F14-Growth_Rate_Cities.xls",
              destfile = "Donnees/population_mondiale/cityevo.xls")
```

Importer les données avec le package **readxl** et faire la jointure :

```
library(readxl)
citypop <- data.frame(read_excel("Donnees/population_mondiale/citypop.xls", skip = 16))
cityevo <- data.frame(read_excel("Donnees/population_mondiale/cityevo.xls", skip = 16))
city <- merge(x = citypop, y = cityevo[, c(4,9:24)], by = "City.Code")
```

Créer un objet de type spatial à partir du jeu de données **city** en utilisant les normes **sp** et/ou **sf**. Vérifier que l'import s'est bien passé en utilisant l'outil leaflet.

3.2 Importation de fichiers de données spatiales

3.2.1 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**

3.2.1.1 Classe sp On peut utiliser la fonction *readOGR()* incluse dans la package **rgdal**, qui permet également d'importer d'autres types de données spatiales comme celles au format **GeoJSON**, ou bien encore celles issues du logiciel MapInfo (extension **.TAB** ou **.MIF**/**.MID**). Pour charger ce package sous Linux ou Mac, l'utilisateur devra installer les mêmes bibliothèques de codes que celles nécessaires pour charger le package **sf** (voir https://github.com/r-spatial/sf).

Cette fonction est intéressante car elle définit automatiquement le CRS quand cette information est disponible (dans le cas d'un fichier **Shapefile**, il faut que le fichier **.prj** existe). L'argument **dsn** indique le chemin du répertoire où se trouve le fichier de données spatiales, dont on renseigne le nom dans l'argument **layer** :

```
library(rgdal)
```

```
## Please note that rgdal will be retired by the end of 2023,
## plan transition to sf/stars/terra functions using GDAL and PROJ
## at your earliest convenience.
##
```

```
## rgdal: version: 1.5-27, (SVN revision 1148)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 3.3.2, released 2021/09/01
## Path to GDAL shared files: /usr/share/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ runtime: Rel. 7.2.1, January 1st, 2021, [PJ_VERSION: 721]
## Path to PROJ shared files: /home/thibault/.local/share/proj:/usr/share/proj
## PROJ CDN enabled: FALSE
## Linking to sp version:1.4-6
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,
## use options("rgdal_show_exportToProj4_warnings"="none") before loading sp or rgdal.
```

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

```
## OGR data source with driver: ESRI Shapefile
## Source: "/media/thibault/My Passport/course/cours_eco_spatiale/chapitre_2/Donnees/World WGS84", laye
## 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\",\n        ELL
```

```
##   ..$ comment: chr "TRUE"
```

On retrouve pratiquement les mêmes attributs que pour un objet **SpatialPointsDataFrame** exceptés les attributs **polygons** et **plotOrder**. Ce dernier indique l'ordre dans lequel les polygones seront dessinés. L'attribut **polygons** décrit quand à lui comment est structuré l'objet géographique qui définit l'unité statistique. 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-ci est constitué de 5 attributs :

- **Polygons** : une liste contenant les coordonnées d'1 ou plusieurs **Polygon** (encore une sous-classe d'objet dont on ne décrira pas ici le contenu mais qui définit les caractéristiques d'un polygone seul). En effet, il peut y avoir plusieurs polygones pour définir une même unité statistique (par exemple un pays contenant des îles sera défini par autant de polygons qu'il y a d'îles),
- **plotOrder** : l'ordre dans lequel les différents **Polygon** seront représentés.
- **labpt** : le centroïde de l'unité statistique (obtenu donc sur le ou les **Polygon** qui caractérisent l'observation statistique),
- **ID** : l'identifiant,
- **area** : l'aire totale de l'unité statistique.

Remarque : on voit bien que le CRS a été correctment importé :

```
cat(wkt(world_sp))
```

```
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["latitude",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["longitude",east,
##             ORDER[2],
```

```
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4326]]
```

3.2.1.2 Classe sf

La plupart des fichiers de données spatiales peuvent s'importer avec la fonction *read_sf()* :

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

```
## Coordinate Reference System:
##   User input: WGS 84
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["latitude",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["longitude",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4326]]
```

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
## # A tibble: 6 x 2
##   NOM                                                              geometry
##   <chr>                                                  <MULTIPOLYGON [°]>
## 1 Baker Island    (((-176.4614 0.2152777, -176.4683 0.2222222, -176.4539 0.2261~
## 2 Howland Island  (((-176.6362 0.7902777, -176.6445 0.7955554, -176.6431 0.8127~
## 3 Johnston Atoll  (((-169.5389 16.72416, -169.5439 16.72638, -169.5375 16.73111~
## 4 Paracel Islands (((112.2714 16.97444, 112.2683 16.98083, 112.2728 16.98611, 1~
## 5 Svalbard        (((27.145 80.00444, 27.10111 79.96748, 27.12138 79.95833, 27.~
## 6 Jan Mayen       (((-9.043091 70.80386, -9.058899 70.80386, -9.073914 70.80775~
```

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"
```

On reconnaît ainsi la classe **MULTIPOLYGON** définie pour caractériser plusieurs polygones.

Exercice 6

Importer un jeu de données spatial de votre choix et afficher le CRS dans lequel sont exprimés les coordonnées géographiques. A défaut, on pourra utiliser les jeux de données iris et velo-toulouse inclus dans le répertoire *Donnees* qu'on appelera respectivement **bike** et **iris**.

3.2.2 Mode raster

On a récupéré des données de vents en France sur le site Global Wind Atlas. Les données sont au format **.tif** géoréférencé. Pour importer ce type de données sous **R**, on va utiliser la package **raster**, optimisé pour le traitement de données d'images. Pour importer les données sous **R**, on utilise la fonction *raster()* (On aurait pu également utiliser la fonction *readOGR()* vue précédemment, mais même si la classe "Spatial" engloble des données raster de type **SpatialGridDataFrame**, celle-ci est beaucoup moins performante pour traiter des données raster) :

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

Pour comprendre comment est constitué un tel objet :

```
wind
```

```
## class      : RasterLayer
## dimensions : 4163, 8188, 34086644  (nrow, ncol, ncell)
## resolution : 0.0025, 0.0025  (x, y)
## extent     : -10.09343, 10.37657, 41.15077, 51.55827  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## source     : FRA_wind-speed_200m.tif
## names      : FRA_wind.speed_200m
```

On peut assimiler ce type d'objet à une matrice pour laquelle les cellules sont géoréférencées. Si on observe plusieurs variables par cellule, dans ce cas on pourra assimiler cet objet à un **array** où la 3ème dimension permet d'accéder aux variables.

- la matrice est de taille 4163×8188.
- la taille d'un carreau est 0.0025×0.0025; l'unité est le degré décimal car les données sont exprimés en longitude/latitude, compte tenu que dans l'attribut **crs**, on peut voir que **+proj=longlat**
- la fenêtre d'observation est donnée par l'attribut **extent**.

Pour accéder aux valeurs, on procède comme avec une matrice ou un **array** (si plusieurs variables sont observées). Dans notre exemple, il n'y a qu'une variable observée donc les deux façons sont possibles :

```
wind[2000:2001, 4000:4002, 1]
```

```
## [1] 9.285499 9.276102 9.276102 9.258187 9.244025 9.258982
```

```
wind[2000:2001, 4000:4002]
```

```
## [1] 9.285499 9.276102 9.276102 9.258187 9.244025 9.258982
```

En principe, c'est possible d'utiliser également l'outil **leaflet** à condition que l'image ne soit pas trop volumineuse (ici, la taille est trop grande et le code suivant devrait créer un message d'erreur)

```
leaflet() %>% addTiles() %>%
  addRasterImage(wind)
```

Cependant, il est possible desélectionner une sous-image avec la fonction `crop()`? Cette dernière utilise la fonction `extend()` qui sélectionne le bounding box (minimum et maximum de la longitude; miminum et maximum de la latitude)

```
e <- extent(1, 2, 43, 44)
re <- crop(wind, e)
leaflet() %>% addTiles() %>%
  addRasterImage(re)
```

```
## Warning in showSRID(uprojargs, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded ellps WGS 84 in Proj4 definition: +proj=merc +a=6378137
## +b=6378137 +lat_ts=0 +lon_0=0 +x_0=0 +y_0=0 +k=1 +units=m +nadgrids=@null
## +wktext +no_defs +type=crs
```

```
## Warning in showSRID(uprojargs, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded datum World Geodetic System 1984 in Proj4 definition
```

Exercice 7

Importer un jeu de données de type raster de votre choix. A défaut, on pourra importer le jeu de données "fra_pd_2020_1km.tif" dans le répertoire *Donnees*, issue de World Pop. Essayer d'afficher ce jeu de données avec leaflet.

3.3 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] [                    [[<-                 $<-
##  [4] aggregate            as.data.frame        cbind
##  [7] coerce               dbDataType           dbWriteTable
## [10] extent               extract              filter
## [13] identify             initialize           mask
## [16] merge                plot                 print
## [19] raster               rasterize            rbind
## [22] select               show                 slotsFromS3
## [25] st_agr               st_agr<-             st_area
## [28] st_as_s2             st_as_sf             st_bbox
## [31] st_boundary          st_buffer            st_cast
## [34] st_centroid          st_collection_extract st_convex_hull
## [37] st_coordinates       st_crop              st_crs
## [40] st_crs<-             st_difference        st_filter
## [43] st_geometry          st_geometry<-        st_inscribed_circle
## [46] st_interpolate_aw    st_intersection      st_intersects
```

```
## [49] st_is_valid           st_is                 st_join
## [52] st_line_merge         st_m_range            st_make_valid
## [55] st_nearest_points     st_node               st_normalize
## [58] st_point_on_surface   st_polygonize         st_precision
## [61] st_reverse            st_sample             st_segmentize
## [64] st_set_precision      st_shift_longitude    st_simplify
## [67] st_snap               st_sym_difference     st_transform
## [70] st_triangulate        st_union              st_voronoi
## [73] st_wrap_dateline      st_write              st_z_range
## [76] st_zm                 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
```

Nous allons par la suite présenter certaines d'entre elles pour les classes **sp** et **sf** (on reviendra sur le traitement de données **raster** par la suite).

3.3.1 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)
```

Remarque : cette commande n'est pas possible pour un objet de classe **sf** car celui-ci est dérivé de la classe **tibble** qui ne recommande pas l'usage de **row.names** pour donner un nom aux observations; au lieu de cela, il est recommandé d'utiliser une colonne qui serve de clé unique.

3.3.2 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** :

```
maghreb_sp <- world_sp[c("Algeria", "Morocco",
                         "Libya", "Mauritania",
                         "Tunisia", "Western Sahara"),]
```

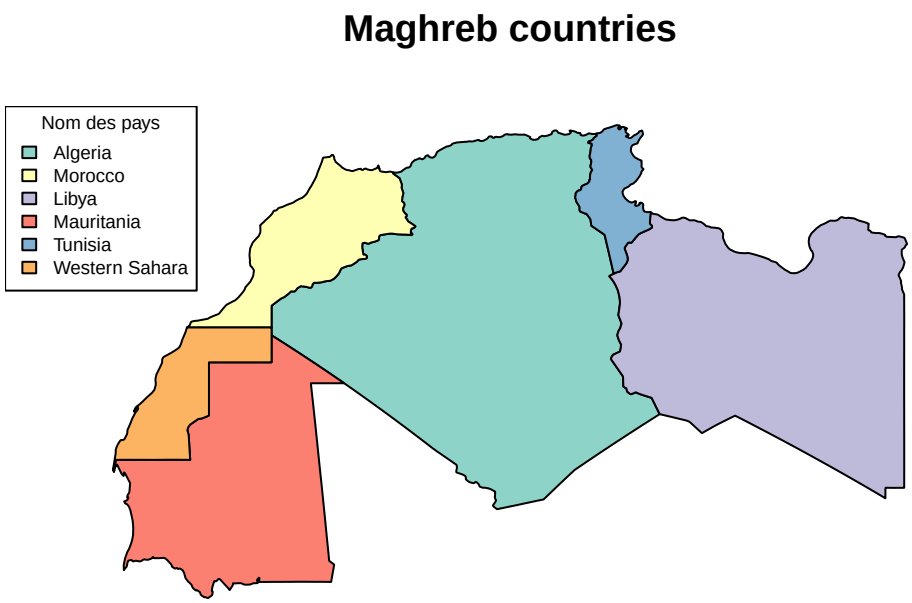
Pour la classe **sf**, on peut utiliser la même syntaxe que pour **sp**, mais en plus, on peut utiliser la syntaxe à la mode **dplyr** :

```
library(tidyverse)
maghreb_sf <- world_sf %>%
  filter(NOM %in% c("Algeria", "Morocco", "Libya",
 "Mauritania", "Tunisia", "Western Sahara"))
```

3.3.3 Représentations de cartes de base

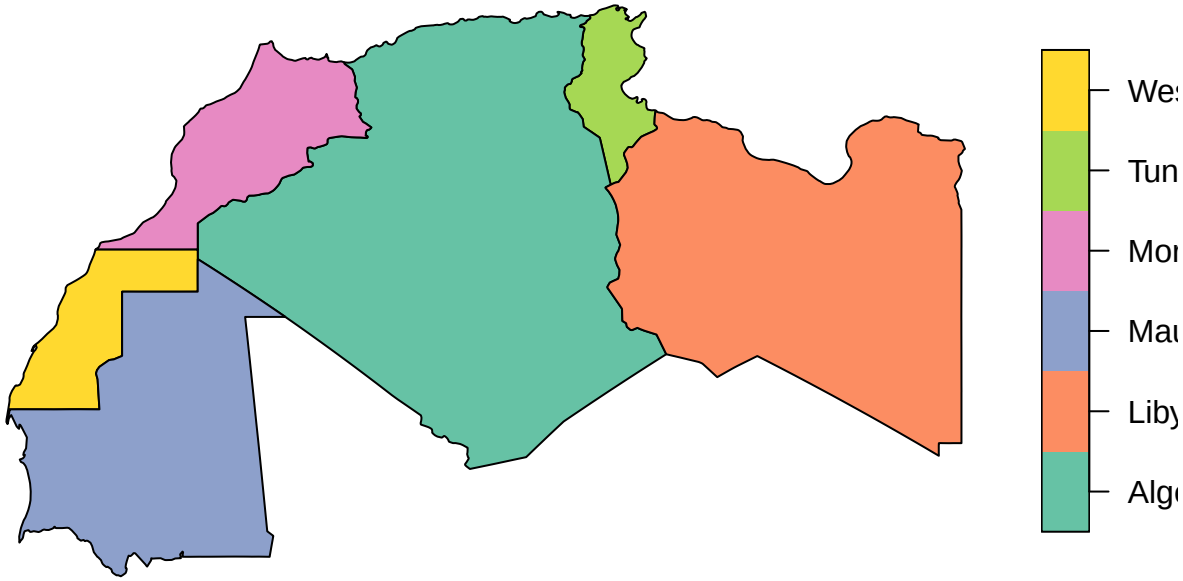
On utilise la fonction `plot()` qui appliquée à un objet "Spatial" va seulement représenter la géométrie de l'objet. On peut ensuite utiliser les fonctions graphiques de base (`title()`, `legend()`, etc.) pour ornementer le graphique.

```
plot(maghreb_sp, col = RColorBrewer::brewer.pal(6, "Set3"))
title("Maghreb countries")
legend("topleft", legend = row.names(maghreb_sp), cex = 0.6,
 title = "Nom des pays", fill = RColorBrewer::brewer.pal(6, "Set3"))
```



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(maghreb_sf, main = "Maghreb countries")
```



Exercice 8

Représenter les géométries spatiales du jeu de données que vous avez choisi en utilisant une des solutions présentées ci-dessus. A défaut, représenter on représentera le contour des **iris** de la ville de Toulouse.

3.3.4 Aggrégation de données spatiales

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. Par exemple, on suppose qu'ou souhaite aggréger le "Maroc" et le "Sahara occidental" et on considère la variable densité de population. La première vaut 80 habitants / km^2 et la seconde vaut 3 habitants / km^2. Pour créer la variable densité de population sur la nouvelle unité spatiale, on sera obligé ici de pondérer par l'aire de chaque pays.

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 "Maroc" et le "Sahara occidental", sachant que le Maroc et le Sahara Occidental sont les éléments 2 et 6 de la table initiale, on fera :

```
library(maptools)
```

```
## Checking rgeos availability: TRUE
## Please note that 'maptools' will be retired by the end of 2023,
## plan transition at your earliest convenience;
## some functionality will be moved to 'sp'.
```

```
maghreb_sp_new <- unionSpatialPolygons(maghreb_sp,
              IDs = c("Algeria", "Morocco", "Libya",
                      "Mauritania", "Tunisia", "Morocco"))
```

```
class(maghreb_sp_new)
```

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

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. On a pris ici les valeurs du PIB par habitant en dollars et la population en milliers d'habitants en 2012 issues du site de la Wold Bank. On a choisi pour nom de lignes les mêmes identifiants (mais pas forcément ordonnés de la même façon), que ceux utilisés dans l'objet de classe **SpatialPolygons** :

```
maghreb2.df <- data.frame(NOM = c("Algeria", "Morocco", "Libya",
                                  "Mauritania", "Tunisia"),
                          pib = c(4177, 2875, 4337, 1139, 3861),
                          pop = c(39728, 34663, 6418, 4046, 11179),
                          region = rep("N", 5))
row.names(maghreb2.df) <- maghreb2.df$NOM
```

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

```
maghreb_sp_new <- SpatialPolygonsDataFrame(maghreb_sp_new, maghreb2.df)
```

Remarque 1 : pour que la jointure entre les géométries et le jeu de données se passe correctement, les deux objets doivent avoir les mêmes identifiants. En effet, en tapant la ligne de code suivante, on peut voir que les géométries n'étaient pas ordonnées de la même façon que le tableau de données, mais malgré cela, la fusion s'est bien opérée. Pour savoir comment ont été ordonnées les unités spatiales dans l'objet **maghreb_sp_new**, on fait :

```
sapply(maghreb_sp_new@polygons, function(x) x@ID)
```

```
## [1] "Algeria"    "Libya"      "Mauritania" "Morocco"    "Tunisia"
```

Remarque 2 : il est également possible d'utiliser la fonction `aggregate()` pour réaliser les deux opérations (sur la géométrie et sur la table de données) en même temps. Par exemple, on souhaite créer une table **big_maghreb** qui fusionne les 5 pays. Par contre, pour obtenir le PIB par habitant sur les 5 pays fusionnés, on est obligé de faire quelques changements. Pour cela, on va pondérer la variable PIB par la taille de la population avant de faire l'aggrégation et diviser ensuite par la somme de la population totale. Dans la fonction `aggregate()`, l'argument **by** joue le même rôle que l'argument **IDs** vu précédemment :

```
maghreb_sp_new$pib_tot <- maghreb_sp_new$pib * maghreb_sp_new$pop
big_maghreb <- aggregate(maghreb_sp_new[, c("pib_tot", "pop")],
                         by = list(name = rep("maghreb", 5)),
          FUN = sum)
big_maghreb$pib <- round(big_maghreb$pib_tot / big_maghreb$pop)
maghreb_sp_new$pib_tot <- NULL
```

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

```
maghreb_sf_new <- maghreb_sf %>%
  group_by(NOM = c("Tunisia", "Algeria", "Morocco",
                   "Libya", "Morocco", "Mauritania")) %>%
  summarise() %>%
  merge(maghreb2.df, by = "NOM")
```

Exercice 9

Dans le jeu de données que vous avez choisi, faire l'aggrégation de toutes les unités spatiales pour n'en former qu'une. Choisir la transformation appropriée sur les variables observées. A défaut, on fera l'aggrégation sur les iris de Toulouse et on choisira la variable **p12_pop**. Représenter sur une carte l'observation aggrégée.

3.3.5 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 l'Egypte aux données précédentes, on créée d'abord un objet "Spatial" :

```
egypt_sp <- world_sp["Egypt", ]
egypt_df <- data.frame(NOM = "Egypt", pib = 3599, pop = 92442, region = "N")
egypt_sp <- merge(egypt_sp, egypt_df, by = "NOM")
row.names(egypt_sp) = "Egypt"
```

On utilise ensuite la fonction `spRbind()` (analogue de la fonction `rbind()`) :

```
northAf_sp <- spRbind(maghreb_sp_new, egypt_sp)
```

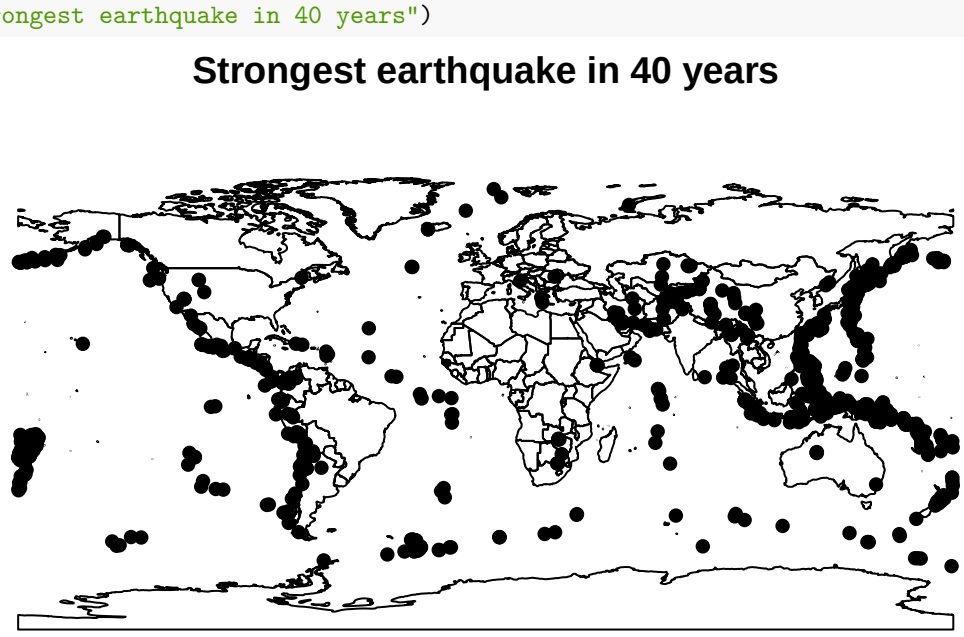
Le principe est le même avec la classe **sf** sauf que la fonction s'appelle `rbind()` et qu'on peut continuer à utiliser la syntaxe **dplyr** :

```
egypt_sf <- world_sf[world_sf$NOM == "Egypt", ]
egypt_sf <- merge(egypt_sf, egypt_df)
northAf_sf <- maghreb_sf_new %>%
  rbind(egypt_sf)
```

3.3.6 Représenter deux sources de données sur la même carte

3.3.6.1 Représenter les entités spatiales On souhaite représenter les tremblements de terre avec les contours géographiques des pays sur la même carte. Pour faire cela avec la classe "Spatial", il suffit d'utiliser la fonction `plot()` (pour objets spatiaux) autant de fois que nécessaire en ajoutant l'argument `add=T` lorsqu'on souhaite garder ce qui a été déjà représenté sur la carte.

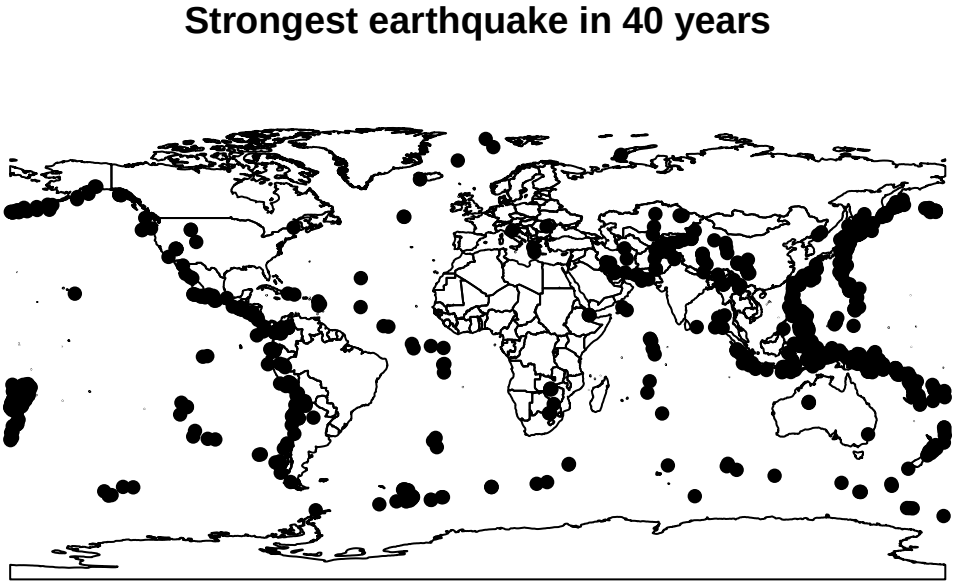
```
plot(world_sp)
plot(seisme_sp[1:1000, ], pch = 16, cex = 1, add = TRUE)
title("Strongest earthquake in 40 years")
```



Avec la classe **sf**, il y a deux possibilité :

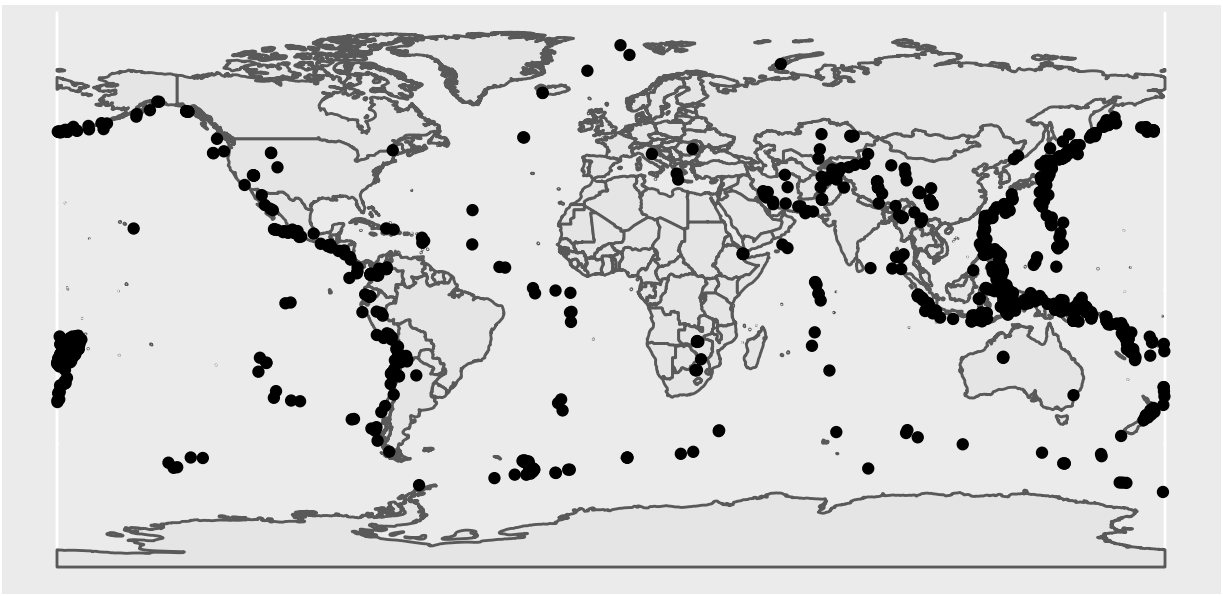
- utiliser la fonction `st_geometry()` qui permet de ne représenter que les contours géographiques et on utlise une syntaxe très similaire à celle de **sp**

```
plot(st_geometry(world_sf))
plot(st_geometry(seisme_sf[1:1000, ]), pch = 16, cex = 1, add = TRUE)
title("Strongest earthquake in 40 years")
```



- utiliser **ggplot** et avec un *geom* dédié aux données spatiales : *geom_sf()*. On utilise autant de fois cette fonction qu'il y a de sources de données à représenter. Par exemple :

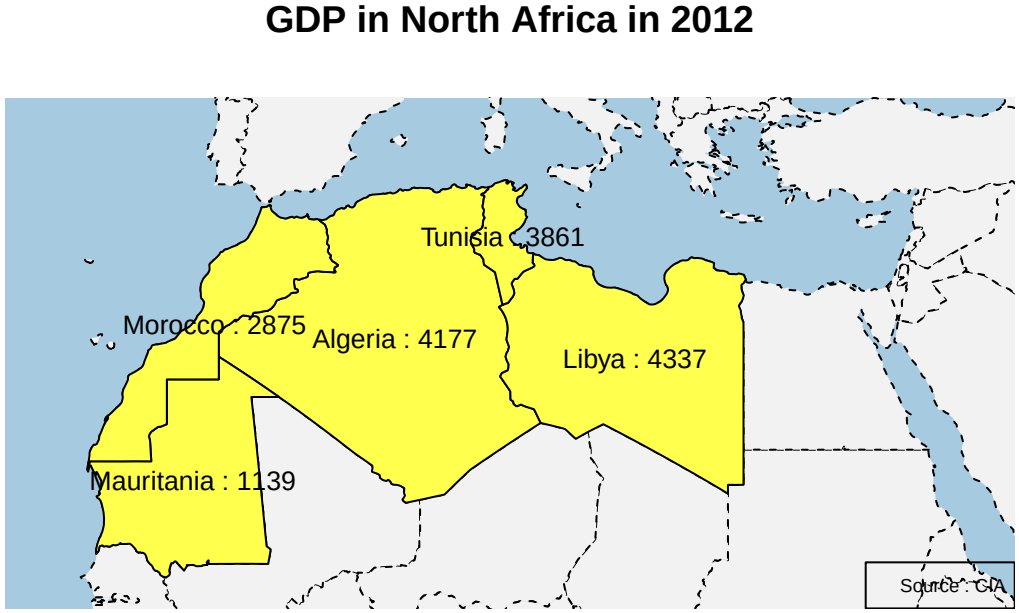
```
ggplot(data = world_sf, aes(geometry = geometry)) +
   geom_sf() +
   geom_sf(data = seisme_sf[1:1000,]) +
   ggtitle("Strongest earthquake in 40 years")
```



Remarque : il est possible d'afficher sur une même carte deux sources différentes si et seulement si les deux sources sont exprimées dans le même référentiel. On verra dans la section suivante comment il est possible de faire des transformations de CRS quand cela est nécessaire.

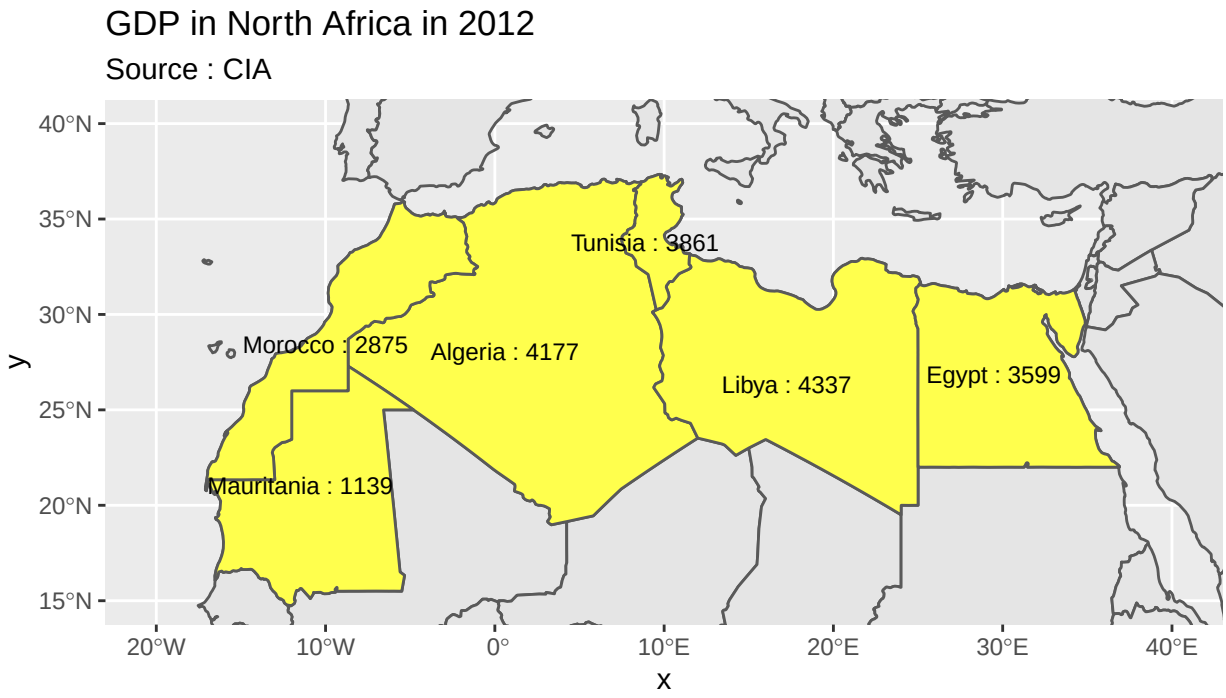
3.3.6.2 Représenter une étiquette A présent, en plus de l'information géographique, on va aussi représenter du texte sur la carte. Pour faire cela avec la classe "Spatial", on peut utiliser les fonctions graphiques de base `text()`, `title()` et `legend()`.

```
plot(world_sp, col = rgb(0.95, 0.95, 0.95), bg="#A6CAE0",
     xlim = c(-20, 40), ylim = c(15, 40), lty = 2)
plot(maghreb_sp_new, col = rgb(1, 1, 0.3), add = T)
title("GDP in North Africa in 2012")
text(coordinates(maghreb_sp_new),
 paste(row.names(maghreb_sp_new), maghreb_sp_new$pib, sep = " : "), cex = 0.8)
legend("bottomright","Source : CIA", cex = 0.6)
```



En utilisant la classe **sf** et la syntaxe **ggplot**, on peut utiliser la fonction `geom_sf_text()` qui permettra de représenter les étiquettes sur les coordonnées des centroïdes et `coord_sf()` qui permet de déterminer la fenêtre d'étude :

```
ggplot(data = world_sf, aes(geometry = geometry)) +
  geom_sf() +
  geom_sf(data = northAf_sf, fill = rgb(1, 1, 0.3)) +
  geom_sf_text(data = northAf_sf,
     aes(label = paste(NOM, pib, sep = " : ")),
      size = 3) +
  coord_sf(xlim = c(-20, 40), ylim = c(15, 40)) +
  ggtitle("GDP in North Africa in 2012",
    subtitle = "Source : CIA")
```



Exercice 10

Représenter sur une même carte deux sources de données différentes. A défaut, on représentera les iris de la ville de Toulouse et les stations de vélo.

3.3.7 Changement de CRS

A présent, on souhaiterait représenter le contour administratif de la France dans le référentiel officiel, à savoir le RGF 93. Pour faire cela avec la classe "Spatial", on va utiliser la fonction `spTransform()` qui permet de faire la conversion d'un CRS vers un autre. Pour que cela marche, il faut que le CRS soit correctement donné. Une façon de ne pas se tromper est d'utiliser la nomenclature EPSG. Par exemple, pour la France, il s'agit de la nomenclature 2154 qu'on va renseigner ainsi `"EPSG:2154"` :

```
france_sp <- world_sp["France", ]
my_crs_fr <- CRS(SRS_string = "EPSG:2154")
```

```
## Warning in showSRID(SRS_string, format = "PROJ", multiline = "NO", prefer_proj
## = prefer_proj): Discarded datum Reseau Geodesique Francais 1993 in Proj4
## definition
```

```
france_sp2 <- spTransform(france_sp, my_crs_fr)
```

La signification du message d'avertissement est expliquée dans ce post. En gros, lorsqu'on execute la fonction `CRS()`, celle-ci utilise les deux nomenclatures **Proj4** et **WKT**. La nomenclature **Proj4** s'affiche par défaut, alors que pour afficher la nomenclature **WKT**, il faut appliquer la fonction `wkt()` :

```
my_crs_fr
```

```
## Coordinate Reference System:
## Deprecated Proj.4 representation:
##  +proj=lcc +lat_0=46.5 +lon_0=3 +lat_1=49 +lat_2=44 +x_0=700000
## +y_0=6600000 +ellps=GRS80 +units=m +no_defs
## WKT2 2019 representation:
## PROJCRS["RGF93 / Lambert-93",
##     BASEGEOGCRS["RGF93",
```

```
##         DATUM["Reseau Geodesique Francais 1993",
##             ELLIPSOID["GRS 1980",6378137,298.257222101,
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4171]],
##     CONVERSION["Lambert-93",
##         METHOD["Lambert Conic Conformal (2SP)",
##             ID["EPSG",9802]],
##         PARAMETER["Latitude of false origin",46.5,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8821]],
##         PARAMETER["Longitude of false origin",3,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8822]],
##         PARAMETER["Latitude of 1st standard parallel",49,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8823]],
##         PARAMETER["Latitude of 2nd standard parallel",44,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8824]],
##         PARAMETER["Easting at false origin",700000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8826]],
##         PARAMETER["Northing at false origin",6600000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8827]]],
##     CS[Cartesian,2],
##         AXIS["easting (X)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["northing (Y)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["Engineering survey, topographic mapping."],
##         AREA["France - onshore and offshore, mainland and Corsica."],
##         BBOX[41.15,-9.86,51.56,10.38]],
##     ID["EPSG",2154]]
```

```
cat(wkt(my_crs_fr))
```

```
## PROJCRS["RGF93 / Lambert-93",
##     BASEGEOGCRS["RGF93",
##         DATUM["Reseau Geodesique Francais 1993",
##             ELLIPSOID["GRS 1980",6378137,298.257222101,
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4171]],
##     CONVERSION["Lambert-93",
##         METHOD["Lambert Conic Conformal (2SP)",
##             ID["EPSG",9802]],
##         PARAMETER["Latitude of false origin",46.5,
##             ANGLEUNIT["degree",0.0174532925199433],
```

```
##             ID["EPSG",8821]],
##         PARAMETER["Longitude of false origin",3,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8822]],
##         PARAMETER["Latitude of 1st standard parallel",49,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8823]],
##         PARAMETER["Latitude of 2nd standard parallel",44,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8824]],
##         PARAMETER["Easting at false origin",700000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8826]],
##         PARAMETER["Northing at false origin",6600000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8827]]],
##     CS[Cartesian,2],
##         AXIS["easting (X)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["northing (Y)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["Engineering survey, topographic mapping."],
##         AREA["France - onshore and offshore, mainland and Corsica."],
##         BBOX[41.15,-9.86,51.56,10.38]],
##     ID["EPSG",2154]]
```

Le message d'avertissement concerne uniquement la nomenclature **Proj4**; en effet, depuis un changement de version de la librairie **PROJ**, certains arguments de la nomenclature Proj4 (comme `+datum`) ne sont plus nécessairement pris en compte. En fait, depuis 2020, les packages **sp** et **sf** utilisent la nomenclature WKT pour effectuer les conversions. Du coup, le message d'avertissement sert de rappel pour celles/ceux qui utiliseraient encore la nomenclature **Proj4** sur un CRS ayant un **datum** non reconnu.

Pour faire cela avec la classe **sf**, on utilisera la fonction `st_transform()` qui marche selon le même principe :

```
france_sf <- world_sf %>%
  filter(NOM == "France")
france_sf2 <- st_transform(france_sf, my_crs_fr)
france_sf2 <- st_transform(france_sf, 2154)
```

Pour vérifier que la transformation a bien été prise en compte :

```
st_crs(france_sf2)
```

```
## Coordinate Reference System:
##   User input: EPSG:2154
##   wkt:
## PROJCRS["RGF93 / Lambert-93",
##     BASEGEOGCRS["RGF93",
##         DATUM["Reseau Geodesique Francais 1993",
##             ELLIPSOID["GRS 1980",6378137,298.257222101,
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
```

```
##         ID["EPSG",4171]],
##     CONVERSION["Lambert-93",
##         METHOD["Lambert Conic Conformal (2SP)",
##             ID["EPSG",9802]],
##         PARAMETER["Latitude of false origin",46.5,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8821]],
##         PARAMETER["Longitude of false origin",3,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8822]],
##         PARAMETER["Latitude of 1st standard parallel",49,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8823]],
##         PARAMETER["Latitude of 2nd standard parallel",44,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8824]],
##         PARAMETER["Easting at false origin",700000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8826]],
##         PARAMETER["Northing at false origin",6600000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8827]]],
##     CS[Cartesian,2],
##         AXIS["easting (X)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["northing (Y)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["Engineering survey, topographic mapping."],
##         AREA["France - onshore and offshore, mainland and Corsica."],
##         BBOX[41.15,-9.86,51.56,10.38]],
##     ID["EPSG",2154]]
```

Application : on va représenter les tremblements de terre et la carte des pays du monde dans le CRS "World Robinson" dont l'EPSG vaut `"ESRI:54030"` et représenter les deux sources de données dans ce nouveau CRS.

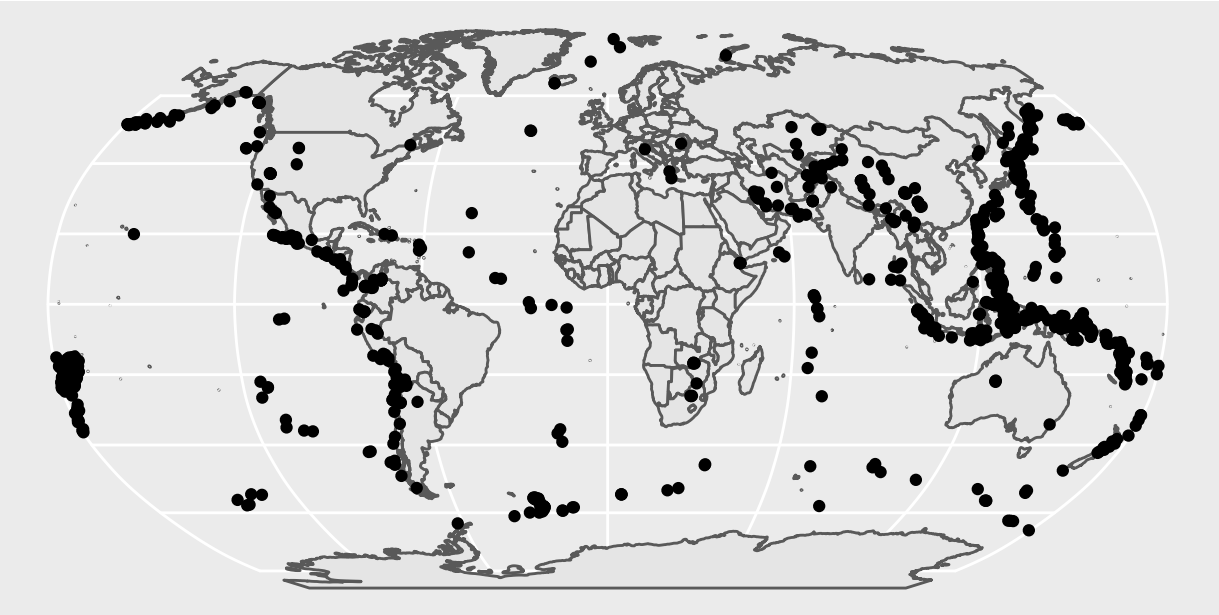
```
crs_robi <- CRS("ESRI:54030")
```

- Dans la norme "Spatial" :

```
world_sp_robi <- spTransform(x = world_sp, CRSobj = crs_robi)
seisme_sp_robi <- spTransform(seisme_sp, CRSobj = crs_robi)
```

- Dans la norme **sf** :

```
world_sf_robi <- st_transform(x = world_sf, crs = crs_robi)
seisme_sf_robi <- st_transform(seisme_sf, crs = crs_robi)
ggplot(data = world_sf_robi, aes(geometry = geometry)) +
   geom_sf() +
   geom_sf(data = seisme_sf_robi[1:1000,]) +
   ggtitle("Strongest earthquake in 40 years")
```



Exercice 11

Afficher le CRS de votre jeu de données. Transformer votre jeu de données dans le référentiel en vigueur.

3.3.8 Transformer les géométries

Il est possible de faire des modifications de type $f(x) = xA + b$ où x est une géométrie de type point/polygone/ligne (il faut penser x comme étant une matrice à deux colonnes de type longitude/latitude), b est un vecteur permettant de translater et A une matrice permettant de réduire-grandir et/ou faire pivoter la géométrie. Nous avons utilisé la référence suivante ici qui marche pour la norme **sf**.

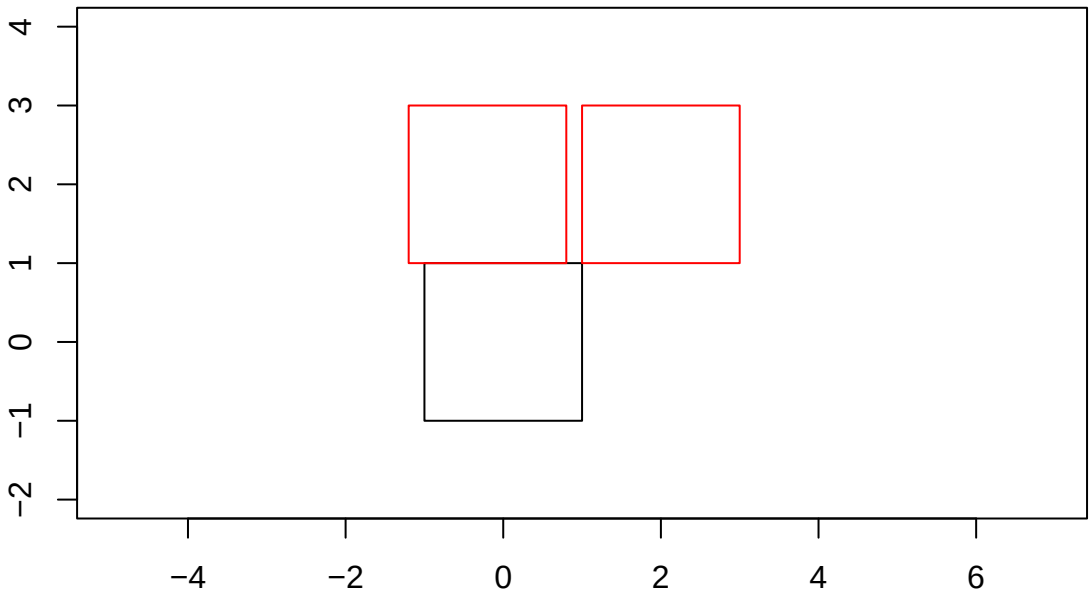
En partant du polygone suivant :

```
b0 <- st_polygon(list(rbind(c(-1,-1), c(1,-1), c(1,1), c(-1,1), c(-1,-1))))
```

On applique les transformations suivantes :

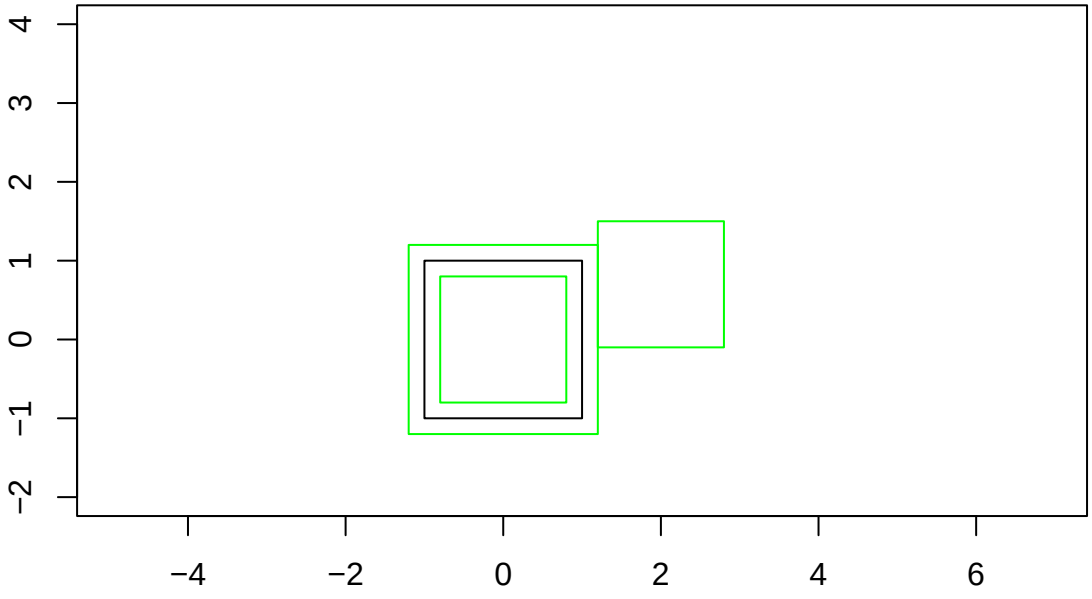
- Translation

```
b1 <- b0 + 2
b2 <- b0 + c(-0.2, 2)
x <- st_sfc(b1, b2)
plot(b0, axes = T, xlim = c(-1, 3), ylim = c(-2, 4))
plot(x, border = 'red', axes = T, add = T)
```



- Agrandir/Réduire

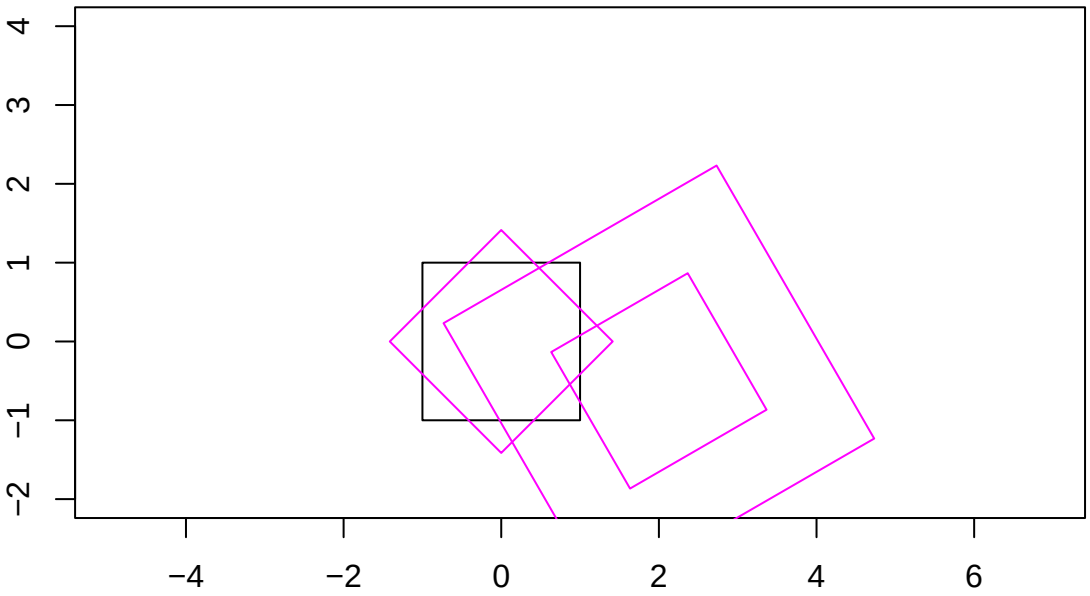
```
a0 <- b0 * 0.8
a1 <- b0 * 1.2
a2 <- b0 * 0.8 + c(2, 0.7)
y <- st_sfc(a0,a1,a2)
plot(b0, axes = T, xlim = c(-1, 3), ylim = c(-2, 4))
plot(y, border = 'green', add = T)
```



- Faire une rotation :

```
rot <- function(a) matrix(c(cos(a), sin(a), -sin(a), cos(a)), 2, 2)
c1 <- b0 * rot(pi/4)
c2 <- b0 * rot(pi/3) + c(2, -0.5)
c3 <- (b0 * 2) * rot(pi/3) + c(2, -0.5)
z <- st_sfc(c1, c2, c3)
plot(b0, axes = T, xlim = c(-1, 3), ylim = c(-2, 4))
```

```
plot(z, border = 'magenta', add = T)
```



3.3.9 Union, intersection, différence, etc. sur les géométries

Il existe un grand nombre de fonctions qui permettent de réaliser des opérations ensemblistes sur les géométries. Les principales fonctions qui proviennent du package **rgeos** (Bivand et Rundell, 2019) sont les suivantes :

- `gIntersection(A, B)` : l'intersection des géométries A et B
- `gUnion(A, B)` : l'union des géométries A et B
- `gDifference(A, B)` : la géométrie de A auquel on enlève la géométrie de B

Il existe beaucoup d'autres fonctions (voir `help(package = "rgeos")`) dont on présente ici les principales :

- `gArea(x)` : calcul l'aire de x. En utilisant l'argument `byid = TRUE*`, cela permet de calculer l'aire de toutes les unités spatiales comprises dans x
- `gConvexHull(x)` : détermine une enveloppe convexe de x
- `gBuffer(x)` : détermine un tampon de x.

Pour faire cela 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
- `st_simplify(x)` : simplifie la géométrie lorsque les contours sont très précis et par conséquent volumineux en mémoire

- `st_bbox()` : retourne les coordonnées du rectangle qui englobe la géométrie

Exemple : on va créer une variable **area** qui calcule l'aire de chaque pays dans la base **world_sf_robi**.

```
library(rgeos)
world_sp_robi$area <- gArea(world_sp_robi, byid = T)
world_sf_robi$area <- st_area(world_sf_robi)
```

Exercice 12

Dans votre jeu de données, créer une variable **area** qui calcule l'aire de chaque unité spatiale.

3.3.9.1 Application 1 : extraction d'une sous-zone géographique

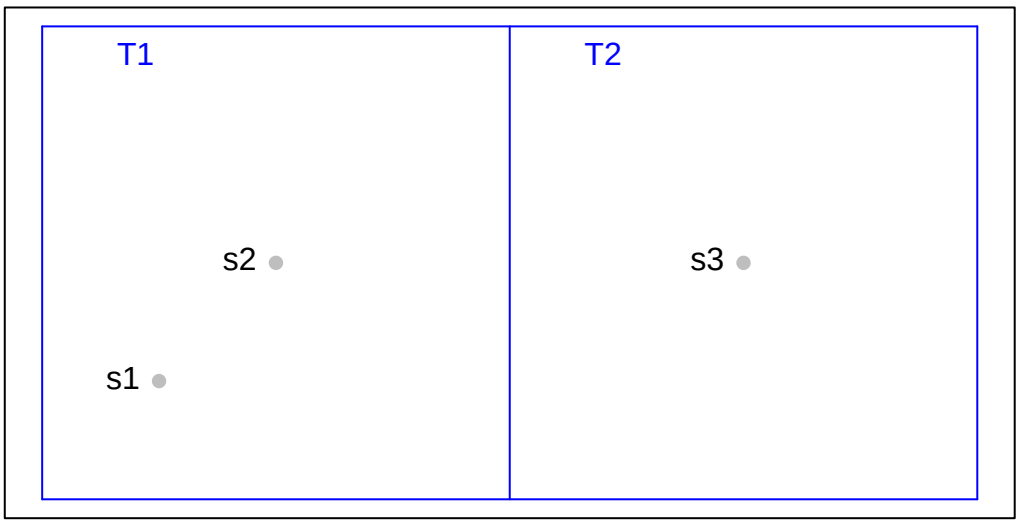
L'idée est la suivante : on souhaite représenter les tremblements de terre qui sont localisés autour du Japon.

On pourrait continuer de travailler sur toute la base, mais cela coûte en temps calcul de conserver toutes les observations alors qu'on s'intéresse uniquement à un sous-échantillon. L'idée est de faire une intersection de deux types d'objets : un polygone correspondant à la zone d'étude et les points correspondants aux tremblements de terre.

Il est bien évidemment possible de faire des intersections entre tous les types d'objets (points, lignes, polygones). L'ordre dans lequel les objets vont apparaître est alors primordial. Par exemple, si on intersecte A et B, où A est de type Points et B de type polygone, alors le résultat sera un objet de type Points, sur lequel on aura fusioné les attributs des Points initiaux avec les attributs du polygone qui intersecte avec ces points.

Exemple d'application d'une intersection spatiale

On considère la figure suivante où il y a deux fichiers spatiales, les polygones d'une part et les points d'autre part. Il y ainsi deux façons possible de faire des intersections :



- si on intersecte les 3 points avec les 2 polygones, le résultat sera 3 points qui auront les attributs des 3 points ainsi que les attributs du polygone T1 (pour les points s1 et s2) et les attributs du polygone T2 (pour le point s3).
- si on intersecte les 2 polygones avec les 3 points, le resultat sera 3 polygones T1a et T1b auront les mêmes géométries et les mêmes attributs que T1 mais T1a héritera des attributs du point s1 alors que T1b héritera des attributs du point s2. T2 aura les attributs de T2 et les attributs du point s3.

Sous-sélection d'une zone d'étude

On va d'abord définir la zone d'étude, à savoir le "bounding box" du Japon. Cette zone peut être sélectionnée avec la fonction `bbox()` sous forme de matrice. Pour transformer cette matrice en objet de type "Spatial", on va passer par la fonction intermédiare `extent()` du package **raster** dont l'objet créé peut ensuite être converti en objet "Spatial".

Il est intéressant de voir que les développeurs du package **raster** n'ont pas uniquement travaillé sur des images, mais aussi des vecteurs. En réalité, les packages sont souvent complémentaires, même s'il traite d'objets ou de spécialités différentes (géostatistique, économétrie spatiale, processus ponctuels spatiaux) et il n'est pas surprenant de les utiliser conjointement.

L'intersection de deux objets "Spatial" peut se faire avec la fonction `gIntersection()` du package **rgeos**. Une autre possibilité est d'utiliser la fonction `over()`.

- Sélection du contour du Japon :

```
japan_sp <- world_sp["Japan",]
```

- Transformation du "bounding box" en objet Spatial de type polygone :

```
bb.jp_sp <- as(extent(bbox(japan_sp)), "SpatialPolygons")
```

L'opération précédente n'a pas conservé le CRS initial. C'est pourquoi il est indispensable de le re-préciser sans quoi il ne sera pas possible d'utiliser la fonction `over()` par la suite.

- Définition du CRS :

```
proj4string(bb.jp_sp) <- CRS(SRS_string = "EPSG:4326")
```

Intersection

Pour faire l'intersection des tremblements de terre avec le "bounding box" du Japon :

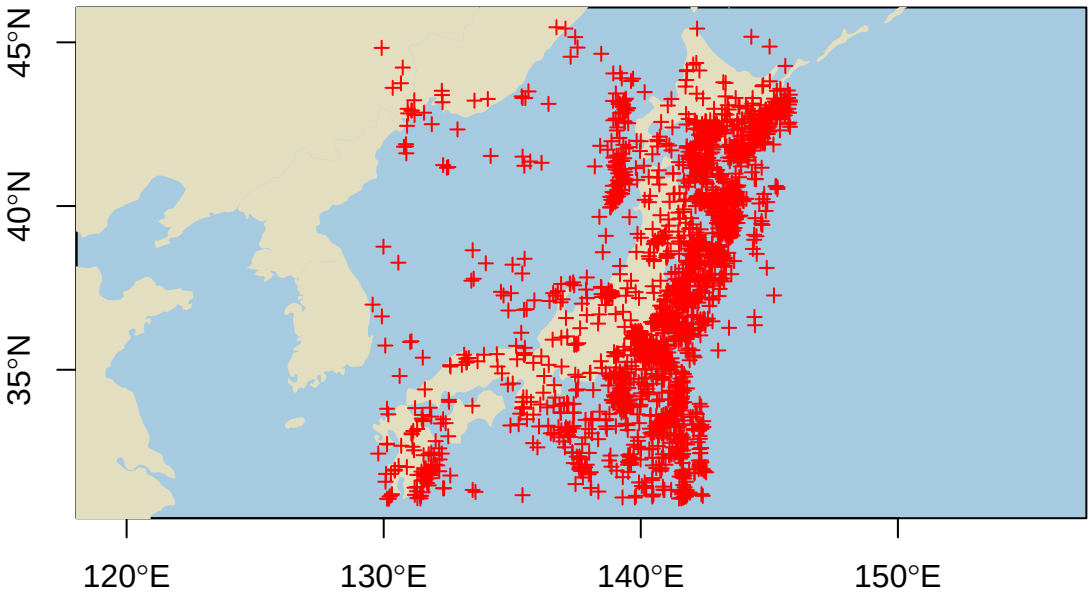
```
ind <- over(seisme_sp, bb.jp_sp)
```

La fonction `over()` retourne un vecteur de la même taille que le 1er argument avec la valeur **1** si le tremblement de terre intersecte le deuxième objet "Spatial" et **NA** sinon. Pour obtenir la sélection désirée, il suffit donc de garder les valeurs différentes de **NA** :

```
seisme_sp.jp <- seisme_sp[which(!is.na(ind)), ]
```

On représente les données :

```
plot(japan_sp, border = NA, col = "NA", bg="#A6CAE0", axes = T)
plot(world_sp, col = "#E3DEBF", border = NA, add = T)
plot(seisme_sp.jp, add = TRUE, col = "red", cex = 0.7)
```



Pour reprendre l'exemple sur le Japon avec la classe **sf**, le principe est exactement le même.

- création du polygone contenant le Japon :

```
japan_sf <- world_sf %>% filter(NOM == "Japan")
```

- conversion du "bounding box" (version **sf**, qui conserve l'information sur le CRS contrairement à la version "Spatial") directement en "simple feature geometry" avec la fonction `st_as_sfc()` :

```
bb.jp_sf <- st_as_sfc(st_bbox(japan_sf))
```

- Intersection des deux objets qui sont des "simple feature geometry" :

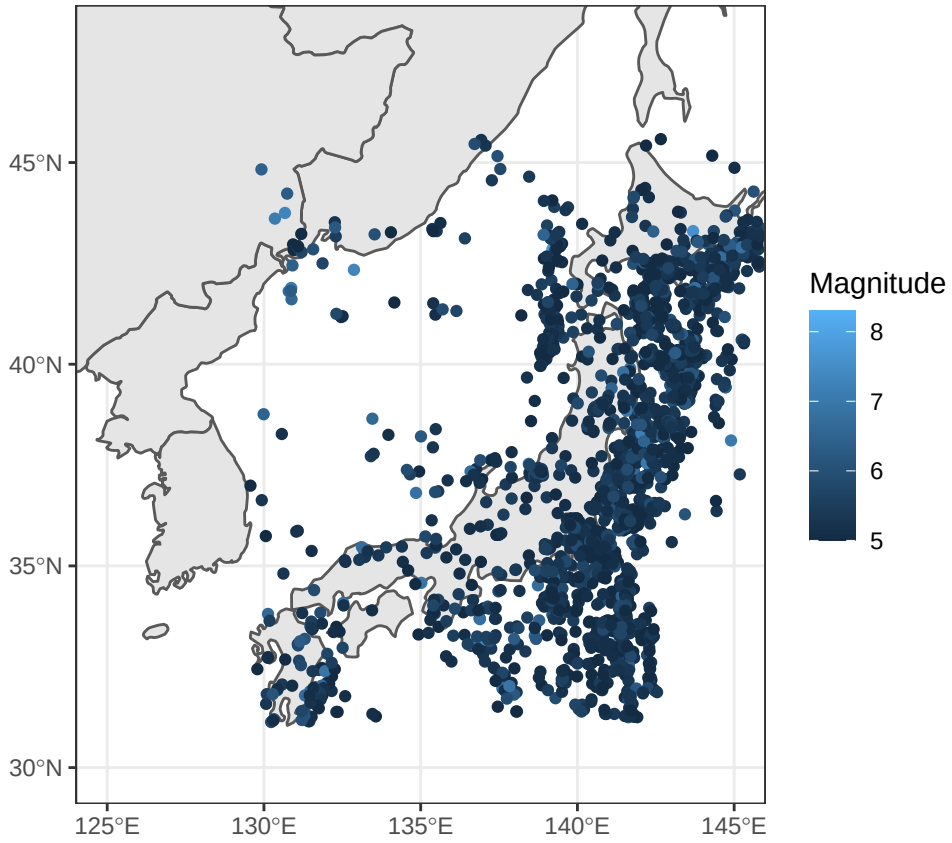
```
seisme.jp_sf <- st_intersection(seisme_sf, bb.jp_sf)
```

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

Remarque : le message retourné indique qu'il est préférable de faire des intersections sur des coordonnées planes plutôt que sur des coordonnées longitude/latitude.

On représente les données sélectionnées :

```
seisme.jp_sf %>% ggplot(aes(geometry = geometry)) +
  geom_sf(data = world_sf) +
  geom_sf(aes(colour = Magnitude)) +
  xlim(125, 145) +
  ylim(30, 48) +
  theme_bw()
```



Exercice 13

Faire l'intersection du jeu de données **bike** avec le jeu de données **iris** dans le but de savoir dans quel iris une station de vélo est contenue.

3.3.9.2 Application 2 : faire des tampons ou "buffer" En reprenant l'exemple précédent, on souhaite sélectionner uniquement les tremblements de terre qui sont localisés à 50 km des côtes du Japon. L'idée est donc de constuire une zone tampon ("buffer") qui consiste à faire un décalage parallèle des géométries d'un objet spatial.

Dans le cas d'un point, une zone tampon de 10 km consiste à créer un cercle de 10km autour de ce point.

Dans le cas d'un polygone, un buffer de +10 km consiste à agrandir le polygone de telle sorte que les nouvelles frontières soient à une distance de 10km des anciennes. Inversement, un buffer de -10km consiste à retrécir le polygone de telle sorte que les nouvelles fontières soient à une distance de 10 km des anciennes.

Dès lors qu'on souhaite calculer des distances sur la surface terrestre, il est nécessaire d'utiliser des coordonnées planes. Dans le cas du Japon, on doit donc trouver le CRS approprié afin de convertir les frontières ainsi que les tremblements de terre exprimés en longitudes/latitudes. On a trouvé sur le site https://epsg.io/2443 un candidat possible.

```
crs_jp <- CRS("EPSG:2443")
```

```
## Warning in showSRID(SRS_string, format = "PROJ", multiline = "NO", prefer_proj =
## prefer_proj): Discarded datum Japanese Geodetic Datum 2000 in Proj4 definition
```

```
seisme_sp.jp2 <- spTransform(seisme_sp.jp, crs_jp)
japan_sp2 <- spTransform(japan_sp, crs_jp)
```

Pour créer la zone tampon, on utilise la fonction `gBuffer()` du package **rgeos** où le second argument représente la distance entre l'ancienne et la nouvelle frontière :

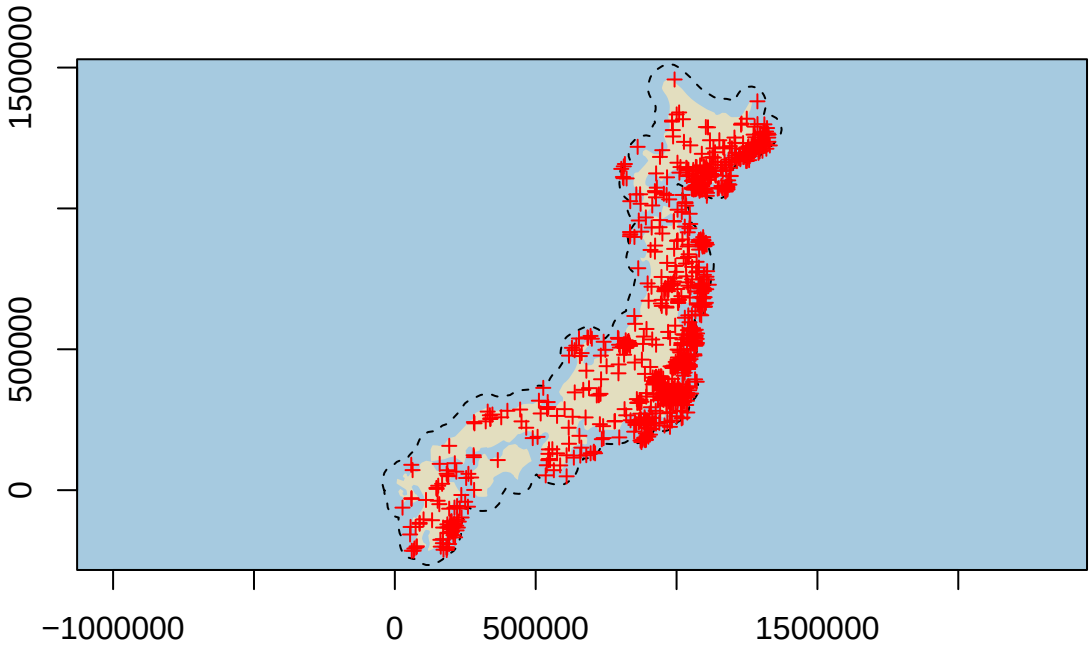
```
japan_sp_buff <- gBuffer(japan_sp2, width = 50000)
```

On utilise ensuite la fonction `gIntersection()` pour ne sélectionner que les tremblements de terre qui tombent dans la zone tampon :

```
seisme_sp.jp3 <- gIntersection(seisme_sp.jp2, japan_sp_buff)
```

On représente finalement les différents objets créés :

```
plot(japan_sp2, border = NA, col = "NA", bg="#A6CAE0", axes = T)
plot(japan_sp2, col = "#E3DEBF", border = NA, add = T)
plot(japan_sp_buff, add = TRUE, lty = 2)
plot(seisme_sp.jp3, add=T, col = "red", cex = 0.7)
```



Remarque : on constate qu'une représentation en coordonnées planes permet de calculer des distances entre deux points directement sans avoir à utiliser la formule qui fait appel au rayon de la Terre.

Pour faire cela avec la classe **sf**, le principe est “exactement” le même :

- conversion des objets dans un système de coordonnées planes :

```
seisme.jp_sf2 <- st_transform(seisme.jp_sf, 2443)
japan_sf2 <- st_transform(japan_sf, 2443)
```

- création de la zone tampon avec la fonction `st_buffer()` :

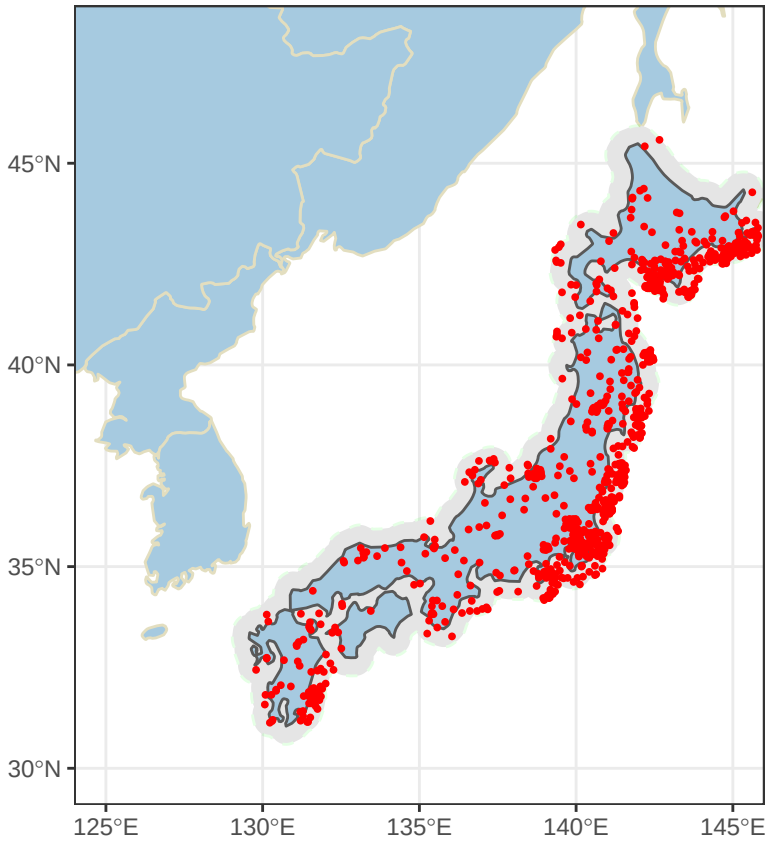
```
japan_sf_buff <- st_buffer(japan_sf2, dist = 50000)
```

- intersection des points et du polygone contenant la zone tampon :

```
seisme.jp_sf3 <- st_intersection(seisme.jp_sf2, japan_sf_buff)
```

- représentation des données :

```
ggplot(data = world_sf, aes(geometry = geometry)) +
  geom_sf(bg = "#A6CAE0", col = "#E3DEBF") +
  geom_sf(data = japan_sf_buff, lty=2, col = scales::alpha("green", 0.1)) +
  geom_sf(data = japan_sf2, fill = "#A6CAE0") +
  geom_sf(data = seisme.jp_sf3, col = "red", cex = 0.7) +
  xlim(125, 145) + ylim(30, 48) + theme_bw()
```



Remarque : on a représenté ici l’objet **world_sf** qui est exprimé en longitude/latitude et les objets sur le Japon qui sont exprimés en coordonnées planes. Lorsqu’on a utilisé la fonction `geom_sf()`, la transformation des coordonnées planes en coordonnées longitude/latitude s’est faite automatiquement de telle sorte que les deux sources de données puissent être représentées dans le même CRS.

3.4 Manipulation de données de type raster

On a vu que la fonction `raster()` permettait d'importer une image spatiale de type raster. On importe ici un jeu de données qui contient le rayonnement annuel reçu par pixel sur une période de 10 ans (pour plus d'informations, voir http://re.jrc.ec.europa.eu/pvgis/) sur l'Europe et dont on sait qu'elles sont exprimées en longitude/latitude.

```
suneu <- raster("Donnees/soleil/pvgis_g13year00.asc",
 crs = CRS(SRS_string = "EPSG:4326"), native = TRUE)
```

```
## Warning in is.na(x): is.na() appliqué à un objet de type 'closure' qui n'est ni
## une liste, ni un vecteur
```

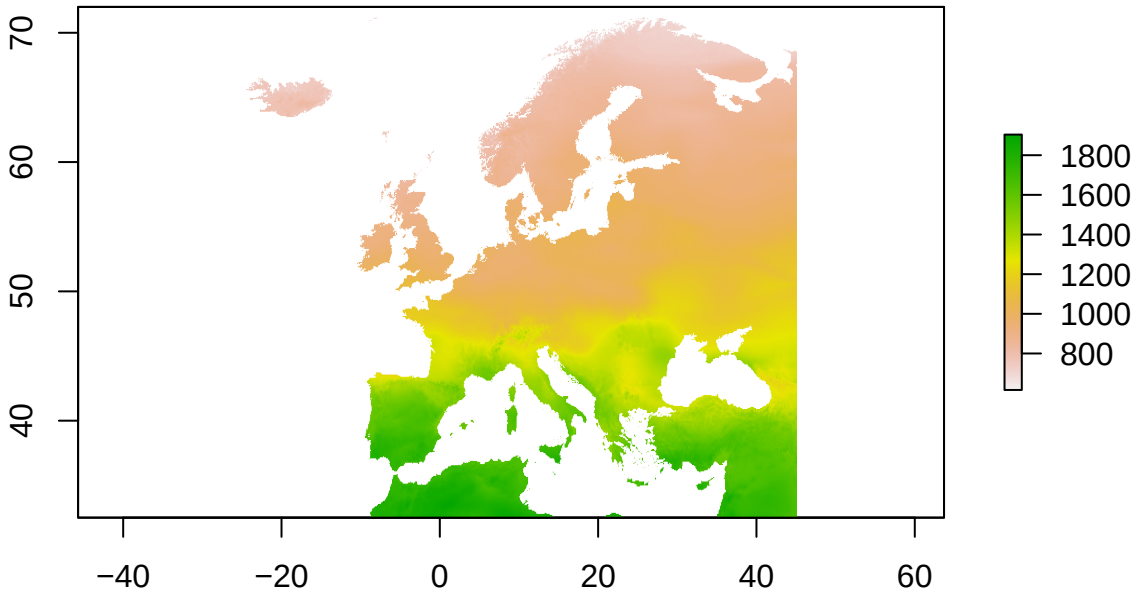
Pour accéder aux données observées sur un **raster**, on a vu qu'il suffisait de procéder comme si on manipulait un **array** ou une matrice.

```
suneu[150, 685, 1]
```

```
##
## 945
```

La fonction `plot()` (ou `image()`) permet de représenter l'image :

```
plot(suneu)
```



3.4.1 Extraction de pixels

On souhaite extraire les pixels qui sont localisés en France. Pour faire cela, on va procéder en deux étapes :

1. Extraire les pixels qui sont dans le "bounding box" de la France. Pour cela, on applique la fonction `bbox()` sur les contours de la France, on lui applique la fonction `extent()` qui crée un objet pouvant être lu par la fonction `crop()`.

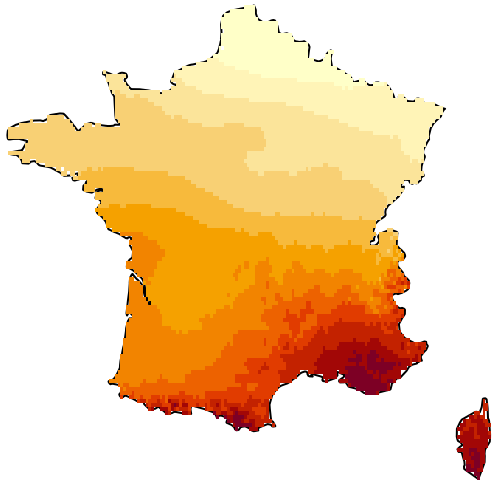
```
suneu.bb <- crop(suneu, extent(bbox(france_sp)))
```

2. Ne conserver que les pixels qui sont inclus à l'intérieur de la France.

```
suneu.fr <- rasterize(france_sp, suneu.bb, mask = TRUE)
```

On obtient ainsi la représentation suivante :

```
plot(france_sp)
image(suneu.fr, add = TRUE)
```



3.4.2 Changer la résolution d'une image

On souhaiterait diminuer la résolution de l'image dont la dimension est :

```
dim(suneu)
```

```
## [1] 474 864   1
```

Pour diminuer la résolution, on va utiliser la fonction `aggregate()` dont le deuxième argument donne l'échelle de réduction. Le 1er élément consiste à dire ici que sur une ligne, on va aggréger par groupe de 15 et le deuxième élément indique qu'on va aggréger par groupe de 10 cellules par colonne.

```
suneu.agg <- aggregate(suneu, fact = c(15, 10), fun = mean)
```

On obtient la nouvelle résolution suivante :

```
dim(suneu.agg)
```

```
## [1] 48 58  1
```

Enfin, on représente les nouvelles données. La fonction `add.masking()` permet de masquer les bouts de cellules qui sont localisées en-dehors du contour géographique de la France :

```
library(GISTools)
plot(france_sp)
image(suneu.agg, add = TRUE)
masker <- poly.outer(suneu.agg, france_sp)
add.masking(masker)
```

On vient de voir qu'il était possible de représenter sur un même graphique des objets de type raster et des vecteurs (comme le contour de la France). Le package **raster** permet d'importer des fonds de carte (raster ou vecteur) provenant de sites web comme GADM ("Global Administrative Areas").

3.4.3 Importation de données spatiales avec le package raster

Pour récupérer des contours administratifs de pays ou région, on peut utiliser la fonction `getData()`. L'argument **level** précise quels découpages administratifs (pays, régions, départements ou communes selon les pays). Par exemple, pour récupérer le contour des départements en France :

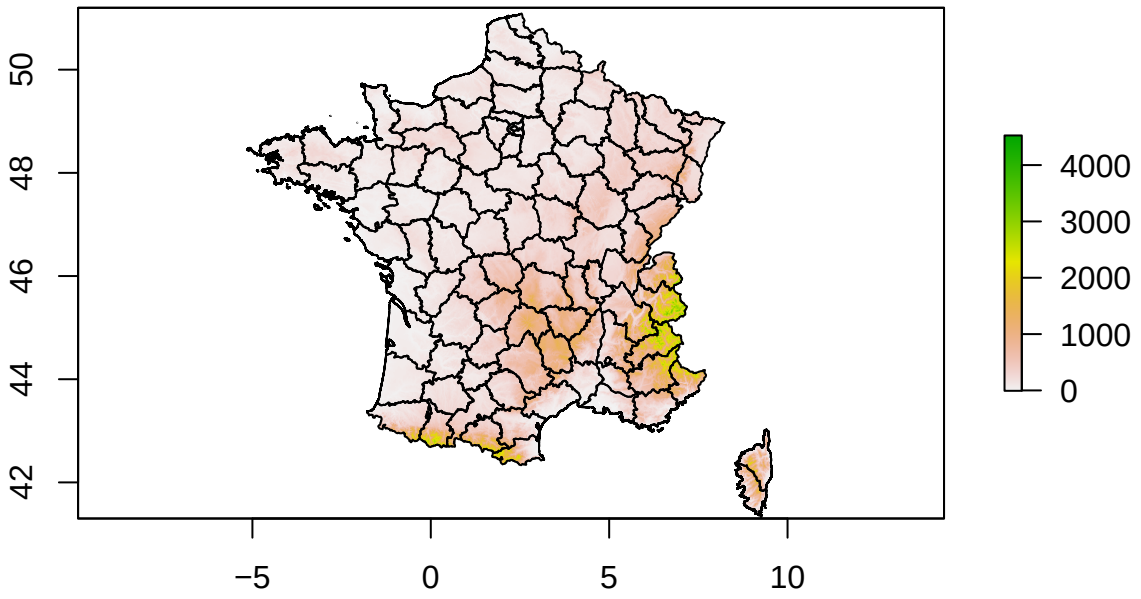
```
france_dep <- getData("GADM", country = "france", level = 2)
```

On peut également récupérer des images, comme par exemple les données d'altitude qui sont diffusées par le site de l'IGN :

```
france_alt <- getData("alt", country = "FRA", mask = TRUE)
```

On représente ici les deux types de données (vectoriel et raster) :

```
plot(france_alt)
plot(france_dep, add = T)
```



Les autres types de données qui peuvent être récupérées avec cette fonction sont disponibles sur le site DIVA-GIS (http://diva-gis.org/gdata).

3.4.4 Importation de fonds de carte OpenStreetMap

Il est possible de récupérer des fonds de carte OpenStreetMap sous forme d'images. Supposons qu'on souhaite représenter le fond de carte correspondant à la zone qui englobe les trois sites que nous avons trouvés dans la section précédente à l'aide la fonction `geocode()`. D'abord, on va créer un objet "Spatial" qui contienne ces trois points :

```
long <- c(1.432022, 1.436879, 1.435409)
lat <- c(43.605417, 43.606688, 43.610372)
xy <- SpatialPoints(cbind(long, lat), CRS("EPSG:4326"))
xy_sf <- as(xy, "sf")
```

Ensuite, on va utiliser la fonction `getTiles()` du package **maptiles** (Giraud, 2021) qui va directement créer un "bounding box" de la zone qui englobe ces trois points. Il est possible d'importer une zone plus ou moins grande avec l'option **zoom** (voir le lien suivant pour définir le paramétrage à utiliser : https://wiki.openstreetmap.org/wiki/Zoom_levels) et d'utiliser d'autres sources de fonds de cartes que celles fournies par OpenStreetMap (voir l'aide de la fonction pour voir toutes les possibilités)

```
library(maptiles)
mtqOSM <- maptiles::get_tiles(x = xy_sf, provider = "Stamen.Terrain",
                              zoom = 13)
```

L'objet importé est de type **SpatRaster** qui devrait devenir l'héritier du package **raster** (voir https://cran.r-project.org/web/packages/terra/index.html). Pour savoir dans quel CRS il a été importé, on fait :

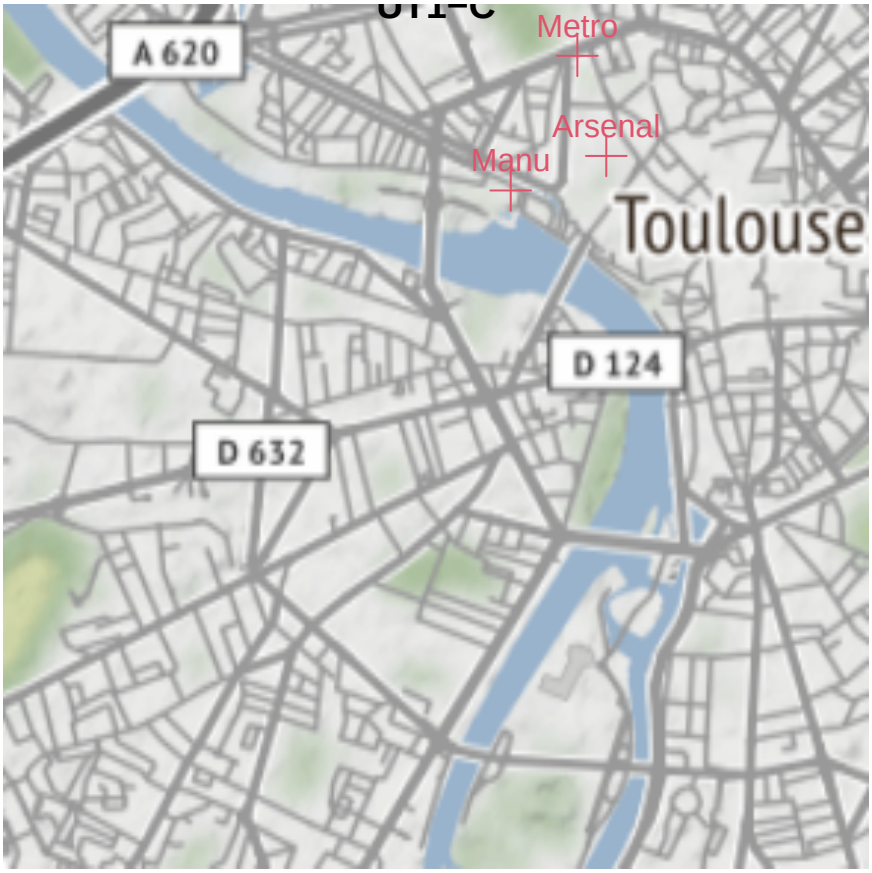
```
st_crs(mtqOSM)
```

```
## Coordinate Reference System:
##   User input: WGS 84 (with axis order normalized for visualization)
##   wkt:
## GEOGCRS["WGS 84 (with axis order normalized for visualization)",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]],
##         ID["EPSG",6326]],
```

```
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433],
##         ID["EPSG",8901]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433,
##                 ID["EPSG",9122]]],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433,
##                 ID["EPSG",9122]]]]
```

On constate que les CRS sont les mêmes. On peut finalement représenter toutes les informations en même temps. Pour représenter le fond de carte issu d'OSM, on utilisera la fonction `plot_tiles()` du package **maptiles** :

```
plot_tiles(mtqOSM)
title("UT1-C")
plot(xy, col = 2, cex = 2, add = TRUE)
text(coordinates(xy), col = 2, pos = 3,
  labels = c("Manu", "Arsenal","Metro"))
```



3.4.5 Calcul de distance routière

La fonction `osrmRoute()` du package **osrm** permet de calculer le plus court chemin entre des points en utilisant le serveur OpenStreetMap (idéalement, pour un utilisateur qui souhaiterait faire un nombre important

de requêtes, il est conseillé de télécharger les données d'OSM et de faire ensuite les requêtes sur une machine locale).

```
library(osrm)
```

```
## Data: (c) OpenStreetMap contributors, ODbL 1.0 - http://www.openstreetmap.org/copyright
```

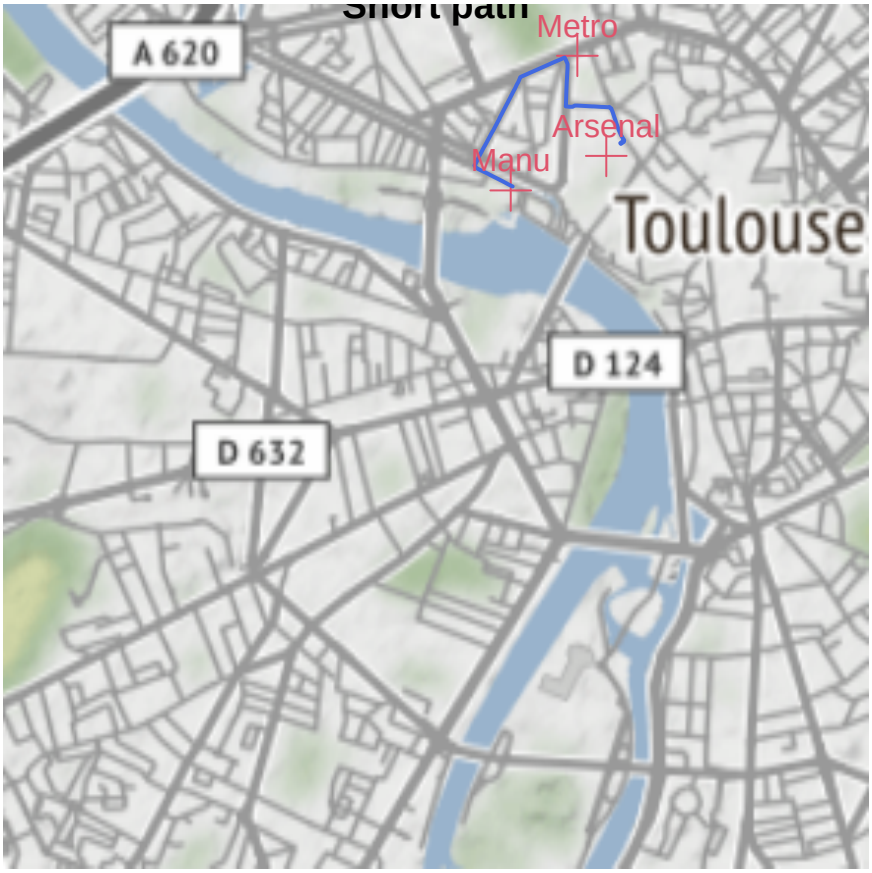
```
## Routing: OSRM - http://project-osrm.org/
```

```
## sp support will be dropped in the next major release, please use sf objects instead.
```

```
short_path <- osrmRoute(
    src = xy_sf[1, ],
    dst = xy_sf[2, ],
    returnclass = "sf",
    overview = "full"
  )
```

On représente les données :

```
plot_tiles(mtqOSM)
title("Short path")
plot(xy, col = 2, cex = 2, add = TRUE)
plot(st_geometry(short_path), add = T, col = "royalblue", lwd = 2)
text(coordinates(xy), col = 2, pos = 3,
  labels = c("Manu", "Arsenal","Metro"))
```



Exercice 14

Représenter le chemin optimal entre votre domicile et votre lieu de travail.

4 Faire des cartes avec R

Représenter des données sur une carte est un vrai challenge. Ce champ doit beacoup à Jacques Bertin (1918–2010) et son ouvrage *Sémiologie graphique. Les diagrammes, les réseaux, les cartes.* Cet article fait un résumé de l'ouvrage en question. Parmi les points relevés, on notera les suivants :

- il existe plusieurs constructions possibles d'un graphique pour une même information

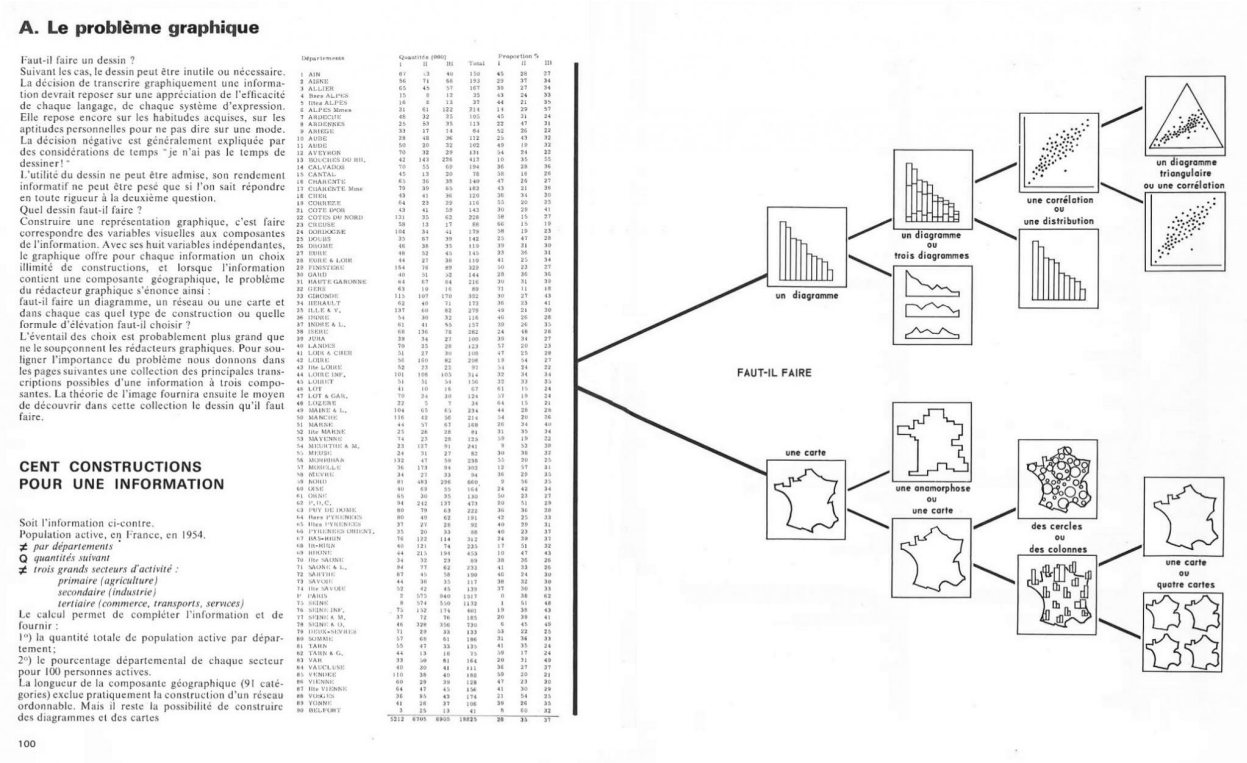


Figure 22: Extrait 1 de l'ouvrage de Bertin

- qu'est-ce qu'une représentation efficace ? "... si pour obtenir une réponse correcte et complète à une question donnée, et, toutes choses égales, une construction requiert un temps d'observation plus court qu'une autre construction, on dira qu'elle est plus efficace pour cette question" (Bertin, 1967, p. 139).
- Une carte est en 3D, sachant que les deux premières sont réservées à l'espace géographique et la troisième dimension représente une information statistique. Lorsque l'information est complexe, il faut la simplifier préalablement.
- Une carte utilise une grammaire, faite de forme, de taille, de couleurs, de hachures, d'orientations.
- La carte est dynamique : "On ne dessine plus un graphique une fois pour toutes. On le construit et on le reconstruit (on le manipule) jusqu'au moment où toutes les relations qu'il recèle ont été perçues" (Bertin, 1977, 5).

Ici, on va s'intéresser à la représentaion de l'information statistique sur des cartes choroplètes. Définition de Wikipedia : "une carte choroplète est une carte thématique où les régions sont colorées ou remplies d'un motif qui montre une mesure statistique, tels la densité de population ou le revenu par habitant. Ce type de carte facilite la comparaison d'une mesure statistique d'une région à l'autre ou montre la variabilité de celle-ci pour une région donnée".

On verra également quelques alternatives à l'utilisation de cartes choroplètes

Préparation des données : dans cette section, on va s'intéresser à la représentation de l'information

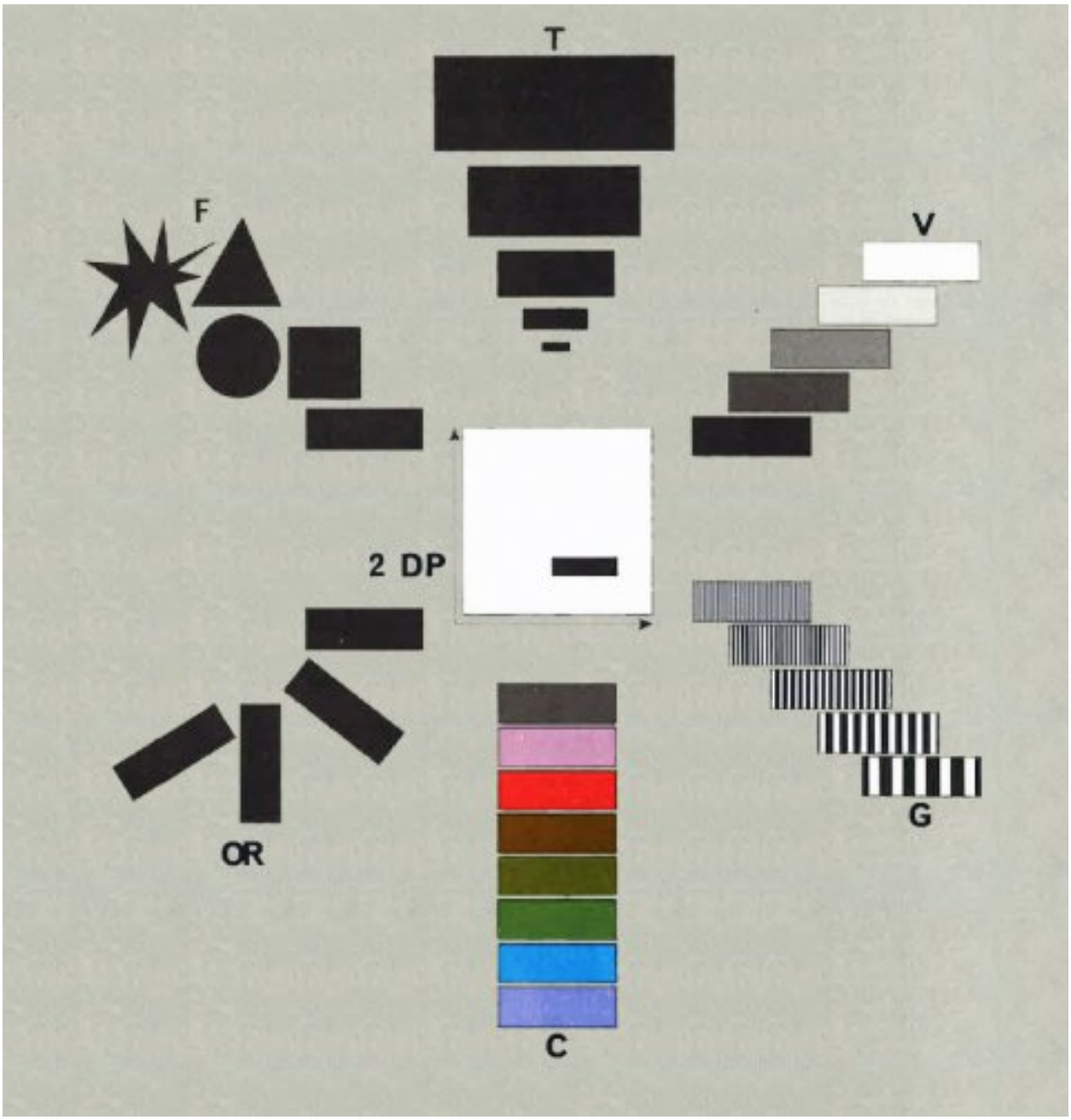


Figure 23: Extrait 2 de l'ouvrage de Bertin

statistique sur des cartes. On prendra comme données celles issues de la World Bank sur le PIB par habitant et la population des pays africains en 2015. Dans un premier temps, on prépare les données sur l'Afrique sub-saharienne :

```
pays.af <- c("Seychelles", "Equatorial Guinea", "Gabon", "Botswana", "Mauritius",
 "South Africa", "Namibia", "Angola", "Swaziland", "Congo", "Cape Verde", "Ghana",
 "Sudan", "Djibouti", "Nigeria", "Sao Tome and Principe", "Cameroon", "Lesotho",
 "Gambia, The", "Chad", "Senegal", "Kenya", "Ivory Coast", "Zambia", "Burkina Faso",
 "Tanzania, United Republic of", "Benin", "Rwanda", "Uganda", "Comoros", "Guinea-Bissau",
 "Mali", "Mozambique", "Guinea", "Ethiopia", "Madagascar", "Malawi", "Togo",
 "Sierra Leone", "Niger", "Central African Republic", "Eritrea", "Burundi", "Somalia",
 "Zimbabwe", "Liberia", "Zaire")
```

```
africa.df <- data.frame(pib = c(14745, 11279, 7382, 6800, 9260, 5735, 5033, 4167, 3689,
 1761, 3043, 1766, 2487, 2676, 2730, 1596, 1327, 1112, 661, 776, 1219, 1337, 1426,
 1338, 575, 947, 784, 728, 709, 1242, 603, 751, 590, 769, 641, 467, 381, 571, 588,
 361, 377, 715, 306, 293, 1445, 710, 497),
                     pop = c(93, 1169, 1948, 2121, 1263, 55386, 2315, 27884, 1104,
 4856, 525, 27849, 38903, 914, 181137, 199, 23298, 2059, 2086, 14111, 14578, 47878,
 23226, 15879, 18111, 51483, 10576, 11369, 38225, 777, 1737, 17439, 27042, 11432,
 100835, 24234, 16745, 7323, 7172, 20002, 4493, 4475, 10160, 13797, 13815, 4472, 76245),
                     region = c("E", "C", "C", "Au", "E", "Au", "Au", "C", "Au", "C",
 "O", "O", "E", "E", "O", "C", "C", "Au", "O", "C", "O", "E", "O", "E", "O", "E", "O",
 "E", "E", "E", "O", "O", "E", "O", "E", "E", "E", "O", "O", "O", "C", "E", "E", "E",
 "E", "O", "C"), NOM = pays.af)
```

On crée les objets "Spatial" et **sf** :

```
africa.sub_sp <- merge(world_sp[pays.af, ], africa.df, by = "NOM")
row.names(africa.sub_sp) <- as.character(africa.sub_sp$NOM)
africa.sub_sf <- merge(world_sf %>% filter(NOM %in% pays.af), africa.df, by = "NOM")
```

On les ajoute au jeu de données sur le Maghreb :

```
africa_sp <- spRbind(northAf_sp, africa.sub_sp)
africa_sf <- rbind(northAf_sf, africa.sub_sf)
```

4.1 Principe des couleurs

Harrower et Brewer (2003) présentent des palettes de couleurs adaptés aux usages suivants.Ces couleurs sont implémentés dans le package **RColorBrewer**. Il existe d'autres packages qui permettent de réaliser des palettes de couleurs, comme **colorspace** ou **viridis** mais que nous n'utiliserons pas ici.

4.1.1 Palette de couleurs séquentielles

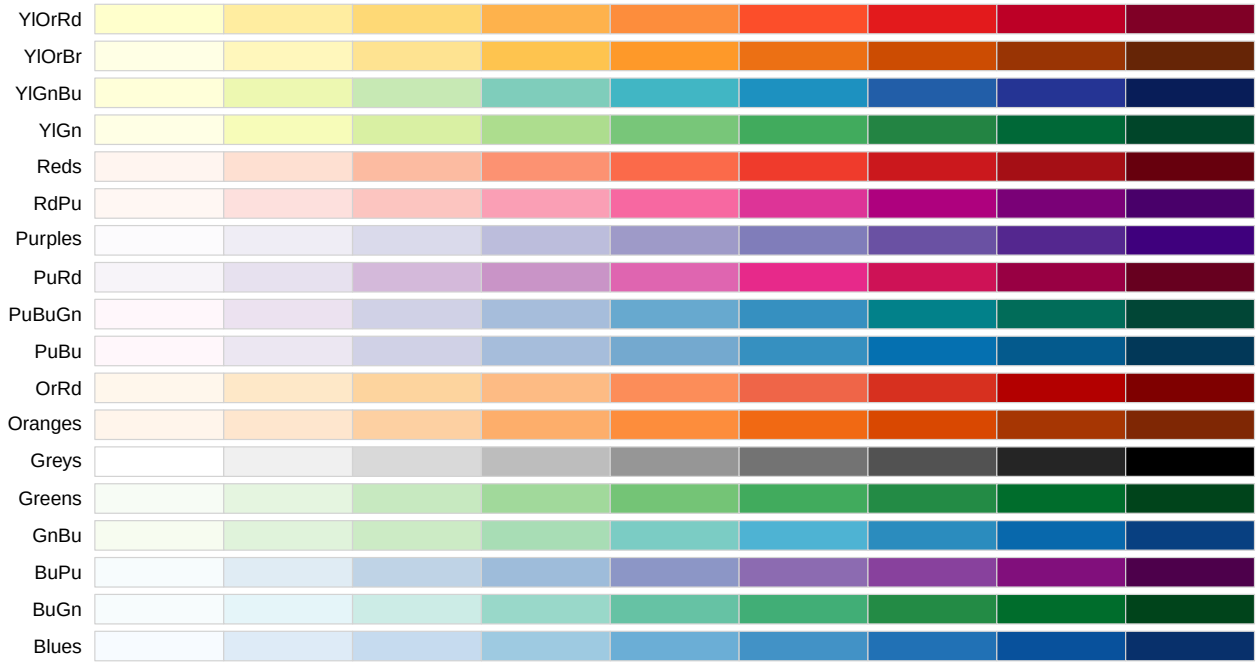
Ce type de palettes de couleurs est utilisé pour représenter des variables qualitatives ordinales (par exemple, une variable avec les modalités "un peu", "moyen", "beaucoup", "passionément") et avec une idée que les niveaux sont ordonnées des moins intéressants aux plus intéressants. Ce type de palette peut également s'appliquer à des variables quantitatives où les valeurs faibles sont jugées inintéressantes alors que les valeurs fortes le sont. Par exemple, la variable "densité de population" par commune, département ou région est souvent représentée dans des classes de type $[0, 5]; [5, 15], \ldots, [15000, +\infty]$ hab/km^2 et les classes qui intéressent sont celles avec des valeurs fortes.

L'approche de Harrower et Brewer (2003) consiste à attribuer des couleurs "claires" pour les classes jugées "faibles" et des couleurs "sombres" pour les modalités jugées "fortes" soulignant le fait qu'en cartographie "sombre" implique "plus de quantités". Sibrel et al. montrent à travers des expériences empiriques que cette perception "plus sombre, plus de quantités" tient toujours mêmes en présence d'autocorrélation spatiale.

La palette peut être basée sur une seule teinte (par exemple du bleu allant du plus clair au plus sombre) ou avec plusieurs teintes (par exemple une palette allant du jaune clair au vert sombre).

Pour jouer sur l'aspect faible / sombre d'une couleur, il est possible de jouer sur la luminosité et/ou la saturation. Harrower et Brewer (2003) définissent un ensemble de palettes de couleurs (https://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3) qui ont été obtenues en utilisant le référentiel CIE.

```
require(RColorBrewer)
display.brewer.all(type="seq")
```



L'inconvénient est que les couleurs sont prédéfinies pour chaque palette en fonction du nombre de classe. Le nombre de classe varie de 3 à 9 (parfois 11). Au-delà, les auteurs arguent que la distinction entre les classes est difficile.

Zeileis et al. (2009) proposent d'utiliser un autre référentiel et s'appuient sur une construction mathématique pour définir les couleurs. Ces couleurs sont disponibles dans le package **colorspace**.

4.1.2 Palette de couleurs qualitatives

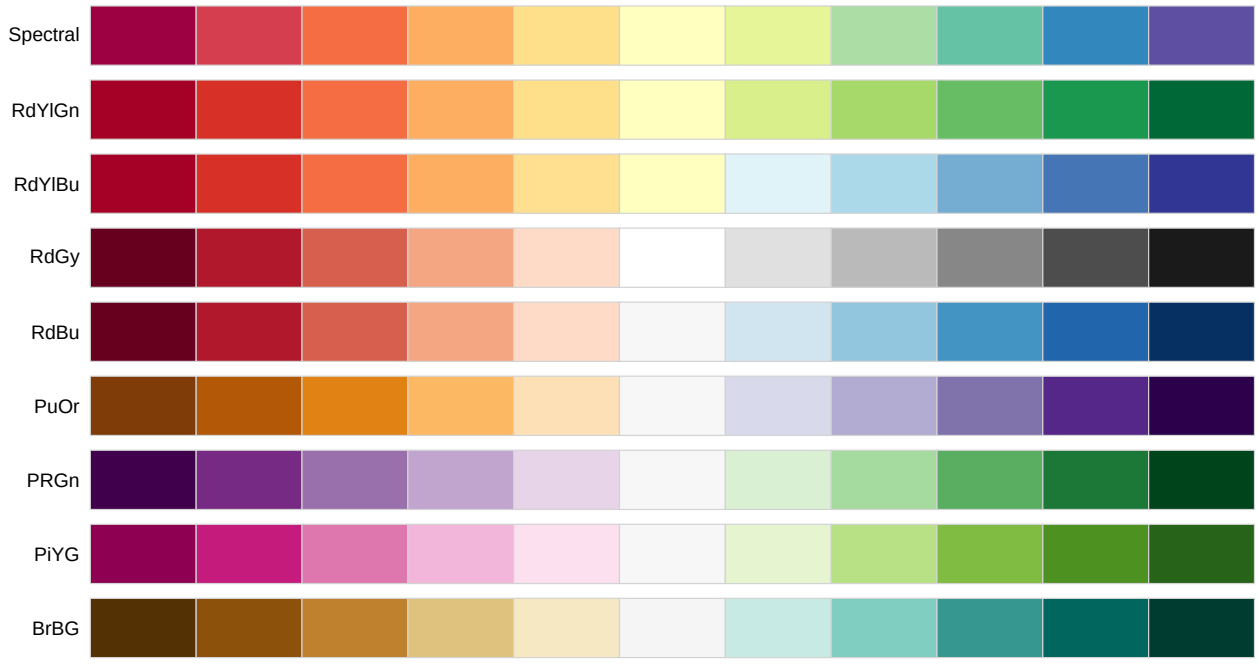
ce type de palette est utilisé pour des variables qualitatives nominales. Autrement, dit, on considère que chaque modalité a la même importance. Le graphique statistique correspondant serait le barplot ou le diagramme circulaire. Une stratégie est garder la même luminosité et la même saturation pour chaque catégorie.

```
display.brewer.all(type="qual")
```

4.1.3 Palette de couleurs divergentes

Ici, on considère une variable quantitative X et on s'intéresse à l'écart de X à une valeur centrale. Par exemple les résidus d'un modèle OLM où on s'intéresse à l'écart à la valeur 0. Les valeurs proches de la valeur centrale seront représentés par une couleur neutre. En s'éloignant d'un côté plutôt qu'un autre on utilisera des couleurs opposées (dans le cercle chromatique par exemple) et on ira vers des couleurs plus sombres plus on s'éloigne du centre

```
display.brewer.all(type="div")
```



4.2 Représenter une variable qualitative

L'objectif est de faire une carte qui ressemble à celle diffusée à ce lien : http://fr.wikipedia.org/wiki/Afrique#mediaviewer/File:Zones_Afrique.jpg

Il s'agit d'une carte choroplète, c'est-à-dire une carte où les régions sont colorées ou remplies d'un motif qui montre une mesure statistique, ici l'appartenance à une zone géographique. Cette variable qui représente les différentes zones géographiques en Afrique est la variable **region** créée précedemment.

On réalise dans un premier temps la carte dans la classe "Spatial". Il y a 5 classes et on va donc créer une palette avec 5 couleurs, de type qualitative puisque la variable est nominale.

```
pal.reg <- RColorBrewer::brewer.pal(5, "Set3")
```

On transforme ensuite la variable qualitative en **integer** où chaque entier correspondra à une couleur :

```
ind_reg <- as.numeric(factor(africa_sp$region))
```

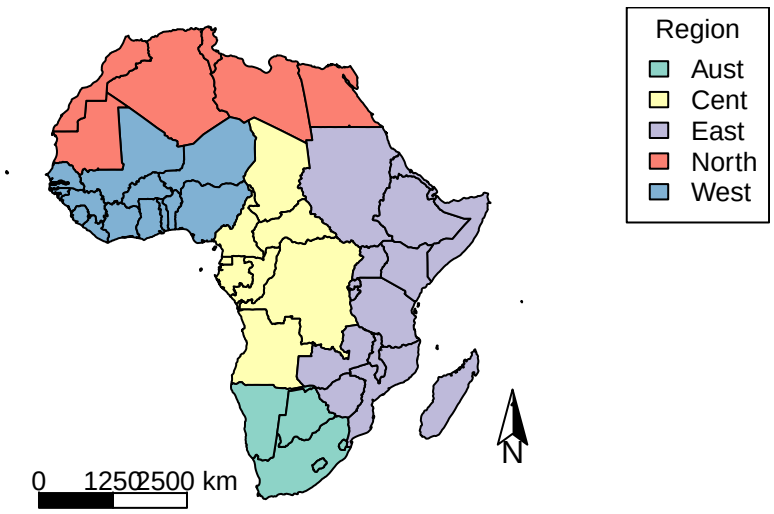
On change la transformation pour avoir un CRS adapté à la géographie du continent Africain :

```
crs_af <- CRS("+proj=sinu +lon_0=0 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m")
africa_sp <- spTransform(africa_sp, crs_af)
```

On ajoute également une boussole et la légende des échelles sur la carte en utilisant deux fonctions programmées et trouvées dans ce lien:

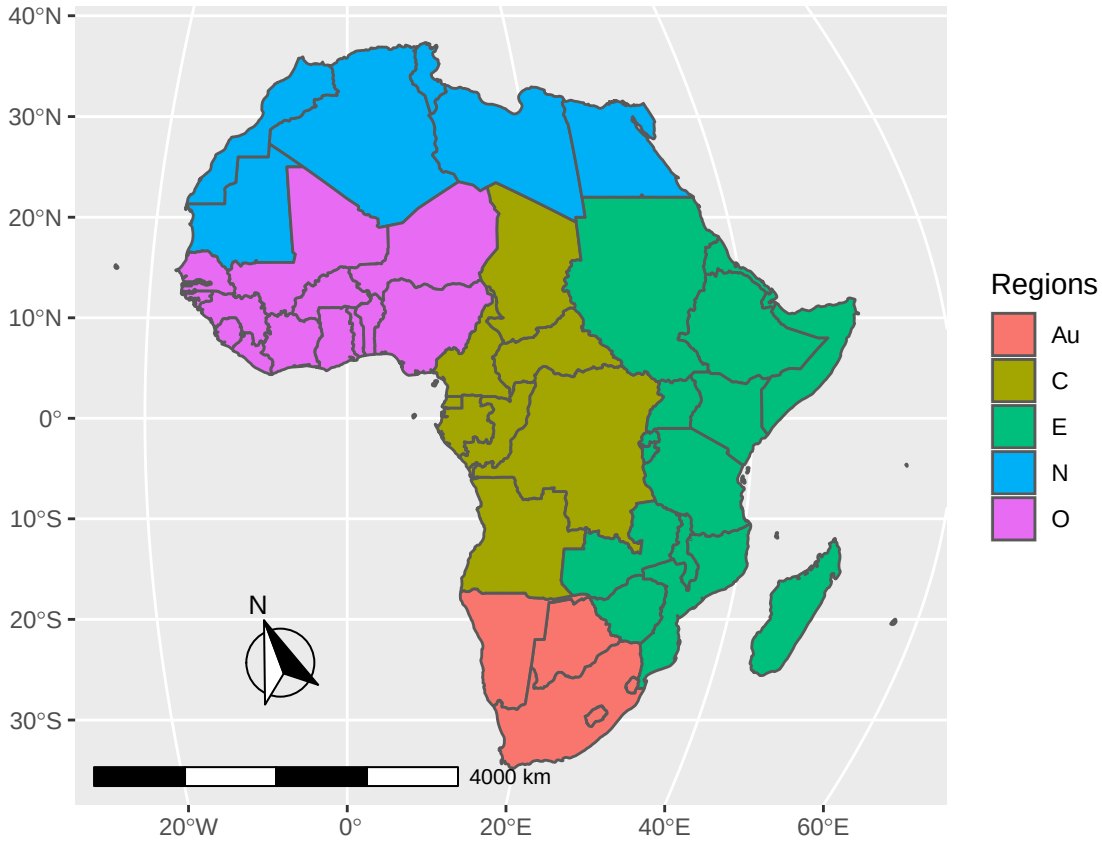
```
scalebar <- function(loc, length, unit = "km", division.cex = 0.8,
                     bg = "white", border = "black",...) {

  x <- c(0, length / 2, length) + loc[1]
  y <- c(0, length / 10, length / 9.5, length / 5.5) + loc[2]
  cols <- c("black", "white")
  for (i in 1:2) rect(x[i], y[1], x[i+1], y[2], col = cols[i])
  for (i in 1:3) segments(x[i], y[2], x[i], y[3])
  labels <- round(c(0, length / 2, length) / 1000)
  labels[3] <- paste0(labels[3], " km")
  text(x[c(1, 2, 3)], y[4], labels, adj = 0.5, cex = division.cex)
}

north.arrow <- function(x, y, h, lab = "North", lab.pos = "below") {
  polygon(c(x, x, x + h/2), c(y - (1.5*h), y, y - (1 + sqrt(3)/2) * h),
          col = "black", border = NA)
  polygon(c(x, x + h/2, x, x - h/2), c(y - (1.5*h), y - (1 + sqrt(3)/2) * h,
                                       y, y - (1 + sqrt(3)/2) * h))
  if(lab.pos == "below") text(x, y - (2.5 * h), lab, adj = c(0.5, 0), cex = 1)
  else text(x, y + (0.25 * h), lab, adj = c(0.5, 0), cex = 1.5)
}
```

```
plot(africa_sp, col = pal.reg[ind_reg])
title("Africa's regions")
legend("topright", legend = c("Aust", "Cent", "East", "North", "West"),
       cex = 0.8, title = "Region", fill = pal.reg)
north.arrow(6*10^6, -2*10^6, 5*10^5, lab = "N")
scalebar(c(-2*10^6, -4*10^6), 2500000)
```



On réalise ensuite la carte dans la classe **sf**. Le package **ggspatial** permet d'ajouter l'échelle et la direction du Nord. On a également changé le CRS pour avoir une représentation la plus adaptée possible à l'Afrique.

```
library(ggspatial)
ggplot(data = africa_sf, aes(geometry = geometry)) +
  geom_sf(aes(fill = as.factor(region)), legend="meh") +
  scale_fill_discrete("Regions") +
  coord_sf(crs = st_crs(crs_af)) +
    annotation_scale(location = "bl", width_hint = 0.5) +
    annotation_north_arrow(location = "bl", which_north = "true",
        pad_x = unit(0.75, "in"), pad_y = unit(0.5, "in"),
        style = north_arrow_fancy_orienteering)
```

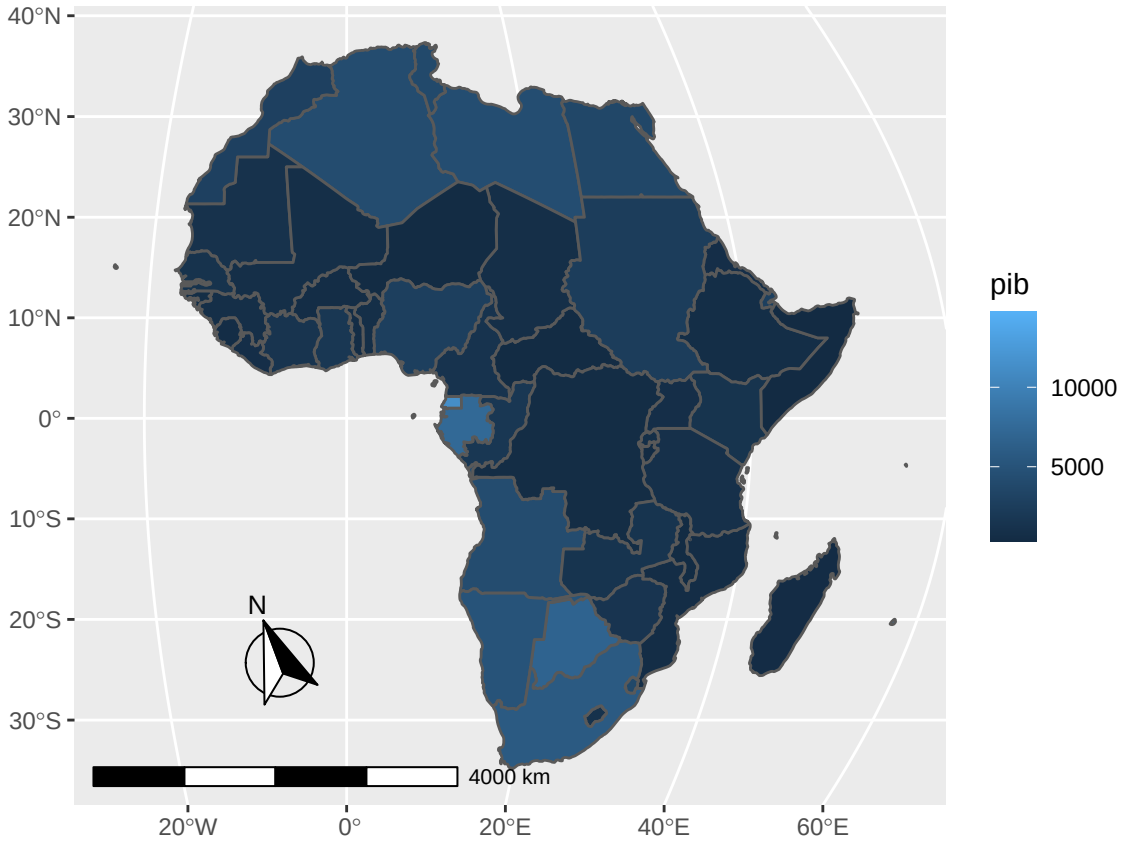


4.3 Représenter une variable quantitative

On souhaite à présent représenter le PIB par habitant des pays africains. On va dans un premier temps dessiner une carte choroplète. En utilisant la classe **sf**, les valeurs numériques sont remplacées par des couleurs, qui vont de la couleur bleu foncée pour les valeurs faibles vers la couleur bleu claire pour les valeurs fortes.

En utilisant la syntaxe à la **ggplot**, le choix des couleurs est automatique. Les couleurs sont attribuées de façon proportionnelle au dégradé de couleurs du foncé vers le clair. On choisit la variable à représenter avec l'argument `fill=`

```
ggplot(data = africa_sf, aes(geometry = geometry)) +
  geom_sf(aes(fill = pib)) +
  coord_sf(crs = st_crs(crs_af)) +
    annotation_scale(location = "bl", width_hint = 0.5) +
    annotation_north_arrow(location = "bl", which_north = "true",
        pad_x = unit(0.75, "in"), pad_y = unit(0.5, "in"),
        style = north_arrow_fancy_orienteering)
```



4.3.1 Discrétiser une variable quantitative

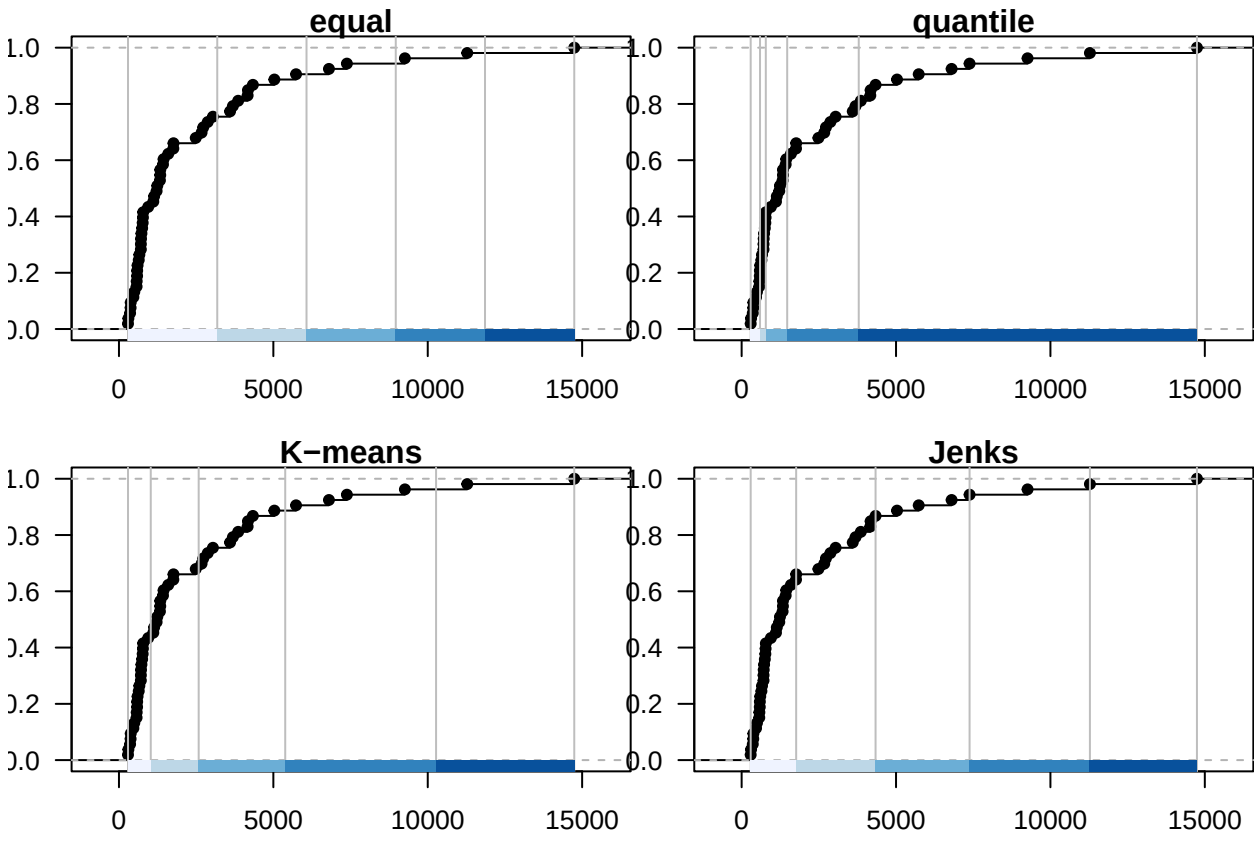
On a vu dans l'exemple précédent que les couleurs étaient attribuées de façon proportionnelle au dégradé de couleurs du foncé vers le clair. De nombreuses cartes prennent le parti de discrétiser la variable quantitative en un certain nombre de classes et associer ensuite à chaque classe une couleur différente. Pour discrétiser une variable quantitative, il existe plusieurs méthodes implémentées dans le package **classInt** (Bivand, 2019). Les choix principaux (argument `style` de la fonction `classIntervals()`) sont basés sur les méthodes suivantes :

- les amplitudes des classes créées sont toutes égales (`style = "equal"`)
- une classe est obtenue à partir des quantiles d'ordre i/K et $(i+1)/K$, $i = 0, ..., K$ où K est le nombre de classes choisies par l'utilisateur (`style = "quantile"`).
- la méthode pour générer les classes est basée sur l'algorithme des K-means (`style = "kmeans"`).
- la méthode pour générer les classes est basée sur l'algorithme de Jenks (`style = "jenks"`) dont l'algorithme (comme celui des K-means) consiste à maximiser la variance inter-classe et minimiser la variance intra-classe.

On représente ici les fonctions de répartition empiriques de la variable PIB par habitant qu'on cherche à discrétiser en 5 classes, ainsi que les classes obtenues selon le choix de la méthode de discrétisation et représentées avec des couleurs différentes sur l'axe des abscisses du graphique. Le package **RColorBrewer** (Neuwirth, 2014) permet de créer des palettes de couleurs avec des dégradés d'une même couleur.

```
pib <- africa_sp$pib
library(classInt)
library(RColorBrewer)
pal1 <- brewer.pal(5, "Blues")
opar <- par(mfrow = c(2, 2), mar = c(3, 2, 1, 0), las = 1)
plot(classIntervals(pib, 5, "equal"), pal = pal1, main = "equal")
plot(classIntervals(pib, 5, "quantile"), pal = pal1, main = "quantile")
```

```
plot(classIntervals(pib, 5, "kmeans"), pal = pal1, main = "K-means")
plot(classIntervals(pib, 5, "jenks"), pal = pal1, main = "Jenks")
```




```
par(opar)
```

Remarque : selon le choix de la méthode de discrétisation, un pays ne se trouvera pas nécessairement dans le même groupe. Par exemple, avec un pib de 5033 dollars par habitants, la Namibie est située dans le groupe 1 (avec les valeurs les plus élevées) en utilisant la méthode des quantiles alors qu'il se trouve dans le groupe 4 avec la méthode des classes d'amplitudes égales.

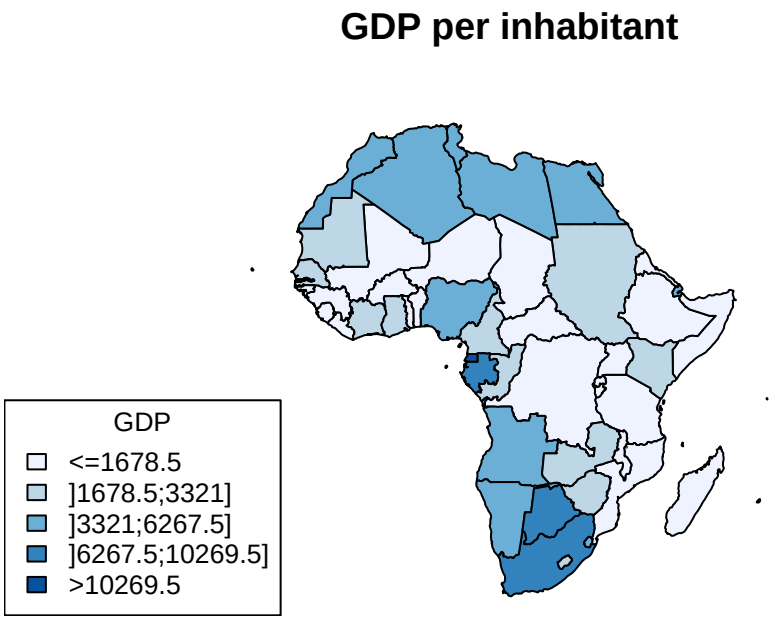
La méthode des quantiles, du fait de sa construction (chaque classe possède environ n/K observations), peut regrouper dans une même classe des observations très différentes (c'est le cas ici de la classe 1). La méthode basée sur les classe d'amplitudes égales présente quant à elle l'inconvénient de construire potentiellement des classes vides.

C'est pourquoi on aurait tendance à recommander d'utiliser une des deux méthodes (K-means ou Jenks) basées sur les critères de minimisation de variance intra. On choisit ici de représenter la carte choroplète obtenue à partir la méthode des K-means.

La fonction `findInterval()` permet d'attribuer une observation à une classe en utilisant le découpage obtenu avec la fonction `classIntervals()`

```
bk <- round(classIntervals(pib, 5, "kmeans")$brks, digits = 1)
ind <- findInterval(pib, bk, all.inside = TRUE)
plot(africa_sp, col = pal1[ind])
decoup <- c("<=1678.5", "]1678.5;3321]", "]3321;6267.5]",
            "]6267.5;10269.5]", ">10269.5")
legend("bottomleft", legend = decoup, cex = 0.8, title = "GDP",
```

```
       fill = pal1)
SpatialPolygonsRescale(layout.north.arrow(1), offset = c(50, -30),
                       scale = 5, plot.grid = F)
title("GDP per inhabitant")
```



Remarque : les cartes choroplètes présentent toutes l'inconvénient que les polygones de petite taille passent souvent au-travers de la lecture de la carte. En l'occurence, le pays avec le PIB/habitant le plus élevé du contient africain est les Seychelles et on ne le visualise pas sur la carte du fait de sa petite taille. On propose ici deux solutions :

- utiliser la même forme pour chaque pays en essayant de préserver la localisation géographique
- utiliser une représentation avec des points proportionnels à l'information statistique

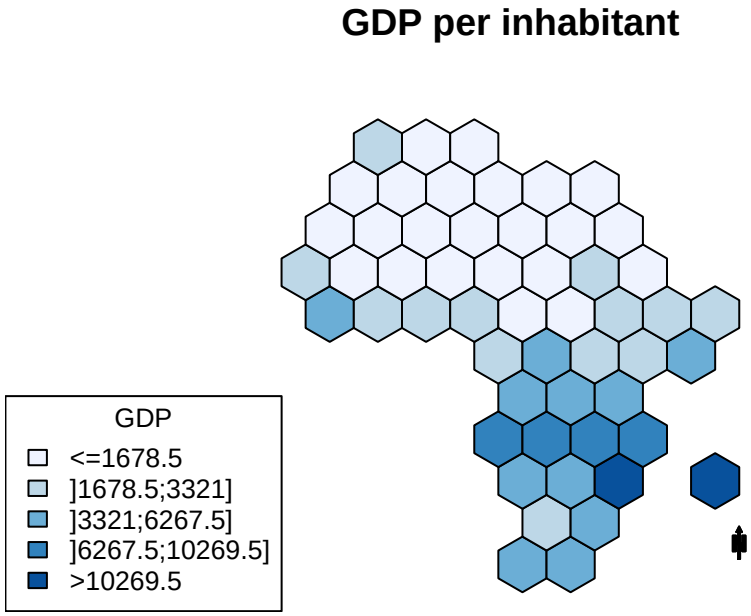
4.3.2 Utiliser des formes géographiques identiques de type hexagonne

Pour cela, on va utiliser le package **geogrid** pour faire la transformation des polygones en hexagonne. On s'est inspiré du blog suivant qui représente les données de Covid par département.

```
library(geogrid)
africa_cells_hex <- calculate_grid(shape = africa_sf, grid_type = "hexagonal",
                                   seed = 3, verbose = F)
africa_hex <- assign_polygons(africa_sf, africa_cells_hex) %>%
  st_set_crs(2154)
```

On utilise ensuite la même syntaxe que tout à l'heure, mais appliquée cette fois-ci aux hexagonnes :

```
plot(st_geometry(africa_hex), col = pal1[ind])
decoup <- c("<=1678.5", "]1678.5;3321]", "]3321;6267.5]",
            "]6267.5;10269.5]", ">10269.5")
legend("bottomleft", legend = decoup, cex = 0.8, title = "GDP",
       fill = pal1)
SpatialPolygonsRescale(layout.north.arrow(1), offset = c(50, -30),
                       scale = 5, plot.grid = F)
title("GDP per inhabitant")
```



Un inconvénient de cette simplification est qu'il y a nécessairement une perturbation de la localisation géographique des pays, cela pouvant créer de la confusion pour identifier plus particulièrement une observation.

Une autre solution est l'utilisation de cartes avec des cercles proportionnels à la variable statistique.

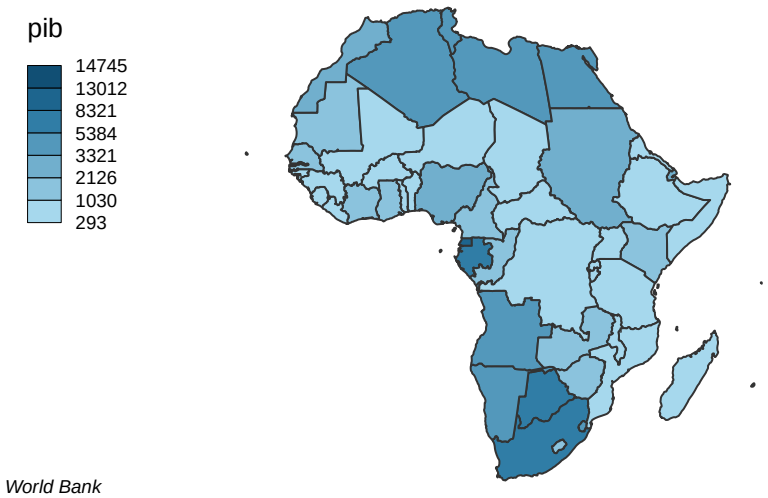
4.3.3 Package mapsf et cartography

Giraud et Lambert (2016) sont deux cartographes qui ont développé le package **cartography** (voir vignette https://cran.r-project.org/web/packages/cartography/vignettes/cartography.html) qui permet de réaliser des cartes très facilement à l'aide la fonction *choroLayer()* (le package fonctionne à la fois sur les classes d'objet "Spatial" et **sf**).

```
library("cartography")
```

```
## This project is in maintenance mode.
## Core functionalities of `cartography` can be found in `mapsf`.
## https://riatelab.github.io/mapsf/
```

```
choroLayer(spdf = africa_sp, var = "pib", method = "fisher-jenks", legend.pos = "topleft")
layoutLayer(title = "GDP in Africa", sources = "World Bank", frame = TRUE,
            col = "NA",  scale = NULL)
```

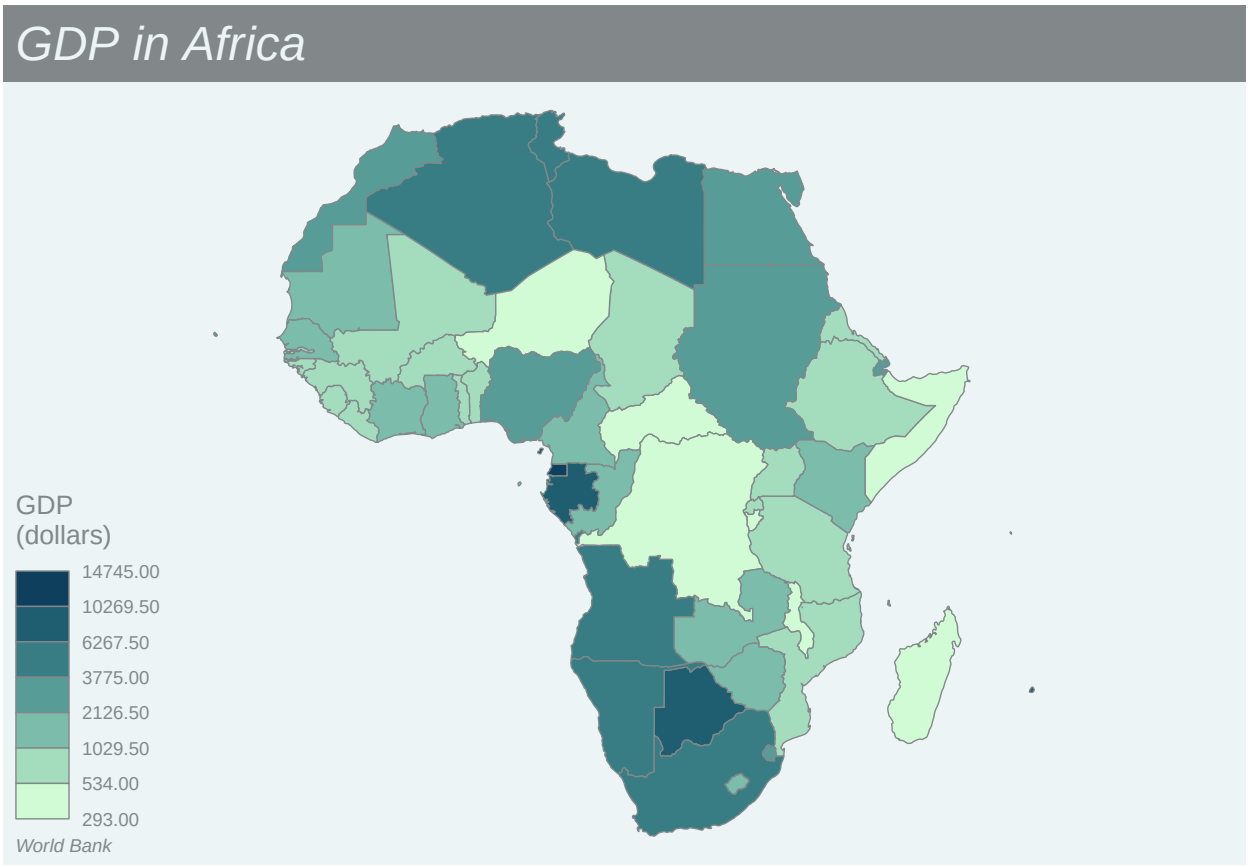


Le package **mapsf** est en train de succéder au package **cartography** et s'utilise en utilisant le même principe que **tidyverse** sur des objets de type **sf** exclusivement:

```
library(mapsf)
mf_theme("agolalight")
sf_use_s2(FALSE)
```

```
## Spherical geometry (s2) switched off
```

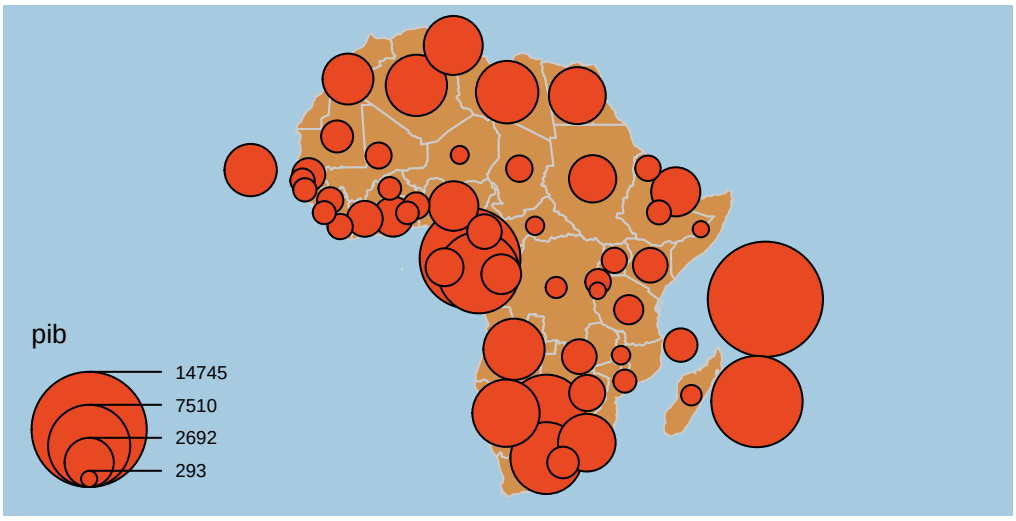
```
africa_sf |>
  mf_map(var = "pib", type = "choro",
         pal = "Dark Mint",
         breaks = "kmeans",
         leg_title = "GDP\n(dollars)")
mf_credits("World Bank")
mf_title("GDP in Africa")
```



4.3.4 Carte avec des tailles de cercles proportionnelles à la variable quantitative

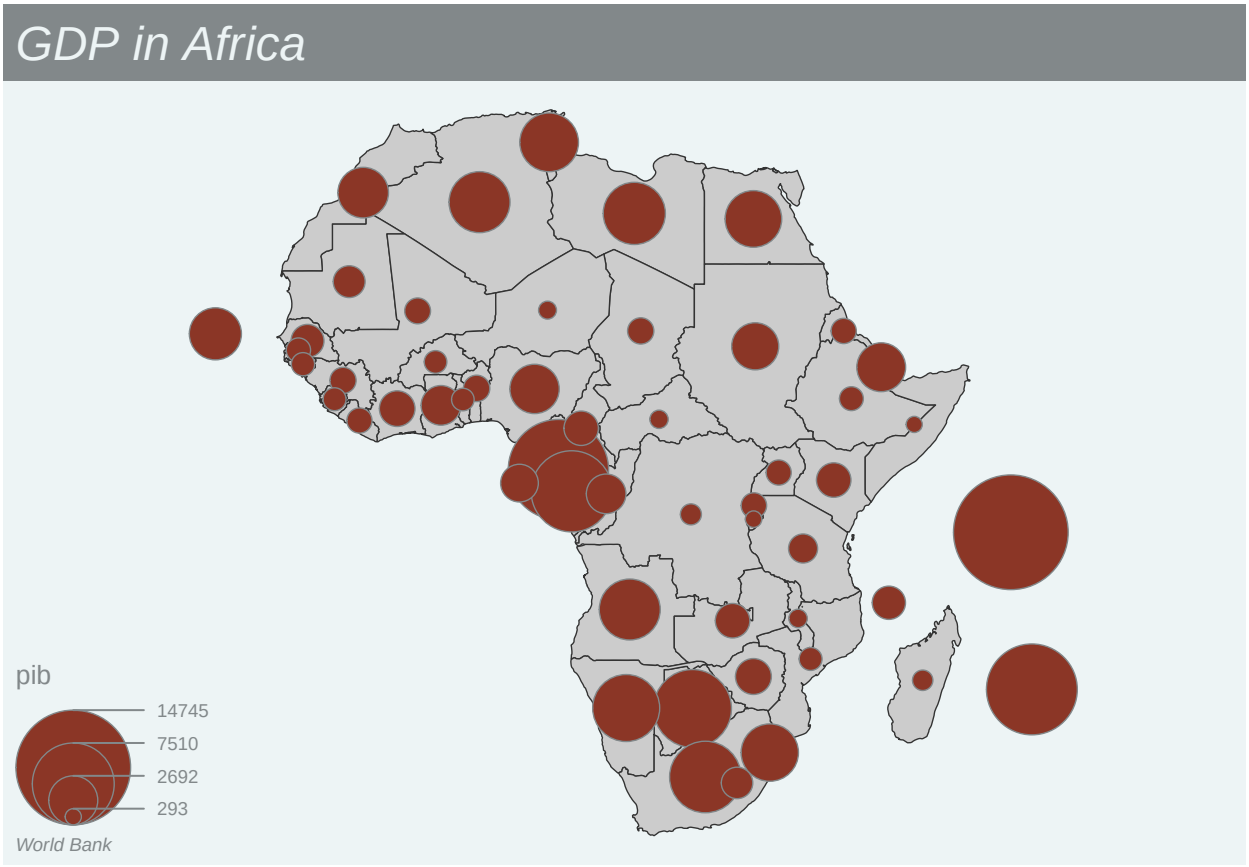
Ce type de représentation, également appelé “buble plot” peut être très utile lorsque l’aire des unités spatiales n’est pas homogène. En effet, le lecteur de la carte va se focaliser sur la grosseur des cercles et ne sera pas influencé par la taille des polygones. Pour les réaliser, on utilise la fonction `propSymbolsLayer()` du package **cartography**.

```
plot(africa_sp, border = NA, col = "NA", bg = "#A6CAE0")
plot(world_sp, col = "#E3DEBF", border = NA, add = T)
plot(africa_sp, col = "#D1914D", border = "grey80", add = T)
propSymbolsLayer(spdf = africa_sp, var = "pib" )
```



La version **mapsf**:

```
library(mapsf)
sf_use_s2(FALSE)
africa_sf |>
  mf_map() |>
  mf_map(var = "pib", type = "prop")
mf_credits("World Bank")
mf_title("GDP in Africa")
```



Remarque : en utilisant des couleurs d'une part et des cercles de taille proportionnelle, on est capable de représenter deux variables statistiques en même temps. Dans ce cas-là, il serait possible soit de colorier les polygones, soit les points.

Exercice 15

En utilisant vos données, représenter sur une même carte deux variables statistiques différentes à l'aide couleurs et de cercles proportionnels. A défaut, on pourra utiliser les données sur les villes de plus de 3 millions d'habitants et représenter la population en 2020 d'une part et le taux de croissance annuel moyen d'autre part.

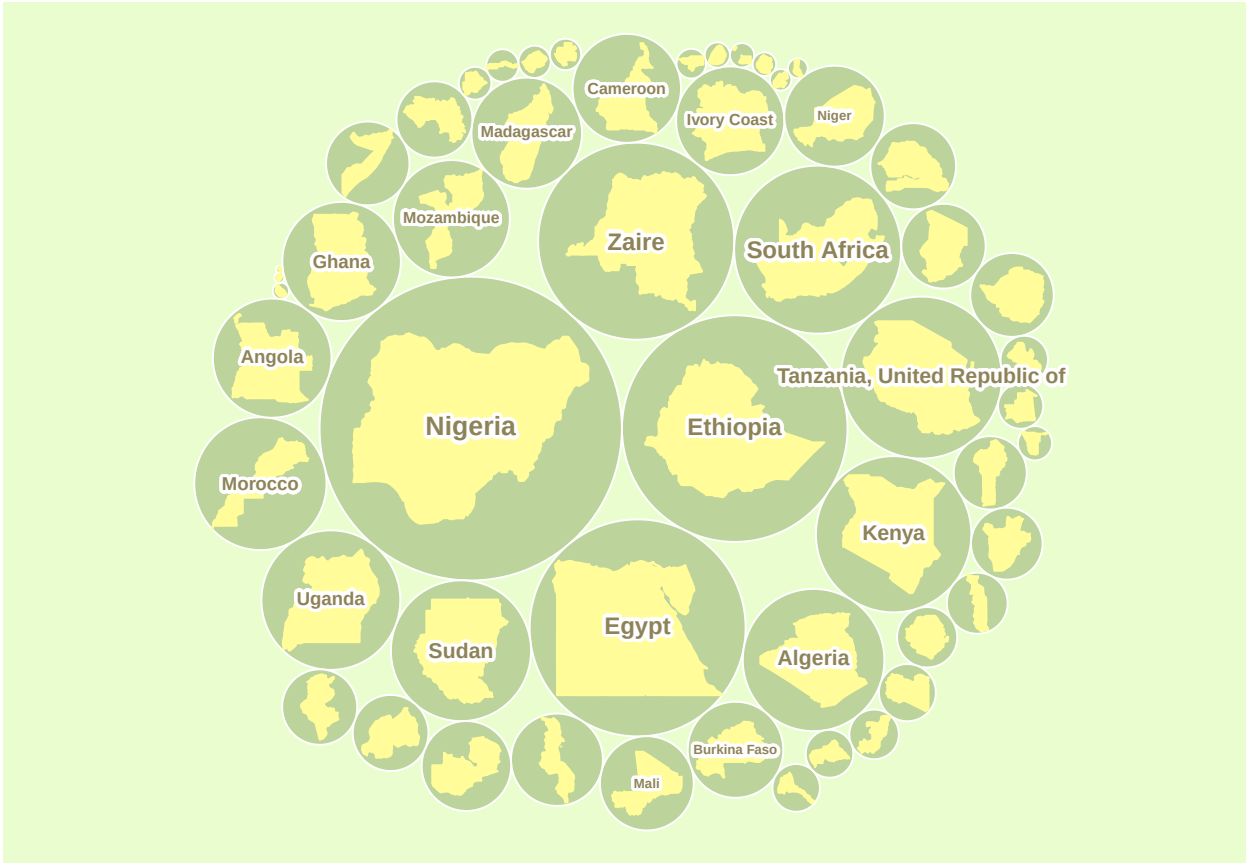
4.3.5 Autres types de représentation

Timothée Giraud propose plusieurs types de représentations différentes (représentation des frontières, données de flux, carroyages, etc) sur son site web https://rgeomatic.hypotheses.org/. Parmi celles-ci, le package **popcircle** (Giraud, 2019) dans la même optique que les "bubble plot", mais les objets géographiques sont ordonnés selon la taille de la variable observée et non plus selon leurs coordonnées géographiques. Dans l'exemple suivant, on s'intéresse à la variable population.

```
library(popcircle)
res <- popcircle(x = africa_sf, var = "pop")
```

```
## dist is assumed to be in decimal degrees (arc_degrees).
```

```
circles <- res$circles
shapes <- res$shapes
par(mar = c(0, 0, 0, 0))
plot(st_geometry(circles), col = "#bcd39c",
     border = "white", bg = "#eafdcf")
plot(st_geometry(shapes), col = "#fffc99",
     border = "#fffc99", add = TRUE)
labelLayer(x = circles[1:20,], txt = "NOM",
           halo = TRUE, col ="#8e8358",
           cex = seq(.8,.4, length.out = 20),
           font = 2, bg = "white", r = .15,
           overlap = FALSE)
```



Tennekes (2018) propose avec le package **tmaptools** de réaliser des cartes en ajoutant des fonds de cartes provenant de OpenStreetMap ou GoogleMaps. La syntaxe utilisée dans son package est différente de ce qui a été vu jusqu'à présent (voir par exemple https://mtennekes.github.io/downloads/presentations/tmap_openeo_muenster.pdf).

Le package **leaflet** (Cheng et al., 2019) permet d'ouvrir les fonds de carte OpenStreetMap sur une interface web en JavaScript et d'y superposer des données spatiales.

```
library(leaflet)
m <- leaflet() %>%
  addTiles() %>%  # Add default OpenStreetMap map tiles
  addMarkers(lng = 1.432022, lat = 43.605417, popup = "MT")
```

Le package **mapview** (voir https://github.com/r-spatial/mapview) est dans le même registe :

```
library(mapview)
mapview(seisme.jp_sf)
```

Les packages **spacetime** et **stars** permettent de manipuler/représenter des données spatio-temporelles.

Ce blog propose de représenter des cartes comme des peintures.

4.3.6 Heat map

La problématique est la suivante : on a observé une variable X sur un certain nombre de points dans une zone S et on souhaite interpoler X sur toute la carte.

Dans ce paragraphe, nous allons utiliser des fonctions du package **spatstat** (Baddeley et al., 2015), dédié plus particulièrement aux processus ponctuels spatiaux, pour faire de l'interpolation spatiale.

Dans un premier temps, il faut utiliser la norme propre à ce package, qui est la classe **ppp**. Cela implique de construire une fenêtre d'observation qui sera ici l'Afrique, sous forme d'objet de classe **owin** :

```
library("spatstat")
library("maptools")
af <- aggregate(africa_sp, list(fr = rep("Af", nrow(africa_sp))), FUN = length)
S <- as(af, "owin")
```

On définit ensuite un objet de type **ppp** qui comprend les points observés et la zone qui les délimite :

```
af_ppp <- as.ppp(coordinates(africa_sp), W = S)
```

```
## Warning: 1 point was rejected as lying outside the specified window
```

On associe à chaque point du processus ponctuel une marque (c'est-à-dire une variable), ici les crimes contre les personnes (le processus ponctuel est alors dit marqué) :

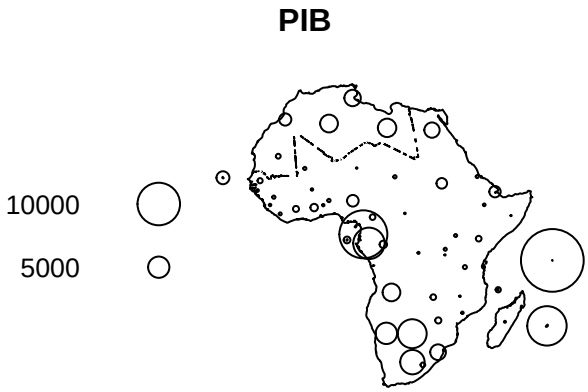
```
marks(af_ppp) <- africa_sp$pib[-25]
```

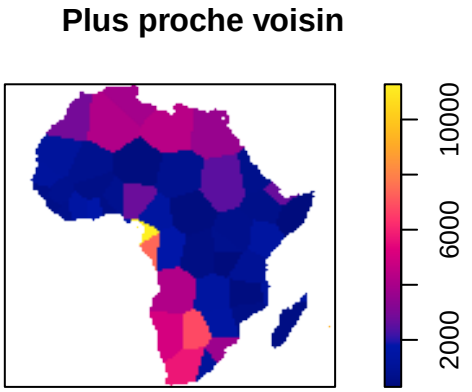
Les méthodes possibles pour faire de l'interpolation spatiale sont les suivantes :

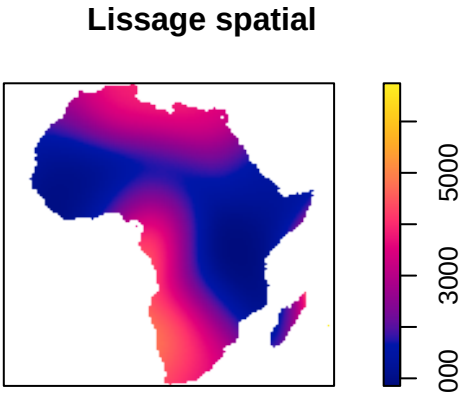
- méthode basée sur le plus proche voisin (fonction `nnmark()`) : on attribue à une localisation spatiale $u = (long, lat)$ la valeur de la marque observée sur le plus proche point observé.
- méthode basée sur un lissage non paramétrique, en utilisant un noyau gaussien par défaut (fonction `Smooth.ppp()`). Si on note $x_1, \ldots, x_n$ les valeurs observés sur les sites $s_1, \ldots, s_n$, la valeur lisée au point u vaut : $g(u) = \frac{\sum K(u-s_i)x_i}{\sum K(u-s_i)}$
- méthode basée sur l'inverse de la distance (fonction `idw()`). Si on note $x_1, \ldots, x_n$ les valeurs observés sur les sites $s_1, \ldots, s_n$, la valeur lissée au point u vaut : $g(u) = \frac{\sum w_i x_i}{\sum w_i}$ uù $w_i = 1/d(u, s_i)^p$ où p est choisi par l'utilisateur.

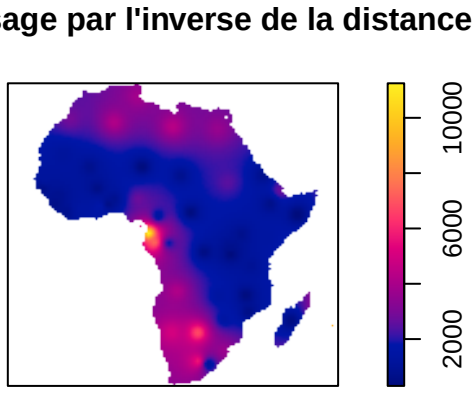
Nous illustrons ici ces méthodes sur la variable crimes contre les personnes.

```
op <- par(mfrow = c(2, 2), mar = c(0, 0, 3.3, 0))
plot(af_ppp, main = "PIB")
plot(nnmark(af_ppp), main = "Plus proche voisin")
plot(Smooth(af_ppp), main = "Lissage spatial")
plot(idw(af_ppp, power = 2), main = "Lissage par l'inverse de la distance")
```










```
par(op)
```

5 Bibliographie

Baddeley A., Rubak E. et Turner R. (2015). *Spatial Point Patterns: Methodology and Applications with* ***R***. London: Chapman and Hall/CRC Press, 2015. URL http://www.crcpress.com/Spatial-Point-Patterns-Methodology-and-Applications-with-R/Baddeley-Rubak-Turner/9781482210200/

Bertin J. (1973). *Sémiologie Graphique. Les diagrammes, les réseaux, les cartes.* EHESS, Paris.

Bivand R. (2019). **classInt**: Choose Univariate Class Intervals. **R** package version 0.4-2.

Bivand R. et Rundel C. (2019). **rgeos** : Interface to Geometry Engine - Open Source ('GEOS'). **R** package version 0.5-1. https://CRAN.R-project.org/package=rgeos

Bivand R.S., Pebesma E., Gómez-Rubio V. (2013). *Applied Spatial Data Analysis with* ***R***. Springer-Verlag.

Cambon J., Hernangómez D., Belanger C., Possenriede D. (2021). **tidygeocoder**: Geocoding Made Easy. **R** package version 1.0.3. DOI: 10.5281/zenodo.4686074. URL: https://CRAN.R-project.org/package=tidygeocoder.

Joe Cheng, Bhaskar Karambelkar et Yihui Xie (2019). **leaflet**: Create Interactive Web Maps with the JavaScript 'Leaflet' Library. **R** package version 2.0.3. https://CRAN.R-project.org/package=leaflet

Cressie N. (1993). *Statistics for Spatial Data, Revised Edition.* John Wiley & Sons.

Déjean, S. et Laurent, T. (2019). ***R*** *avancé.* http://www.thibault.laurent.free.fr/cours/R_avance

Gimond M. (2019). Intro to GIS and Spatial Analysis. https://mgimond.github.io/Spatial/index.html

Giraud T. (2017). **photon**: Geocode Locations with Photon API. **R** package version 1.1. https://github.com/rCarto/photon

Giraud T. (2019). **osrm**: Interface Between **R** and the OpenStreetMap-Based Routing Service OSRM. **R** package version 3.3.2. https://CRAN.R-project.org/package=osrm

Giraud T. (2019). **popcircle**: Proportional Circles and Shapes. **R** package version 0.1.0.

Giraud T. (2021). maptiles: Download and Display Map Tiles. R package version 0.3.0. https://CRAN.R-project.org/package=maptiles

Giraud T. (2021). mapsf: Thematic Cartography. R package version 0.3.0. https://CRAN.R-project.org/package=mapsf

Giraud, T. et Lambert, N. (2016). **cartography**: Create and Integrate Maps in your **R** Workflow. *JOSS*, **1(4)**. doi: 10.21105/joss.00054.

Harrower M. et Brewer C. (2003). ColorBrewer.org: An online tool for selecting colour schemes for maps. *Cartographic Journal*, **40**.

Hijmans R.J. (2019). **raster**: Geographic Data Analysis and Modeling. **R** package version 2.9-23. https://CRAN.R-project.org/package=raster

Robert J. Hijmans (2021). terra: Spatial Data Analysis. R package version 1.4-22. https://CRAN.R-project.org/package=terra

Neuwirth E. (2014). **RColorBrewer**: ColorBrewer Palettes. **R** package version 1.1-2. https://CRAN.R-project.org/package=RColorBrewer

Pebesma, E.J. et R.S. Bivand (2005). Classes and methods for spatial data in **R**. ***R*** *News*, **5** (2), https://cran.r-project.org/doc/Rnews/

Pebesma E. (2018). Simple Features for **R**: Standardized Support for Spatial Vector Data. *The **R** Journal*, **10** (1), 439-446, https://doi.org/10.32614/RJ-2018-009

Pebesma E. (2021). **stars**: Spatiotemporal Arrays, Raster and Vector Data Cubes. **R** package version 0.5-1. https://CRAN.R-project.org/package=stars

Tennekes M (2018). **tmap**: Thematic Maps in **R**. *Journal of Statistical Software*, **84**(6), 1-39. doi: 10.18637/jss.v084.i06 (URL: https://doi.org/10.18637/jss.v084.i06).

Tennekes M (2019). **tmaptools**: Thematic Map Tools. **R** package version 2.0-2. https://CRAN.R-project.org/package=tmaptools

Wickham H, François R., Henry L. et K. Müller (2019). **dplyr**: A Grammar of Data Manipulation. **R** package version 0.8.3. https://CRAN.R-project.org/package=dplyr