Handling Geographic ‘Poly’ Data (Shapefiles) in R

Adam B. Smith

June 14, 2018

In this tutorial we will manipulate geographic “poly” data (e.g., polygons, lines, and points, often stored in *shapefile* format). You will learn:

* How to load/save a shapefile
* How to select components of and crop a shapefile
* How to overlay shapefiles so information from one is transferred to another # Some common geographic operations like calculating area.
* How to make a nice-ish map

# Libraries

Please install then load these packages:

library(sp) # classes for spatial objects  
library(raster) # manipulates rasters  
library(dismo) # species distribution modeling  
library(rgeos) # common geographic calculations  
library(geosphere) # for calculating geographic distances  
library(geojsonio) # for obtaining GeoJSON geographic data  
library(scales) # for rescaling data

You will also need the omnibus, enmSdm, and legendary packages, but these are available only from GitHub:

install.packages('devtools') # if you haven't done this already  
  
library(devtools)  
install\_github('adamlilith/omnibus')  
install\_github('adamlilith/legendary')  
install\_github('adamlilith/enmSdm')

library(omnibus)  
library(legendary)  
library(enmSdm)

# Loading, selecting, projecting, and plotting shapefiles

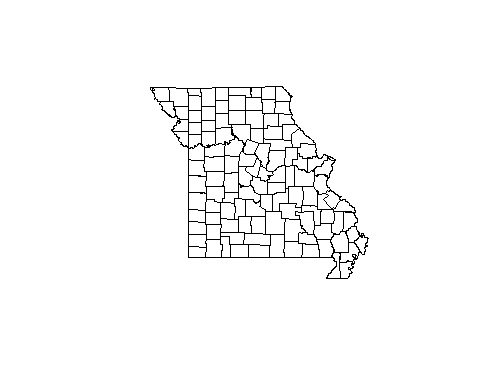
We will do our analysis on the US state of Missouri. To start we’ll download data for the entire US from the Database of Global Administrative Areas (GADM).

gadm <- raster::getData('GADM', country='USA', level=2)  
plot(gadm)



This is a little too much–let’s choose just Missouri.

mo <- gadm[gadm$NAME\_1 == 'Missouri', ]  
plot(mo)



Note that the Missouri object is also a data frame, meaning you can do most operations on it that you could do on a data frame:

nrow(mo)

## [1] 114

names(mo)

## [1] "OBJECTID" "ID\_0" "ISO" "NAME\_0" "ID\_1"   
## [6] "NAME\_1" "ID\_2" "NAME\_2" "HASC\_2" "CCN\_2"   
## [11] "CCA\_2" "TYPE\_2" "ENGTYPE\_2" "NL\_NAME\_2" "VARNAME\_2"

Calling just the name of the object will give you important information about it:

mo

## class : SpatialPolygonsDataFrame   
## features : 114   
## extent : -95.77441, -89.10059, 35.99509, 40.61403 (xmin, xmax, ymin, ymax)  
## coord. ref. : +proj=longlat +datum=WGS84 +no\_defs +ellps=WGS84 +towgs84=0,0,0   
## variables : 15  
## names : OBJECTID, ID\_0, ISO, NAME\_0, ID\_1, NAME\_1, ID\_2, NAME\_2, HASC\_2, CCN\_2, CCA\_2, TYPE\_2, ENGTYPE\_2, NL\_NAME\_2, VARNAME\_2   
## min values : 1487, 244, USA, United States, 26, Missouri, 1487, Adair, US.MO.AD, NA, , County, County, ,   
## max values : 1600, 244, USA, United States, 26, Missouri, 1600, Wright, US.MO.WY, NA, , County, County, ,

Here you can see the object type (SpatialPolygonsDataFrame), number of “sub-polygons” (counties in this case–there are 114 of them), the geographic extent, coordinate reference system (more on this immediately below), number of columns (15), and an abbreviated version of the data table (the data frame part).

If you are using multiple map objects to do an analysis, you almost always need to put them into a common **coordinate reference system** (CRS), which is a particular geometric convention used to locate things (e.g., x and y coordinates or longitude and latitude). Map objects sometimes also have a **projection**, which is a transformation from a plane (i.e., your screen) to a simulated sphere or ellipsoid (i.e., the Earth). CRSs and projections are specified by a **proj4 string** which you can see by calling the name of the object (as above) or using

raster::projection(mo)

## [1] "+proj=longlat +datum=WGS84 +no\_defs +ellps=WGS84 +towgs84=0,0,0"

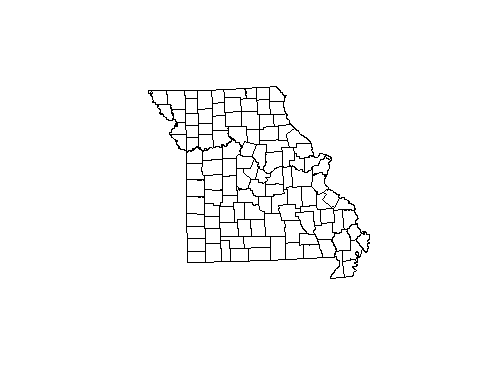
This particular proj4 string is for the “WGS84”" CRS, which used quite often. It’s an *unprojected* CRS, meaning that it forces the map to look good on a flat surface (your screen), but it doesn’t accurately reflect distance, area, or angles. Projected CRSs can correct *one* of these aspects at a time, but none can correct more than one. We will project the Missouri polygon object to the Albers North America Equal-Area CRS. The proj4 string for this is

enmSdm::getCRS('albersNA')

## [1] "+proj=aea +lat\_1=29.5 +lat\_2=45.5 +lat\_0=37.5 +lon\_0=-96 +x\_0=0 +y\_0=0 +ellps=GRS80 +datum=NAD83 +units=m +no\_defs"

So let’s project Missouri to this CRS. (Note: I have found that you usually *must* put the package (sp::) part before the function call spTransform because there is a function with the same name in the rgdal library which is often automatically loaded when you use the packages we loaded above.)

mo <- sp::spTransform(mo, getCRS('albersNA', TRUE))  
plot(mo)



The projected map of Missouri looks only *slightly* different from the unprojected map, but this is because the state is fairly small compared to the curvature of the Earth.

## Creating a “points”" shape from raw data

We will also use a database of major Ozarkian springs surveyed by Julian Steyermark in the 1920s-2030s. You can obtain it from [www.earthSkySea.org/!ecology/data/steyermarkSprings.zip](file:///C:\Ecology\Drive\R\rTutorials\www.earthSkySea.org\!ecology\data\steyermarkSprings.zip). Unzip this and load it into R as a data frame.

steyer <- './Steyermark Springs 2014-07-08 Updated taxonomy.csv', as.is=TRUE)  
head(steyer)

The fields spring1, spring2, etc. are the name of the spring (sometimes there is more than one name), county is county, latMoDnr and longMoDnr are the latitude and longitude of the springs obtained from the Missouri Department of Natural Resources, richness is number of plant taxa in the springs, and the remainder of the columns are presence/absence of the given taxa in the springs. Please be aware that subsequent work by George Yatskievych (author of the revised version of Steyermark’s *Flora of Missouri*, has found that Steyermark mis-identified some plants, but for now we’ll use the Steyermark’s data as-is.

Let’s convert this to a **SpatialPointsDataFrame** (a points “shapefile”):

springs <- sp::SpatialPointsDataFrame(  
 coords=cbind(steyer$longMoDnr, steyer$latMoDnr),  
 data=steyer,  
 proj4string=enmSdm::getCRS('wgs84', TRUE)  
)

## Error in .local(obj, ...): NA values in coordinates

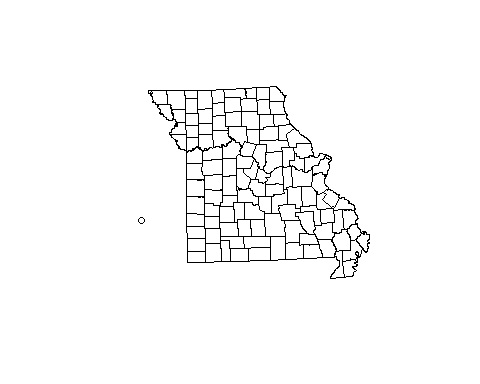
Oops! I was not able to locate some of the springs, so they don’t have coordinates (in georeferencing the springs, I found Steyermark’s “compass” didn’t always seem to point north…). In this case I happen to know the coordinates have an unprojected (WGS84) CRS. Also, note that unlike how we often say “latitude and longitude,” in R we put longitude first (the x-coordinate) and latitude second (the y-coordinate).

steyer <- steyer[!is.na(steyer$longMoDnr), ]  
  
steyer <- sp::SpatialPointsDataFrame(  
 coords=cbind(steyer$longMoDnr, steyer$latMoDnr),  
 data=steyer,  
 proj4string=enmSdm::getCRS('wgs84', TRUE)  
)  
  
steyer

## class : SpatialPointsDataFrame   
## features : 86   
## extent : -94.3723, -89.98519, 36.55568, 38.34891 (xmin, xmax, ymin, ymax)  
## coord. ref. : +proj=longlat +datum=WGS84 +no\_defs +ellps=WGS84 +towgs84=0,0,0   
## variables : 97  
## names : num, spring1, spring2, spring3, spring4, county, latMoDnr, longMoDnr, richness, Acorus\_calamus, Agrostis\_perennans, Agrostis\_stolonifera, Alisma\_subcordatum, Anacharis\_occidentalis, Batrachyspermum\_sp, ...   
## min values : 1, Alley, , , , Barry, 36.55568, -89.98519, 1, 0, 0, 0, 0, 0, 0, ...   
## max values : 92, Yancy Mill, Waynesville, Freeman, Spring Creek, Wayne, 38.34891, -94.37230, 21, 1, 1, 1, 1, 1, 1, ...

Let’s plot the springs!

plot(mo)  
points(steyer)



Weird! What happened? Here’s a hint: compare the CRS of the two objects:

raster::projection(mo)

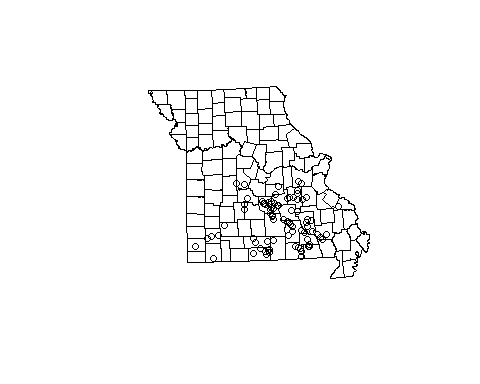
## [1] "+proj=aea +lat\_1=29.5 +lat\_2=45.5 +lat\_0=37.5 +lon\_0=-96 +x\_0=0 +y\_0=0 +ellps=GRS80 +datum=NAD83 +units=m +no\_defs +towgs84=0,0,0"

raster::projection(steyer)

## [1] "+proj=longlat +datum=WGS84 +no\_defs +ellps=WGS84 +towgs84=0,0,0"

We need to project the steyer spatial object to the Albers North America Equal-Area CRS. We can get the proj4 string from the enmSdm::getCRS() function or from the Missouri object directly using raster::projection():

steyer <- sp::spTransform(steyer, raster::projection(mo))  
  
plot(mo)  
points(steyer)



## Work with GeoJSON objects, do overlays, and calculate polygon area

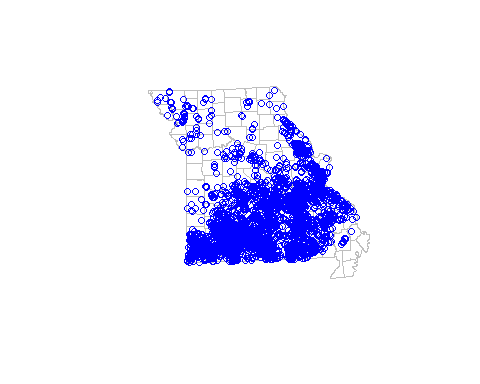
Now let’s get a data set for \*\*all\* (known) springs in Missouri. We can download this from the Missouri DNR’s website [manually](https://data-msdis.opendata.arcgis.com/datasets/0745217e2674450e8afc6a40fa4201ff_0) or do it in R using a GeoJSON service.

springs <- geojsonio::geojson\_read('https://opendata.arcgis.com/datasets/0745217e2674450e8afc6a40fa4201ff\_0.geojson', what='sp')  
  
springs

## class : SpatialPointsDataFrame   
## features : 4456   
## extent : -95.50388, -89.4751, 36.49785, 40.55278 (xmin, xmax, ymin, ymax)  
## coord. ref. : +proj=longlat +datum=WGS84 +no\_defs +ellps=WGS84 +towgs84=0,0,0   
## variables : 28  
## names : OBJECTID, GISID, NAMEPRIM, NAMESEC, FIPS, COUNTY, QUADNUM, QUADNAME, LOCATION, TOWNSHIP, RANGE, ELEVATION, NEEDFC, REPORTBY, RPTDATE, ...   
## min values : 1, 10001, ABBOT SPRING, , 5049, ADAIR, 36089G8, ALBANY SOUTH, , , , 310, , , 1972/02/19 00:00:00+00, ...   
## max values : 4456, 9490001, ZANONI SPRING, WORKMAN SPRING, 40041, WRIGHT, 40095E1, YANCY MILLS, W1/2, NE1/4, SW1/4 (projected), 66N, 9W, 1600, FIELD CHECK NEEDED, Vaughn, 2009/11/17 00:00:00+00, ...

GeoJSON objects are always assumed to come in the WGS84 CRS.

springs <- sp::spTransform(springs, raster::projection(mo))  
  
plot(mo, border='gray')  
points(springs, col='blue')



That is a lot of springs, especially in the Ozarks! But is spring **density** really higher there? Now, let’s calculate the density of springs in each county. First, we’ll calculate the area of each county, then count how many springs there are in each.

areas <- rgeos::gArea(mo)  
areas

## [1] 1.80569e+11

That didn’t seem to work… the problem is that it gave us the area of all of Missouri, not each county. To get the area of each county, use:

areas <- rgeos::gArea(mo, byid=TRUE)  
head(areas)

## 1487 1488 1489 1490 1491 1492   
## 1456944324 1124437502 1430379620 1800353350 2059209962 1532312706

There is one value per county. You can infer this from:

length(areas)

## [1] 114

nrow(mo)

## [1] 114

**Special note**: *In this case* we are safe assuming that each value here represents a single county. *However*, some states have counties represented by multiple polygons, so you would get one value per polygon. You would have to do a **union** operation which merges polygons (see ?rgeos::gUnaryUnion or ?rgeos::gUnion).

What are the units of area? You can find out by looking at the CRS of the Missouri object:

raster::projection(mo)

## [1] "+proj=aea +lat\_1=29.5 +lat\_2=45.5 +lat\_0=37.5 +lon\_0=-96 +x\_0=0 +y\_0=0 +ellps=GRS80 +datum=NAD83 +units=m +no\_defs +towgs84=0,0,0"

Do you see the part that says +units=m? This means that the map units are meters, so area is in meters-squared. Let;’s convert to km^2 and add this as a field to the springs shape object:

areas <- areas / 1000000  
mo$area\_km2 <- areas  
head(mo)

## OBJECTID ID\_0 ISO NAME\_0 ID\_1 NAME\_1 ID\_2 NAME\_2 HASC\_2  
## 1487 1487 244 USA United States 26 Missouri 1487 Adair US.MO.AD  
## 1488 1488 244 USA United States 26 Missouri 1488 Andrew US.MO.AN  
## 1489 1489 244 USA United States 26 Missouri 1489 Atchison US.MO.AT  
## 1490 1490 244 USA United States 26 Missouri 1490 Audrain US.MO.AU  
## 1491 1491 244 USA United States 26 Missouri 1491 Barry US.MO.BR  
## 1492 1492 244 USA United States 26 Missouri 1492 Barton US.MO.BT  
## CCN\_2 CCA\_2 TYPE\_2 ENGTYPE\_2 NL\_NAME\_2 VARNAME\_2 area\_km2  
## 1487 NA County County 1456.944  
## 1488 NA County County 1124.438  
## 1489 NA County County 1430.380  
## 1490 NA County County 1800.353  
## 1491 NA County County 2059.210  
## 1492 NA County County 1532.313

Now we’ll do an **overlay** operation that uses one spatial object to extract data from another. We’ll do this to calculate the density of springs in each county.

countiesBySprings <- sp::over(springs, mo)  
  
head(countiesBySprings)

## OBJECTID ID\_0 ISO NAME\_0 ID\_1 NAME\_1 ID\_2 NAME\_2 HASC\_2 CCN\_2  
## 1 1525 244 USA United States 26 Missouri 1525 Greene US.MO.GE NA  
## 2 1525 244 USA United States 26 Missouri 1525 Greene US.MO.GE NA  
## 3 1525 244 USA United States 26 Missouri 1525 Greene US.MO.GE NA  
## 4 1525 244 USA United States 26 Missouri 1525 Greene US.MO.GE NA  
## 5 1525 244 USA United States 26 Missouri 1525 Greene US.MO.GE NA  
## 6 1525 244 USA United States 26 Missouri 1525 Greene US.MO.GE NA  
## CCA\_2 TYPE\_2 ENGTYPE\_2 NL\_NAME\_2 VARNAME\_2 area\_km2  
## 1 County County 1772.57  
## 2 County County 1772.57  
## 3 County County 1772.57  
## 4 County County 1772.57  
## 5 County County 1772.57  
## 6 County County 1772.57

nrow(countiesBySprings)

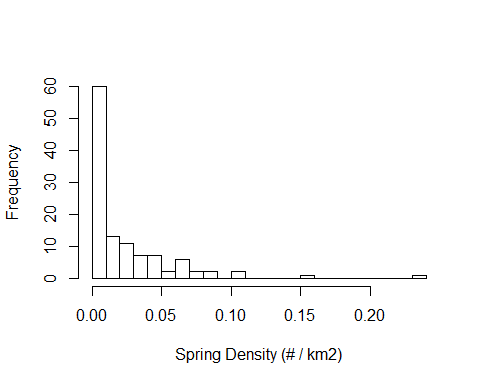
## [1] 4456

nrow(springs)

## [1] 4456

You can see there’s one line per spring, and that lines repeat. In this case the lines in countiesBySpring indicate which county each of the ~4500 springs is in. We can now use this to enumerate the number of springs per county and calculate their density.

mo$springs <- NA  
counties <- unique(mo$NAME\_2)  
  
for (county in counties) {  
   
 numSprings <- sum(countiesBySprings$NAME\_2 == county, na.rm=TRUE)  
 mo$springs[mo$NAME\_2 == county] <- numSprings  
   
}  
  
mo$springDensity\_numPerKm2 <- mo$springs / mo$area\_km2  
hist(mo$springDensity\_numPerKm2, breaks=20, xlab='Spring Density (# / km2)', main='')



Most counties have low spring density. To visualize this, let’s create a **chloropleth**, a map with areas colored or shaded in proportion to some numerical value (e.g., the density of springs). We will first have to make a color scale based on rescaling spring density.

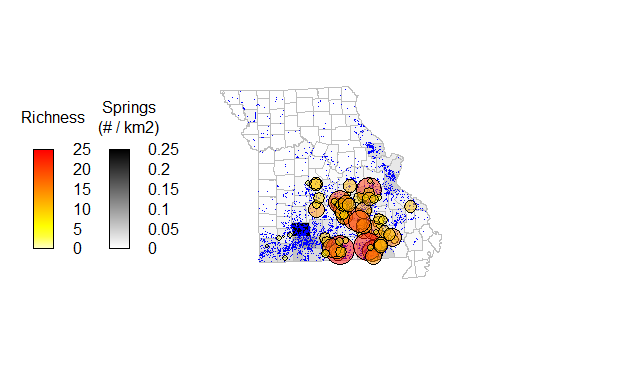
# rescale spring density to range [0, 100]  
springDensity <- mo$springDensity\_numPerKm2  
springDensityScaled <- springDensity / max(springDensity)  
springDensityScaled <- round(100 \* springDensityScaled)  
  
# get grayscale palette for spring density  
shades <- rev(gray.colors(101, start=0, end=1))  
countyCols <- shades[springDensityScaled + 1]

Let’s do the same for taxon richness in Steyermark’s spring data.

richness <- steyer$richness  
richnessScaled <- richness / max(richness)  
richnessScaled <- ceiling(100 \* richnessScaled)  
  
shades <- rev(heat.colors(101))  
steyerCols <- shades[richnessScaled + 1]

Now, let’s make a map that displays: \* Missouri counties with shade indicative of spring density  
\* A legend for spring density  
\* Steyermark’s springs with the size of each symbols proportional to number of plant taxa he found in each spring

plot(mo, col=countyCols, border='gray')  
points(springs, col='blue', pch='.')  
points(steyer, pch=21, cex=steyer$richness / 5, bg=scales::alpha(steyerCols, 0.5))  
  
# create legend for Steyermark springs  
labs <- pretty(richness)  
  
legendary::legendGrad(  
 'left',  
 inset=-0.2,  
 height=0.8,  
 width=0.2,  
 gradAdjX=c(0.3, 0.5),  
 gradAdjY=c(0.3, 0.9),  
 labels=labs,  
 labAdj=0.6,  
 col=rev(heat.colors(10)),  
 title='Richness',  
 boxBg=NULL,  
 boxBorder=NULL,  
 xpd=NA  
)  
  
# create legend for counties  
labs <- pretty(springDensity)  
  
legendary::legendGrad(  
 'left',  
 height=0.8,  
 width=0.2,  
 gradAdjX=c(0.3, 0.5),  
 gradAdjY=c(0.3, 0.9),  
 labels=labs,  
 labAdj=0.6,  
 col=c('white', 'black'),  
 title='Springs\n(# / km2)',  
 boxBg=NULL,  
 boxBorder=NULL,  
 xpd=NA  
)



Finally, we can save the Steyermark spatial object. The smallest kind of file will be a normal .RData file (or .RDS file):

save(steyer, file=‘./Steyermark Springs.Rdata’)

But we can also save it in “shapefile”" format for importing into a GIS program like ArcMap or QGIS:

dismo::shapefile(steyer, ‘./steyermarkSprings’)

**FINIS!**