Maps and geolocalized data

Lecture 14

Louis SIRUGUE

01/2022

Last time we saw

Regular expressions

- Regular expressions are strings of codified characters describing a pattern
  - For instance the character "^" indicates the start of the string
  - So the regular expression "^a" would match any "a" that is a the beginning of a string

- Regular expressions in R can be used in different functions with different purposes:
  - grep: returns elements that match the regexpr
  - grepl: returns TRUE for elements that match the regexpr and FALSE otherwise
  - gsub: replaces the elements that match the regexpr with what you want
  - ...

Regexpr	Meaning
^	Start of string (or 'not')
$	End of string
.	Any character
*	0 or more occurences
+	1 or more occurences
[^abc]	Not a, b or c
[a-z]	Any lowercase letter from a to z
[A-Z]	Any capital letter from A to Z
[0-9]	Any digit from 0 to 9

Last time we saw

Tokenization

- Tokenization is the fact of cleaning the data so that there is one unit of text per row
  - A unit of text (token) can be a character, a letter, a word, a sentence, etc.

line	direction
ACT I	FALSE
SCENE I. A public place.	FALSE
Enter Sampson and Gregory armed with swords and bucklers.	FALSE
SAMPSON.	FALSE
Gregory, on my word, we’ll not carry coals.	FALSE
GREGORY.	FALSE
No, for then we should be colliers.	FALSE

id_act	id_scene	id_line	id_char	line
1	1	1	SAMPSON	Gregory, on my word, we’ll not carry coals.
1	1	2	GREGORY	No, for then we should be colliers.
1	1	3	SAMPSON	I mean, if we be in choler, we’ll draw.

Last time we saw

Stopwords and sentiments

- The first step of a sentiment analysis is usually to get rid of *stopwords*
  - Stopwords are common words that do not carry much semantic meaning but take space and computing time

```
matrix(get_stopwords()[["word"]][1:24],ncol=3)
```

```
##       [,1]        [,2]         [,3]
## [1,] "i"         "you"        "himself"
## [2,] "me"        "your"       "she"
## [3,] "my"        "yours"      "her"
## [4,] "myself"    "yourself"   "hers"
## [5,] "we"        "yourselves" "herself"
## [6,] "our"       "he"         "it"
## [7,] "ours"      "him"        "its"
## [8,] "ourselves" "his"        "itself"
```

- The second step is to join the words to their corresponding sentiment using a dictionary
  - Some dictionaries are very simple: positive/negative
  - And some are more elaborate: trust/fear/sadness/anger/...

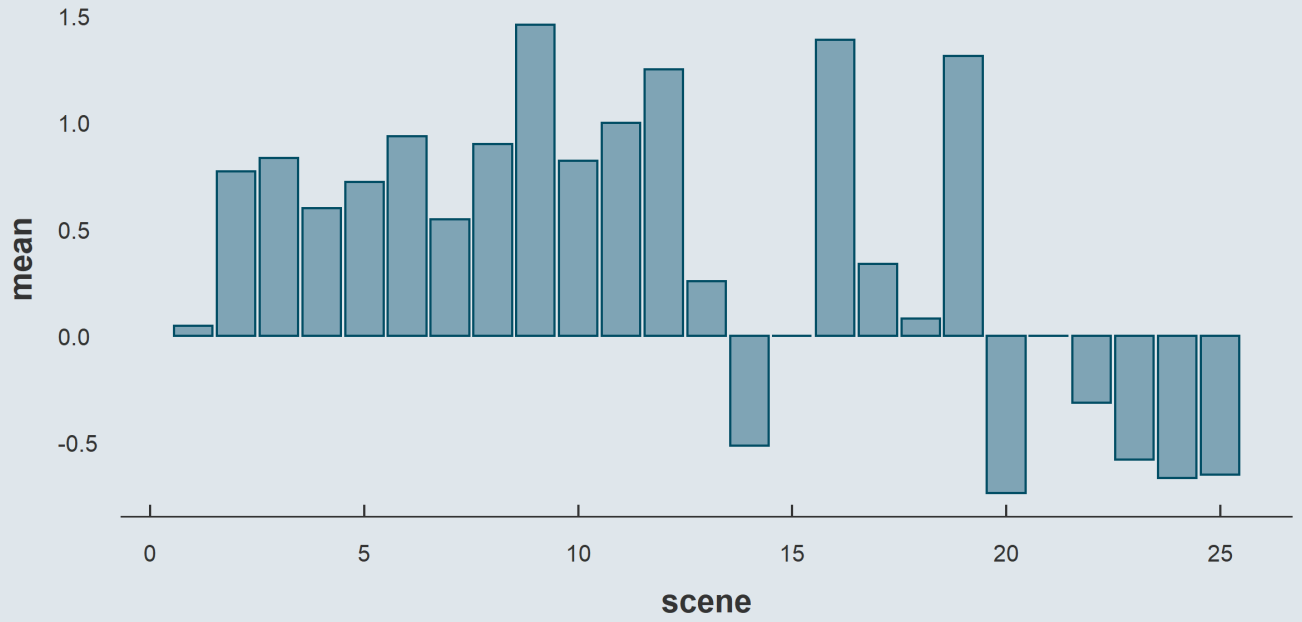
```
head(get_sentiments("bing"), 5)
```

```
## # A tibble: 5 x 2
##   word       sentiment
##   <chr>      <chr>
## 1 2-faces    negative
## 2 abnormal   negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
```

Last time we saw

Analysis

- Evolution of the average sentiment over the play



- Sentiment of characters (rows) when mentioning other characters (columns)

```
## # A tibble: 3 x 4
##   id_char  ROMEO  JULIET NURSE
##   <chr>    <dbl>   <dbl> <dbl>
## 1 JULIET  -0.255 -1.5    0.246
## 2 NURSE    0.826 -0.267  0.111
## 3 ROMEO   -0.737 -0.0746 2.4
```

Today: Maps and geolocalized data

1. Geolocalized data

- 1.1. Shapefiles and rasters
- 1.2. Opening geolocalized data
- 1.3. Coordinate Reference Systems
- 1.4. Subsetting geolocalized data

2. Geographic variables

- 2.1. Import from csv
- 2.2. Zonal statistics
- 2.3. Centroids and distance

3. Wrap up!

Today: Maps and geolocalized data

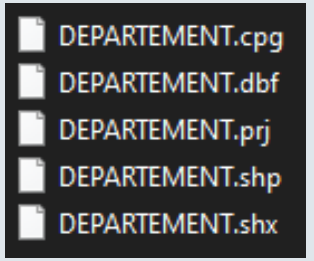
1. Geolocalized data

- 1.1. Shapefiles and rasters
- 1.2. Opening geolocalized data
- 1.3. Coordinate Reference Systems
- 1.4. Subsetting geolocalized data

1. Geolocalized data

1.1. Shapefiles and rasters

→ The **shapefile** is a format for storing **geographic location** and associated attribute information

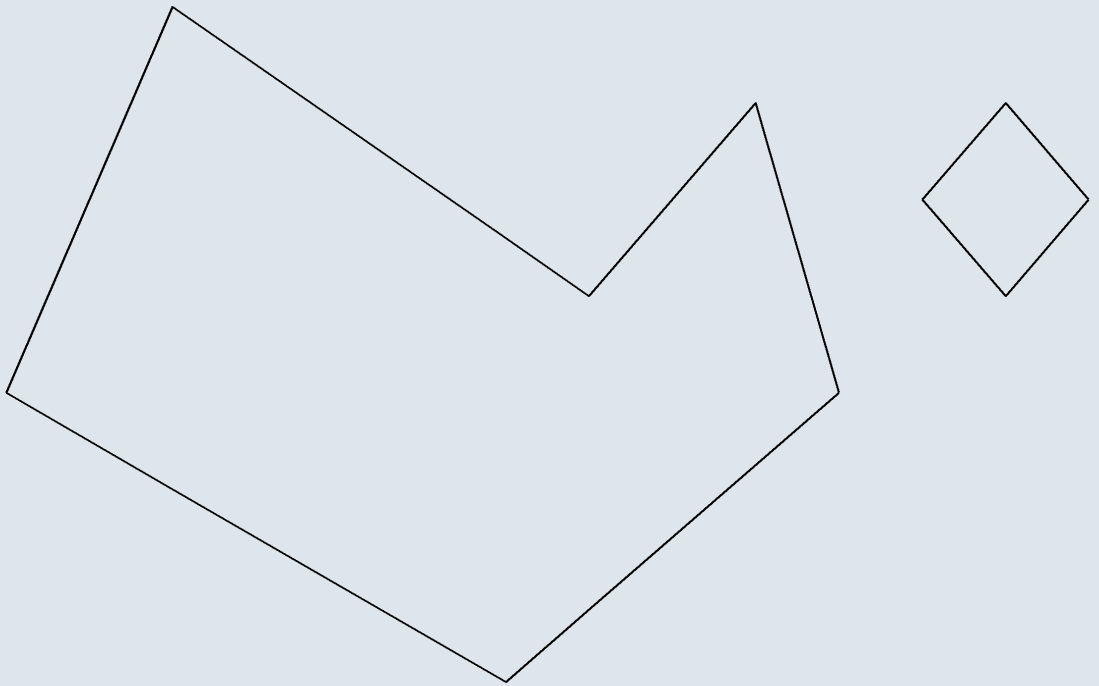


- Out-software it **can seem quite complicated:**
  - There are several files for a single dataset
  - Always the necessary .shp, .shx, and .dbf
  - Sometimes other files such as .prj, .cpg, etc.
- But in-software, it is **not so far from what we're used to:**
  - There is one line per geographic entity
  - And one column per variable about these entities
  - But there is one non-standard variable: the **geometry**
- The geometry can be:
  - **Points** at given coordinates: location of weather stations/of your phone
  - **Polylines** that link sets of points: rivers/roads
  - **Polygons** that link sets of points to form a closed shape: countries/land parcels

1. Geolocalized data

1.1. Shapefiles and rasters

- For instance consider a shapefile
  - With two observations
  - And a polygon geometry



1. Geolocalized data

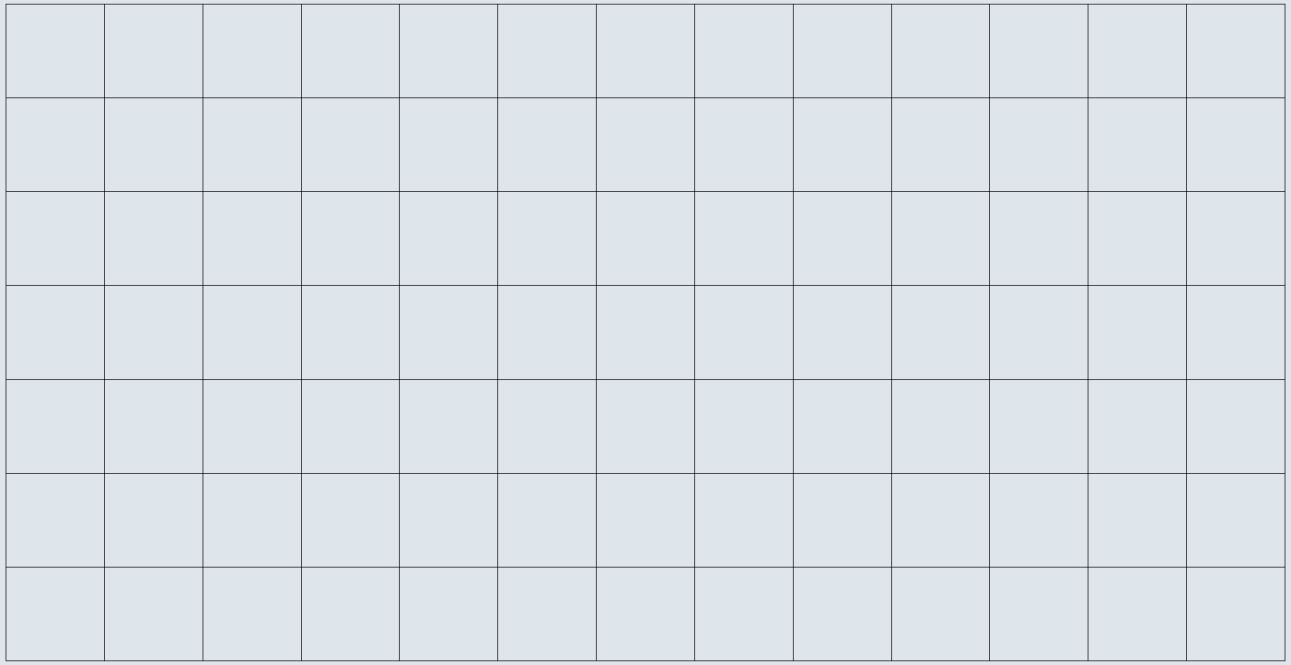
1.1. Shapefiles and rasters

- The shapefile can also contain **attribute variable**
  - For instance the **median income** of residents of the area delimited by the polygon
  - We can plot this attribute variable using the fill aesthetic of the polygon geometry

1. Geolocalized data

1.1. Shapefiles and rasters

- The **raster** is another type of format for storing geolocalized data
  - It works **like a picture**, with cells like pixels



1. Geolocalized data

1.1. Shapefiles and rasters

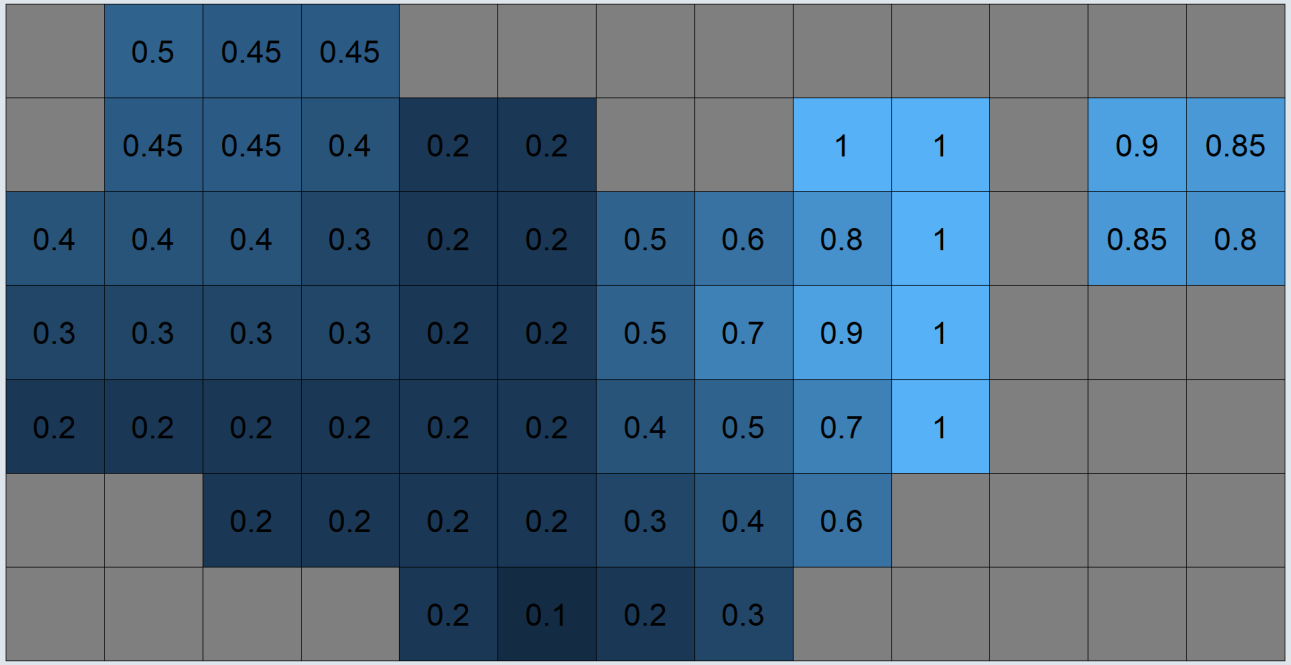
- The **raster** is another type of format for storing geolocalized data
  - It works **like a picture**, with cells like pixels
  - And each cell can take a given value, e.g. pollution observed from satellites

	0.5	0.45	0.45									
	0.45	0.45	0.4	0.2	0.2			1	1		0.9	0.85
0.4	0.4	0.4	0.3	0.2	0.2	0.5	0.6	0.8	1		0.85	0.8
0.3	0.3	0.3	0.3	0.2	0.2	0.5	0.7	0.9	1			
0.2	0.2	0.2	0.2	0.2	0.2	0.4	0.5	0.7	1			
		0.2	0.2	0.2	0.2	0.3	0.4	0.6				
				0.2	0.1	0.2	0.3					

1. Geolocalized data

1.1. Shapefiles and rasters

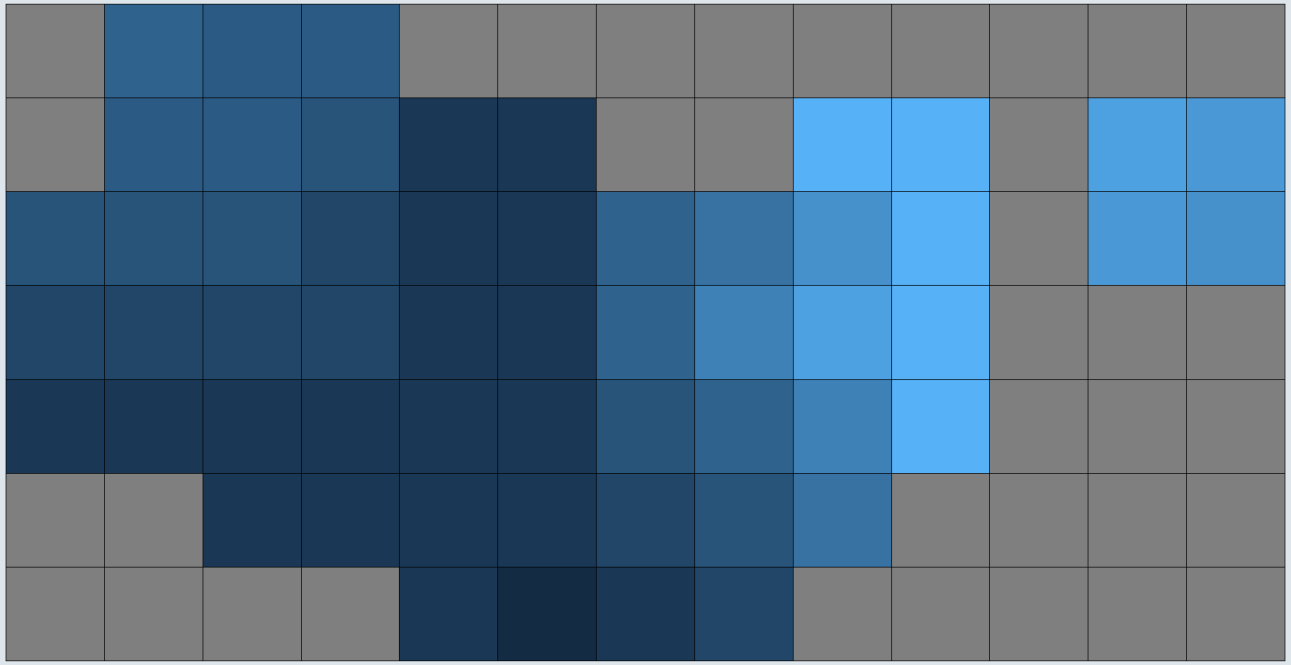
- The **raster** is another type of format for storing geolocalized data
  - It works **like a picture**, with cells like pixels
  - And each cell can take a given value, e.g. pollution observed from satellites



1. Geolocalized data

1.1. Shapefiles and rasters

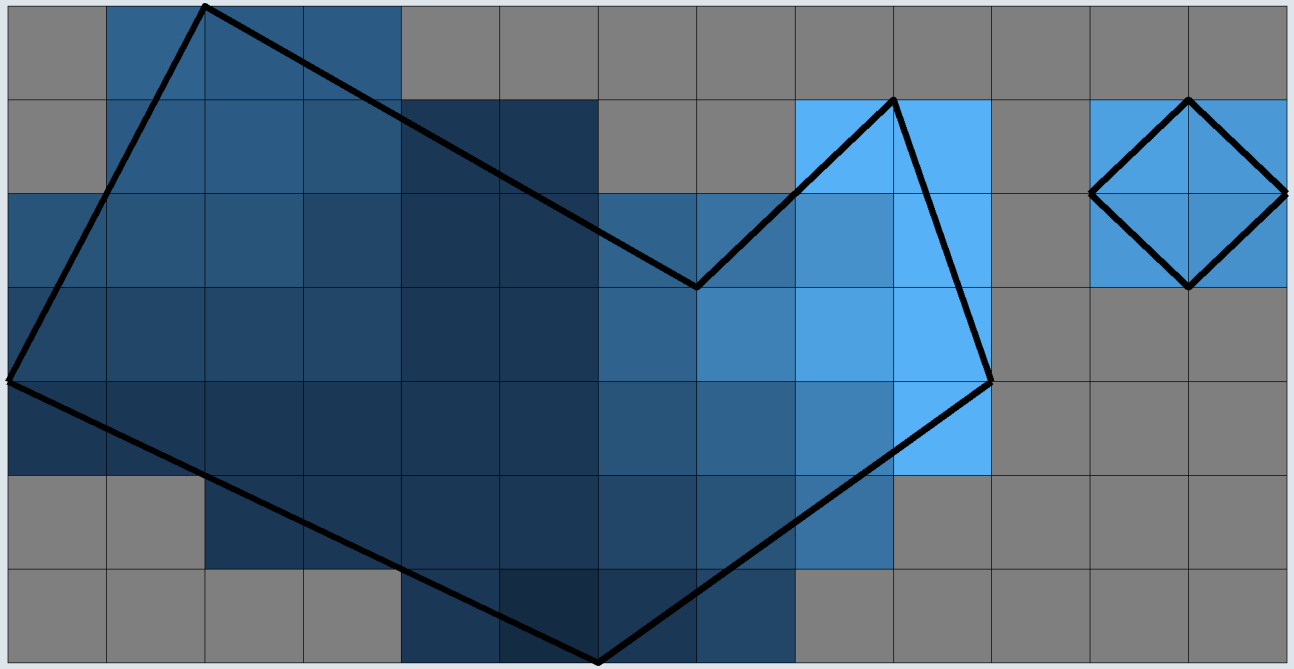
- Today we're gonna use this to estimate the relationship between **income and exposure to air pollution**
  - Using **raster** satellite data on pollution level



1. Geolocalized data

1.1. Shapefiles and rasters

- Today we're gonna use this to estimate the relationship between **income and exposure to air pollution**
  - Using **raster** satellite data on pollution level
  - And a **shapefile** to compute the average exposure in different location and merge it with income data



1. Geolocalized data

1.2. Opening geolocalized data

- We can read shapefiles in R using the `read_sf()` function from the `sf` package (data from IGN)

```
library(sf)
dep_shp <- read_sf("data/dep_shp/DEPARTEMENT.shp")
head(dep_shp, 5)
```

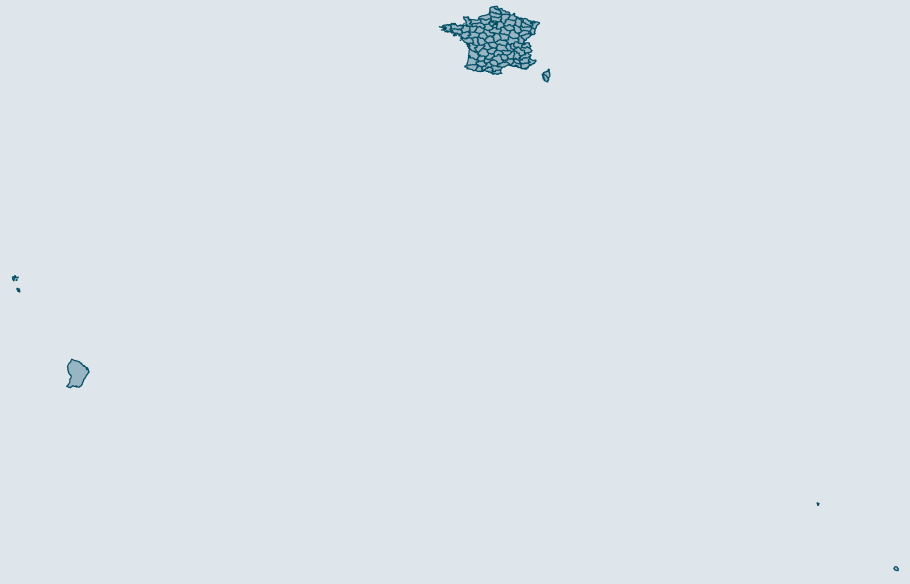
```
## Simple feature collection with 5 features and 5 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 0.06558525 ymin: 45.23103 xmax: 7.621946 ymax: 50.0722
## Geodetic CRS:  WGS 84
## # A tibble: 5 x 6
##   ID                       NOM_DEP_M      NOM_DEP INSEE_DEP INSEE_REG                  geometry
##   <chr>                    <chr>          <chr>   <chr>     <chr>            <MULTIPOLYGON [°]>
## 1 DEPARTEM0000000000000001 JURA           Jura    39        27        (((5.442842 46.84592, 5.~
## 2 DEPARTEM0000000000000002 SEINE-MARITIME Seine-~ 76        28        (((0.3875418 49.77178, 0~
## 3 DEPARTEM0000000000000003 YONNE          Yonne   89        27        (((3.126677 47.99127, 3.~
## 4 DEPARTEM0000000000000004 LOIRE          Loire   42        84        (((3.831716 45.99944, 3.~
## 5 DEPARTEM0000000000000005 HAUT-RHIN      Haut-R~ 68        44        (((6.923645 47.95226, 6.~
```

1. Geolocalized data

1.2. Opening geolocalized data

- We can then view the data using the `geom_sf()` geometry
  - It will understand that the `geometry` variable contains the coordinates of the polygons to be plotted

```
library(tidyverse)
ggplot(dep_shp) + geom_sf(fill = "#6794A7", color = "#014D64", alpha = .6) + theme_void()
```



1. Geolocalized data

1.2. Opening geolocalized data

- There are several formats of raster files (.tif, .nc, ...)
  - We're gonna use NetCDF satellite data on $PM_{2.5}$ from the Atmospheric Composition Analysis Group
  - So we're gonna need the packages `raster` and `ncdf4`

```
library(raster)
select <- dplyr::select # The raster packages has an overriding select function
library(ncdf4)

pm_data <- raster("data/acag_2016.nc")
pm_data
```

```
## class      : RasterLayer
## dimensions : 2300, 5000, 11500000  (nrow, ncol, ncell)
## resolution : 0.01, 0.009999999  (x, y)
## extent     : -15, 35, 36, 59  (xmin, xmax, ymin, ymax)
## crs        : +proj=longlat +datum=WGS84 +no_defs
## source     : acag_2016.nc
## names      : Hybrid.PM_2_._5...mug.m.3.
## zvar       : GWRPM25
```

- As you can see NetCDFs are complex objects:
  - Not a simple table with figures
  - Note that the $PM_{2.5}$ variable is Hybrid.PM_2_._5...mug.m.3.

1. Geolocalized data

1.2. Opening geolocalized data

- We can rename the $PM_{2.5}$ variable into something more convenient

```
names(pm_data) <- "pm2.5"
```

- And convert the data into a standard dataset using the `rasterToPoints()` function
  - It will generate a table with 1 row per cell
  - An x variable for the longitude, y for latitude, and pm2.5 for pollution

```
head(as_tibble(rasterToPoints(pm_data)), 5)
```

```
## # A tibble: 5 x 3
##       x     y pm2.5
##   <dbl> <dbl> <dbl>
## 1 -3.38  59.0  4.60
## 2 -3.37  59.0  4.60
## 3 -3.36  59.0  4.60
## 4 -3.35  59.0  4.70
## 5 -3.34  59.0  4.70
```

1. Geolocalized data

1.2. Opening geolocalized data

- This table format allows to easily plot raster data using the `geom_tile` geometry:

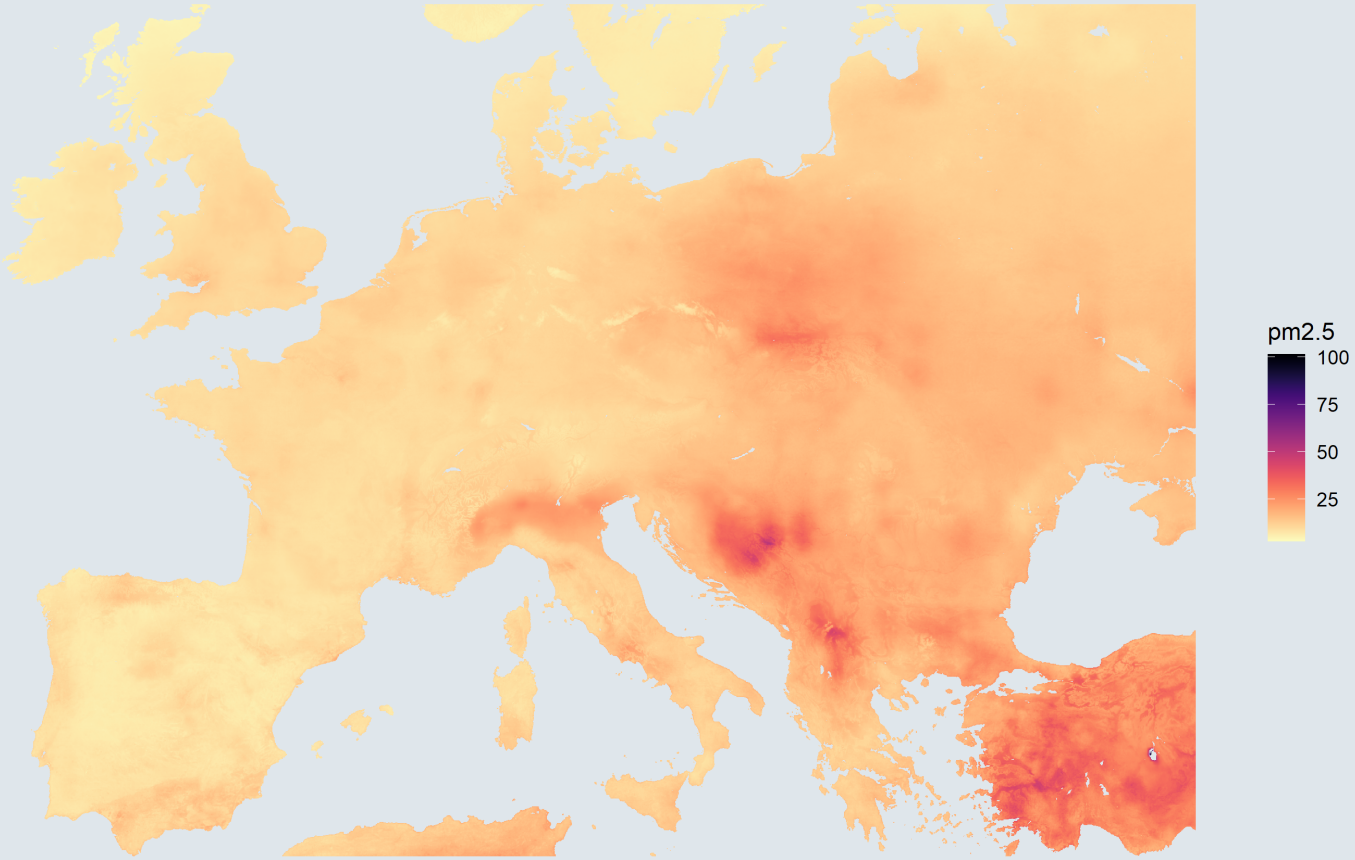
```
library(viridis) # for nice color gradient

ggplot(as_tibble(rasterToPoints(pm_data)), # Data converted in table
       aes(x = x,                          # Longitude on the x-axis
           y = y,                          # Latitude on the y-axis
           fill = pm2.5)) +                # Pollution on the fill-axis
  geom_tile() +                            # One filled square per cell
  scale_fill_viridis(option = "B",         # Nice color gradient
                     direction = -1) +     # The darker the more polluted
  theme_void()                             # Remove axes and grid
```

- This is gonna plot the 2,300 x 5,000 = 11,500,000 cells
  - With a color that is proportional to the value of `pm2.5`
  - Like the pixels of picture

1. Geolocalized data

1.2. Opening geolocalized data



1. Geolocalized data

1.3. Coordinate Reference Systems

- You might have noticed that France does not look the same on the two graphs
  - This is because have not **reprojected the data** yet
  - But this is the **first thing to do** when working with geolocalized data

- Right now our two maps may not be in the same CRS
  - CRS stands for **Coordinate Reference System**
  - It is a model of the Earth in which each location is coded using degrees
  - WGS84 is the standard CRS used by GPS, Google maps, ...
  - But it is **not suited to all applications**

- To visualize spatial data, we need to **project the surface of the globe on a plane**
  - There is **no correct way of doing that**
  - Like you cannot flatten an orange peel without distorting it

➜ There is a tradeoff between shape and scale preservation

1. Geolocalized data

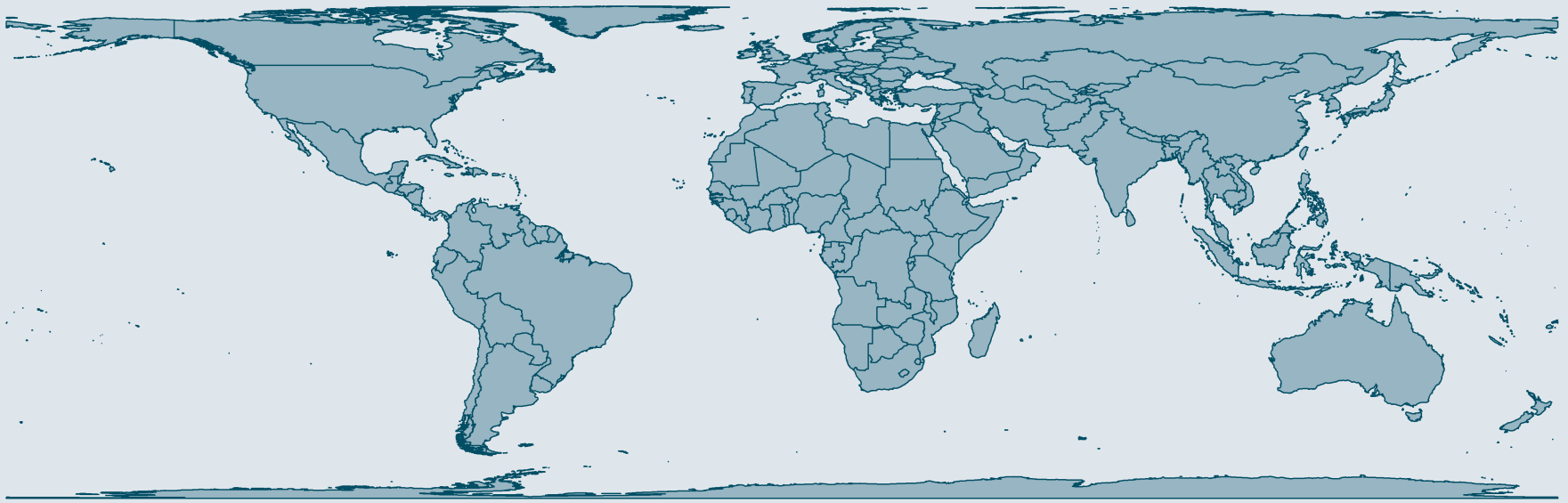
1.3. Coordinate Reference Systems

- For instance the Mercator projection preserves shape but distorts scale:

1. Geolocalized data

1.3. Coordinate Reference Systems

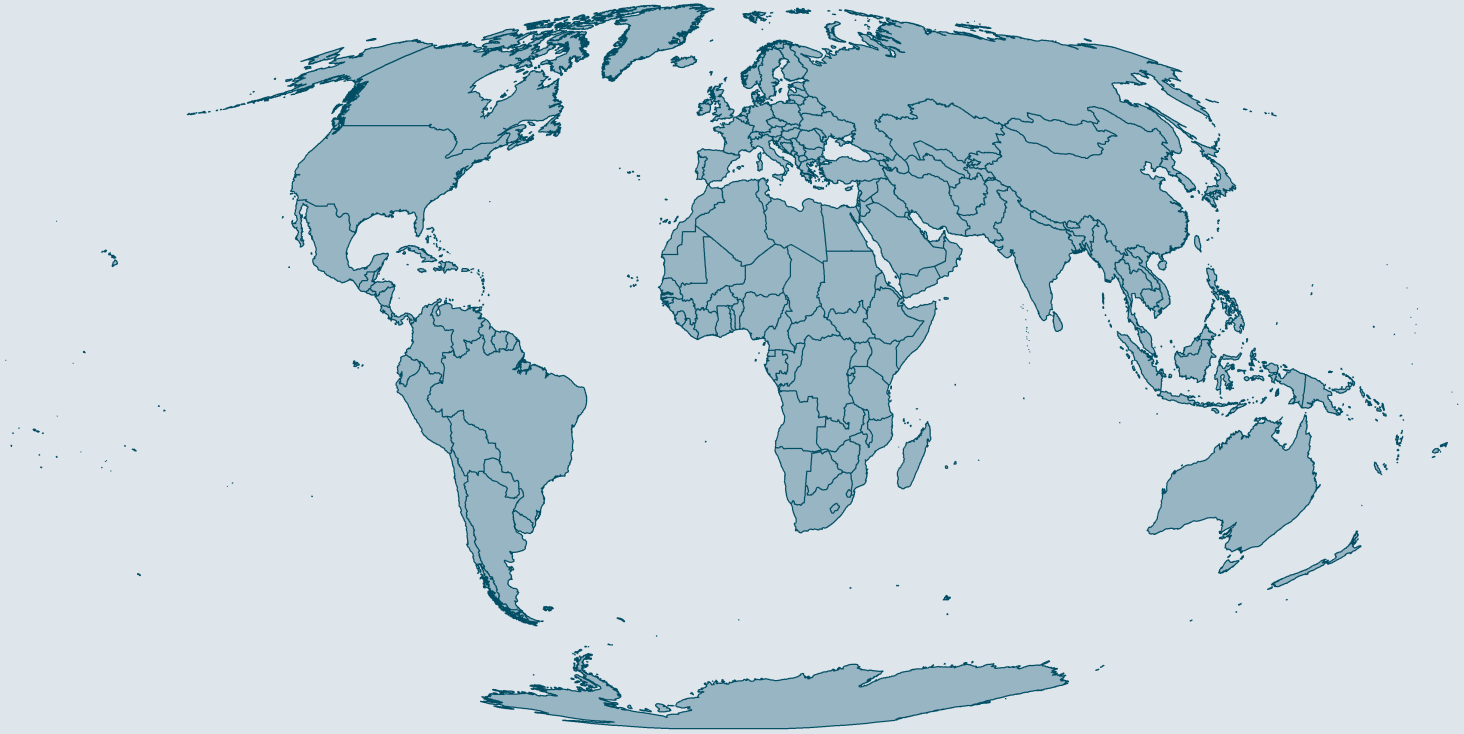
- While the Equal-Area Cylindrical projection preserves scale but distorts shape:



1. Geolocalized data

1.3. Coordinate Reference Systems

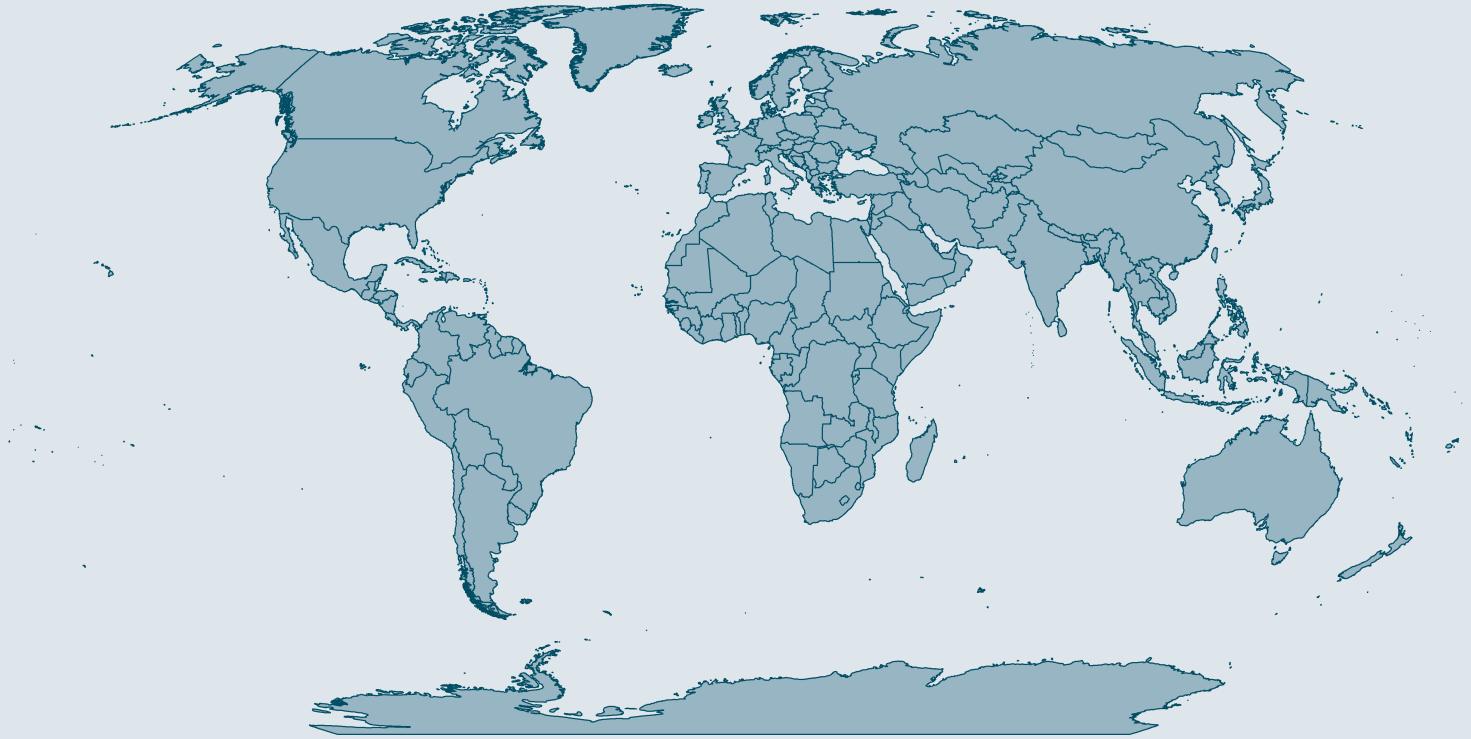
- This is also the case for the Mollweide projection:



1. Geolocalized data

1.3. Coordinate Reference Systems

- But most projections are in between, like the Robin projection:



1. Geolocalized data

1.3. Coordinate Reference Systems

- The smaller the area you want to map the less it is a problem
  - While you would have a hard time flatten the whole orange peel
  - Flattening a tiny bit of orange peel wouldn't require to distort it too much
- Even though there's no perfect projection, some are more suited to specific regions
  - You wouldn't use the Mercator projection if you focus on the poles
- The website epsg.io allows you to find the appropriate CRS for the area you want to map
  - For France, the recommended projection is Lambert 93, the corresponding EPSG code is 2154
  - The most common projection is WGS84, the corresponding EPSG code is 4326
  - What is important is that all the datasets are projected the same way

- For shapefile: `st_transform()`

```
dep_shp <- st_transform(dep_shp, "EPSG:4326")
```

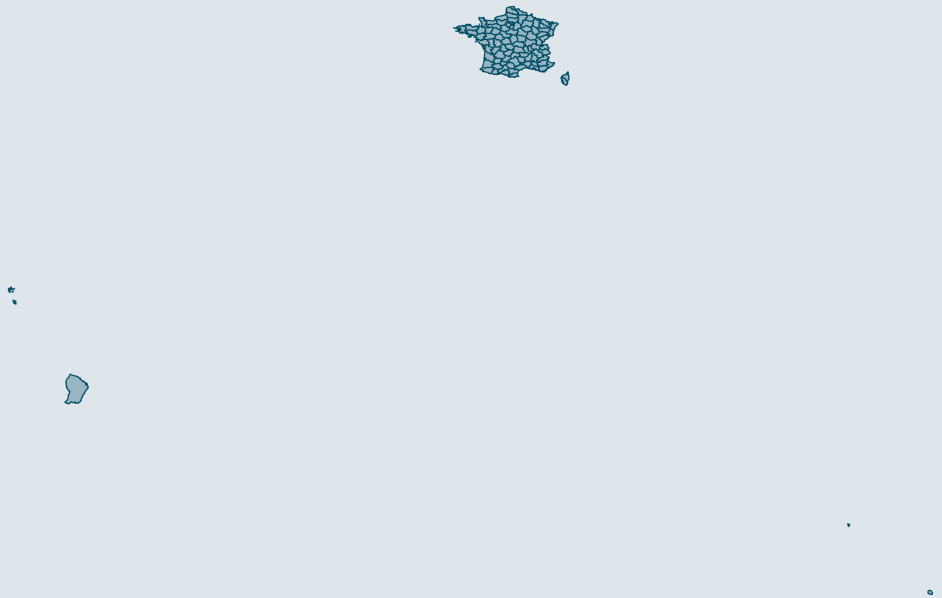
- For raster: `crs()`

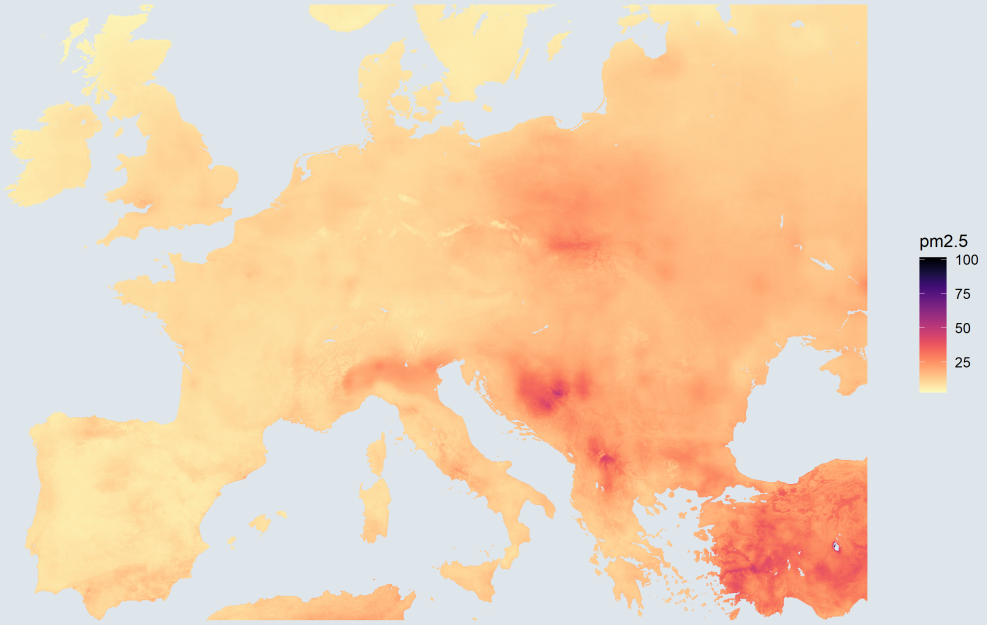
```
crs(pm_data) <- "EPSG:4326"
```

1. Geolocalized data

1.3. Coordinate Reference Systems

- Data projected in WGS84:



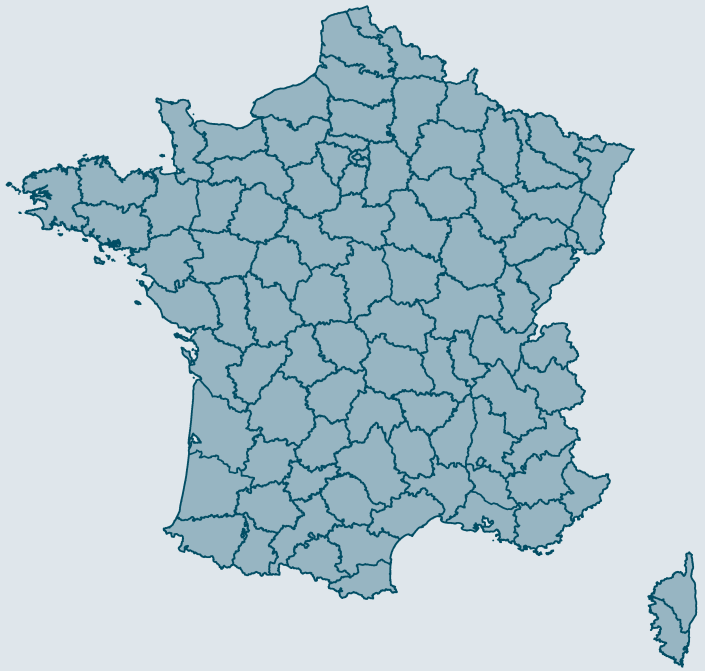


1. Geolocalized data

1.4. Subsetting geolocalized data

- To keep only metropolitan France in the shapefile, we can filter 2-digit department codes

```
dep_shp <- dep_shp %>% filter(nchar(INSEE_DEP) == 2)
```

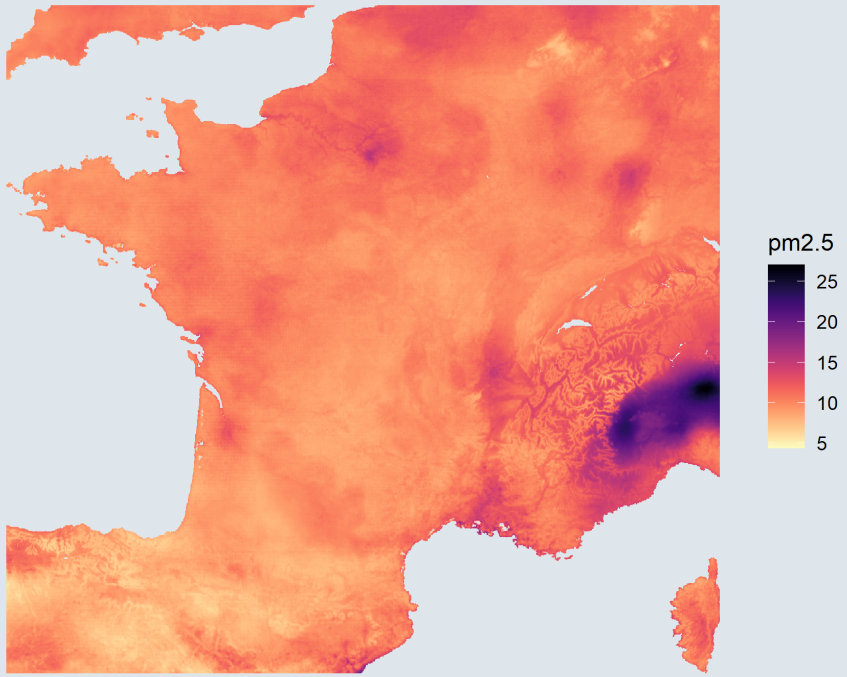


1. Geolocalized data

1.4. Subsetting geolocalized data

- Then, we can drop from the raster everything outside the longitude/latitude frame of our shapefile

```
pm_data <- crop(pm_data, extent(dep_shp))
```

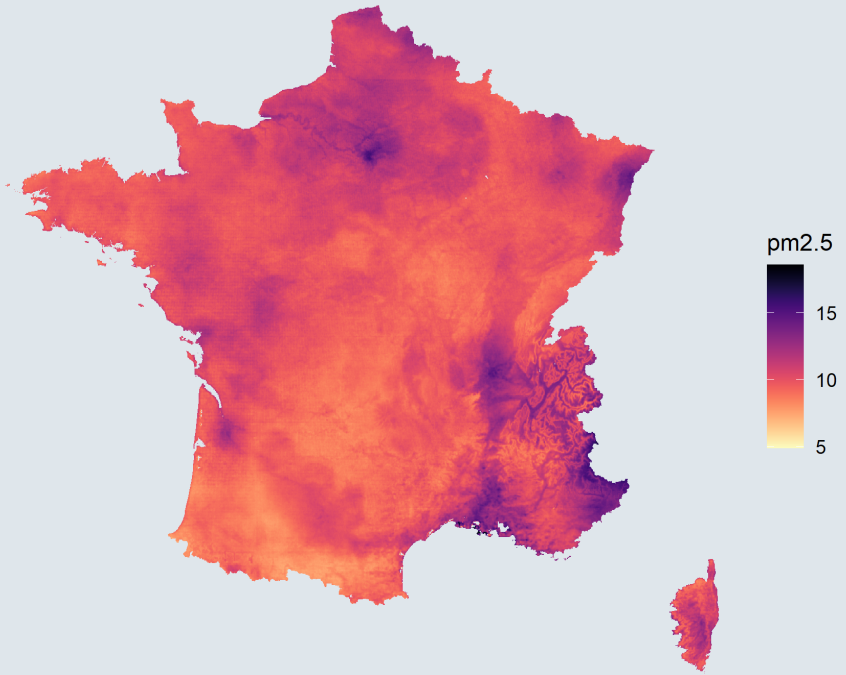


1. Geolocalized data

1.4. Subsetting geolocalized data

- We can then replace everything that is not in a polygon of the shapefile by missing values

```
pm_data <- mask(pm_data, dep_shp)
```

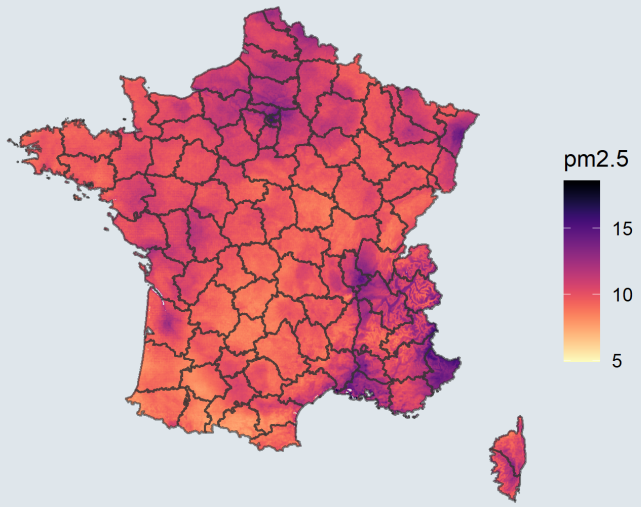


1. Geolocalized data

1.4. Subsetting geolocalized data

- The two datasets can now be overlaid:

```
ggplot() +
  geom_tile(data = as_tibble(rasterToPoints(pm_data)), aes(x = x, y = y, fill = pm2.5))  +
  geom_sf(data = dep_shp, fill = NA, color = alpha("grey20", .6)) +
  scale_fill_viridis(option = "A", direction = -1) + theme_void()
```



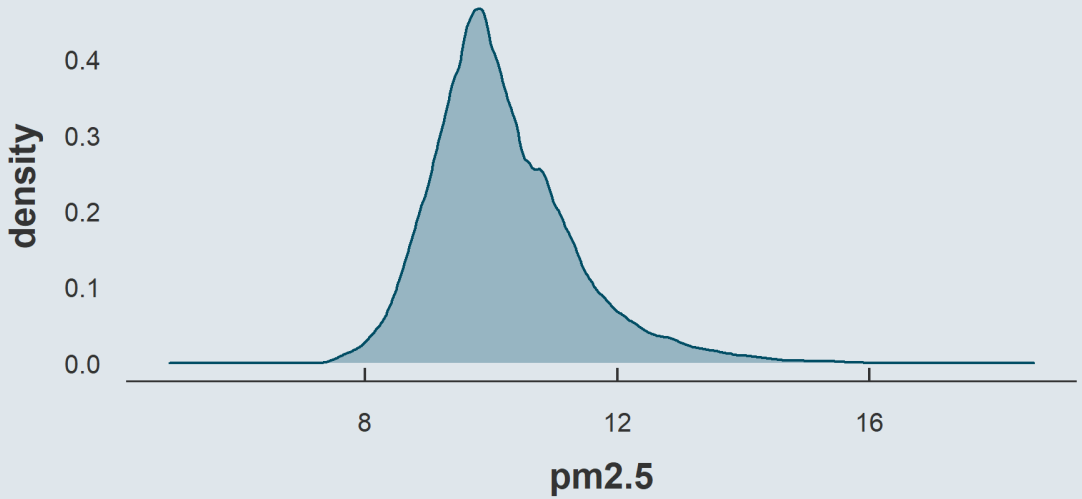
1. Geolocalized data

1.4. Subsetting geolocalized data

- We do not see much variation... Let's take a look at the $PM_{2.5}$ distribution

```
subset_data <- as_tibble(rasterToPoints(pm_data))

ggplot(subset_data, aes(x = pm2.5)) +
  geom_density(fill = "#6794A7", color = "#014D64", alpha = .6)
```



- There are **few** very high and very low **values that stretch the scale**

➜ *To better visualize the variations, we can* ***discretize the variable into deciles***

1. Geolocalized data

1.4. Subsetting geolocalized data

- To convert a continuous into deciles we should:
  1. Arrange the values in ascending order
  2. Compute the ratio of their index of N to get smthg $\in (0; 1]$
  3. Multiply by the desired number of quantiles and take the ceiling
  4. Set as factor so that R does not interpret it as continuous

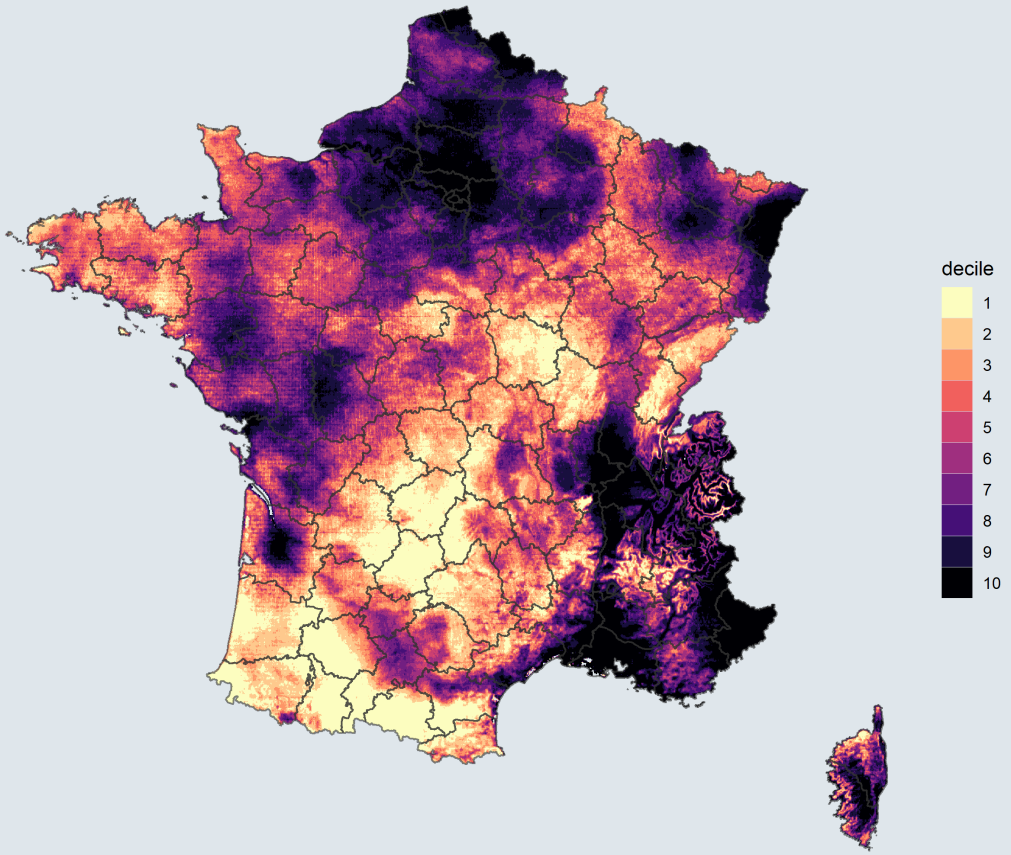
```
subset_data <- subset_data %>%
  arrange(pm2.5) %>%
  mutate(decile = as.factor(ceiling(10 * row_number()/n())))
```

- And plot the deciles instead of the continuous values

```
ggplot() +
  geom_tile(data = subset_data, aes(x = x, y = y, fill = decile))  +
  geom_sf(data = dep_shp, fill = NA, color = alpha("grey20", .6)) +
  scale_fill_viridis_d(option = "A", direction = -1) + theme_void()
```

1. Geolocalized data

1.4. Subsetting geolocalized data



Practice

First, make sure we're on the same page

```
# Load packages
library(tidyverse)
library(viridis)
library(sf)
library(raster)
select <- dplyr::select
library(ncdf4)

# Import data
dep_shp <- read_sf("data/dep_shp/DEPARTEMENT.shp")
pm_data <- raster("data/acag_2016.nc")

# Reproject data
dep_shp <- st_set_crs(dep_shp, "EPSG:4326")
crs(pm_data) <- "EPSG:4326"

# Rename PM 2.5 layer
names(pm_data) <- "pm2.5"
```

Practice

1) Filter only the Île-de-France region both in the shapefile and the raster

Hint: The Île-de-France geographic code is 11

2) Plot the $PM_{2.5}$ concentration in Île-de-France and overlay the department shapefile

You've got 5 minutes!

Solution

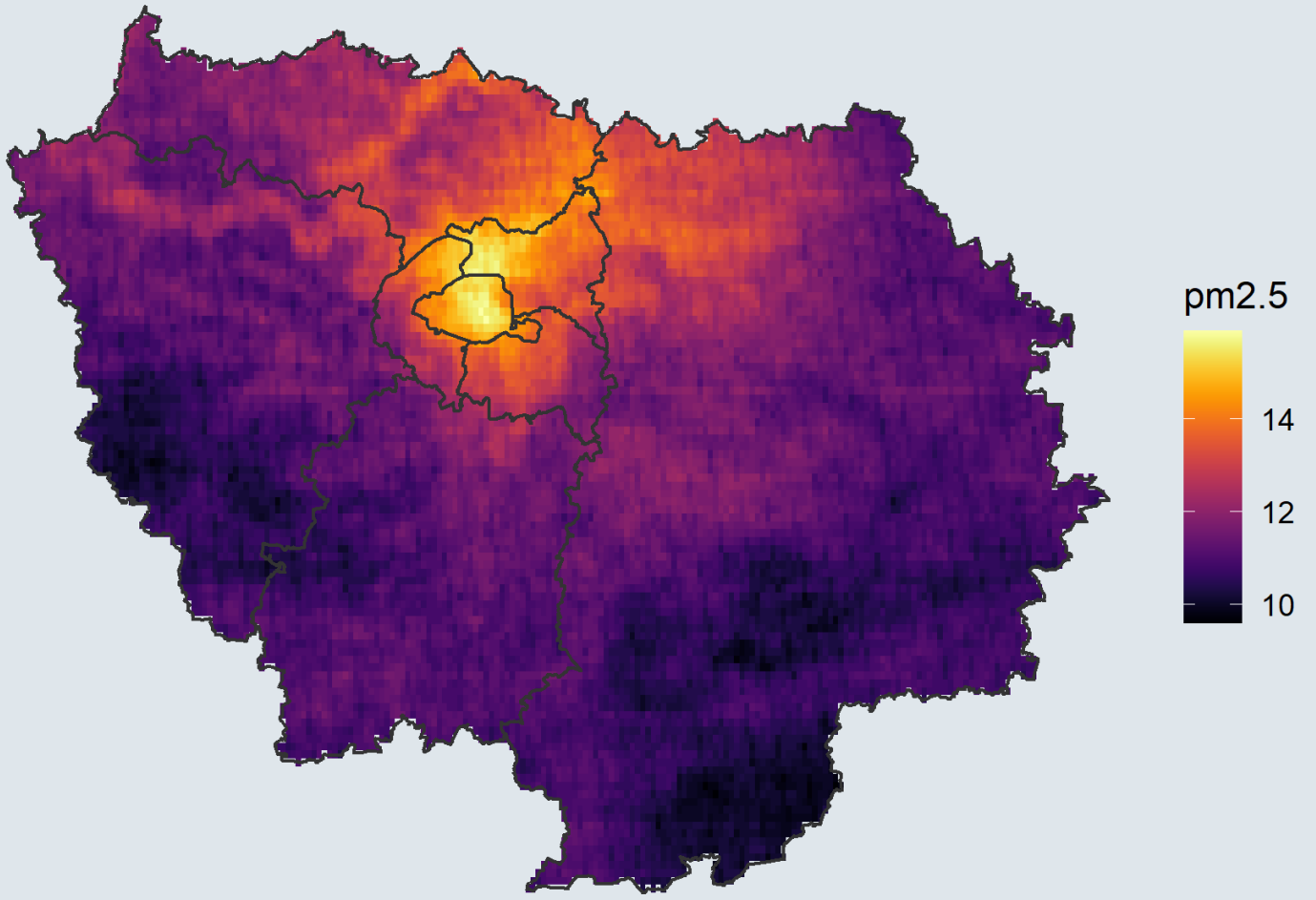
1) Filter only the Île-de-France region both in the shapefile and the raster

```
# Filter only departments of Île-de-France in shapefile
dep_shp <- dep_shp %>% filter(INSEE_REG == 11)
# Drop fromraster everything outside the lon/lat frame of the shapefile
pm_data <- crop(pm_data, extent(dep_shp))
# Replace everything that is not in a polygon of the shapefile by NA
pm_data <- mask(pm_data, dep_shp)
```

2) Plot the $PM_{2.5}$ concentration in Île-de-France and overlay the department shapefile

```
ggplot() +
  # Plot the raster
  geom_tile(data = as_tibble(rasterToPoints(pm_data)), aes(x = x, y = y, fill = pm2.5))  +
  # Plot the shapefile
  geom_sf(data = dep_shp, fill = NA, color =" grey20") +
  # Custom scale
  scale_fill_viridis(option = "B") + theme_void()
```

Solution



Overview

1. Geolocalized data ✓

- 1.1. Shapefiles and rasters
- 1.2. Opening geolocalized data
- 1.3. Coordinate Reference Systems
- 1.4. Subsetting geolocalized data

2. Geographic variables

- 2.1. Import from csv
- 2.2. Zonal statistics
- 2.3. Centroids and distance

3. Wrap up!

Overview

1. Geolocalized data ✓

- 1.1. Shapefiles and rasters
- 1.2. Opening geolocalized data
- 1.3. Coordinate Reference Systems
- 1.4. Subsetting geolocalized data

2. Geographic variables

- 2.1. Import from csv
- 2.2. Zonal statistics
- 2.3. Centroids and distance

2. Geographic variables

2.1. Import from csv

- The simplest type of geographic variable is those you can directly import in csv:
  - The Gini index of inequality of countries
  - The unemployment rate of departments
  - The number of inhabitants of cities
  - ...

- In this case you can simply
  - Import the csv data you need
  - Join it to the shapefile as you would do with any other data

➜ ***Let's study the relationship between income and exposure to air pollution at the city level in Île-de-France***

- We need:
  1. To import the shapefile of cities in Île-de-France
  2. To import and join median income by city

2. Geographic variables

2.1. Import from csv

- Import shapefile of cities in Île-de-France and project it in WGS84

```
idf_shp <- read_sf("data/idf_shp/idf.shp")
idf_shp <- st_set_crs(idf_shp, "EPSG:4326")
head(idf_shp, 5)
```

```
## Simple feature collection with 5 features and 5 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 1.643507 ymin: 48.36067 xmax: 2.365918 ymax: 49.17332
## Geodetic CRS:  WGS 84
## # A tibble: 5 x 6
##   NOM_COM           INSEE_COM INSEE_DEP INSEE_REG POPULATION                  geometry
##   <chr>             <chr>     <chr>     <chr>          <int>        <MULTIPOLYGON [°]>
## 1 Haute-Isle        95301     95        11               278 (((1.691893 49.07523, 1.~
## 2 Ambleville        95011     95        11               372 (((1.691683 49.12863, 1.~
## 3 Chalou-Moulineux  91131     91        11               431 (((2.043785 48.36067, 2.~
## 4 Morsang-sur-Orge  91434     91        11             20909 (((2.36231 48.64807, 2.3~
## 5 Vienne-en-Arthies 95656     95        11               416 (((1.74087 49.07267, 1.7~
```

2. Geographic variables

2.1. Import from csv

- Join csv data on median income by city and plot together with department shapefile
  - We shall make sure that the joining variables have the same name and class

```
insee_data <- as_tibble(read.csv("data/insee_2016.csv"))
head(insee_data, 1)
```

```
## # A tibble: 1 x 3
##   INSEE_COM CITY  MED16
##       <int> <chr> <dbl>
## 1     75056 Paris  26.8
```

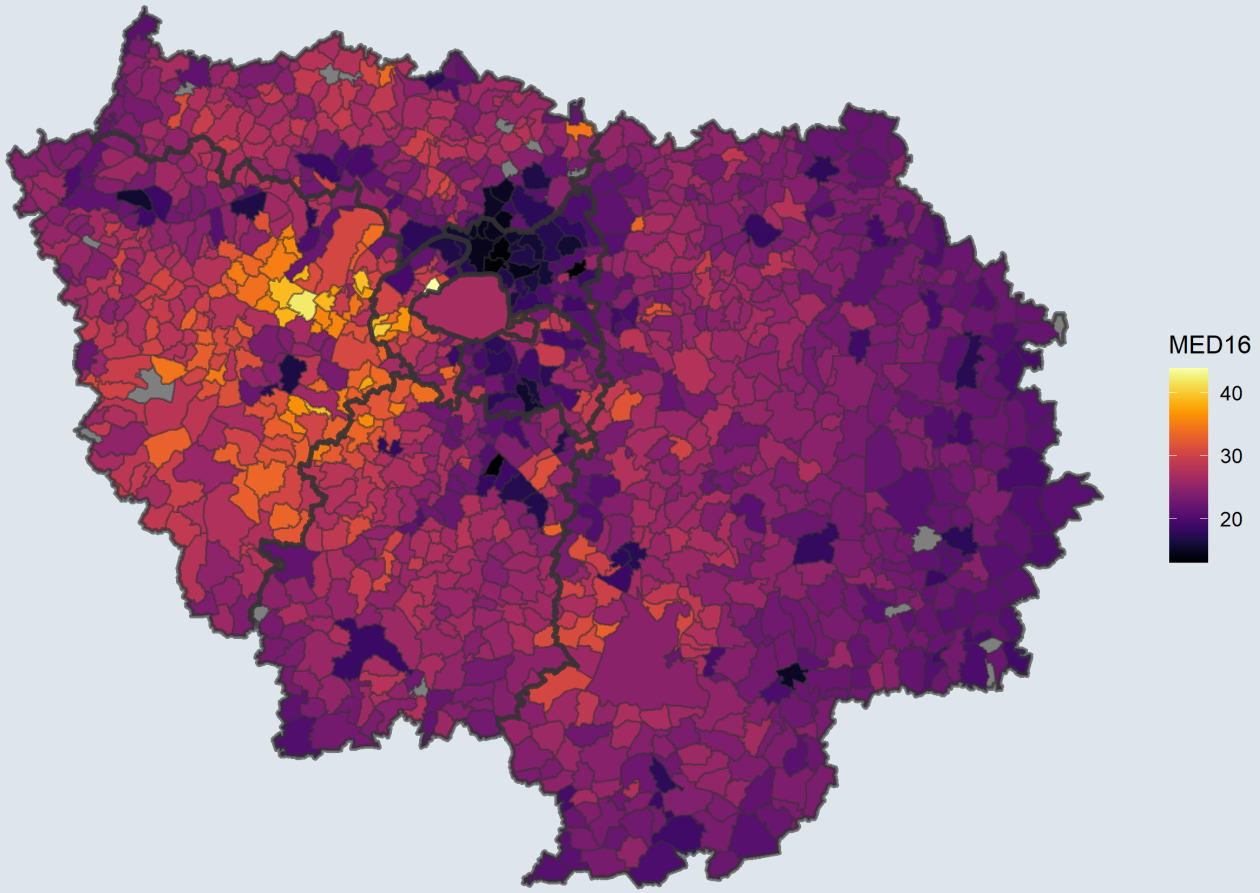
- Same name but not same class

```
idf_shp <- idf_shp %>% mutate(INSEE_COM = as.numeric(INSEE_COM)) %>% left_join(insee_data)

ggplot() + geom_sf(data = idf_shp, aes(fill = MED16), color = alpha("grey20", .4)) +
  geom_sf(data = dep_shp, fill = NA, color = alpha("grey20", .6), size = 1.2) +
  scale_fill_viridis(option = "B") + theme_void()
```

2. Geographic variables

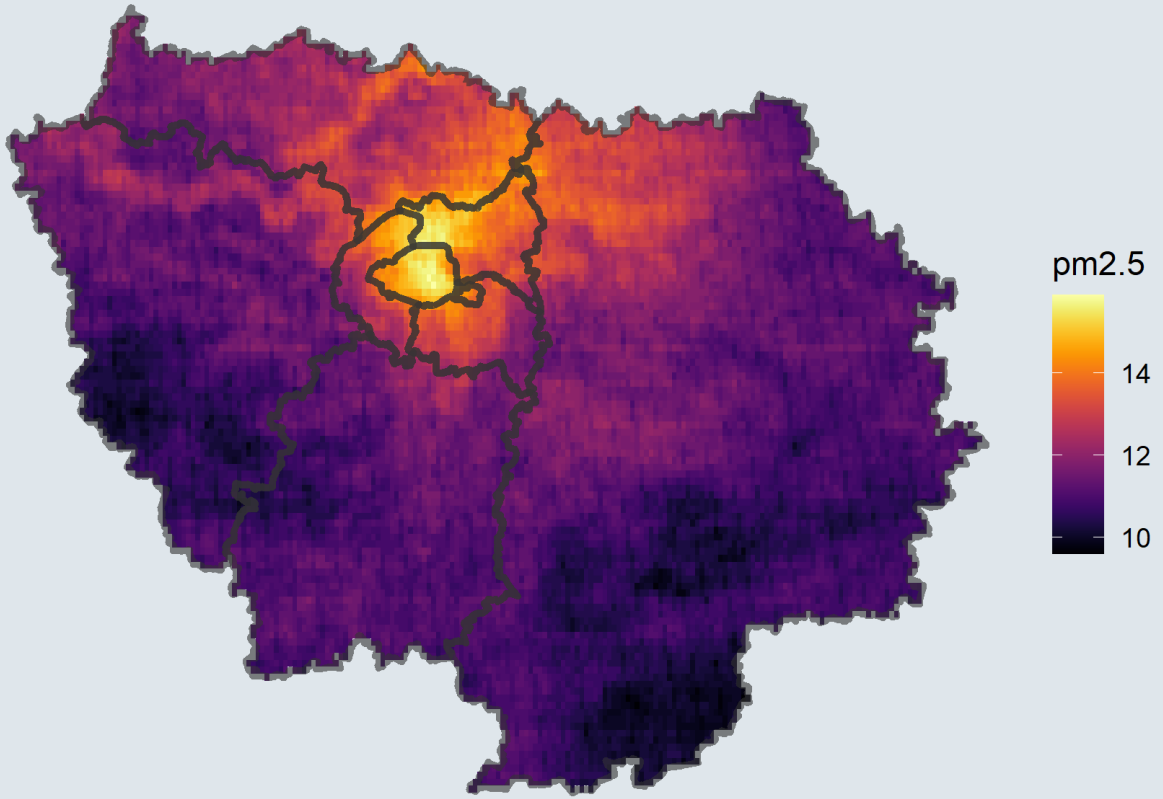
2.1. Import from csv



2. Geographic variables

2.2. Zonal statistics

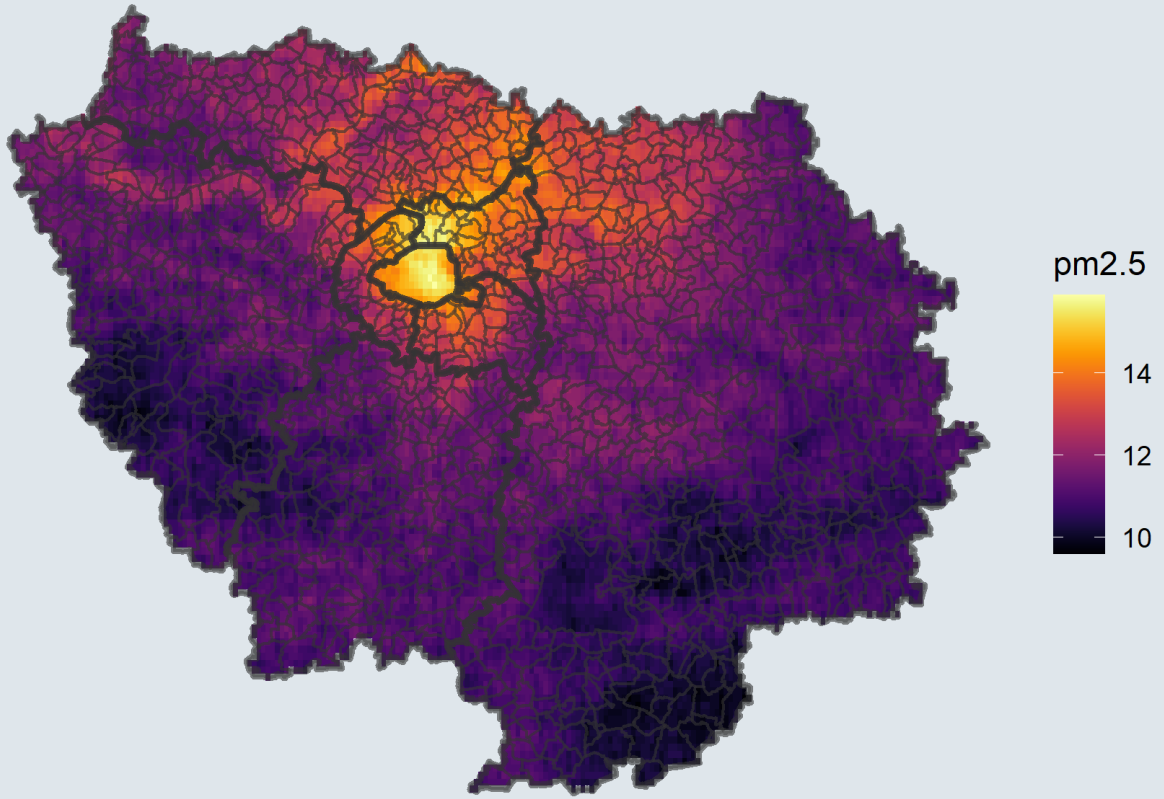
- Right now we have pollution at the cell level in our raster



2. Geographic variables

2.2. Zonal statistics

- And a shapefile delimiting the cities in which we want to know the pollution level



2. Geographic variables

2.2. Zonal statistics

- This is the principle of **zonal statistics:**
  - Compute **statistics on areas delimited by a shapefile from values of a raster**
  - We want the average $PM_{2.5}$ value of the cells that fall within the municipality geometry
  - With our shapefile and raster that cover the same area and are projected the same way

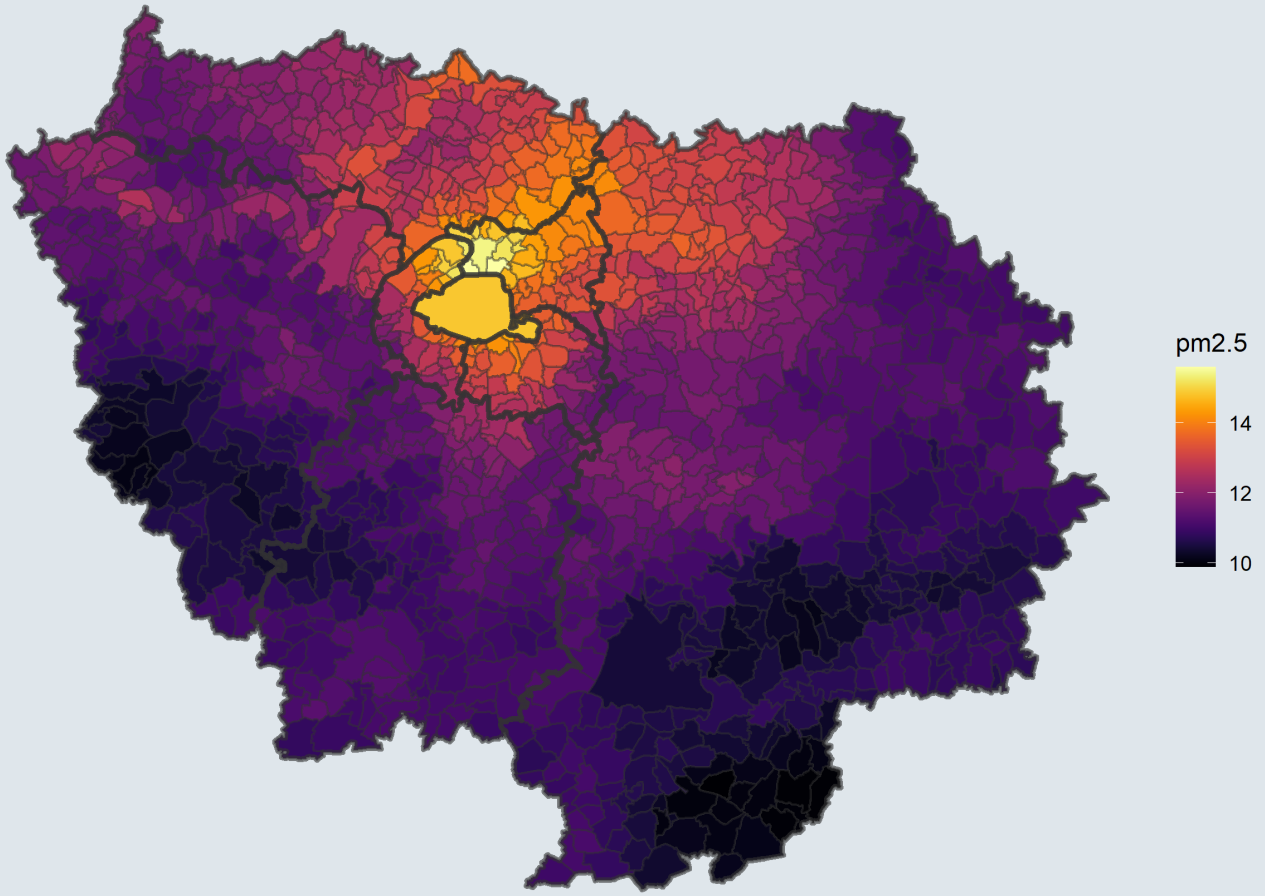
- In R zonal statistics can be computed with the function **extract()**
  - **x:** the data containing the values to compute statistics from
  - **y:** the data delimiting the zones in which to compute statistics
  - **FUN:** the statistic to compute (min, max, median, mean, ...)

```
idf_shp <- idf_shp %>% mutate(pm2.5 = extract(x = pm_data, y = ., fun = mean))

ggplot() +
  geom_sf(data = idf_shp, aes(fill = pm2.5), color = alpha("grey20", .4)) +
  geom_sf(data = dep_shp, fill = NA, color = alpha("grey20", .6), size = 1.2) +
  scale_fill_viridis(option = "B") + theme_void()
```

2. Geographic variables

2.2. Zonal statistics



2. Geographic variables

2.2. Zonal statistics

- We now have in the same dataset:
  - Average exposure to $PM_{2.5}$
  - Median income
  - Both at the city level

➜ Let's do the regression

```
stargazer(lm(pm2.5 ~ MED16, idf_shp),
          type = "text",
          keep.stat = c("n", "rsq"))
```

```
##
## ========================================
##                   Dependent variable:
##               ---------------------------
##                          pm2.5
## ----------------------------------------
## MED16                  -0.051***
##                         (0.007)
##
## Constant               13.037***
##                         (0.183)
##
## ----------------------------------------
## Observations             1,250
## R2                       0.040
## ========================================
## Note:        *p<0.1; **p<0.05; ***p<0.01
```

2. Geographic variables

2.3. Centroids and distances

Results indicate that the city level in in Île-de-France an increase of 1,000 euros is associated with a reduction of 0.05 µg/m^3 in the concentration of $PM_{2.5}$, ceteris paribus

- But could this effect just be do to the distance to Paris?
  - Maybe richer individuals tend to live away from the urban center
  - And thus to be less exposed to pollution

- We need to control for the distance to Paris
  - To do so we can find the location of the centroid of each municipality with st_centroid()
  - It corresponds to the arithmetic mean position of all the points in the polygon

```
idf_shp <- idf_shp %>%
  mutate(centroid = st_centroid(geometry))

ggplot() +
  geom_sf(data = idf_shp$geometry, fill = "#6794A7", color = "#014D64", alpha = .6) +
  geom_sf(data = idf_shp$centroid, color = "#014D64") + theme_void()
```

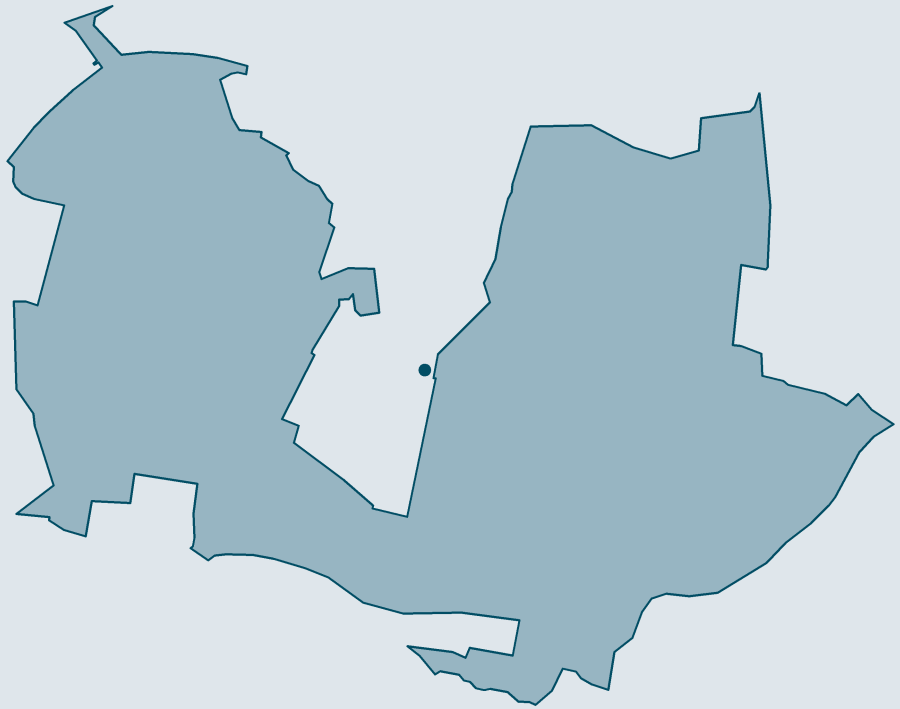
2. Geographic variables

2.3. Centroids and distances

2. Geographic variables

2.3. Centroids and distances

- Note that the centroid of a polygon can be outside the polygon
  - Take for instance the municipality Les Ulis:



2. Geographic variables

2.3. Centroids and distances

- To compute distances we should first convert the centroid variable into a longitude and a latitude variable

```
idf_shp <- idf_shp %>%
  group_by(INSEE_COM) %>%
  mutate(cent_lon = unlist(centroid)[1], cent_lat = unlist(centroid)[2])
```

- And store the coordinates of Paris

```
paris <- idf_shp %>%
  filter(NOM_COM == "Paris") %>%
  select(cent_lon, cent_lat) %>% st_drop_geometry() # Specific function to drop geometry

paris
```

```
## # A tibble: 1 x 2
##   cent_lon cent_lat
## *    <dbl>    <dbl>
## 1     2.34     48.9
```

2. Geographic variables

2.3. Centroids and distances

- We can now compute the distance between each centroid and that of Paris using `geodist_vec()`

```
library(geodist)
idf_shp <- idf_shp %>% mutate(dist_paris = geodist_vec(x1 = cent_lon, y1 = cent_lat,
                                                       x2 = paris$cent_lon, y2 = paris$cent_lat,
                                                       measure = "geodesic") / 1000)
```

- We can plot this new variable

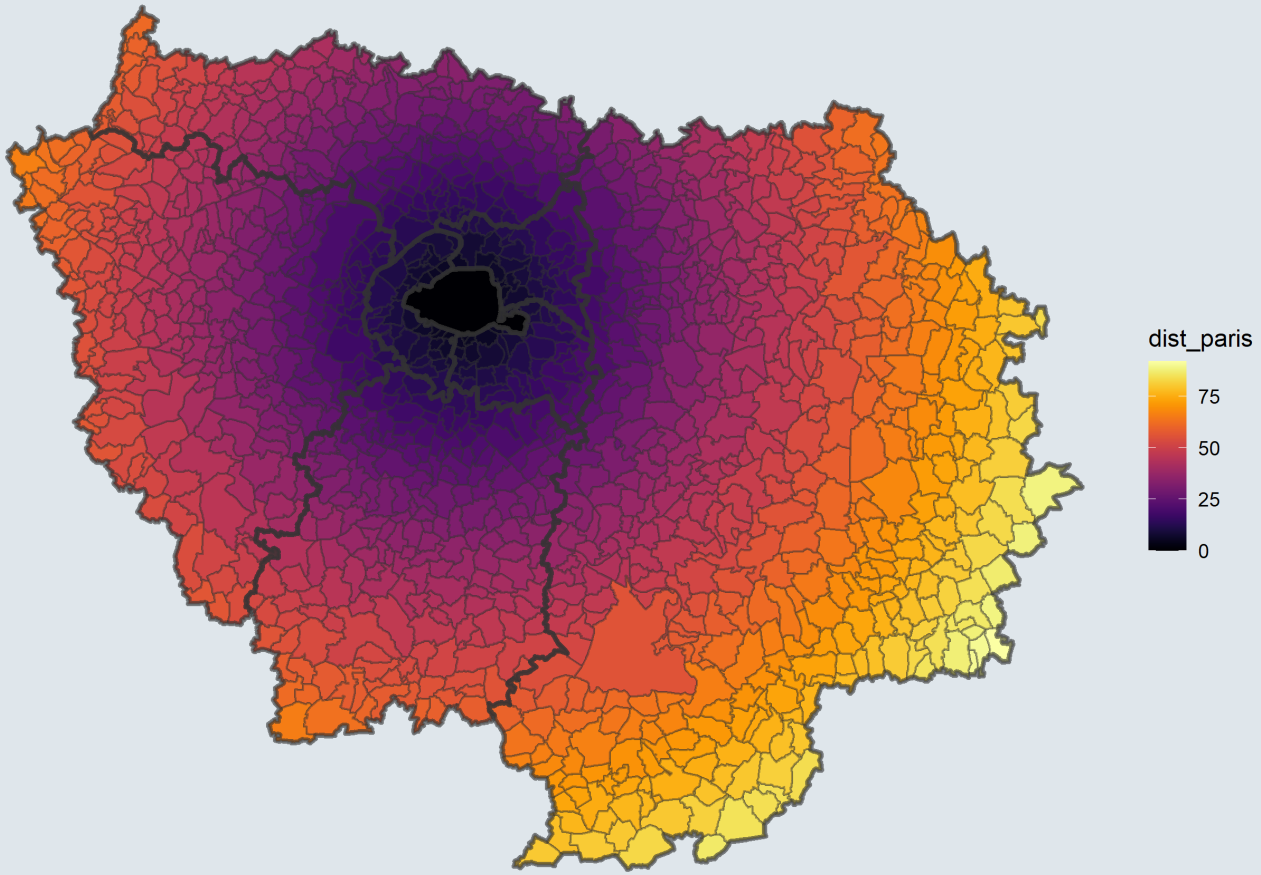
```
ggplot() +
  geom_sf(data = idf_shp, aes(fill = dist_paris), color = alpha("grey20", .4)) +
  geom_sf(data = dep_shp, fill = NA, color = alpha("grey20", .6), size = 1.2) +
  scale_fill_viridis(option = "B") + theme_void()
```

- And add it in the regression

```
stargazer(lm(pm2.5 ~ MED16 + dist_paris, idf_shp), type = "text", keep.stat = c("n", "rsq"))
```

2. Geographic variables

2.3. Centroids and distances



2. Geographic variables

2.3. Centroids and distances

```
##
## ========================================
##                   Dependent variable:
##               ---------------------------
##                           pm2.5
## ----------------------------------------
## MED16                   -0.092***
##                          (0.005)
##
## dist_paris              -0.041***
##                          (0.001)
##
## Constant                15.749***
##                          (0.131)
##
## ----------------------------------------
## Observations              1,250
## R2                        0.620
## ========================================
## Note:         *p<0.1; **p<0.05; ***p<0.01
```

- Once controlling for distance to Paris, the coefficients associated with median income is even more negative
  - Distance to Paris has opposite relationships with pollution and median income
  - Richer municipalities tend to be closer to Paris
  - But also to be less polluted than poorer municipalities

- In Île-de-France the relationship between exposure to $PM_{2.5}$ and median income (in thousand euros) is even stronger than with distance to Paris (in km)
  - Both coefficients are significant at 99% confidence level

Overview

1. Geolocalized data ✓

- 1.1. Shapefiles and rasters
- 1.2. Opening geolocalized data
- 1.3. Coordinate Reference Systems
- 1.4. Subsetting geolocalized data

2. Geographic variables ✓

- 2.1. Import from csv
- 2.2. Zonal statistics
- 2.3. Centroids and distance

3. Wrap up!

3. Wrap up!

Shapefiles and rasters

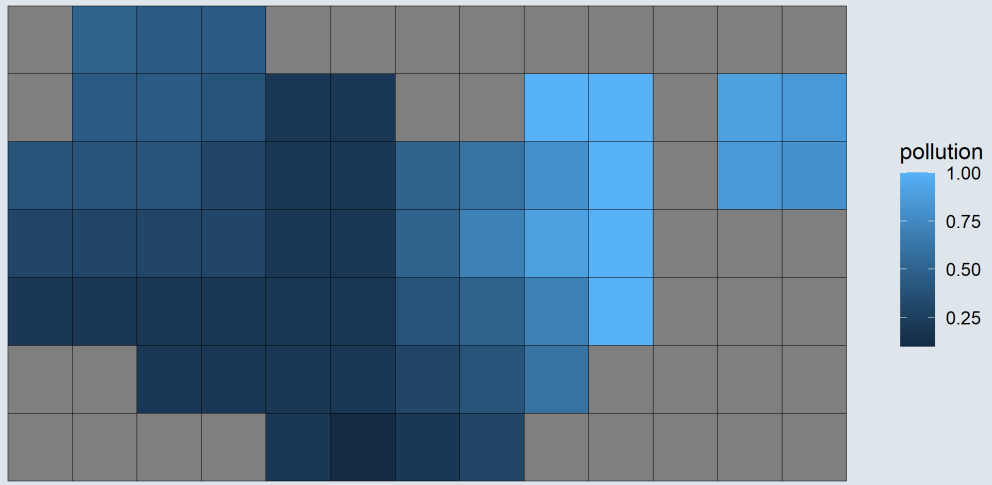
Two main types of geolocalized datasets:

Shapefiles

- One row per entity/one column per variable
- A geometry variable with the coordinates of the points/polylines/polygons

Rasters

- Works like a picture, with cells like pixels
- And each cell can take a given value, e.g. pollution observed from satellites



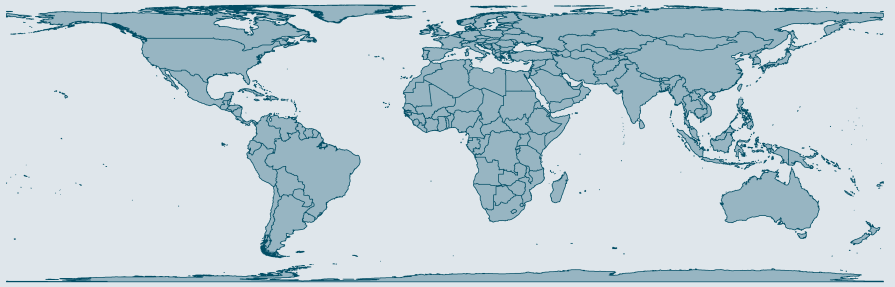
3. Wrap up!

Coordinates Reference Systems

- A Coordinate Reference System (CRS) is a model of the Earth in which each location is coded using degrees
  - It allows to **project the surface of the globe on a plane**
  - But there is a **tradeoff** between preserving:

Shape (like the Mercator projection)

Scale (like the Equal-Area Cylindrical projection)



  - Most projections are somewhere in between
  - For France: Lambert 93 projection (EPSG:2154)

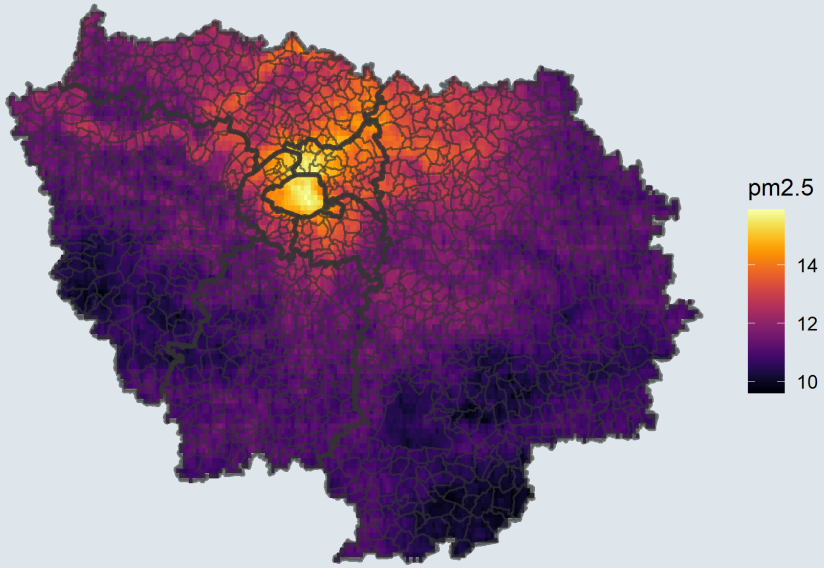
➜ *First thing to do:* ***reprojection***

3. Wrap up!

Operations on geolocalized data

Zonal statistics

- Computing statistics on areas delimited by a shapefile from values of a raster
  - Project shapefile and raster the same way
  - Compute the mean/max/... of cell values



Centroids

- The centroid is the arithmetic mean position of all the points in the polygon
  - To compute distances between polygons
  - A centroid is not always within its polygon