Spatial Data in R: Overview and Examples
Earth Observation Workshop, Basel

Michael Dorman
Geography and Environmental Development
dorman@post.bgu.ac.il

2018-02-26 - 2018-03-01

Ben-Gurion University of the Negev

Contents

- The aim of this tutorial is to provide an **overview** to **spatial analysis in R**
- **Slides and code** available on GitHub
- **Data** available on Dropbox

Requirements

- Several packages **need to be installed** to run code examples -

```
install.packages("rgdal")
install.packages("sf")
install.packages("raster")
install.packages("rgeos")
install.packages("geosphere")
install.packages("rmapshaper")
install.packages("gstat")
install.packages("automap")
install.packages("spdep")
install.packages("spatstat")
install.packages("osmdata")
install.packages("mapview")
```

Requirements

- Set to **working directory** with the data -
  - `cb_2015_us_state_5m` - **US states (Shapefile)**
  - `us-states.geojson` - **US states (GeoJSON)**
  - `dem.tif` - **Digital Elevation Model (GeoTIFF)**

Introduction

- **R** is a **programming language** and **environment**, originally developed for **statistical computing** and **graphics**
- Over time, there was an increasing number of **contributed packages** for handling and analysing spatial data in R
- Today, spatial analysis is a **major functionality in R**
- As of November 2017, there are over **180 packages** specifically addressing spatial analysis in R

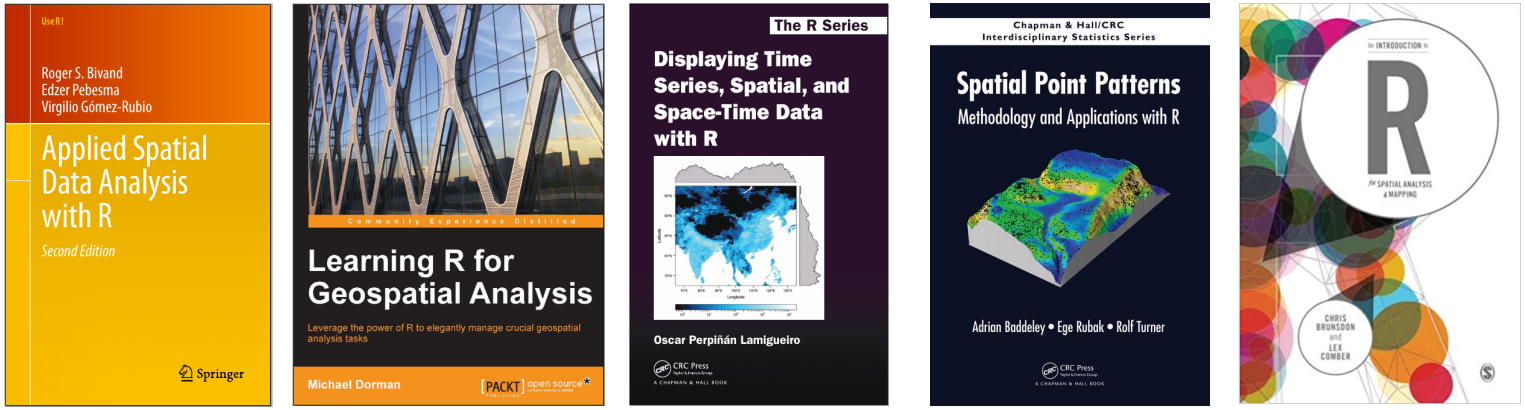

Figure 1: Books on Spatial Data Analysis with R

History

- pre-2003: Variable and incomplete approaches (`MASS`, `spatstat`, `maptools`, `geoR`, `splancs`, `gstat`, ...)
- 2003: Concensus that a package with base classes should be useful; `rgdal` released on CRAN
- 2005: `sp` released on CRAN; `sp` support in `rgdal`
- 2008: **Applied Spatial Data Analysis with R**, 1st ed.
- 2010: `raster` released on CRAN
- 2011: `rgeos` released on CRAN
- 2013: **Applied Spatial Data Analysis with R**, 2nd ed.
- 2016: `sf` released on CRAN

History

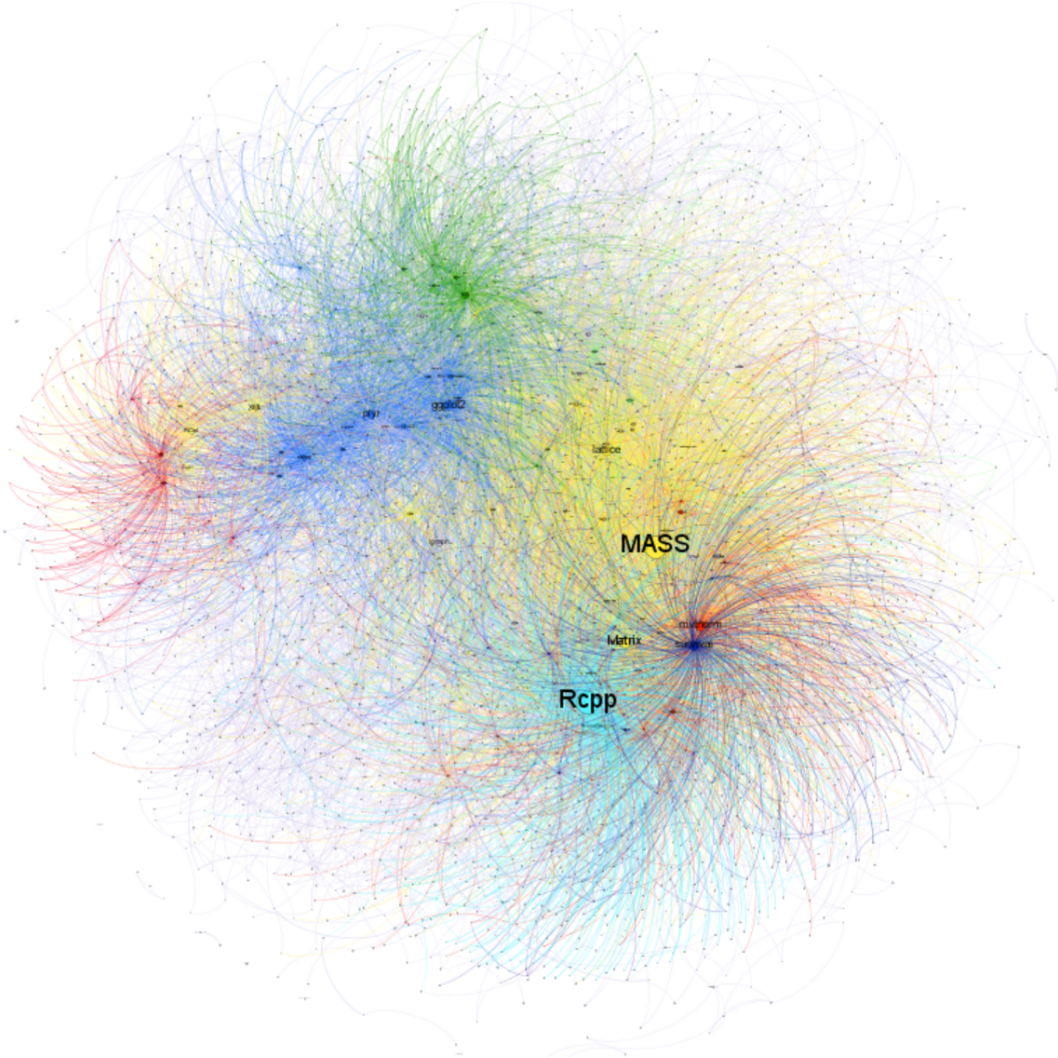

Figure 2: The network structure of CRAN[1]; `sp` ecosystem shown in green

[1] blog.revolutionanalytics.com/2015/07/the-network-structure-of-cran.html

R as a GIS?

- **Strengths** of R as a GIS
  - R capabilities in **data processing** and **visualization**, combined with dedicated **packages for spatial data**
  - A **single environment** encompassing all analysis aspects - acquiring data, computation, statistics, visualization, Web, etc.
- **General** advantages of programming
  - **Automation** - Doing otherwise unfeasible repetitive tasks
  - **Reproducibility** - Precise control of instructions to the computer
- Situations when **other tools** are needed
  - **Interactive editing or georeferencing** (but see `mapedit`)
  - Unique **GIS algorithms** (label placement, network routing, splitting lines at intersections, etc.)
  - Data that **cannot fit in RAM** (but R can **connect to spatial databases**)

Notable packages

- Handling spatial data
  - `sp`, `rgdal`, `rgeos`, `sf`, `raster`
- Geometry
  - `geosphere`, `mapshaper`
- Statistics
  - `gstat`, `automap`, `spatstat`, `spdep`
- Data access
  - `osmdata`
- Visualization
  - `ggplot2`, `ggmap`, `mapview`, `leaflet`

Vector: Types of data

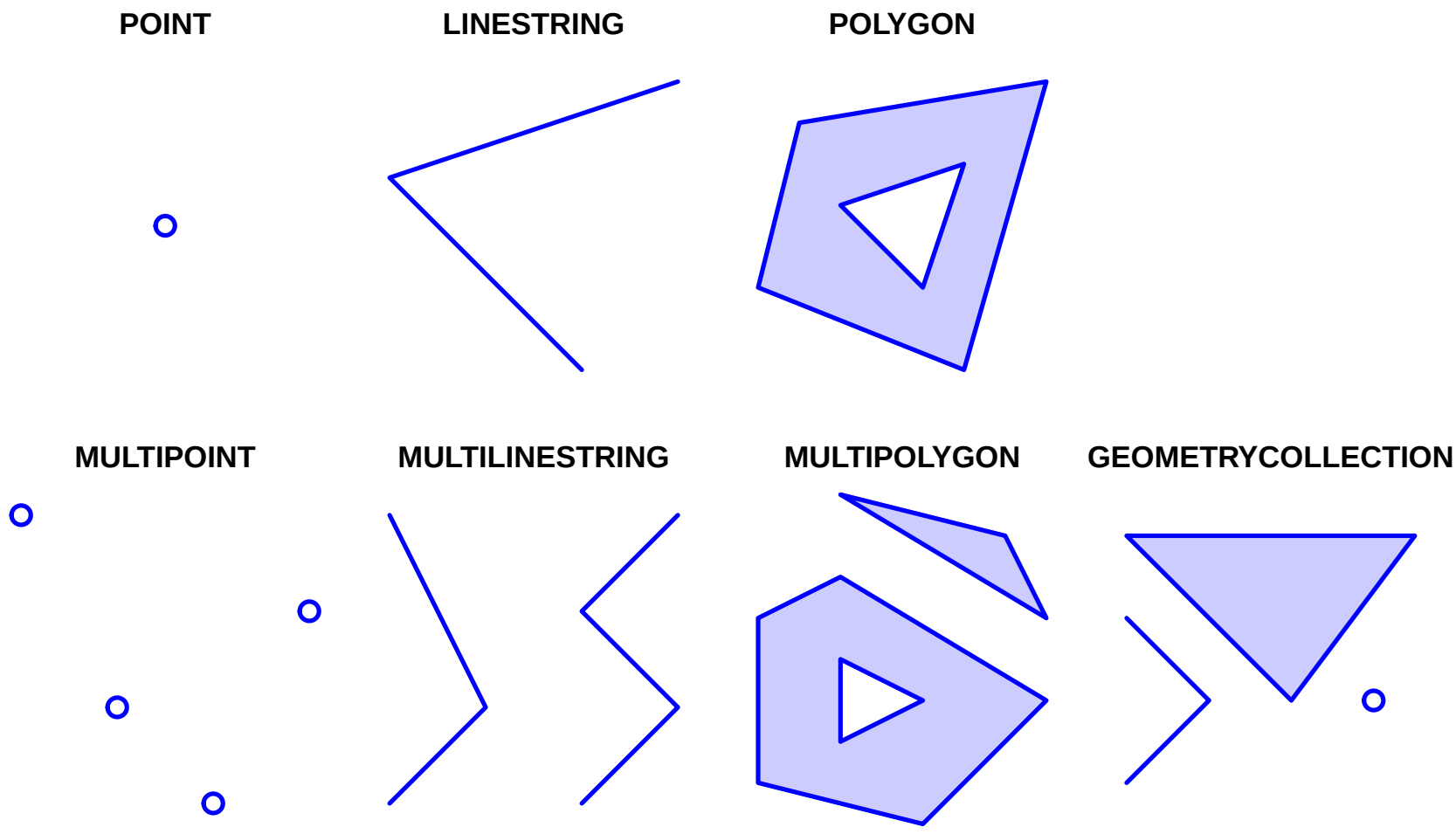

Figure 3: Vector data types (Simple Features)

Vector: File formats

- Binary
  - **ESRI Shapefile** (`.shp`, `.shx`, `.dbf`, `.prj`, ...)
- Plain Text
  - **GeoJSON** (`.json` or `.geojson`)
  - **GPX** (`.gpx`)
- Spatial Databases
  - **PostGIS / PostgreSQL**

Vector: Data structures (sp)

- Package sp has **6 main classes** for vector layers
  - One for each **geometry type** (points, lines, polygons)
  - One for **geometry only** and one for **geometry with attributes**

Table 1: Spatial data structures in package sp

class	geometry	attributes
SpatialPoints	Points	-
SpatialPointsDataFrame	Points	data.frame
SpatialLines	Lines	-
SpatialLinesDataFrame	Lines	data.frame
SpatialPolygons	Polygons	-
SpatialPolygonsDataFrame	Polygons	data.frame

Vector: Data structures (`sf`)

- Package `sf` defines a **hierarchical class system**
  - Class `sfg` is for a **single geometry**
  - Class `sfc` is a **set of geometries** with a CRS
  - Class `sf` is a **layer with attributes**

Table 2: Spatial data structures in package `sf`

class	hierarchy	data
`sfg`	geometry	coords, type, dimension
`sfc`	geometry column	Set of `sfg` + CRS
`sf`	layer	`sfc` + attributes

Raster: Types of data

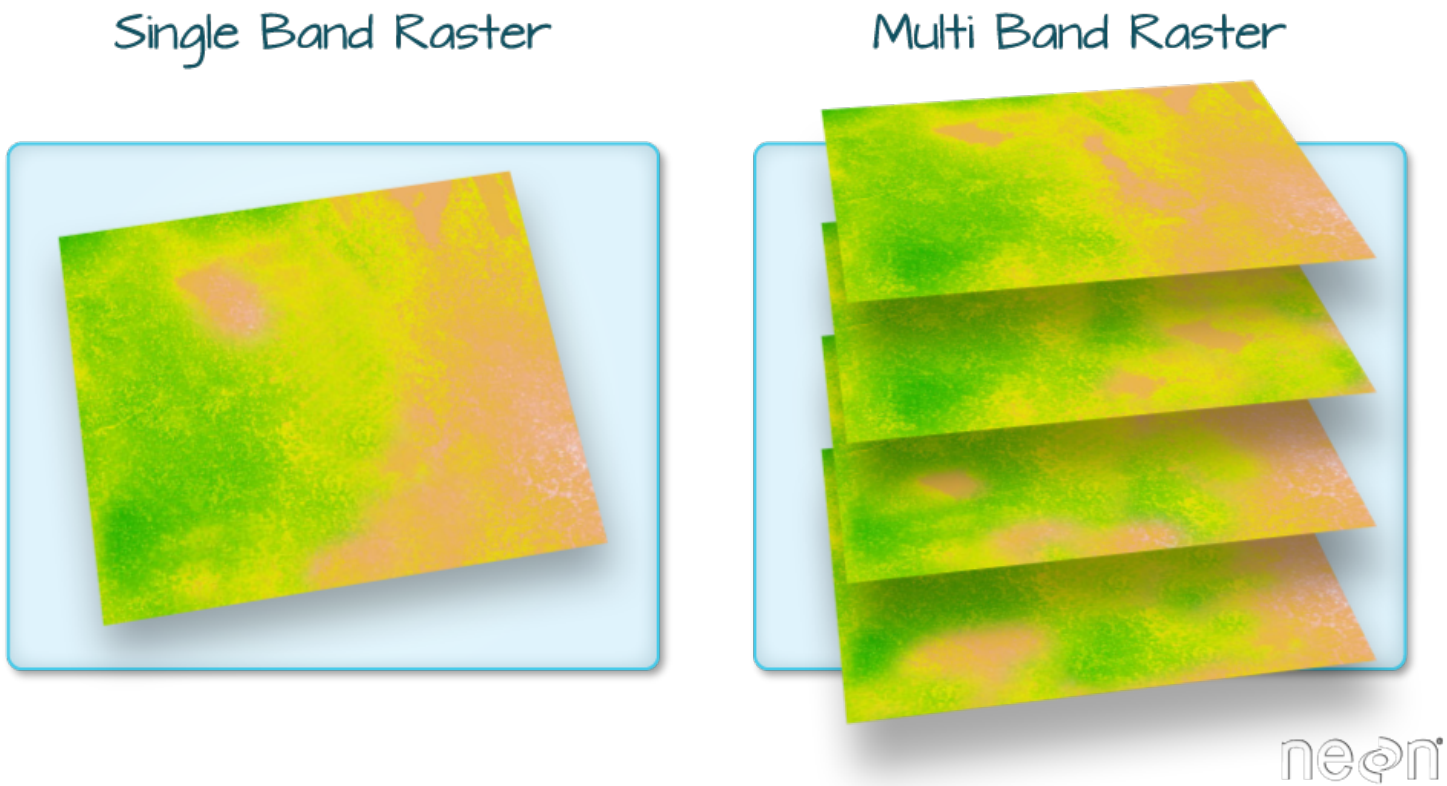

Figure 4: Raster data types[2]

[2] http://neondataskills.org/R/Introduction-to-Raster-Data-In-R/

Raster: File formats

- "Simple" rasters
  - **GeoTiff** (`.tif`)
  - **Erdas Imagine Image** (`.img`)
- "Complex" rasters (>3D and / or metadata)
  - **HDF** (`.hdf`)
  - **NetCDF** (`.nc`)

Raster: Data structures

- The `raster` package defines **3 classes** for raster data
  - `RasterLayer` for **single-band**
  - `RasterStack` and `RasterBrick` for **multi-band**

Table 3: Spatial data structures in package `raster`

class	layers	storage
`RasterLayer`	1	Disk or RAM
`RasterStack`	>=1	Disk and/or RAM
`RasterBrick`	>=1	Disk (single file) or RAM

Input and output of spatial data

- Reading spatial layers from a file into an R data structure, or writing the R data structure into a file, are handled by external libraries -
  - **OGR** is used for reading/writing vector files, with `rgdal` / `sp` or `sf`
  - **GDAL** is used for reading/writing raster files, with `raster`
  - **PROJ4** is used for handling CRS, in both `rgdal` / `sp`, `sf` and `raster`
  - Working with specialized formats, e.g. **HDF** with `gdalUtils` or **NetCDF** with `ncdf4`

Coordinate Reference Systems (CRS)

- A **Coordinate Reference System (CRS)** defines how the coordinates in the data relate to the surface of the Earth
- Two types of CRS
  - **Geographic** - longitude and latitude, in degrees
  - **Projected** - implying flat surface, associated with measurement units (e.g. meters)

Coordinate Reference Systems (CRS)

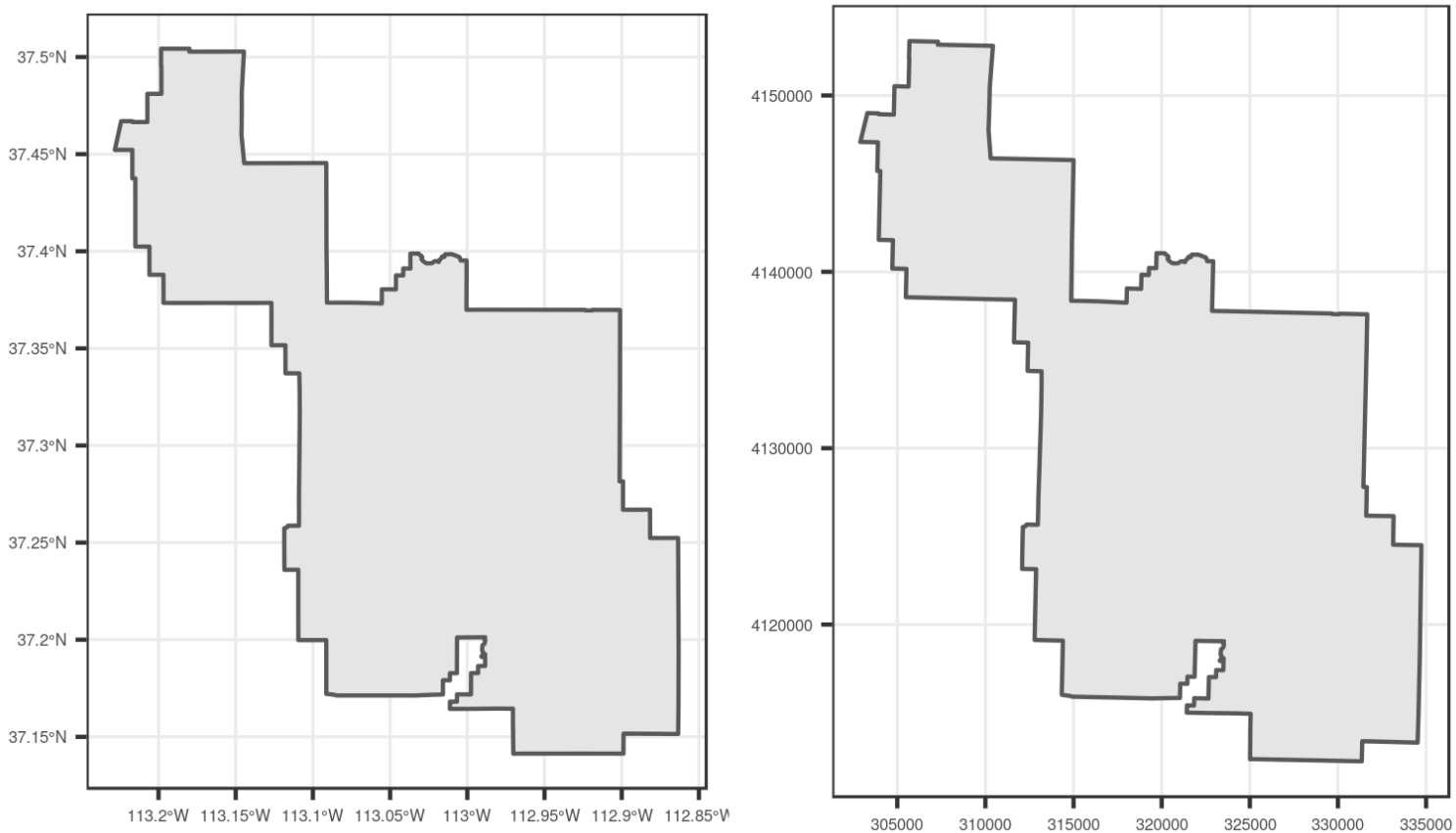

Figure 5: Geographic (WGS 84; left) and projected (NAD83 / UTM zone 12N; right) CRS[3]

[3] https://geocompr.robinlovelace.net/spatial-class.html

Coordinate Reference Systems (CRS)

- In R, CRS are defined using an **ESPG code** or **PROJ.4 string**
- Comparing Map Projections
- A table with EPSG code and PROJ.4 strings for all projections

-

```
crs = rgdal::make_EPSG()
```

Coordinate Reference Systems (CRS)

- For example, the **WGS84 geographic projection** can be defined using its **EPSG code**

```
4326
```

- Or using its **PROJ.4 string**

```
"+proj=longlat +datum=WGS84 +no_defs"
```

`rgdal`, `sf`, `raster`: Handling spatial data

- Function **`readOGR`** can be used to read vector layers into `sp` data structures -

```
library(rgdal)

pol_sp = readOGR("data", "cb_2015_us_state_5m")
## OGR data source with driver: ESRI Shapefile
## Source: "data", layer: "cb_2015_us_state_5m"
## with 49 features
## It has 1 fields

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

rgdal, sf, raster: Handling spatial data

- Function **st_read** can be used to read vector layers into sf data structures -

```
library(sf)

pol_sf = st_read("data/cb_2015_us_state_5m.shp")
## Reading layer `cb_2015_us_state_5m' from data source `/home/michael/Dropbox/
## Simple feature collection with 49 features and 1 field
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -124.7332 ymin: 24.51496 xmax: -66.9499 ymax: 49.38436
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs

class(pol_sf)
## [1] "sf"         "data.frame"
```

`rgdal`, `sf`, `raster`: Handling spatial data

- Since `sf` is new, the majority of the R-Spatial ecosystem **only works with `sp`**
- **"Migration"** document between `sp` and `sf`
- Conversion **`sp`** $\rightarrow$ **`sf`**

```
x = st_as_sf(pol_sp)
class(x)
## [1] "sf"           "data.frame"
```

- Conversion **`sf`** $\rightarrow$ **`sp`**

```
x = as(pol_sf, "Spatial")
class(x)
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

rgdal, sf, raster: Handling spatial data

- Functions **raster**, **stack** and **brick** can be used to read single-band rasters into raster data structures -

```
library(raster)

r = raster("data/dem.tif")
r
## class       : RasterLayer
## dimensions  : 333, 286, 95238  (nrow, ncol, ncell)
## resolution  : 90, 90  (x, y)
## extent      : 673414, 699154, 3615239, 3645209  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=utm +zone=36 +ellps=WGS84 +units=m +no_defs
## data source : /home/michael/Dropbox/Presentations/p_2018_02_EO_Workshop_Base
## names       : dem
## values      : -12, 537  (min, max)

class(r)
## [1] "RasterLayer"
## attr(,"package")
## [1] "raster"
```

rgdal, sf, raster: Handling spatial data

- CRS of **sp vector layer**

```
proj4string(pol_sp)
## [1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +t
```

- CRS of **raster layer**

```
proj4string(r)
## [1] "+proj=utm +zone=36 +ellps=WGS84 +units=m +no_defs"
```

- CRS of **sf vector layer**

```
st_crs(pol_sf)
## Coordinate Reference System:
##   EPSG: 4326
##   proj4string: "+proj=longlat +datum=WGS84 +no_defs"
```

`rgeos`, `sf`: Geoprocessing Vector Layers

- **GEOS** is used for geometric operations on **vector layers** with `rgeos` / `sf` -
  - **Numeric operators** - Area, Length, Distance...
  - **Logical operators** - Contains, Within, Within distance, Crosses, Overlaps, Equals, Intersects, Disjoint, Touches...
  - **Geometry generating operators** - Centroid, Buffer, Intersection, Union, Difference, Convex-Hull, Simplification...

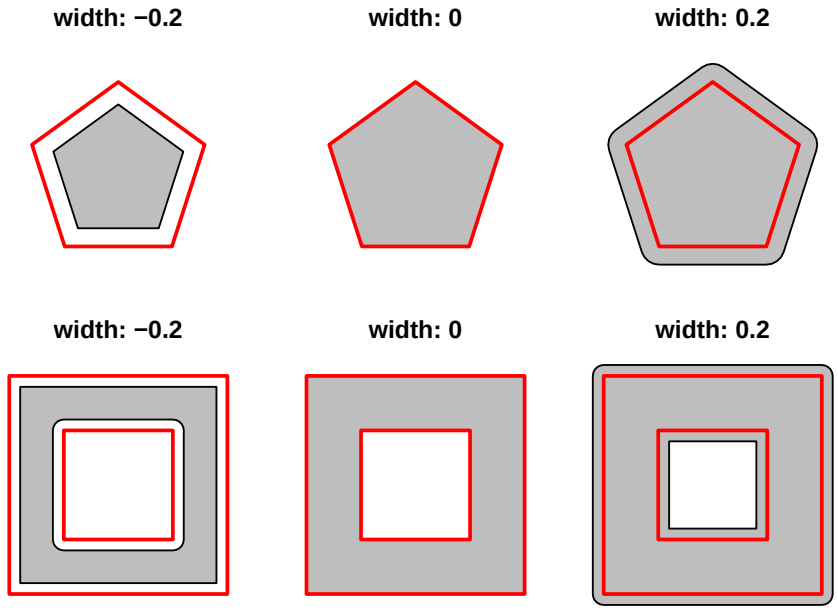

Figure 6: Buffer function

rgeos, sf: Geoprocessing Vector Layers

- Distance **using package rgeos**

```
library(rgeos)

gDistance(pol_sp[1, ], pol_sp[2, ], byid = TRUE)
##           0
## 1 20.57536
```

- Distance **using package sf**

```
st_distance(pol_sf[1, ], pol_sf[2, ])
## Units: m
##          [,1]
## [1,] 1888200
```

`raster`: Geoprocessing Rasters

- Geometric operations on **rasters** can be done with package `raster` -
  - **Accessing cell values** - As vector, As matrix, Extract to points / lines / polygons, random / regular sampling, Frequency table, Histogram...
  - **Raster algebra** - Arthmetic (+, −, ...), Math (`sqrt`, `log10`, ...), logical (`!`, ==, >, ...), summary (`mean`, `max`, ...), Mask, Overlay...
  - **Changing resolution and extent** - Crop, Mosaic, (Dis)aggregation, Reprojection, Resampling, Shift, Rotation...
  - **Focal operators** - Distance, Direction, Focal Filter, Slope, Aspect, Flow direction...
  - **Transformations** - Vector layers <-> Raster...

raster: Geoprocessing Rasters

```
slope_asp = terrain(
  r,
  opt = c("slope", "aspect"),
  unit = "degrees"
)
```

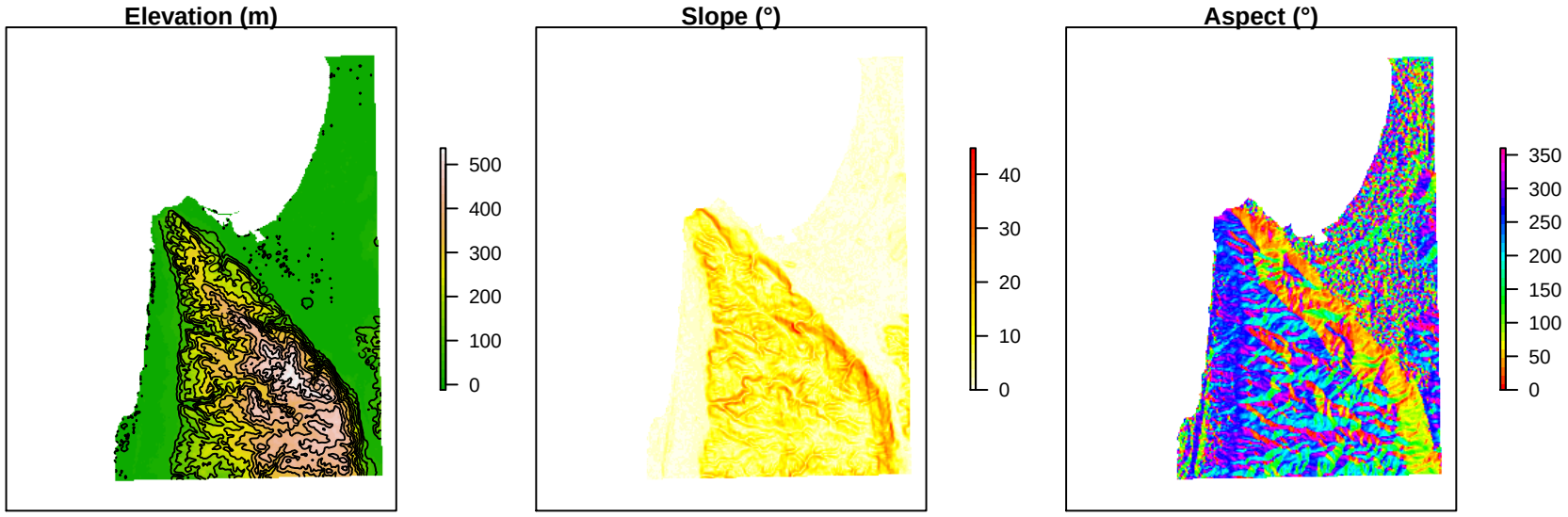

Figure 7: Topographic aspect and slope calculation

`geosphere`: Geometric calculations on longitude/latitude

- Package `geosphere` implements **spherical trigonometry** functions for distance and direction-related calculations on **geographic coordinates (lon-lat)**

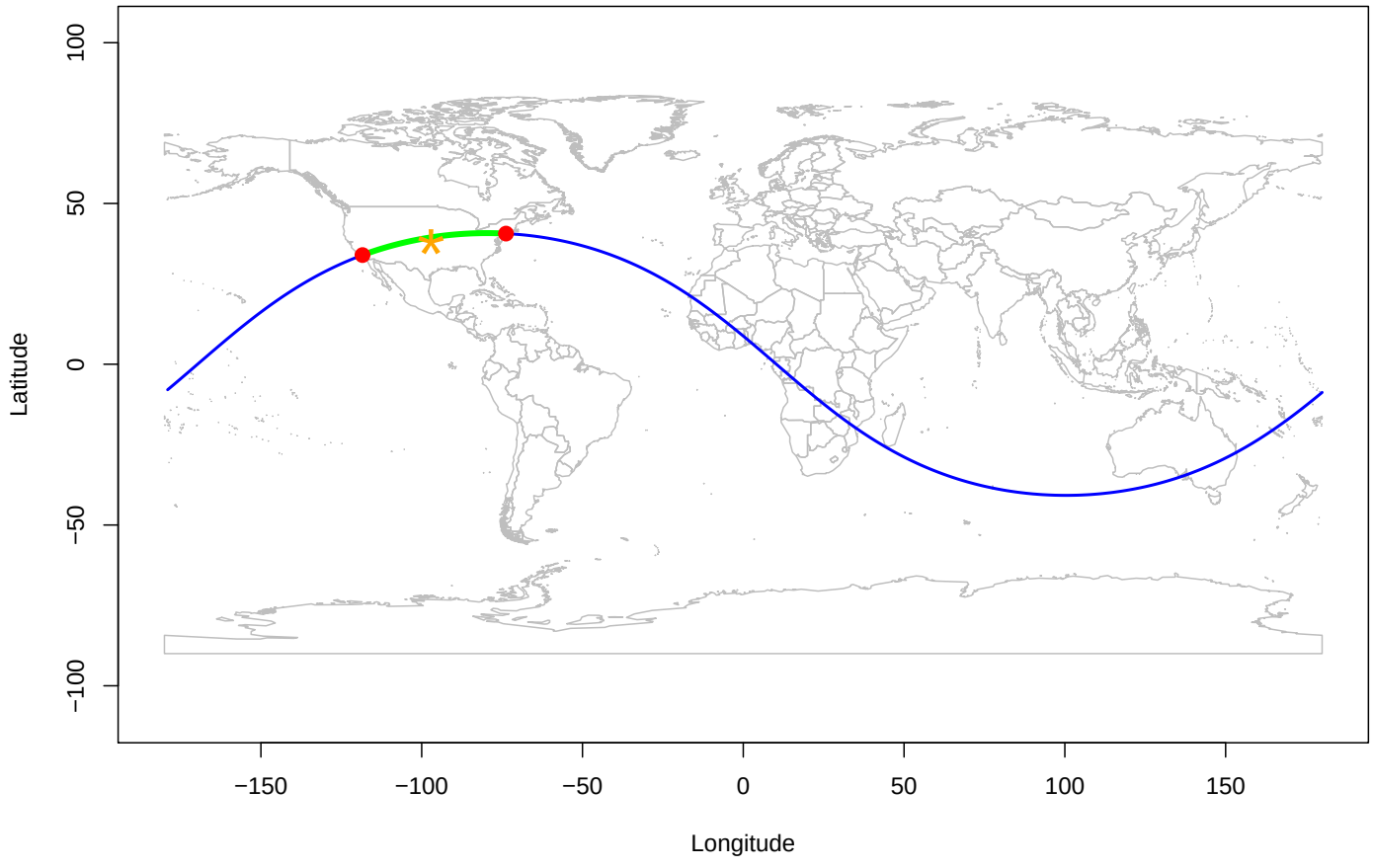

Figure 8: Points on Great Circle

`geosphere`: Geometric calculations on longitude/latitude

Figure 9: Visualizing Facebook Friends with `geosphere`[4]

[4] http://paulbutler.org/archives/visualizing-facebook-friends/

geosphere: Example

```
library(geosphere)

LA = c(-118.40, 33.95)
NY = c(-73.78, 40.63)
gci = gcIntermediate(LA, NY)

head(gci)
##              lon      lat
## [1,] -117.6264 34.23369
## [2,] -116.8477 34.51250
## [3,] -116.0638 34.78633
## [4,] -115.2747 35.05510
## [5,] -114.4804 35.31872
## [6,] -113.6810 35.57712
```

rmapshaper

- `rmapshaper` is an R wrapper around the **mapshaper** tool, which also has a nice web-interface
- The main advantage of `rmapshaper` is the topologically-aware simplification algorithm (also see Vignette)
- Shared boundaries between polygons are always kept intact, with no gaps or overlaps

```
library(rmapshaper)
pol_sp_s1 = ms_simplify(pol_sp, keep = 0.05)
pol_sp_s2 = ms_simplify(pol_sp, keep = 0.01)
```

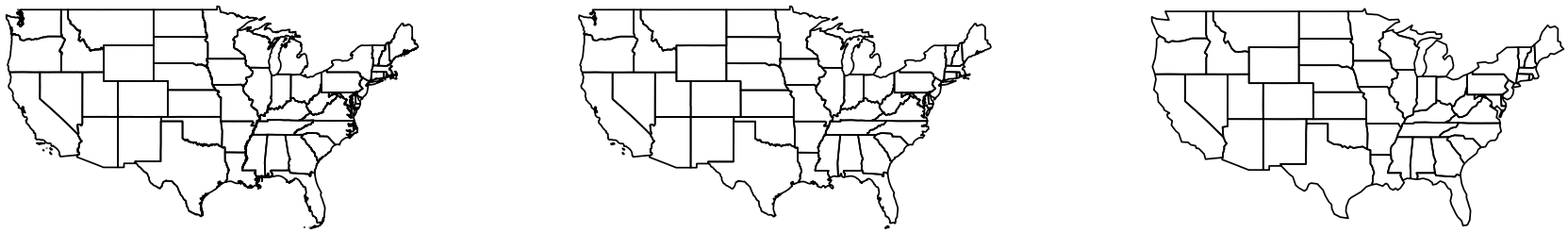

Figure 10: Original (left), 0.05 points retained (middle) and 0.01 retained (right)

gstat: Geostatistical Modelling

- Univariate and multivariate geostatistics -
  - Variogram modelling
  - Ordinary and universal point or block (co)kriging
  - Cross-validation

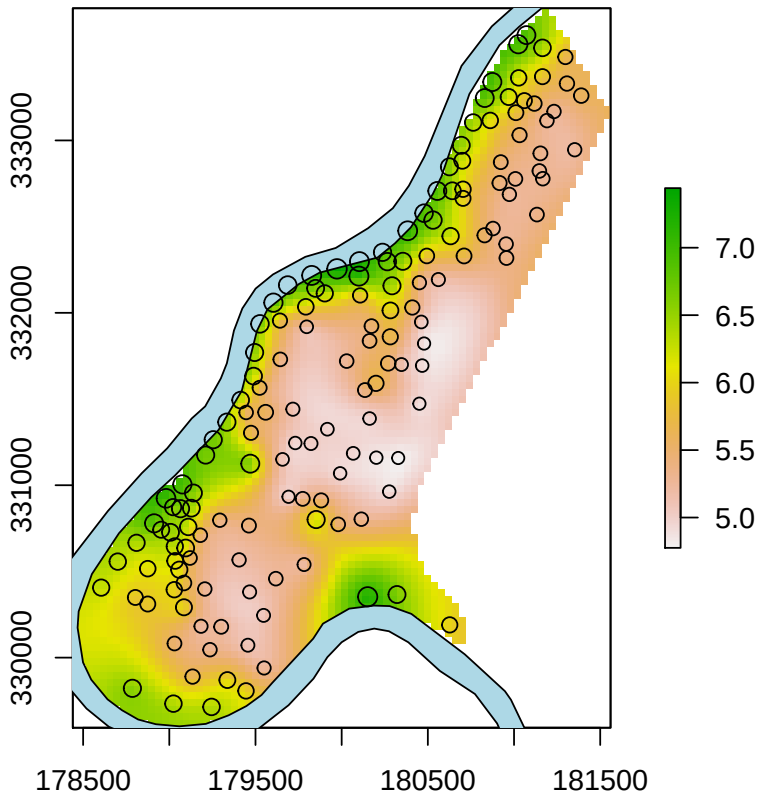

Figure 11: Predicted Zinc concentration, using Ordinary Kriging

gstat: Example

- Preparing input data

```
library(gstat)
library(automap)

data(meuse)
data(meuse.riv)
coordinates(meuse) = ~ x + y
data(meuse.grid)
gridded(meuse.grid) = ~ x + y
grid = raster(meuse.grid)
grid[!is.na(grid)] = 1
```

- Plot

```
plot(grid)
plot(meuse, add = TRUE)
```

gstat: Example

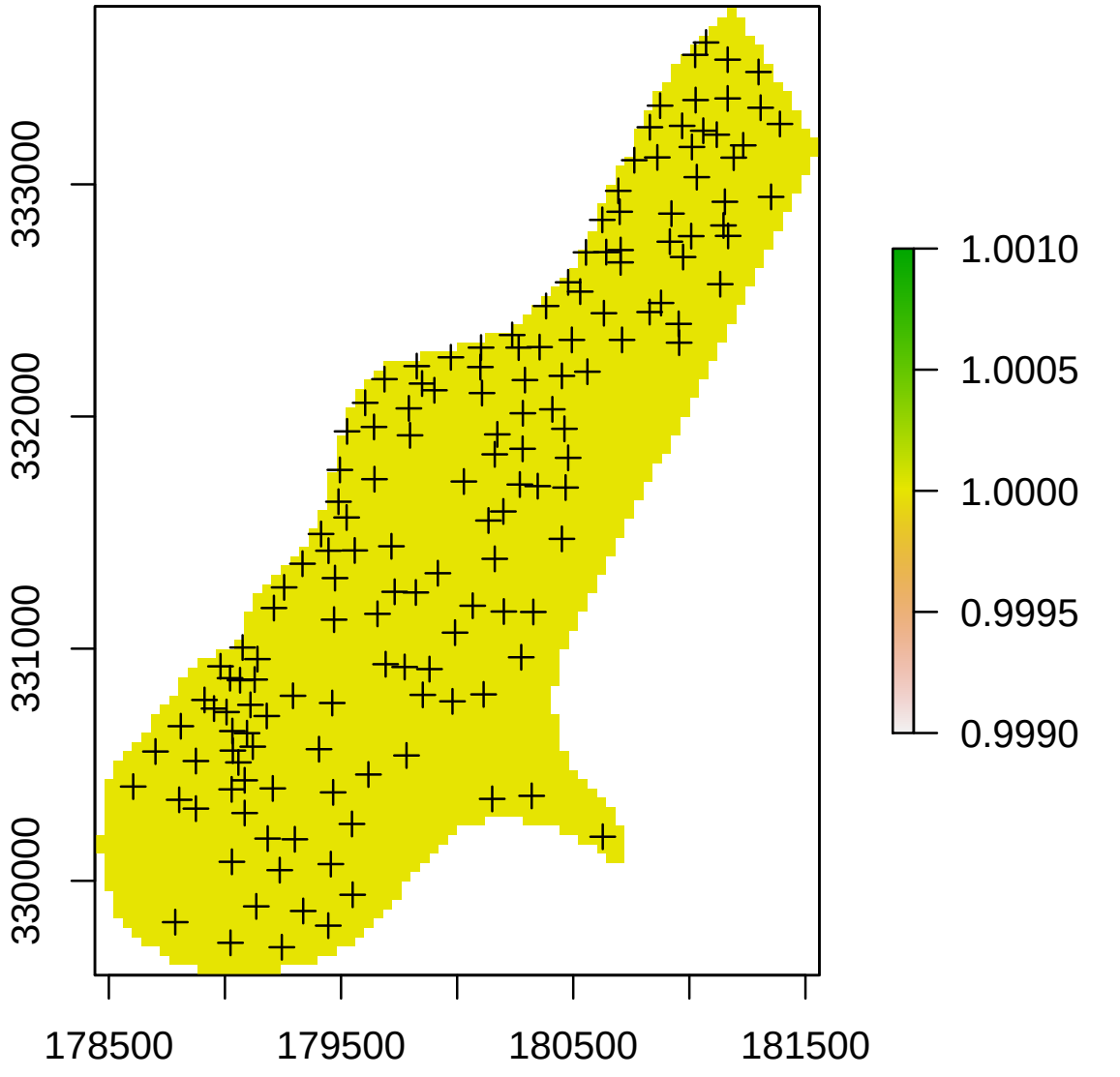

Figure 12: Area of interest and point measurements

gstat: Example

- Function autofitVariogram in package automap provides an automatic **variogram model fitting** procedure

```
v = autofitVariogram(log(zinc) ~ 1, meuse)

v$var_model
##   model      psill    range
## 1   Nug 0.04848089   0.0000
## 2   Sph 0.58754741 889.9084
```

gstat: Example

- The **fitted variogram model** is shown below -

```
plot(v)
```

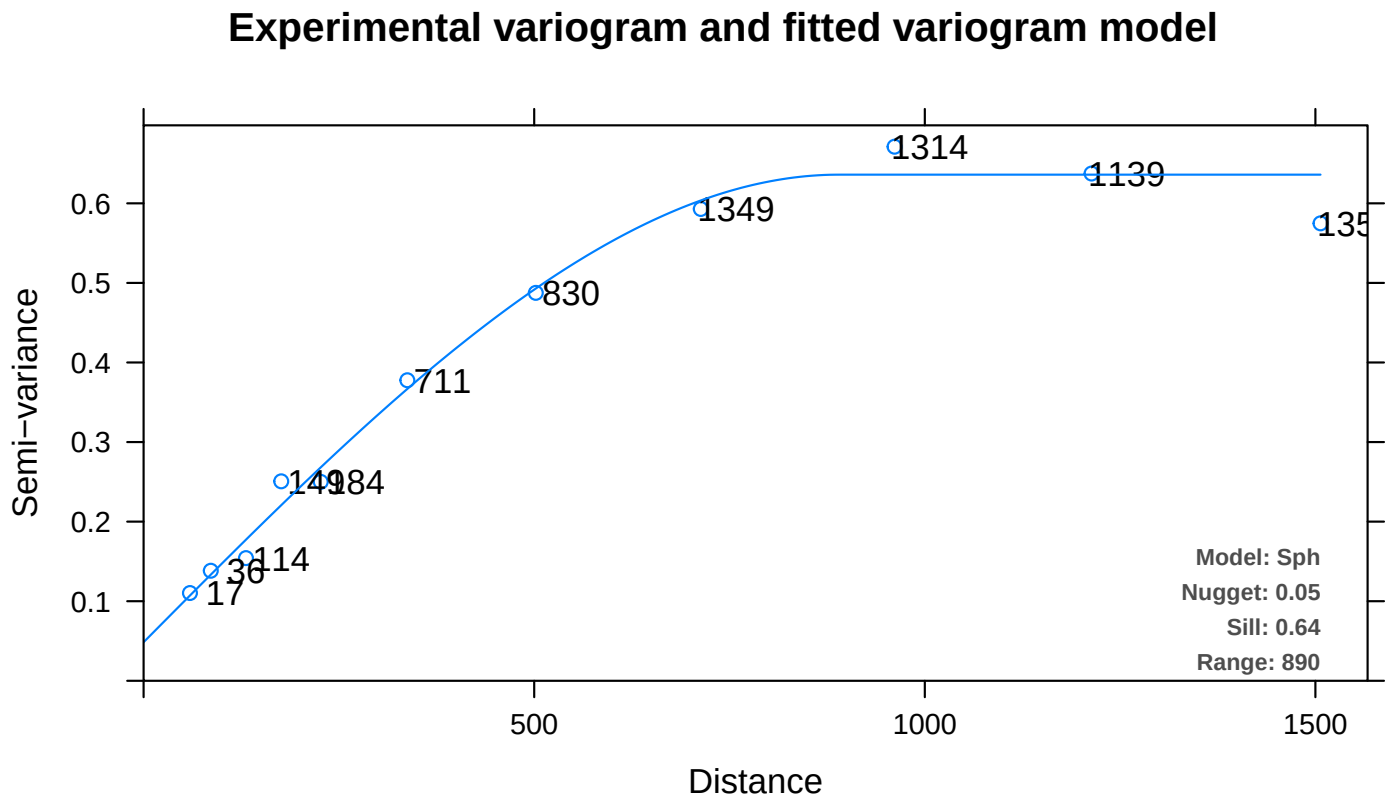

Figure 13: Fitted variogram model

gstat: Example

- Function gstat creates a **geostatistical model** object (class gstat)
- The object holds **all necessary information** for making predictions -
  - The **model definition** (formula, model)
  - The **calibration data** (data)

```
g = gstat(
  formula = log(zinc) ~ 1,
  model = v$var_model,
  data = meuse
)
```

gstat: Example

- Interpolation requires the **geostatistical model** and the **new locations** -

```
predicted = interpolate(grid, g)
## [using ordinary kriging]
```

- Plotting the result -

```
plot(predicted)
contour(predicted, add = TRUE)
```

gstat: Example

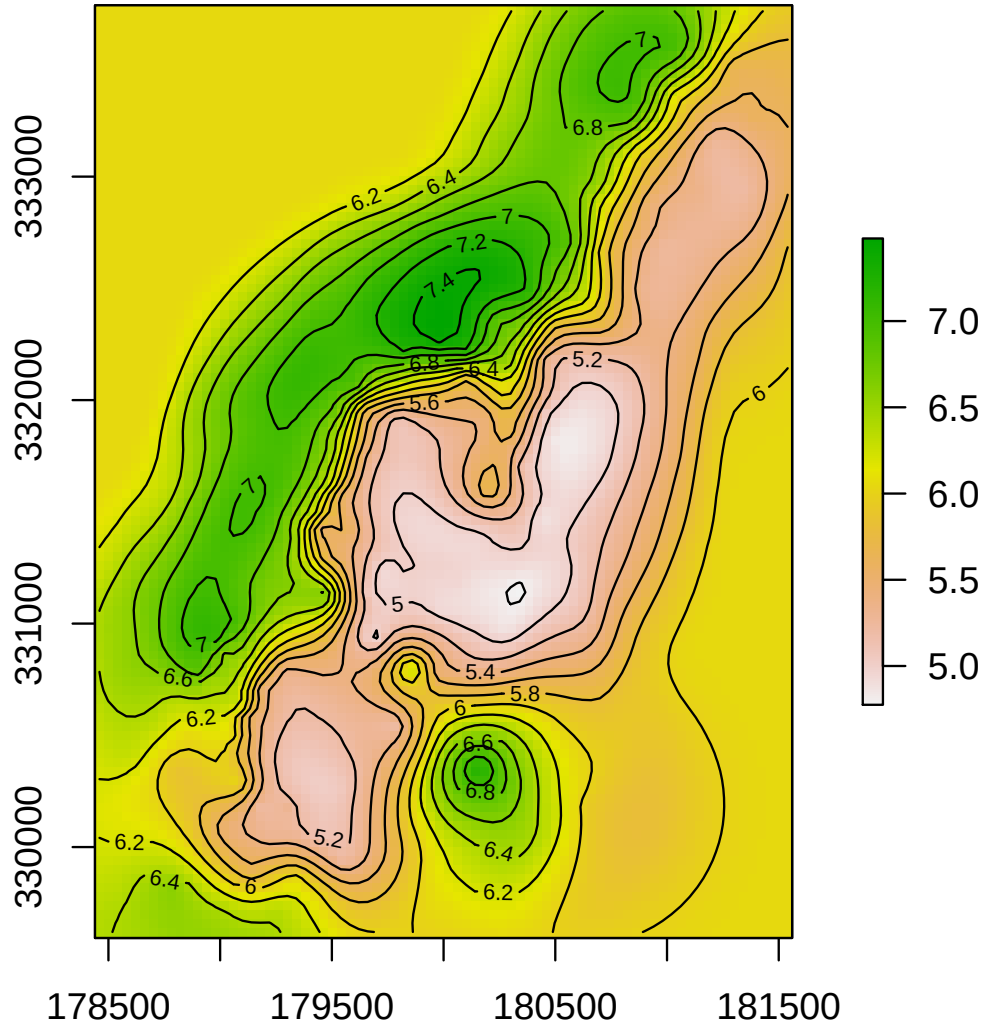

Figure 14: Predicted Zinc concentration, using Ordinary Kriging

gstat: Example

- We are usually interested in predictions within an area of interest rather than the entire rectangular extent of the "template" raster
- The `mask` function can be used to "erase" the irrelevant area

```
predicted = mask(predicted, grid)
```

gstat: Example

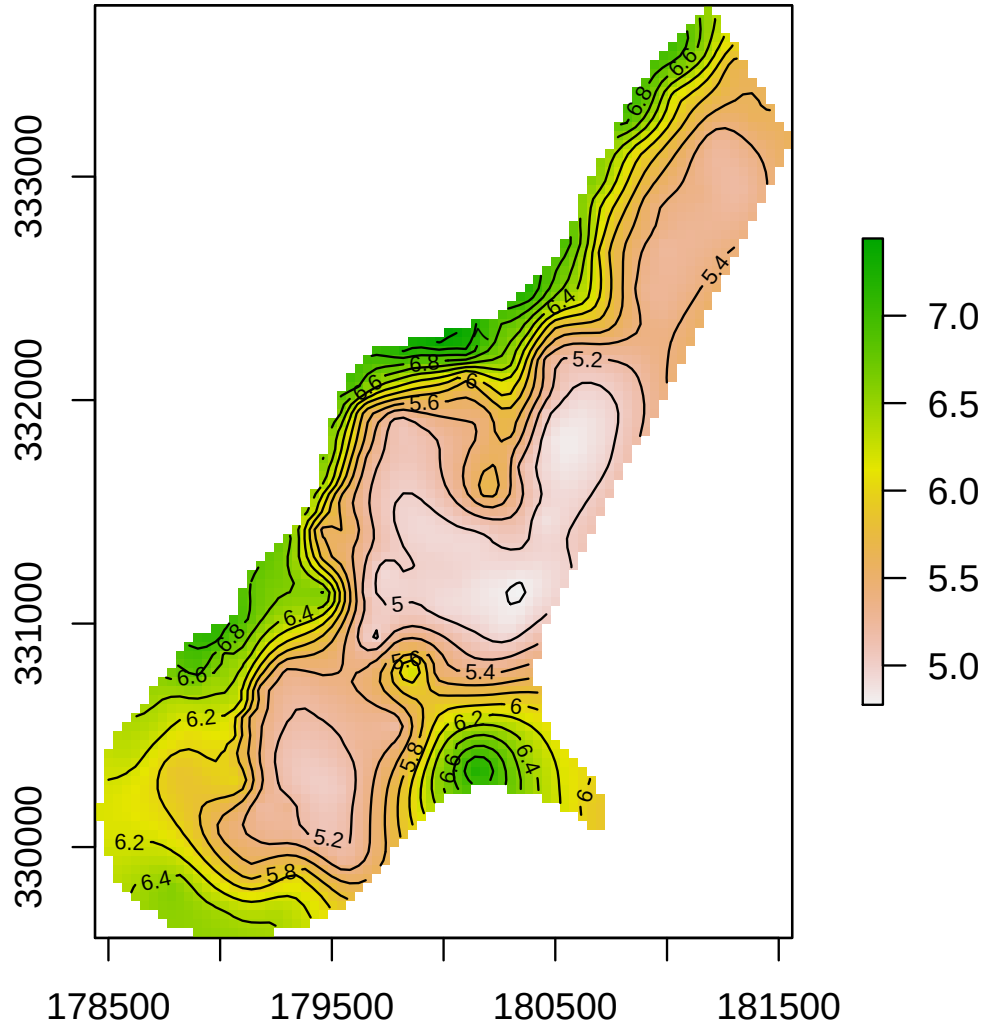

Figure 15: Predicted Zinc concentration, using Ordinary Kriging

`spdep`: Spatial dependence modelling

- Modelling with spatial weights -
  - Building neighbour lists and spatial weights
  - Tests for spatial autocorrelation for areal data (e.g. Moran's I)
  - Spatial regression models (e.g. SAR, CAR)

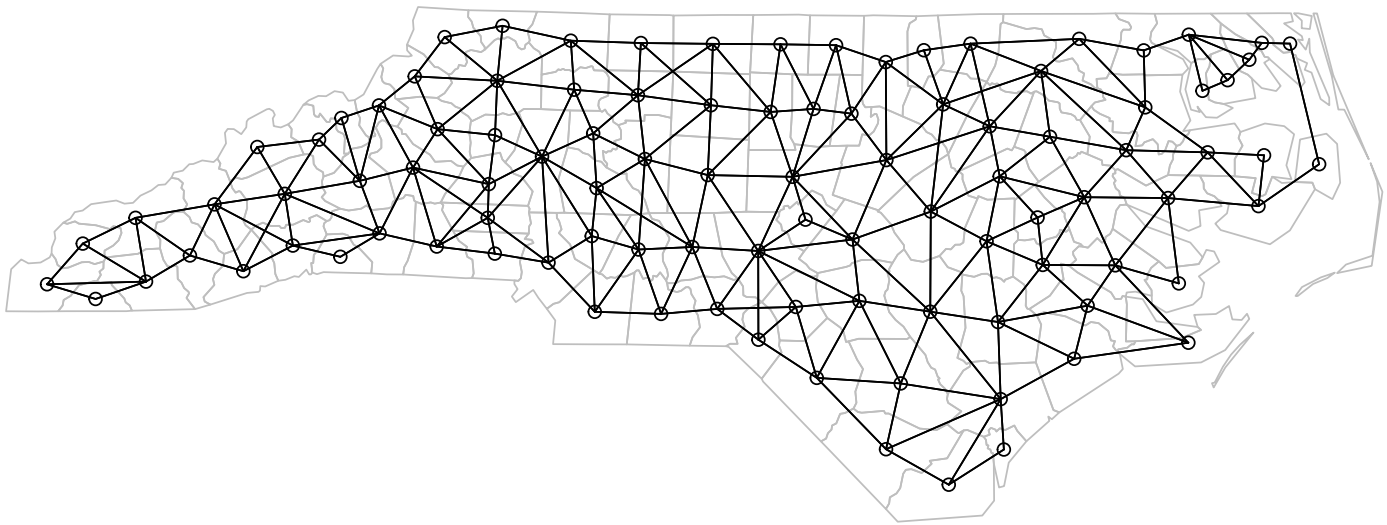

Figure 16: Neighbours list based on regions with contiguous boundaries

spdep: Example

- Loading the North Carolina dataset, see `?spData::nc.sids`

```
file = system.file("shape/nc.shp", package = "sf")
nc = st_read(file)
```

- Calculating Sudden Infant Death (SID) rate

```
nc$rate = nc$SID79 / nc$BIR79
```

spdep: Example

```
plot(nc[, "rate"])
```

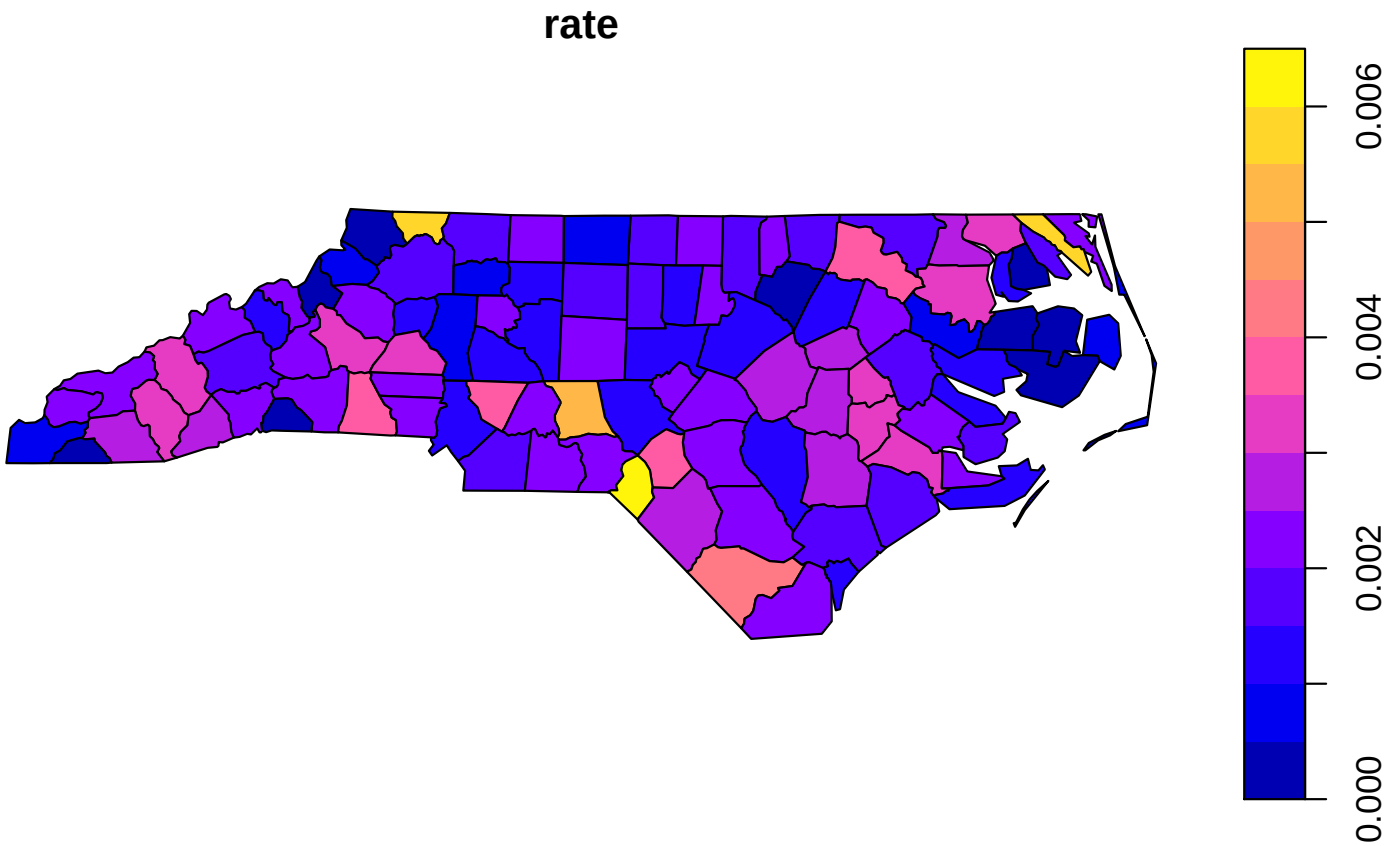

Figure 17: SIDS rate in North Carolina

spdep: Example

- Creating **spatial weights** is a necessary step in using areal data -
  - Defining which areas are concidered **"neighbors"**
  - Assigning **weights** to the identified neighbor links
- A **common approach** is to give a weight of $1/n$ to all n areas that share a common boundary with given focal area, and 0 to all other areas
- Function `poly2nb` can be used to define the neighbors -

```
library(spdep)
nc = as(nc, "Spatial")
nb = poly2nb(nc)
```

spdep: Example

```
summary(nb)
## Neighbour list object:
## Number of regions: 100
## Number of nonzero links: 490
## Percentage nonzero weights: 4.9
## Average number of links: 4.9
## Link number distribution:
##
##  2  3  4  5  6  7  8  9
##  8 15 17 23 19 14  2  2
## 8 least connected regions:
## 4 21 45 56 77 80 90 99 with 2 links
## 2 most connected regions:
## 39 67 with 9 links
```

spdep: Example

```
plot(nc, border = "grey")
plot(nb, coordinates(nc), add = TRUE, col = "black")
```

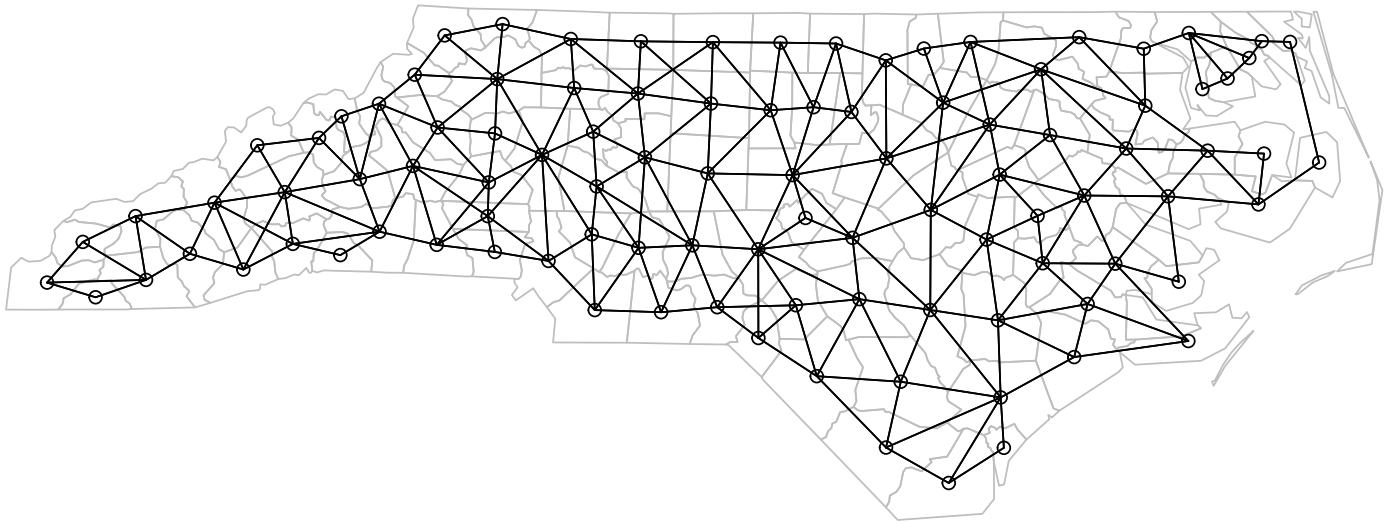

Figure 18: Neighbours list based on regions with contiguous boundaries

spdep: Example

- Then function `nb2listw` can then be used to calculate the neighbor weights -

```
nbw = nb2listw(nb)
```

spdep: Example

```
summary(nbw)
## Characteristics of weights list object:
## Neighbour list object:
## Number of regions: 100
## Number of nonzero links: 490
## Percentage nonzero weights: 4.9
## Average number of links: 4.9
## Link number distribution:
##
##  2  3  4  5  6  7  8  9
##  8 15 17 23 19 14  2  2
## 8 least connected regions:
## 4 21 45 56 77 80 90 99 with 2 links
## 2 most connected regions:
## 39 67 with 9 links
##
## Weights style: W
## Weights constants summary:
##     n    nn  S0       S1       S2
## W 100 10000 100 44.65023 410.4746
```

spdep: Example

- **Moran's I** is the most common global test for spatial auto-correlation
- Values of I range from -1 to $+1$, with the expected value being $-1/(N-1)$
- **Low** values indicate **negative** spatial autocorrelation
- **High** values indicate **positive** spatial autocorrelation

```
moran.test(nc$rate, nbw)
##
##  Moran I test under randomisation
##
## data:  nc$rate
## weights: nbw
##
## Moran I statistic standard deviate = 2.3625, p-value = 0.009075
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##       0.142750422      -0.010101010       0.004185853
```

Disease mapping

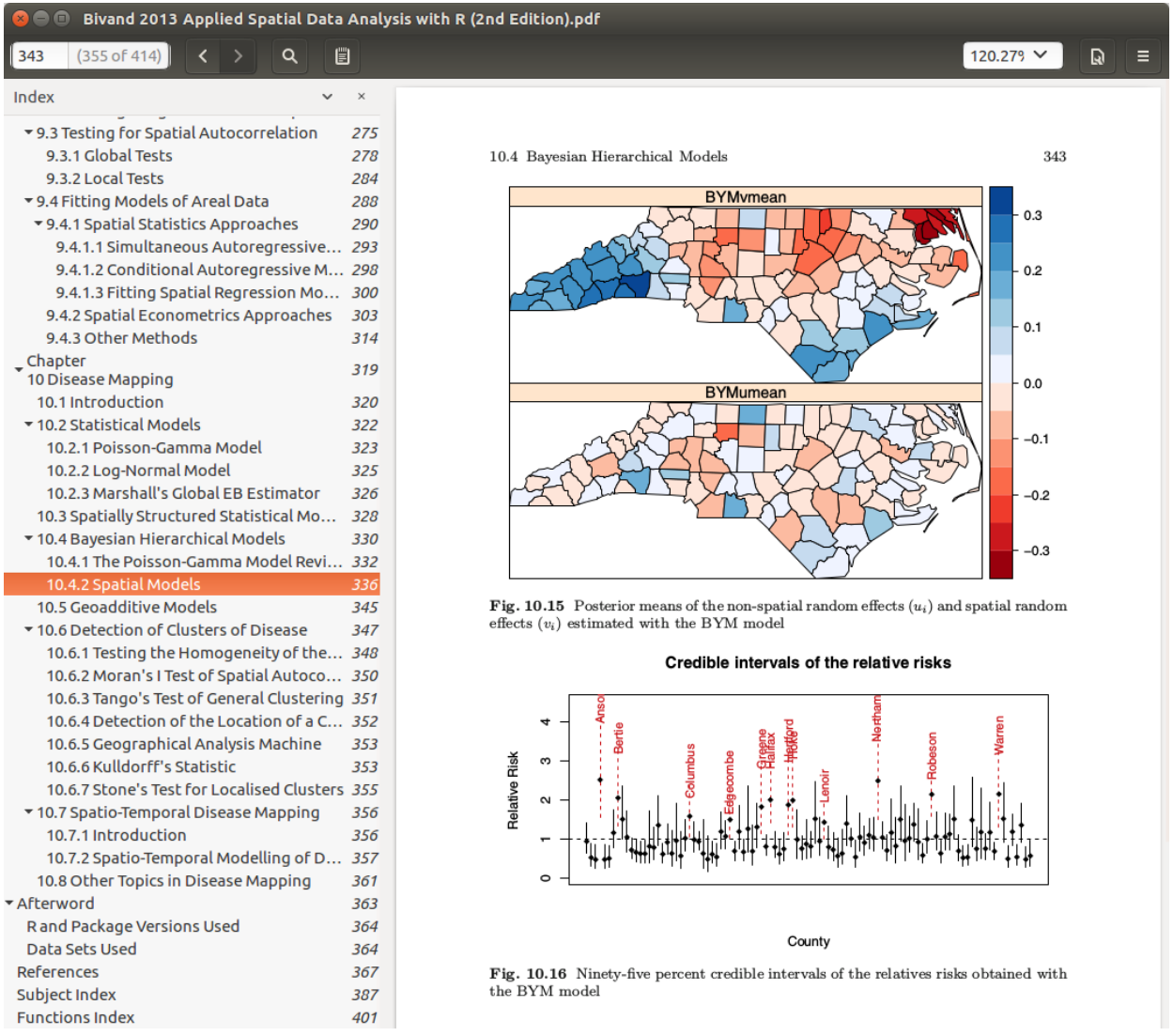

Figure 19: Chapter 10 Disease Mapping, in Bivand *et al.* 2013

`spatstat`: Spatial point pattern analysis

- Techniques for statistical analysis of spatial point patterns, such as -
  - Kernel density estimation
  - Detection of clustering using Ripley's K-function
  - Spatial logistic regression

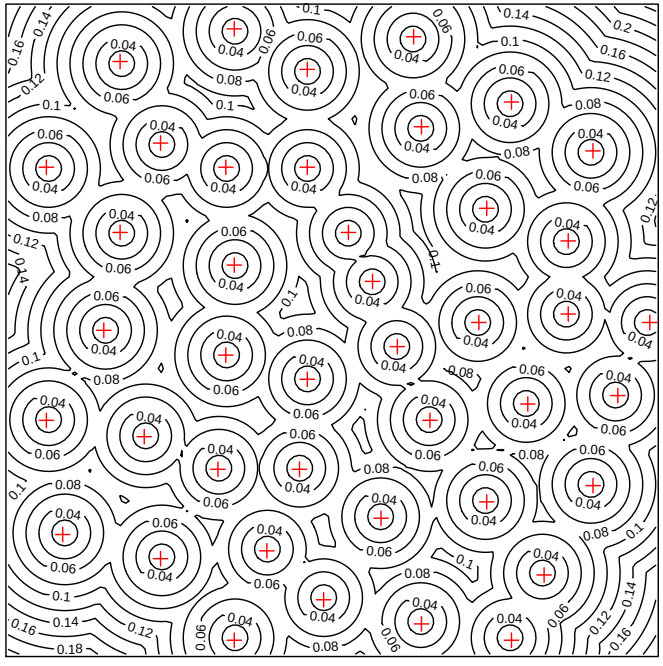

Figure 20: Distance map for the Biological Cells Point Pattern dataset

spatstat: Example

- The spatstat package comes with several sample datasets, including -
  - cells - Biological Cells Point Pattern
  - japanesepines - Japanese Pines Point Pattern
  - redwood - California Redwoods Point Pattern

```
library(spatstat)
cells
## Planar point pattern: 42 points
## window: rectangle = [0, 1] x [0, 1] units
japanesepines
## Planar point pattern: 65 points
## window: rectangle = [0, 1] x [0, 1] units (one unit = 5.7 metres)
redwood
## Planar point pattern: 62 points
## window: rectangle = [0, 1] x [-1, 0] units
```

spatstat: Example

```
plot(cells)
plot(japanesepines)
plot(redwood)
```

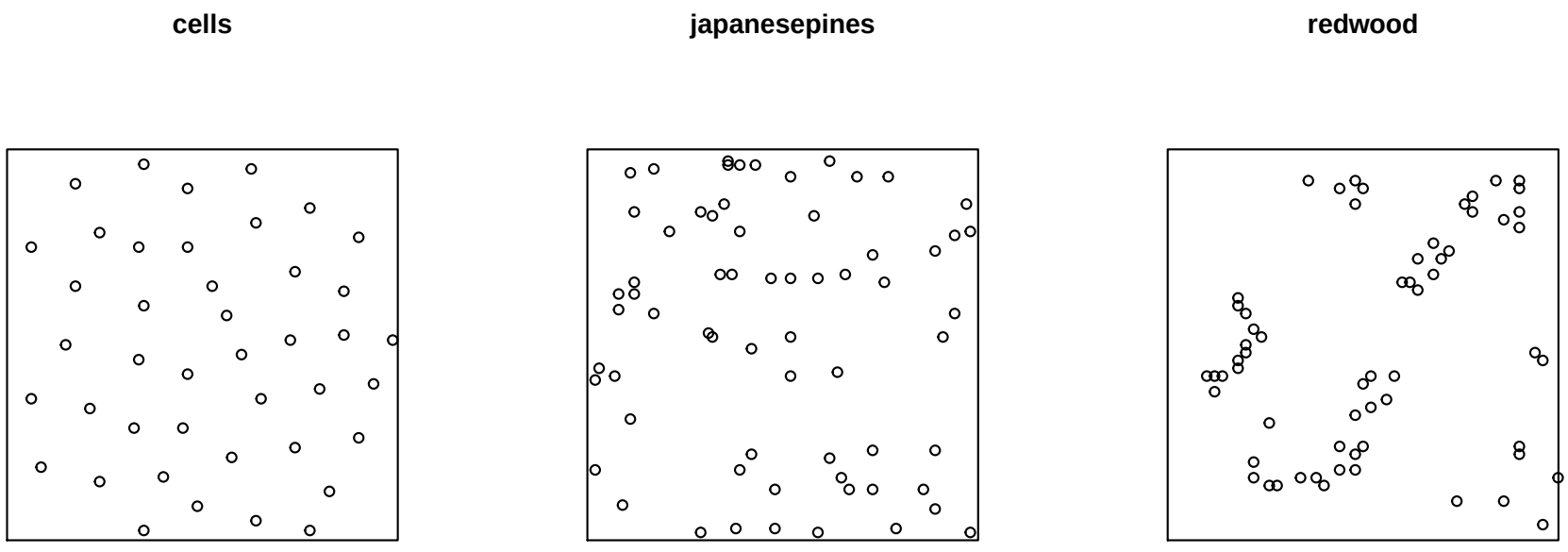

Figure 21: Distance map for the Biological Cells Point Pattern dataset

spatstat: Example

- **Ripley's K-function** measures the expected number of random points within a distance **r** of a typical random point of a point pattern **X**

```
r = seq(0, sqrt(2)/6, by = 0.005)
Kcells = envelope(
  cells,
  fun = Kest,
  r = r,
  nsim = 99
  )
## Generating 99 simulations of CSR  ...
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1
## 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
## 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
##
## Done.
```

spatstat: Example

```
Kpines = envelope(
  japanesepines,
  fun = Kest,
  r = r,
  nsim = 99
  )
Kredwood = envelope(
  redwood,
  fun = Kest,
  r = r,
  nsim = 99
  )
```

spatstat: Example

```
plot(Kcells)
plot(Kpines)
plot(Kredwood)
```

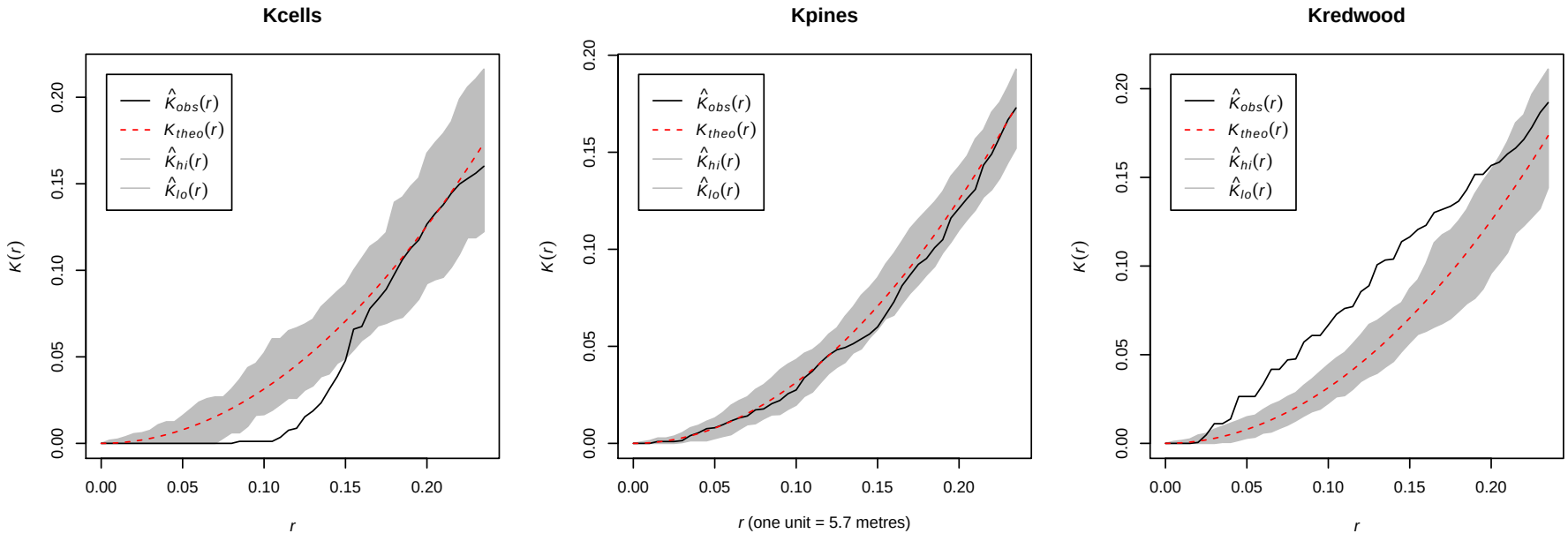

Figure 22: Distance map for the Biological Cells Point Pattern dataset

`osmdata`: Access to OpenStreetMap data

- Accessing OpenStreetMap (OSM) data using the Overpass API

```
library(osmdata)

q = opq(bbox = "Beer-Sheva, Israel")
q = add_osm_feature(q, key = "highway")
dat = osmdata_sf(q)
lines = dat$osm_lines
pol = dat$osm_polygons
pol = st_cast(pol, "MULTILINESTRING")
pol = st_cast(pol, "LINESTRING")
lines = rbind(lines, pol)
lines = lines[, c("osm_id", "highway")]
lines = st_transform(lines, 32636)

plot(lines)
```

`osmdata`: Access to OpenStreetMap data

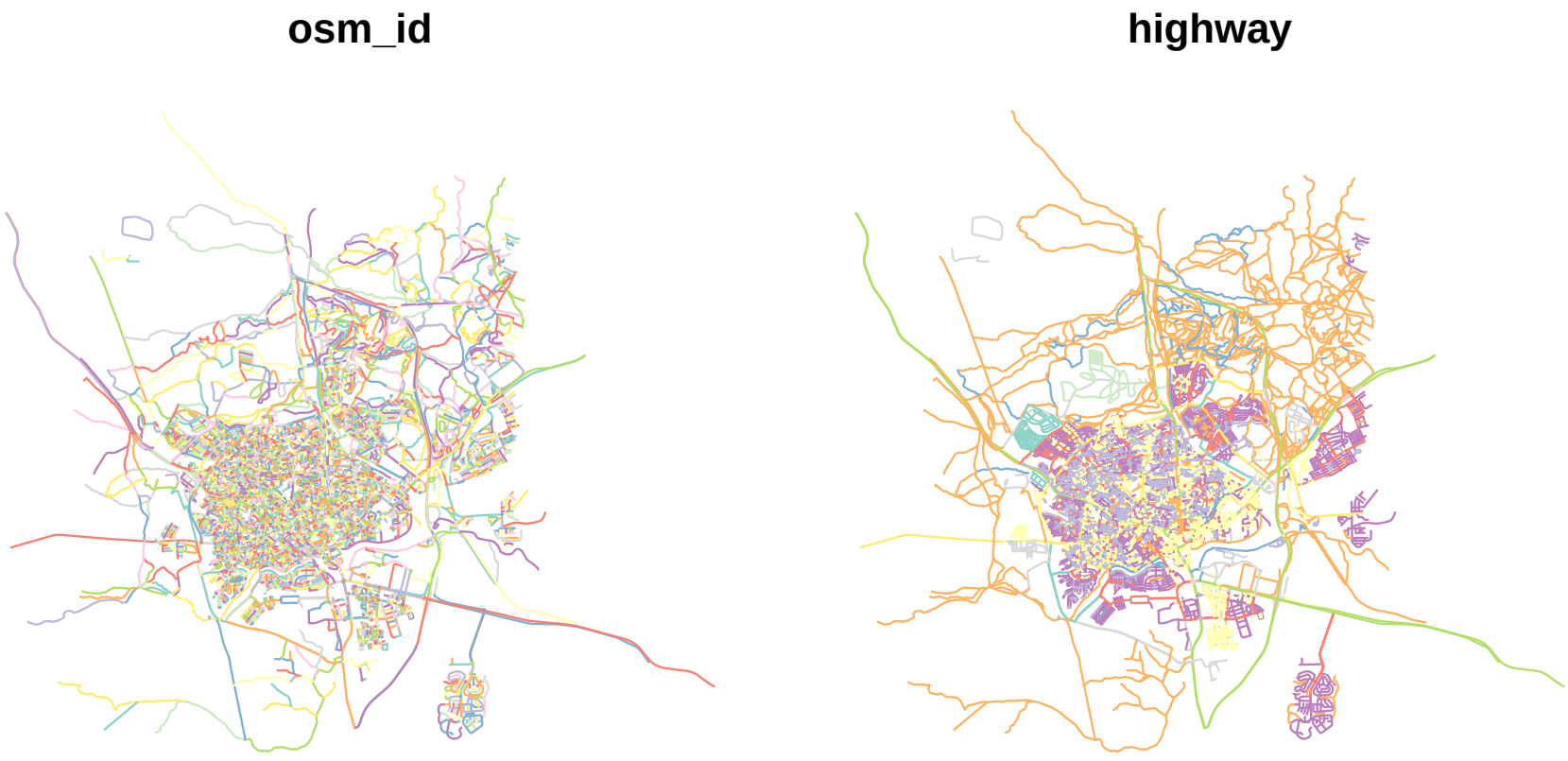

Figure 23: Beer-Sheva road network

ggplot2, ggmap: Visualization

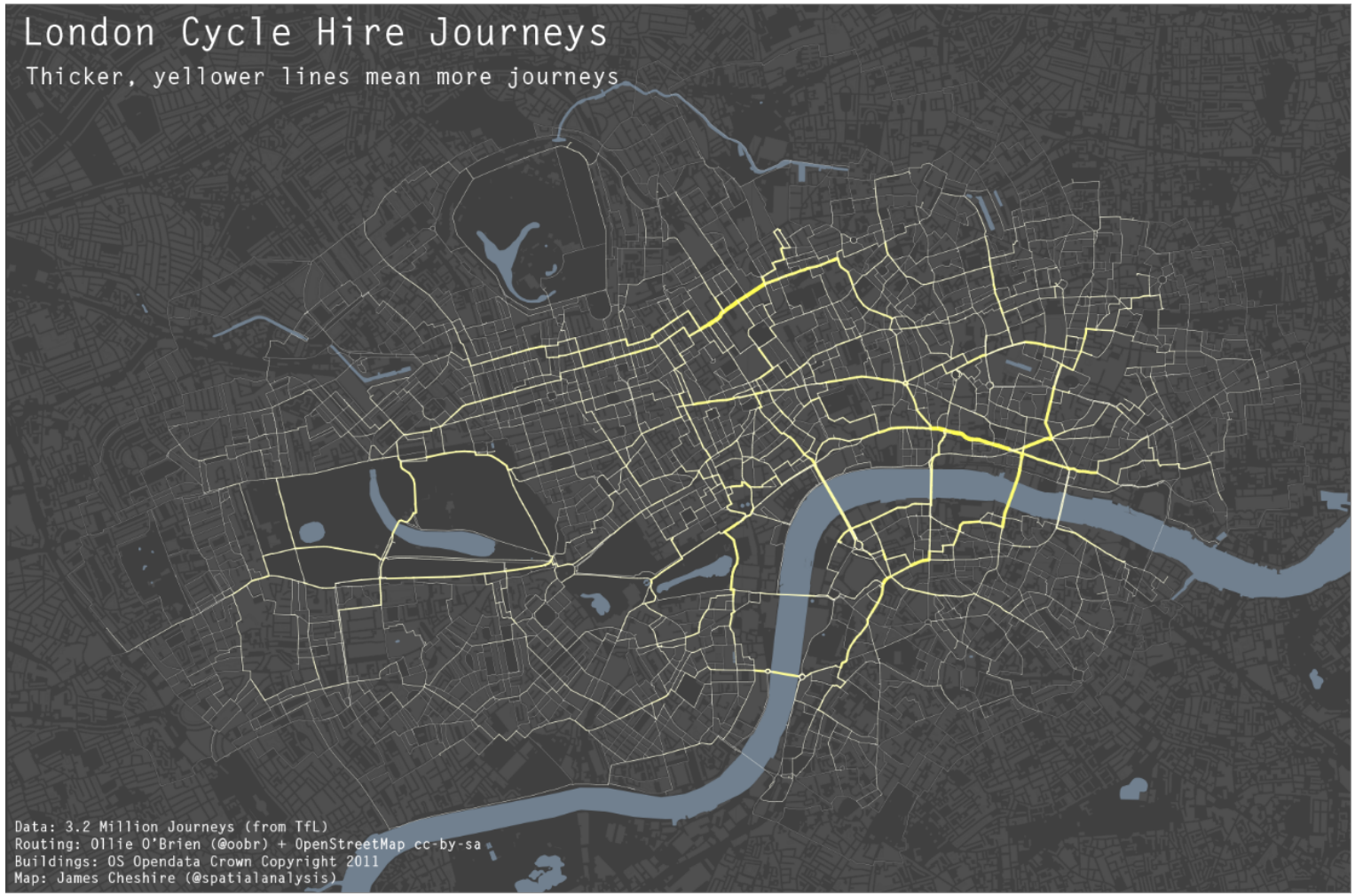

Figure 24: London cycle hire journeys with ggplot2[5]

[5] http://spatial.ly/2012/02/great-maps-ggplot2/

ggplot2, ggmap: Visualization

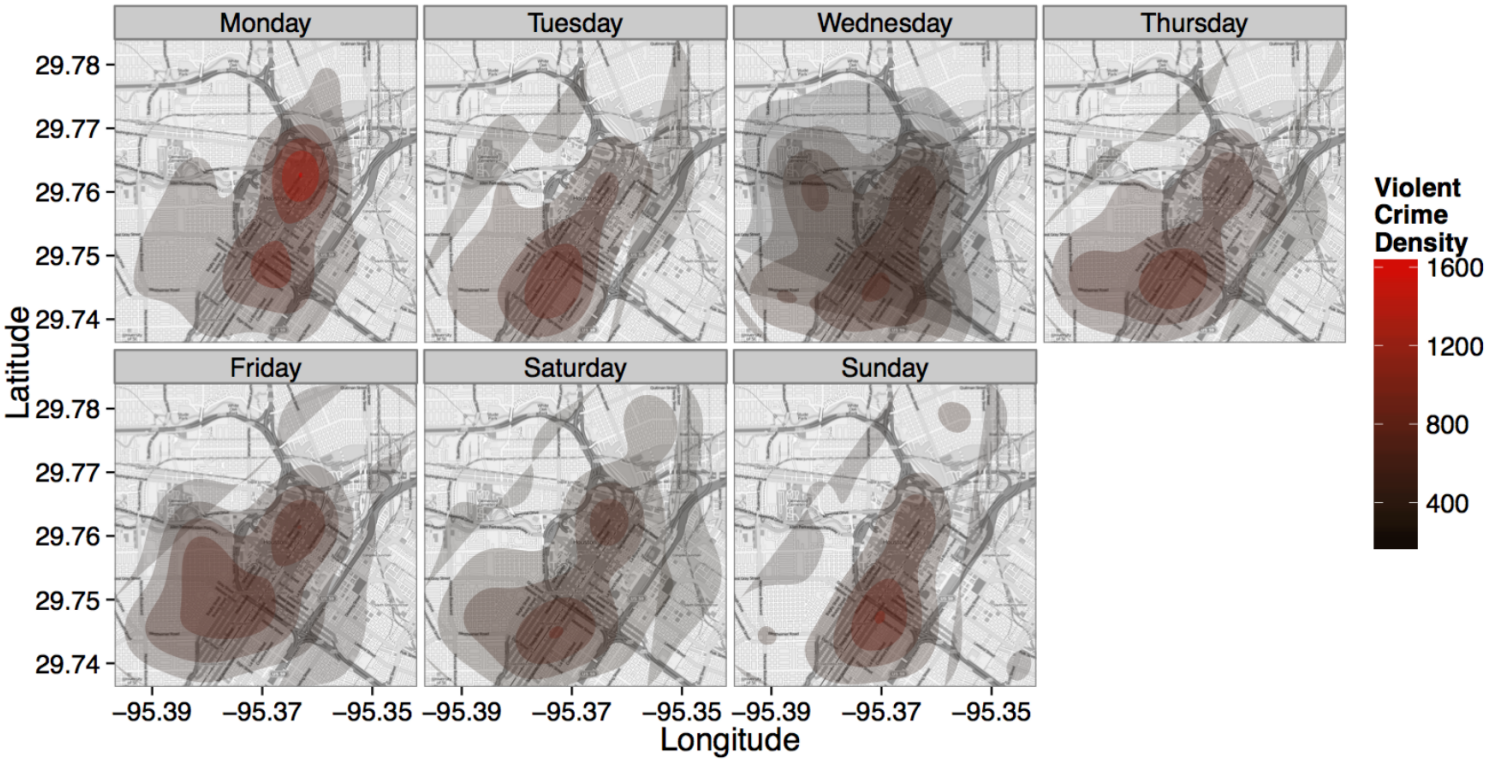

Figure 25: Crime density by day with ggplot2[6]

[6] http://spatial.ly/2012/02/great-maps-ggplot2/

`leaflet`, `mapview`: Web mapping

- Packages `leaflet` and `mapview` provide methods to produce **interactive maps** using the Leaflet JavaScript library
- Package `leaflet` gives more low-level control
- Package `mapview` is a wrapper around `leaflet`, automating addition of useful features -
  - Commonly used **basemaps**
  - **Color scales** and **legends**
  - **Labels**
  - **Popups**

`mapview`: Example

- **Function `mapview`** produces an interactive map given a spatial object
  - `zcol="..."` specifies the **attribute** used for symbology
  - `legend=TRUE` adds a **legend**

```
library(mapview)

states = st_read("data/us-states.geojson")

mapview(states, zcol = "density", legend = TRUE)
```

mapview: Example

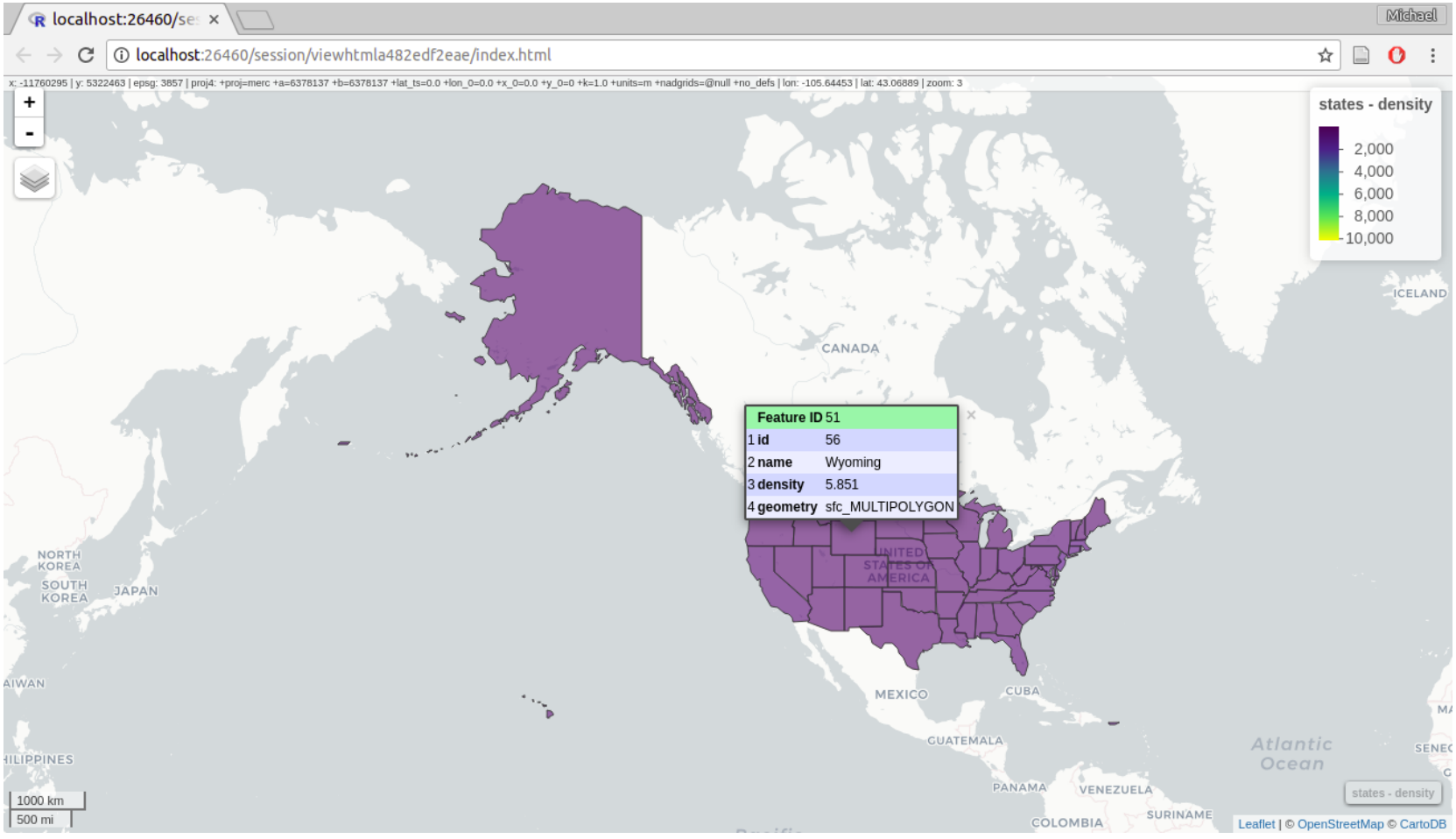

Figure 26: Intractive `mapview` map

`leaflet`: Example

- **The `leaflet` package** can be used to produce similar interactive maps, but with finer control
- Defining a **color scale** function with `colorBin`

```
pal = colorBin(
  palette = "YlOrRd",
  domain = states$density,
  bins = c(0, 10, 20, 50, 100, 200, 500, 1000, Inf)
)
```

- The color scale function accepts a **numeric value** and returns a **color code** -

```
pal(153)
## [1] "#FD8D3C"
```

`leaflet`: Example

- Creating **HTML popups**

```
labels = paste0(
    "<b>",
    states$name,
    "</b><br/>",
    format(round(states$density, 2), nsmall = 2),
    " people / mi<sup>2</sup>"
  ) %>%
  lapply(htmltools::HTML)
```

`leaflet`: Example

- Initiating **map object** with `leaflet`
- Setting **initial extent** with `setView`
- Adding **tile layer** with `addProviderTiles`

```
leaflet(states) %>%
  setView(-96, 37.8, 4) %>%
  addProviderTiles("CartoDB.DarkMatterNoLabels") %>%
```

leaflet: Example

- Adding **polygonal layer** with addPolygons

```
addPolygons(
  fillColor = ~pal(density),
  weight = 2,
  opacity = 1,
  color = "white",
  dashArray = "3",
  fillOpacity = 0.7,
  highlight = highlightOptions(
    weight = 5,
    color = "#666",
    dashArray = "",
    fillOpacity = 0.7,
    bringToFront = TRUE
    ),
  label = labels,
  labelOptions = labelOptions(
    style = list("font-weight" = "normal", padding = "3px 8px"),
    textsize = "15px",
    direction = "auto"
    )
  ) %>%
```

leaflet: Example

- Adding **legend** with addLegend

```
addLegend(
  pal = pal,
  values = ~density,
  opacity = 0.7,
  title = NULL,
  position = "bottomright"
  )
```

leaflet: Example

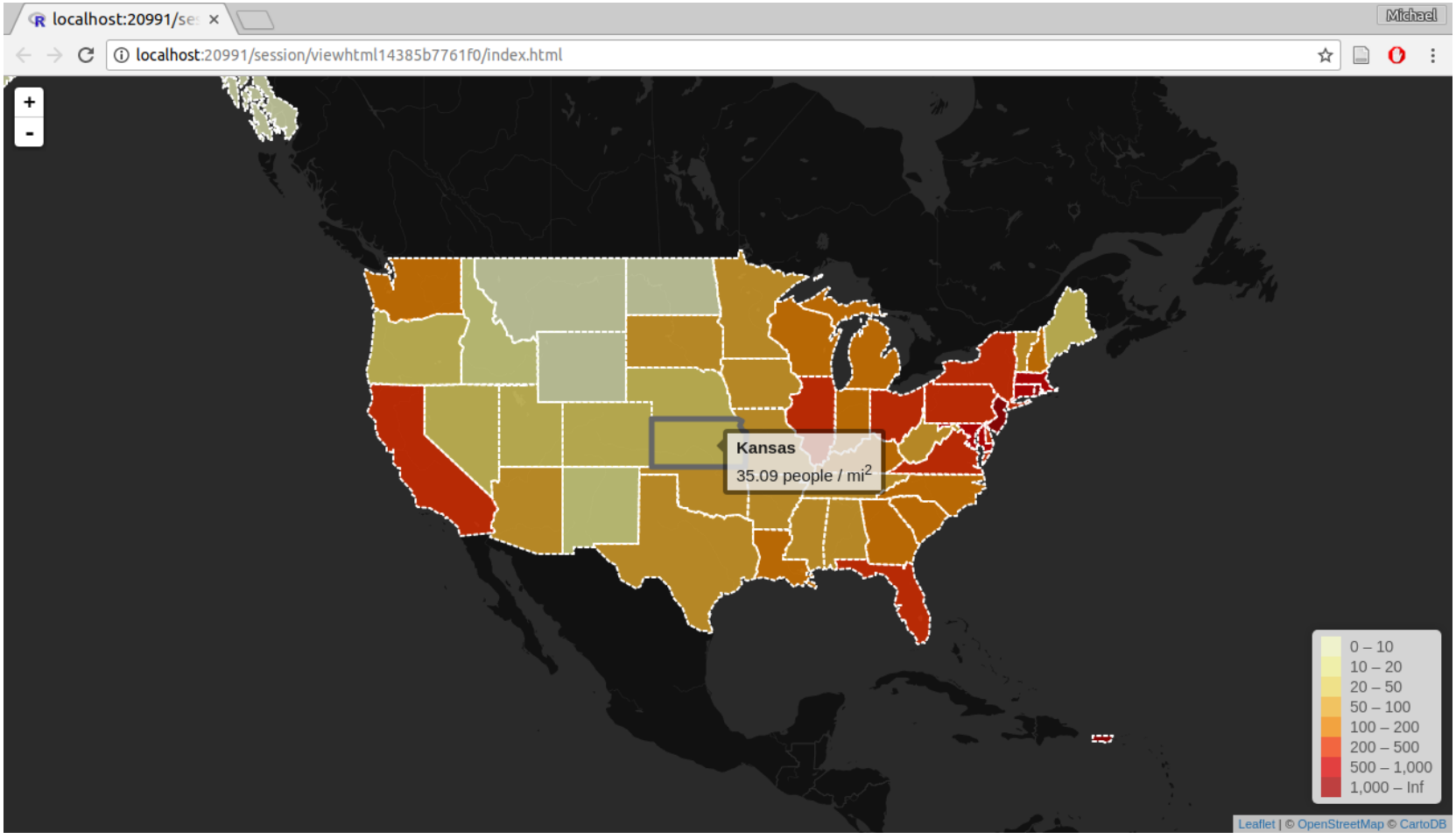

Figure 27: Intractive leaflet map

Other useful packages

- Geometry
  - `maptools`, `dggridR`
- Routing
  - `gdistance`, `stplanr`
- Statistics
  - `dbscan`, `gwrr`, `geoR`, `nlme`, `FRK`
- Visualization
  - `rasterVis`
- Web APIs
  - `ggmap`

Books

- **Hierarchical Modeling and Analysis for Spatial Data** (1st ed 2003, 2nd ed. 2014)
- **Model-based Geostatistics** (2007)
- **Applied Spatial Data Analysis with R** (1st ed. 2008, 2nd ed. 2013)
- **A Practical Guide for Geostatistical Mapping** (2009)
- **Spatial Data Analysis in Ecology and Agriculture using R** (2012)
- **Displaying Time Series, Spatial, and Space-Time Data with R** (2014)
- **Learning R for Geospatial Analysis** (2014)
- **An Introduction to R for Spatial Analysis and Mapping** (2015)
- **Spatial Point Patterns: Methodology and Applications with R** (2015)
- **Geocomputation with R** (2018)[7]

[7] https://geocompr.robinlovelace.net/

Online courses and tutorials

- Courses
  - https://mgimond.github.io/Spatial/index.html
  - http://adamwilson.us/RDataScience/index.html
  - http://geog.uoregon.edu/bartlein/courses/geog490/index.html
- Tutorials
  - http://www.nickeubank.com/gis-in-r/
  - http://neondataskills.org/spatial-data-gis/

Thank you for listening!