Lecture 16
Introduction to GIS Programming

GEOG 489 SPRING 2020

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.

Spatial objects

The core of a vector spatial object is the class "Spatial".

This class has only two slots:

 - a bounding box of class "matrix": column names "min " and "max; first row eastings (x-axis); second row northings (y-axis)

 - a coordinate reference system of class "CRS"

Spatial objects

1. Make a spatial object:

1) Make a bounding box:

```
myBoundingBox <- matrix(c(0,0,1,1),ncol=2,
dimnames=list(NULL,c("min","max")))
```

2) Make a coordinate reference system:

```
# We can leave the CRS blank. If the projection is unknown, use
as.character(NA).

myCRS <- CRS(projargs = as.character(NA))
```

Spatial objects

1. Make a spatial object:

3) Make a spatial object including a bounding box (matrix) and a coordinate system (CRS)

```
mySpatialObject <- Spatial(bbox=myBoundingBox,
    proj4string=myCRS)
```

Spatial objects

2. Spatial points

A SpatialPoints class EXTENDS the Spatial class by adding a slot called "coords", which is a matrix of point coordinates:

1) Let's make a matrix of the coordinates:

```
CRAN_mat <- cbind(CRAN_df$long, CRAN_df$lat)
```

2) This dataset uses a Geographic Projection with a WGS84 ellipsoid:

```
CRAN_CRS <- CRS("+proj=longlat +ellps=WGS84")
```

Spatial objects

2. Spatial points

A SpatialPoints class EXTENDS the Spatial class by adding a slot called "coords", which is a matrix of point coordinates:

3) Make a SpatialPoints object

CRAN_SpatialPoints <- SpatialPoints(coords=CRAN_mat, proj4string=CRAN_CRS)

Notice that we didn't need to "manually" assign the bounding box, it was calculated based on the points.

Spatial objects

3. Spatial points data frame

Spatial points are "raw points" that have no attribute data.

What if we want to now construct something more akin to a shapefile, where we have geographic information linked to a table?

(1) We use a new class called SpatialPointsDataFrame that is basically a SpatialPoints object with a linked data frame

```
CRAN_SpatialPointsDataFrame1 <-
  SpatialPointsDataFrame(coords=CRAN_mat, data=CRAN_df,
                         proj4string=CRAN_CRS, match.ID=TRUE)
```

when match.ID=TRUE, rownames are used from both the coordinates matrix and the data frame to link the two together.

Spatial objects

3. Spatial points data frame

(2) We could use a SpatialPoints object as the coordinates, rather than a matrix:

```
CRAN_SpatialPointsDataFrame4 <- SpatialPointsDataFrame(coords=
                                        CRAN_SpatialPoints, data=CRAN_df)
```

(3) We could also directly assign points to a data frame

```
CRAN_df_new <- CRAN_df
coordinates(CRAN_df_new) <- CRAN_mat
proj4string(CRAN_df_new) <- CRAN_CRS
```

Summary of SpatialPoints

Spatial: bounding box and coordinate reference system

SpatialPoints: Spatial + coordinates

SpatialPointsDataFrame: SpatialPoints + data frame

SpatialLines objects

- A SINGLE line (can have multiple arcs)

```
getClass("Line")
```

- Multiple Line objects arranged in a list with a single ID

```
getClass("Lines")
```

- a Spatial object + a list of Lines objects

```
getClass("SpatialLines")
```

SpatialLines objects

Import a map as a SpatialLines object

SpatialLines object of Japan

japan <- map("world","japan",plot=FALSE)

p4s <- CRS("+proj=longlat +ellps=WGS84")

SLjapan <- map2SpatialLines(japan,proj4string=p4s)

plot(SLjapan)

```
> str(SLjapan,max.level=2)
Formal class 'SpatialLines' [package "sp"] with 3 slots
  ..@ lines      :List of 34
  ..@ bbox       : num [1:2, 1:2] 123.7 24.3 145.8 45.5
  .. ..- attr(*, "dimnames")=List of 2
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
```

SpatialLines objects

Import a map as a SpatialLines object

```
# The lines slot:
SLjapan_lines <- slot(SLjapan,"lines") # same as SLjapan@lines
class(SLjapan_lines) # “list” (actually a list of Lines)
class(SLjapan_lines[[1]])  #”Lines”
```

A Lines can, in theory, have multiple single lines.

```
# Check to see how many individual Line objects are in each Lines
Lines_len <- sapply(slot(SLjapan,"lines"),
                    function(x) length(slot(x,"Lines")))
table(Lines_len)
```

SpatialLinesDataFrame objects

Each Lines object can be linked to an attribute

Volcano is a built-in dataset

volcano_sl <- ContourLines2SLDF(contourLines(volcano))

plot(volcano_sl)

Ten Lines objects

```
> t(volcano_sl@data)
      C_1   C_2   C_3   C_4   C_5   C_6   C_7   C_8   C_9   C_10
level "100" "110" "120" "130" "140" "150" "160" "170" "180" "190"
```

Summary of number of lines within Lines objects

Lines_len <- sapply(slot(volcano_sl,"lines"),

function(x) length(slot(x,"Lines")))

```
> table(Lines_len)
Lines_len
1 2 3 4
4 3 2 1
```

Summary of SpatialLines

- **Spatial:** bounding box and coordinate reference system
- **Line:** a single connected set of arcs, defined by the coordinates of their nodes.
- **Lines:** a list of Line objects that share a SINGLE ID. In ESRI terminology, this represents a "multipart line". This is the level that is linked to a data frame.
- **SpatialLines:** Spatial + a list of Lines
- **SpatialLinesDataFrame:** SpatialLines + data frame linked to the Lines IDs

SpatialPolygons objects

- A polygon is essentially just a line except the last coordinate must be the same as the first coordinate.
- To build polygons

getClass("Polygon")

This class extends a basic Line in the following ways:

1) The Line is confirmed to have the first and last coordinates be equal

2) A label point, defined at the centroid of the polygon

3) The area of the polygon in the units of the coordinates

4) Whether the polygon is a hole (NA by default)

5) Ring direction of the polygon. These last two deal with topological issues.

SpatialPolygons objects

getClass("Polygons")

This is a list of individual Polygon objects, and from a GIS standpoint, is the unit of attribution. The slots are as follows:

1) A list of Polygon objects.

2) The order the polygons should be plotted (if > 1 Polygon)

3) A label point (the centroid of the largest Polygon)

4) An ID (to be linked to the attribute table)

5) The total area of all individual Polygon objects.

getClass("SpatialPolygons")

Fuses a standard Spatial object with a list of Polygons

SpatialPolygons objects

auck_shore example (SpatialLines object)

lns <- slot(auck_shore, "lines") # Pull the list of Lines out

```
> table(sapply(lns,function(x) length(slot(x,"Lines"))))

 1
80
```

This tells us that the lines slot in auck_shore has 80 Lines objects, each of which contain a single Line.

SpatialPolygons objects

auck_shore example (SpatialLines object)

We can check for islands as Lines in which the first coordinate of the Line matches the last coordinate:

```
islands_auck <- sapply(lns,function(x){
        # Pull out the Line coordinates from the first Lines entry:
        crds <- slot(slot(x,"Lines")[[1]],"coords")
        # Are the first coordinates equal to the last coordinates?
        identical(crds[1,],crds[nrow(crds),]) })
table(islands_auck) # 64 of the lines in auck_shore are islands
```

SpatialPolygons objects

auck_shore example (SpatialLines object)

Pull out the island subset of the Auckland Shoreline

islands_sl <- auck_shore[islands_auck]

plot(islands_sl)

Notice these are still SpatialLines:

class(islands_sl)

We'll pull out just the list of Lines from this:

list_of_Lines <- islands_sl@lines

list_of_Lines[[1]]

SpatialPolygons objects

auck_shore example (SpatialLines object)

```
> list_of_Lines[[1]]
An object of class "Lines"
Slot "Lines":
[[1]]
An object of class "Line"
Slot "coords":
          [,1]      [,2]
 [1,] 174.6996 -37.43261
 [2,] 174.6996 -37.43173
 [3,] 174.7005 -37.43085
 [4,] 174.7005 -37.42997
 [5,] 174.7008 -37.42967
 [6,] 174.7016 -37.42967
 [7,] 174.7022 -37.42997
 [8,] 174.7022 -37.43261
 [9,] 174.7016 -37.43290
[10,] 174.6999 -37.43290
[11,] 174.6996 -37.43261



Slot "ID":
[1] "L_4"
```

SpatialPolygons objects

Convert SpatialLines objects to SpatialPolygons objects

```
# convert each individual Line to a Polygon, then to a Polygons, then to a list of Polygons:
list_of_Polygons <- lapply(list_of_Lines,function(x){
        # Convert the first (and only) Line in the Lines to a Polygon:
        single_Line <- x@Lines[[1]]@coords
        single_Line_ID <- x@ID
        single_Polygon <- Polygon(coords=single_Line)
        single_Polygons <- Polygons(list(single_Polygon),ID=single_Line_ID)})
islands_sp <-  SpatialPolygons(list_of_Polygons,
                proj4string=CRS("+proj=longlat +ellps=WGS84"))
summary(islands_sp)
plot(islands_sp) # Looks the same as the SpatialLines version...
```

SpatialPolygons objects

Convert SpatialLines objects to SpatialPolygons objects

For example:

single_Line <- list_of_Lines[[1]]@Lines[[1]]@cords

```
> list_of_Lines[[1]]@Lines[[1]]@coords
          [,1]      [,2]
 [1,] 174.6996 -37.43261
 [2,] 174.6996 -37.43173
 [3,] 174.7005 -37.43085
 [4,] 174.7005 -37.42997
 [5,] 174.7008 -37.42967
 [6,] 174.7016 -37.42967
 [7,] 174.7022 -37.42997
 [8,] 174.7022 -37.43261
 [9,] 174.7016 -37.43290
[10,] 174.6999 -37.43290
[11,] 174.6996 -37.43261
```

single_Line_ID <- list_of_Lines[[1]]@ID

SpatialPolygonsDataFrame objects

Just like SpatialPoints and SpatialLines, we can add a data frame that is linked up against an individual Polygons' ID.

```
state.map <- map("state",plot=TRUE,fill=FALSE)
```

SpatialPolygonsDataFrame objects

Define IDs based on the state name

state.map$names # Notice some of these have subregions

```
$names
 [1] "alabama"                  "arizona"
 [3] "arkansas"                 "california"
 [5] "colorado"                 "connecticut"
 [7] "delaware"                 "district of columbia"
 [9] "florida"                  "georgia"
[11] "idaho"                    "illinois"
[13] "indiana"                  "iowa"
[15] "kansas"                   "kentucky"
[17] "louisiana"                "maine"
[19] "maryland"                 "massachusetts:martha's vineyard"
[21] "massachusetts:main"       "massachusetts:nantucket"
[23] "michigan:north"           "michigan:south"
[25] "minnesota"                "mississippi"
[27] "missouri"                 "montana"
[29] "nebraska"                 "nevada"
[31] "new hampshire"            "new jersey"
[33] "new mexico"               "new york:manhattan"
[35] "new york:main"            "new york:staten island"
[37] "new york:long island"     "north carolina:knotts"
[39] "north carolina:main"      "north carolina:spit"
```

IDs <- sapply(strsplit(state.map$names,":"),function(x) x[1])

IDs

SpatialPolygonsDataFrame objects

Import a map as a SpatialPolygons object

state.sp <- map2SpatialPolygons(state.map, IDs=IDs,

proj4string=CRS("+proj=longlat +ellps=WGS84"))

Link the mean SAT scores from 1999 to this:

sat <- read.table("state.sat.data_mod.txt",row.names=5,header=TRUE)

state.spdf <- SpatialPolygonsDataFrame(state.sp,sat_good_ids)

```
> state.spdf@data
                     oname vscore mscore pc
alabama                ala    561    555  9
arizona               ariz    524    525 34
arkansas               ark    563    556  6
california           calif    497    514 49
colorado              colo    536    540 32
connecticut           conn    510    509 80
delaware              dela    503    497 67
```

Summary of SpatialPolygons

- **Spatial:** bounding box and coordinate reference system
- **Polygon:** a single connected set of arcs, defined by the coordinates of their nodes, having the first and last nodes being identical. Also, slots for whether the polygon is a hole and what the ring direction is (clockwise or anti-clockwise).
- **Polygons:** a list of Polygon objects that share a SINGLE ID. In ESRI terminology, this represents a "multipart polygon". This is the level that is linked to a data frame.
- **SpatialPolygons:** Spatial + a list of Polygons
- **SpatialPolygonsDataFrame:** SpatialPolygons + data frame linked to the Polygons IDs

Assignment 4

Your goal is to:

1) Read a CSV file from the working directory given a filename

2) Create a multi-page PDF file that contains plots of all combinations of the input data's columns given specific formatting requirements, as well as plotting a line generated from a linear regression through the points.

3) Write a CSV file (with a header) LINE BY LINE, one line for each plot, containing the x and y column names and the correlation between those two columns.

The assignment is due on Tuesday, March 24, 2020 at midnight.

Assignment 4

Requirements:

1) The function should be named "plotTableFromDisk" and have the following parameters:

dataFile : the filename of the input CSV file

outpdf : the filename used for the output PDF, and should default to "mypdf.pdf"

outcor : the filename used for the output CSV, and should default to "mycor.csv"

2) All combinations of the input columns should be plotted and have their correlations calculated/stored, but you don't have to do symmetrical plotting, e.g. for two columns A and B, plot only A vs. B, not B vs. A.

Hint: ?combn

Assignment 4

Requirements:

3) Each plot should be formatted as follows:

- The axes should range between the minimum to the maximum of ALL VALUES in the input dataFile -- e.g. the ranges of x and y will be the same, and the axes will be constant across all plots.

- The x- and y-axis labels should be the column names.

- The title of the plot should be "[x-column name] vs. [y-column name]"

- The points should be "triangle point-up" (see ?points), have a red outline, and be 1.5 times the normal size.

- The line should be derived from a linear regression (?lm), and should be blue.

4) The output CSV should have a header which, in a text editor, would look like: xcol,ycol,correlation

Assignment 4

Requirements:

5) The CSV must be written line-by-line, not all at once.

6) The PDF and the CSV file must be properly closed.

7) The function should return a NULL.

8) Comment your code in at least 3 places.

9) The code should be submitted to Compass 2g as a single function with the filename:

LastName-FirstName-geog489-s20-assignment-04.R

and should have at the top:

[Your name]

Assignment #4

Package Introduction

Please see the package introduction guideline PPT for more info

Description

Each student will be expected to introduce a package (or two) that is relevant to their research interests through discussion forum on Compass.

The objectives are:

- Learn how to find/download/install a new package and learn how to use it
- Teach your peers about existing R packages that may be useful in their research

Package Introduction

The package introduction PPT must include:

1. Brief introduction: what does the package do and why is it useful? (1-2 slides)

2. Author introduction: a short background (affiliation and other packages, etc.) on at least one of the package authors (1 slide)

3. Simple demonstration of package code: example input/output from the examples or custom coded examples (4-5 slides, 2.5 minutes)

Please submit your PPT on Compass by March 31, 2020