Spatial analysis with the R Project for Statistical Computing

D G Rossiter
Cornell University
ISRIC-World Soil Information

December 26, 2022

Topics

1. Spatial analysis in R
2. The sf and sp packages: spatial classes
3. The raster and terra packages: complete, efficient "raster" GIS functions
   - e.g., for image processing
4. The stars package for space-time cubes (can also handle raster stacks).
5. External file formats
6. Interfaces with other spatial analysis tools; Coördinate Reference Systems
7. The gstat package: geostatistical modelling, prediction and simulation

Topic: Spatial analysis in R

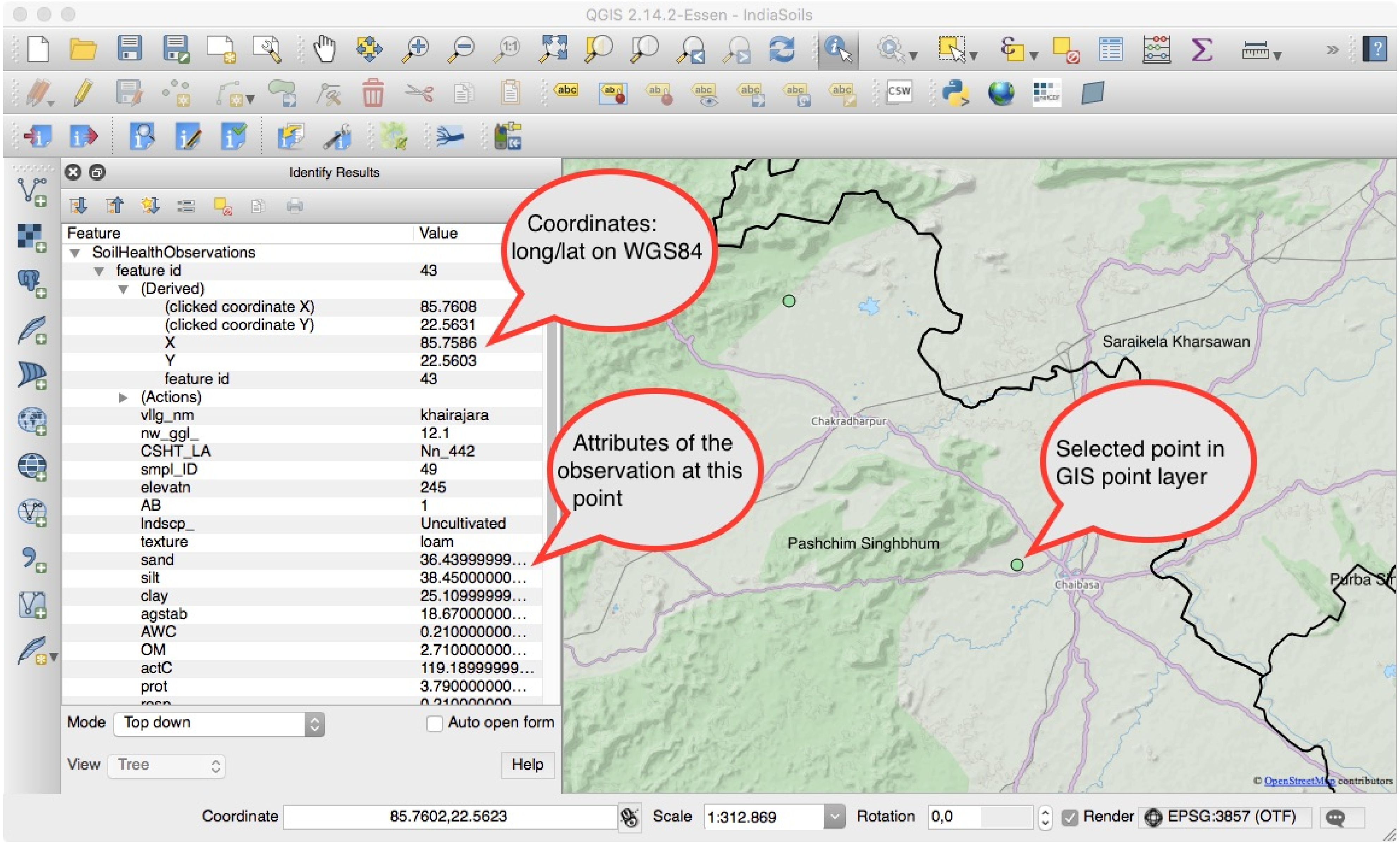
1. Spatial data

2. R approaches to spatial data

Spatial data – definition

- Spatial data have **coördinates**, i.e. known locations
  - The location itself is **spatial information**
  - Often have **attributes** – other information about the spatial object
- Coördinates are **absolute locations** in one, two or three dimensions . . .
  - . . . in some defined **coördinate reference system** (CRS)
  - If referring to real objects referenced to the Earth, must have a defined **projection** and **datum**
- **Gridded data** ("rasters") have an absolute location as an **origin**, other locations are found by **row/column**, each with a **resolution**
  - **One attribute per raster layer**; may be a code linking to a database or a **stack** of co-located attributes.

Spatial objects: location and attributes



Types of spatial objects

Points 0-dimensional

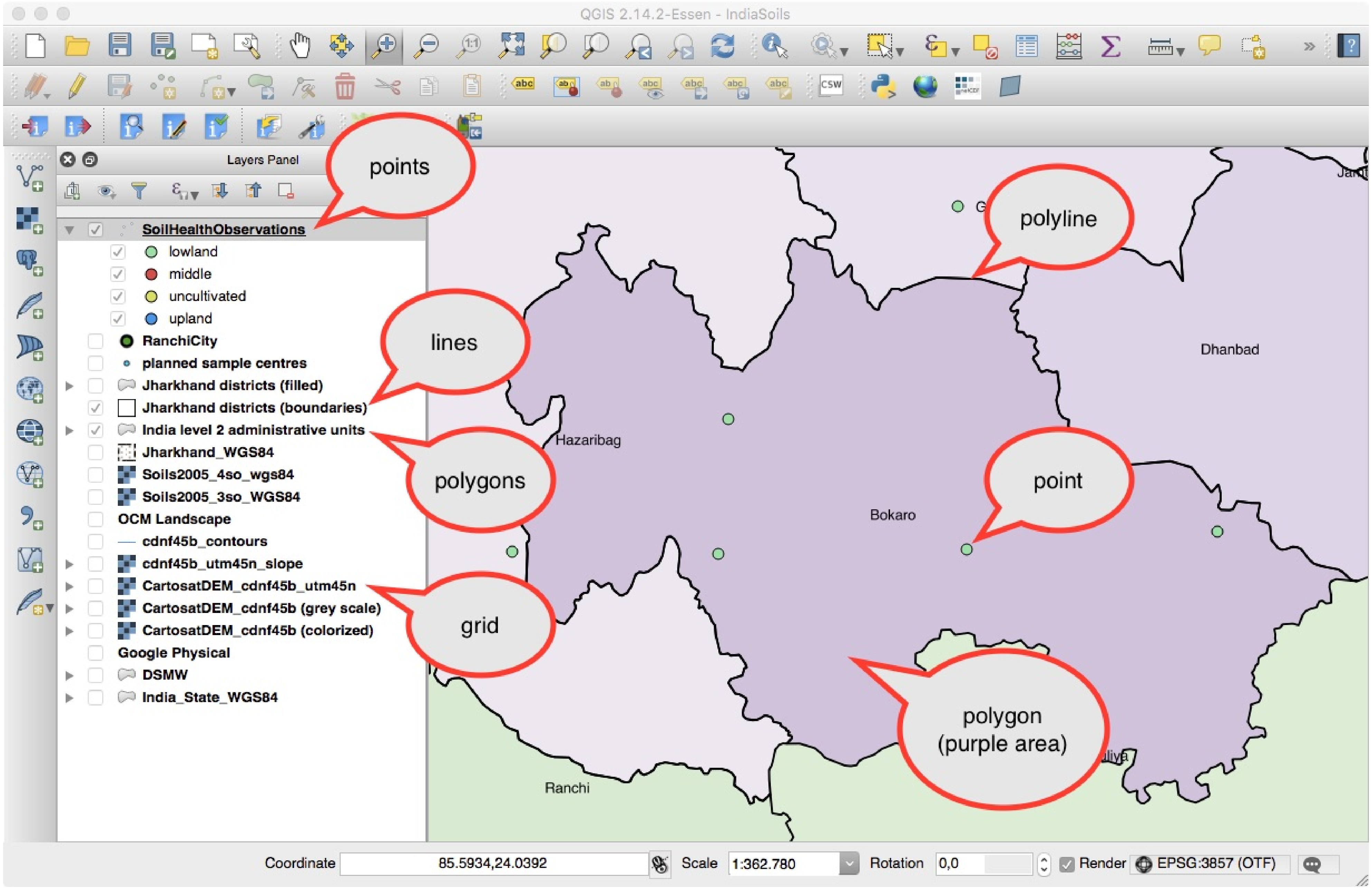
Lines 1-dimensional, defined by points, linked as **polylines**

Polygons area enclosed by connected polylines

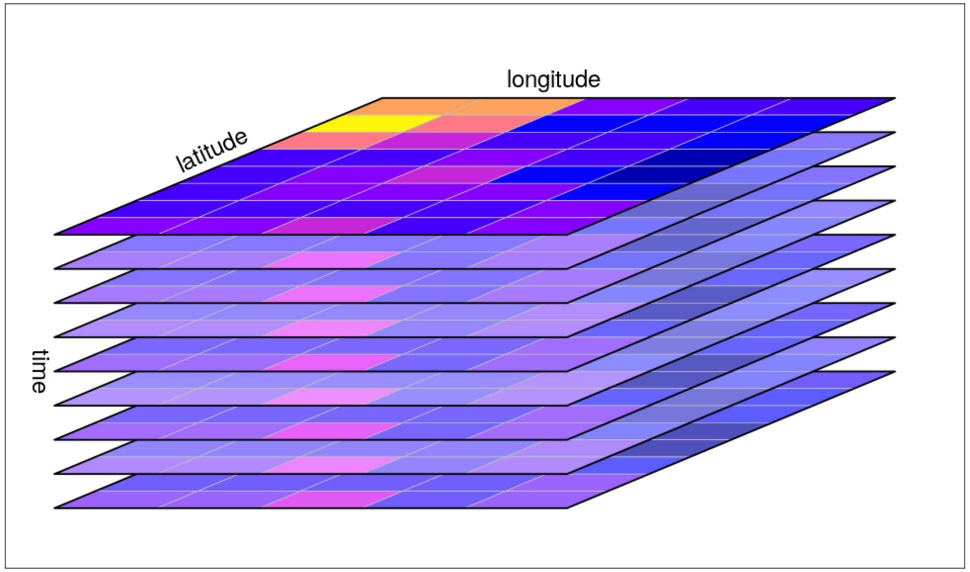
Curves defined by control points and piecewise polynomials of the coördinates

Grid cells "rasters"; (usually regular) tesselation of space

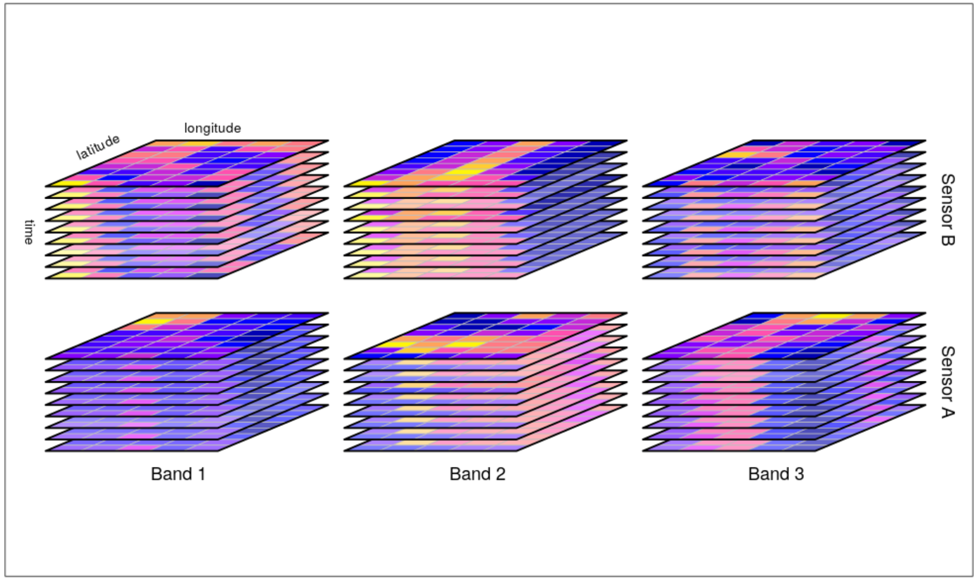
Types of spatial objects



Grid objects



"brick", "cube"; z-dimension is **time**



"hypercube"; w-dimension is the **sensor band**

source: `https://r-spatial.github.io/stars/`

Spatial data – what is special?

- Points are implicitly related by **distance** and **direction** of **separation**
- Polygons are implicitly related by **adjacency**, **containment**, distance between centroids
- Polygons have **shape**, **perimeter**, **compactness** . . .
- Such data requires **different analysis** than non-spatial data
  - e.g. can't assume independence of observations (**spatial dependence/correlation**)
- All objects with the same CRS are implicitly related, can be **overlayed**
- Some analysis is **purely spatial** (e.g., point-pattern analysis)
- They need special **data structures** which recognize the special status of coördinates

R approaches

- No native S classes for these, but S is **extensible** with new classes, methods and packages

- Add-in package which defines **spatial classes and methods**: `sf` (Simple Features), `sp` (Spatial classes), `raster` and `terra`; `stars` (Spatiotemporal Arrays, Raster and Vector Data Cubes)

- Add-in packages to **access external files**: e.g., `ncdf4` (netCDF)

- Add-in packages for **spatial analysis**, e.g., (among many others):
  - `spatial` (Ripley)
  - `gstat` (Pebesma)
  - `spatstat` (Baddeley & Turner): point patterns
  - `RandomFields` (Schlather)
  - `geoR`, `geoRglm` (Ribeiro & Diggle Model-based geostatistics)
  - `circular`: directional statistics
  - ...

Interface to GIS

- Add-in packages:
  - `sf`: interface to Simple Features specification[1]
  - `sp`: interface to `Spatial*` structures
  - `stars`: interface to space-time structures
  - `rgdal`: interface to Geospatial Data Abstraction Library (GDAL) (*deprecated*)
    * Coördinate Reference Systems, transformations
    * Read/write to foreign files
  - `raster`, `terra` read, write, manipulate grid ("raster") data structures
  - `ncdf4` read, write, manipulate netCDF data structures (large raster "bricks")
  - `maptools`: interface to external spatial data structures e.g. shapefiles (*deprecated*)

[1] `https://r-spatial.github.io/sf/`

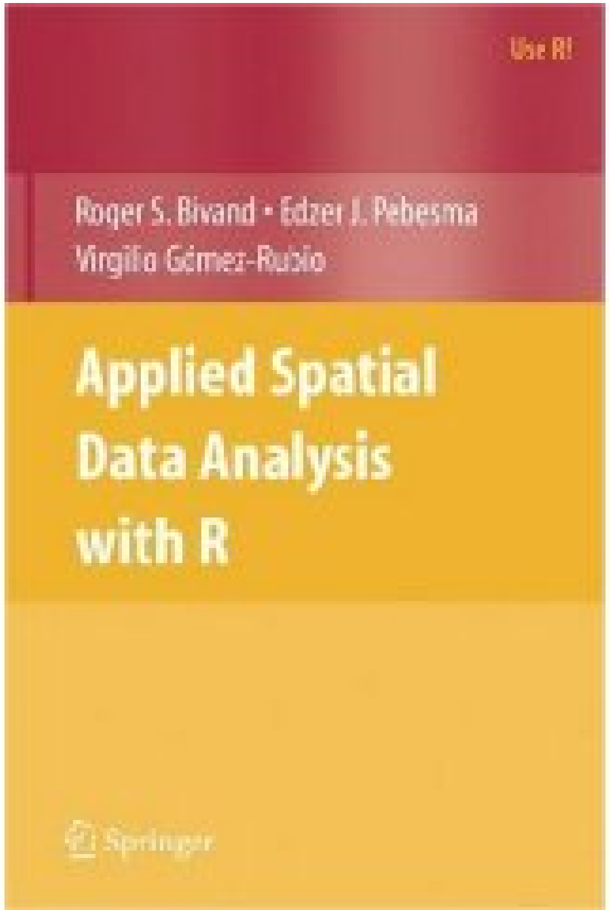
CRAN Task View: Analysis of Spatial Data

`http://cran.r-project.org/web/views/Spatial.html` explains which packages are available:

- Classes for spatial data
- Handling spatial data
- Reading and writing spatial data
- Point pattern analysis
- Geostatistics
- Disease mapping and areal data analysis
- Spatial regression
- Ecological analysis

Textbook with examples and code

Bivand, R. S., Pebesma, E. J., & Gómez-Rubio, V. (2013). *Applied Spatial Data Analysis with R*, 2nd ed.: Springer. `http://www.asdar-book.org/`; ISBN 978-1-4614-7617-7; 978-1-4614-7618-4 (e-book)



Topic: The sf package

- `sf` = "Simple Features"

- Extensive documentation (including tutorials) at the "Simple Features for R"[2] R-spatial project

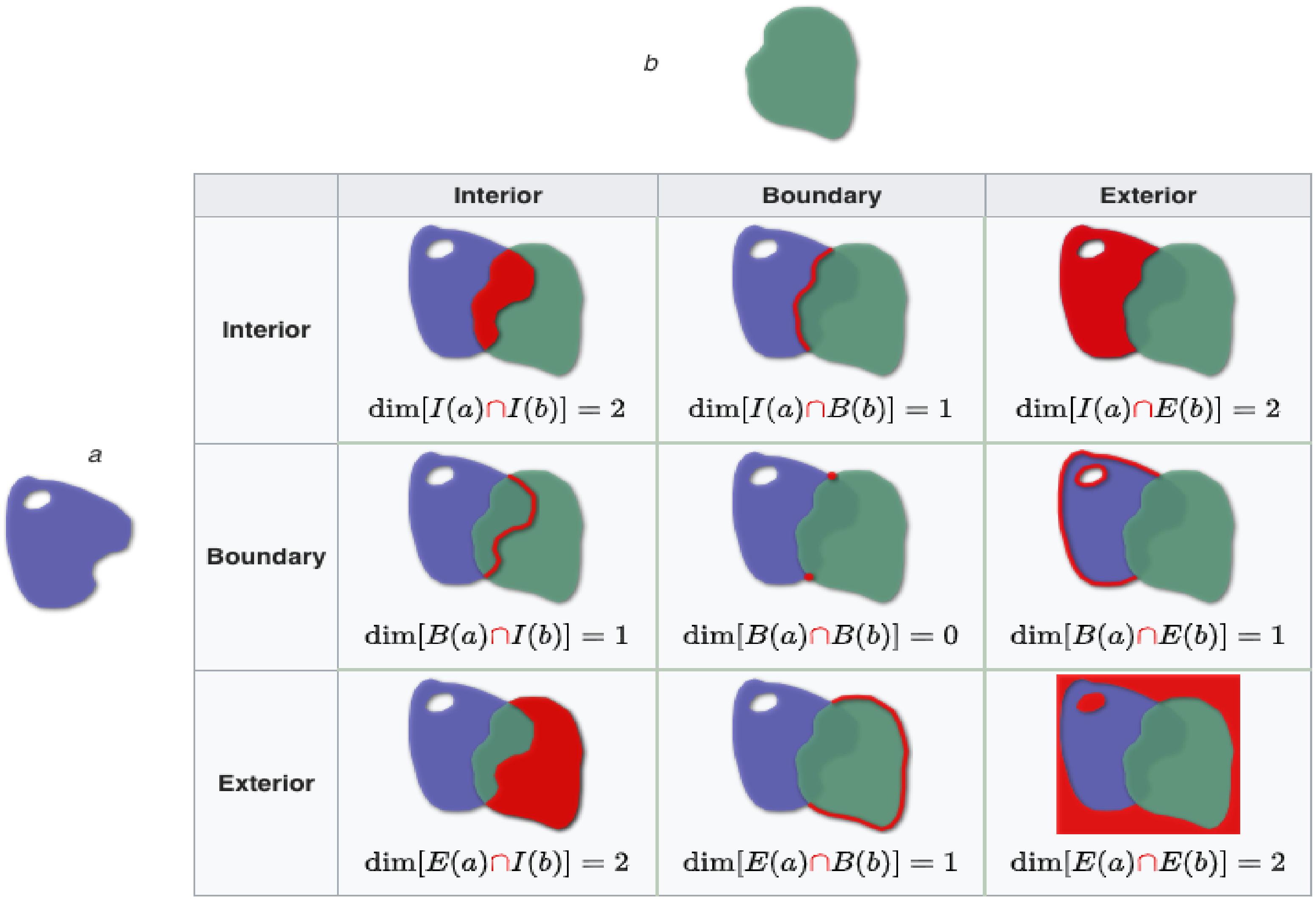
"Simple features ... refers to a formal standard (ISO 19125-1:2004) that describes how **objects in the real world** can be **represented in computers**, with emphasis on the **spatial geometry** of these objects. It also describes how such objects can be **stored in and retrieved from databases**, and which **geometrical operations** should be defined for them."[3]

[2] https://r-spatial.github.io/sf/index.html
[3] https://r-spatial.github.io/sf/articles/sf1.html

Simple features

- specifies a common **storage and access model** of geometries used by GIS
  - e.g., 2D point, line, polygon, multi-point, multi-lines; also 3D
- geometric information contained in a special `geometry` field in a `data.frame` structure
  - does not require a hierarchy of S4 classes (see `sp`, below)
- supports the Dimensionally Extended nine-Intersection Model (DE-9IM), which specifies topological relations
  - replaces `rgeos` "Interface to Geometry Engine - Open Source (GEOS)"

Dimensionally Extended nine-Intersection Model (DE-9IM)



source: `https://en.wikipedia.org/wiki/DE-9IM`

Building spatial objects in sf

- All methods begin with `st_` = "space-time"
- `st_sfc`: "Create `simple feature geometry list column`, set `class`, and add `coordinate reference system` and `precision`"
- `st_crs`: "Specify a Coördinate Reference System" → relates coördinates to the real world

```
st_sfc> pt1 = st_point(c(-76,42)); pt2 = st_point(c(-77,43)) # two points
st_sfc> sfc = st_sfc(pt1, pt2)                          # combine as a spatial object
st_sfc> d = st_sf(data.frame(a=1:2, geom=sfc))          # add attributes
st_sfc> st_crs(d) <- 4326
st_sfc> print(d)                                    # define CRS

Geometry set for 2 features
geometry type:  POINT
dimension:      XY
bbox:           xmin: 0 ymin: 1 xmax: 1 ymax: 1
CRS:            WGS84
POINT (-76 42)
POINT (-77 43)
```

Working with spatial objects in sf

- See the "cheat sheet"[4] and the tutorial introduction[5]
- topology: `st_contains`, `st_covers`, `st_intersects` ...
- operations: `st_intersection`, `st_difference`, `st_crop` ...
- new geometry: `st_centroid`, `st_buffer`, `st_make_grid`, `st_coordinates` ...
- coördinate systems: `st_crs`, `st_transform`

[4] `https://github.com/rstudio/cheatsheets/blob/main/sf.pdf`

[5] `https://r-spatial.github.io/sf/articles/sf1.html`

Topic: External file formats

R can deal with geographic data in “all” file formats.

- vector

- gridded (“raster”)

Vector data structure: ESRI shapefiles

- Proprietory format, but reasonably well-documented[6]

- Geometry + attributes of 2D geometries

- Does not explictly store topology, must be built on-the-fly

[6] https://en.wikipedia.org/wiki/Shapefile

Vector data structure: File Geodatabases

- Originally from ESRI
- Allows storage of multiple data layers; allows very large data layers
- Specification is known, so drivers developed for `GDAL` (`rGDAL` package, QGIS ...)

Grid data structures

- Many standards: GeoTIFF, ASCII, NetCDF (Network Common Data Form, from UCAR [University Corporation for Atmospheric Research])[7] ...
- represented in R as (sparse) matrices, sometimes implicit by the grid topology (e.g., `sp` package)
- `raster` package[8]; now updated as `terra`[9]; both from Robert Hijmans
- `stars` "space-time data structures"; includes space-time **cubes**
- `fields` package from NCAR "curve and function fitting with an emphasis on spatial data and spatial statistics."
- analysis of grids also in `sp`, `spatial`

[7] `https://www.image.ucar.edu/GSP/Software/Netcdf/`
[8] `https://rspatial.org/raster/pkg/index.html`
[9] `https://rspatial.org/terra/pkg/index.html`

Topic: Interfaces with support libraries

Most spatial data is prepared outside of R, usually in a GIS or spreadsheet.

Results of R analyses are often presented outside of R, e.g., as GIS maps. How is information exchanged?

- Import and export
- Projections and Datums
- Transformations

GDAL

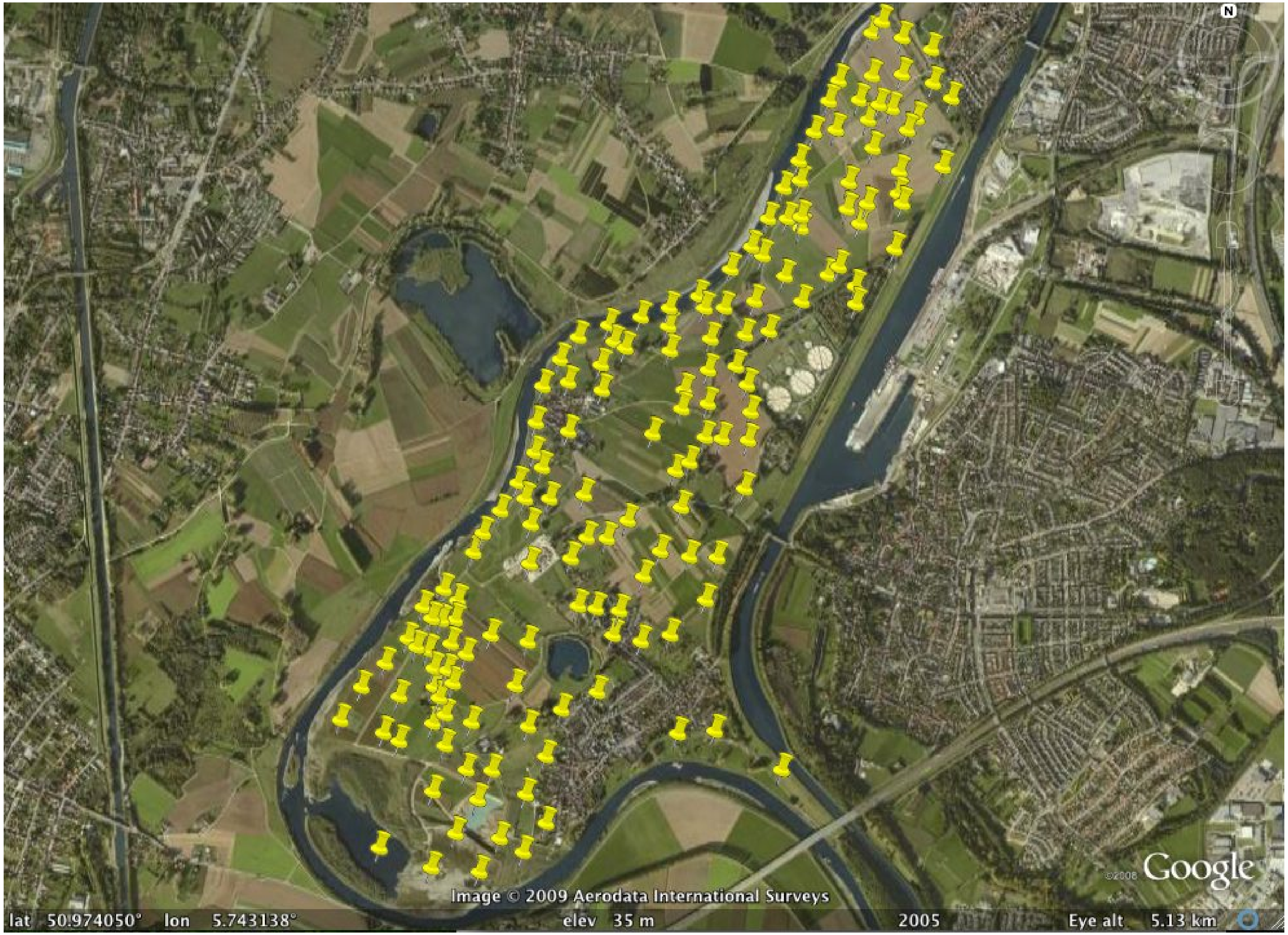
GDAL is the **Geospatial Data Abstraction Library**[10], an open-source **translator** library for geospatial data formats. It is accessed from almost all geospatial programs (GIS, spatial statistics).

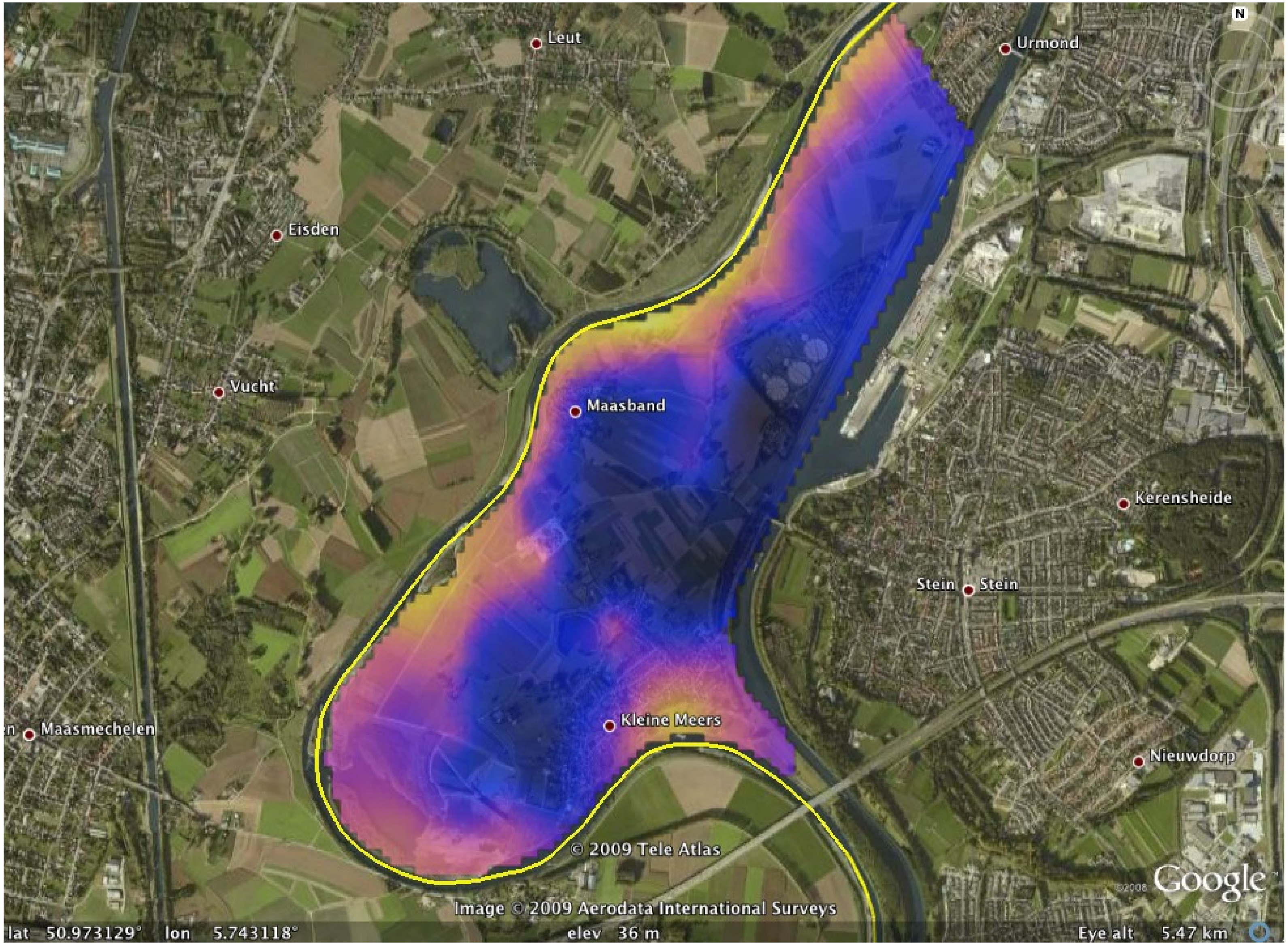
- The `sf` package uses this, it has functions to interact with GDAL not meant to be called directly by users
  - Various `stars` functions, e.g. `read_stars` to read rasters

- R has a *deprecated* `rgdal` package, which uses the `sp` classes.
  - `readGDAL`; `writeGDAL`: Read/write between GDAL grid maps and Spatial objects
  - `readOGR`, `writeOGR`: Read/write spatial vector data using OGR (including **KML** for Google Earth)

[10] `http://www.gdal.org/`

Example Google Earth layers created with `writeOGR`




Leut
Urmond
Eisden
Vucht
Maasband
Kerensheide
Stein
Stein
Maasmechelen
Kleine Meers
Nieuwdorp
© 2009 Tele Atlas
Image © 2009 Aerodata International Surveys
Google
lat 50.973129° lon 5.743118°
elev 36 m
Eye alt 5.47 km

Converting between CRS

- The `st_transform` method of the `sf` R package
- The `spTransform` method of the `rgdal` R package
- implement the PROJ.4 system[11] also found in the GDAL "Geospatial Data Abstraction Library" program[12]
- can convert between projections, backwards and forwards to/from geodetic coördinates
- with or without a datum transformtation

[11] http://trac.osgeo.org/proj/
[12] http://www.gdal.org/

Example – Meuse River soil pollution dataset (NL)

```
> require(sp)
> ## load an example dataset from the sp package
> data(meuse)
> ## convert to a spatial object; we must know which fields represent coordinates
> coordinates(meuse) <- ~x + y
> proj4string(meuse)

[1] NA

>  ## no CRS yet; define the CRS from metadata; first load GDAL
> require(rgdal)
> proj4string(meuse) <- CRS("+init=epsg:28992")
> proj4string(meuse)

    [1] "+init=epsg:28992 +proj=sterea
    +lat_0=52.15616055555555 +lon_0=5.38763888888889 +k=0.9999079
    +x_0=155000 +y_0=463000 +ellps=bessel
    +towgs84=565.417,50.3319,465.552,-0.398957,0.343988,-1.8774,4.0725
    +units=m +no_defs"

> ## EPSG database contained all required parameters
```

. . . continued

```
> ## un-project to geodetic system, also changing the datum
> meuse.wgs84 <- spTransform(meuse, CRS("+proj=longlat +datum=WGS84"))
> proj4string(meuse.wgs84)

 [1] "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"

> ## now this could be forward-projected into, e.g., UTM 31N on the WGS84 datum
> ## we find the appropriate initializtion in the EPSG database
> meuse.wgs84.utm31 <- spTransform(meuse.wgs84, CRS("+init=epsg:32631"))
> proj4string(meuse.wgs84.utm31)

[1] "+init=epsg:32619 +proj=utm +zone=31 +datum=WGS84 +units=m
      +no_defs +ellps=WGS84 +towgs84=0,0,0"

> ## note the origin is implicit in the definition of UTM31N
> ## now go directly from RDH to geographic coords, keeping the ellipsoid
> meuse.rdh.geo <- spTransform(meuse, CRS("+proj=longlat"))
> ## and to UTM31N on that ellipsoid
> meuse.rdh.utm31 <- spTransform(meuse, CRS("+proj=utm +zone=31"))
```

Compare coördinates of the first point in different CRS:

```
> coordinates(meuse)[1,]                    # RDH
181072 333611
> coordinates(meuse.wgs84)[1,]              # WGS84 datum long/lat
5.758536 50.991562
> coordinates(meuse.rdh.geo)[1,]            # RDH datum long/lat
5.759029 50.992415
> coordinates(meuse.wgs84.utm31)[1,]        # UTM31N on WGS84 datum
693585 5652509
> coordinates(meuse.rdh.utm31)[1,]          # UTM31N on RDH datum
693616 5652605
```

Discrepency in UTM coördinates $\sqrt{(585-616)^2+(509-605)^2} = 100.9$ m

Example – with `sf`

```
> (meuse.sf <- st_as_sf(meuse, coords=c("x", "y"))) # specify columns with coords
> st_crs(meuse.sf) <- 28992    # set CRS from the EPSG code
> meuse.sf

Simple feature collection with 155 features and 12 fields
geometry type:  POINT
dimension:      XY
bbox:           xmin: 178605 ymin: 329714 xmax: 181390 ymax: 333611
projected CRS:  Amersfoort / RD New
First 10 features:
   cadmium copper lead zinc  elev       dist   om ffreq dist.m              geometry
1     11.7     85  299 1022 7.909 0.00135803 13.6     1       50 POINT (181072 333611)
2      8.6     81  277 1141 6.983 0.01222430 14.0     1       30 POINT (181025 333558)
3      6.5     68  199  640 7.800 0.10302900 13.0     1      150 POINT (181165 333537).
```

Topic: The gstat package

- R implementation of the stand-alone **gstat** package for geostatistics
- Author and maintainer Edzer Pebesma
  - mostly developed at Physical Geography, University of Utrecht (NL)
  - since Oct 2007 at Institute for Geoinformatics, University of Münster (D)
- 1992 – ???
- Purpose: "**modelling**, **prediction** and **simulation** of geostatistical data in one, two or three dimensions"
- Can use either the `sf` or `sp` spatial data structures

There are other R packages with overlapping aims but different methods and interfaces (e.g, `geoR`, `spatial`, `RandomFields`).

Modelling with gstat

- `variogram`: Compute **experimental variograms** (also directional, residual)
  - User-specifiable cutoff, bins, anisotropy angles and tolerances
  - Can use Matheron or robust estimators
  - Optional argument to produce a **variogram cloud** (all point-pairs)
  - Optional argument to produce a **directional variogram surface** ("map")
- `vgm`: specify a **theoretical variogram model** for an empirical variogram
  - Many authorized models
  - Can specify models with **multiple structures**
- `fit.variogram`: least-squares adjustment of a variogram model to the empirical model
  - User-selectable fitting criteria
  - Can also use restricted maximum likelihood `fit.variogram.reml`
- `fit.lmc`: fit a linear model of **coregionaliztion** (for cokriging)
- `gstat`: complicated procedures, e.g., co-kriging.

Conclusion

- A **disadvantage** of working in R is the lack of interactive graphical analysis (e.g. in ArcGIS Geostatistical Analyst)

- The main **advantage** of doing spatial analysis in R is that the full power of the R environment (data manipulation, non-spatial modelling, user-defined functions, graphics . . .) can be brought to bear on spatial analyses

- The advantages of R (**open-source**, **open environment**, packages contributed and vetted by statisticians) apply also to spatial analysis