Simple Handling of Spatial Data and Viz

Mattia Albertini

7/7/2022

Disclaimer

For this example, the georeferenced data are made up. Moreover, for simplicity I will avoid most of the discussion on Coordinate Reference Systems. To have more info visit Google :)

Set Up

```
# Useful applications for coding
library(tidyverse)  # include many packages such as dplyr, readxl etc

library(magrittr)   # operator  %>%

# Make stats
library(gtsummary)
library(qwraps2)


# Deal with Geocoding
library(sf)         # loads classes and functions for vector data.

library(rmapshaper) # geodata

# Making maps
library(tmap)

library(mapview)

library(leaflet)    # Maps
```

Load Data

In this section we learn how to spatial data, i.e. a set of data that has one column more than a simple dataset: a “geometry” (o geom) column. This column contains the spatial information of the spatial object.

A correct handling of the data requires two set of data:

- A “shape file”: Each rows represent a geographical unit, i.e. a municipality, a country etc. In the geometry column we have “polygons” contructed with sets of points and lines so to represent the geographical unit.

- A “Georeferenced Dataset”: Each rows contains information about a unit of observations. For example each row can be a firm and we may have information about the firm’s revenue and the geometry column represents the lat-long coordinates of the firm.

What we want to do is:

- Plot the firms on a map and on the shape file
- Merge the firms with the shape file (so to get rid of firms outside the geographical unit boundaries)
- Make summary stats at level of geographical unit and represent them

Load Georeferenced Data

The first step is uploading our geographical data about firms in Switzerland (these are not real) and to do so we use the command “st_read”. There are 2 remarks:

1) In the options of st_read() we need to specify the column names of latitude and longitude (remember Longitude is “X” and Latitude is “Y”).

2) We need to specify in which crs our original data were written: this must be asked to the data provider

The following command will upload our data and automatically read the excel/csv columns containing the longitude and latitude in a unique column in R: YES, the GEOMETRY COLUMN!

```
firms <- st_read(dir, options = c("X_POSSIBLE_NAMES=longitude2",
"Y_POSSIBLE_NAMES=latitude2"), stringsAsFactors=FALSE, crs = 4326)
```

Let’s generate a unique index by firm,

```
firms$key <- seq.int(nrow(firms))
```

Now, let’s take a look at the geometry column

```
head(firms[,c("key", "geometry")])

## Simple feature collection with 6 features and 1 field
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 7.77698 ymin: 47.0532 xmax: 9.44179 ymax: 47.4463
## Geodetic CRS:  WGS 84
##          key                geometry
## 196084   1    POINT    (9.44179 47.0532)
## 187207   2    POINT    (8.70176 47.4432)
## 169094   3    POINT    (7.77698 47.2172)
## 210418   4    POINT    (8.2317 47.373)
## 148321   5    POINT    (8.16728 47.4463)
## 103359   6    POINT    (8.32728 47.409)
```

We can see that for each firm we have an attribute containing a couple of coordinates, which identify our firm location (a point).

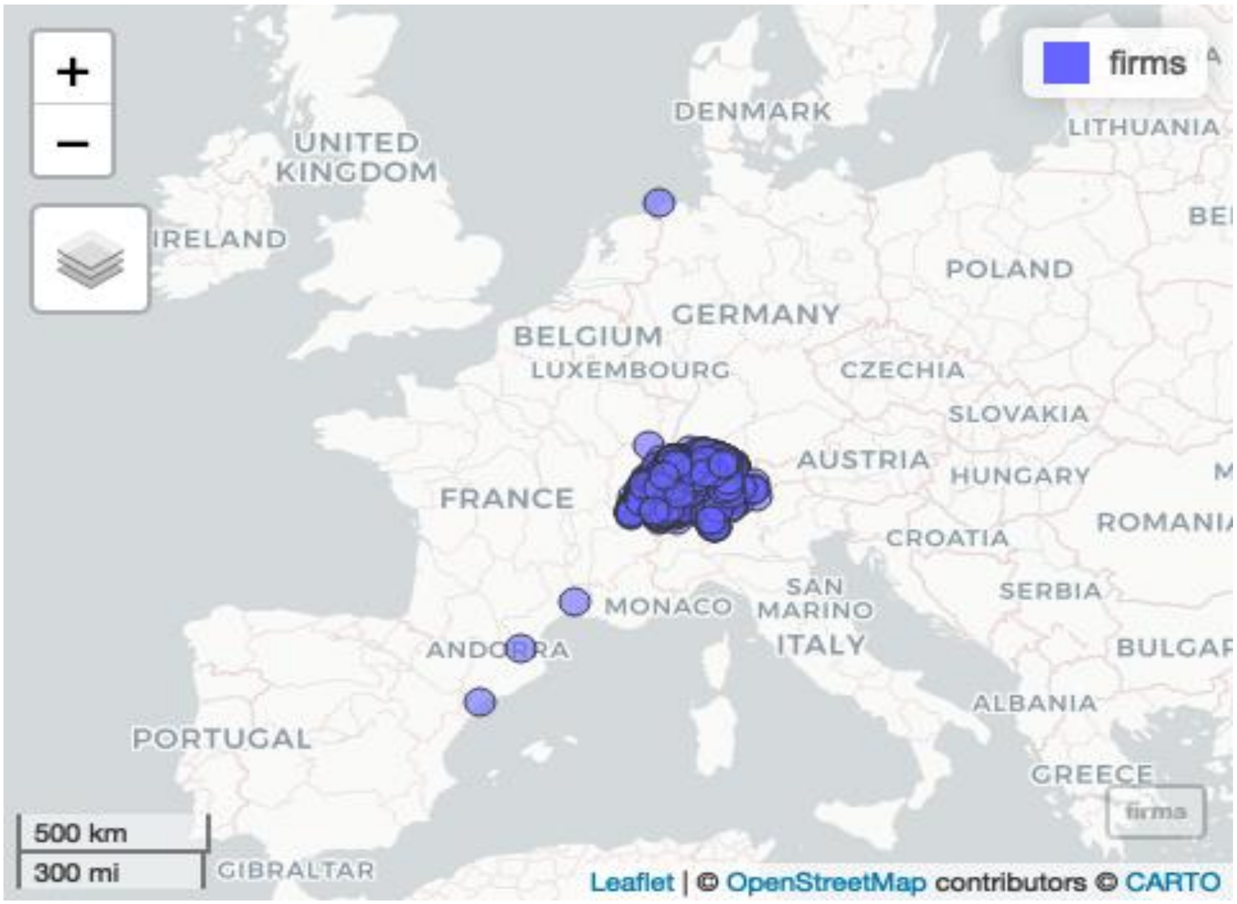
We can plot them to visualize these firms: We can do so in three ways:

1) Using the command "mapview"
2) Using Leaflet

Plot

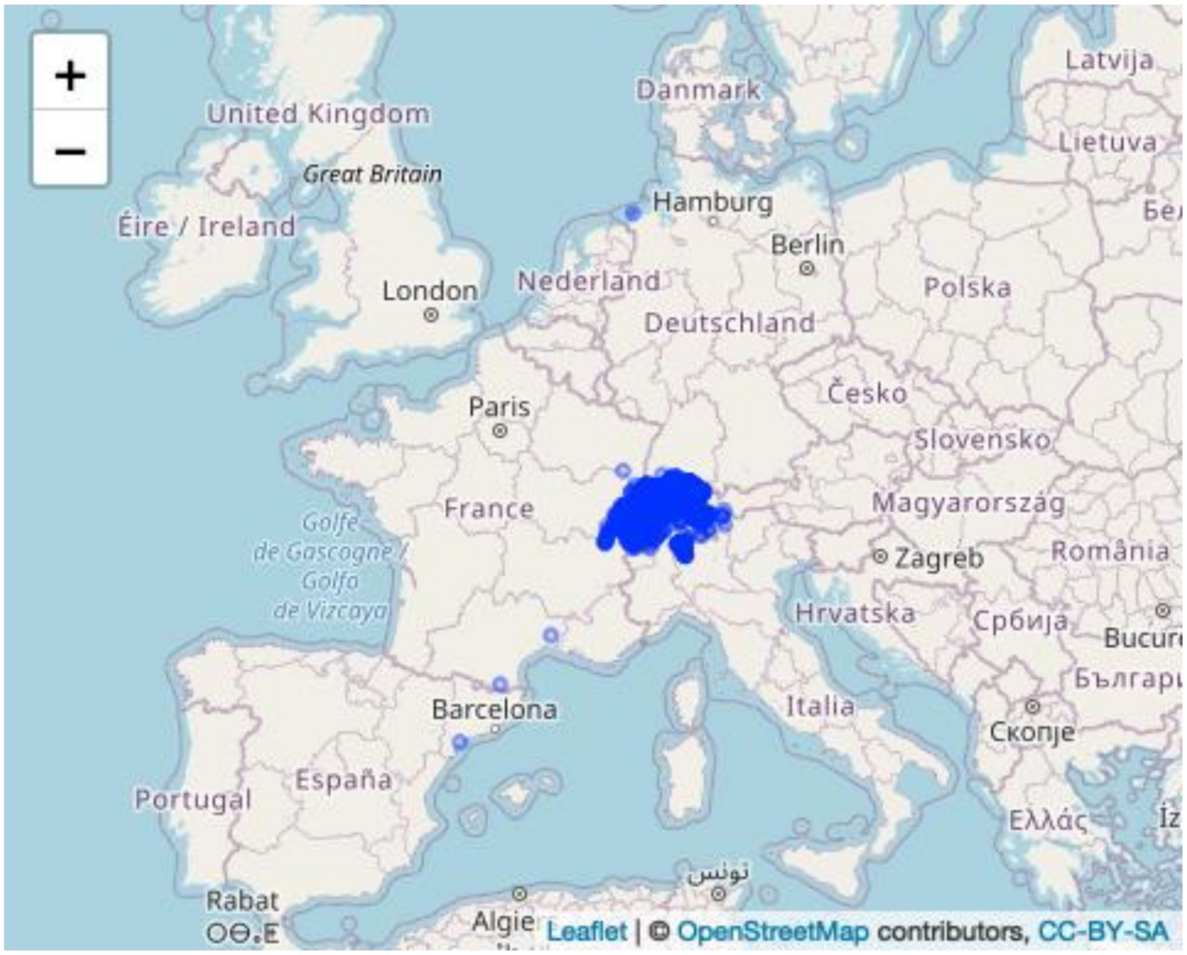
Mapview plots the coordinates on one of the google maps (you can switch between layers)

```
mapview(firms)
```



Leaflet plots the coordinates on one of the leaflet maps

```
firms %>% leaflet() %>% addTiles %>% addCircles
```



It may happen that some data have been included wrongly in our dataset: i.e. they lie out outside our boundaries. I.e. we have firms in Spain, in France etc that do not belong to our dataset. Because we do not know where to place these firms, we want to get rid of them. To do so we merge these data with the SHAPE FILE! Let's see how we do it.

Load Shape File boundaries

The first step is uploading the shape file of swiss municipalities. It can be downloaded from the BAFU website (Federal statistical office of Switzerland > Geodata)

This shape files is called "swissBOUNDARIES3D_1_3_TLM_HOHEITSGEBIET.shp": Note that the extension is .shp. To upload it we simply use st_read as before, but now we do not specify the long and lat:the shape file is special, it contains polygons, and the st_read automatically detects the shape file and crs and uploads it as simply as follows.

```
mun <- st_read(dir2)
```

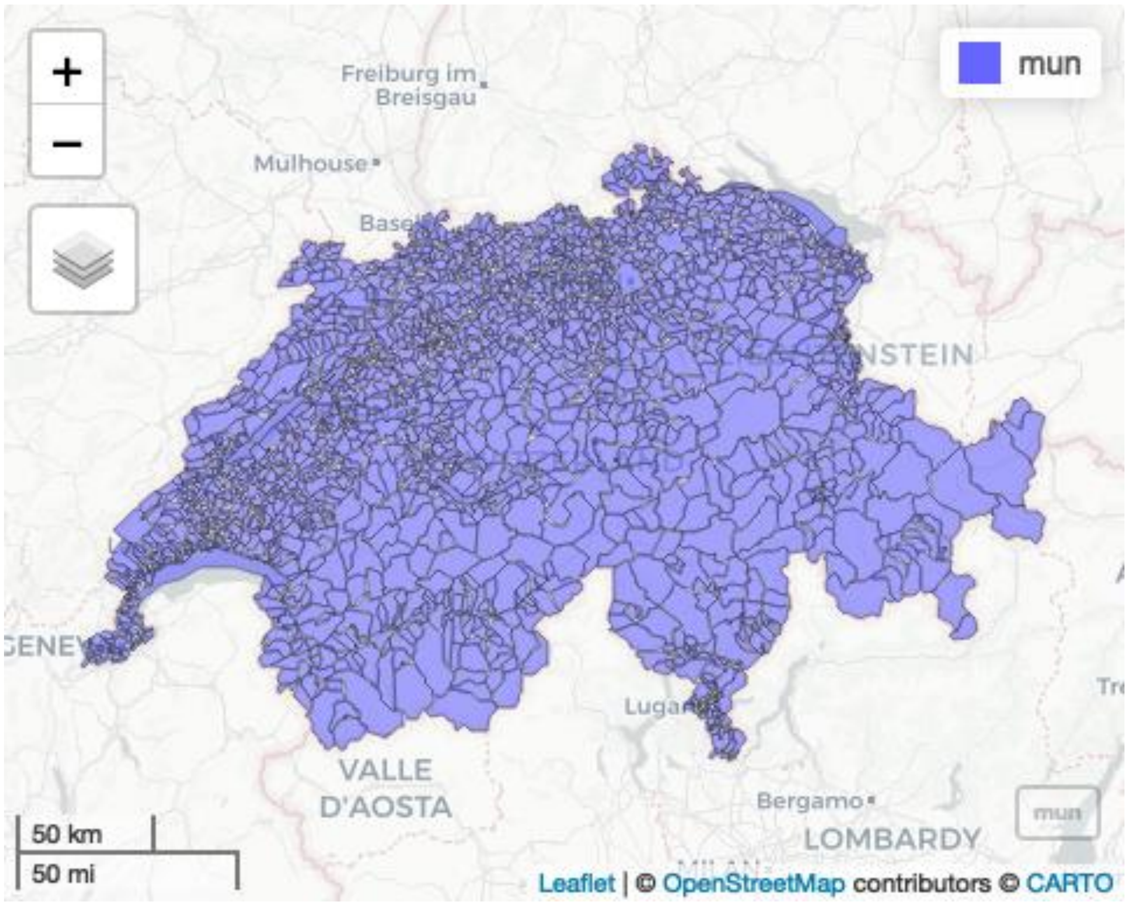
Clearly, in place of "dir2" you should put your directory.

```
head(mun[,c("BFS_NUMMER", "geometry")])
```

```
## Simple feature collection with 6 features and 1 field
## Geometry type: POLYGON
## Dimension:     XYZ
## Bounding box:  xmin: 2505634 ymin: 1082762 xmax: 2829555 ymax: 1211272
## z_range:       zmin: 371.6562 zmax: 4311.194
## Projected CRS: CH1903+ / LV95 + LN02 height
##      BFS_NUMMER                    geometry
## 1       3762         POLYGON Z ((2812956 1194956...
## 2       1631         POLYGON Z ((2715953 1184819...
## 3       3746         POLYGON Z ((2803369 1191207...
## 4       3543         POLYGON Z ((2757776 1165789...
## 5       6037         POLYGON Z ((2588811 1085591...
## 6       9758         POLYGON Z ((2554944 1145201...
```

As you can see for every BFS_NUMMER (identifies in Switzerland the mucipality number) we have a complicated geomtetrical object. Now lets visualize these objects or polygons using “mapview”

```
mapview(mun)
```



As you can see these are the Swiss municiaplities!

Join the Shape Files

As mentioned earlier, we want to get rid of spatial outliers and contrust statistics at municipality level. To do so we need to join the georeferenced data with firms to the municipality shape file. Before doing that straight, we need to do one thing: make sure the CRS is the same between the georeferenced data and the shape file (see disclaimer on CRS, there are many, and any data provider may use the one he wants to geocode the data)

Make sure the Coordinate Reference System (CRS) is the same

Let's verify if the crs between our data is the same. We use the function st_crs() which returns the crs of the spatial object, and the logical operator "==" to verify if it returns TRUE or FALSE when making the comparison.

```
# Returns false
st_crs(firms)==st_crs(mun)

## [1] FALSE
```

So they are not the same... Let's reproject the data! Re-projecting means converting the CRS or "the scale" of the coordinates of the georeferenced data in the same crs as the shape file. We use the function st_transform() which takes as first argument the data to be transformed, and as second argument the CRS we want to transform the data to. We specify the crs of the "mun" shape file.

```
firms_proj <- st_transform(firms, st_crs(mun))
```

And now..

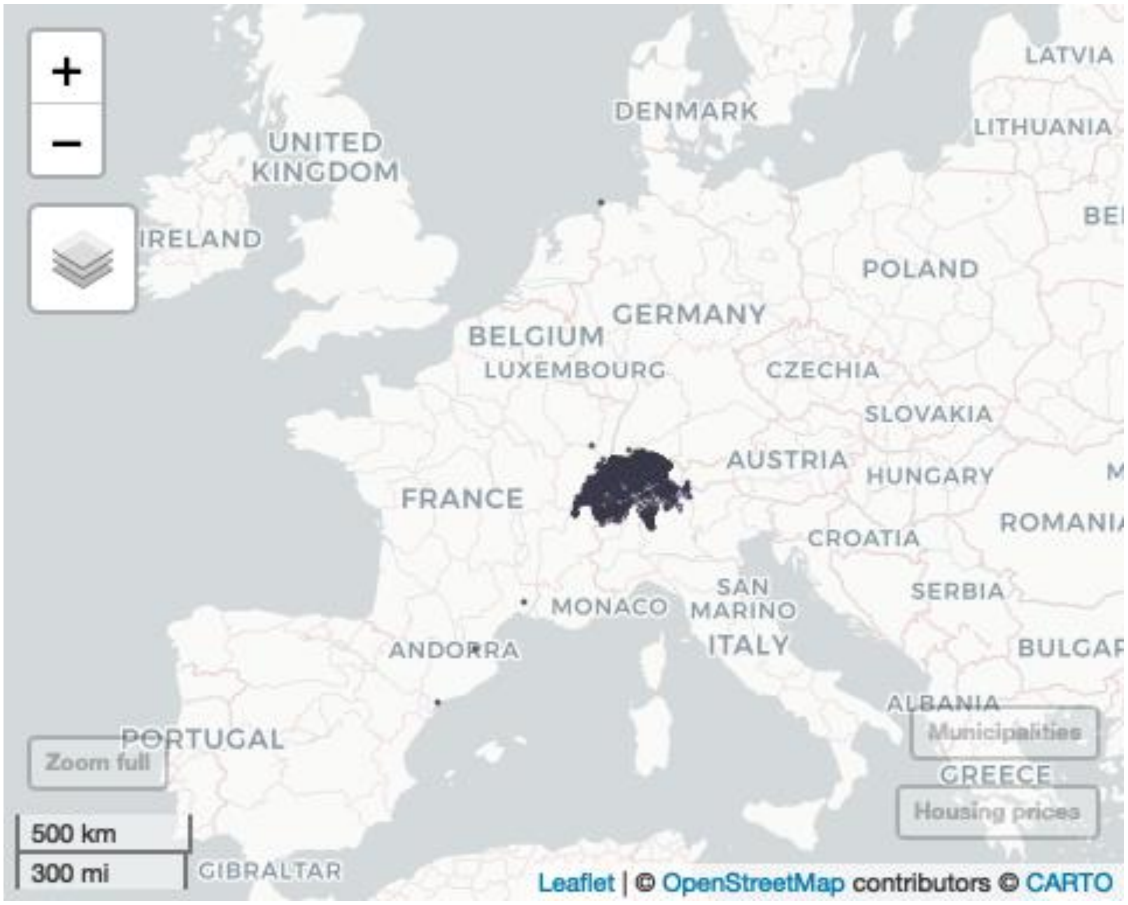
```
# Returns true
st_crs(firms_proj)==st_crs(mun)

## [1] TRUE
```

Eureka!

Visualize Spatial Outliers

Let's see if these spatial outliers exist

```
mapview(firms_proj, cex = 0.1, layer.name=("Housing prices"), legend = FALSE) +
mapview(mun, layer.name=("Municipalities"), legend = FALSE)
```



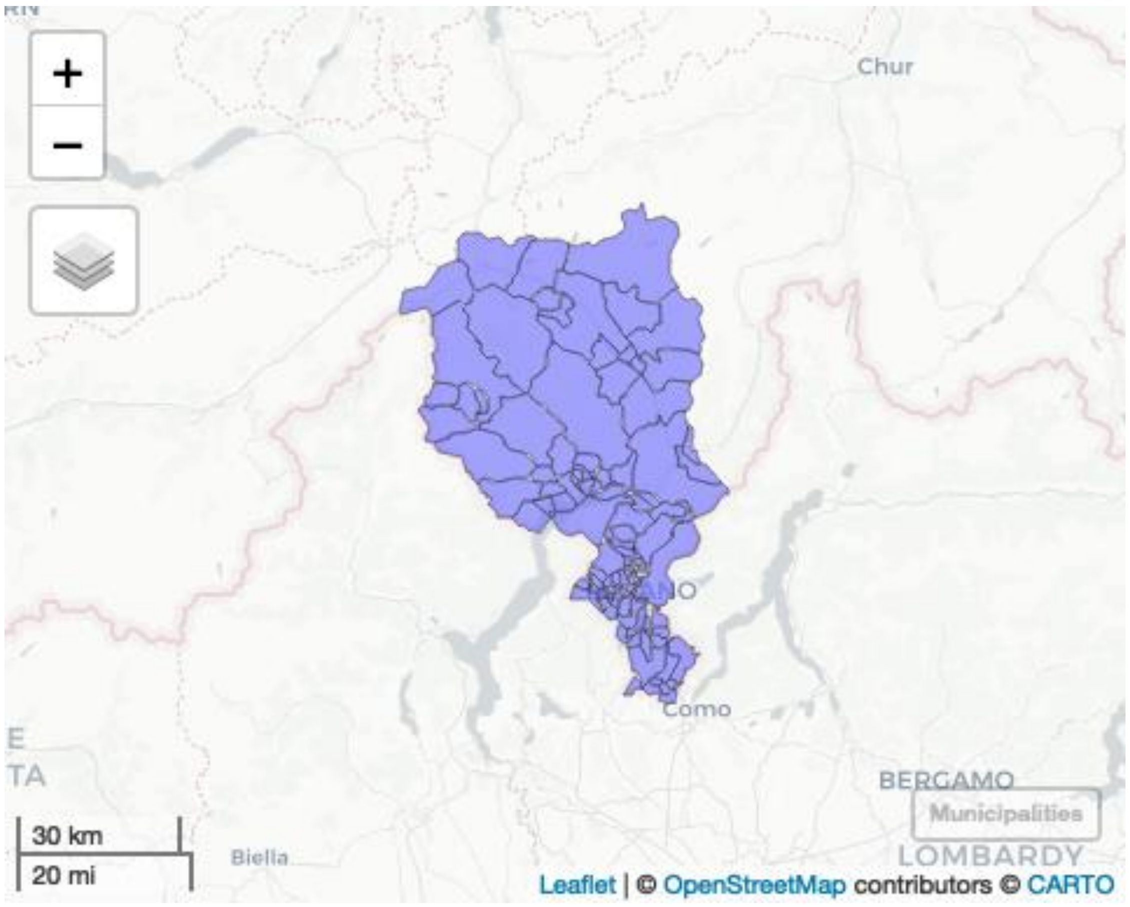
As you can see some data lie outside the boundaries, with the join we will get rid of them.

Select subsample

Before joining I still want to show you how flexible are these data in the sense that the handling of regular dataframe operations is exactly the same as we know it.

For instance, suppose that instead of all Switzerland I am only interested in Ticino. The shape file with municipalities contains for every municipality the number of the canton it belongs to, so, to select the canton Ticino (no 21) I simply select all municipalities with attribute KANTONSNUM==21:

```r
mun_ticino <- mun[mun$KANTONSNUM==21,]

mapview(mun_ticino, layer.name=("Municipalities"), legend = FALSE)
```



Note I do not even need to select the firms in ticino from the firms dataset, the join between all swiss firms and the Ticino's municipalities will simply return all firms that are in the Ticino municipalities.

st_join

To do the Join we use the command st_join:

- first argument has to be the shape
- second the geodata
- third specifies the type of join: here I want an intersection, i.e. all firms that intersect the boundaries or the interior of the shape file, but there are several types of join that you should check out!
- argument left=TRUE means that I want to get in return only the matched rows of the left dataset

```
df <- st_join(mun_ticino, firms_proj, st_join=st_intersect, left= TRUE)
```

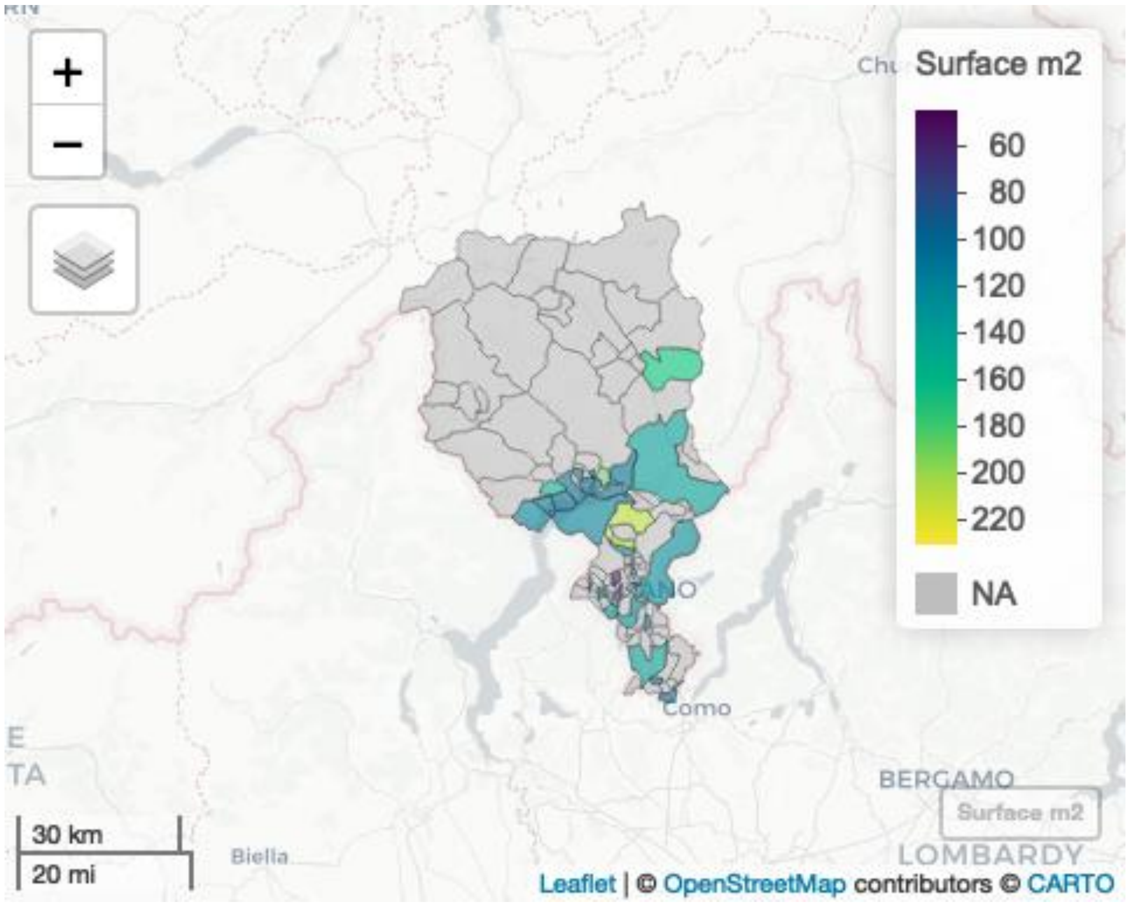
Once the merge is done I can create stats at municipality Level. For example, suppose I have the surface in m2 of each firm, then I can plot average surface of firms by each municipality.

First compute the statistic

```r
# First onvert the surface as numeric
df$surface_m2 <-as.numeric(df$surface_m2)

# Create Stat
sur_m2 <- df %>%
  st_drop_geometry() %>%
  dplyr::select(BFS_NUMMER, KANTONSNUM, surface_m2) %>%
  group_by(KANTONSNUM, BFS_NUMMER) %>%
  summarise(sur_m2 = mean(surface_m2, na.rm=TRUE))

## `summarise()` has grouped output by 'KANTONSNUM'. You can override using the
`.groups` argument.

data_price_mun <- st_as_sf(merge(sur_m2, mun_ticino, by ="BFS_NUMMER"))
```

Then plot it

```r
mapview(data_price_mun, layer.name=("Surface m2"), zcol="sur_m2", legend = TRUE)
```



And we are plotting the information for the municipalities where there are firms. We are done!