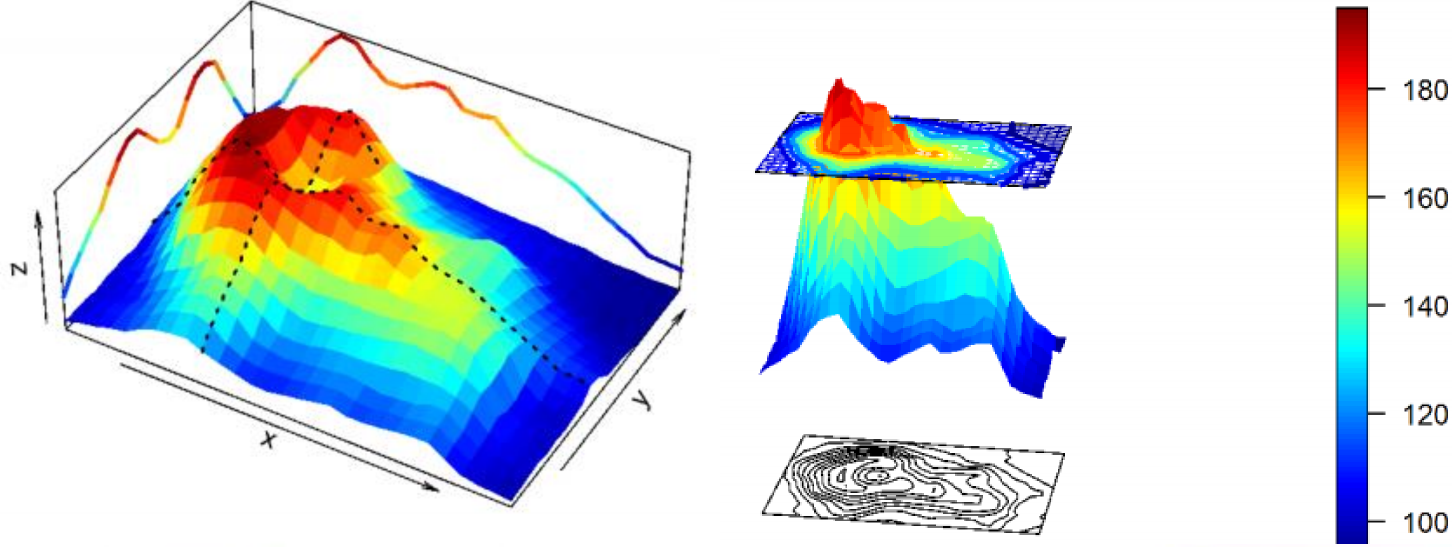
SPATIAL DATA ANALYSIS

Mengqian LU

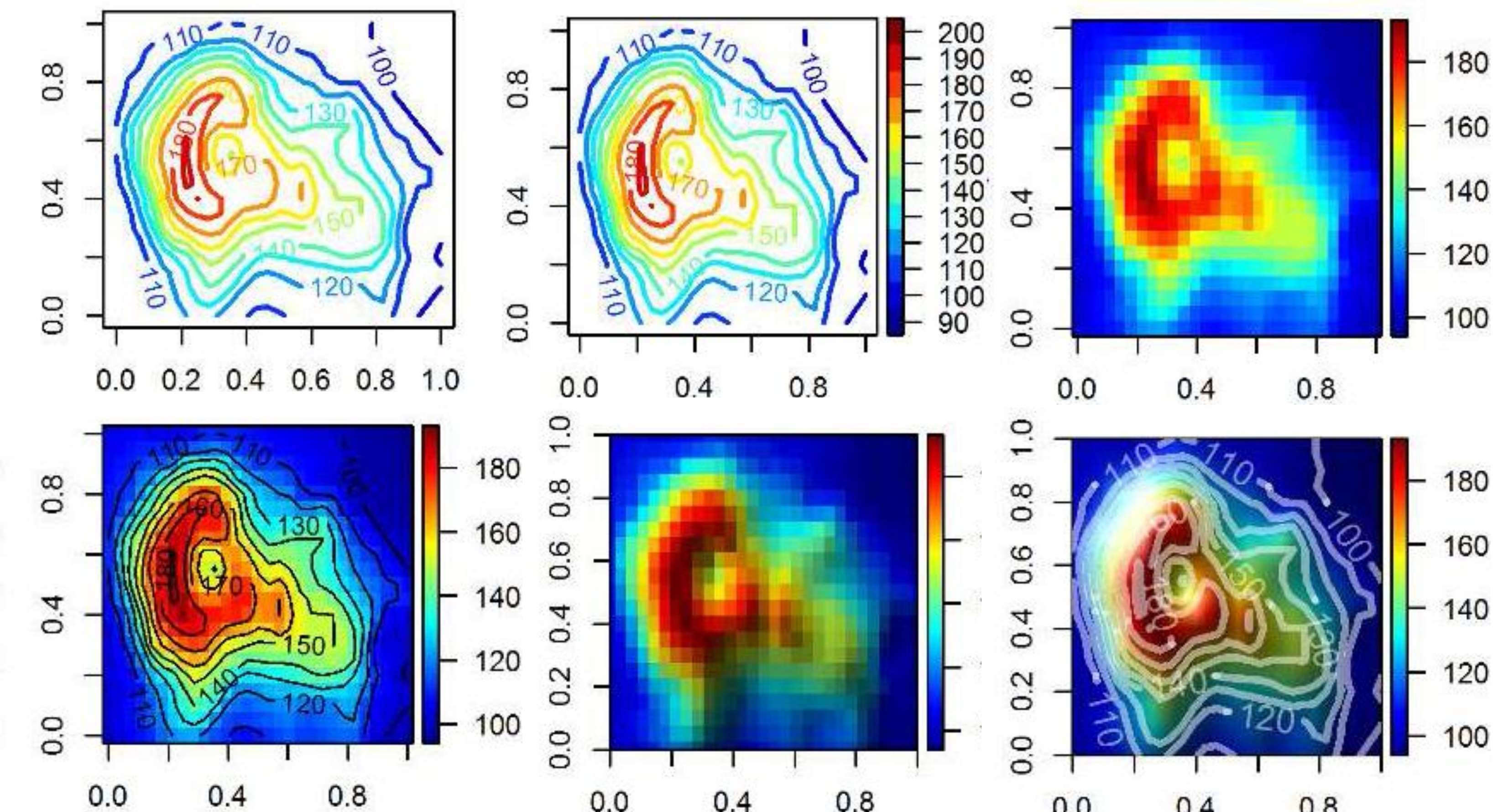
What is spatial data?

1. Spatial data (geospatial data): information/observations that identifies the geographic location of features and boundaries on Earth, e.g. rainfall over continental United States.
2. Typically stored as coordinates and topology, thus can be mapped
3. Traditional accessed, manipulated or analyzed by Geographic Information Systems (GIS), but now we have R.



Numerous ways…

R package: plot3D



What is spatial data? (Cont'd)

4. Spatial data store the information about location, scale, dimension and other geographic properties, e.g. data frame with [lat lon] and observation@[lat lon]; matrix with stand-alone [lat lon] matrix etc.
5. Vector vs. Raster

- ❑ Vector Data: a representation of the world using points, lines and polygons.
- ❑ Raster Data: a representation of the world as a surface divided into a regular grid of cells.

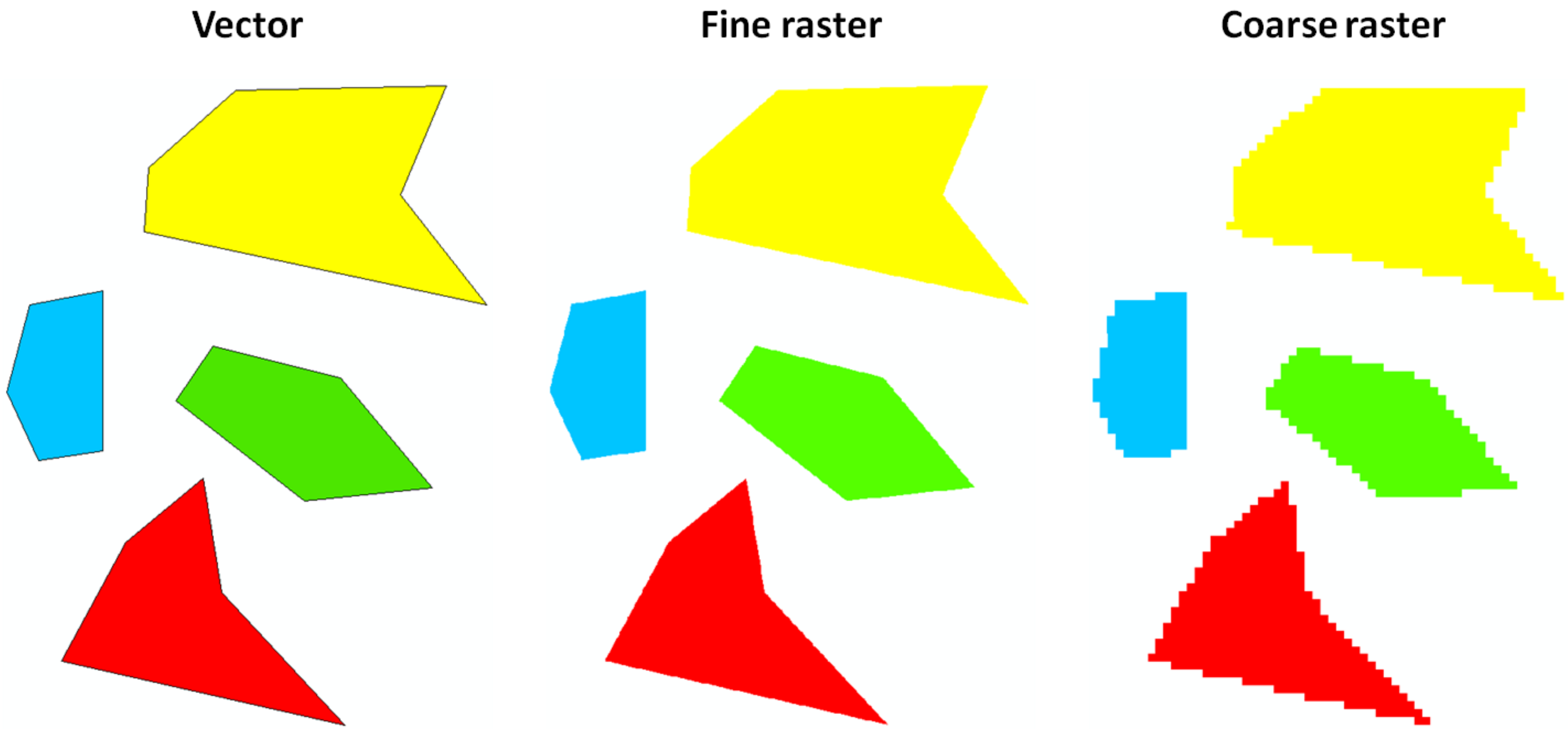
What is spatial data? (Cont'd)

5. Vector vs. Raster

   - Vector Data: For data that has discrete boundaries, such as country borders, land parcels and streets.
   - Raster Data: For data that varies continuously, as in an aerial photograph, a satellite image, a surface of biological concentrations, or an elevation surface

- Note that raster data consists of an array of regularly spaced cells, the points in a vector dataset need not be regularly spaced.

In many cases, both vector and raster representations of the same data are possible:

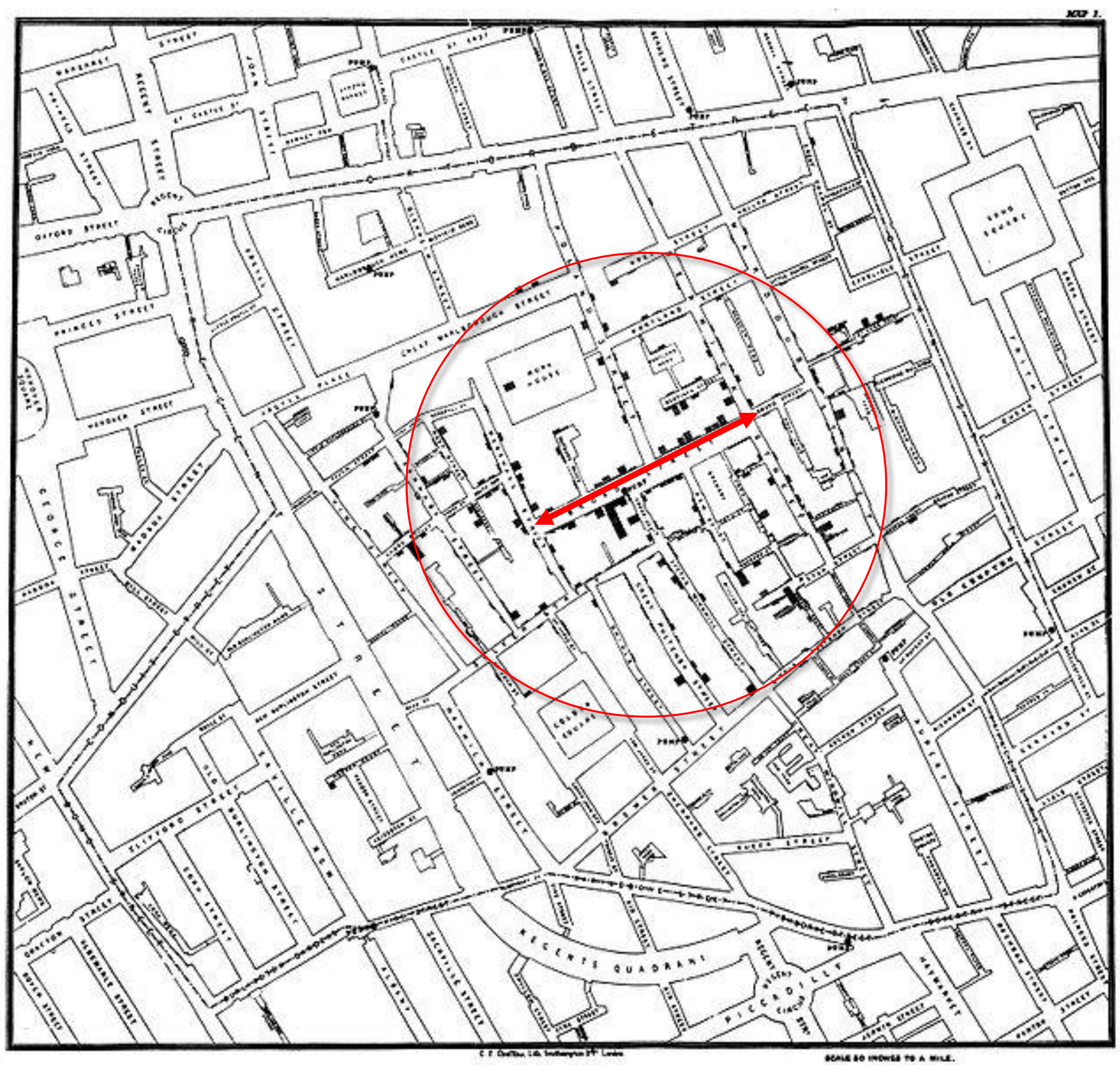


R packages with spatial data analysis tools

- Point Patterns: spatstat, VR:spatial, splancs
- Geostatistics: gstat, geoR, geoRglm, fields, spBayes,

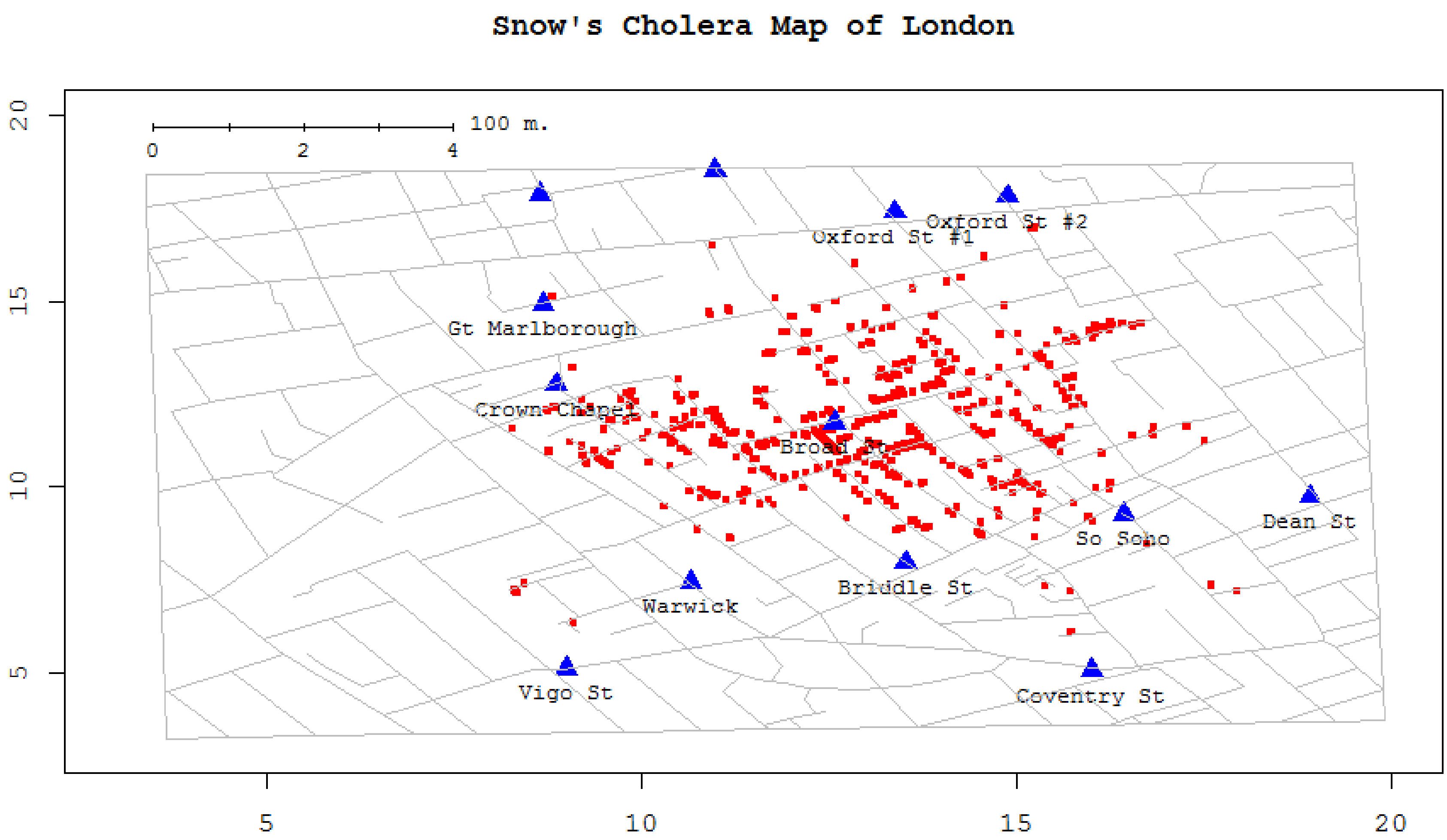
  RandomFields, VR:spatial, sgeostat, vardiag;
- Lattice/Area Data: spdep, DCluster, spgwr, ade4

Revisit: John Snow's Dot Distribution Map of 1854 London Cholera outbreak

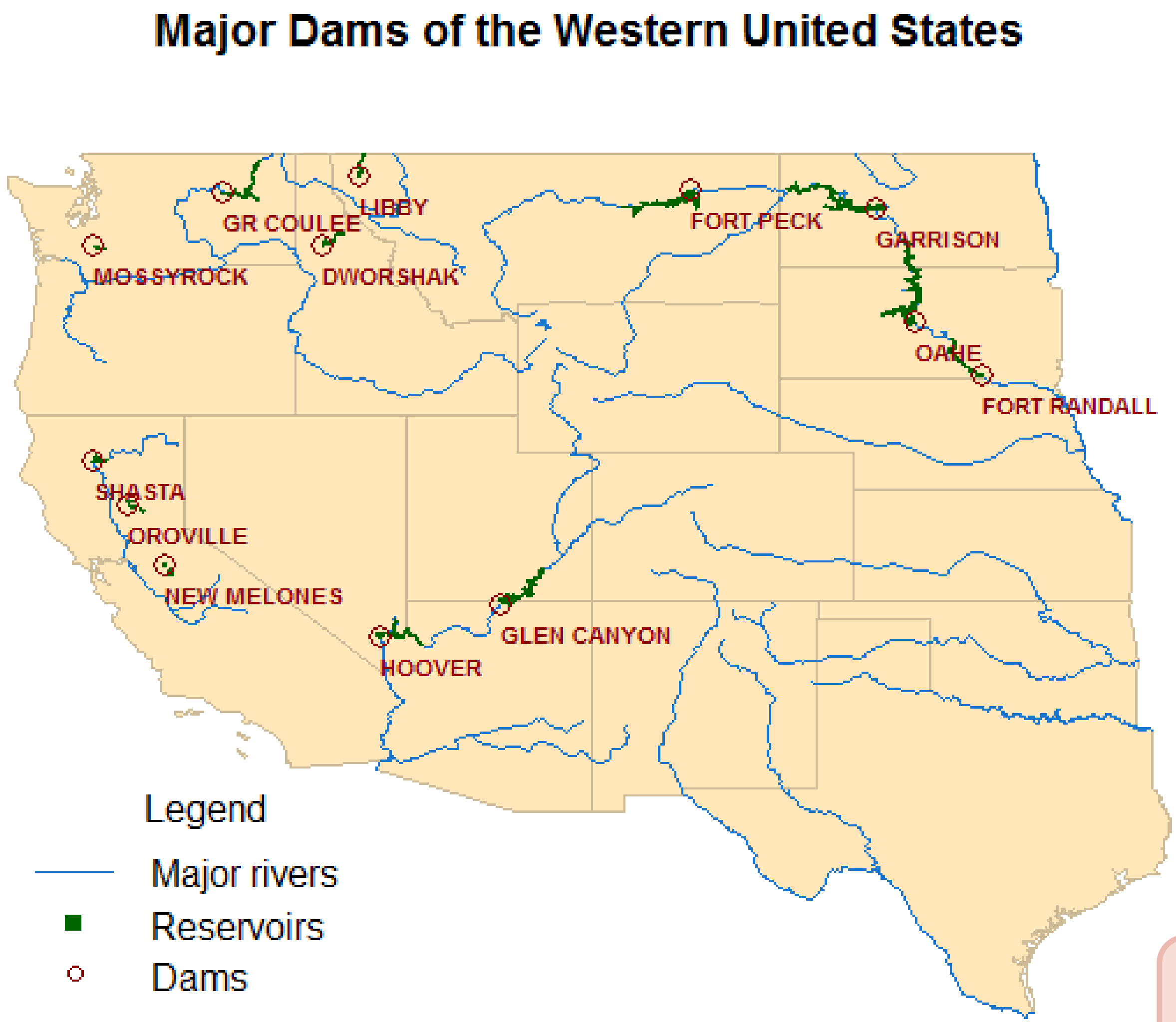


source: [Wikimedia Commons]

We will see the example of how to make the plot again in R by layering different information



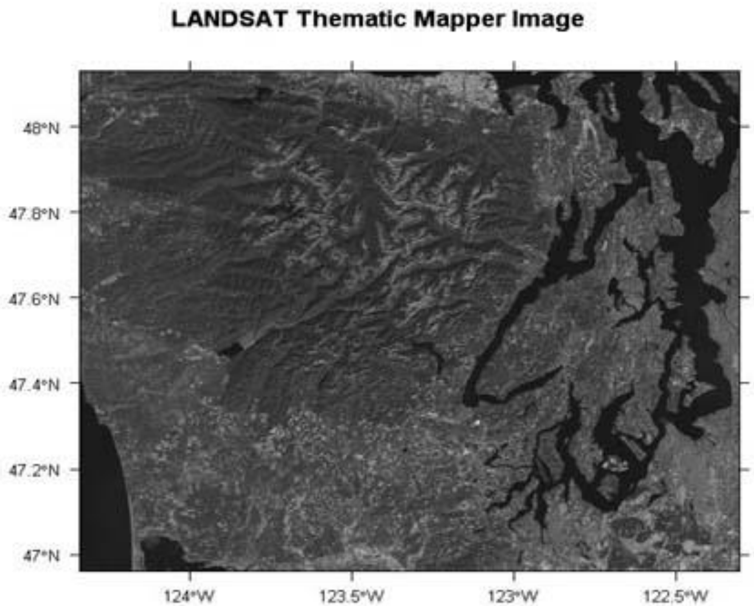
Map with points, lines & polygons



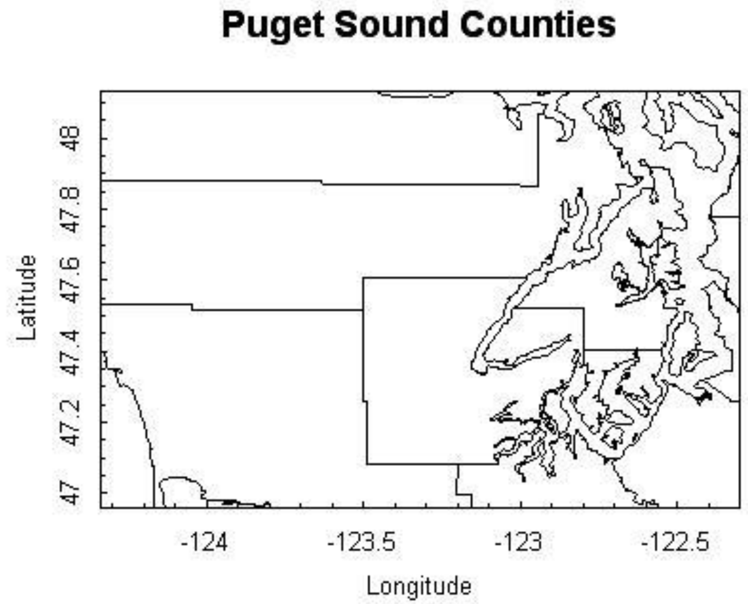
Shapefiles data

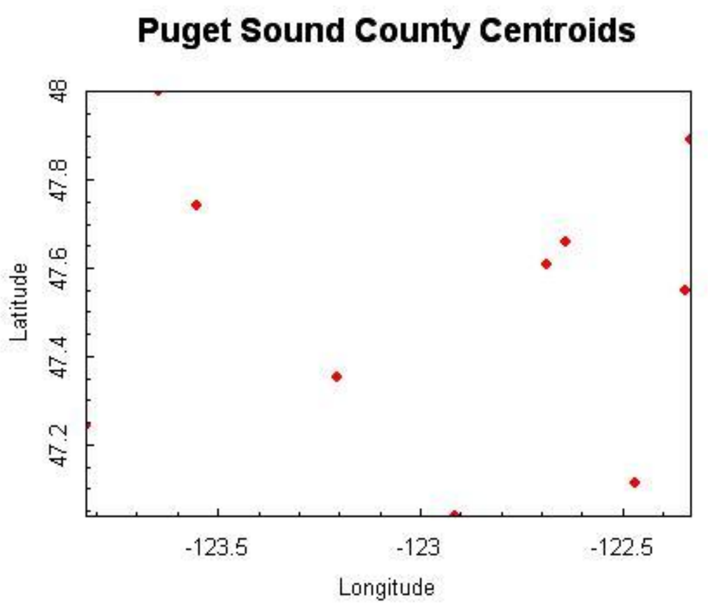
```r
library(sp)
library(maptools) # used here to read shapefiles

# read in the spatial data
# ...western US state outlines
states <- readShapePoly("western-states")
# ...major western US reservoirs
reservoirs <- readShapePoly("western-reservoirs")
# ...major western US rivers
rivers <- readShapeLines("western-rivers")
# ...locations of several western US dams
dams <- readShapePoints("western-dams")

# start by plotting the states
plot(states, border="wheat3", col="wheat1")
# add the river lines
lines(rivers, col="dodgerblue3")
# add the reservoirs
plot(reservoirs, col="darkgreen", border="darkgreen",
    add=TRUE)
# add dams (circled)
points(dams, cex=1.4, col="darkred")
# add dam labels (using trial and error for placement)
text(dams, labels=as.character(dams$DAM_NAME), col="darkred",
    cex=0.6, font=2, offset=0.5, adj=c(0,2))

# add a plot title and legend
title("Major Dams of the Western United States")
legend("bottomleft", legend=c("Major rivers", "Reservoirs", "Dams"),
    title="Legend", bty="n", inset=0.05,
    lty=c( 1,-1,-1), pch=c(-1,15, 1),
    col=c("dodgerblue3", "darkgreen", "darkred"))
```

Raster Grid: Satellite Image



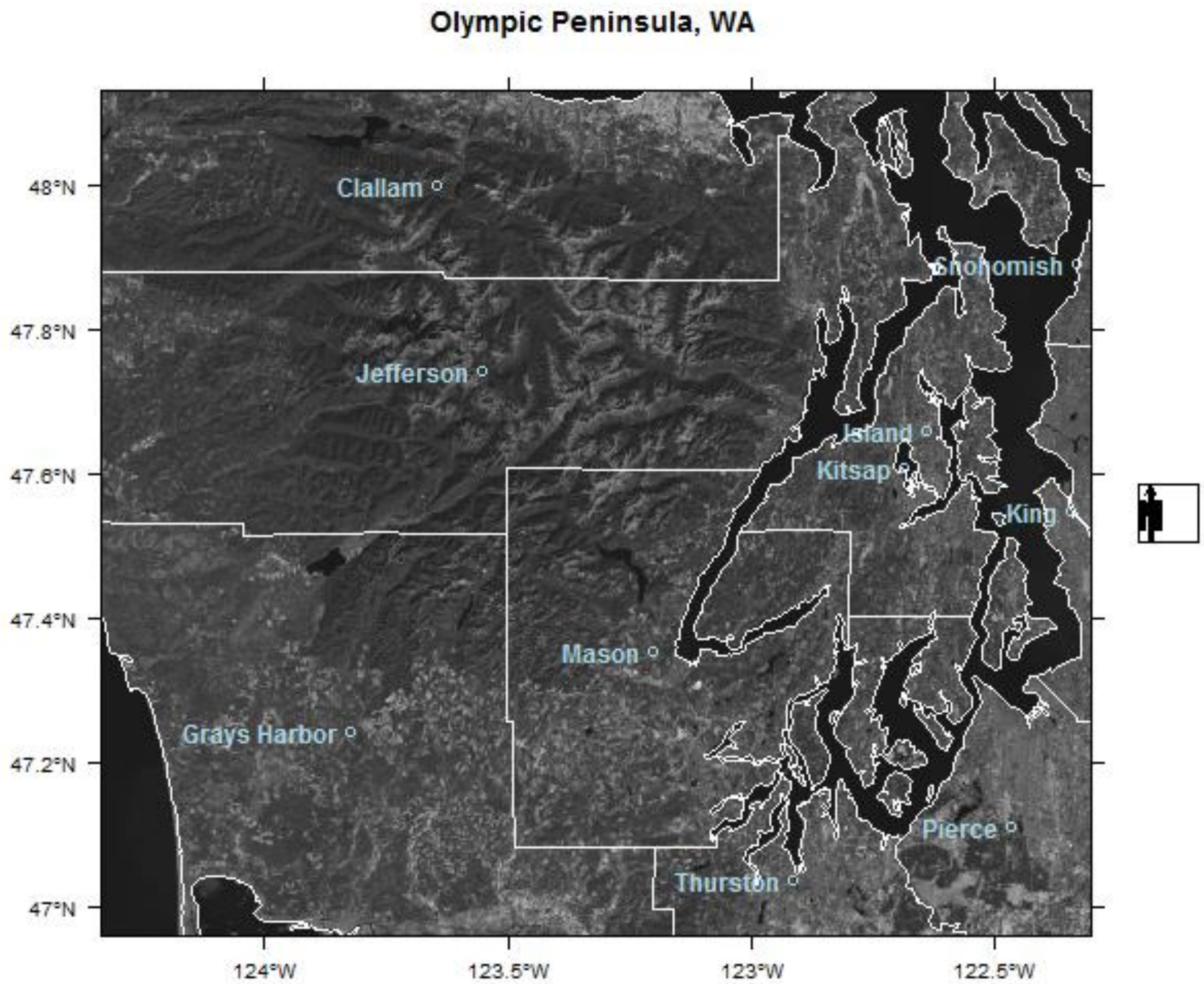
Polygons: Counties





Points: County Centroids

Raster base map with point and polygon overlays



Olympic Peninsula, WA
48°N
47.8°N
47.6°N
47.4°N
47.2°N
47°N
124°W
123.5°W
123°W
122.5°W
Clallam
Jefferson
Snohomish
Island
Kitsap
King
Mason
Grays Harbor
Pierce
Thurston

```r
library(sp)
library(rgdal)
library(maptools)

# read in the counties and their centroids
centroids <- readShapePoints("op-county-centroids")
counties <- readShapePoly("op-counties")

# read raster in as a SpatialGridDataFrame object
psImg <- readGDAL("op-landsat.img")

# specify overplot layers for use by spplot; in order for polygons to be
# plotted atop the raster, we need to convert them to SpatialLines
polys <- list("sp.lines", as(counties, "SpatialLines"), col="white")
points <- list("sp.points", centroids, col="lightblue", pch=1)
labels <- list("panel.text",
    coordinates(centroids)[,1], coordinates(centroids)[,2],
    labels=sub(" County", "", centroids$COUNTY),
    col="lightblue", font=2, pos=2)

# plot the raster with polygons, points, and labels
spplot(psImg, "band1", col.regions=grey(0:256/256),
    sp.layout=list(points, labels, polys), cuts=256,
    colorkey=FALSE, scales=list(draw=TRUE),
    main="Olympic Peninsula, WA",
    legend=list(right=list(fun=mapLegendGrob(layout.north.arrow()))))
```

R packages with spatial data analysis tools

- Point Patterns: spatstat, VR:spatial, splancs
- Geostatistics: gstat, geoR, geoRglm, fields, spBayes,

  RandomFields, VR:spatial, sgeostat, vardiag;
- Lattice/Area Data: spdep, DCluster, spgwr, ade4

Get started with spatial data

1. All contributed packages for spatial data in R have different representations
2. Thus incompatibility problems occur when exchanging data among packages
3. There is an attempt to develop shared classes to represent spatial data, efforts have done, allowing some shared methods and many-to-one, one-to-many conversions
4. Today's lecture will lead you to as many classes as possible, including new classes and most common ones

Spatial Objects in R (package "*sp*")

- The foundation object is the *Spatial* class, with just two slots (objects have pre-defined components called slots)
- The 1st is a bounding box, and is used to set up plots
- The 2nd is a *CRS* class object (**C**oordinate **R**eference **S**ystem), telling the geographic projection, used when converting or transforming one CRS to another (package *rgdal*)
- Operations on *Spatial** objects should update or copy these values to the new *Spatial** objects being created

Spatial Points (package "*sp*")

- The basic spatial data object is a point, e.g. 2d or 3d
- A single coordinate or a set of such coordinates may be used to define a *SpatialPoints* object; coordinates should be of mode *double;*
- The points in a *SpatialPoints* object may be associated with a row of attributes to create a *SpatialPointsDataFrame* object
- The coordinates and attributes may, but do not have to keyed to each other using ID values

Spatial Points

- Make a *SpatialPoints* object, e.g. Meuse bank data set of soil samples and measurements of heavy metal pollution provided with **sp**, we'll make a *SpatialPoints* object

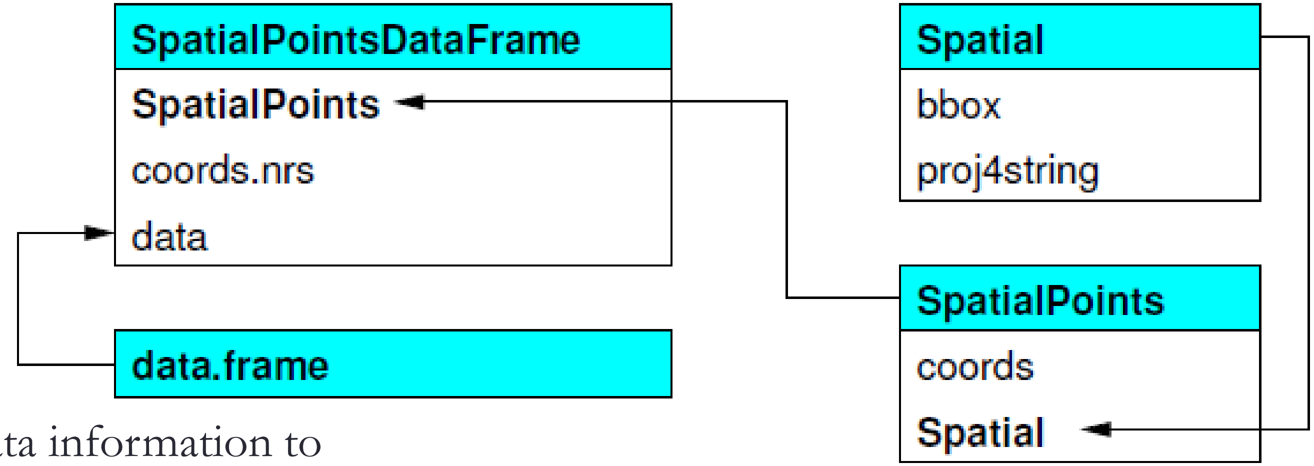
```
> library(sp)
> data(meuse)
> coords <- SpatialPoints(meuse[, c("x", "y")])
> summary(coords)

Object of class SpatialPoints
Coordinates:
     min    max
x 178605 181390
y 329714 333611
Is projected: NA
proj4string : [NA]
Number of points: 155
```

Spatial Points

- Now we'll add the original data frame to make a *SpatialPointsDataFrame* object.

```
> meuse1 <- SpatialPointsDataFrame(coords, meuse)
> names(meuse1)

 [1] "x"        "y"        "cadmium" "copper"  "lead"     "zinc"
 [7] "elev"     "dist"     "om"       "ffreq"   "soil"     "lime"
[13] "landuse" "dist.m"

> summary(meuse1$zinc)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  113.0   198.0   326.0   469.7   674.5  1839.0
```

Spatial Points classes and their slots



Add data information to the *SpatialPointsDataFrame*

Put the coordinates to *SpatialPointsDataFrame*

CRS to define the coordinates of a point → *SpatialPoints*

Spatial Line/Lines and Polygon/Polygons

- A *Line* object is just a spaghetti collection of 2D coordinates; a *Polygon* object is a *Line* object with equal *first* and *last* coordinates
- A *Lines* object is a list of *Line* objects, such as all the contours at a single elevation; the same relationship holds between a *Polygons* object and a list of *Polygon* objects, such as islands belonging to the same county
- *SpatialLines* and *SpatialPolygons* objects are made using lists of *Lines* or *Polygons* objects respectively
- *SpatialLinesDataFrame* and *SpatialPolygonsDataFrame* objects are defined using *SpatialLines* and *SpatialPolygons* objects and standard data frames, and the *ID* fields are here required to match the data frame row names

Spatial Polygons

- Make a *SpatialPolygons* object, e.g. Meuse bank data set has coordinates of the edge of the river, linked together at the edge of the study area to form a polygon.

```
> data(meuse.riv)
> str(meuse.riv)

 num [1:176, 1:2] 182004 182137 182252 182314 182332 ...

> river_polygon <- Polygons(list(Polygon(meuse.riv)), ID = "meuse")
> rivers <- SpatialPolygons(list(river_polygon))
> summary(rivers)

Object of class SpatialPolygons
Coordinates:
        min      max
r1 178304.0 182331.5
r2 325698.5 337684.8
Is projected: NA
proj4string : [NA]
```

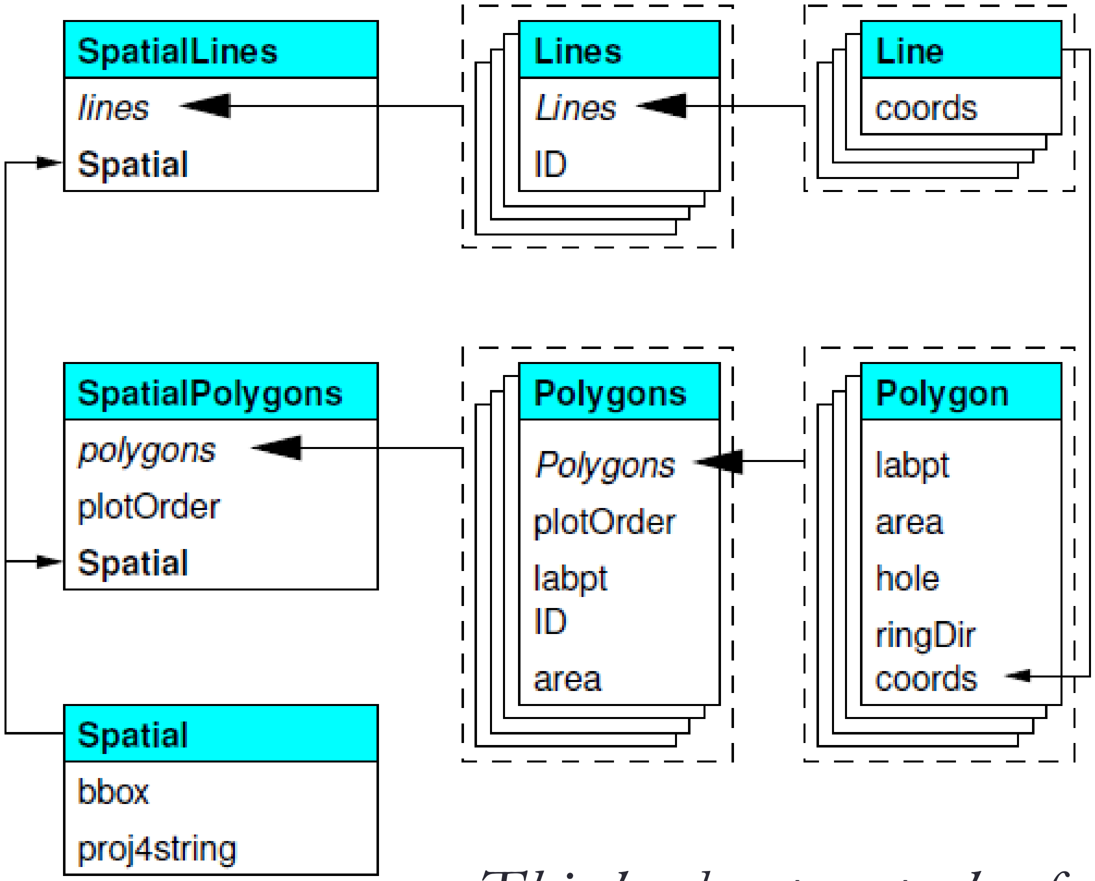
Spatial Lines

- Use *contourLines2SLDF()* to convert the list of contours returned by *contourLines* into a *SpatialLinesDataFrame* object

```
> library(maptools)

> volcano_sl <- ContourLines2SLDF(contourLines(volcano))
> row.names(slot(volcano_sl, "data"))

 [1] "C_1"  "C_2"  "C_3"  "C_4"  "C_5"  "C_6"  "C_7"  "C_8"  "C_9"
[10] "C_10"

> sapply(slot(volcano_sl, "lines"), function(x) slot(x,
+     "ID"))

 [1] "C_1"  "C_2"  "C_3"  "C_4"  "C_5"  "C_6"  "C_7"  "C_8"  "C_9"
[10] "C_10"

> sapply(slot(volcano_sl, "lines"), function(x) length(slot(x,
+     "Lines")))

 [1] 3 4 1 1 1 2 2 3 2 1

> volcano_sl$level

 [1] 100 110 120 130 140 150 160 170 180 190
Levels: 100 110 120 130 140 150 160 170 180 190
```

Note that some *Lines* objects include multiple *Line* objects

Spatial Polygons classes and their slots



Think about a stack of cards

Spatial Grids and Pixels

- On regular rectangular grids (oriented N-S, E-W): *SpatialPixels* and *SpatialGrid*
- *SpatialPixels* are like *SpatialPoints* objects, but regularly spaced; stored as are grid indices
- *SpatialPixelsDataFrame* objects only store attribute data where it is present, but need to store the coordinates and grid indices of those grid cells
- *SpatialGridDataFrame* objects do not need to store coordinates, because they fill the entire defined grid, but they need to store *NA* values where attribute values are missing

Spatial Pixels – example

- Make a *SpatialPixelsDataFrame* object for the *Meuse bank*

```
> data(meuse.grid)
> coords <- SpatialPixels(SpatialPoints(meuse.grid[, c("x",
+      "y")]))
> meuseg1 <- SpatialPixelsDataFrame(coords, meuse.grid)
> names(meuseg1)

[1] "x"       "y"       "part.a" "part.b" "dist"    "soil"    "ffreq"

> slot(meuseg1, "grid")

                       x      y
cellcentre.offset 178460 329620
cellsize              40     40
cells.dim             78    104

> object.size(meuseg1)

[1] 339036

> dim(slot(meuseg1, "data"))

[1] 3103    7
```

The data include soil types, flood frequency classes and distance from the river bank

The data are regular points at a 40m spacing, it has more on y-axis than x-axis

Spatial Grids – example

- Convert the *SpatialPixelsDataFrame* object to a *SpatialGridDataFrame*

```
> meuseg2 <- meuseg1
> fullgrid(meuseg2) <- TRUE
> slot(meuseg2, "grid")

                        x      y
cellcentre.offset 178460 329620
cellsize              40     40
cells.dim             78    104
> class(slot(meuseg2, "grid"))

[1] "GridTopology"
attr(,"package")
[1] "sp"
> object.size(meuseg2)

[1] 425684

> dim(slot(meuseg2, "data"))

[1] 8112    7
```

Usually, the *GridTopology* object in the *grid* slot is created directly

Spatial Classes in *sp*

This tabulates the classes supported by package '*sp*', and shows how they build up to the objects of most practical use, the '*Spatial__DataFrame*'

data type	class	attributes	extends
points	SpatialPoints	none	Spatial
points	SpatialPointsDataFrame	data.frame	SpatialPoints
pixels	SpatialPixels	none	SpatialPoints
pixels	SpatialPixelsDataFrame	data.frame	SpatialPixels SpatialPointsDataFrame
full grid	SpatialGrid	none	SpatialPixels
full grid	SpatialGridDataFrame	data.frame	SpatialGrid
line	Line	none	
lines	Lines	none	Line list
lines	SpatialLines	none	Spatial, Lines list
lines	SpatialLinesDataFrame	data.frame	SpatialLines
polygon	Polygon	none	Line
polygons	Polygons	none	Polygon list
polygons	SpatialPolygons	none	Spatial, Polygons list
polygons	SpatialPolygonsDataFrame	data.frame	SpatialPolygons

Special methods in *sp*

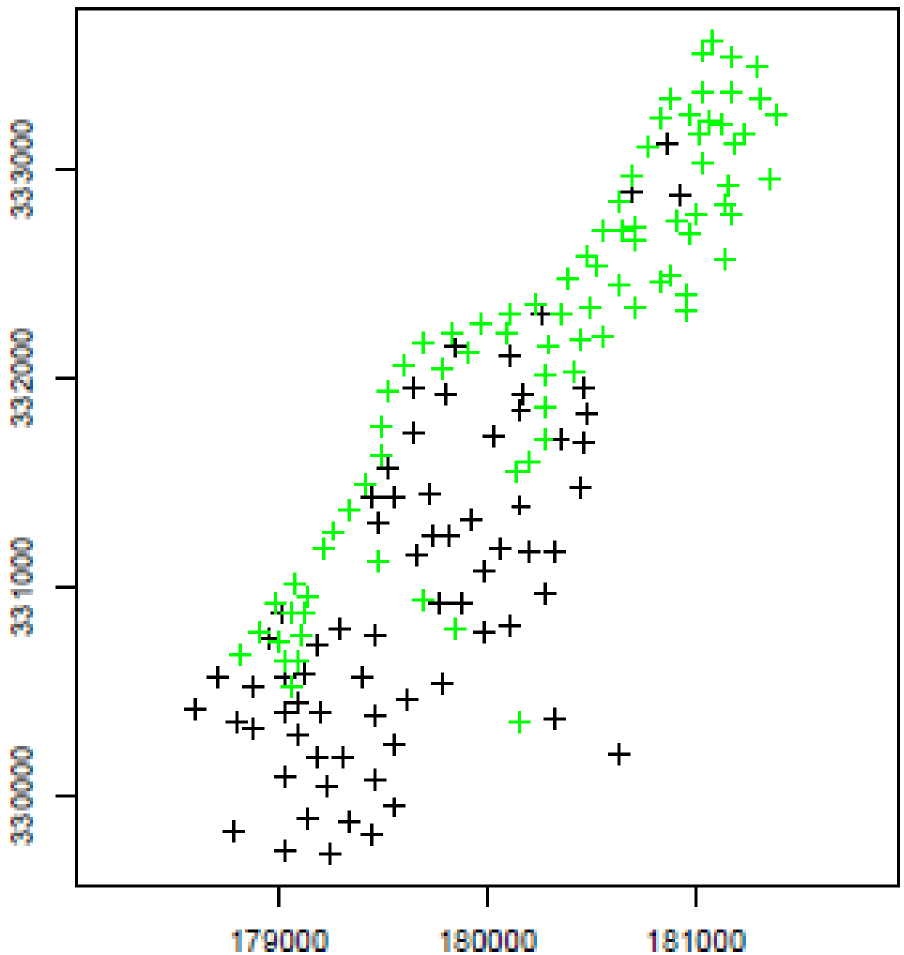
This tabulates the methods provided by the package '*sp*', and their usage

method	what it does
`[`	select spatial items (points, lines, polygons, or rows/cols from a grid) and/or attributes variables
`$`, `$<-`, `[[`, `[[<-`	retrieve, set or add attribute table columns
`spsample`	sample points from a set of polygons, on a set of lines or from a gridded area
`bbox`	get the bounding box
`proj4string`	get or set the projection (coordinate reference system)
`coordinates`	set or retrieve coordinates
`coerce`	convert from one class to another
`overlay`	combine two different spatial objects

Visualizing Spatial Data

- Base graphics for the key *Spatial__ classes*
- Additional plots added by layering

Spatial Visualization - *SpatialPoints*

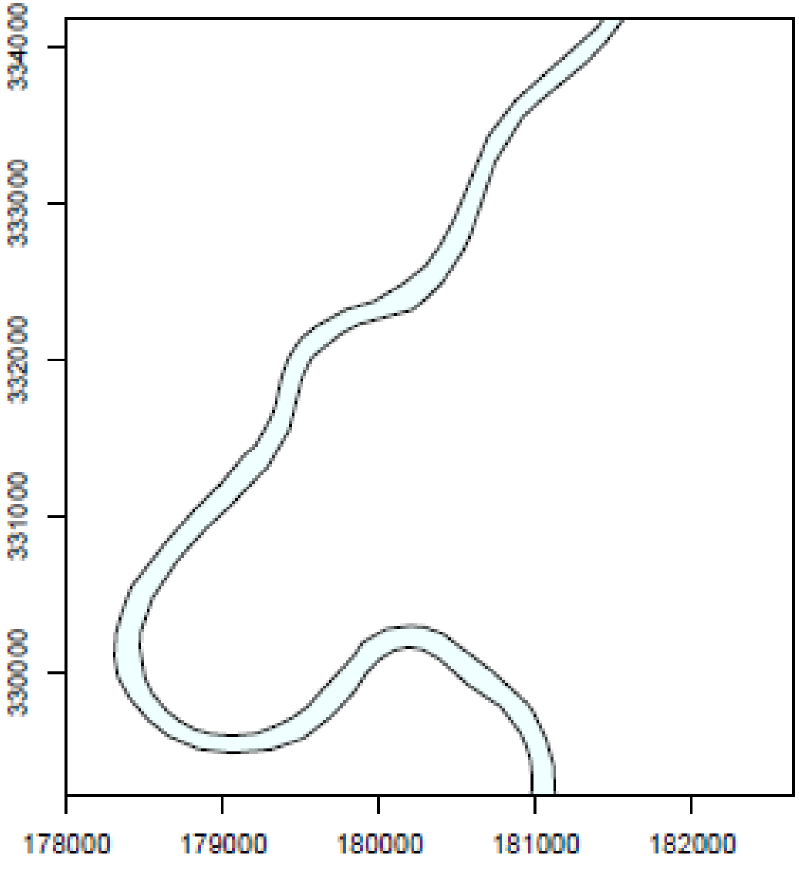


> plot(as(meuse1, 'Spatial'), axes = TRUE)

> plot(meuse1, add = TRUE)

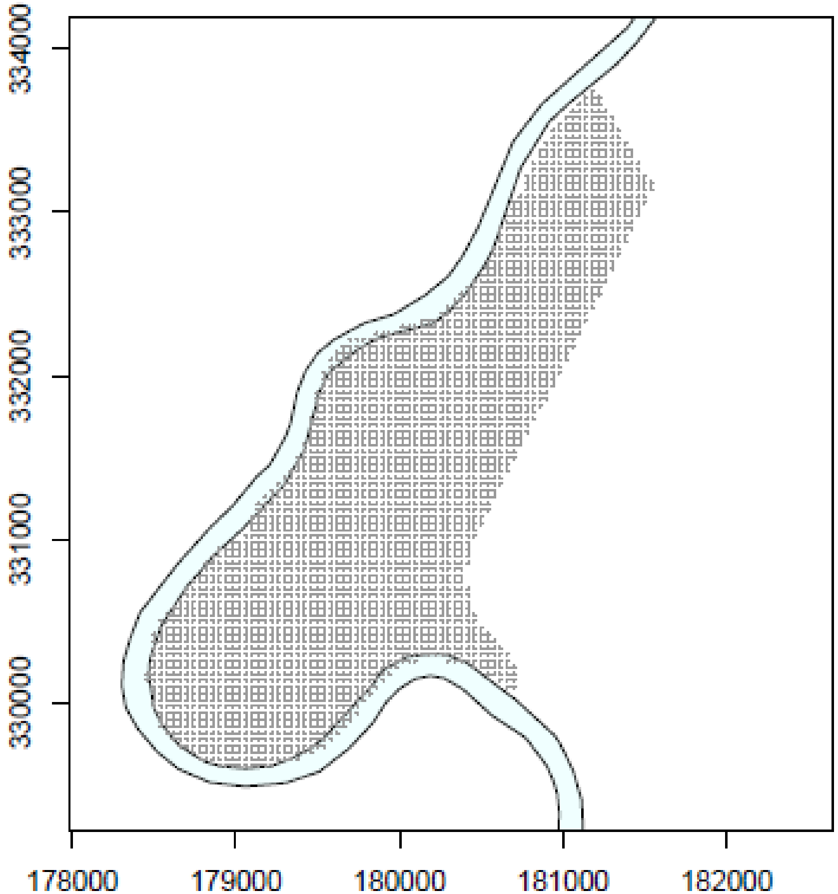
> plot(meuse1[meuse1$ffreq ==
+ 1,], col = 'green', add = TRUE)

Spatial Visualization - *SpatialPolygons*




```
> plot(rivers, axes = TRUE),
+ col = 'azurel',
+ ylim=c(329400,334000))
> box()
```

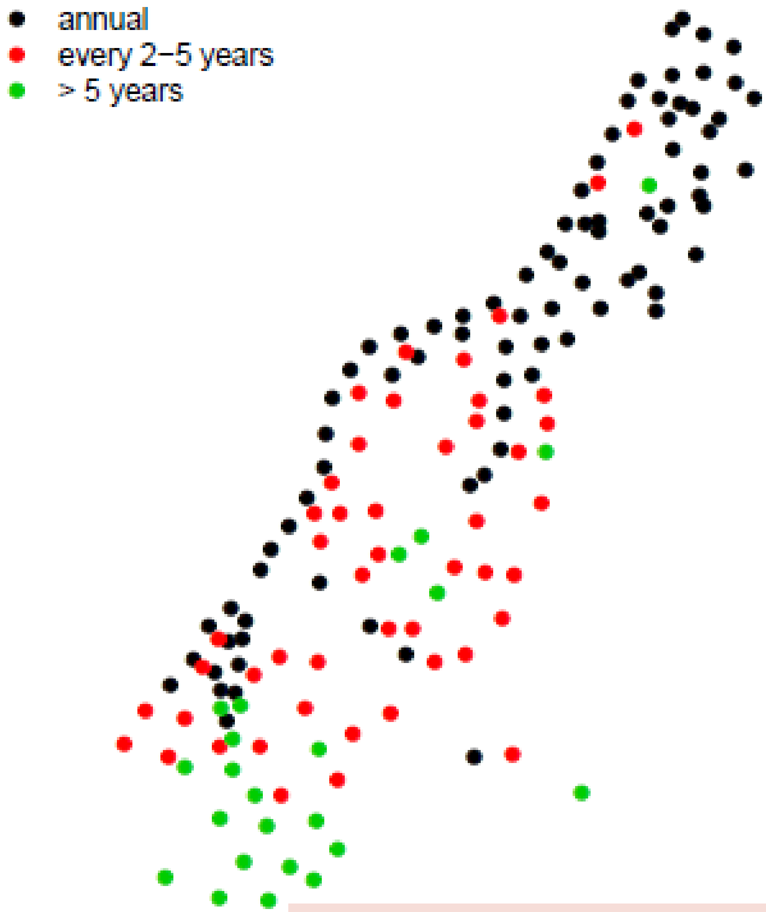
Spatial Visualization - *SpatialPixels*




```
> plot(rivers, axes = TRUE),
+ col = 'azurel',
+ ylim=c(329400,334000))
> box()
> plot(meuseg1,add = TRUE,
+ col = 'grey60', cex = 0.15)
```

Points, lines, and polygons are often plotted without attributes, this is rarely the case for gridded objects

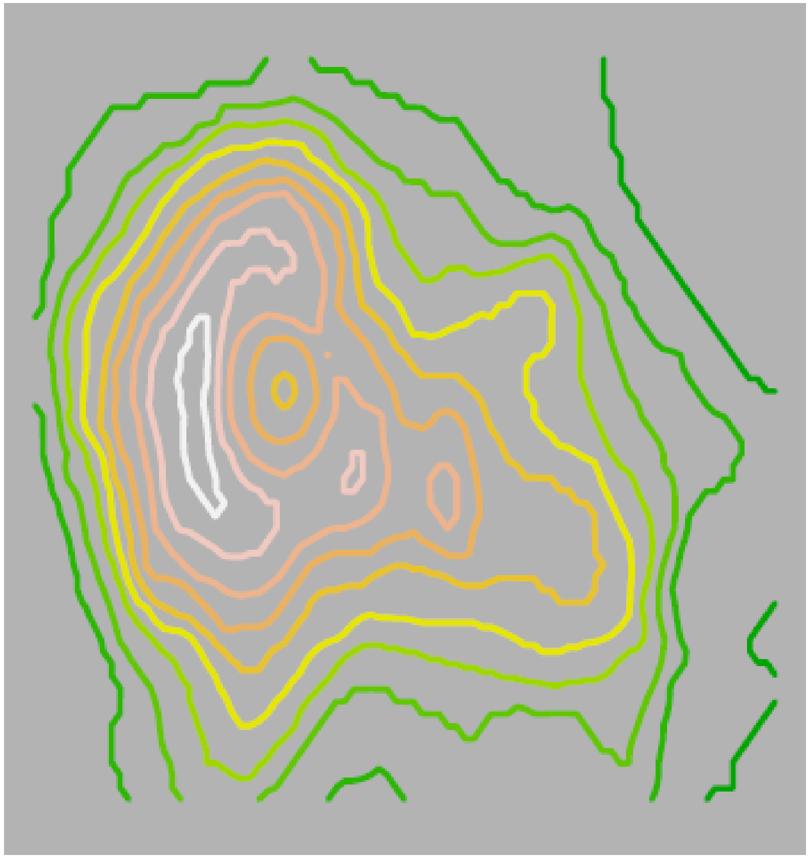
Spatial Visualization - Attributes




```
> meuse1$ffreq1 =
as.numeric(meuse1$ffreq)
> plot(meuse1, col = meuse1$ffreq1,
+ pch = 19)
> labs = c("annual", "every 2-5 years",
+ "> 5 years")
> cols = 1:nlevels(meuse1$ffreq)
> legend("topleft", legend = labs,
+ col = cols, pch = 19, bty = "n")
```

Colors for different categories (nominal)

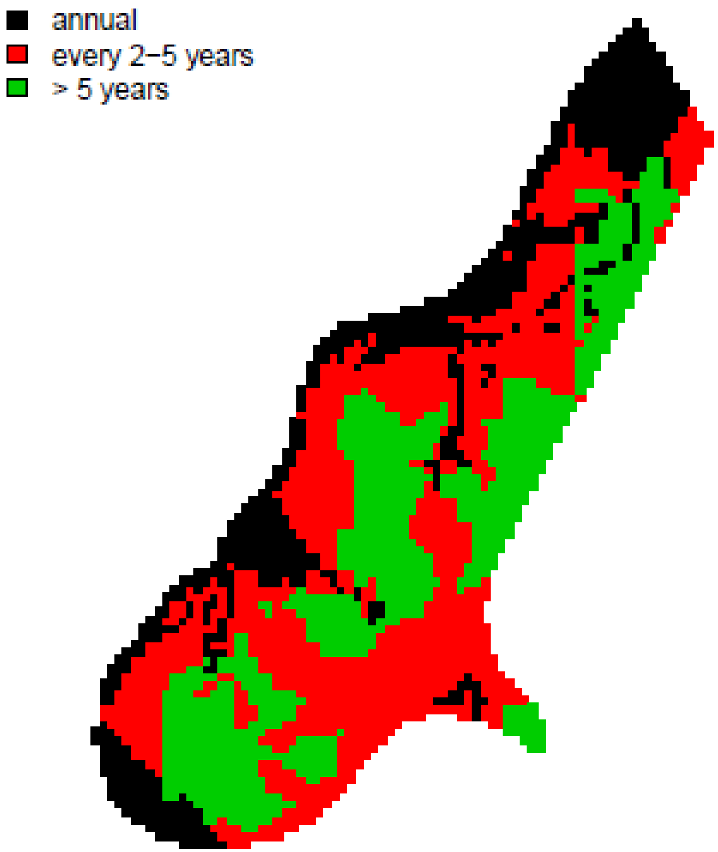
Spatial Visualization - Attributes



```
> volcano_sl$level1 =
as.numeric(volcano_sl$level)
> pal = terrain.colors(nlevels(volcano_sl$level))
> plot(volcano_sl, bg = "grey70",
+ col = pal[volcano_sl$level1],
+ lwd = 3)
```

Colored contour lines for different levels (ordered)

Spatial Visualization - Gridded data




```
> meuseg1$ffreq1 <-
as.numeric(meuseg1$ffreq)
> image(meuseg1, "ffreq1", col = cols)
> legend("topleft", legend = labs,
+ fill = cols, bty = "n")
```

Spatial Visualization - User-defined Class

- R package for choosing class intervals: *classInt*
- Classification techniques may be used: pretty, quantile, natural breaks among others, or fixed values of your choice based on your domain knowledge
- Then the intervals can be used to generate colors from a color palette, using *colorRampPalette()*

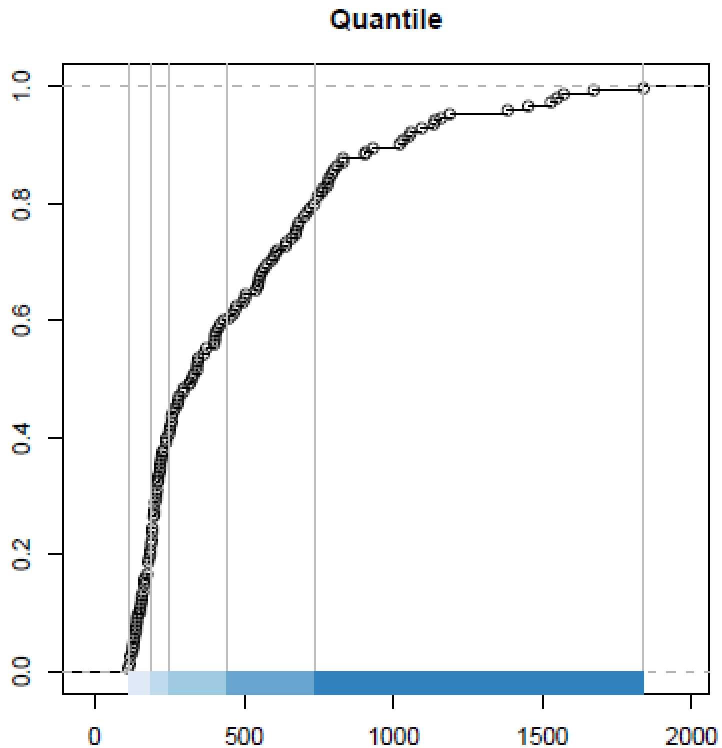
Spatial Visualization - User-defined Class

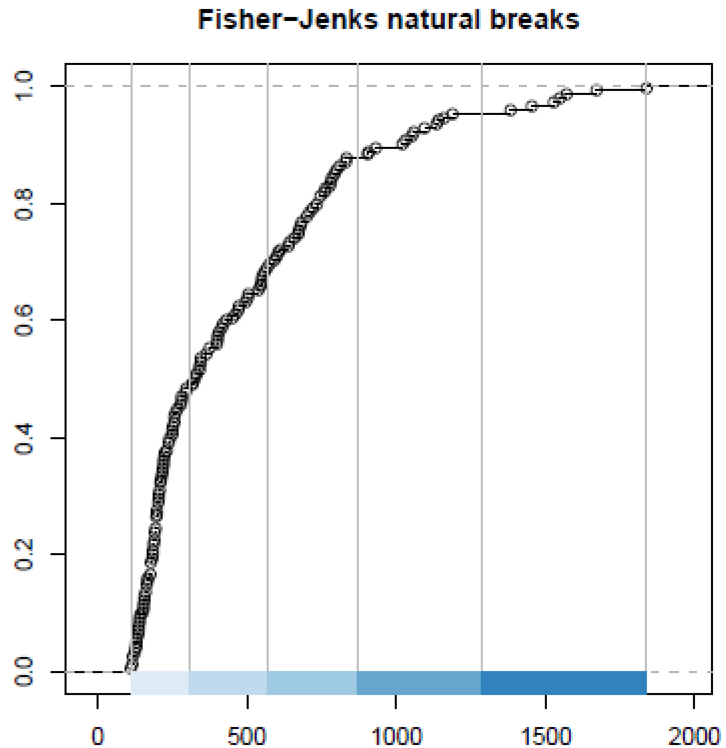
```
> library(classInt); library(RColorBrewer)
> pal <- brewer.pal(3, "Blues")
> q5 <- classIntervals(meuse1$zinc, n = 5, style = "quantile")
> q5
style: quantile
one of 14,891,626 possible partitions of this variable into 5 classes
under 186.8 186.8 - 246.4 246.4 - 439.6 439.6 - 737.2 over 737.2
31 31 31 31 31
> fj5 <- classIntervals(meuse1$zinc, n = 5, style = "fisher")
> fj5
style: fisher
one of 14,891,626 possible partitions of this variable into 5 classes
under 307.5 307.5 - 573.0 573.0 - 869.5 869.5 - 1286.5
        75          32                    29          12
over 1286.5
        7
```

Spatial Visualization - <u>User-defined Class</u>

```
> plot(q5, pal = pal)
> plot(fj5, pal = pal)
```

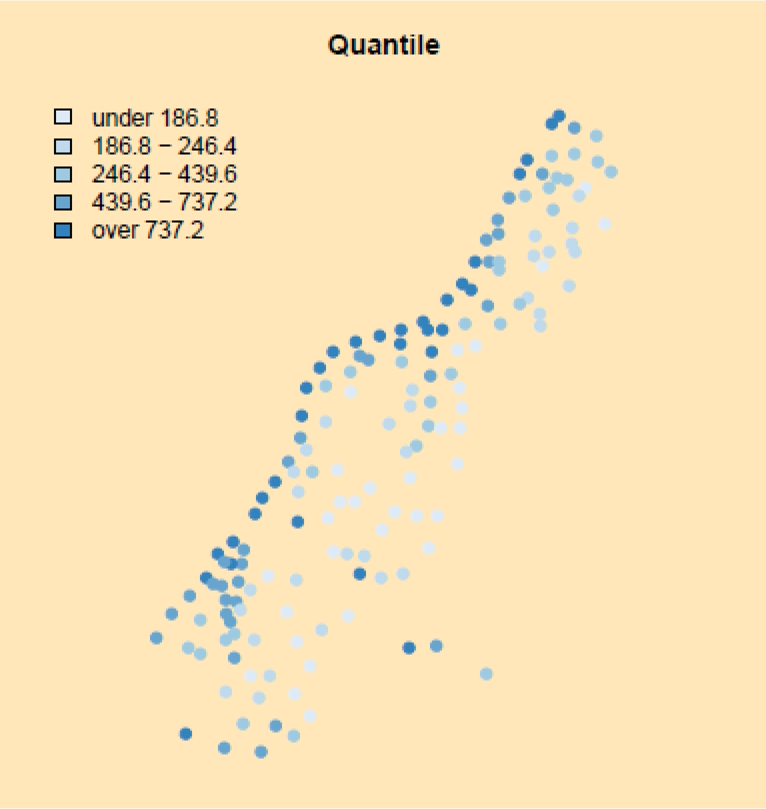
Q: Which one do you prefer, why?

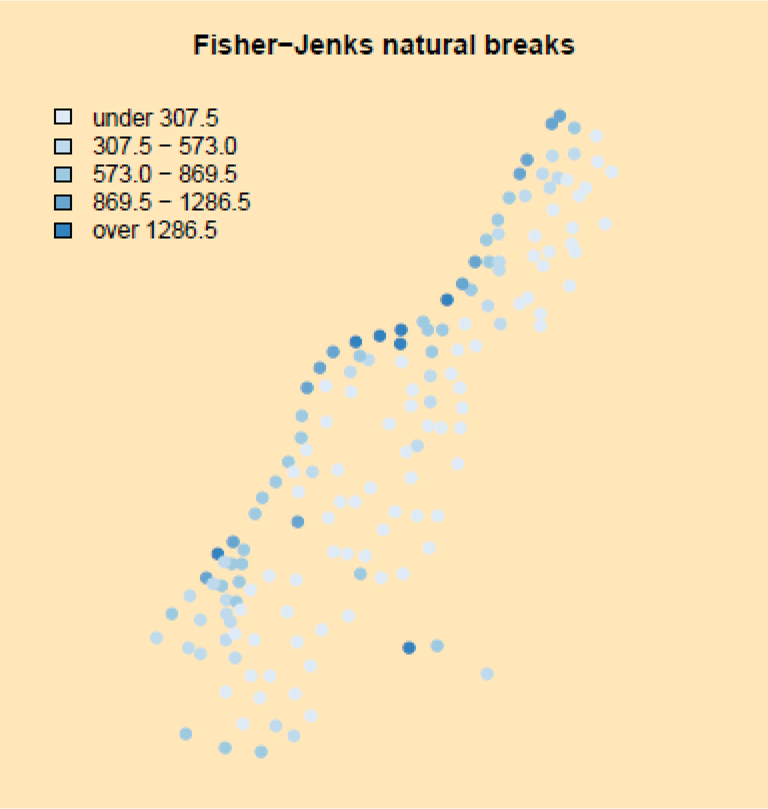




Spatial Visualization - User-defined Class

Results using the defined classes:

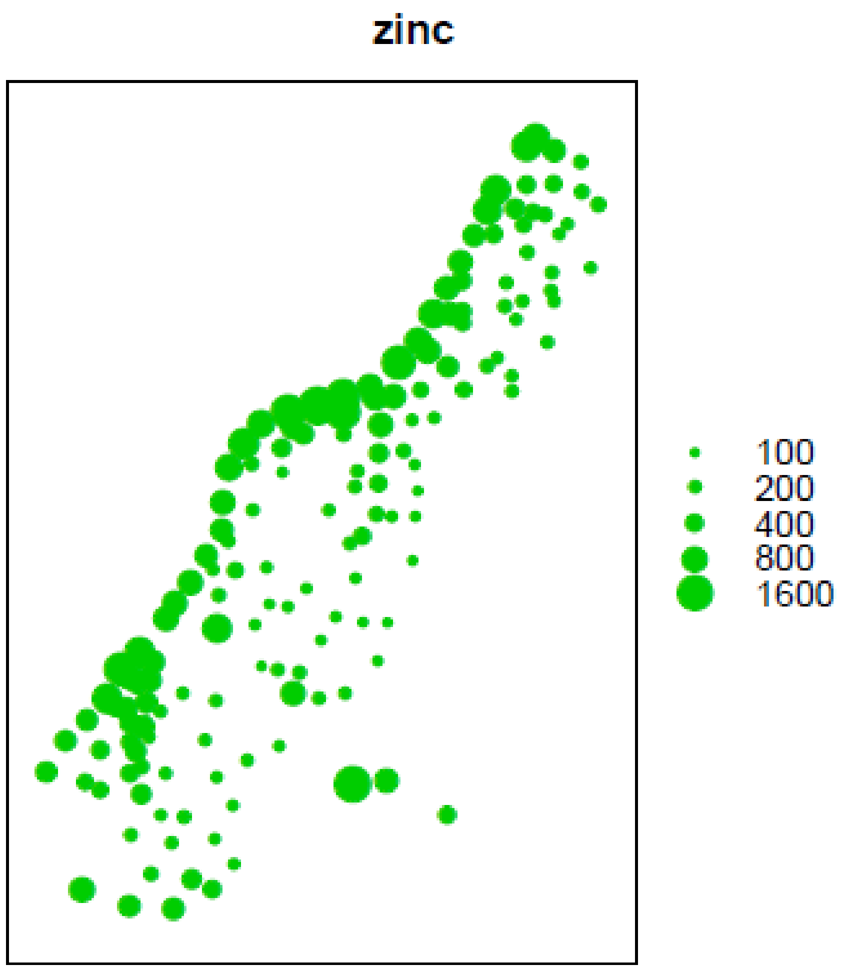




Spatial Visualization - *More Styles*

Lattice graphics –Bubble plots

```
> library(lattice)

> bubble(meuse1, "zinc", maxsize = 2,

+ key.entries = 100 * 2^(0:4))
```



Spatial Visualization - *More Styles*

Lattice graphics –Level plots

```
> bpal <- colorRampPalette(pal)(41)
> spplot(meuseg1, "dist",
+ col.regions = bpal, cuts = 40)
```