Spatial data in R

Konstantin A. Kholodilin

DIW Berlin and NRU-HSE

R user group DIW
September 21, 2017

Introduction

- Input of spatial data
  - vectors and rasters.
- Transformation
  - assigning attributes;
  - transforming data;
  - merging spatial polygons.
- Representation of spatial data.
- Sources of spatial data.

Types of spatial data

- GIS data = standardized data format of geographic information systems, which are used to collect, process, organize, analyze, and represent geographical data.
- Geographical data can be complemented by the data of interest (attributes).
- Types of GIS (spatial) data formats:
  - rasters = grid of identical, regularly spaced pixels.
  - vectors = polygons based on control points, or nodes.

Input of spatial data

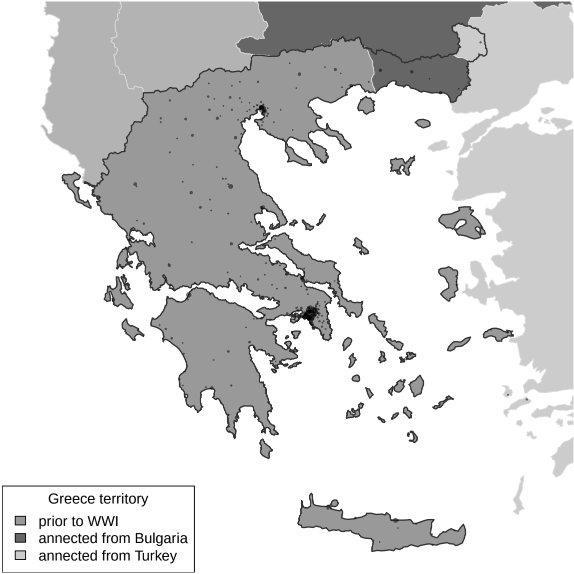
- Shapefile = geospatial vector data format for GIS software created by the ESRI (Environmental Systems Research Institute);
  - www.esri.com/library/whitepapers/pdfs/shapefile.pdf
- Structure of shapefiles:
  - coordinates:
    - points (shops, cities);
    - lines (roads);
    - polygons (borders of regions, islands).
  - attributes as data.frame.
    - population;
    - sales;
    - customer evaluation.

Input of spatial data

- Typically, shapefile consists of several files:
  - *.shp = main file (shape type, coordinates, bounding box);
  - *.shx = index file;
  - *.dbf = dBase file with non-geographical data;
  - *.prj = projection.
- Other formats of shapefiles:
  - kml, dxf, xyz, gml, json, tab, ntf.

Pieces of water in St. Petersburg: polygons

Greek cities: polygons and points



Tram network in St. Petersburg, 1902: polygons, lines, and points

Input of spatial data

- Function *readOGR* of package *rgdal*.

Listing 1:

```
library{rgdal}
sInFile1 = "d:/KKholodilin/SHP/Europe.kml"
ogrListLayers(sInFile1) # List of layers
Map = readOGR(sInFile1, layer = "Italy, 1925")
```

- Functions *readShapePoints*, *readShapeLines*, and *readShapePoly* of package *maptools*.

Listing 2:

```
library{maptools}
sInFile2 = "d:/KKholodilin/SHP/Petrograd.shp"
Map = readShapePoly(sInFile2)
```

Input of spatial data

- Sometimes, projection must be changed:

Listing 3:

```
library{maptools}
sInFile2 = "d:/KKholodilin/SHP/Petrograd.shp"
Map = readShapePoly(sInFile2)
Map = spTransform(Map,
 CRS("+proj=longlat +datum=WGS84"))
```

Projection

Original projection

WGS84 projection

Input of spatial data

- Projection must be changed also, if the data have different projections.
- In Germany, Soldner coordinates are very widespread.
- Other data providers (e.g., Google or Eurostat) work with ETRS89 and WGS84 coordinate systems.
- Transformation of Soldner to WGS84:

Listing 4:

```
sInFile = "d:/KKholodilin/SHP/Berlin_Soldner.shp"
Map = readShapePoly(sInFile)
proj4string(Map) = CRS("+proj=cass
 +lat_0=52.41864827777778 +lon_0=13.62720366666667
 +x_0=40000 +y_0=10000 +ellps=bessel
 +datum=potsdam +units=m +no_defs")
Map = spTransform(Map,
 CRS("+proj=longlat +datum=WGS84"))
```

Transformation of spatial data: additional info

- Shapefiles are rather useless without attributes.
- Attributes in shapefile are organized as data.frame.
- So, additional information can be fed in by merging:

Listing 5:

```
Map = readShapePoly(paste(sFolder_SHP, sInFile_SHP, sep=""))
    # Load shapefile

proj4string(Map) <- CRS("+proj=longlat +datum=WGS84")

X = read.xlsx(paste(sFolder, sInFile, sep=""), sheet="Data")
    # Load data of interest

Map@data = merge(Map@data, X, by.x="District", by.y="
   District", sort=F) # Merge shapefile with data
```

Transformation of spatial data: population density

- Areas of spatial polygons can be computed using package *geosphere*.
- If we have population figures, population density can be computed.

Listing 6:

```
library(geosphere)
Map$Area = areaPolygon(Map) # Area in square meters
Map$Pop_density = Map$Pop / Map$Area
```

- Centroids of polygons can be found using package *rgeos*:

Listing 7:

```
Centr = gCentroid(Map, byid=T)
Map$Cent_X = Centr@coords[,1]
Map$Cent_Y = Centr@coords[,2]
```

Transformation of spatial data: population density

- Attributes (e.g., population density) can be plotted using different colors to obtain chloropleth map.

Listing 8:

```
NBreak = 7
vBreak = pretty(Map$Pop_density, n=NBreak)
svCol = gray.colors(NBreak, start = 0.95, end = 0, gamma=1)
Map$Interv = cut(Map$Pop_density, breaks=vBreak, dig.lab=4)
svInterv = levels(Map$Interv)

Min = min(Map$Pop_density, na.rm=T)
Max = max(Map$Pop_density, na.rm=T)
Map$Col = (Map$Pop_density - (Min + Max)/2) / (Max - Min)
Map$Col = Map$Col * 0.95 + 0.95/2
Map$Col = round(100-100*Map$Col)
Map$Col = paste("gray", Map$Col, sep="")
```

Transformation of spatial data: population density

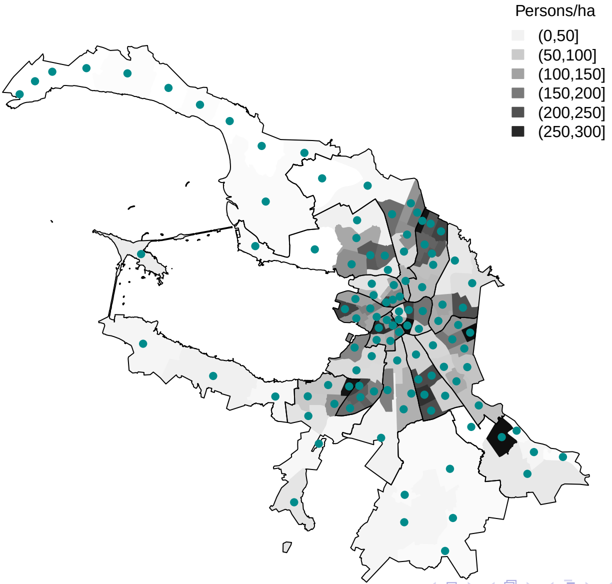
- The chloropleth map can be plotted in a (traditional) way:

Listing 9:

```
par(mar=c(1,1,1,1))
plot(Map, col=Map$Col, border=Map$Col) # Population density
    by subdistricts

par(new=T)
plot(Map_dis, bg="transparent") # Borders of districts
sTitle = "Persons/ha"
legend("topright", legend=svInterv, fill=svCol, border=svCol
    , bty="n", title=sTitle, cex=1)
points(Map$Cent_X, Map$Cent_Y, pch=19, col="cyan4") #
    Centroids of subdistricts
```

- Alternatively, it can be done using function *geom_map* of package *ggplot2*.

Chloropleth map: population density in St. Petersburg



Transformation of spatial data: union of polygons

- Several regions can be merged using the function *unionSpatialPolygons* from *rgeos* package.

Listing 10:

```
library(rgeos)
svAlt_Berlin = c("Mitte", "Tiergarten", "Wedding", "
    Prenzlauer Berg",
                 "Friedrichshain", "Kreuzberg")

Map$ID = 0 # Create common IDs for polygons to merge
Map$ID[vSel_ab] = 1

  if(rgeosStatus())
  {
Map_center_periphery = unionSpatialPolygons(Map, Map$ID) #
    Merging polygons by IDs
  }
```

Merging spatial polygons

Combination of vector and raster images: OSM

Listing 11:

```
library(OpenStreetMap)
Map_OSM = spTransform(Map, osm())

BBox = bbox(Map)

ul = c(BBox[2,2], BBox[1,1])
lr = c(BBox[2,1], BBox[1,2])

url <- "https://a.tiles.mapbox.com/v4/mapquest.streets-mb/{z
    }/{x}/{y}.png?access_token=pk.
    eyJ1IjoibWFwcXVlc3QiLCJhIjoiY2Q2N2RlMmNhY2NiZTRkMzlmZjJmZDk
    .mPRiEubbajc6a5y9ISgydg"
map <- openmap(ul,lr, minNumTiles=20, type="osm")
map_longlat <- openproj(map, projection = "+proj=longlat +
    datum=WGS84")
```

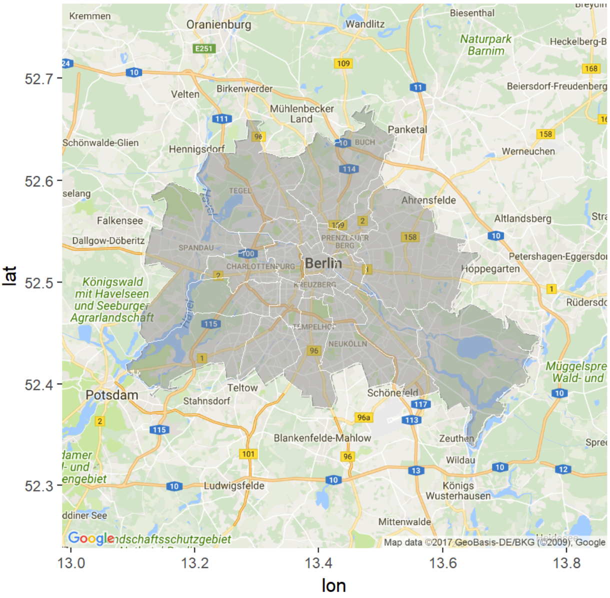
Combine shapefile with OSM

Combination of vector and raster images: Google Maps

Listing 12:

```
library(ggmap)
BBox = bbox(Map)

#--- Download the Google Maps city plan of Berlin
map <- get_map(location = c(lon = mean(BBox[1,]), lat = mean
    (BBox[2,])), zoom = 10,
                  maptype = "roadmap", source = "google")

data <- fortify(Map) # Conversion of an object to data.frame

png(paste(sFolder, sOutFile, sep=""), res=200, width=1000,
    height=1000)

ggmap(map) +
  geom_polygon(aes(x = long, y = lat, group = group), data =
       data, colour = 'white', fill = 'gray50', alpha = .4,
      size = .3)

dev.off()
```

Combine shapefile with Google Maps



Sources of spatial data

- Berlin's maps — FIS broker:
  - http://www.stadtentwicklung.berlin.de/geoinformation/fis-broker/
- EU NUTS regions — Eurostat:
  - http://ec.europa.eu/eurostat/de/web/gisco/geodata/reference-data/administrative-units-statistical-units/nuts
- OpenStreetMap — Geofabrik:
  - http://download.geofabrik.de/