Handling Spatial Data in R

Romaike Middendorp

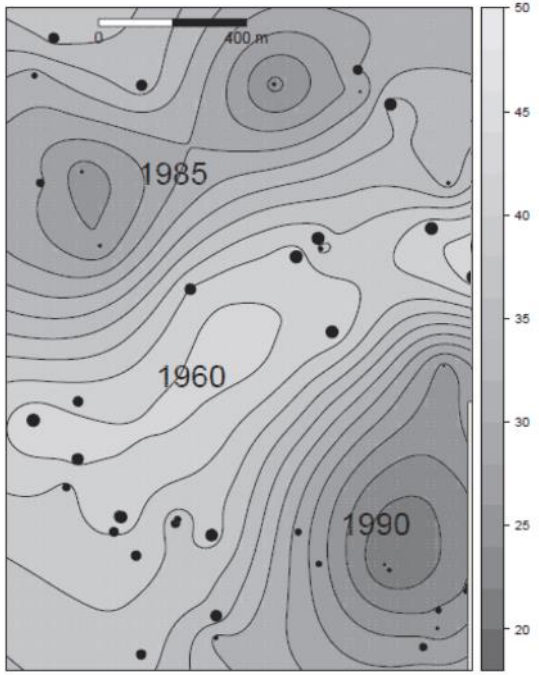
17/02/2020

KUL Datathon

Hello, I’m Romaike Middendorp

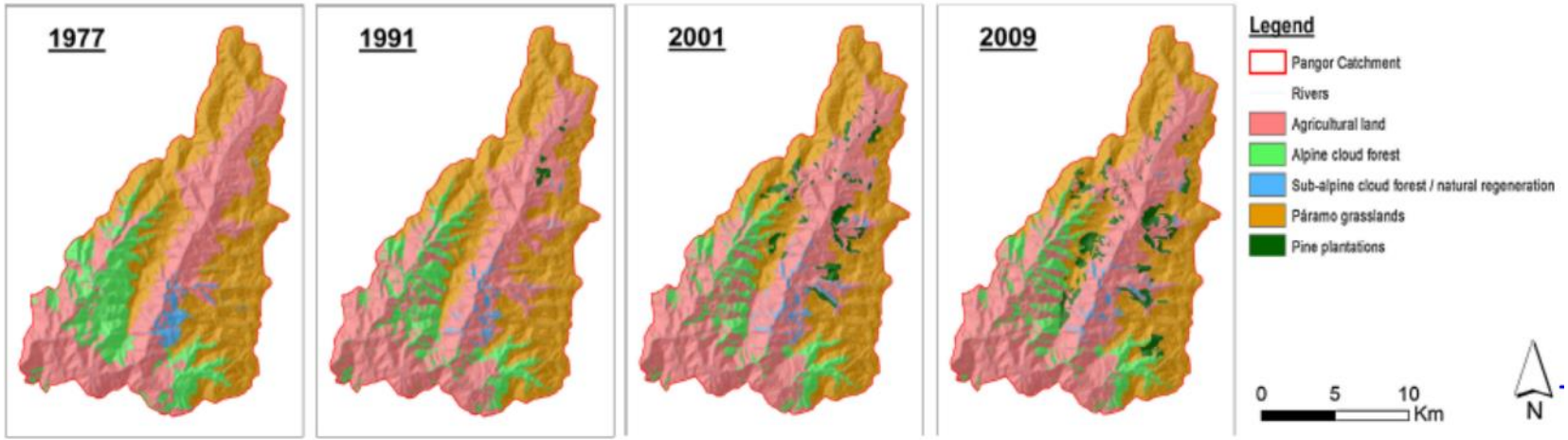
- Data Scientist at XploData
- PhD in Geography & Life Sciences, UCLouvain, BE
- BSc/MSc Biology & Nature Conservation, Utrecht, NL

Spatial data tells a story



Middendorp, et al. 2013 *Biotropica*

Spatial data tells a story



Middendorp, et al. 2016 *J. Landsc. Ecol.*

Spatial data tells a story

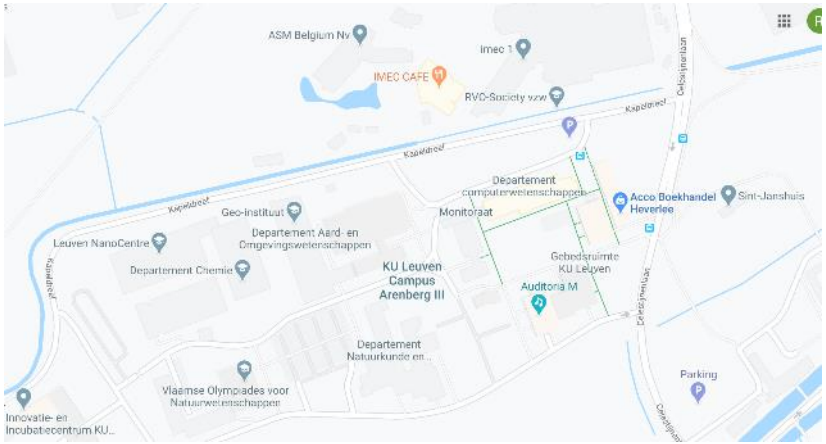
Agenda

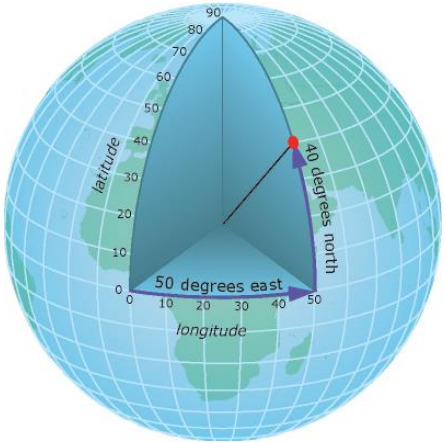
- What are spatial data?
- Spatial data objects
- Vizualizing spatial data
- Spatial operations on vector data
- *Spatio-temporal data*

What are spatial data?

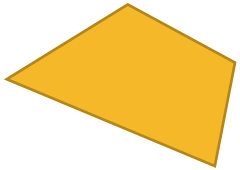
What are spatial data?

- Data are associated with locations
- Any place on Earth is described by coordinates and a **coordinate reference system (CRS)**
  - Latitude/longitude (unprojected, e.g., WGS84)
  - X/Y coordinates (projected, e.g., UTM)





Types of spatial data

- Point 
- Line 
- Polygon 
- *Raster (i.e., gridded)*
  - *Single band*
  - *Multi-band*

1	*2*	*1*
2	*4*	*3*
5	*4*	…

Spatial data objects

Introducing `sp` objects

- Data frames are unpractical to store spatial data:
  - No CRS information
  - Inefficient storage
  - Inefficient display

`sp` package:

- provides classes for storing different types of spatial data
- provides methods for spatial object manipulation
- is useful for point, line and polygon data
- is (still) the standard, so new spatial packages expect data in an `sp` object
- uses S4-type objects

Read in Leuven Air data `SpatialPoints` object

```
> air_stations <- fromJSON("https://data.leuvenair.be/data/LEUVENAIRmeta_final.json")

> rownames(air_stations) <- air_stations$SDS011ID

> coordinates(air_stations) <- ~ LON + LAT

> proj4string(air_stations) <- CRS("+proj=longlat +ellps=WGS84")

> air_stations
class       : SpatialPointsDataFrame
features    : 71
extent      : 4.5982, 4.8329, 50.8507, 50.9477  (xmin, xmax, ymin, ymax)
crs         : +proj=longlat +ellps=WGS84
variables   : 15
names       : SDS011ID, DHTID, EXPORT,
...
```

Introducing `sf` objects

- successor of `sp` package
- stores spatial data as a data frame with geometry column, which is a simple features list column
- allows for object manipulation as if it where a data frame
- `sf::st_read()` reads in vector data (shapefiles, GeoJSON, GPS, etc.)
- `sf::st_as_sf()` to convert data frame to sf object

Read in Telraam data as `sf` object

```
> url <- list(hostname = "telraam-api.net/v0/segments/active",
                scheme = "https",
                query = list(request = "GetFeature",
                outputFormat = "application/json")) %>%
          setattr("class","url")
> request <- build_url(url)
> active <- st_read(request)
> active
Simple feature collection with 806 features and 17 fields
geometry type:  MULTILINESTRING
dimension:      XY
bbox:           xmin: -2.446084 ymin: 46.04589 xmax: 15.45145 ymax: 53.37766
epsg (SRID):    4326
proj4string:    +proj=longlat +datum=WGS84 +no_defs
First 10 features:
…
```

Visualizing spatial data

`tmap` displays spatial data

`tmap`:

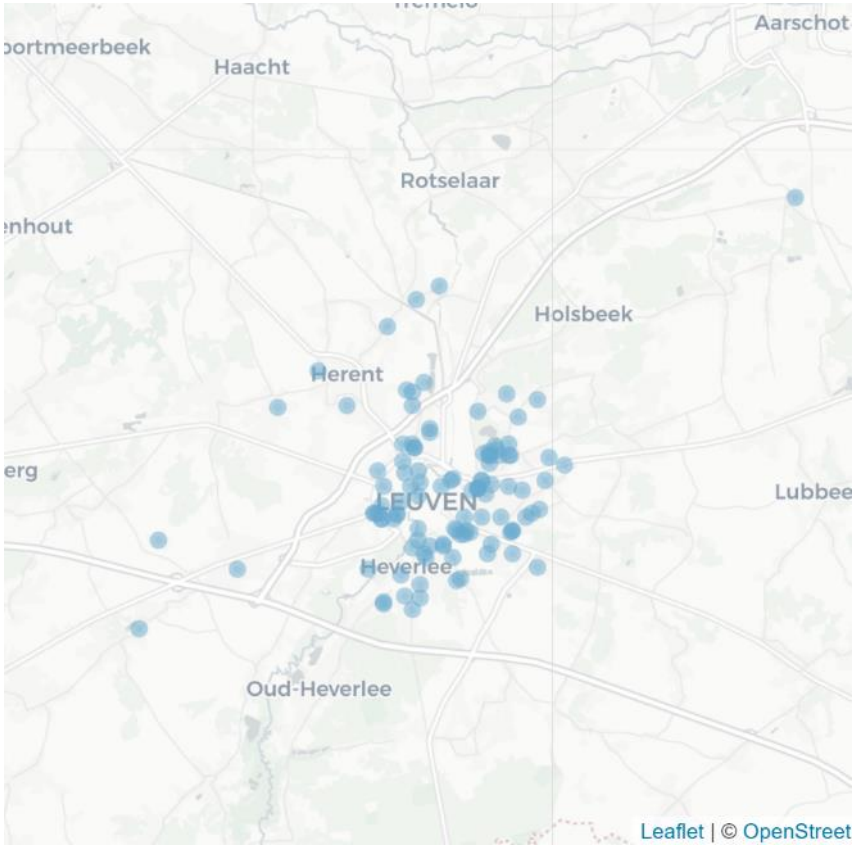
- has a similar philosophy to `ggplot2`; a plot is built up in layers
- expects data in spatial objects
- the `tm_shape` function defines the data for the subsequent layers, you can have many in a single plot
- mapping variables must be “quoted” (unlike in `ggplot2`)

Active Telraam road segments in Leuven: tmap

```
> tmap_mode('view')
> tm_shape(active_leuven) +
    tm_lines(col = "car", scale = 3) +
    tm_shape(leuven)
```

Vizualizing Air quality stations: `leaflet` map

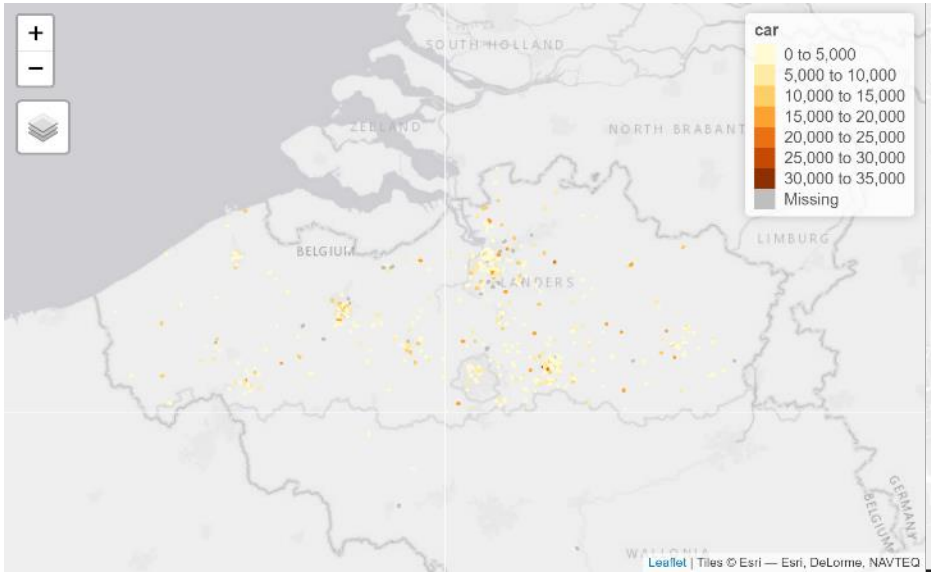
```
> air_stations <-
        fromJSON("https://data.leuvenair.b
        e/data/LEUVENAIRmeta_final.json")

> air_stations <- st_as_sf(air_stations,
        coords = c('LON', 'LAT'))

> leaflet() %>%
        addTiles(group = " CartoDB ") %>%
        addCircleMarkers(stations,
        radius = 2, color = '#67a9cf')
```

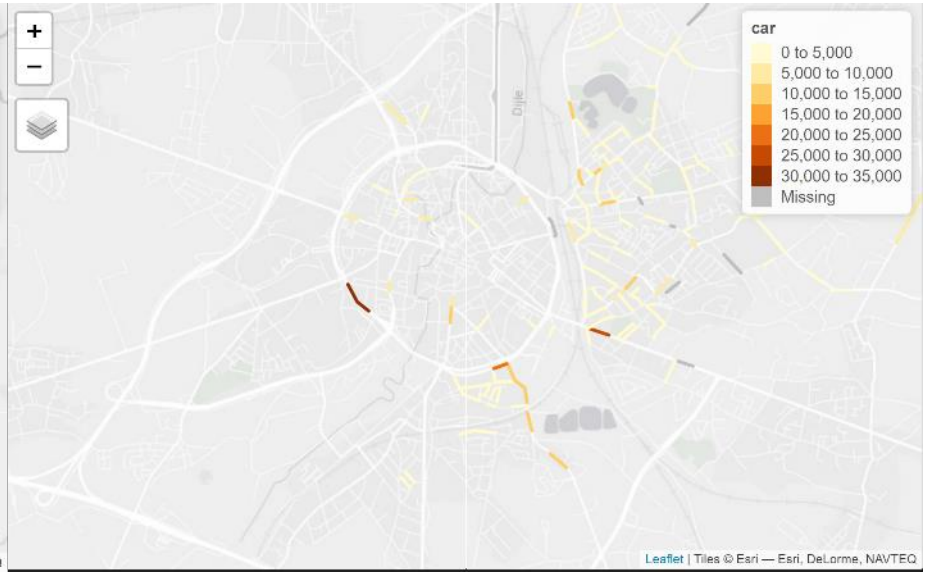


Spatial operations

Intersect: subset road segments in Leuven

```
municipalities <- readRDS("data/gadm36_BEL_4_sf.rds")
leuven <- municipalities %>% filter(NAME_4 == "Leuven")
active_leuven <- active_sf[leuven, op = st_intersects]
```





Closest neighbor: points to lines

```
> air_stations_sp <- as(air_stations, "Spatial")
> active_leuven_sp <- as(active_leuven, "Spatial")
> library(geosphere)
> dist_mat <- dist2Line(p = air_stations_sp, line = active_leuven_sp)
> dist_df <- data.frame(dist_mat)
> dist_df$station_id <- air_stations$SDS011ID
> active_air <- merge(dist_df, active_leuven, by.x = "ID", by.y = "row.names")
> head(active_air)
   distance      lon      lat station_id  segment_id
 1855.30334 4.685237 50.91683       9753     311302
   13.07523 4.687143 50.91866      13649     311302
   15.57644 4.710441 50.88419      22953     347179
   93.54251 4.707756 50.86897       9613     347345
   17.93855 4.711442 50.87295       8813     347365
…
```

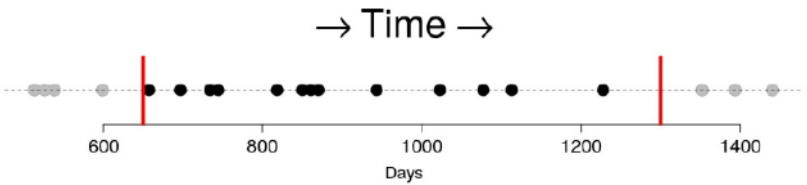
Spatial joins: air and weather stations

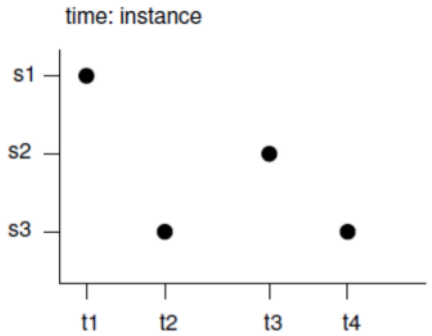
```
> library(nngeo
> nn_stations_join <- st_join(air_stations, meteo_stations, st_nn, k = 1)
```

Spatio-temporal data

Introducting `spacetime` package

- Space, time and data component

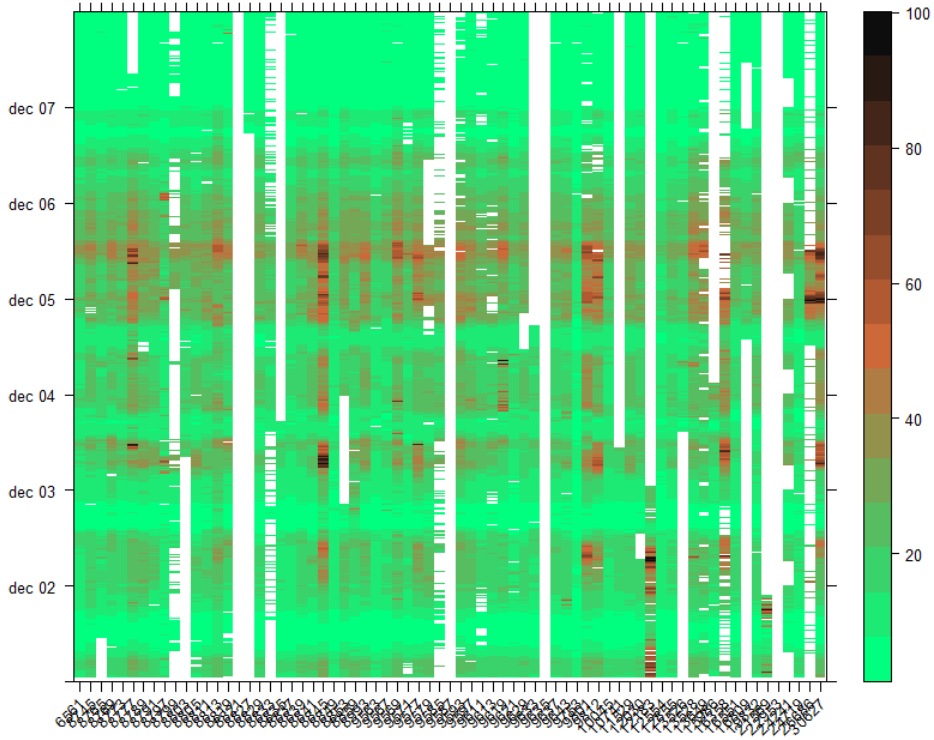




Source: Pebesma, 2012

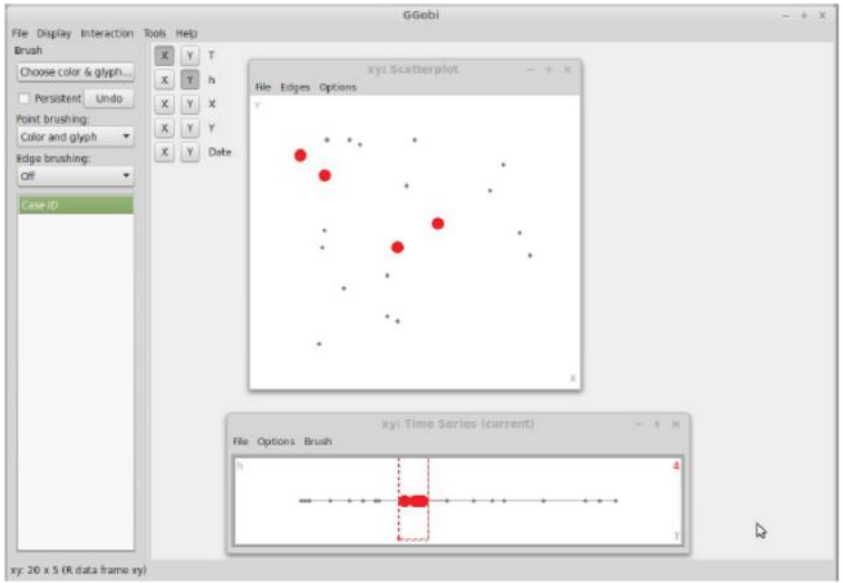
PM2.5 measurements for first week of December

```
> library(reshape2)

> library(zoo)

> library(spacetime)


> air <- read.csv('LEUVENAIRfulldump2019.csv)

> air <- air %>%
  mutate(DATEUTC = ymd_hms(DATEUTC, tz = "UTC")) %>%
  filter(month(DATEUTC) == 12 & day(DATEUTC) %in% c(1:7))

> PMwide <- dcast(air [, c('SDS011ID', 'PM2.5', 'DATEUTC')],
                  DATEUTC ~ SDS011ID, value.var = "PM2.5")

> PMzoo <- zoo(PMwide[,-1], PMwide$DATEUTC)

> dats <- data.frame(vals = as.vector(t(PMzoo)))

> PMst <- STFDF(sp = air_stations,
                time = index(PM2.5zoo_med),
                data = dats)

> stplot(PMst, mode = 'xt',
         scales = list(x = list(rot = 45)))
```

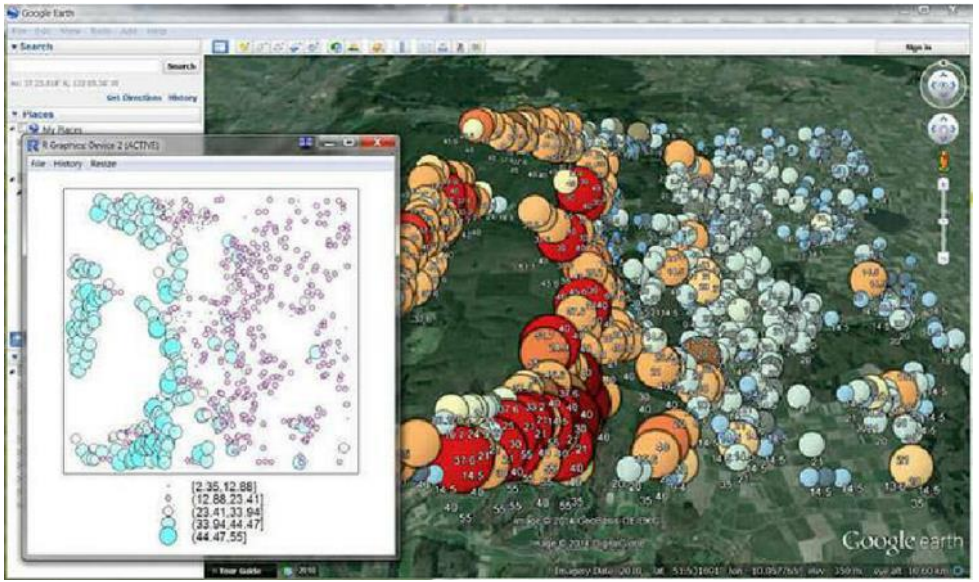


Other packages for spatiotemporal visualizations

rggobi



plotKML



info@xplodata.be

www.xplodata.be

Romaike Middendorp