R and geodata

Andreja Radović, PhD

4th of October 2017

What is R?

- a programming **language** and **environment** for statistical analysis and visualisation
- open-source == large community
- great number of contributed packages (diverse disciplines)

Loading

CRAN

https://cran.r-project.org/ (https://cran.r-project.org/)

https://cran.r-project.org/web/packages/available_packages_by_name.html (https://cran.r-project.org/web/packages/available_packages_by_name.html)

https://CRAN.R-project.org/view=Spatial (https://cran.r-project.org/view=Spatial)

https://cran.r-project.org/web/views/SpatioTemporal.html (https://cran.r-project.org/web/views/SpatioTemporal.html)

CRAN "Spatial" - three general types of packages

Packages specialised for:

1. accessing geodata and other software functions, interoperability
2. visualisation of geodata
3. analysis of geodata

1) Packages for access and interoperability

- Geospatial Data Abstraction Library (GDAL): *rgdal*
- GRASS GIS: *spgrass6*
- SAGA GIS: *RSAGA*
- Access, PostgreSQL/PostGIS: *RODBC*, *rpostgis*, *postGIStools*
- …

2) Packages for geodata visualisation

- rasterVis
- maps
- plotKML
- plotGoogleMaps
- …
- (also, base, lattice and ggplot general packages)

Much more on GitHub

2) Packages for analysis

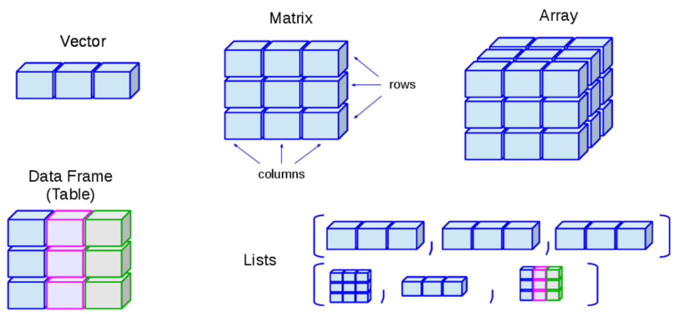
- point patterns: spatial, spatstat
- geostatistics (2D/3D), likelihood approach, Bayes methods: *geoR*, *gstat*, *RandomFields*
- 2D carthesian geometry operations: *Rgeos*
- circular, directional statistics: *circular*
- cluster's detection: *Dcluster*
- analytics on greeded data: *raster*
- geometry on the sphere, great circle analyses: *geosphere*
- remotely sensed data: *remotesensing*
- …

Basic data types in R

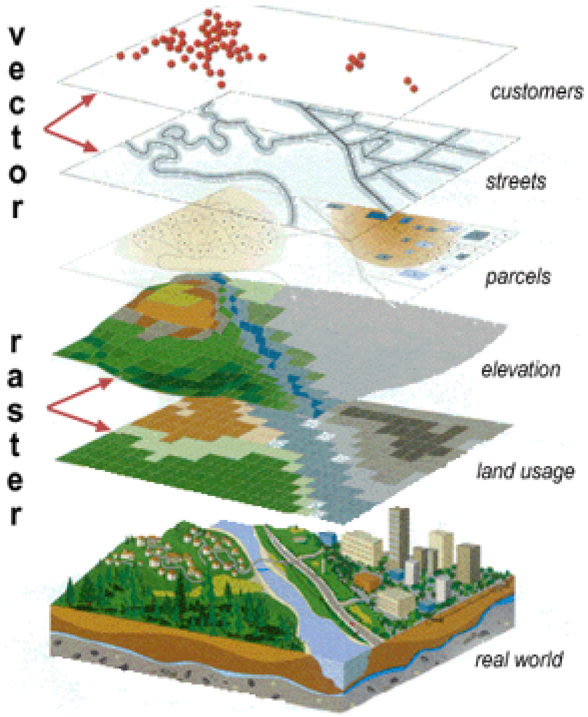
N dimensions	Homogenious	Heterogenious
1	Atomic Vector	List
2	Matrix	Data Frame

N dimensions	Homogenious	Heterogenious
N	Array	

R, object oriented language



But we need a model of the real world



points, lines, polygons

“Spatial” in R, short history

- pre-2003, sporadic
- 2003: workshop at DSC
- 2003: start of *r-sig-geo*
- 2003: *rgdal* on CRAN
- 2005: *sp* on CRAN; *sp* support in *rgdal*

“Spatial” in R, short history 2

- 2008: *Applied Spatial Data Analysis with R*

- 2011: *rgeos* on CRAN
- 2013: second edition of Applied Spatial Data Analysis with R
- 2016-7: simple features for R (finished?)
- 2017-8: spatiotemporal tidy arrays for R (design phase)

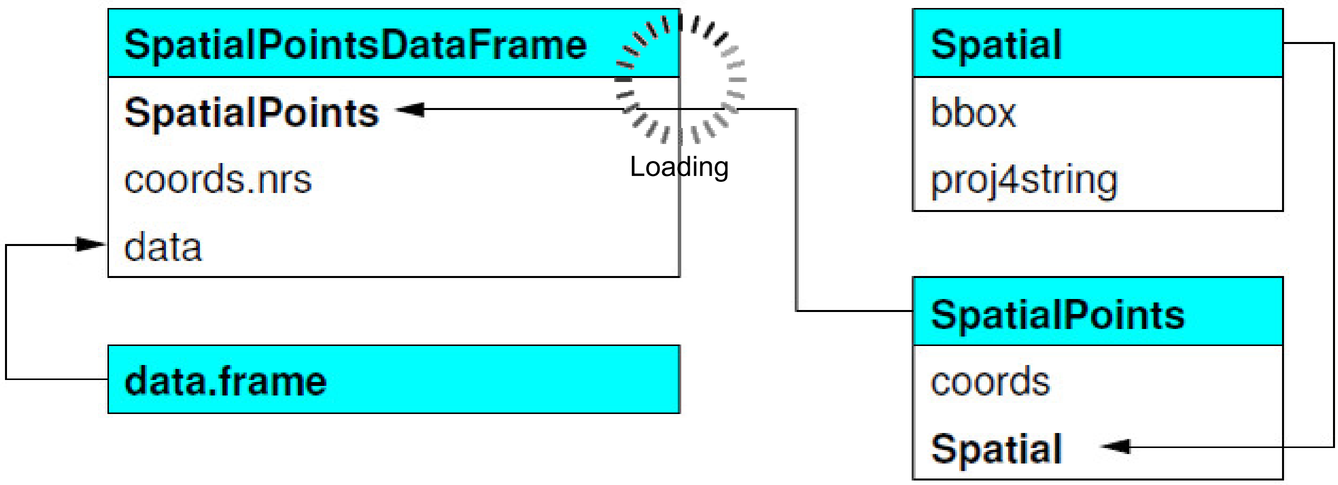
Package ‘sp’

First package for classes and methods for spatial data in R

- classes and methods for geo data in R/S language
- does not provide any analytical methods
- S4 data class
- generic methods for **points**, **lines**, **polygons** and **grids/surfaces**

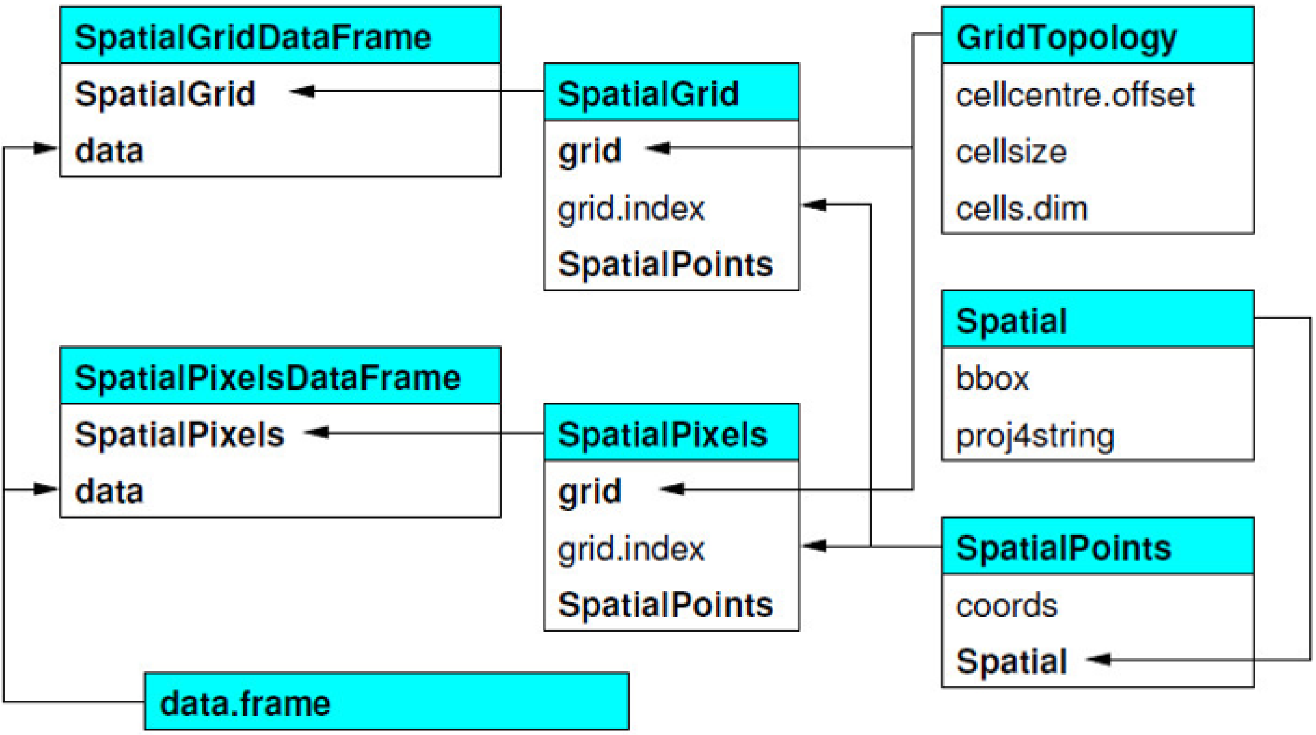
Clases in ‘sp’ - vector, example for points

Data structure for points



Clases in ‘sp’ - rasters/grids

Data structure for grids



General structure Spatial*DataFrames

- @data - data.frame, attributes
- @coords.nrs - variables in data.frame with coordinate info
- @coords - coordinates
- @bbox - spatial extent
- @proj4string - spatial reference

Loading

Methods for ‘sp’ classes

- dimensions(x)
- spTransform(x, CRS)
- bbox(x)
- coordinates(x)
- gridded(x)
- spplot(x)
- over(x, y)
- spsample(x)
- geometry(x)

Simple features - CRAN

Package ‘sf’

JAN
03

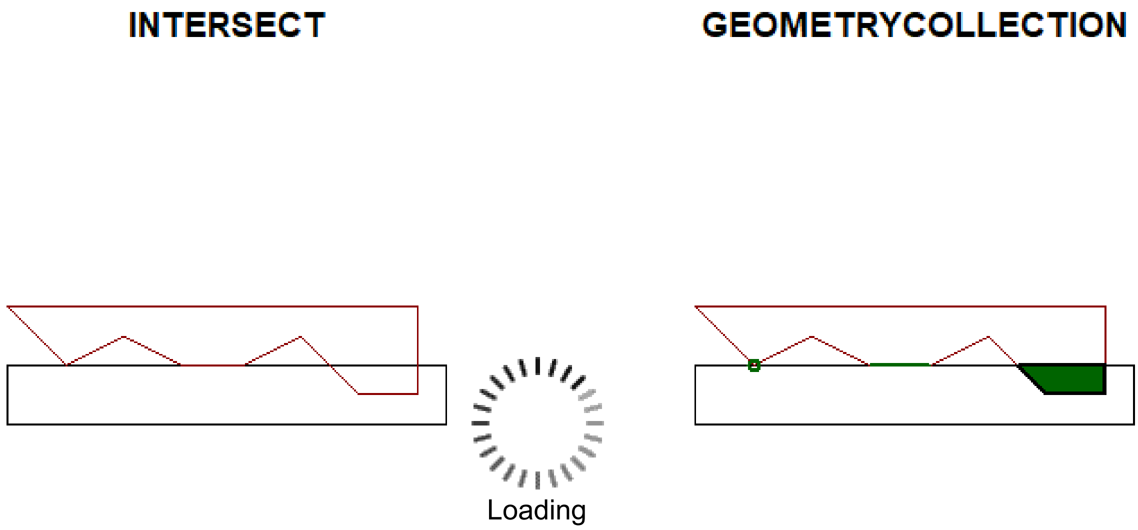
Simple Features Now on CRAN

By Brett Preston | Blog, R Consortium Project, R Language

by Edzer Pebesma

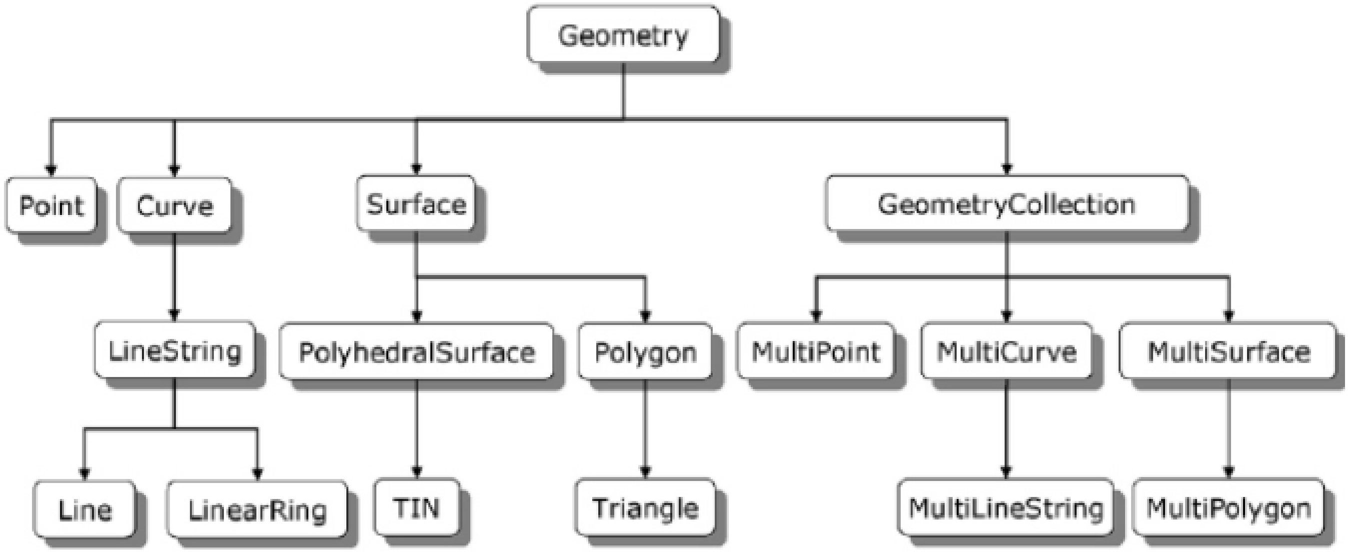
Why ‘sp’ was not enough?

But still no support for gridded data!



OGC/ISO - simple features

Simple feature hierarchy



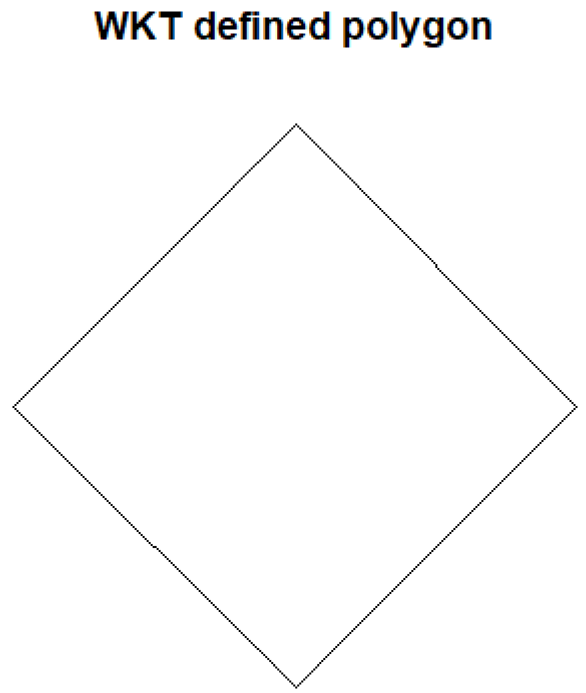
Simple features?

- ISO standard that is widely adopted
- used in spatial databases
- *feature*: abstraction of real world phenomena
- *simple feature*: feature with all geometric attributes described piecewise by straight line or planar interpolation between sets of points
- 7 + 10 types, 68 classes, 7 classes ok for 99% of the use cases
- text and binary serialisations (WKT, WKB)
- support for mixed type (`GEOMETRYCOLLECTION`), and type mix (`GEOMETRY`)
- support for empty geometry

Before ‘sf’ package

Simple feature objects automatically imported as ‘sp’ object using package *rgeos*

```
library(rgeos)
my_poly <- readWKT("POLYGON((1 0,0 1,1 2,2 1,1 0))")
plot(my_poly, main="WKT defined polygon")
```



Start preparing spatial objects from…

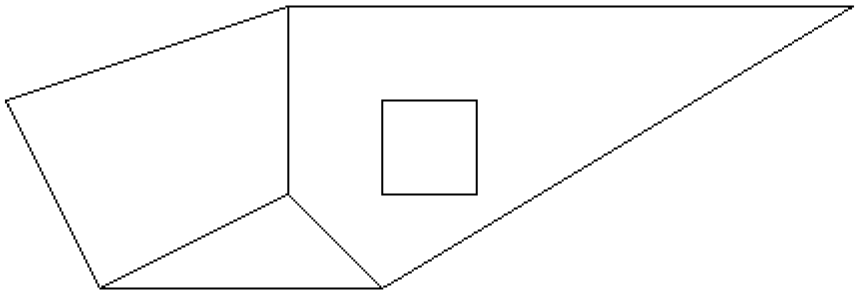
Not usual way to prepare geodata this way

```
#polygons from set of coordinates
Sr1 <- Polygon(cbind(c(2, 4, 4, 1, 2), c(2, 3, 5, 4, 2)))
Sr2 <- Polygon(cbind(c(5, 4, 2, 5), c(2, 3, 2, 2)))
Sr3 <- Polygon(cbind(c(4, 4, 5, 10, 4), c(5, 3, 2, 5, 5)))
Sr4 <- Polygon(cbind(c(5, 6, 6, 5, 5), c(4, 4, 3, 3, 4)), hole = TRUE)
Srs1 <- Polygons(list(Sr1), "s1")
Srs2 <- Polygons(list(Sr2), "s2")
#several polygons into list
Srs3 <- Polygons(list(Sr4, Sr3), "s3/4")
#create spatial polygons, still no CRS/SRID
SpP <- SpatialPolygons(list(Srs1, Srs2, Srs3), 1:3)
```

Inspect the data…

Stil necessary to attach *Spatial* information (CRS/SRID) and optionally attributes *data.frame*

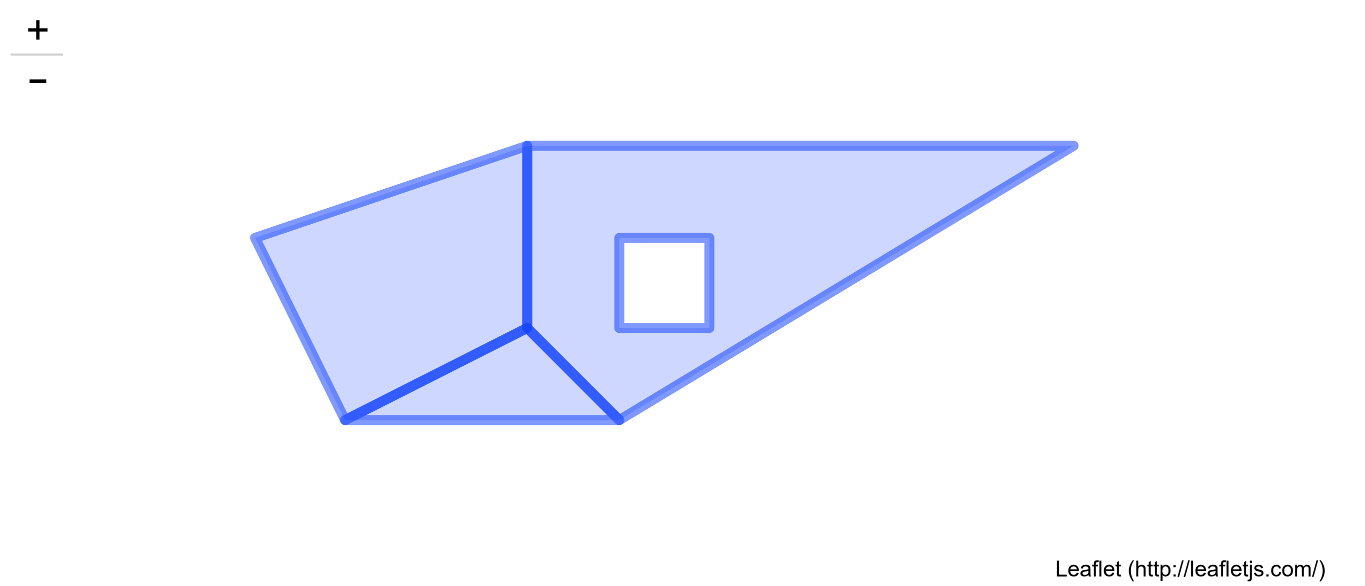
```
plot(SpP)
```



Interactive visualisations with Leaflet

… still no CRS/SRID but we can visualise it in different ways

```
n <-leaflet()
leaflet(height = "300px") %>% addPolygons(data = SpP)
```



More usually, import the geodata

Import via GDAL/OGR library

```
HRV_adm2 <- readOGR(getwd(),"HRV_adm2", verbose = TRUE, stringsAsFactors=FALSE, encoding="UTF
-8")
```

```
OGR data source with driver: ESRI Shapefile
Source: "D:/SRCE_razno/meetup", layer: "HRV_adm2"
with 560 features
It has 11 fields
```

```
#take a look from how many level 1 administrative division
levels(as.factor(HRV_adm2@data$ID_1))
```

```
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
[15] "15" "16" "17" "18" "19" "20" "21"
```

Manipulate geodata

Diverse functions from Geos library

Access functionality of other GIS software

Result exported via Rgdal (GDAL/OGR library)

Manipulate geodata


```
library(maptools);
head(HRV_adm2@data[1:6], n=2)
```

```
  ID_0 ISO  NAME_0 ID_1             NAME_1 ID_2
0   57 HRV Croatia    8 Krapinsko-Zagorska  180
1   57 HRV Croatia    8 Krapinsko-Zagorska  181
```

```
#levels of variable, ID for county
levels(as.factor(HRV_adm2@data$ID_1))
```

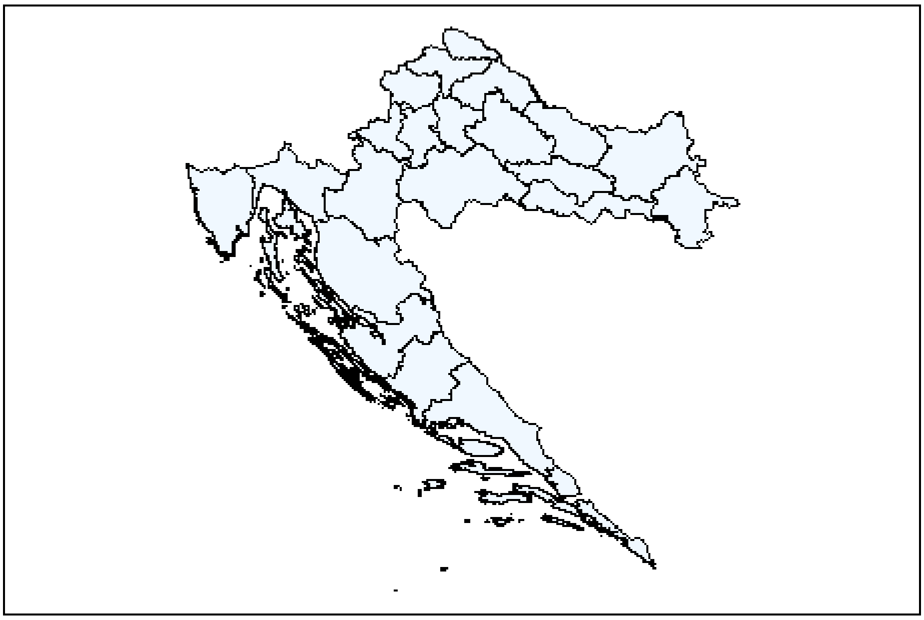
```
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
[15] "15" "16" "17" "18" "19" "20" "21"
```

```
#make union of polygons, union by county ID
HRV_adm2_union <-  unionSpatialPolygons(HRV_adm2, HRV_adm2@data$ID_1)
```

Visualise the result

```
plot(HRV_adm2_union, col="aliceblue", main="Union of polygons")
box(lty = 'solid')
```

Union of polygons



Loading

Integrate data

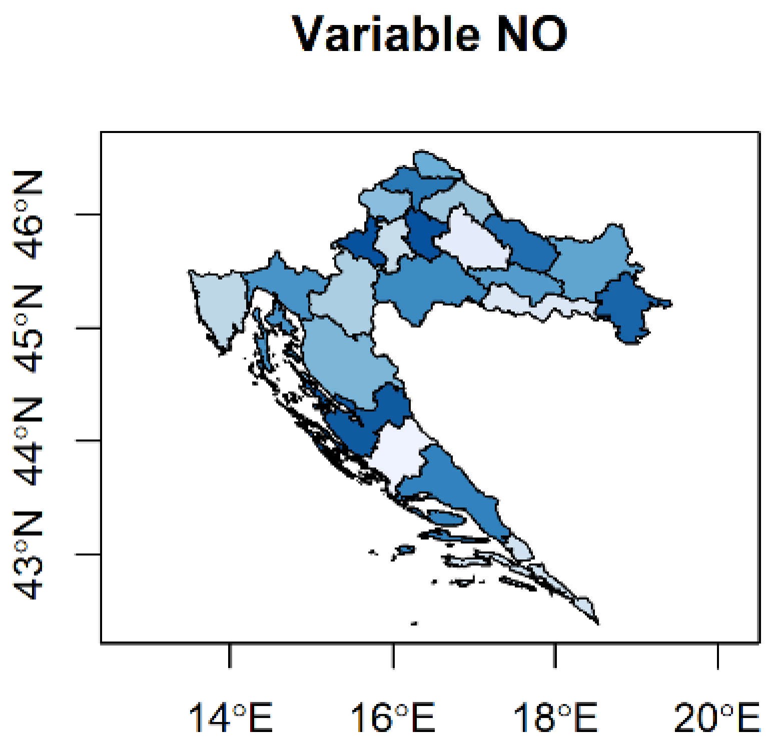
Attach data from other sources, e.g. Bureau of Statistics

```
HRV_adm1 <- readOGR("D:\\podloge\\GADM\\Croatia\\HRV_adm_shp","HRV_adm1", encoding="Windows-1
250",verbose = F, stringsAsFactors=FALSE)

data <- read.xlsx("county_fake_data.xlsx",
                  as.data.frame=T, stringsAsFactors = T,
                  sheetIndex=1, encoding="UTF-8" )
#merge by spatial polygons
merged_data <- merge(HRV_adm1, data, by.x="ID_1", by.y="ID_COUNTY", all.x=T)
```

Visualise the result

```
plot(merged_data[13], col=colorRampPalette(brewer.pal(5,"Blues"))(21),
      axes=T, main="Variable NO")
```



Example, select Lobor and Oroslavje

Loading

```
head(HRV_adm2@data[1:4], n=2)
```

```
  ID_0 ISO  NAME_0 ID_1
0   57 HRV Croatia    8
1   57 HRV Croatia    8
```

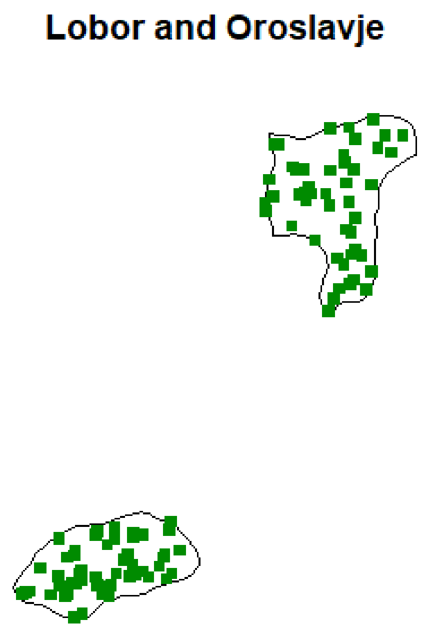
```
select <- HRV_adm2[HRV_adm2@data$NAME_2=="Lobor" | HRV_adm2@data$NAME_2=="Oroslavje",]
class(select)
```

```
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

```
#prepare random sample in selection
p_100 <- spsample(select,100, "random")
```

Visualise the result

```
plot(select, main="Lobor and Oroslavje")
points(p_100, pch=15, cex=0.9, col="green4")
```



Export some elements

Export selected polygons as Esri shapefile, KML, PostgreSQL/PostGIS…

```
writeOGR (select, ".", "Lobor_Orosavje", driver="ESRI Shapefile", overwrite=T)
writeOGR (select, "Lobor_Orosavje.kml", ".", driver="KML", overwrite=T)
writeOGR(select, "PG:dbname=GEO_meetup", layer_options = "geometry_name=geom", "states", "Pos
tgreSQL")
```

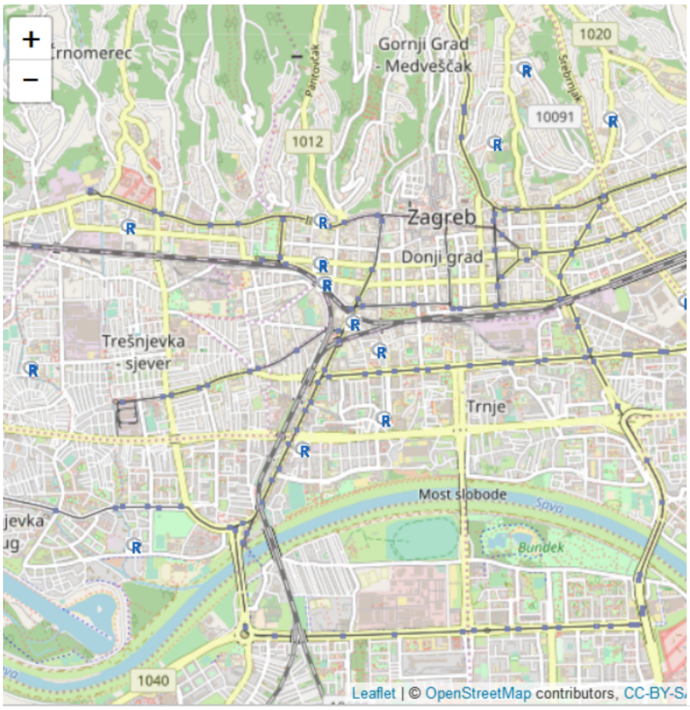
Example: additional customisation for Leaflet

```
HRV_adm0 <- readOGR("D:\\podloge\\GADM\\Croatia\\HRV_adm_shp","HRV_adm0", verbose = F, stringsAsFactors=FALSE, encoding="UTF-8")
HRV_adm1 <- readOGR("D:\\podloge\\GADM\\Croatia\\HRV_adm_shp","HRV_adm1", verbose = F, stringsAsFactors=FALSE, encoding="UTF-8")
HRV_adm2 <- readOGR("D:\\podloge\\GADM\\Croatia\\HRV_adm_shp","HRV_adm2", verbose = F, stringsAsFactors=FALSE, encoding="UTF-8")
#create icon
rest_icon <- makeIcon(
  iconUrl = "R_logo.png",
  iconWidth = 12, iconHeight = 12,
  iconAnchorX = 15, iconAnchorY = 10)
#make our own pallette
colors<- colorRampPalette(brewer.pal(5,"Blues"))(21)
#points to visualise
load("df.RData")
#make it spatial
coordinates(df) <- ~ lng+lat
#what is CRS?
proj4string(df) <- CRS("+proj=longlat +datum=WGS84") #EPSG:4326
```

Our data in Leaflet

```
m <- leaflet()  %>% addTiles()
m<- m %>% addPolygons(data=HRV_adm1, fillColor = colors, stroke = F, fillOpacity = 0.5,weight=3)
m<- m %>% addPolygons(data=HRV_adm0, fill=F, color="#f93", stroke = T, fillOpacity = 0.5,weight=1)
m<- m %>% addPolygons(data=HRV_adm2, fill=F, color="gray", stroke = T, fillOpacity = 0.5,weight=1)
m<- m %>% addMarkers(coordinates(df)[,1],coordinates(df)[,2],  icon = rest_icon)
m
library(htmlwidgets)
saveWidget(m, file="m.html")
```

Use capture from Leaflet for dynamic documents from RMarkdown



Gridded data in ‘sp’

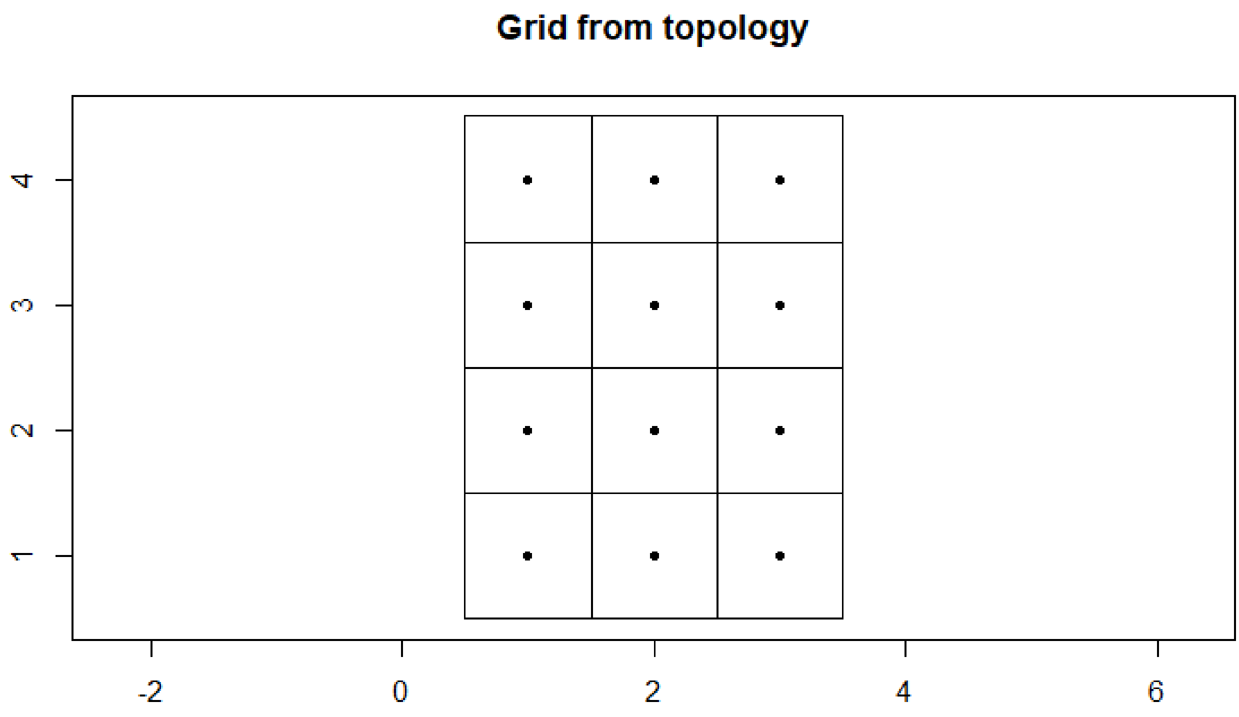
Again, we can create objects, here start from grid topology definition

```
topology <- GridTopology(cellcentre.offset = c(1,1,2), cellsize=c(1,1,1), cells.dim =
c(3,4,6)) #cellcentre.offset  numeric; vector with min coordinates of cell centre in each dim
sp_grid <- SpatialGrid(topology)
summary(sp_grid)
```

Loading

```
 Length       Class        Mode
     72 SpatialGrid          S4
```

What we have made so far

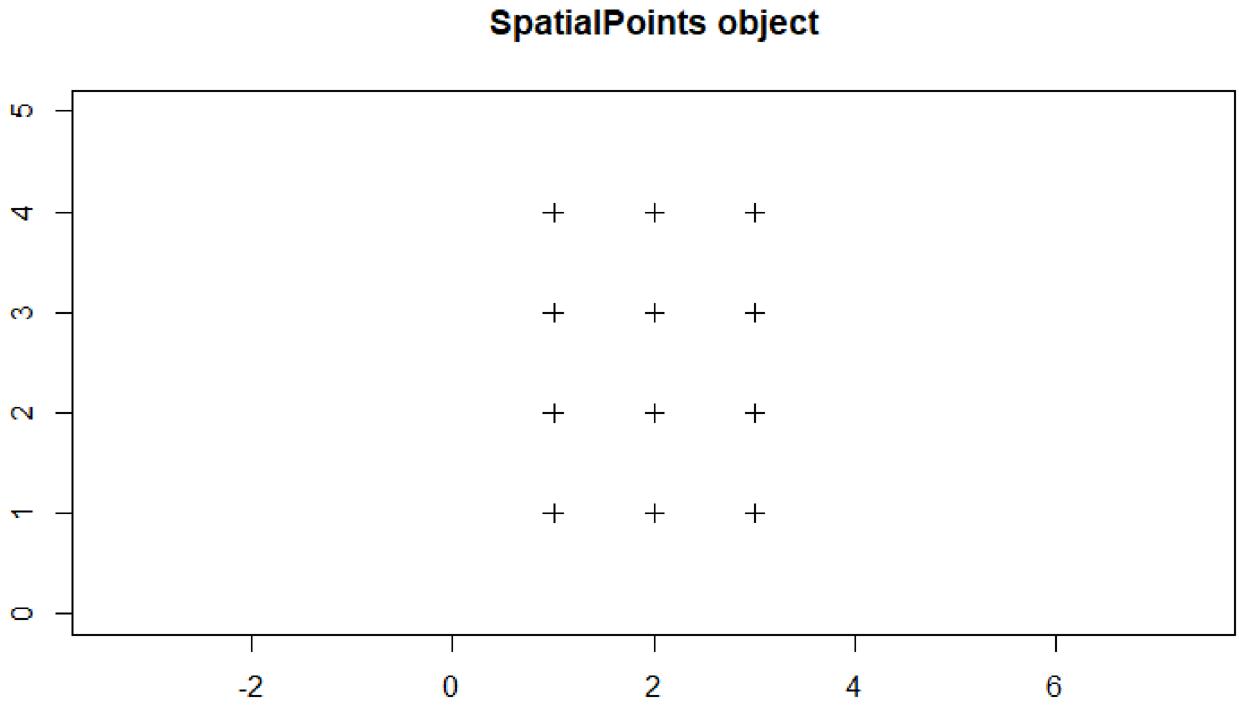


Create Grid object from points

Start to build objects from grid topology, coordinates or read gridded data

```
points_grid <-  expand.grid(x = 1:3, y = 1:4)
sp_points_grid <- SpatialPoints(points_grid)
```

Visualize prepared grid



Continuation to full grid

```
#from SpatialPoints SpatialPixels
pix_grid <- SpatialPixels(sp_points_grid)
#or this way object gridded (SpatialPoints into SpatialPixels)
class(sp_points_grid)
```

```
[1] "SpatialPoints"
attr(,"package")
[1] "sp"
```

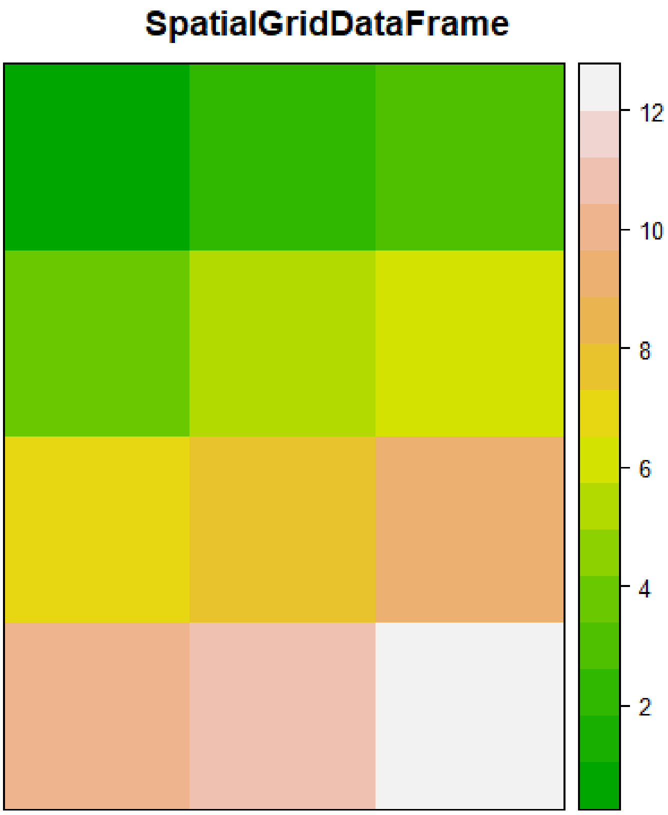
```
gridded(sp_points_grid) <- T
#is objet full grid?
fullgrid(pix_grid)
```

```
[1] FALSE
```

Continuation to full grid

```
#set for TRUE and create SpatialGrid
fullgrid(pix_grid) <- T
#add atributes (data.frame), to spatial components
atr_grid <- data.frame(matrix(1:12)) #data.frame
my_grid <- SpatialGridDataFrame(pix_grid,  atr_grid)
```

Visualize prepared grid



Import gridded data with *sp* package

Common way of importing gridded data; calculate summary statistics…

```
dem <- readGDAL ("dem_1k.asc")
```

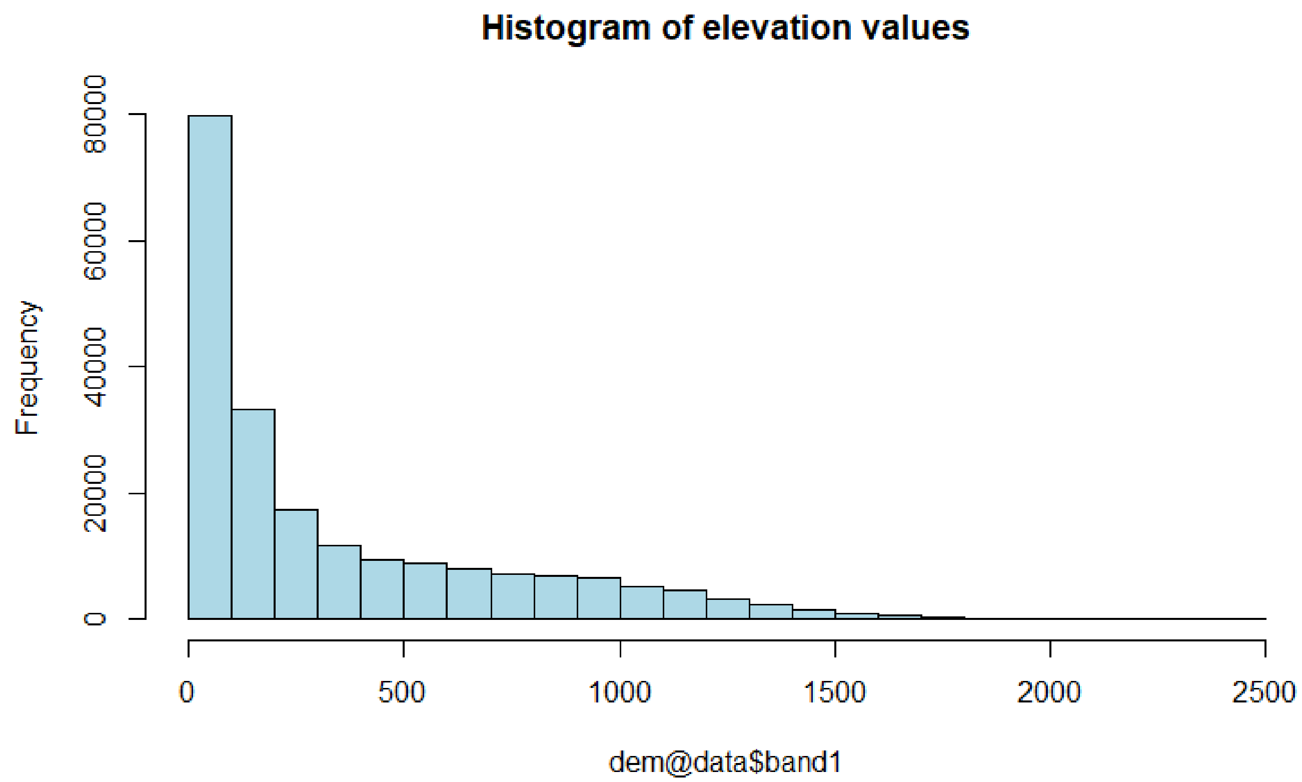
dem_1k.asc has GDAL driver AAIGrid and has 445 rows and 470 columns

```
#summary statistics
summary(dem@data$band1)
```

Min. 1st Qu. Median Mean 3rd Qu. Max. 0,0 0,0 166,4 344,0 560,6 2484,0

With all types of spatial data, statistical procedures from R

As example plot histogram of elevation values



Prepare additional data, spTransform

When data not on the same CRS

```
ETRS_LEA <- CRS("+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80 +units
=m +no_defs")
#sp transform to European equal area projection as dem
HRV_adm1_ETRS <- spTransform(HRV_adm1, ETRS_LEA)

#spatial extents
bbox(HRV_adm1_ETRS)
```

```
  min      max
```

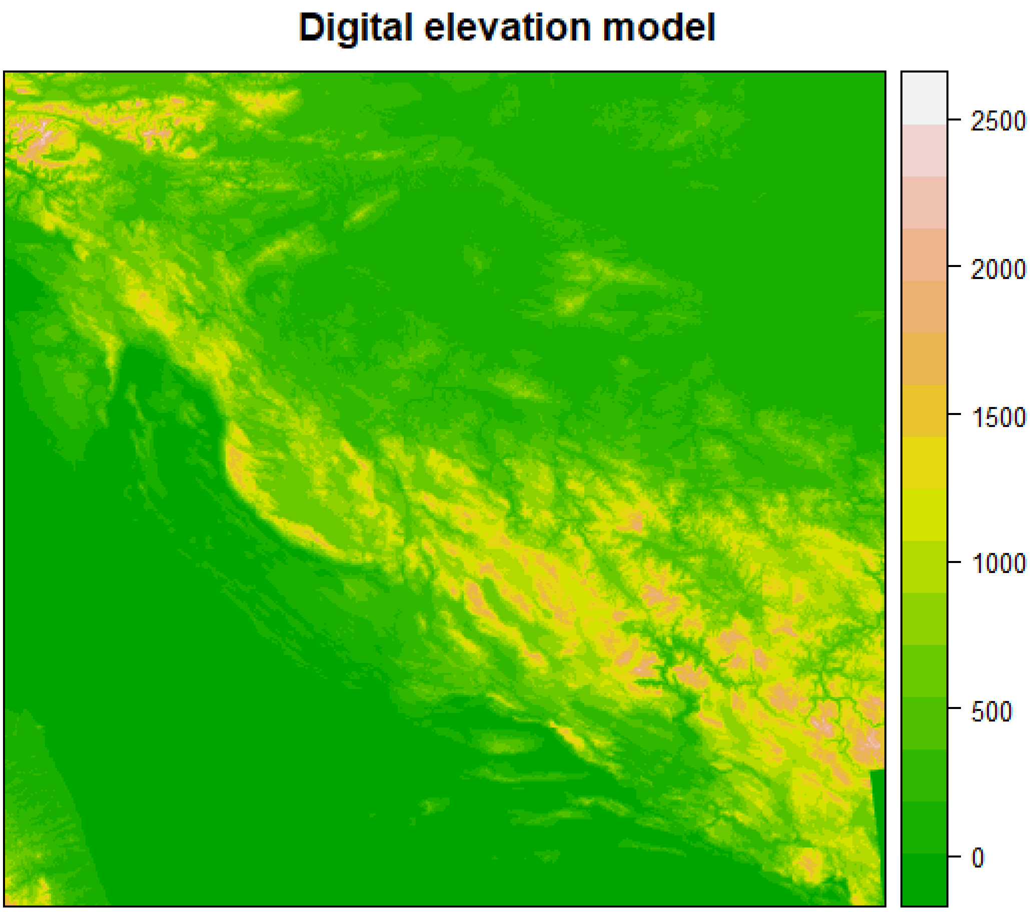
x 4594092 5061199 y 2163680 2624962

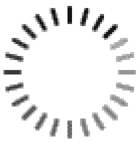
```
bbox(dem)
```

```
  min      max
```

x 4593479 5063479 y 2181875 2626875

Visualise gridded data with attributes





Export as …

ESRI asc, SAGA sgrd…

```
writeGDAL(dem, fname="dem.asc", drivername="AAIGrid",  mvFlag=99999)
writeGDAL (dem, fname= "dem.sdat",  drivername="SAGA")
writeGDAL (dem, fname="dem", drivername="GSBG")
```

Gridded data with *raster* package

Extremely popular for analysis of big rasters

Data does not need to be in memory

Easy shift to ‘sp’ classes

Extract wanted spatial extent or aggregate by polygon, select on points…

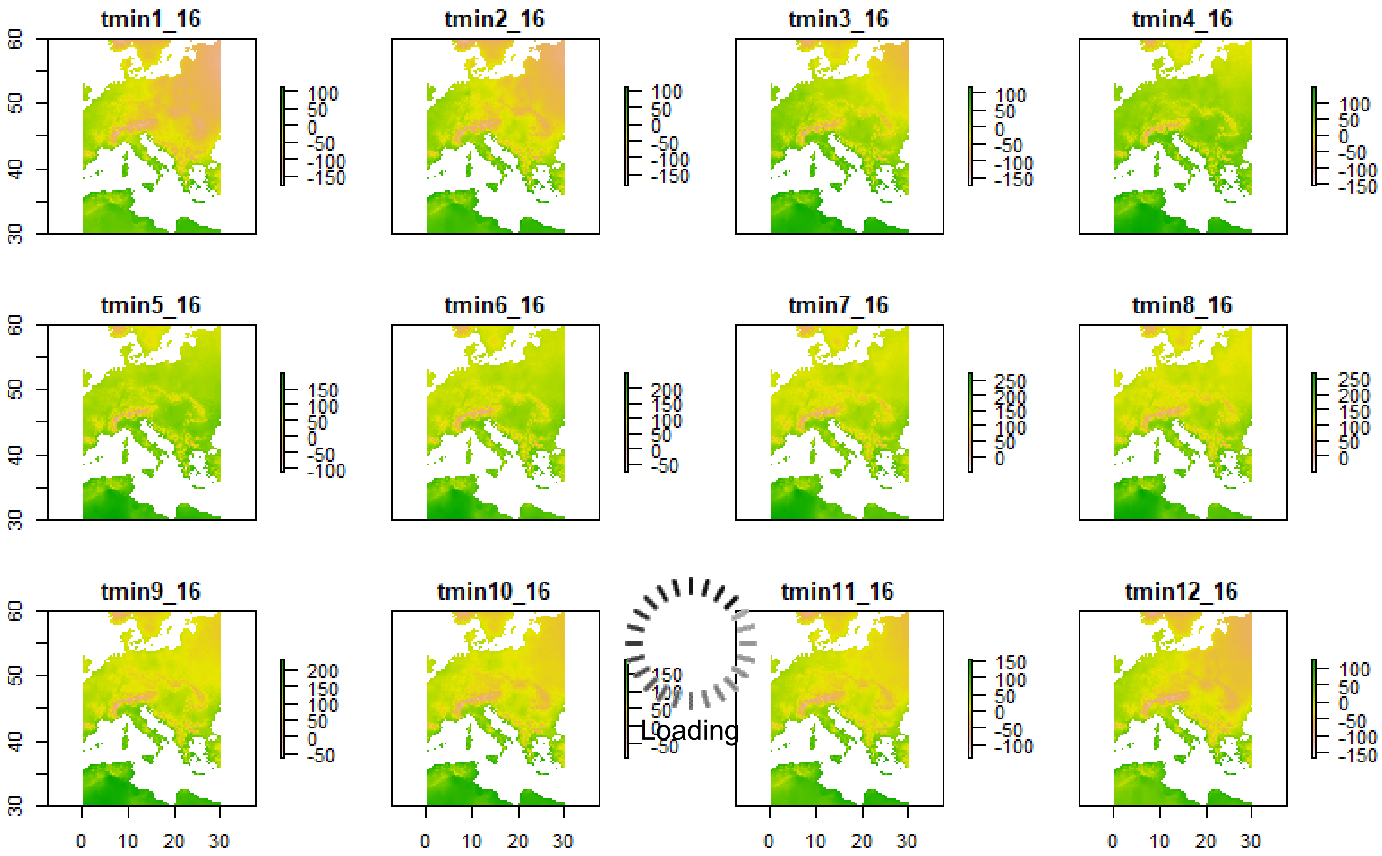
Import gridded data from Worldclim database, minimal temperature

Get data from Worldclim database - *raster* object

```
library(raster)
getData('worldclim', var='tmin', res=0.5, lon=5, lat=45)
```

class : RasterStack dimensions : 3600, 3600, 12960000, 12 (nrow, ncol, ncell, nlayers) resolution : 0,008333333, 0,008333333 (x, y) extent : 0, 30, 30, 60 (xmin, xmax, ymin, ymax) coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0 names : tmin1_16, tmin2_16, tmin3_16, tmin4_16, tmin5_16, tmin6_16, tmin7_16, tmin8_16, tmin9_16, tmin10_16, tmin11_16, tmin12_16 min values : -195, -202, -193, -176, -131, -96, -66, -66, -81, -110, -159, -181 max values : 116, 113, 120, 148, 198, 249, 271, 268, 232, 192, 158, 130

Plot *raster* object



Extract values for polygons

Values of Digital elevation model for each county

```
dem_r <- raster(dem)
#extracted values for each 1 km cell in polygons
#object of class list, length 21, as many counties
dem_county <- extract(dem_r, HRV_adm1_ETRS)
```

R specific loops

Ask for maximal value of elevation by county

[[1]][1] 1711,7

[[2]][1] 758,4

[[3]][1] 948,5

[[4]][1] 1030,4

[[5]][1] 930,7

[[6]][1] 1164,9

[[7]][1] 1354,6

[[8]][1] 496,8

[[9]][1] 975,3

[[10]][1] 1608,2

[[11]][1] 297

[[12]][1] 576,8

[[13]][1] 964,3

[[14]][1] 1370

[[15]][1] 563,2

[[16]][1] 1742,3

[[17]][1] 797,9

[[18]][1] 835,8

[[19]][1] 213,9

[[20]][1] 1424,1

[[21]][1] 1007,3

Static map Visualisation example

Example: static map - locally stored data

First we need to prepare data

```
#import geo data into R
library(rgdal)
#read via GDAL/OGR
W <- readOGR(getwd(),"TM_WORLD_BORDERS-0.3", verbose = F, stringsAsFactors=FALSE)
```

Inspect the data

```
#inspect CRS/SRID
W@proj4string
```

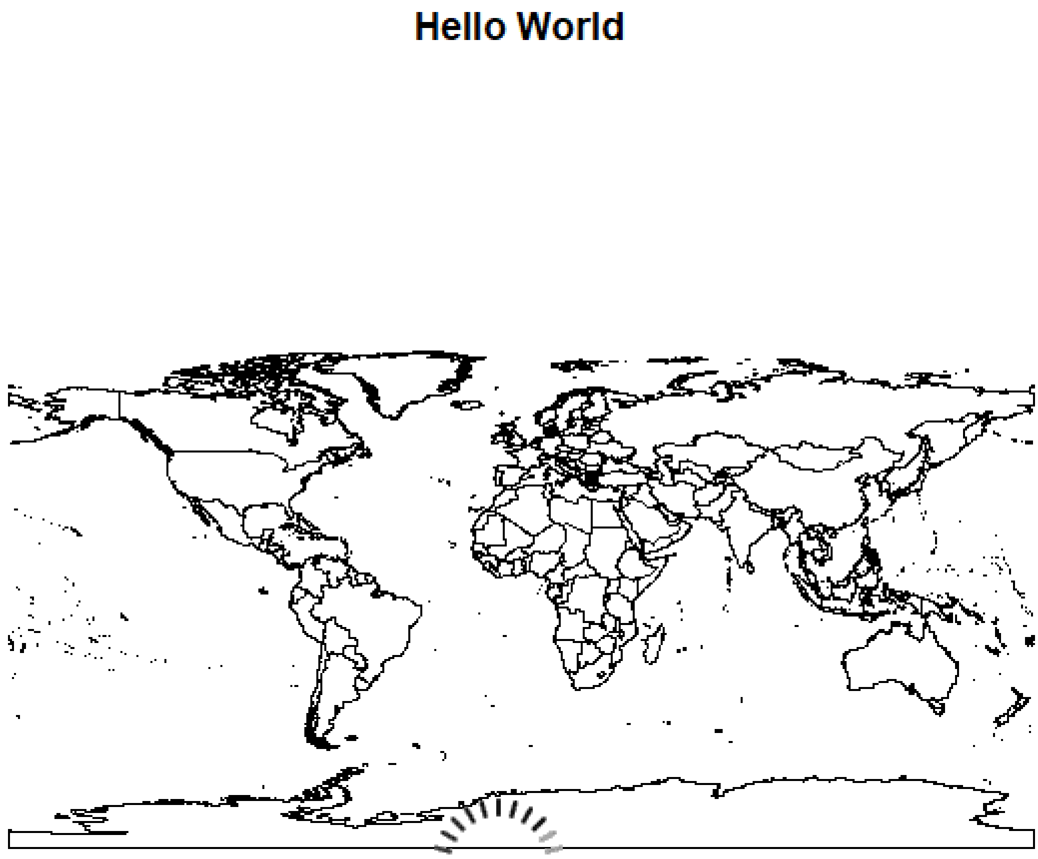
CRS arguments: +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0

```
#take a look at the data
head(W@data[1:5], n=1)
```

FIPS ISO2 ISO3 UN NAME 0 AC AG ATG 28 Antigua and Barbuda

Base graphic system, static map

```
plot(W, main="Hello World")
```



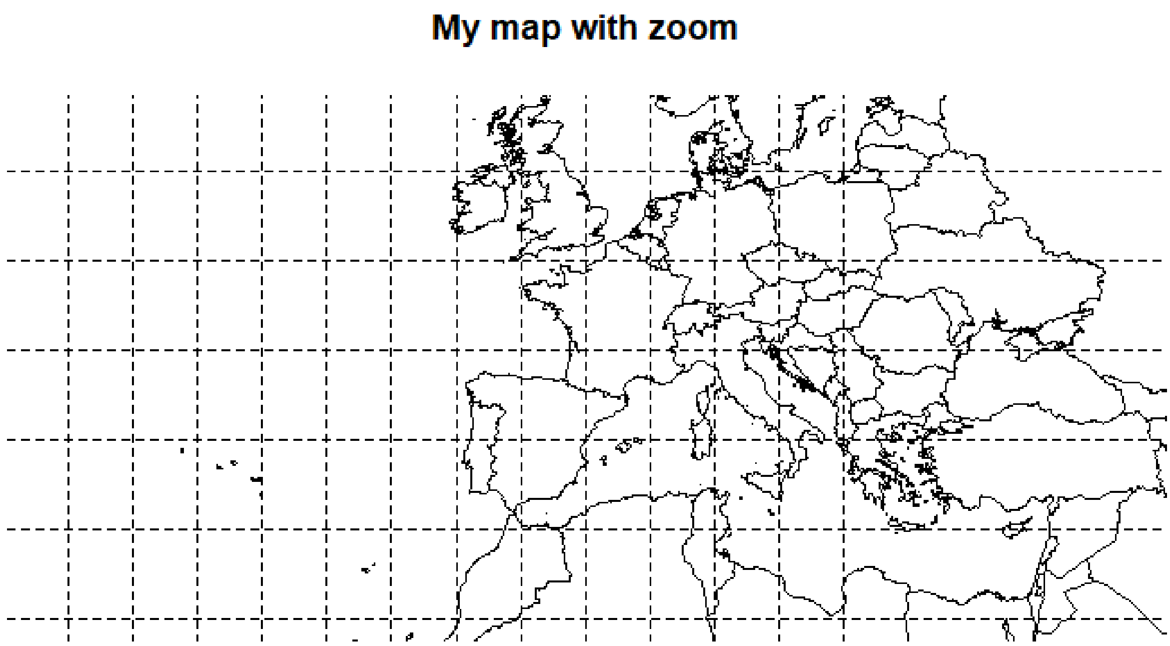
Loading

Base graphics

Zoom, additional improvements

```
#create gridlines
grat <- gridlines(W, easts=seq(-40,20,by=5), norths=seq(10,80,by=5), ndiscr = 20)
#zoom to specified spatial extent on plot
plot(W, xlim=c(-30,30), ylim=c(30,58))
#add gridlines
lines(grat, lty=2)
#add title
title(main="My map with zoom", sub="Gridlines added")
```

Static map



Gridlines added

Base graphic, different adjustments

Prepare additional objects to be plotted

```
load("df.RData") #load data.frame saved in R format
#create spatial objects from data.frame
coordinates(df) <- ~ lng  + lat
#define CRS to the points
#World Geodetic System - WGS
proj4string(df) <- CRS("+proj=longlat +datum=WGS84") #EPSG:4326
#colors(2)
col_2 <- colors()[351]
x <-Polygon(cbind(x=c(13.9, 20.2, 20.2, 13.9), y=c(42.5, 42.5,46.5, 46.5)))
```

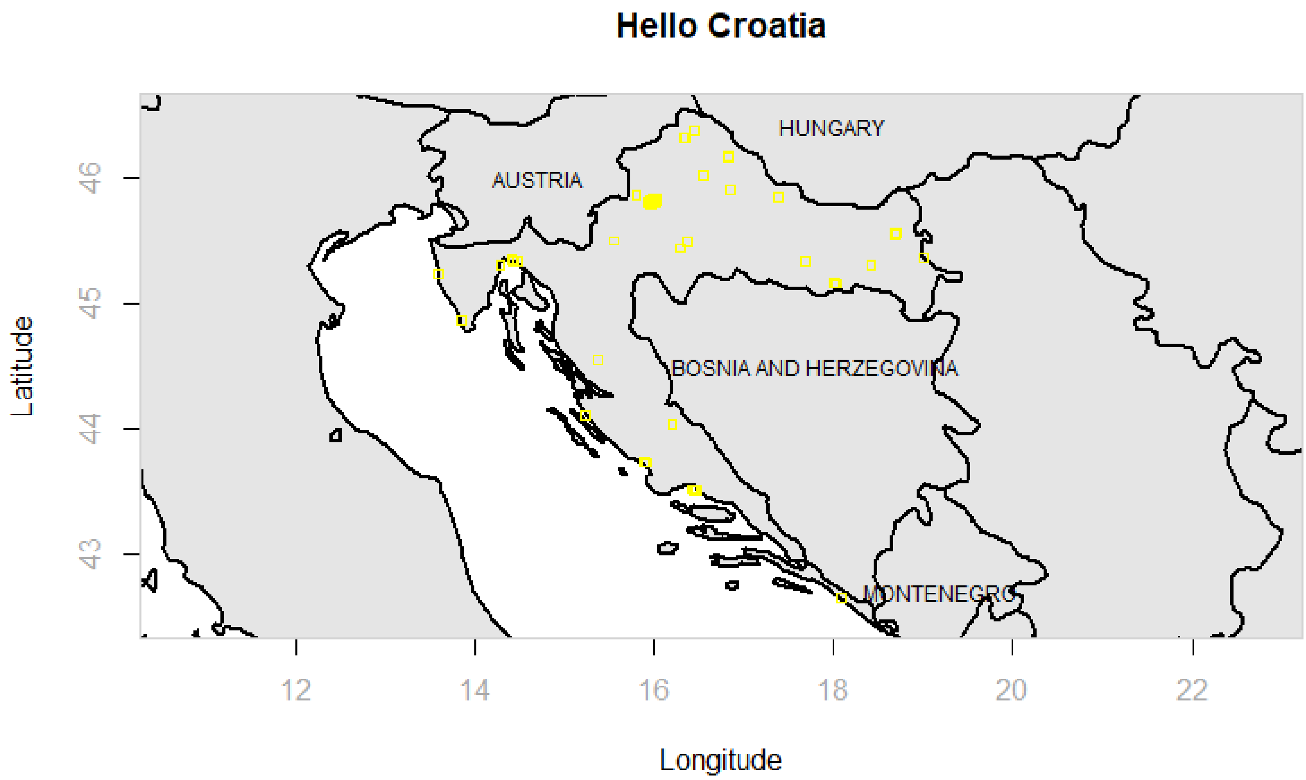
Improved map

```
plot(W,  col=col_2, xlim=c(13.9, 19.6), ylim=c(42.5, 46.5), main="Hello Croatia", lwd=2,  axes=F,  xlab="Longitude", ylab="Latitude")

#add points
points(df,  pch=22, cex=0.75, col="yellow")

axis(1, col.axis = "dark gray")
axis(2, col.axis = "dark gray")
box(lty = 'solid', col = 'light gray')
text(x=14.7, y=46, labels="AUSTRIA", cex=0.75)
text(x=17.8, y=44.5, labels="BOSNIA AND HERZEGOVINA", cex=0.75)
text(x=18, y=46.4, labels="HUNGARY", cex=0.75)
text(x=19.2, y=42.7, labels="MONTENEGRO", cex=0.75)
```

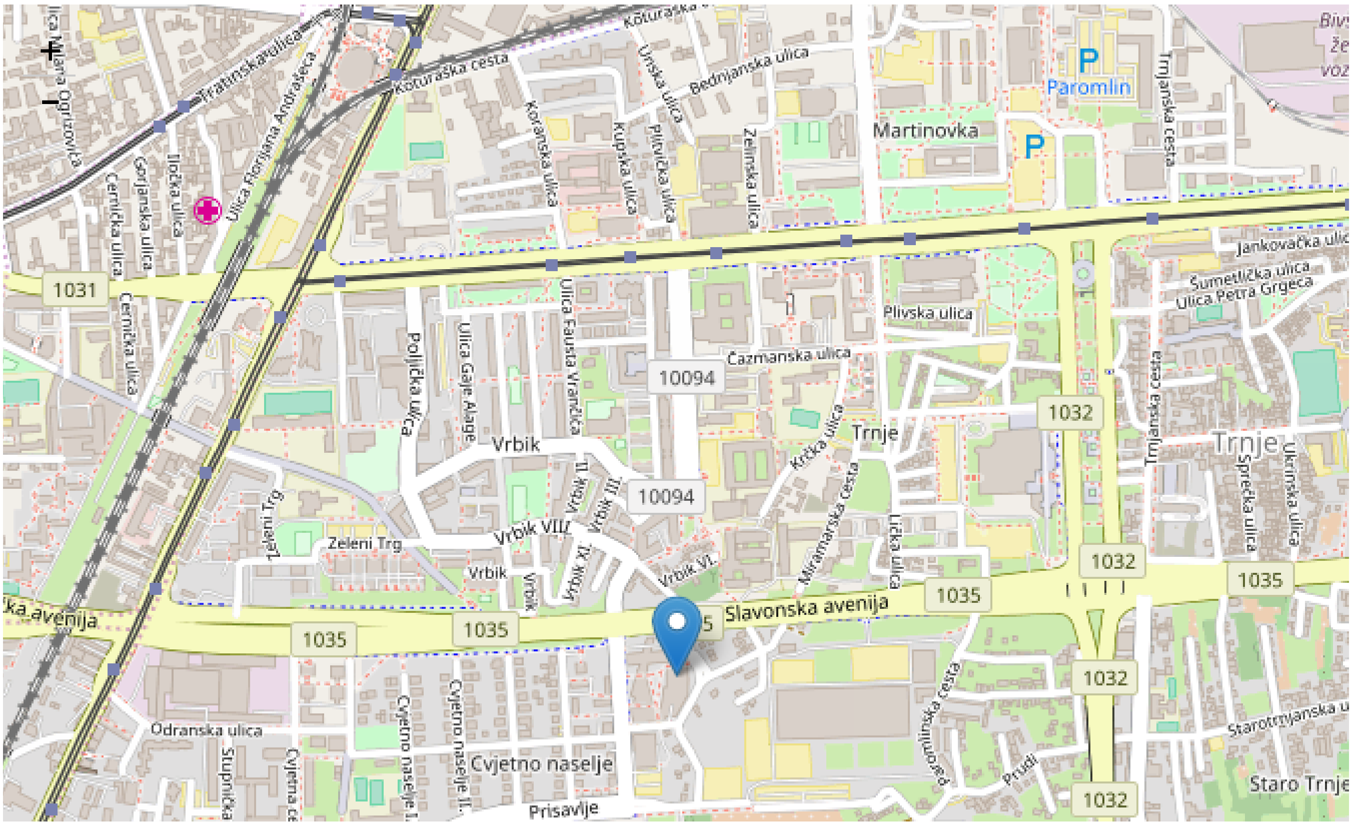
Improved map

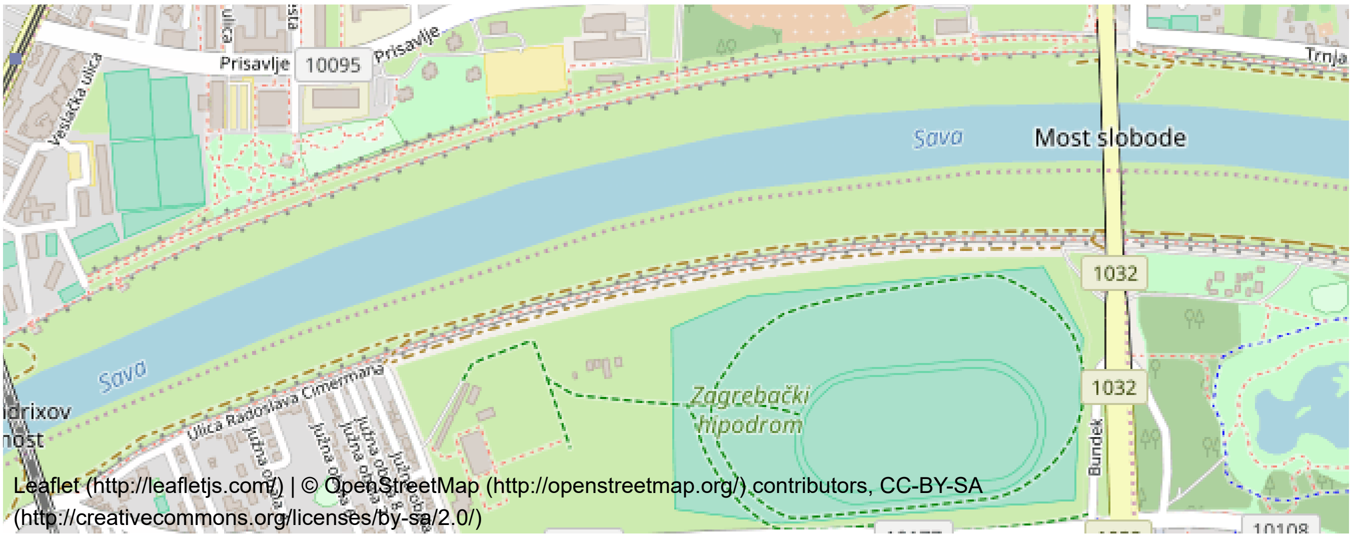


Leaflet example, point marker

Loading

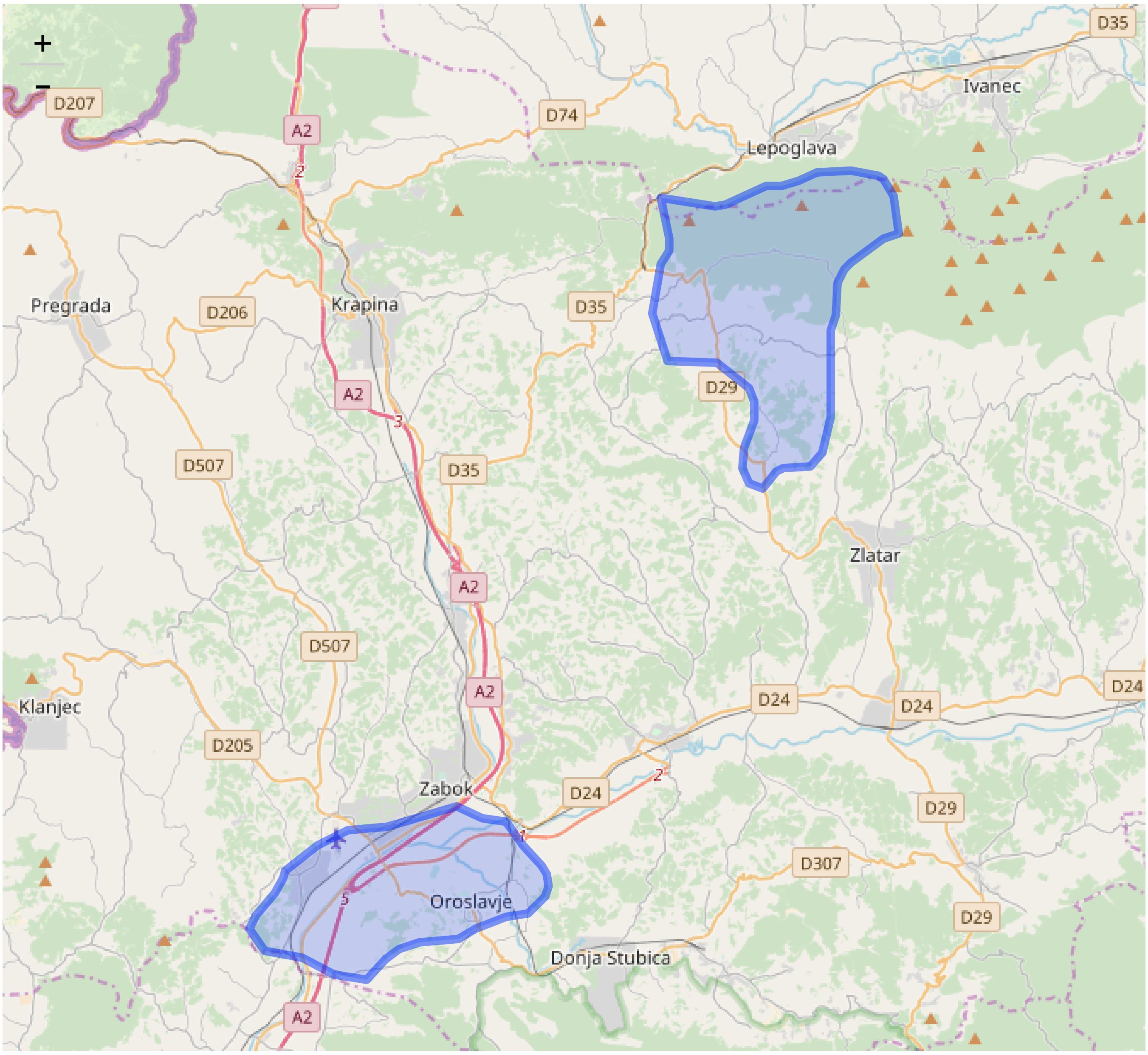
```
m <- leaflet()
m <- addTiles(m)
m <- addMarkers(m, lng= 15.970281, lat= 45.793467, popup="Plazza centar")
m
```





Leaflet example, polygons

```
m <- leaflet()
m <- addTiles(m) %>%
addPolygons(data=select,color = "#03F", weight = 5, opacity = 0.5,)
m
```



Example: data from MS Access

```
#connect to the DB
ptice.db <-odbcDriverConnect("ptice_gps.mdb")

#SQL query
galeb_52 <- sqlQuery(ptice.db, query="SELECT  (latitude) AS lat, (longitude) AS lon , gps_fix
status ,altitude, date_time
    FROM galebovi_2008
    WHERE serijski_broj = 52")

galeb_52_3d <- galeb_52[which(galeb_52$gps_fixstatus == '3D'),]
str(galeb_52_3d)
```

Create ‘spacetime’ class object - package ‘spacetime’

```
sp <- SpatialPoints(galeb_52_3d[,c("lon","lat")])
proj4string(sp) <- CRS("+proj=longlat +datum=WGS84")
library(spacetime)
galeb_52_3d.st <- STIDF(sp, time = galeb_52_3d$date_time, data = galeb_52_3d)

#select and sort by time
galeb_52_3d.st_600   <- galeb_52_3d.st[1:600,]

#shepe of an icon
shape <- "R_logo.png"

#export as kml
library(plotKML)
kml(galeb_52_3d.st_600, labels = date_time, colour = altitude,shape=shape,
colour_scale=rep("#FF00FF", 2), kmz = F)
```

Loading

```
KML file opened for writing...
```

```
Writing to KML...
```

```
Closing  galeb_52_3d.st_600.kml
```

Example: data from PostGres/PostGIS

```
library(RPostgreSQL) #R / PostgreSQL connect
library(RODBC) #connect to diff ODBC DB
library(rpostgis) #additional functionality for PostgreSQL/PostGIS
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, user="postgres",  dbname = "GEO_meetup", host = "PostgreSQL9.6(localhos
t:5432)", password="*****")
dbListTables(con)
HRV_adm1 <- "SELECT * FROM HRV_adm1*)
dbDisconnect(con)
```

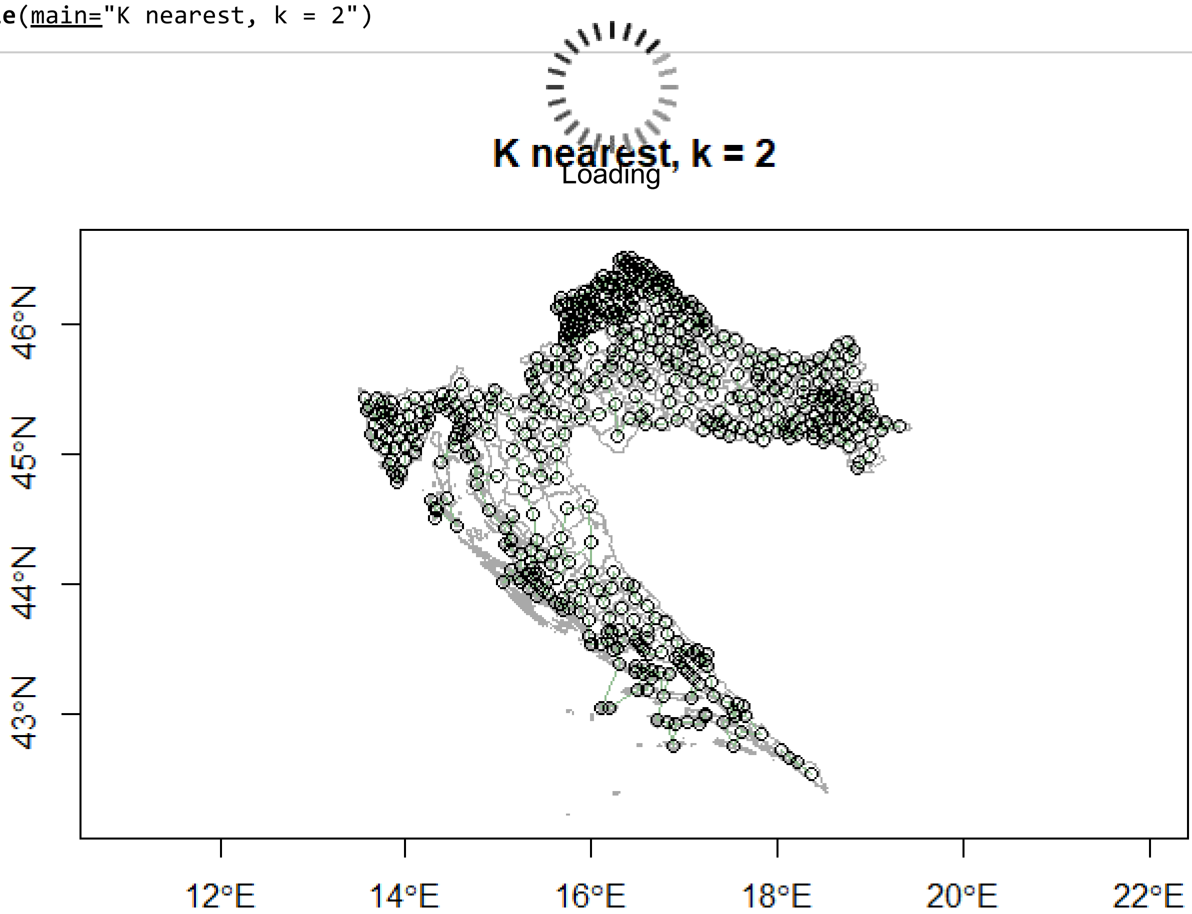
Examples of analysis

Nearest neighbourgs, k=2; library(spdep)

```
library(spdep)
HRV_adm2_nn2<-knearneigh(coordinates(HRV_adm2), k=2)
```

Visualisation of neighbours

```
plot(HRV_adm2, border="darkgrey", axes=T)
plot(knn2nb(HRV_adm2_nn2), coordinates(HRV_adm2), add=TRUE, col="darkseagreen")
title(main="K nearest, k = 2")
```



Analysis of point pattern

```
p_HR_300 <-  spsample(HRV_adm2,300, "random")
class(p_HR_300)
```

[1] “SpatialPoints” attr(,“package”) [1] “sp”

```
points_ppp <- as.ppp(p_HR_300) #create point pattern
points_ppp
```

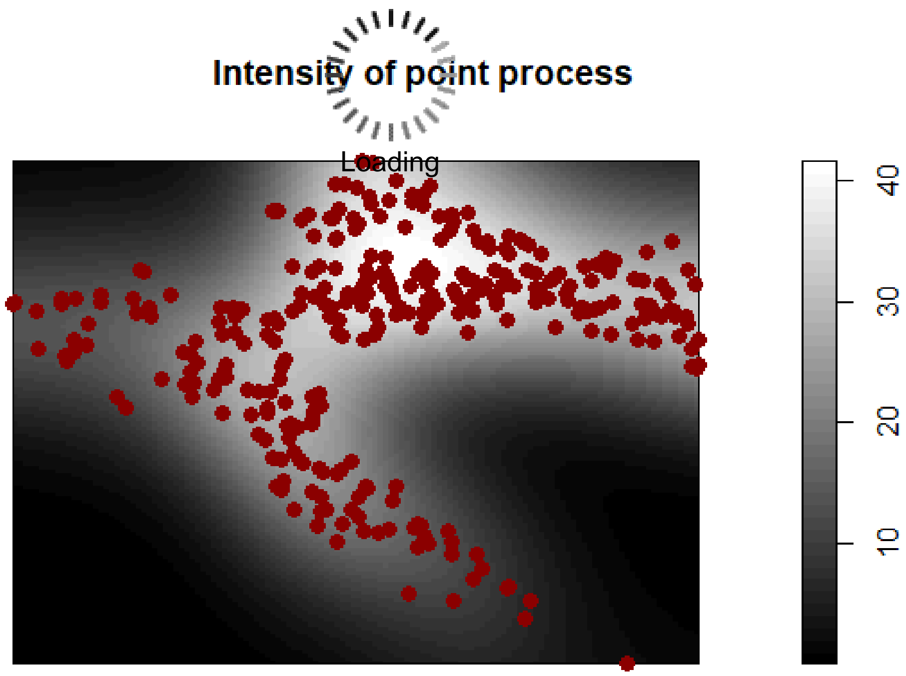
Planar point pattern: 300 points window: rectangle = [13,537576, 18,993982] x [42,52764, 46,52316] units

```
den_10000 <-  density(points_ppp, 10000) #density of point pattern at 10000 m
summary(den_10000)#summary of the result
```

real-valued pixel image 128 x 128 pixel array (ny, nx) enclosing rectangle: [13,53758, 18,99398] x [42,52764, 46,52316] units dimensions of each pixel: 0,0426 x 0,03121499 units Image is defined on the full rectangular grid Frame area = 21,8011742470826 square units Pixel values range = [13,76073, 13,76073] integral = 300 mean = 13,76073

Visualisation of result, export is necessary

```
plot(density(points_ppp), main="Intensity of point process", col=gray(0:40/40))
points(p_HR_300,  cex=1.2, pch=19, col="darkred",add==T)
```



Library *geosphere*

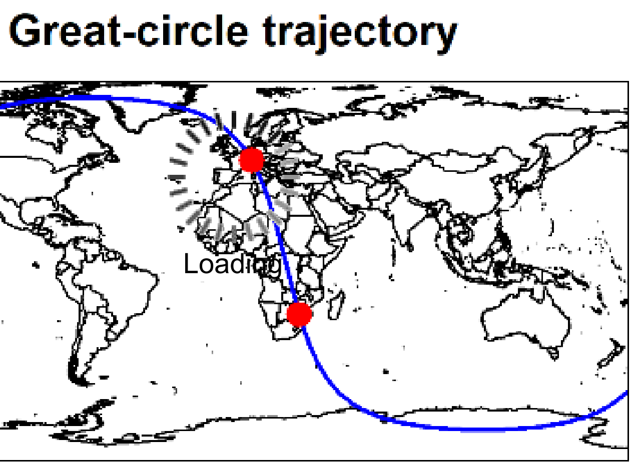
```
pts <- SpatialPoints(data.frame(x = c(8.33, 30), y = c(47.22, -23)))
proj4string(pts) <- CRS("+proj=longlat +datum=WGS84")

# compute the great-circle
gcLine <- greatCircle(pts[1], pts[2], sp = TRUE)
```

Visualising results

```
spplot(W[1], col.regions = "grey80", sp.layout = list(list("sp.lines", gcLine,
    lwd = 2, col = "blue"), list("sp.points", pts, pch = 16, cex = 1.5, col = "red")), colork
ey = FALSE)
```

Visualising results



Call external software for some algorithm

Example calling SAGA GIS geoprocessor

Possible to have sevaral versions of SAGA, specify in the call

```
library(RSAGA)
rsaga.get.modules(env = rsaga.env() )
myenv <-  rsaga.env(path="C:\\Program Files (x86)\\SAGA-GIS")
rsaga.get.modules(env=myenv)
rsaga.get.modules("shapes_polygons")
rsaga.get.usage("shapes_polygons", 2)
rsaga.geoprocessor (lib="shapes_polygons", 2, param=list(POLYGONS="my_shapefile.shp",
OUTPUT="result_shapefile.shp", BAREA=T))
```

Thank you for your attention!

Loading