Zoo955 - Working With Shapefiles

Hilary Dugan

February 6, 2018

Download data

Easy to search for 'spatial data' under NTL categories. Today we'll be working with Lake Mendota spatial data including the lake boundary, bathymetry/contours, and watershed shapefiles.

Watersheds: https://lter.limnology.wisc.edu/dataset/north-temperate-lakes-lter-yahara-lakes-district-lake-watersheds

Bathymetry: https://lter.limnology.wisc.edu/dataset/north-temperate-lakes-lter-yahara-lakes-district-bathymetry

Lakes: https://lter.limnology.wisc.edu/dataset/north-temperate-lakes-lter-yahara-lakes-district-boundary

Extract Lake Mendota data

Read in shapefiles using `readOGR`. Because there is an attached data.frame, you can use `stringsAsFactors = F` just like a normal data.frame

```
library(rgdal)
lakes = readOGR('Data/yld_study_lakes.shp',layer = 'yld_study_lakes',stringsAsFactors = F)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "Data/yld_study_lakes.shp", layer: "yld_study_lakes"
## with 4 features
## It has 9 fields
```

```
summary(lakes)
```

```
## Object of class SpatialPolygonsDataFrame
## Coordinates:
##        min    max
## x 547589.4 574950
## y 286020.8 313254
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=0 +lon_0=-90 +k=0.9996 +x_0=520000
## +y_0=-4480000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m
## +no_defs]
## Data attributes:
##       AREA            PERIMETER         SHAID_         SHAID_ID
##  Min.   :  803713   Min.   : 4063   Min.   :72988   Min.   :73642
##  1st Qu.: 1221591   1st Qu.: 7619   1st Qu.:75075   1st Qu.:75752
##  Median : 7474258   Median :17629   Median :76038   Median :76739
##  Mean   :13833562   Mean   :19434   Mean   :75372   Mean   :76065
##  3rd Qu.:20086229   3rd Qu.:29444   3rd Qu.:76334   3rd Qu.:77052
##  Max.   :39582017   Max.   :38417   Max.   :76422   Max.   :77140
##     SHAID_NO         SHAIDNAME            LAKEID
##  Min.   : 8000298   Length:4           Length:4
##  1st Qu.: 8000370   Class :character   Class :character
##  Median : 8000412   Mode  :character   Mode  :character
##  Mean   : 8501470
```

```
##  3rd Qu.: 8501511
##  Max.   :10004757
##   LAKE_NAME              WBIC
##  Length:4           Min.   :804600
##  Class :character   1st Qu.:804900
##  Mode  :character   Median :805200
##                     Mean   :850025
##                     3rd Qu.:850325
##                     Max.   :985100
```

We can see that there are many attributes associated with the shapefiles. Since we're interested in Lake Mendota, we can subset based on SHAIDNAME, LAKEID, or LAKE_NAME

```
mendota = lakes[lakes@data$LAKEID == 'ME',]
```

The watersheds and bathymetry are already subsetted, which is handy

```
mendota.ws = readOGR('Data/YaharaBasins/Mendota_Basin.shp',layer = 'Mendota_Basin',stringsAsFactors = F
mendota.bathy = readOGR('Data/Bathymetry/mendota-contours-all.shp',layer = 'mendota-contours-all',verbos
```

SpatialPolygonsDataFrame Structure

The structure of `SpatialPolygonsDataFrame` make a lot more sense when you take a minute to look at the internal structure. It gets overwhelming when there are lots of polygons, so let's start with our Mendota object

```
str(mendota)
```

```
## Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
##   ..@ data       :'data.frame':  1 obs. of  9 variables:
##   .. ..$ AREA     : num 39582017
##   .. ..$ PERIMETER: num 38416
##   .. ..$ SHAID_   : int 75771
##   .. ..$ SHAID_ID : int 76455
##   .. ..$ SHAID_NO : int 8000298
##   .. ..$ SHAIDNAME: chr "Lake Mendota"
##   .. ..$ LAKEID   : chr "ME"
##   .. ..$ LAKE_NAME: chr "Lake Mendota"
##   .. ..$ WBIC     : int 805400
##   ..@ polygons   :List of 1
##   .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
##   .. .. .. ..@ Polygons :List of 1
##   .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
##   .. .. .. .. .. .. ..@ labpt  : num [1:2] 567199 292638
##   .. .. .. .. .. .. ..@ area   : num 39582017
##   .. .. .. .. .. .. ..@ hole   : logi FALSE
##   .. .. .. .. .. .. ..@ ringDir: int 1
##   .. .. .. .. .. .. ..@ coords : num [1:1136, 1:2] 570842 570862 570887 570928 570948 ...
##   .. .. .. ..@ plotOrder: int 1
##   .. .. .. ..@ labpt    : num [1:2] 567199 292638
##   .. .. .. ..@ ID       : chr "1"
##   .. .. .. ..@ area     : num 39582017
##   ..@ plotOrder  : int 1
##   ..@ bbox       : num [1:2, 1:2] 562020 289177 571477 297311
##   .. ..- attr(*, "dimnames")=List of 2
```

```
##   .. .. ..$ : chr [1:2] "x" "y"
##   .. .. ..$ : chr [1:2] "min" "max"
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
##   .. .. ..@ projargs: chr "+proj=tmerc +lat_0=0 +lon_0=-90 +k=0.9996 +x_0=520000 +y_0=-4480000 +ellps
```

Extract Properties

Knowing the structure makes extracting properties much easier.

```
mendota@proj4string
```

```
## CRS arguments:
##  +proj=tmerc +lat_0=0 +lon_0=-90 +k=0.9996 +x_0=520000
## +y_0=-4480000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m
## +no_defs
```

```
mendota@polygons[[1]]@area
```

```
## [1] 39582017
```

```
mendota@polygons[[1]]@Polygons[[1]]@area
```

```
## [1] 39582017
```

Use slot function

Can also use the `slot` function. Usage: `slot(object, name)`

or `slotNames(x)`

```
slotNames(lakes)
```

```
## [1] "data"        "polygons"    "plotOrder"   "bbox"        "proj4string"
```

```
slotNames(lakes@polygons[[1]]) # Can use within the lists as well
```

```
## [1] "Polygons"  "plotOrder" "labpt"     "ID"        "area"
```

```
slot(lakes,'proj4string')
```

```
## CRS arguments:
##  +proj=tmerc +lat_0=0 +lon_0=-90 +k=0.9996 +x_0=520000
## +y_0=-4480000 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m
## +no_defs
```

Can nest within an `apply` function to extra all the properties

```
sapply(slot(lakes, "polygons"), slot, "ID")
```

```
## [1] "0" "1" "2" "3"
```

```
sapply(slot(lakes, "polygons"), slot, "area")
```

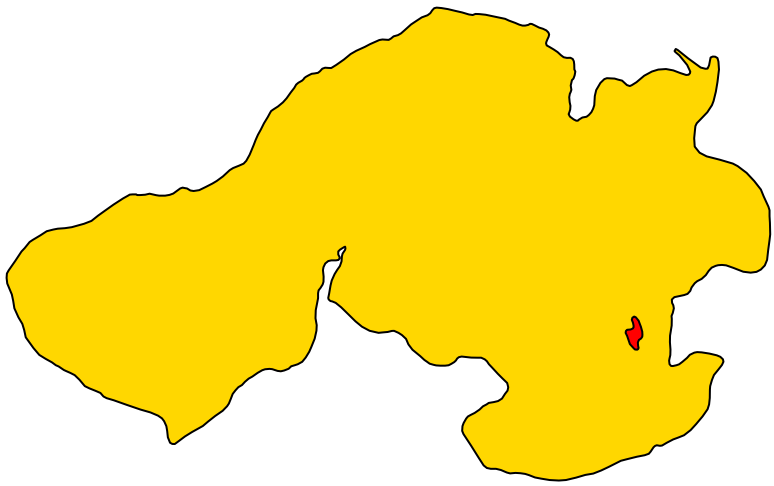
```
## [1]   803713.3 39582017.4 13715117.4  1383731.1
```

Working with the bathymetry data

There are 88 polygons `nrow(mendota.bathy)`. Each depth interval has multiple polygons.

What do the polygons look like?

```
plot(mendota.bathy[10,],col='gold')
plot(mendota.bathy[25,],col='red',add=T)
```



Looking at `mendota.bathy@data`, there are only three columns **ID**, **DEPTH_FT**, and **DEPTH_M**

Would be handy to have the area of each polygon in the table. Can use our `sapply` function to extract the areas and add them to the data.frame

```
mendota.bathy@data$AREA_m2 = sapply(slot(mendota.bathy, "polygons"), slot, "area")
head(mendota.bathy@data)
```

```
##   ID Depth_ft Depth_m  AREA_m2
## 0  0        0  0.0000 38798532
## 1  3        3  0.9144 37516636
## 2  5        5  1.5240 35023715
## 3 10       10  3.0480 31717657
## 4 15       15  4.5720 29133013
## 5 20       20  6.0960 27573949
```

We can work with this data.frame the same we would with any data.frame

```
library(dplyr)
polygon.areas = mendota.bathy@data %>% group_by(ID) %>%
  summarise_at('AREA_m2',sum) %>%
  arrange(ID)
head(polygon.areas)
```

```
## # A tibble: 6 x 2
##      ID  AREA_m2
##   <int>    <dbl>
## 1     0 38798532
## 2     3 37590567
## 3     5 35033054
## 4    10 31759311
## 5    15 29172795
## 6    20 27630501
```

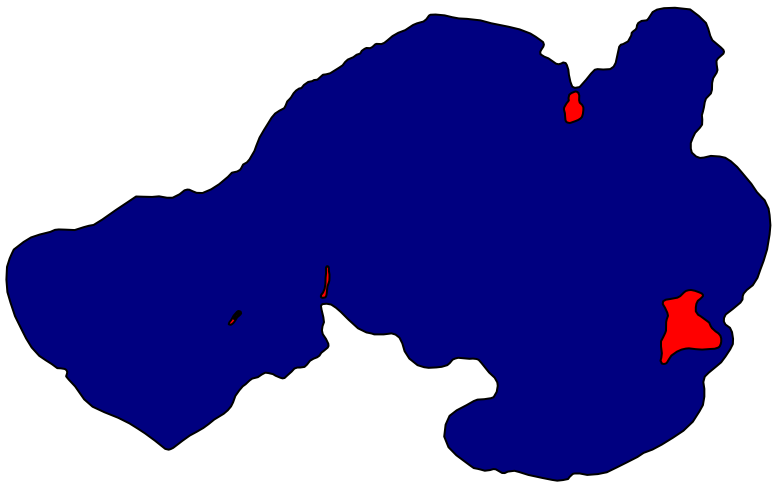
But wait.

Take a look at one individual depth. The polygons overlap. So we're overestimating areas.

```
thirty = mendota.bathy[mendota.bathy@data$ID == '30',]
sapply(slot(thirty, "polygons"), slot, "area") # Get areas
```

```
## [1] 25536759.888   293236.084    12402.829     6984.356     1618.697
## [6]    49667.127
```

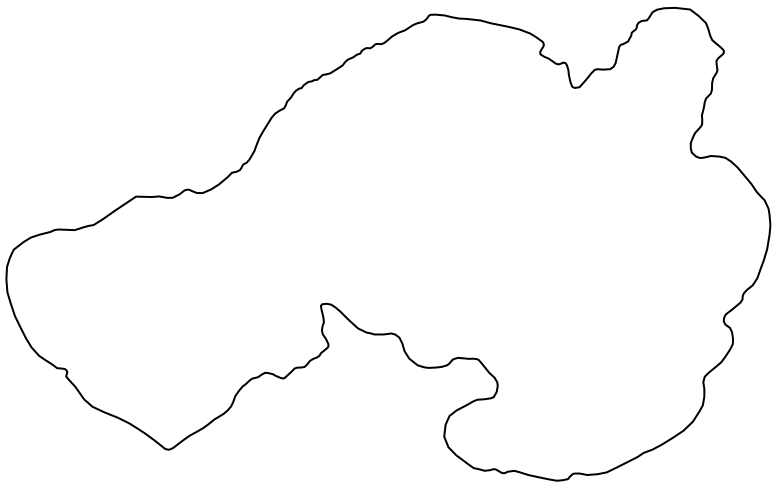
```
plot(thirty[1,],col='navy')
plot(thirty[2:6,],col='red',add=T)
```



We can create a **union** of these polygons by joining intersecting geometries

```
library(rgeos)
union.thirty = gUnaryUnion(thirty)
```

```
plot(union.thirty)
```



```
sum(sapply(slot(union.thirty, "polygons"), slot, "area")) # new area
```

```
## [1] 25536760
```

```
sum(sapply(slot(thirty, "polygons"), slot, "area")) # compare to old area
```

```
## [1] 25900669
```

Now it would be nice if we could `group_by` to create a summarised data.frame with updated areas. But we can't, because dplyr won't read `sp` objects. Would have to do some crazy apply function, or a loop.

Welcome to sf

https://cran.r-project.org/web/packages/sf/vignettes/sf1.html

Simple features refer 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.

The standard is widely implemented in spatial databases (such as PostGIS), commercial GIS (e.g., ESRI ArcGIS) and forms the vector data basis for libraries such as GDAL. A subset of simple features forms the GeoJSON standard.

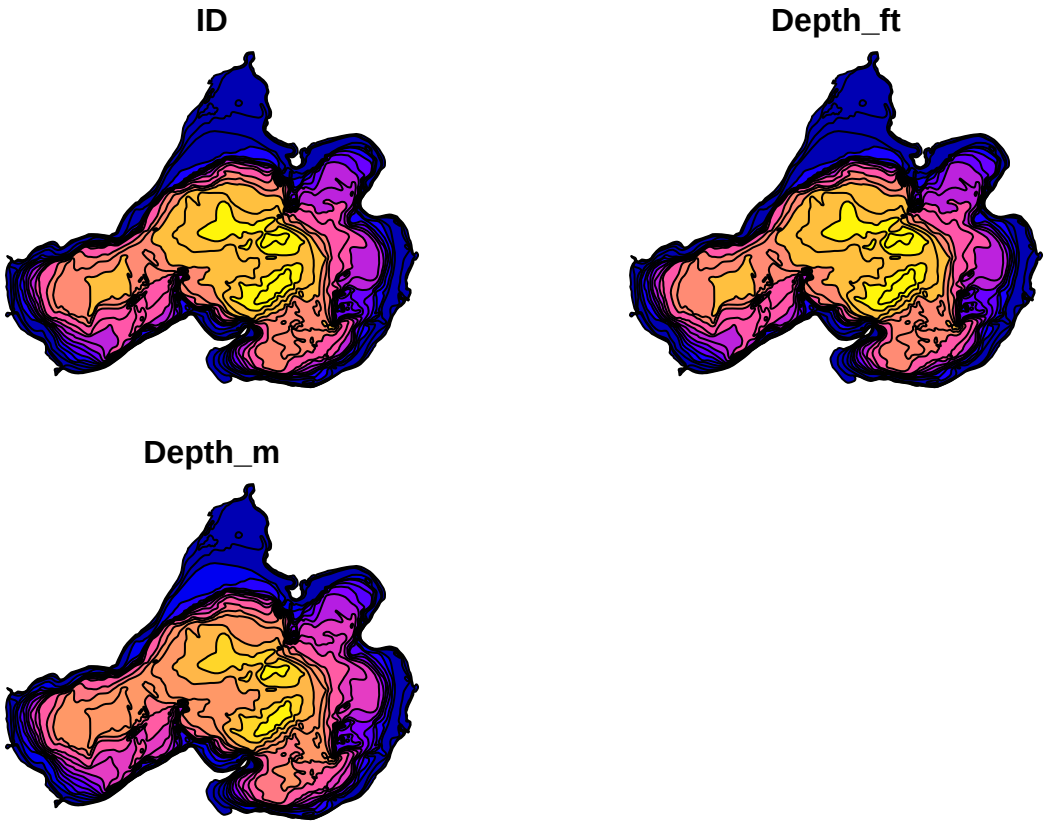
R has well-supported classes for storing spatial data (sp) and interfacing to the above mentioned environments (rgdal, rgeos), but has so far lacked a complete implementation of simple features, making conversions at

times convoluted, inefficient or incomplete. The package sf tries to fill this gap, and **aims at succeeding sp in the long term**.

```
library(sf)
sf.bathy <- st_read('Data/Bathymetry/mendota-contours-all.shp')
```

```
## Reading layer `mendota-contours-all' from data source `C:\Users\hdugan\Documents\Rpackages\Zoo955\Le
## Simple feature collection with 89 features and 3 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 562047 ymin: 289630.4 xmax: 571470.6 ymax: 297292.4
## epsg (SRID):    NA
## proj4string:    NA
```

```
plot(sf.bathy)
```




```
lakes = st_read('Data/yld_study_lakes.shp')
```

```
## Reading layer `yld_study_lakes' from data source `C:\Users\hdugan\Documents\Rpackages\Zoo955\Lecture3
## Simple feature collection with 4 features and 9 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 547589.4 ymin: 286020.8 xmax: 574950 ymax: 313254
## epsg (SRID):    NA
## proj4string:    +proj=tmerc +lat_0=0 +lon_0=-90 +k=0.9996 +x_0=520000 +y_0=-4480000 +ellps=GRS80 +tou
```

```
mendota = lakes %>% filter(LAKEID == 'ME')
```

You *can* convert `sf` objections to `sp` objects: `as(mendota, "Spatial")`

sf areas

Most geometries are WAY easier to calculate

```
head(st_area(sf.bathy))
```

```
## [1] 38798532 37516636 35023715 31717657 29133013 27573949
```

```
areas = sf.bathy %>% mutate(AREA.m2 = st_area(sf.bathy)) %>%
  group_by(ID) %>%
  summarise(areas = st_union(geometry))

data.frame(ID = areas$ID,area = st_area(areas),old.areas = polygon.areas)
```

```
##    ID     area old.areas.ID old.areas.AREA_m2
## 1   0 38798532            0          38798532
## 2   3 37516636            3          37590567
## 3   5 35033054            5          35033054
## 4  10 31735226           10          31759311
## 5  15 29146679           15          29172795
## 6  20 27573949           20          27630501
## 7  25 26548067           25          26743939
## 8  30 25536760           30          25900669
## 9  35 23993455           35          23997721
## 10 40 22449025           40          22501881
## 11 45 20115070           45          20155055
## 12 50 18002017           50          18006229
## 13 55 14918470           55          14943897
## 14 60 12245714           60          12348533
## 15 65  8510010           65           8518724
## 16 70  4843634           70           4843634
## 17 75  2042268           75           2042268
## 18 80   394689           80            394689
```

sf CRS

sf ojects work with EPSG codes nicely, and prefer them over proj4strings. Mostly for convience.

Coordinate reference systems can be applied using `st_crs`

Coordinate reference system transformations can be carried out using `st_transform`

```
st_crs(areas) = st_crs(mendota)
#Could also do: st_crs(areas) <- 4326 (but need to know espg code)
```

sf geometrical operations

The commands `st_intersects`, `st_disjoint`, `st_touches`, `st_crosses`, `st_within`, `st_contains`, `st_overlaps`, `st_equals`, `st_covers`, `st_covered_by`, `st_equals_exact` and `st_is_within_distance` return a sparse matrix with matching (TRUE) indexes, or a full logical matrix:

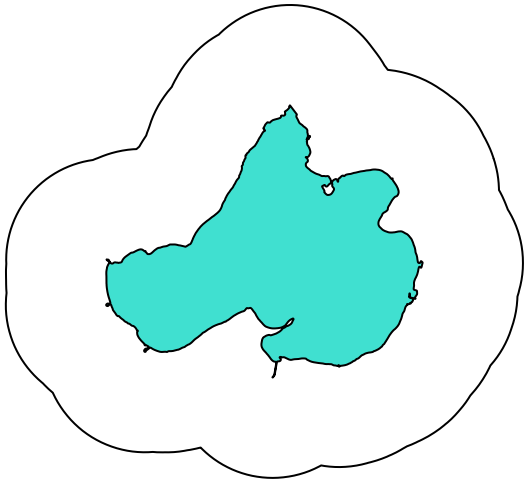
```
st_intersects(areas,mendota)
```

```
## Sparse geometry binary predicate list of length 18, where the predicate was `intersects'
## first 10 elements:
```

```
##  1: 1
##  2: 1
##  3: 1
##  4: 1
##  5: 1
##  6: 1
##  7: 1
##  8: 1
##  9: 1
##  10: 1
```

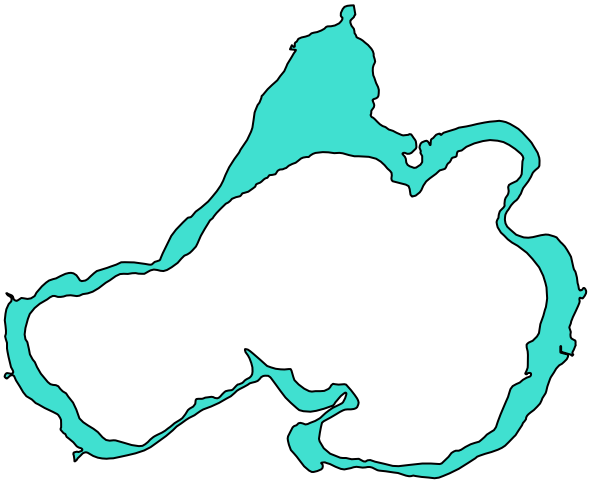
The commands `st_buffer`, `st_boundary`, `st_convexhull`, `st_union_cascaded`, `st_simplify`, `st_triangulate`, `st_polygonize`, `st_centroid`, `st_segmentize`, and `st_union` return new geometries. Example of st_buffer:

```
buffer <- st_buffer(mendota, dist = 3000)
plot(buffer$geometry)
plot(mendota$geometry,add=T,col='turquoise')
```



Commands `st_intersection`, `st_union`, `st_difference`, `st_sym_difference` return new geometries that are a function of pairs of geometries:

```
bathy.diff = st_difference(areas[1,]$geometry,areas[6,]$geometry)
plot(bathy.diff,col='turquoise')
```



Homework

1) Define 7 of these using simple language (1 sentence if possible): `st_intersects`, `st_disjoint`, `st_touches`,`st_crosses`, `st_within`, `st_contains`, `st_overlaps`, `st_equals`, `st_covers`, `st_covered_by`, `st_equals_exact` and `st_is_within_distance`,`st_buffer`, `st_boundary`, `st_convexhull`, `st_union_cascaded`, `st_simplify`, `st_triangulate`, `st_polygonize`, `st_centroid`, `st_segmentize`, and `st_union`

Preferably choose ones that you don't already know.

2) Make a 500 m buffer of the 4 southern LTER lakes. Which buffers overlap?

3) [This question is considerably more difficult. Try your best, but don't spend more than 30 minutes on this.] Increase the size of the lakes by 2x. What is the percent of Mendota that overlaps with Monona?