

# Cartography with R

## M2 Statistics and econometrics

Thibault LAURENT  
thibault.laurent@tse-fr.eu

Toulouse School of Economics, CNRS

5th and 13th of January, 2021

## Introduction

### Goals

### Example

### Softwares

## Cartography

## Spatial Data with R

## Mapping

# Goals of cartography

*“Cartography is the art and science of making maps”*

- ▶ Communicate geographical information graphically.
- ▶ The look of a map depends on the needs of the audience and the point that you aim to convey.

As statisticians, we will try to keep in mind that the spatial data are associated to observed data :

Individuals	Variables		
	1	...	$p$
1	$x_{11}$	...	$x_{1p}$
2	$x_{21}$	...	$x_{2p}$
$\vdots$	$\vdots$	...	$\vdots$
$n$	$x_{n1}$	...	$x_{np}$

## Introduction

Goals

Example

Softwares

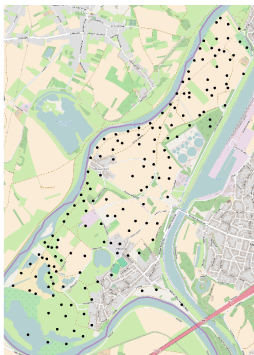
## Cartography

## Spatial Data with R

## Mapping

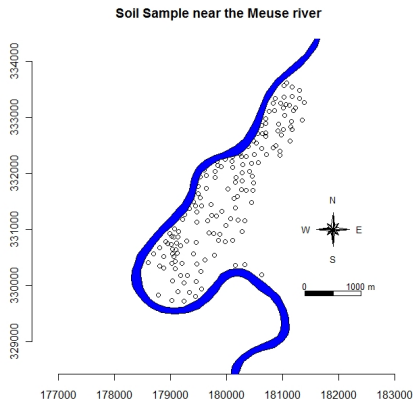


# Soil measurement collected in a plain of the river Meuse

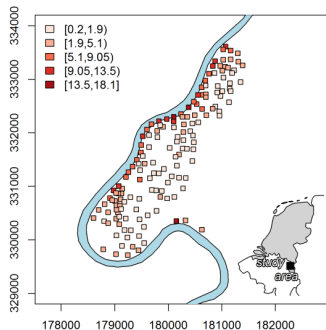


- ▶ Spatial coordinates are given by the GPS (Global Positioning System).
- ▶ For each location, measurements are given by the biologist.

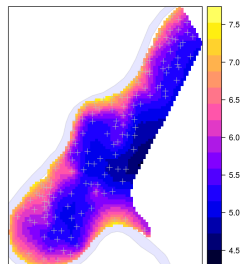
# Representation of the geographical information



# Representation of the statistical information



log(zinc); universal kriging using sqrt(dist to Meuse)



## Introduction

Goals

Example

Softwares

## Cartography

## Spatial Data with R

## Mapping

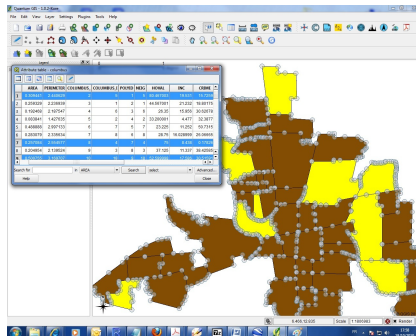
# What is GIS?

GIS (Geographic Information System; in French, Système d'Information Géographique, SIG) : computer system designed to capture, store, manipulate, analyze, manage, and present all types of spatial or geographical data.

- ▶ Free softwares : QGIS (<http://www.qgis.org/fr/site/>), gvSIG, Kartoza.
- ▶ Commercial softwares : ArcGIS, MapInfo.

# Example of GIS

GIS can be used for databases management, and its tools are better suited to some analyses and operations. But for data analysis, it is preferable to use another software.



- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

# General information

- ▶ Download presentation, R code (file `codes_carto.R`) and databases at <http://www.thibault.laurent.free.fr/pedago.html>.

- ▶ Required packages:

```
> install.packages(c("cartography", "classInt", "devtools",
                     "dismo", "ggmap", "GISTools", "lwgeom",
                     "maptools", "mapview", "OpenStreetMap", "osmar",
                     "raster", "RColorBrewer", "tmaptools",
                     "rgdal", "rgeos", "scales", "sf",
                     "tidyverse"))
```

```
> devtools::install_github('rCarto/photon')
```

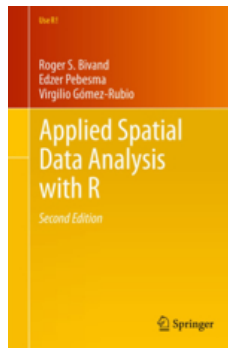
```
> devtools::install_github("rCarto/popcircle")
```

- ▶ Set working directory: indicates the path which contains the file `codes_carto.R` and folder `Donnees`:

```
> setwd("K:/...")
```



# Bibliography

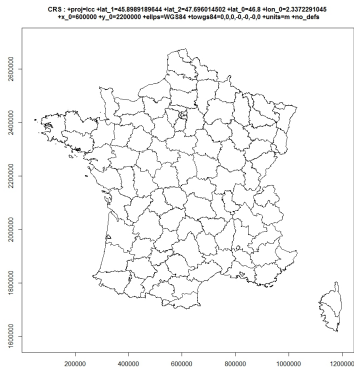
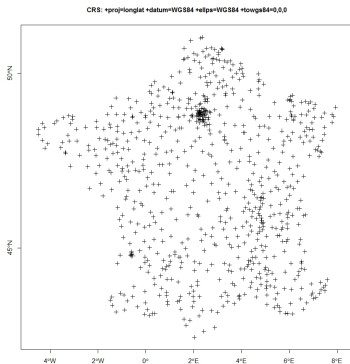


<http://www.asdar-book.org/>

<https://geocompr.robinlovelace.net/>



# Why should I know the basics in cartography?



## Choose the shape of earth

## Introduction

## Cartography

Choose the shape of earth

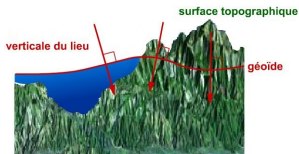
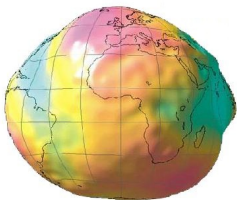
Geographic coordinate system

Projected coordinate system

## Spatial Data with R

## Mapping

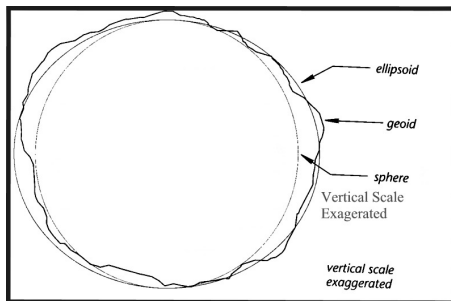
# Shape of earth



**Geoid** is the mathematically “true” shape of Earth. It represents a motionless global ocean but takes into account the effects of the Earth’s rotation, weight difference resulting from the position of mountains and ocean trenches, and uneven mass distribution and density variations in the planet’s interior.

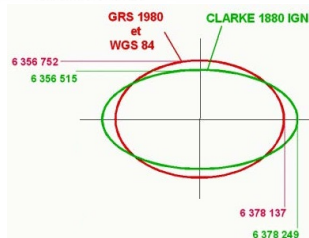
# How to simplify the geoid?

- Use of a mathematical model such as spheroids or ellipsoids.



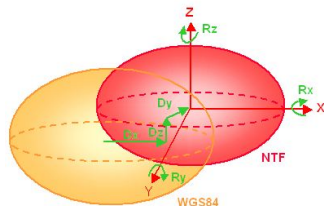
- Remark : maximum distance observed between Geoid and the ellipsoid is supposed to be the most representative  $\sim 100$  meters.

## Example of Ellipsoid



- ▶ **IAG GRS80**  
 $a = 6\,378\,137,0m$  and  
 $b = 6\,356\,752,314\,140m$
- ▶ **WGS 84**  
 $a = 6\,378\,137,0m$  and  
 $b = 6\,356\,752,314\,245m$
- ▶ **Clarke 1880**  
 $a = 6\,378\,249,2m$  and  
 $b = 6\,356\,515,0m$

# Global V.S. Local Datum



1. Global Datum : **WGS 84** (World Geodesic System 84). Used by GPS
  - ▶ ellipsoid WGS 84.
  - ▶ Geocenter is defined as center of mass (CM) of the whole Earth.
2. Local Datum : **NTF** (Nouvelle Triangulation de la France) used until 2000
  - ▶ ellipsoid **Clarke 1880**.
  - ▶ Fondamental Point : Croix du Panthéon, Paris.
  - ▶  $Dx = -168m$ ,  $Dy = -60m$ ,  $Dz = 320m$ ,  
 $Rx = 0m$ ,  $Ry = 0m$ ,  $Rz = 0m$

# Today in France

1. Datum **NTF** is not official anymore, but still in use in some IGN maps.
2. **RGF 93** is the official French Datum compatible with European datum.
  - ▶ ellipsoid IAG GRS80.
  - ▶ Geocenter is defined as center of mass (CM) of the whole Earth.
3. **WGS 84** is used in many applications (GPS, google maps, etc.).

⇒ necessary to know how to convert data from one datum to another.



## Introduction

## Cartography

Choose the shape of earth

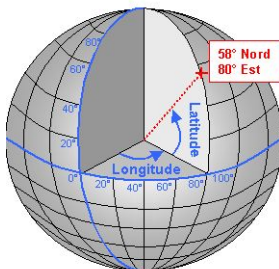
Geographic coordinate system

Projected coordinate system

## Spatial Data with R

## Mapping

# Geographic coordinate system



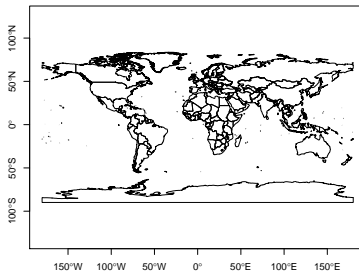
- ▶ Locations are defined in terms of the position on a globe using longitude  $\lambda$  and latitude  $\phi$  values.
- ▶ Point of reference for Latitude is the Equator. From Equator to North Pole :  $+90^\circ$ . From Equator to South Pole :  $-90^\circ$ .
- ▶ The Prime or Greenwich meridian is the reference point for lines of Longitude. The degrees of longitude run  $180^\circ$  East (positive values) and  $180^\circ$  West (negative values) from the prime meridian.

# Some remarks

- ▶ Longitude/Latitude can be given in degrees/minutes/second or decimal degrees (almost used in GIS). Example of conversion :  $10^{\circ}50'59'' = 10.849722^{\circ}$ .
- ▶ Geographic coordinate system depends on the datum. For example, for one location, we get ( $7^{\circ}44'16''$  E,  $48^{\circ}36'00''$  N) in Datum **NTF** and ( $7^{\circ}44'12''$  E,  $48^{\circ}35'59''$  N) in Datum **WGS84**.
- ▶ As locations are defined by degrees, distances cannot be accurately measured directly.

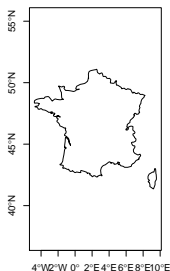
# Representation of Longitude/Latitude (global)

```
> CRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0")
```

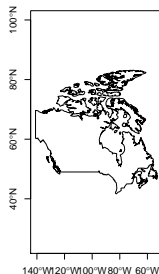


# Representation of Longitude/Latitude (local)

France is OK



Canada should be improved



## Introduction

## Cartography

Choose the shape of earth

Geographic coordinate system

Projected coordinate system

## Spatial Data with R

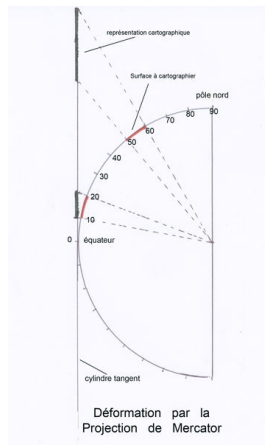
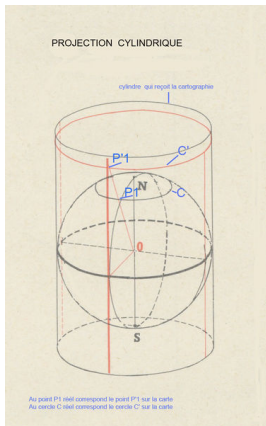
## Mapping

# Projected coordinate system

- ▶ Projected coordinate systems are based on a Datum (for example WGS84). Locations are defined using Cartesian  $x$ ,  $y$  coordinates on a flat, two-dimensional surface
- ▶ Why all world map are wrong? [Click here](#).
- ▶ Objective: minimize distortions in the shape, distance, angle and area.
- ▶ The three main families of map projections are : Cylindrical Projections, Conic Projections, Azimuthal Projections.
- ▶ Mathematically, we choose  $f$  et  $g$  depending on the projection such as  $(x = f(\lambda, \phi), y = g(\lambda, \phi))$ .



# Cylindrical Projections

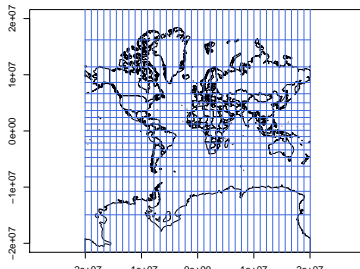




## Example : Mercator projection (global)

The Mercator projection represents angles correctly but causes distortions on distances and areas. Used in Google Maps til 2018 (and used again since 2019).

```
> CRS("+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0  
+x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +no_defs")
```

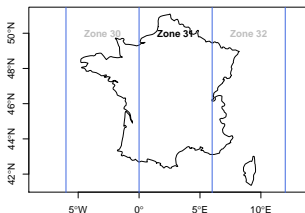




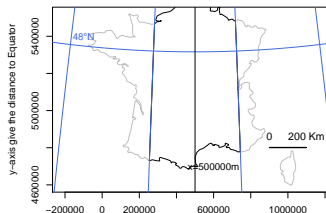
## Example : UTM 31 projection (local)

- Universal Transverse Mercator (UTM) coordinate system : not a single map projection. 60 zones, 1 projection by zone.  
 > `CRS("+proj=utm +zone=31 +datum=WGS84 +units=m +no_defs")`

Example with the zone 31



UTM 31 accurates in Zone 31 only



- Cartesian  $x$ ,  $y$  coordinate are given in some IGN maps and GPS.



# Simplified conversion formula

Wikipedia print screen ([http://en.wikipedia.org/wiki/Universal\\_Transverse\\_Mercator\\_coordinate\\_system](http://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system))

From UTM coordinates (E, N, Zone, Hemi) to latitude, longitude (φ, λ) [\[edit\]](#)

Note: Hemi=+1 for Northern, Hemi=-1 for Southern

First let's compute some intermediate values:

$$\xi = \frac{N - N_0}{k_0 A}, \quad \eta = \frac{E - E_0}{k_0 A},$$

$$\xi' = \xi - \sum_{j=1}^3 \beta_j \sin(2j\xi) \cosh(2j\eta), \quad \eta' = \eta - \sum_{j=1}^3 \beta_j \cos(2j\xi) \sinh(2j\eta),$$

$$\sigma' = 1 - \sum_{j=1}^3 2j\beta_j \cos(2j\xi) \cosh(2j\eta), \quad \tau' = \sum_{j=1}^3 2j\beta_j \sin(2j\xi) \sinh(2j\eta),$$

$$\chi = \sin^{-1} \left( \frac{\sin \xi'}{\cosh \eta'} \right),$$

The final formulas are:

$$\varphi = \chi + \sum_{j=1}^3 \delta_j \sin(2j\chi),$$

$$\lambda_0 = \text{Zone} \times 6^\circ - 183^\circ$$

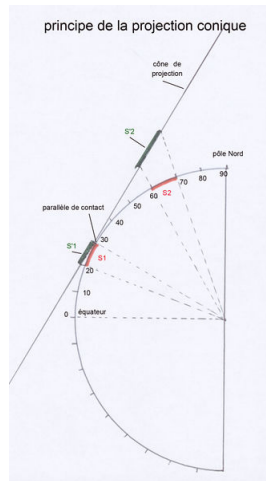
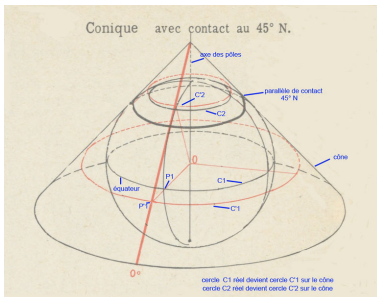
$$\lambda = \lambda_0 + \tan^{-1} \left( \frac{\sinh \eta'}{\cos \xi'} \right),$$

$$k = \frac{k_0 A}{a} \sqrt{\left\{ 1 + \left( \frac{1-n}{1+n} \tan \varphi \right)^2 \right\} \frac{\cos^2 \xi' + \sinh^2 \eta'}{\sigma'^2 + \tau'^2}},$$

$$\gamma = \text{Hemi} \times \tan^{-1} \left( \frac{\tau' + \sigma' \tan \xi' \tanh \eta'}{\sigma' - \tau' \tan \xi' \tanh \eta'} \right).$$



# Conic Projections



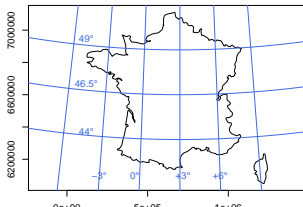
- ▶ This Conic projection preserves areas.
- ▶ Shape along the standard parallels is accurate.



## Example : RGF 93, Lambert 93 (local)

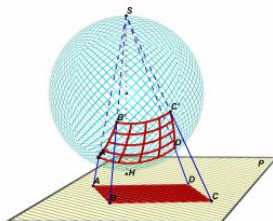
- ▶ LCC (Lambert Conformal Conic) projection. Preserves angles locally.
- ▶ Official projection in France.
 

```
> 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")
```
- ▶ References parallels : 44°N and 49°N. Reference meridian: 3°E



# Azimuthal Projections

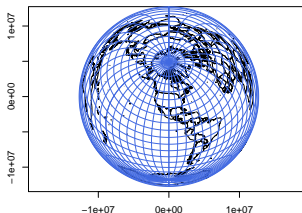
- ▶ Azimuthal projections are used often for mapping Polar Regions. Preserves direction property from the center point of the projection.
- ▶ The flag of the United Nations contains an example of a polar azimuthal equidistant projection.



## Example : US National Atlas Equal Area (global)

### ► Lambert azimuthal equal-area projection

```
> CRS("+proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0  
+a=6370997 +b=6370997 +units=m +no_defs")
```



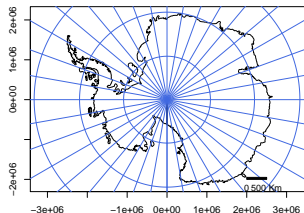
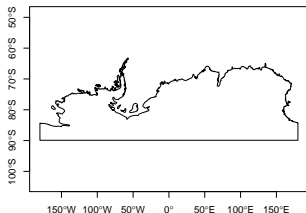
### ► Looks like Google earth.



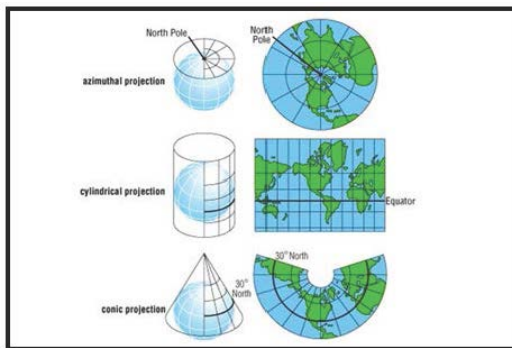
# Example : Antarctic Polar Stereographic (local)

```
> CRS("+proj=stere +lat_0=-90 +lat_ts=-71 +lon_0=0 +k=1
+x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs")
```

WGS84 Long/Lat



# Summary



# General information

- ▶ Coordinate Reference Systems (CRS) = Datum + Projection.
- ▶ Natural Earth : <http://www.naturalearthdata.com/>
- ▶ GADM database of Global Administrative Areas:  
<http://www.gadm.org/>
- ▶ French administrative boundaries and other data (roads, rivers, altitude, etc.):  
<https://geoservices.ign.fr/documentation/diffusion/telechargement-donnees-libres.html>
- ▶ The list of the CRS used in the World:  
<http://www.spatialreference.org/>
- ▶ How to find a CRS with R:
  - > EPSG <- make\_EPSG()
  - > EPSG[grep("France", EPSG[, "note"]),]

## General information (2)

CRS in France : [click here](#)

```
> EPSG[which(EPSG[, "code"] == "2154"),]
```

```
      code      note
1353 2154 RGF93 / Lambert-93
```

```
1353 +proj=lcc +lat_0=46.5 +lon_0=3 +lat_1=49 +lat_2=44 +x_0=700000 +y_0=6300000
      prj_method
```

```
1353 Lambert Conic Conformal (2SP)
```

```
> EPSG[grep("27572", EPSG[, "code"]),]
```

```
      code      note
2470 27572 NTF (Paris) / Lambert zone II
```

```
2470 +proj=lcc +lat_1=46.8 +lat_0=46.8 +lon_0=0 +k_0=0.99987742 +x_0=600000 +y_0=6300000
      prj_method
```

```
2470 Lambert Conic Conformal (1SP)
```

## General information (3)

To get administrative boundaries directly in R, one can use the function `getData()` from package `raster` :

```
> require("raster")
> france_metro <- getData("GADM", country = "france", level = 0)
```

Most countries in the world are available. The option `level =` (usually between 0 and 4) corresponds to different administrative levels (0 for the country, 1 for regions, 2 for départements, etc.). Other information can be obtained such that the temperature or the altitude (see <http://diva-gis.org/gdata>). For example :

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

## General information (4)

How to get the longitude/latitude coordinates with an adress ?

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

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

Introduction

Cartography

Spatial Data with R

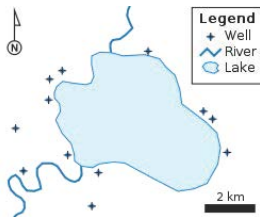
Importing GIS file formats

Manipulating Spatial objects

Manipulating raster objects

Mapping

# GIS file formats



## 1. **raster** data type

- ▶ Any type of digital image represented by reducible and enlargeable grids.
- ▶ Formats: GeoTIFF, etc.

## 2. A **vector** data type

- ▶ Different geographical features expressed by different types of geometry: points, lines or polygons.
- ▶ Formats: Shapefile, MapInfo TAB format, GeoJSON, GeoPackage, etc.



# Example 1 : Shapefile format

Popular geospatial vector data format for GIS software (compatible with MapInfo, ArcGIS, QGIS, etc). It includes at least 3 files:

- ▶ `.shp` (shape format): the feature geometry itself.
- ▶ `.dbf` (attribute format): columnar attributes for each shape.
- ▶ `.shx` (shape index format): a positional index of the feature geometry to allow moving forwards and backwards quickly.

Other files are also possible such as `.prj` (projection format): the coordinate system and projection information, a plain text file describing the projection using well-known text format.

## Example 2 : GeoJSON format

```
{
  "type": "FeatureCollection",
  "crs": {
    "type": "name",
    "properties": {
      "name": "EPSG:26904"
    }
  },
  "features": [
    {
      "type": "Feature",
      "properties": {
        "name": "Van Dorn Street",
        "marker-color": "#0000ff",
        "marker-symbol": "rail-metro",
        "line": "blue"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [
          -77.12911152370515,
          38.79930767201779
        ]
      }
    }
  ]
}
```

# Importing a Shapefile with R

- Option 1: Use rgdal package (sp norm). CRS is automatically imported. readOGR() can also be used for many **vector** data type.

```
> require("rgdal")
```

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

```
OGR data source with driver: ESRI Shapefile
```

```
Source: "/media/thibault/My Passport/2020_mars_backup/M2/2020-2021/  
with 251 features
```

```
It has 1 fields
```

```
> proj4string(world)
```

```
[1] "+proj=longlat +datum=WGS84 +no_defs"
```

## Importing a Shapefile with R (2)

- Option 2: Use sf (Simple Features) package (see Pebesma 2017).

```
> require("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,
```

# Importing a **raster** data type

- ▶ Option 1: Use raster package (see vignette("Raster"))
  - > `require("raster")`
  - > `wind <- raster("Donnees/wind/FRA_wind-speed_100m.tif")`
- ▶ Option 2: Use rgdal package
  - > `wind <- readGDAL("Donnees/wind/FRA_wind-speed_100m.tif")`

**Remark:** rgdal and maptools use Spatial class object. raster uses a different class of object. Some links exist between the 2 classes. raster is more powerful for image treatment functions.

# Importing non spatial data type

We import data corresponding to the strongest earthquake which occurred the last 40 years:

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

Columns Longitude and Latitude are the Geographic coordinates.

We create a Spatial object like this:

```
> seisme <- seisme_df
> coordinates(seisme) <- ~Longitude + Latitude
> class(seisme)
```

Finally, we give the appropriate CRS to the data :

```
> proj4string(seisme) <- CRS("+proj=longlat +datum=WGS84 +no_defs
+ellps=WGS84 +towgs84=0,0,0")
```

With sf norm:

```
> seisme_sf <- st_as_sf(seisme_df, coords = c("Longitude", "Latitude"),
crs = 4326)
```



# The sp class

4 different subclasses of Spatial objects (see `vignette("intro_sp")`):

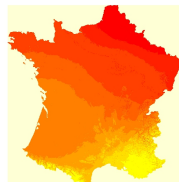
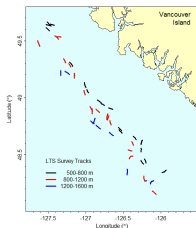
- ▶ **SpatialPolygons**: spatial unit is represented by a polygon (country, region, city, etc).
- ▶ **SpatialPoints**: spatial unit is represented by a point (earthquake, etc).
- ▶ **SpatialLines**: spatial unit is represented by a line (fault, “failles sismiques”).
- ▶ **SpatialGrids**: spatial unit is represented by a cell (Satellite imagery). A grid must be defined by using a reference point, the width of the cell and the number of row/column.

**NB**: when a spatial unit is associated to a `data.frame`, we get `SpatialXXXDataFrame` object.





# Representation of the 4 subclasses



## Manipulating Spatial objects

# Basics with sp class

- ▶ Class object and spatial/statistical summaries:  
`> str(world)`
- ▶ Dimension:  
`> dim(world)`
- ▶ Manipulate the data.frame:  
`> head(world@data)`
- ▶ Changing the ID of the spatial units and data.frame:  
`> row.names(world) <- as.character(world@data$NOM)`
- ▶ Plotting:  
`> plot(world, axes = TRUE) # Points/Polygons/Lines`  
`> image(sunfr) # Grids/Pixels`



# Selection of spatial unit

We want to select all the Maghreb countries.

- ▶ Option 1: we use a vector of boolean with the same size of the initial data

```
> maghreb <- world[world@data$NOM%in%c("Algeria", "Morocco",  
  "Libya", "Mauritania", "Tunisia", "Western Sahara"),]
```

- ▶ Option 2: we use directly the spatial unit IDs

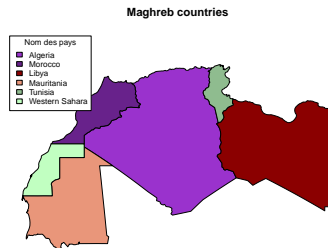
```
> maghreb <- world[c("Algeria", "Morocco", "Libya",  
  "Mauritania", "Tunisia", "Western Sahara"),]
```

- ▶ With sf object, it works like a data.frame/tbl object :

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

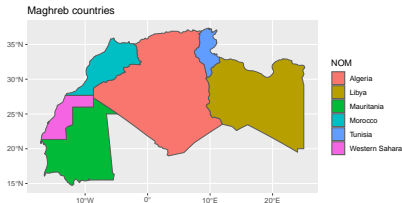
# Plotting of the Maghreb region

```
> plot(maghreb, col = colors()[98:103])
> title("Maghreb countries")
> legend("topleft", legend = row.names(maghreb), cex = 0.8,
  title = "Nom des pays", fill = colors()[98:103])
```



# Plotting of the Maghreb region (with sf object)

```
> ggplot(data = maghreb_sf, aes(fill = NOM)) +  
  geom_sf() +  
  ggtitle("Maghreb countries")
```



# Aggregating spatial units (with sp object)

We want to aggregate Morocco and Western Sahara.

1. Create the new spatial contours:

```
> require("maptools")
> maghreb2 <- unionSpatialPolygons(maghreb, c("Algeria",
  "Morocco", "Libya", "Mauritania", "Tunisia", "Morocco"))
```

2. Create (or aggregate) the new data attributes (Source:

<https://www.cia.gov/library/publications/the-world-factbook/>, 2012):

```
> maghreb2.df <- data.frame(ID = c("Algeria", "Morocco", "Libya",
  "Mauritania", "Tunisia"), pib = c(7300, 5200, 11900, 2100, 9700),
  row.names = "ID")
```

3. Associate the spatial units to the attributes (IDs of Spatial contours and Data attributes must be the same !):

```
> maghreb2.spdf <- SpatialPolygonsDataFrame(maghreb2, maghreb2.df)
```

# Aggregating spatial units (with sf object)

With sf package:

1. create the new ID variable:

```
> maghreb_sf$NOM_2 <- c("Algeria",  
  "Morocco", "Libya", "Mauritania", "Tunisia", "Morocco")
```

2. Use the dplyr syntax:

```
> library(tidyverse)  
> maghreb2_sf <- maghreb_sf %>%  
  group_by(NOM = NOM_2) %>%  
  summarise()
```

3. Add a new variable :

```
> maghreb2_sf$pib <- c(7300, 5200, 11900, 2100, 9700)
```



# Adding a spatial unit (with sp object)

We want to add the country Egypt to the previous data.

1. Data attributes must be the same:

```
> egypt <- world[world@data$NOM%in%c("Egypt"),]
> egypt@data <- data.frame(pib = 6500, row.names = "Egypt")
```

2. We can create the new object:

```
> northAf <- spRbind(maghreb2.spdf, egypt)
```

# Adding a spatial unit (with sf norm)

1. Data attributes must be the same:

```
> egypt_sf <- world_sf["Egypt", ]
> egypt_sf$pib <- 6500
```

2. We can create the new object:

```
> northAf_sf <- rbind(maghreb2_sf, egypt_sf)
> row.names(northAf_sf) <- northAf_sf$NOM
```



## Plotting two `spatial` objects (`sp` norm)

We want to represent the earthquakes and the contours of the countries. For that, CRS must be the same.

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

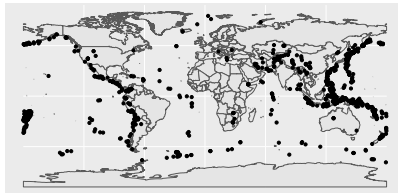
Strongest earthquake in 40 years



# Plotting two spatial objects (sf norm)

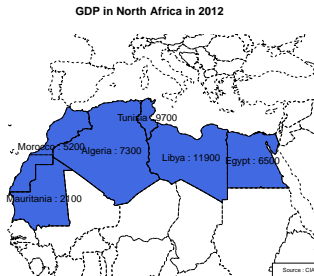
```
> ggplot(data = world_sf) +  
  geom_sf() +  
  geom_sf(data = seisme_sf[1:1000,]) +  
  ggtitle("Strongest earthquake in 40 years")
```

Strongest earthquake in 40 years



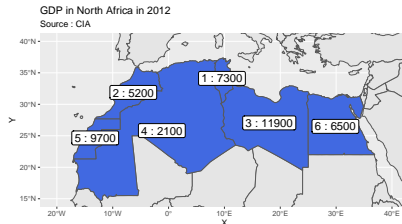
# Plotting the GDP

```
> plot(northAf, col = "royalblue", xlim = c(-20, 40), ylim = c(15, 40))
> plot(world, add = T, lty = 2)
> title("GDP in North Africa in 2012")
> text(coordinates(northAf),
  paste(row.names(northAf), northAf$pib, sep = " : "))
> legend("bottomright", "Source : CIA", cex = 0.6)
```



# Plotting the GDP (sf norm)

```
northAf_sf[ , c("X", "Y")] <- st_coordinates(st_centroid(northAf_sf))
ggplot(data = world_sf) + geom_sf() +
  geom_sf(data = northAf_sf, fill = "royalblue") +
  geom_label(data = northAf_sf, aes(X, Y, label = paste(row.names(northAf_sf),
    northAf_sf$pib, sep = " : ")), size = 5) +
  coord_sf(xlim = c(-20, 40), ylim = c(15, 40)) +
  ggtitle("GDP in North Africa in 2012", subtitle = "Source : CIA")
```



# CRS conversion

We would like to get the boundary of France converted in the official RGF 93 - Lambert 93 CRS.

```
> france <- world[world@data$NOM%in%c("France"),]
```

1. We check if the actual CRS is identified:

```
> proj4string(france)
```

2. We define the new CRS:

```
> epsg2154 <- CRS("+proj=lcc +lat_1=49 +lat_2=44 +lat_0=46.5  
+lon_0=3 +x_0=700000 +y_0=6600000 +ellps=GRS80  
+towgs84=0,0,0,0,0,0,0 +units=m +no_defs")
```

3. We convert the data:

```
> france2 <- spTransform(france, epsg2154)
```

# CRS conversion (sf norm)

We would like to get the boundary of France converted in the official RGF 93 - Lambert 93 CRS.

```
> france_sf <- world_sf["France", ]
```

1. We check if the actual CRS is identified:

```
> st_crs(france_sf)
```

2. We convert the data:

```
> france2_sf <- st_transform(france_sf, 2154)
```



# Spatial overlay

We want to select the earthquakes which occurred around Japan.

1. We create a Spatial object for Japan:

```
> japan <- world[world@data$NOM%in%c("Japan"),]
```

2. We create a Spatial object by using bounding box:

```
> bb.jp <- as(extent(bbox(japan)), "SpatialPolygons")
```

```
> proj4string(bb.jp) <- proj4string(japan)
```

3. We use the function `over` which creates a vector or integer with 1 if located in the bbox of Japan and NA otherwise:

```
> ind <- over(seisme, bb.jp)
```

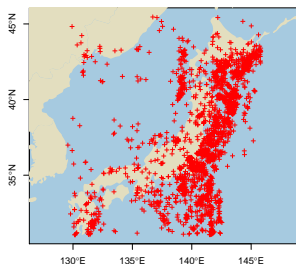
4. We create the new Spatial object :

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

# Spatial overlay (2)

- Representation of the new data :

```
> plot(japan, border = NA, col = "NA", bg="#A6CAE0", axes = T)
> plot(world, col = "#E3DEBF", border = NA, add = T)
> plot(seisme.jp, add = TRUE, col = "red", cex = 0.7)
```



# Spatial overlay (sf norm)

1. We create a sf object for Japan:  

```
> japan_sf <- world_sf %>% filter(NOM == "Japan")
```
2. We create a sf object by using bounding box:  

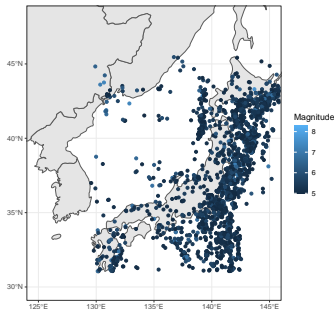
```
> bb_jp_sf <- st_as_sfc(st_bbox(japan_sf))
```
3. We create the new Spatial object :  

```
> seisme.jp_sf <- st_intersection(seisme_sf, bb_jp_sf)
```

# Spatial overlay (2)

- Representation of the new data :

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



# rgeos package : union, intersection, buffer, etc.

We want to select only the earthquakes which occurred inside Japan and at less than 50km of the coast.

1. We convert Japan and earthquake data in a metric projection:

```
> crsjap <- "+proj=tmerc +lat_0=36
+lon_0=139.83333333333333 +k=0.9999 +x_0=0 +y_0=0 +ellps=bessel
+towgs84=-146.414,507.337,680.507,0,0,0,0 +units=m +no_defs"
> seisme.jp2 <- spTransform(seisme.jp, crsjap)
> japan2 <- spTransform(japan, crsjap)
```

2. We create a buffer of 50km:

```
> japan2buff <- gBuffer(japan2, width = 50000)
```

3. We create the new Spatial object :

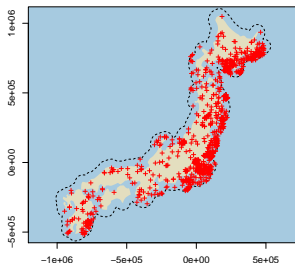
```
> seisme.jp3 <- gIntersection(seisme.jp2, japan2buff)
```



## rgeos package : union, intersection, buffer, etc. (2)

### ► Representation of the new data :

```
> plot(japan2, border = NA, col = "NA", bg="#A6CAE0", axes = T)
> plot(japan2, col = "#E3DEBF", border = NA, add = T)
> plot(japan2buff, add = TRUE, lty = 2)
> plot(seisme.jp3, add=T, col = "red", cex = 0.7)
```



# Union, intersection, buffer, etc. (with sf norm)

1. We convert Japan and earthquake data in a metric projection:

```
> seisme.jp2_sf <- st_transform(seisme.jp_sf, crsjap)
> jp2_sf <- st_transform(japan_sf, crsjap)
```

2. We create a buffer of 50km:

```
> jp2buf_sf <- st_buffer(jp2_sf, dist = 50000)
```

3. We create the new Spatial object :

```
> seisme.jp3_sf <- st_intersection(seisme.jp2_sf, jp2buf_sf)
```

# Union, intersection, buffer, etc. (with sf norm) (2)

```
> ggplot(data = world_sf) +
  geom_sf(bg = "#A6CAE0", col = "#E3DEBF") +
  geom_sf(data = jp2buf_sf, lty=2, col = scales::alpha("green", 0.1))
  geom_sf(data = jp2_sf, fill = "#A6CAE0") +
  geom_sf(data = seisme.jp3_sf, col = "red", cex = 0.7) +
  xlim(125, 145) + ylim(30, 48) + theme_bw()
```





## Introduction

## Cartography

## Spatial Data with R

Importing GIS file formats

Manipulating Spatial objects

Manipulating raster objects

## Mapping

# Basics with `raster` object

- ▶ Import the yearly sum of global irradiation, 10-years average of the period 1981-1990 [kWh/m<sup>2</sup>], in WGS 84 datum (see <http://re.jrc.ec.europa.eu/pvgis/>) :

```
> suneu <- raster("Donnees/soleil/pvgis_g13year00.asc",
  crs = CRS("+proj=longlat +ellps=WGS84"), native = TRUE)
```

- ▶ The dimension (rows × columns × variables) :

```
> dim(suneu)
[1] 474 864 1
```

- ▶ Can be use as an array object :

```
> suneu[150, 685, 1]
945
```

- ▶ Plotting :

```
> image(suneu)
```

# Extracting pixels from a `raster`

We would like to extract only the pixels included in the polygon of France.

- ▶ We define the bounding box of France (in WGS 84) :  
`> fr.bb <- bbox(france)`
- ▶ We select the pixels included in that BBox :  
`> suneu.bb <- crop(suneu, extent(fr.bb))`
- ▶ We extract the pixels included in the polygons  
`> suneu.fr <- rasterize(france, suneu.bb, mask = TRUE)`
- ▶ Plotting :  
`> plot(france)`  
`> image(suneu.fr, add = TRUE)`

# Image resolution

We would like to decrease the image resolution. Actually, the number of cells is :

```
> dim(suneu.fr)
[1] 117 172 1
```

We aggregate the cells by a factor 15 horizontally and 10 vertically. The new values are computed by using the mean :

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

We mask the part of cells outside the polygon of France :

```
> require("GISTools")
> plot(france)
> image(suneu.fr.agg, add = TRUE)
> masker = poly.outer(suneu.fr.agg, france); add.masking(masker)
```

# Importing a OSM image

We would like to import the OpenStreetMap image covering TSE, UT1 and metro Compans.

1. We identify the Geographic coordinates of that locations :

```
> long <- c(1.432022, 1.436879, 1.435409)
> lat <- c(43.605417, 43.606688, 43.610372)
> xy <- SpatialPoints(cbind(long, lat),
                      CRS("+proj=longlat +ellps=WGS84"))
```

2. We import the map from OSM :

```
> require("cartography")
> mtqOSM <- getTiles(x = xy, type = "osm", zoom = 13)
```

3. map\_osm is a raster object. We get the CRS like this :

```
> projMer <- st_crs(mtqOSM)
```

which is the WGS 84 projection.

# Importing a OSM image (2)

4 We plot the results :

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



# Interactive maps with other packages

- ▶ Importing a googlemap image: since June, 2018, users are now required to provide an API key and enable billing (see <https://developers.google.com/maps/documentation/geocoding>).
- ▶ Interactive maps with mapview package :
 

```
> require("mapview")
> mapview(seisme.jp[, "Magnitude"],
          col.regions = sf.colors(10))
```

# Map and find the distance of routes with OSM

```
> library(osrm)
> short_path <- osrmRoute(src = xy[1, ],
                           dst = xy[2, ],
                           returnclass = "sp", overview = "full")
> tilesLayer(mtgOSM)
> title("Short path")
> plot(xy, col = 2, cex = 2, add = TRUE)
> plot(short_path, add = T, col = 2, lwd = 2)
> text(coordinates(xy_merc), col = 2, pos = 3,
        labels = c("Manu", "Arsenal", "Metro"))
```





Introduction

Cartography

Spatial Data with R

Mapping

Representing a categorical variable

Representing a numeric variable

Exercise

# Preparation of the data (1)

First, we select the countries of Sub-Saharan Africa :

```
> pays.af<-c("Seychelles","Equatorial Guinea","Gabon","Botswana",
  "Mauritius","South Africa","Namibia","Angola","Swaziland","Congo",
  "Cape Verde","Ghana","Sudan","Djibouti","Nigeria","Sao Tome and Princi
  "Cameroon","Lesotho","Gambia, The","Chad","Senegal","Kenya","Ivory Coa
  "Zambia","Burkina Faso","Tanzania, United Republic of","Benin","Rwanda
  "Uganda","Comoros","Guinea-Bissau","Mali","Mozambique","Guinea","Ethio
  "Madagascar", "Malawi","Togo","Sierra Leone","Niger","Central African
  "Eritrea","Burundi","Somalia","Zimbabwe","Liberia","Zaire")
> africa.sub <- world[world@data$NOM%in%pays.af,]
```

Then we create a data.frame of the GDP :

```
> africa.df<-data.frame(pib=c(25000,26500,18100,15700,15400,11300,7800,
  5700,4600,4400,3300,2500,2600,2700,2100,2300,2100,1900,2500,2000,1800,
  1700,1400,1600,1600,1400,1400,1300,1200,1100,1200,1100,1300,900,800,11
  1300,800,900,700,600,600,600,700,400),row.names=pays.af)
```

## Preparation of the data (2)

We create the `SpatialPolygonsDataFrame` object :

```
> africa.sub <- SpatialPolygonsDataFrame(africa.sub, africa.df)
```

We merge Maghreb and Sub-Saharan Africa :

```
> africa <- spRbind(northAf, africa.sub)
```

We add a variable `region` to the country (E, East Af, C, Central Af, N, North Af, Au, Austral Af and O, West Af) :

```
> africa@data$region <- factor(c("N","N","N","N","N","N","O",  
  "O","C","E","E","O","O","O","E","O","O","C","E","O","O",  
  "O","E","O","O","C","O","O","C","O","E","E","C","C","C",  
  "C","E","E","E","E","C","E","E","E","E","E","E","Au","Au",  
  "E","Au","Au","Au"))
```

# Representing a categorical variable

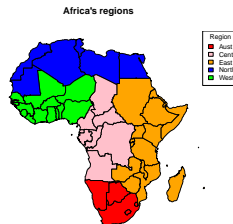
Objective: reproduce this map [http://fr.wikipedia.org/wiki/Afrique#mediaviewer/File:Zones\\_Afrique.jpg](http://fr.wikipedia.org/wiki/Afrique#mediaviewer/File:Zones_Afrique.jpg)

Choice of appropriate colors:

```
> pal.reg <- c("red","pink","orange",  
  "blue","green")
```

Plotting the map :

```
> ind=as.numeric(africa@data$region)  
> plot(africa, col=pal.reg[ind])  
> title("Africa's regions")  
> legend("topright",legend=c("Aust","Cent"  
  "East","North","West"),cex=0.8,  
  title="Region",fill=pal.reg)
```





Introduction

Cartography

Spatial Data with R

**Mapping**

Representing a categorical variable

**Representing a numeric variable**

Exercise

# Representing a numeric variable

How to cut the GDP variable ? R package classInt.

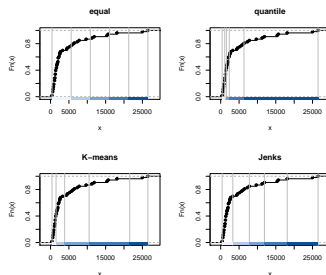
```
> require("classInt")
> pib <- africa@data$pib
```

- ▶ The `equal` style divides the range of the variable into  $n$  parts.
- ▶ The `quantile` style provides quantile breaks; arguments to quantile may be passed through ....
- ▶ The `kmeans` style uses `kmeans` to generate the breaks.
- ▶ The `jenks` style has been imported from Jenks' Basic code, and has been checked for consistency with ArcView, ArcGIS, and MapInfo.

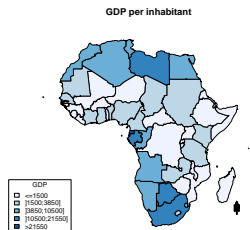


# Application

```
> require("RColorBrewer")
> plotclr <- brewer.pal(5,"Blues")
> pal1 <- plotclr[c(1,5)]
> opar <- par(mfrow=c(2,2))
> 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)
```



```
> ind=findInterval(pib, bk, all.inside=TRUE)
> plot(africa, col=plotclr[ind])
> decoup <- c("<=1500","]1500;3850]",
  "3850;10500]","]10500;21550]", ">21550")
> legend("bottomleft", legend = decoup,
  cex = 0.8,title="GDP", fill=plotclr)
> SpatialPolygonsRescale(layout.north.arrow,
  offset = c(50,-30), scale = 5, plot.grid=FALSE)
> title("GDP per inhabitant")
```

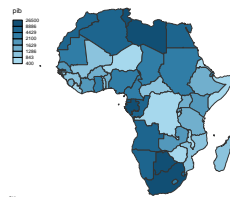




# Package cartography

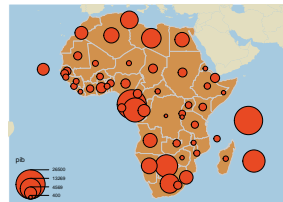
- ▶ package cartography by T. Giraud (2015). See tutorial: [http://wukan.ums-riate.fr/RUSS/RUSS\\_2016/](http://wukan.ums-riate.fr/RUSS/RUSS_2016/).

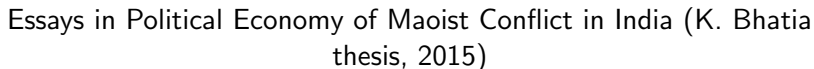
```
> library("cartography")
> choroLayer(spdf = africa, var = "pib",
  legend.pos = "topleft")
> layoutLayer(title = "GDP in Africa",
  sources = "CIA",
  frame = TRUE, col = "NA",
  scale = NULL)
```



# Bubble plot of spatial data

```
> plot(africa, border = NA,
      col = "NA", bg="#A6CAE0")
> plot(world, col = "#E3DEBF",
      border = NA, add = T)
> plot(africa, col = "#D1914D",
      border = "grey80", add = T)
> require("lwgeom")
> propSymbolsLayer(spdf = africa,
      var="pib" )
```



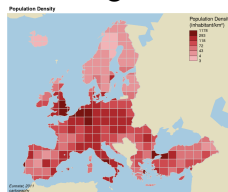
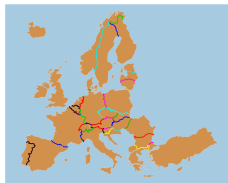




# Other kind of maps with package cartography

Flows, grid, etc. See

<http://rgeomatic.hypotheses.org/author/rgeomatic>.



Introduction

Cartography

Spatial Data with R

Mapping

Representing a categorical variable

Representing a numeric variable

Exercise

# Velo Toulouse data (1)

- ▶ Question: Do the inhabitants of Toulouse have access to the same offer of “velo Toulouse” whatever the place they live?
- ▶ First approach (before modelling): using spatial exploratory data analysis (SEDA).
- ▶ Objective “Visualising spatial distributions and local patterns of spatial autocorrelation” (L. Anselin).
- ▶ Source of the data:  
<https://data.toulouse-metropole.fr/page/home/>

## Velo Toulouse data (2)

1. Import the Spatial data ("iris" and "velo-toulouse") included in the folder "Donnees".

2. (If necessary), convert the data into the same CRS.

OGR data source with driver: ESRI Shapefile

Source: "/media/thibault/My Passport/2020\_mars\_backup/M2/2020-2021/  
with 282 features

It has 11 fields

OGR data source with driver: ESRI Shapefile

Source: "/media/thibault/My Passport/2020\_mars\_backup/M2/2020-2021/  
with 60 features

It has 7 fields

3. Plot on the same map the informations related to the administrative boundaries and the locations of the bike station.

## Velo Toulouse data (3)

1. In `bike` object, the variable `nb_bornette` corresponds to the number of “bornettes”. If it is equal to 0, that means that the station was not opened yet. In that case, replace the value 0 by the value of your choice.
2. In `bike` object, create the variable `grd_quart`, which corresponds to the Id of the iris where the bike station bike is located (see function `gIntersects` in package `rgeos` or `st_intersection` in package `sf`).
3. In `iris` object, create the variable `nb_bornette_iris` which gives the total number of bornette by iris.



# Velo Toulouse data (4)

- Compute the variable number of bornette per inhabitant and plot this variable on a map.

