# Lecture 17

## Introduction to Raster and Visualizing Spatial Data

GEOG 489 SPRING 2020

# Quiz 6

Generate a data frame (50 rows and 4 columns) with random samples from the normal distribution (mean = 0, sd = 1). Please write the steps to calculate the mean of each row in the data frame using parallel computing with 4 cores.

The R file needs to be named:
LastName_FirstName_Quiz6.R

Please submit the quiz R file on Compass by the end of this Friday (March 27).

# Data Models

In GIS, we generally deal with four types of data models:

1) Points: a single location, given by a 2- or 3-d coordinate.

2) Lines: a set of ordered points connected by straight line segments.

3) Polygons: an area, comprised of a set of enclosing lines

4) **Rasters (aka "grids" aka "images"): a set of regularly spaced rectangles, arranged in a lattice.**

# Raster

Rasters (or "grids" or "images") represent a set of regularly spaced areas ("cells" or "pixels") on the earth's surface.

What differentiates them from vectors is that the coordinates of each cell are not stored explicitly.

A raster file has two parts: the data, and the header information.

# Raster

**The data:** in R terms, the data is a vector of data, one value per grid cell.

**The header:** this carries the information needed to define, minimally, a 2- or 3-d image, e.g. the number of columns, rows, and bands.

To put the grid into geographic space, the CRS is defined, the (typically) coordinate of the upper left cell, and the resolution of the grid must be known.

# Raster Package

**The 'raster' package:** 'raster' is the defacto standard for raster processing in R.

1) Chunking/memory management: You don't need to worry about running out of memory when working with even with extremely large rasters.

2) RGDAL support: you don't need to worry about the format of a file you are working with -- all commands will work with essentially all raster formats.

# Raster Classes

**Raster class slots:**

- The number of columns and rows in the raster;
- The CRS info;
- The spatial extent (similar to the bbox object in Spatial);
- Whether the raster is rotated and what the angle is;
- The information on the z- values, e.g. do they represent wavelengths, times, etc?

# Raster Classes

**Raster class is inherited by the following classes:**

- **RasterLayer:** a single layer (band) raster
- **RasterBrick:** a multiband raster originating from a SINGLE file.
- **RasterStack:** a multiband raster comprised of MULTIPLE files.

# RasterLayer

**RasterLayer:** inherits Raster; but adds a file slot (if the data is stored on disk); data slot (the data itself); legend and history.

A RasterLayer can be created using:

x <- raster() # Default makes a 1 degree raster of the planet.

```
class       : RasterLayer
dimensions  : 180, 360, 64800  (nrow, ncol, ncell)
resolution  : 1, 1  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

# RasterLayer

**RasterLayer:** inherits Raster; but adds a file slot (if the data is stored on disk); data slot (the data itself); legend and history.

A RasterLayer can be created using:

```
x <- raster(ncol=36, nrow=18, xmn=-1000, xmx=1000, ymn=-100, ymx=900)
```

To set the projection of a raster, use the PROJ4 string and projection():

```
projection(x) <- "+proj=utm +zone=48 +datum=WGS84“
```

```
class       : RasterLayer
dimensions  : 18, 36, 648  (nrow, ncol, ncell)
resolution  : 55.55556, 55.55556  (x, y)
extent      : -1000, 1000, -100, 900  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=48 +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

# RasterLayer

**RasterLayer:** inherits Raster; but adds a file slot (if the data is stored on disk); data slot (the data itself); legend and history.

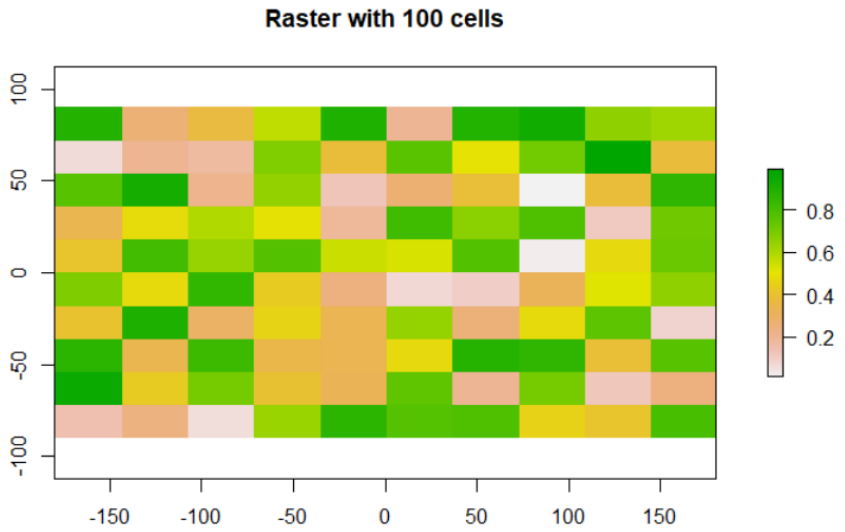
To assign the values to the raster:

values(x) <- runif(ncell(x))

The values are stored as a vector

and can be indexed as such:

values(x)[1:10]



# RasterStack

**RasterStack:** Multi-layered Raster from multiple sources

```
r1 <- r2 <- r3 <- raster(nrow=10,ncol=10)

values(r1) <- runif(ncell(r1))

values(r2) <- runif(ncell(r2))

values(r3) <- runif(ncell(r3))

# Combine multiple RasterLayer objects:

s <- stack(r1,r2,r3)
```

```
class       : RasterStack
dimensions  : 10, 10, 100, 3  (nrow, ncol, ncell, nlayers)
resolution  : 36, 18  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
names       :    layer.1,    layer.2,    layer.3
min values  : 0.01307758, 0.02778712, 0.06380247
max values  :  0.9926841,  0.9815635,  0.9960774
```

# RasterStack

**RasterStack:** Multi-layered Raster from multiple sources

```
r1 <- r2 <- r3 <- raster(nrow=10,ncol=10)

values(r1) <- runif(ncell(r1))

values(r2) <- runif(ncell(r2))

values(r3) <- runif(ncell(r3))

# Combine multiple RasterLayer objects:

s <- stack(r1,r2,r3)
```

```
class       : RasterStack
dimensions  : 10, 10, 100, 3  (nrow, ncol, ncell, nlayers)
resolution  : 36, 18  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
names       :    layer.1,    layer.2,    layer.3
min values  : 0.01307758, 0.02778712, 0.06380247
max values  :  0.9926841,  0.9815635,  0.9960774
```

# RasterBrick

**RasterBrick:** Multi-layered Raster from a single source

If the layers are in memory, there is no real difference between a RasterBrick and RasterStack:

b1 <- brick(r1,r2,r3)

```
class       : RasterBrick
dimensions  : 10, 10, 100, 3  (nrow, ncol, ncell, nlayers)
resolution  : 36, 18  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       :    layer.1,    layer.2,    layer.3
min values  : 0.01307758, 0.02778712, 0.06380247
max values  :  0.9926841,  0.9815635,  0.9960774
```

Convert from a RasterStack to a RasterBrick:

b2 <- brick(s)

# Visualize Spatial Data

We can use plot() for some basic Spatial objects plotting, and the 'lattice' package for some more complex plotting

#The Traditional Plot System.

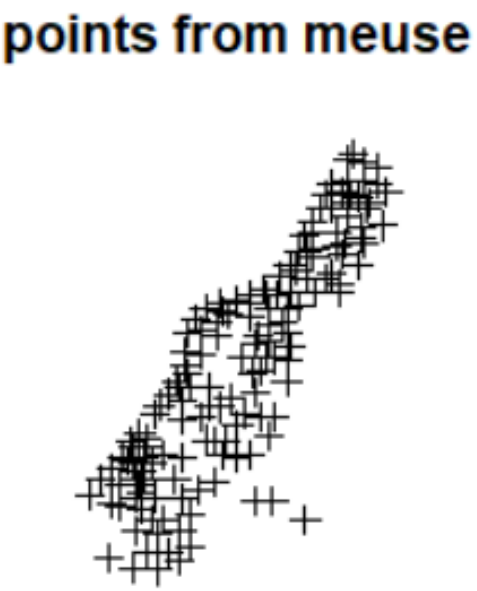
```
library(sp)

data(meuse) # A data frame with x and y coordinates:

# Let's make a SpatialPointsDataFrame:

coordinates(meuse) <- c("x","y")

plot(meuse)
```



# Visualize Spatial Data

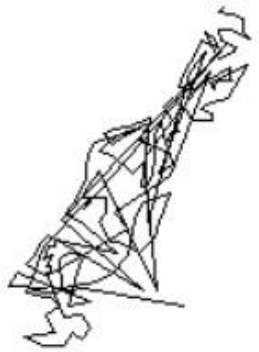
We can use plot() for some basic Spatial objects plotting, and the 'lattice' package for some more complex plotting

#Now a SpatialLines object

```
cc <- coordinates(meuse)

meuse.SpatialLines <- SpatialLines(list(Lines(list(Line(cc)),ID="oneLine")))

plot(meuse.SpatialLines)
```



# Visualize Spatial Data

We can use plot() for some basic Spatial objects plotting, and the 'lattice' package for some more complex plotting

#Now a SpatialPolygons object

```
data(meuse.riv)

meuse.List <- list(Polygons(list(Polygon(meuse.riv)),ID="meuse.riv"))

meuse.SpatialPolygons <- SpatialPolygons(meuse.List)

plot(meuse.SpatialPolygons,col="yellow")
```

polygons from meuse.riv

# Visualize Spatial Data

We can use plot() for some basic Spatial objects plotting, and the 'lattice' package for some more complex plotting

#Now a raster:

convert from a set of x,y coordinates to a raster:

```
data(meuse.riv)

xyz <- data.frame(meuse.grid$x,meuse.grid$y)

xyz$z <- vector(length=length(meuse.grid$x))

meuse.raster <- rasterFromXYZ(xyz)
```

| | x | y | part.a | part.b | dist | soil | ffreq |
|---|---|---|---|---|---|---|---|
| 1 | 181180 | 333740 | 1 | 0 | 0.0000000 | 1 | 1 |
| 2 | 181140 | 333700 | 1 | 0 | 0.0000000 | 1 | 1 |
| 3 | 181180 | 333700 | 1 | 0 | 0.0122243 | 1 | 1 |
| 4 | 181220 | 333700 | 1 | 0 | 0.0434678 | 1 | 1 |
| 5 | 181100 | 333660 | 1 | 0 | 0.0000000 | 1 | 1 |
| 6 | 181140 | 333660 | 1 | 0 | 0.0122243 | 1 | 1 |

# Visualize Spatial Data

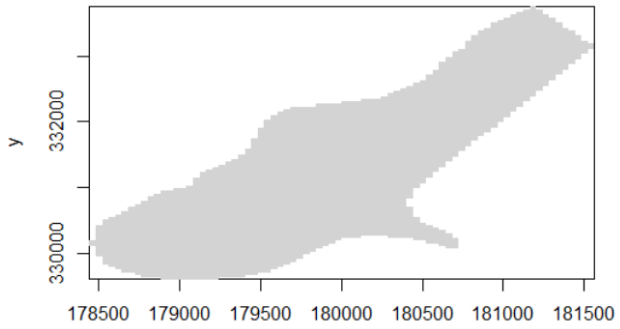
We can use plot() for some basic Spatial objects plotting, and the 'lattice' package for some more complex plotting

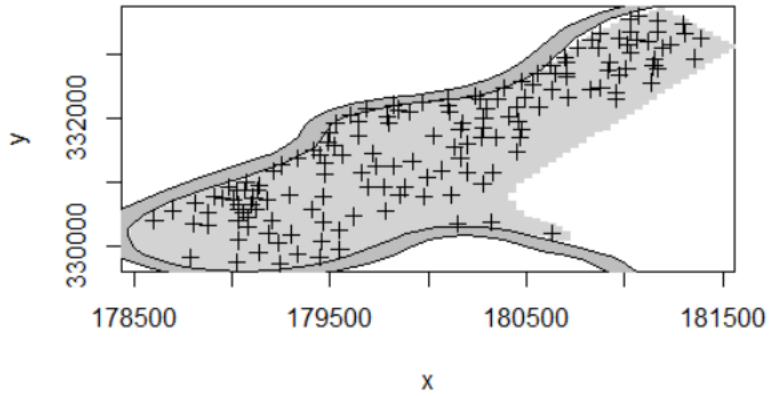
# Use the add=TRUE to add multiple layers to the plot:

image(meuse.raster,col="lightgrey")

plot(meuse.SpatialPolygons,col="grey",add=TRUE)

plot(meuse,add=TRUE))



# Visualize Spatial Data

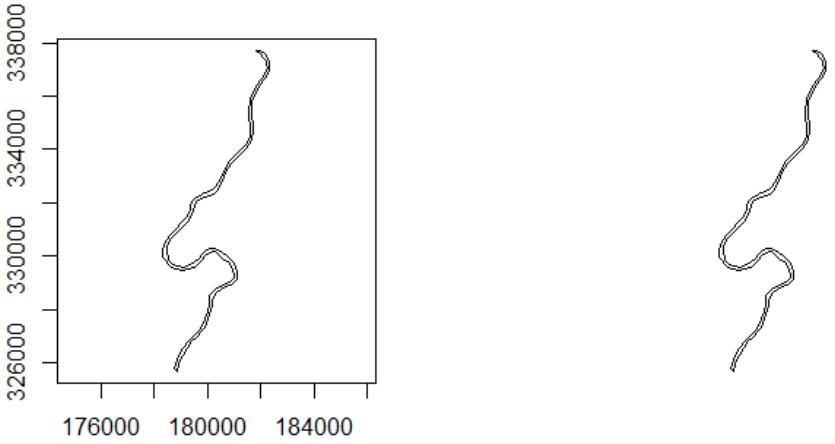
**Axes and Layout Elements**

# Layout() allows us to plot two maps arranged in columns:

layout(mat=matrix(c(1,2),nrow=1,ncol=2))

plot(meuse.SpatialPolygons,axes=TRUE) # Goes to the first column

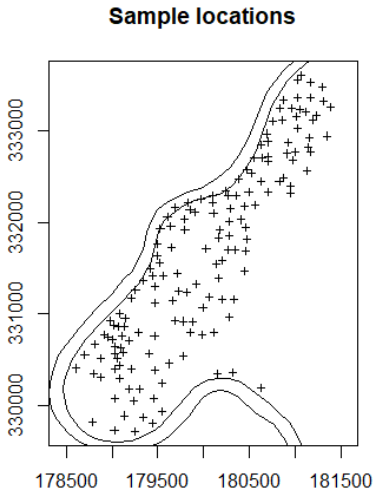
plot(meuse.SpatialPolygons,axes=FALSE) # Goes to the second column

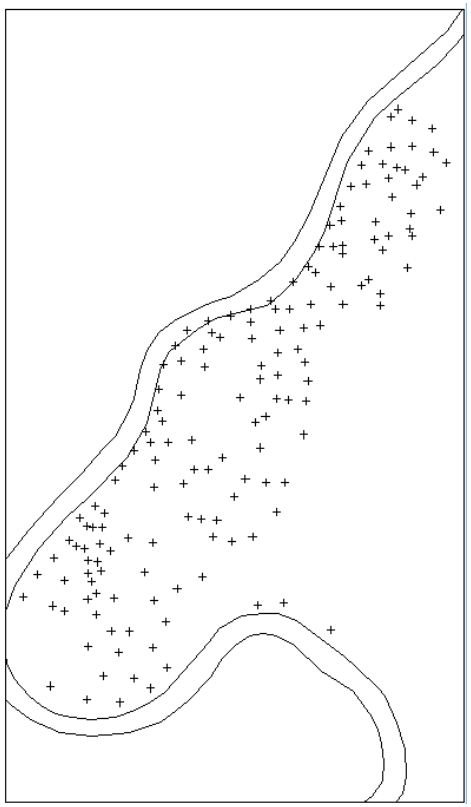


# Visualize Spatial Data

## Axes and Layout Elements

```
# Modify the plotting margins in units of lines of text
par(mar=c(0,0,0,0)+0.1)
plot(meuse,axes=FALSE,cex=0.6)
plot(meuse.SpatialPolygons,add=TRUE)
box()
```



# Visualize Spatial Data

## Axes and Layout Elements

# Degrees in Axes Labels and Reference Grid

```
# define the gridline
wrld_grd <- gridlines(wrld_sp,easts=c(-179,seq(-150,150,50),179.5),
                      norths=seq(-75,75,15),ndiscr=100)
# grid labels
at_sp <- gridat(wrld_sp,easts=0,norths=seq(-75,75,15),offset=0.3)
```