Week 3: Use R as GIS

Guofeng Cao (guofeng.cao@ttu.edu)

Basic R cheat sheet

R cheat sheet

Spatial objects in R

	Without attributes	With attributes
Points	SpatialPoints	SpatialPointsDataFrame
Lines	SpatialLines	SpatialLinesDataFrame
Polygons	SpatialPolygons	SpatialPolygonsDataFrame
Raster	SpatialGrid	SpatialGridDataFrame
Raster	SpatialPixels	SpatialPixelsDataFrame

Commonly used GIS-related packages

- Basic: `sp`
- Input and output: `rgdal`
- Mapping: `RColorBrewer`, `classInt`
- Raster: `raster`
- Overlay `sp`, `rgeos`

Open a spatial dataset

- Open a shapefile: `readShapePoly` or `readOGR`
- Open a raster: `raster`
- Open from a remote data repository: `getData`

```
# Open a shapefile
LubbockBlock<-readShapePoly("Data/LubbockBlockNew.shp") #read polygon shapefile
# or
LubbockBlock<-readOGR("./Data", "LubbockBlockNew") #read polygon shapefile
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "./Data", layer: "LubbockBlockNew"
## with 167 features
## It has 69 fields
```

```
class(LubbockBlock)
```

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

```
# Open a csv file with lat/lon information
HouseLocation<-read.csv("Data/HouseLatLon.csv") #read GPS data
class(HouseLocation)
```

```
## [1] "data.frame"
```

```
#Specifiy the coordinates to make it as a spatial dataset
coordinates(HouseLocation)<-c('Lon', 'Lat')
class(HouseLocation)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
# Open a raster/image
cropland<-raster("Data/Lubbock_CDL_2013_USDA.tif")
class(cropland)
```

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

```
# Get data from online
tmin <- getData("worldclim", var = "tmin", res = 10)  # this will download
class(tmin)
```

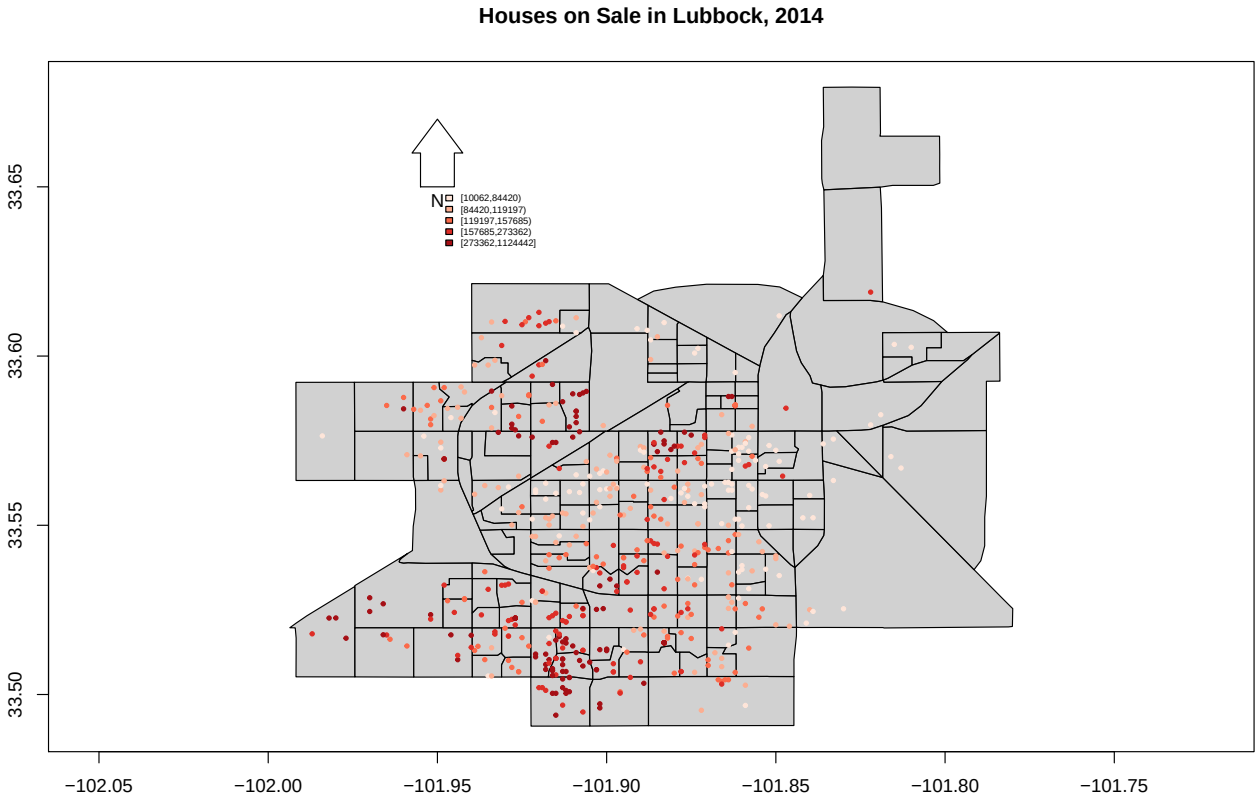
```
## [1] "RasterStack"
## attr(,"package")
## [1] "raster"
```

Mapping with R

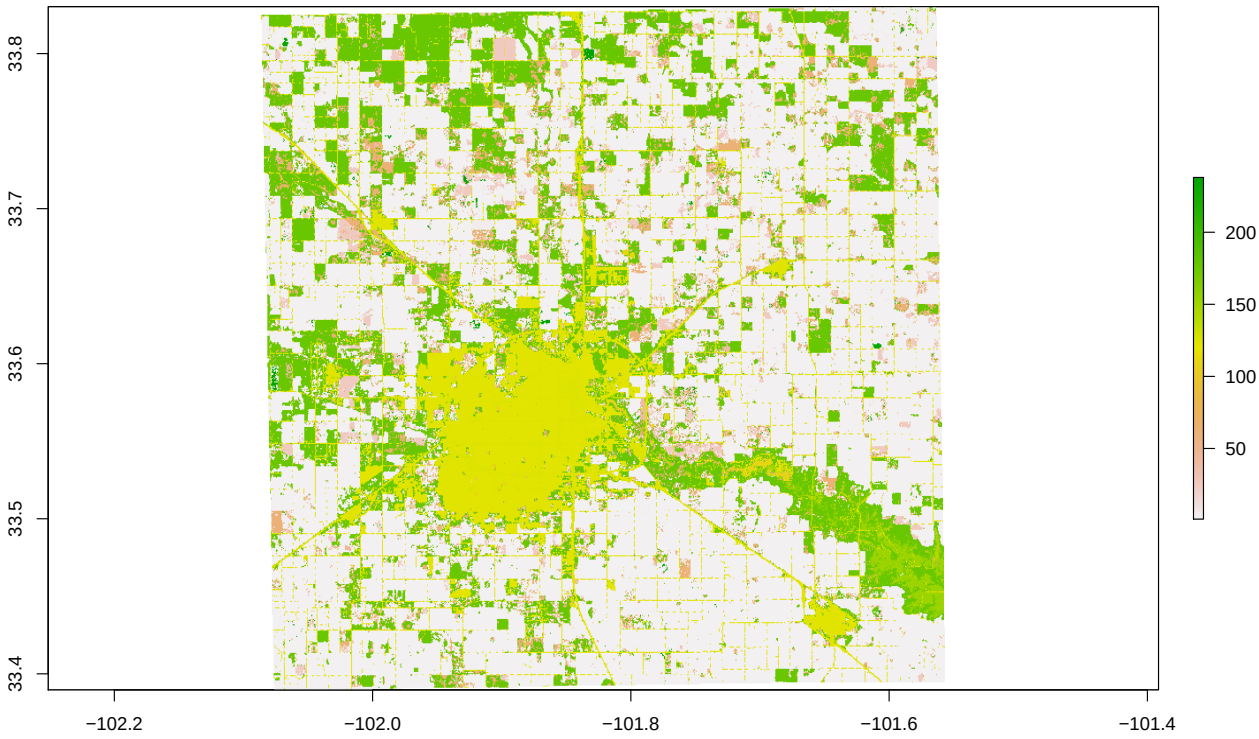
Basic Mapping

- Display a map: `plog`
- Color selection: `brewer.pal`, `classIntervals` and `findColours`
- Legend: `legend`
- North arrow: `north.arrow`

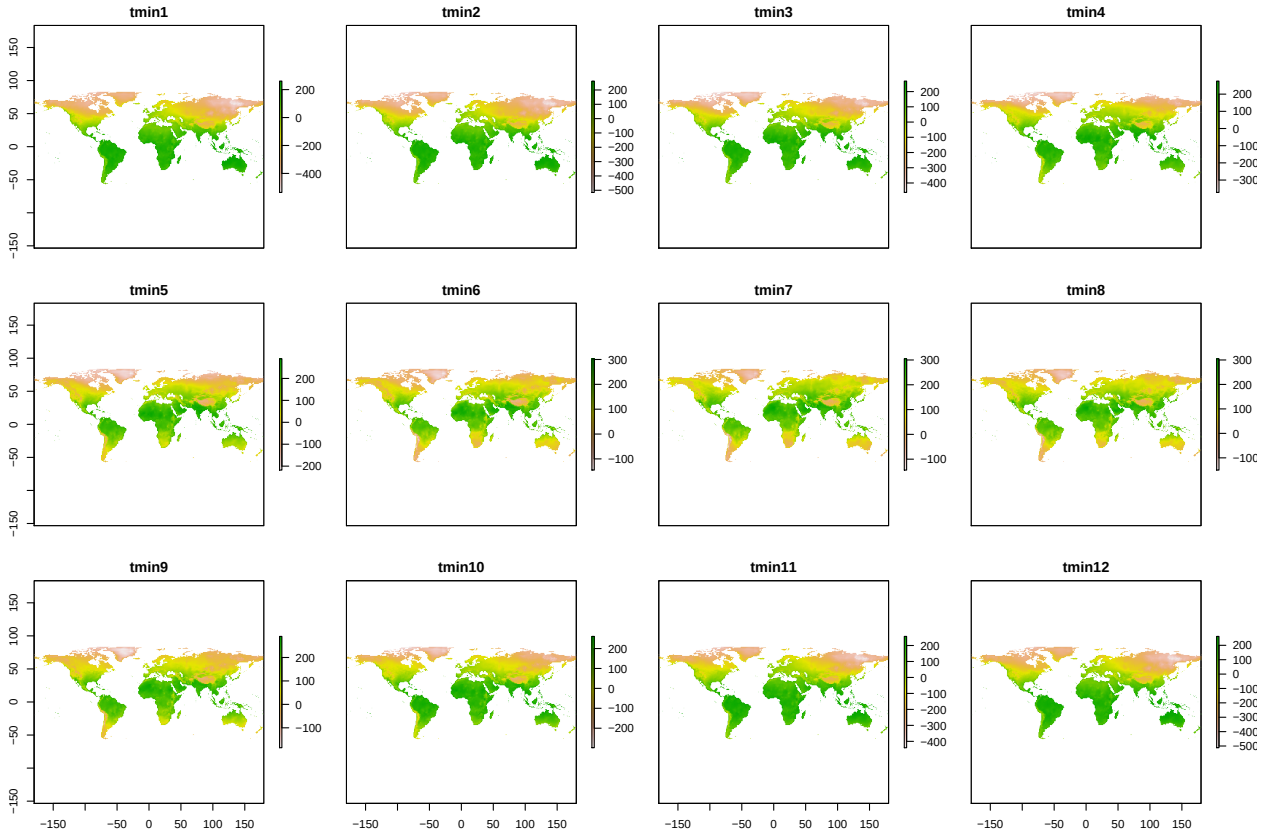
```
LubbockBlock<-readShapePoly("Data/LubbockBlockNew.shp") #read polygon shapefile
plot(LubbockBlock,axes=TRUE, col=alpha("gray70", 0.6)) #plot Lubbock block shapefile
#add title, scalebar, north arrow, and legend
HouseLocation<-read.csv("Data/HouseLatLon.csv") #read GPS data
price<-HouseLocation$TotalPrice
nclr<-5
priceclr<-brewer.pal(nclr, "Reds")
class<-classIntervals(price, nclr, style="quantile")
clocode<-findColours(class, priceclr)

points(HouseLocation$Lon, HouseLocation$Lat, pch=19, col=clocode, cex=0.5) #add houses on top of Lubboc
title(main="Houses on Sale in Lubbock, 2014")

legend(-101.95, 33.65, legend=names(attr(clocode, "table")), fill =attr(clocode, "palette"), cex=0.5, b
#map.scale(x=-101.85, y=33.49,0.001,"Miles",4,0.5,sfcol='red')
north.arrow(xb=-101.95, yb=33.65, len=0.005, lab="N")
```




```
#plot raster
plot(cropland)
```



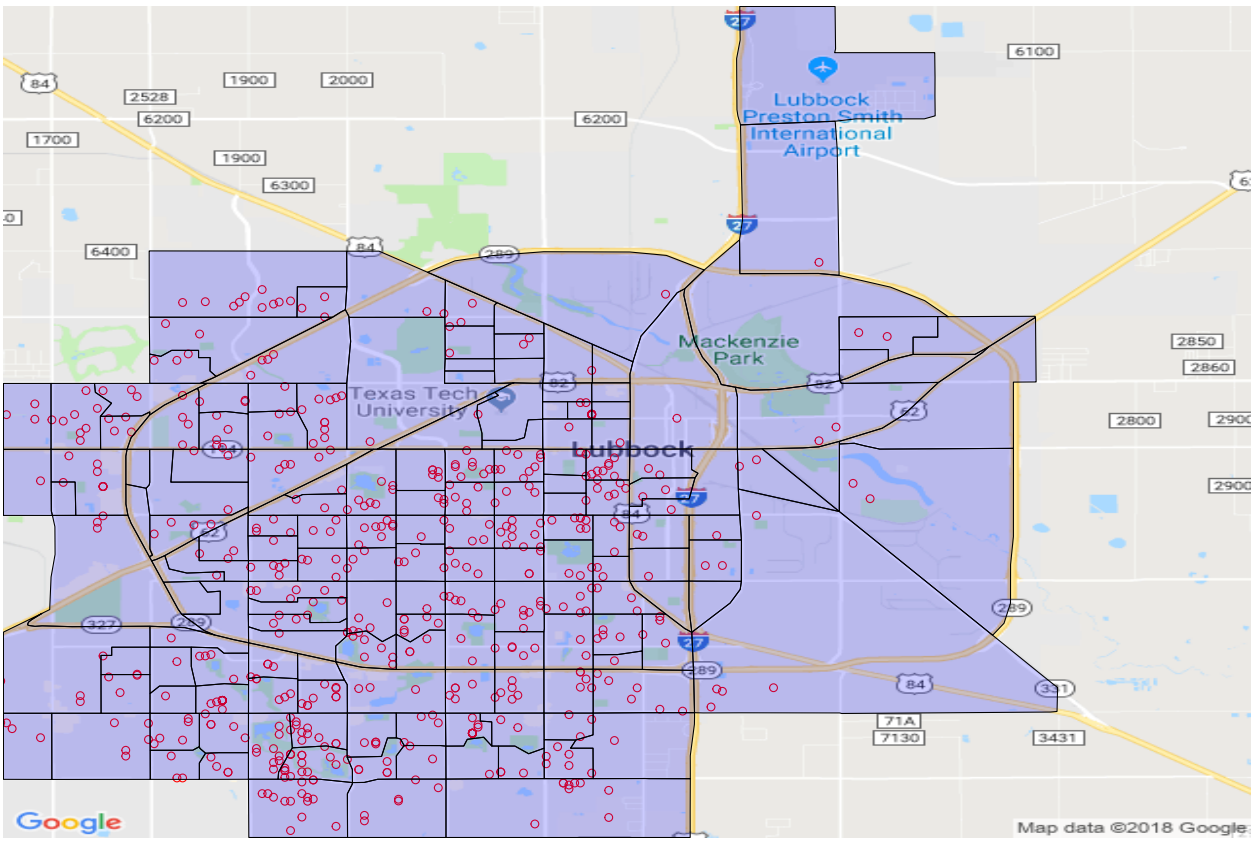

```
#plot raster stack
tmin <- getData("worldclim", var = "tmin", res = 10)  # this will download
plot(tmin)
```



Mapping with static Google Maps

- Use Google Maps as a base map: `GetMap` from library `RgoogleMaps`
- Make sure spatial data has right projection information

```
library(RgoogleMaps)
#lubbock=geocode('lubbock', output="latlon")
lubbock=c(33.583794, -101.855836)

newmap <- GetMap(center = lubbock, zoom = 12, destfile = "newmap.png", maptype = "roadmap")
#newmap <- GetMap(center = c(lubbock$lat, lubbock$lon), zoom = 12, destfile = "newmap.png", maptype = "

PlotOnStaticMap(newmap, lat=HouseLocation$Lat, lon=HouseLocation$Lon, col='red')
lubbock<-SpatialPolygons(LubbockBlock@polygons, proj4string=CRS("+init=EPSG:4326"))
PlotPolysOnStaticMap(newmap, lubbock, col=alpha('blue', 0.2))
```



Mapping with dynamic Google Maps

- Use Google Maps as a base map: `plotGoogleMaps` from library `plotGoogleMaps`
- Make sure spatial data has right projection information

```
library(plotGoogleMaps)

data(meuse)
coordinates(meuse)=~x+y
proj4string(meuse) = CRS('+init=epsg:28992')
plotGoogleMaps(meuse, filename='meuse.html')

HouseLocation<-read.csv("Data/HouseLatLon.csv") #read GPS data
coordinates(HouseLocation)<-c('Lon', 'Lat')
proj4string(HouseLocation)=CRS('+init=EPSG:4326')
plotGoogleMaps(HouseLocation, filename='house.html')

ic = iconlabels(meuse$zinc, height=12)
plotGoogleMaps(meuse, iconMarker=ic, mapTypeId='ROADMAP', filename='meuse2.html')

#plot raster
data(meuse.grid)
coordinates(meuse.grid)<-c('x', 'y')
meuse.grid<-as(meuse.grid, 'SpatialPixelsDataFrame')
proj4string(meuse.grid) <- CRS('+init=epsg:28992')
mapMeuseCl<- plotGoogleMaps(meuse.grid,zcol= 'dist',at=seq(0,0.9,0.1),colPalette= brewer.pal(9,"Reds"),
```

```
#plot polygons
proj4string(LubbockBlock)=CRS("+init=epsg:4326")
m<-plotGoogleMaps(LubbockBlock,zcol="Pop2010",filename= 'MyMap6.htm' , mapTypeId= ' TERRAIN ' ,colPalet

#plot line
meuse.grid<-as(meuse.grid, 'SpatialPixelsDataFrame')
im<-as.image.SpatialGridDataFrame(meuse.grid[ 'dist' ])
cl<-ContourLines2SLDF(contourLines(im))
proj4string(cl) <- CRS( '+init=epsg:28992')
mapMeuseCl<- plotGoogleMaps(cl,zcol= 'level' ,strokeWeight=1:9, filename= 'myMap6.htm',mapTypeId= 'ROAD
```

Changing map projections

- Query or specify projection information: `proj4string()`
- Change map project of a vector dataset: `spTransform()`
- Change map project of a raster dataset: `projectRaster()`

```
#project a vector

boudary=readShapePoly('Data/boundary');
proj4string(boudary) <-CRS("+proj=utm +zone=17 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0
proj4string(boudary)
```

```
## [1] "+proj=utm +zone=17 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

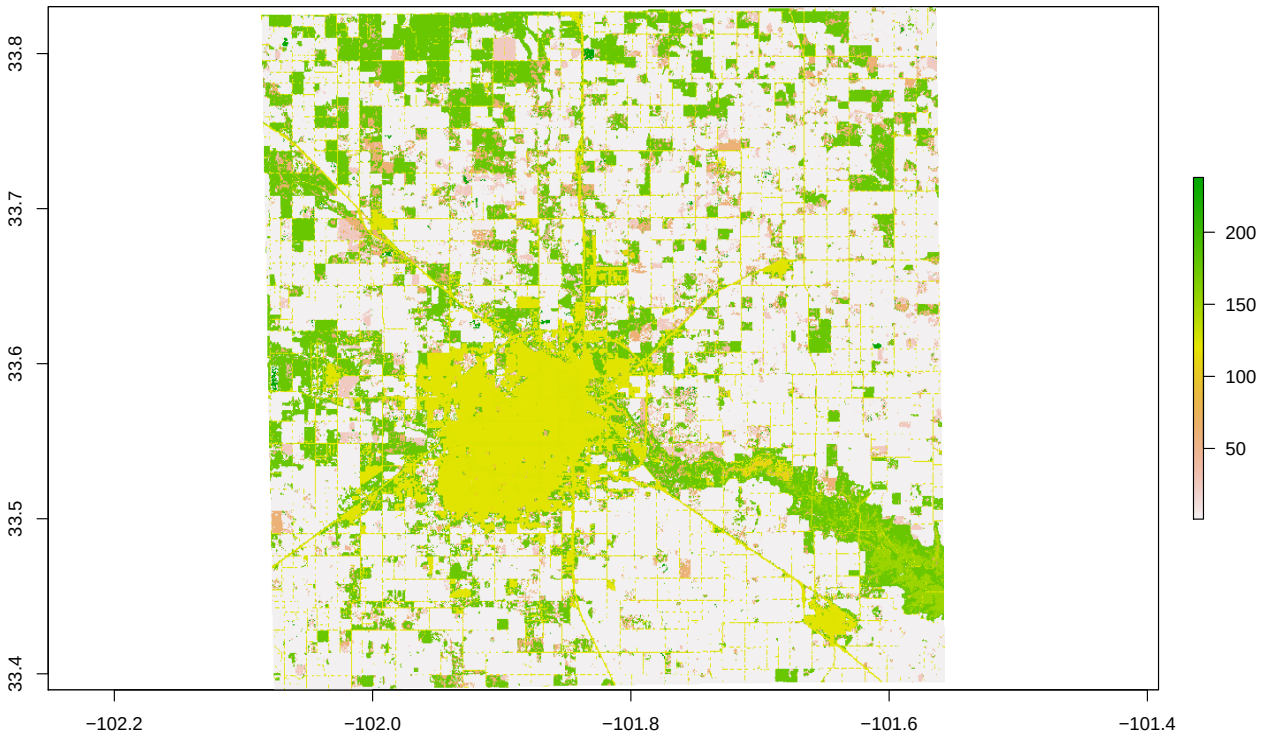
```
# EPSG code
## epsg:3857, web Merator
## epsg:4326, WGS84
boudaryProj<-spTransform(boudary, CRS("+init=epsg:3857"))
proj4string(boudaryProj)
```

```
## [1] "+init=epsg:3857 +proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +
```

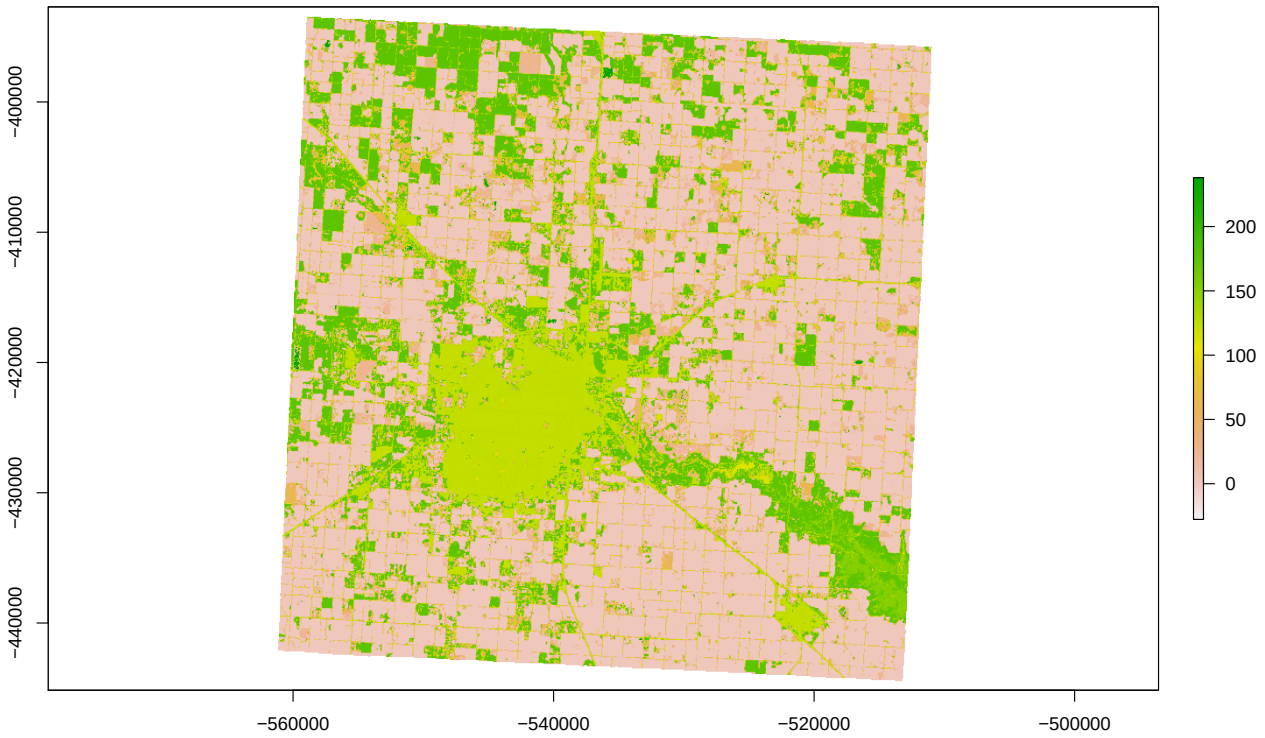
```
#project a raster
proj4string(cropland)
```

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

```
plot(cropland)
```




```
aea <- CRS("+init=ESRI:102003")   #Albert equal area
projCropland=projectRaster(cropland, crs=aea)
plot(projCropland)
```



Spatial analysis with R

Basic operation

- crop or erase

```
library(rgdal)
#subsetting a spatial dataframe
LubbockBlock<-readOGR("./Data", "LubbockBlockNew") #read polygon shapefile

selection = LubbockBlock[LubbockBlock$Pop2010>500,]
plot(selection)

#select by clicking
selected = click(LubbockBlock)

extent = drawExtent()

extent=as(extent,'SpatialPolygons')
proj4string(extent)=proj4string(selection)

# performace erase
plot(erase(selection, extent))


poly = drawPoly()
proj4string(poly) = proj4string(LubbockBlock)

# performe clip
cropselection = crop(LubbockBlock,poly)
plot(cropselection)
```

vector analysis (overlay)

- Overlay: over()

```
#project a vector

# Datasets
#  * CSV table of (fictionalized) brown bear sightings in Alaska, each
#    containing an arbitrary ID and spatial location specified as a
#    lat-lon coordinate pair.
#  * Polygon shapefile containing the boundaries of US National Parks
#    greater than 100,000 acres in size.

bears <- read.csv("Data/bear-sightings.csv")
coordinates(bears) <- c("longitude", "latitude")

# read in National Parks polygons
parks <- readOGR("Data", "10m_us_parks_area")
```

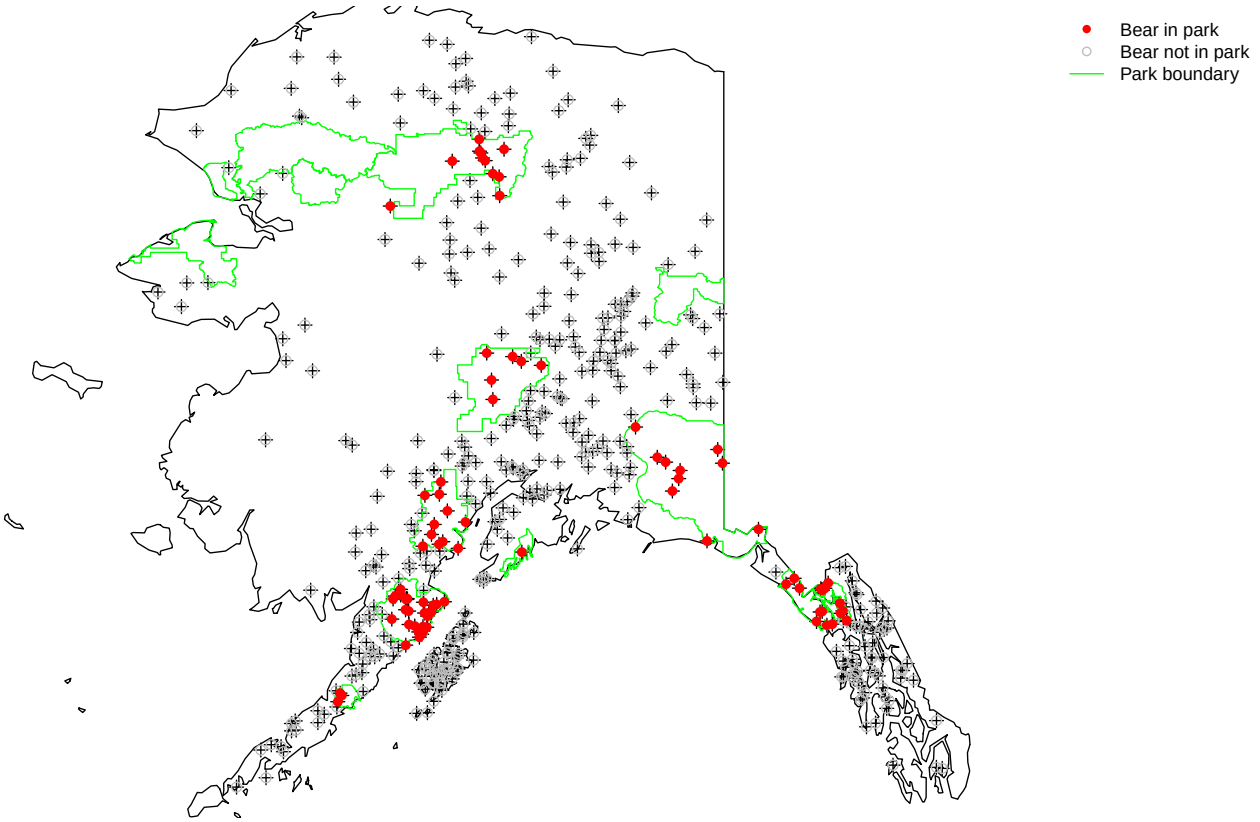
```
## OGR data source with driver: ESRI Shapefile
## Source: "Data", layer: "10m_us_parks_area"
## with 61 features
```

```
## It has 8 fields
```

```
# tell R that bear coordinates are in the same lat/lon reference system as the parks data
proj4string(bears) <- proj4string(parks)

# combine is.na() with over() to do the containment test; note that we
# need to "demote" parks to a SpatialPolygons object first
inside.park <- !is.na(over(bears, as(parks, "SpatialPolygons")))

# calculate what fraction of sightings were inside a park
mean(inside.park)
```

```
## [1] 0.1720648
```

```
## [1] 0.1720648

# determine which park contains each sighting and store the park name as an attribute of the bears data
bears$park <- over(bears, parks)$Unit_Name

# draw a map big enough to encompass all points, then add in park boundaries superimposed upon a map of
plot(bears)
map("world", region="usa", add=TRUE)
plot(parks, border="green", add=TRUE)
legend("topright", cex=0.85, c("Bear in park", "Bear not in park", "Park boundary"), pch=c(16, 1, NA), l
title(expression(paste(italic("Ursus arctos"), " sightings with respect to national parks")))

# plot bear points with separate colors inside and outside of parks
points(bears[!inside.park, ], pch=1, col="gray")
points(bears[inside.park, ], pch=16, col="red")
```




```
# write the augmented bears dataset to CSV
write.csv(bears, "bears-by-park.csv", row.names=FALSE)

# ...or create a shapefile from the points
writeOGR(bears, ".", "bears-by-park", driver="ESRI Shapefile")
```

Raster analysis

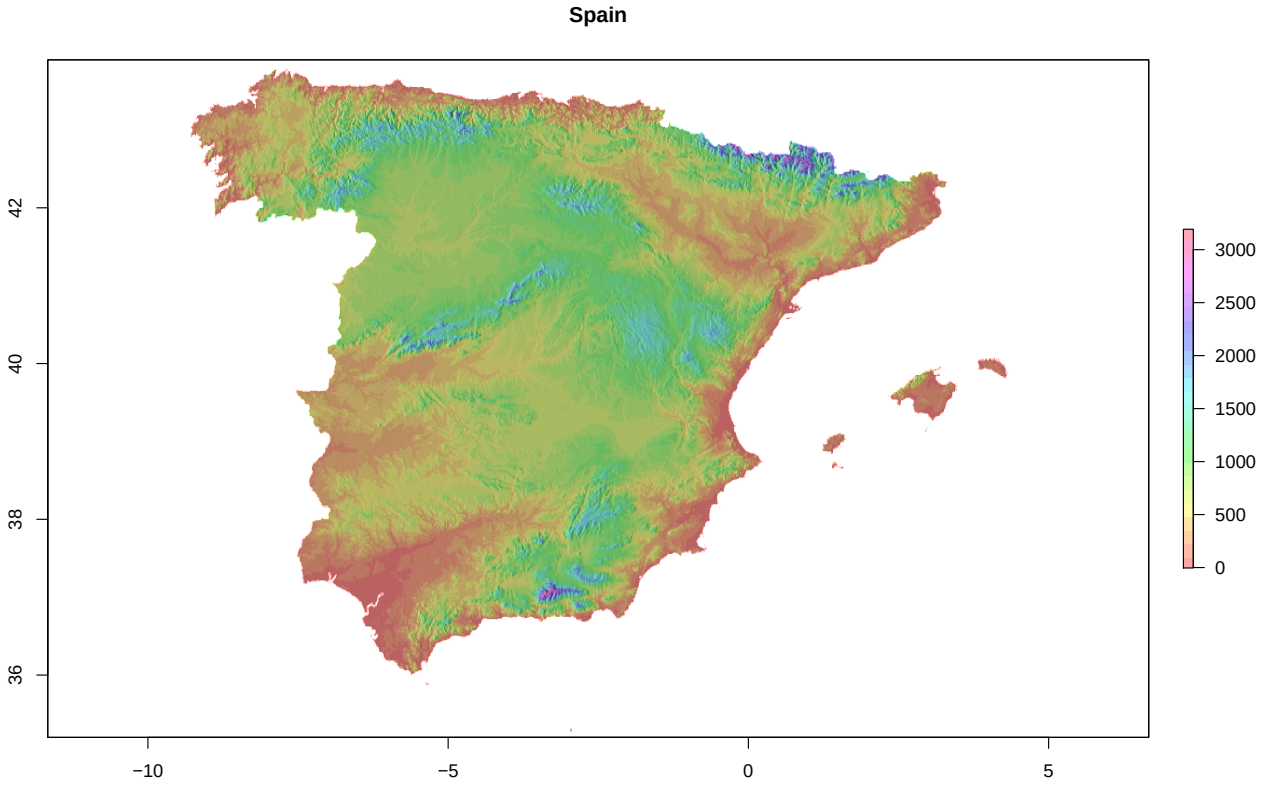
- Raster overlay, raster calculator
- Slope, aspect: `terrain()`
- Hill shade: `hillShade()`
- Contour lines: `contour()`
- Crop a raster: `crop()`

```
tmin=getData('worldclim', var='tmin', res=10)

# Raster calculator
diff=tmin$tmin1 - tmin$tmin10

## the following code is faster for large datasets.
overlay(tmin$tmin1, tmin$tmin10, fun=function(x,y){return (x-y)})
```

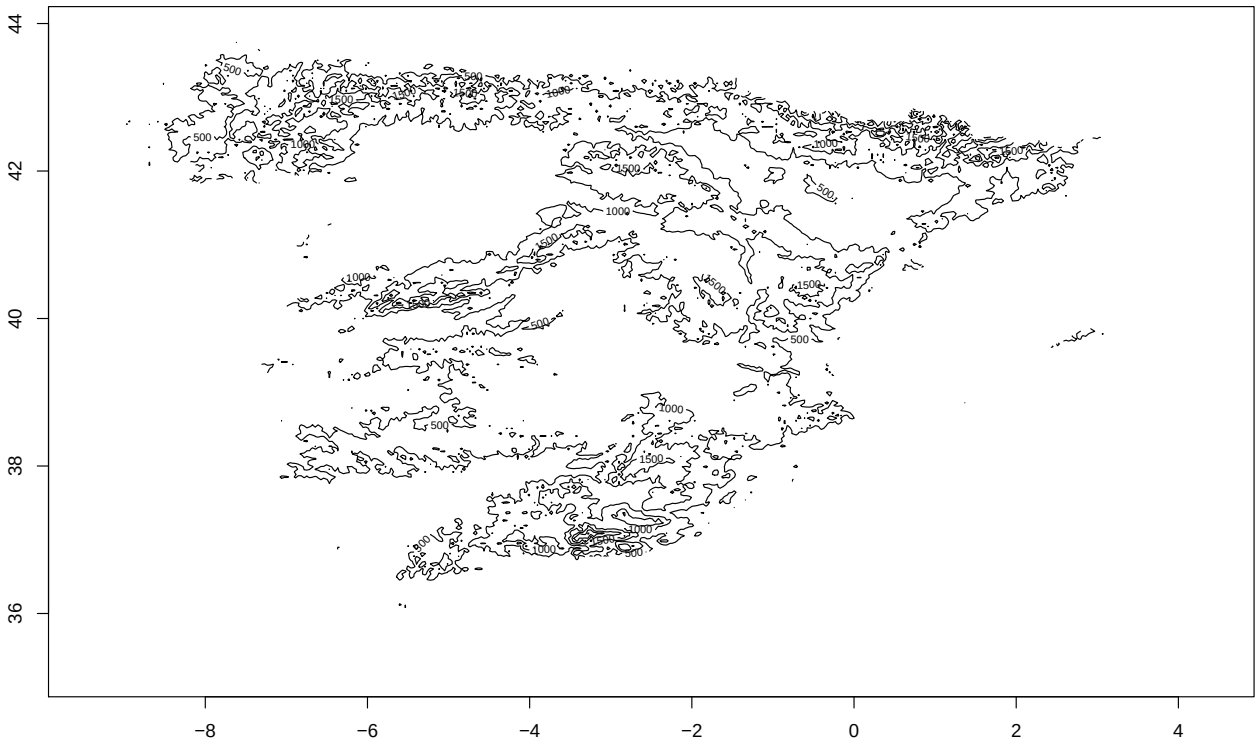
```
## class       : RasterLayer
## dimensions  : 900, 2160, 1944000  (nrow, ncol, ncell)
## resolution  : 0.1666667, 0.1666667  (x, y)
```

```
## extent      : -180, 180, -60, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -343, 102  (min, max)
```

```
elevation <- getData("alt", country = "ESP")
slope <- terrain(elevation, opt = "slope")
aspect <- terrain(elevation, opt = "aspect")
hill <- hillShade(slope, aspect, 40, 270)
plot(hill, col = grey(0:100/100), legend = FALSE, main = "Spain")
plot(elevation, col = rainbow(25, alpha = 0.35), add = TRUE)
```




```
#contours

contour(elevation)
```




```
#crop raster
plot(hill, col = grey(0:100/100), legend = FALSE, main = "Spain")
plot(elevation, col = rainbow(25, alpha = 0.35), add = TRUE)
extent=drawExtent()
cropElev <- crop(elevation, extent)
plot(cropElev)
```

Reading

Chapters 2-4 of Applied Spatial Data Analysis with R