Spatial Data Visualization

Code

Geospatial Data Visualization using R

Data Analysis and Visualization (Spring 2019)

Instructor: Debopriya Ghosh

Spatial Data in R

Loading the required libraries

Hide

```
library(sp)
library(rgdal)
```

```
rgdal: version: 1.4-3, (SVN revision 828)
 Geospatial Data Abstraction Library extensions to R successfully loaded
 Loaded GDAL runtime: GDAL 2.1.3, released 2017/20/01
 Path to GDAL shared files: /Library/Frameworks/R.framework/Versions/3.5/Resources/library/rgdal/gdal
 GDAL binary built with GEOS: FALSE
 Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
 Path to PROJ.4 shared files: /Library/Frameworks/R.framework/Versions/3.5/Resources/library/rgdal/proj
 Linking to sp version: 1.3-1
```

Hide

```
library(raster)
library(rgeos)
```

```
rgeos version: 0.4-2, (SVN revision 581)
 GEOS runtime version: 3.6.1-CAPI-1.10.1
 Linking to sp version: 1.3-1
 Polygon checking: TRUE
```

Hide

```
library(plyr)
library(maptools)
```

```
Checking rgeos availability: TRUE
```

Hide

```
library(RColorBrewer)
library(ggmap)
```

```
Loading required package: ggplot2
Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
Please cite ggmap if you use it! See citation("ggmap") for details.
```

Hide

```
library(gridExtra)
library(classInt)
library(geosphere)
library(spdep)
```

```
Loading required package: Matrix
Loading required package: spData
To access larger datasets in this package, install the spDataLarge
package with: `install.packages('spDataLarge',
repos='https://nowosad.github.io/drat/', type='source')`
Loading required package: sf
Linking to GEOS 3.6.1, GDAL 2.1.3, PROJ 4.9.3
```

Hide

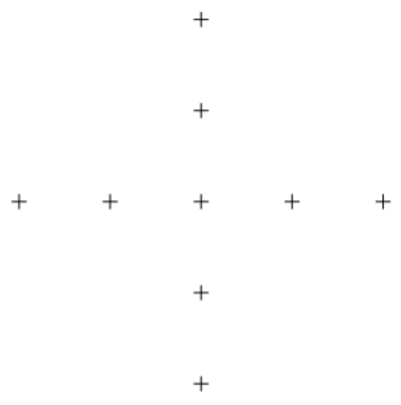
```
library(gmapsdistance)
```

How R sees Spatial Data?

Just like basic data points, spatial data points are specified by a pair of x-y coordinates.

Hide

```
coordinates <- rbind(c(1.5, 1.25), c(1.75, 1.25),
                          c(2.25, 1.25), c(2.5, 1.25), #horizontal
                         c(2, 1.75), c(2, 1.5),
                         c(2, 1), c(2, 0.75), #vertical
                         c(2, 1.25)) #center
#points converted into spatial object
spatialpoints <- SpatialPoints(coordinates)
plot(spatialpoints)
```



To understand how R sees these points, we can ask use the summary() command. Output show the following:(i) class of the object; (ii) minimum and maximum coordinates; (iii) whether the object is projected or not; (iv) if projected, which coordinate reference system it uses (CRS); (v) total number of points in the data;

Hide

```
summary(spatialpoints)
```

```
Object of class SpatialPoints
Coordinates:
           min  max
coords.x1 1.50 2.50
coords.x2 0.75 1.75
Is projected: NA
proj4string : [NA]
Number of points: 9
```

To reach all the coordinate values use coordinates() command.

Hide

```
coordinates(spatialpoints)
```

```
      coords.x1 coords.x2
 [1,]      1.50      1.25
 [2,]      1.75      1.25
 [3,]      2.25      1.25
 [4,]      2.50      1.25
 [5,]      2.00      1.75
 [6,]      2.00      1.50
 [7,]      2.00      1.00
 [8,]      2.00      0.75
 [9,]      2.00      1.25
```

Adding Coordinate Reference System

What we have created is a simple spatial object. We need to convert this SpatialPoints object to geospatial through a Coordinate Reference System (CRS) such that the coordinates of the points relate to places in real world. CRS combines information on the geographic coordinate system and the projection. The CRS of an object can be found through proj4string() or the summary of the object. NA value means that SpatialPoints does not have a CRS defined yet. We can also use is.projected() command, which will tell you whether a CRS is defined or not.

Hide

```
is.projected(spatialpoints)
```

```
[1] NA
```

We need to define a CRS so that the spatial object becomes a geospatial object. The CRS can be defined by simply passing the reference code for the projection. We'll use the most common CRS, EPSG:4326, although there are many other possible reference systems: CRS("+init=EPSG:4326").

Hide

```
CRS("+init=EPSG:4326")
```

```
CRS arguments:
 +init=EPSG:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
+towgs84=0,0,0
```

Hide

```
crs.geo <- CRS("+init=EPSG:4326")
#assign a projection to our data
proj4string(spatialpoints) <- crs.geo
#check it is defined now
is.projected(spatialpoints)
```

```
[1] FALSE
```

This SpatialPoints object doesn't contain any variables. We can move from a SpatialPoints to a SpatialPointsDataFrame by adding a data.frame of variables to the points. Here, we will add some randomly generated data. The SpatialPoints will merge with the data.frame based on the order of observations, and therefore the number of spatial points should be equal to the number of rows in the new data.

Hide

```
vars <- data.frame(vars1 = rnorm(9), vars2 = c(101:109))
vars
```

```
        vars1 vars2
1 -0.07816939   101
2 -0.47924608   102
3 -0.24970657   103
4  0.49653740   104
5 -0.76928307   105
6  0.41586802   106
7 -0.23480981   107
8  0.63670068   108
9  1.13371190   109
```

Hide

```
spdf <- SpatialPointsDataFrame(spatialpoints, vars)
summary(spdf)
```

```
Object of class SpatialPointsDataFrame
Coordinates:
           min  max
coords.x1 1.50 2.50
coords.x2 0.75 1.75
Is projected: FALSE
proj4string :
[+init=EPSG:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
+towgs84=0,0,0]
Number of points: 9
Data attributes:
     vars1              vars2
```

```
 Min.   :-0.76928   Min.   :101  
 1st Qu.:-0.24971   1st Qu.:103  
 Median :-0.07817   Median :105  
 Mean   : 0.09684   Mean   :105  
 3rd Qu.: 0.49654   3rd Qu.:107  
 Max.   : 1.13371   Max.   :109  
```

Suppose you don't have a SpatialPoints object but only a data.frame with two columns indicating longitude and latitude values. SpatialPointsDataFrame object can also be created directly from this data frame. Use coordinates() command to specify which columns contain the coordinates.

Spatial Polygons

SpatialPolygons object consists of polygons. To obtain a SpatialPolygons object, we need three steps: (i) First create Polygon objects, two-dimensional geometrical shapes, from points. (ii) Then combine those into Polygons objects (Need to give each polygon a unique ID). (iii) Finally we will combine Polygons into SpatialPolygons.

Hide

```
#create polygon objects from points.
triangle1 <- Polygon(rbind(c(1, 1), c(2, 1), c(1.5, 0)))
```

```
less than 4 coordinates in polygon
```

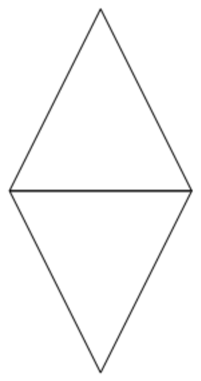
Hide

```
triangle2 <- Polygon(rbind(c(1, 1), c(1.5, 2), c(2, 1)))
```

```
less than 4 coordinates in polygon
```

Hide

```
#create polygons objects from polygon
t1 <- Polygons(list(triangle1), "triangle1")
t2 <- Polygons(list(triangle2), "triangle2")
#create spatial polygons object from polygons, equal to shapefiles.
spatialpolygons <- SpatialPolygons(list(t1, t2))
plot(spatialpolygons)
```



As with SpatialPoints, we can add variables to SpatialPolygons. Note that the rownames of the data.frame that includes our variables should match the unique IDs of the polygon objects. Ideally the number of our spatial polygons should be equal to the number of rows in the new data so that we don't have any missing values in our SpatialPolygonsDataFrame.

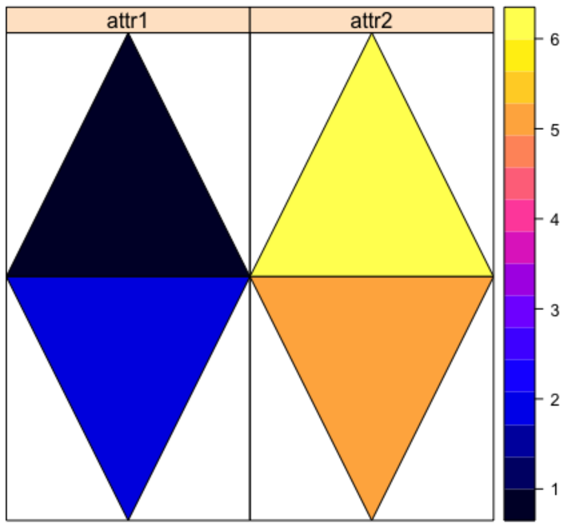
Hide

```
vars <- data.frame(attr1 = 1:2, attr2 = 6:5,
                   row.names = c("triangle2", "triangle1"))
spoldf <- SpatialPolygonsDataFrame(spatialpolygons, vars)
as.data.frame(spoldf)
```

```
          attr1 attr2
triangle1     2     5
triangle2     1     6
```

Hide

```
spplot(spoldf)
```



In order the SpatialPoints object to understand how the coordinates of its points relate to places in the real world, we assigned to it a Coordinate Reference System (CRS). We'll do the same for this SpatialPolygons, using exactly the same method.

Hide

```
crs.geo <- CRS("+init=EPSG:4326") #geographical, datum WGS84
proj4string(spoldf) <- crs.geo
```

Importing and Exporting Spatial Data

Using existing spatial data by reading them from external sources: (a) Shapefiles In order to read shapefiles, which will usually be in form of an ESRI shapefile, use readOGR() and writeOGR() commands. readOGR() needs at least the following two arguments: dsn =, path to the folder that contains the files, and layer =, name of the shapefile (without the extension .shp).

Hide

```
boston <- readOGR(dsn = "~/Dropbox/Priya-PhD- Documents/Courses/Data Analysis and Visualization-Spring 2019/Datasets/BostonD
ata",
                  layer = "city_council_districts")
```

```
OGR data source with driver: ESRI Shapefile
Source: "/Users/priya/Dropbox/Priya-PhD- Documents/Courses/Data Analysis and Visualization-Spring 2019/Datasets/BostonData",
layer: "city_council_districts"
with 9 features
It has 5 fields
Integer64 fields read as strings:  OBJECTID
```

To inspect the shapefile, use the str command to check what it includes. It will show that boston is actually a combination of data, coords, bbox, and a proj4string. We can call each single attribute either by the corresponding command or by using the @ command.

Hide

```
str(boston)
```

```
Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
  ..@ data       :'data.frame': 9 obs. of  5 variables:
  .. ..$ OBJECTID  : Factor w/ 9 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9
  .. ..$ DISTRICT  : int [1:9] 1 2 3 4 5 6 8 9 7
  .. ..$ Councillor: Factor w/ 9 levels "Councillor Baker",..: 5 6 1 9 3 7 8 2 4
  .. ..$ SHAPE_area: num [1:9] 1.88e+08 1.33e+08 1.34e+08 1.02e+08 2.18e+08 ...
  .. ..$ SHAPE_len : num [1:9] 220959 112447 121915 68001 90278 ...
  ..@ polygons   :List of 9
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. .. .. ..@ Polygons :List of 4
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 787005 2961184
  .. .. .. .. .. .. ..@ area   : num 1.32e+08
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:703, 1:2] 779358 779250 779195 779109 779168 ...
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 774054 2963433
  .. .. .. .. .. .. ..@ area   : num 36672591
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:242, 1:2] 769487 769465 769437 769410 769679 ...
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 776159 2956817
  .. .. .. .. .. .. ..@ area   : num 1.9e+07
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:231, 1:2] 774034 774213 774903 775481 775346 ...
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 772537 2968368
  .. .. .. .. .. .. ..@ area   : num 8e+05
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:16, 1:2] 771947 772190 772325 772620 772571 ...
  .. .. .. ..@ plotOrder: int [1:4] 1 2 3 4
  .. .. .. ..@ labpt    : num [1:2] 787005 2961184
  .. .. .. ..@ ID       : chr "0"
  .. .. .. ..@ area     : num 1.88e+08
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. .. .. ..@ Polygons :List of 1
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 777534 2948923
  .. .. .. .. .. .. ..@ area   : num 1.33e+08
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:352, 1:2] 769882 769771 769640 769453 769074 ...
  .. .. .. ..@ plotOrder: int 1
  .. .. .. ..@ labpt    : num [1:2] 777534 2948923
  .. .. .. ..@ ID       : chr "1"
  .. .. .. ..@ area     : num 1.33e+08
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. .. .. ..@ Polygons :List of 5
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 776282 2933061
  .. .. .. .. .. .. ..@ area   : num 1.26e+08
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:2126, 1:2] 776002 776002 776002 775996 775996 ...
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 788729 2940353
  .. .. .. .. .. .. ..@ area   : num 7284492
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:160, 1:2] 786624 786627 786637 786650 786660 ...
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 775999 2925609
  .. .. .. .. .. .. ..@ area   : num 20.8
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:6, 1:2] 775996 776002 776002 775999 775996 ...
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 776096 2925634
  .. .. .. .. .. .. ..@ area   : num 10.7
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:5, 1:2] 776094 776098 776098 776094 776094 ...
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 779215 2926254
  .. .. .. .. .. .. ..@ area   : num 1.85
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:4, 1:2] 779214 779218 779215 779214 2926255 ...
  .. .. .. ..@ plotOrder: int [1:5] 1 2 3 4 5
  .. .. .. ..@ labpt    : num [1:2] 776282 2933061
  .. .. .. ..@ ID       : chr "2"
  .. .. .. ..@ area     : num 1.34e+08
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. .. .. ..@ Polygons :List of 1
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 769012 2930656
  .. .. .. .. .. .. ..@ area   : num 1.02e+08
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:200, 1:2] 765347 765692 765784 765873 765957 ...
  .. .. .. ..@ plotOrder: int 1
  .. .. .. ..@ labpt    : num [1:2] 769012 2930656
  .. .. .. ..@ ID       : chr "3"
  .. .. .. ..@ area     : num 1.02e+08
```

```
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. .. .. ..@ Polygons :List of 1
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 757771 2922005
  .. .. .. .. .. .. ..@ area   : num 2.18e+08
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:234, 1:2] 751963 751807 750306 750908 751018 ...
  .. .. .. ..@ plotOrder: int 1
  .. .. .. ..@ labpt    : num [1:2] 757771 2922005
  .. .. .. ..@ ID       : chr "4"
  .. .. .. ..@ area     : num 2.18e+08
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. .. .. ..@ Polygons :List of 1
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 753001 2931516
  .. .. .. .. .. .. ..@ area   : num 2.56e+08
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:339, 1:2] 741364 740436 739594 743042 743607 ...
  .. .. .. ..@ plotOrder: int 1
  .. .. .. ..@ labpt    : num [1:2] 753001 2931516
  .. .. .. ..@ ID       : chr "5"
  .. .. .. ..@ area     : num 2.56e+08
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. .. .. ..@ Polygons :List of 2
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 766376 2951219
  .. .. .. .. .. .. ..@ area   : num 70939332
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:402, 1:2] 760730 760735 761171 761410 761337 ...
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 772654 2959644
  .. .. .. .. .. .. ..@ area   : num 100855
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:15, 1:2] 772397 772994 772859 772748 772698 ...
  .. .. .. ..@ plotOrder: int [1:2] 1 2
  .. .. .. ..@ labpt    : num [1:2] 766376 2951219
  .. .. .. ..@ ID       : chr "6"
  .. .. .. ..@ area     : num 7.1e+07
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. .. .. ..@ Polygons :List of 1
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 751817 2953458
  .. .. .. .. .. .. ..@ area   : num 1.22e+08
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:252, 1:2] 744045 744286 744619 745049 745223 ...
  .. .. .. ..@ plotOrder: int 1
  .. .. .. ..@ labpt    : num [1:2] 751817 2953458
  .. .. .. ..@ ID       : chr "7"
  .. .. .. ..@ area     : num 1.22e+08
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. .. .. ..@ Polygons :List of 1
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] 768326 2942187
  .. .. .. .. .. .. ..@ area   : num 1.14e+08
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:217, 1:2] 764464 763897 763820 763649 763882 ...
  .. .. .. ..@ plotOrder: int 1
  .. .. .. ..@ labpt    : num [1:2] 768326 2942187
  .. .. .. ..@ ID       : chr "8"
  .. .. .. ..@ area     : num 1.14e+08
  ..@ plotOrder  : int [1:9] 6 5 1 3 2 8 9 4 7
  ..@ bbox       : num [1:2, 1:2] 739594 2908187 795022 2970073
  .. ..- 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=lcc +lat_1=41.7166666666667 +lat_2=42.68333333333333 +lat_0=41 +lon_0=-71.5 +x_0=199999.99
99999999 +y_0="| __truncated__
```

Hide

```
bbox(boston)
```

```
        min       max
```

```
x  739594.1  795021.6
y 2908187.3 2970072.8
```

Hide

```
coordinates(boston)[1:5, ]
```

```
      [,1]    [,2]
0 787005.1 2961184
1 777534.0 2948923
2 776281.9 2933061
3 769011.6 2930656
4 757770.8 2922005
```

Hide

```
as.data.frame(boston)[1:5, ]
```

```
  OBJECTID DISTRICT           Councillor SHAPE_area SHAPE_len
0        1        1 Councillor LaMattina  188299817 220959.06
1        2        2   Councillor Linehan  132852622 112447.08
2        3        3     Councillor Baker  133669338 121914.90
3        4        4    Councillor Yancey  102142499  68001.37
4        5        5  Councillor Consalvo  218415447  90278.33
```

Hide

```
proj4string(boston)
```

```
[1] "+proj=lcc +lat_1=41.71666666666667 +lat_2=42.68333333333333 +lat_0=41 +lon_0=-71.5 +x_0=199999.9999999999 +y_0=750000 +
datum=NAD83 +units=us-ft +no_defs +ellps=GRS80 +towgs84=0,0,0"
```

Combining Spatial Data with Other Data Sources

Merging

First, load the elections tabular data.

Hide

```
boston.elections <- read.csv("~/Dropbox/Priya-PhD- Documents/Courses/Data Analysis and Visualization-Spring 2019/Datasets/Bo
stonData/dist_results.csv")
```

Now we have i) a SpatialPolygons object named boston, and ii) a data frame called boston.elections, where DISTRICT is the column that contains the unique identifier in boston, and district is the column that contains the unique identifier in boston.elections. We will merge the Spatial object (boston) with other tabular data (boston.elections), using the common unique identifier.

Hide

```
boston <- merge(boston, boston.elections,
                by.x = "DISTRICT", by.y = "District")
boston@data[1:5, ]
```

```
  DISTRICT OBJECTID           Councillor SHAPE_area SHAPE_len Winning_Perc
1        1        1 Councillor LaMattina  188299817 220959.06    0.9690193
2        2        2   Councillor Linehan  132852622 112447.08    0.5027723
3        3        3     Councillor Baker  133669338 121914.90    0.5578289
4        4        4    Councillor Yancey  102142499  68001.37    0.8853764
5        5        5  Councillor Consalvo  218415447  90278.33    0.9884647
  Runner_Perc        MOV
1  0.03098067 0.93803867
2  0.49316832 0.00960396
3  0.43676455 0.12106435
4  0.09893109 0.78644530
5  0.01153534 0.97692933
```

Spatial Merges

Before merging two Spatial datasets, check whether they have the same CRS. Because both files should agree on which exact point on Earth they refer to with a given value. If you try to merge two spatial objects with a different CRS, results won't make much sense.

If they don't have the same CRS, you'll need to reproject one of your objects. (Reprojecting changes not just the proj4string associated with an object, but also all the actual x and y coordinates.)

Reproject any vector data very simply using the sptransform() command, which will need two inputs: i) the object you want to reproject, and ii) the CRS code you want for your reprojection.

Hide

```
common.crs <- CRS("+init=EPSG:4326")
MyCity.reprojected <- spTransform(MyCity, common.crs)
```

Hide

```
common.crs <- CRS(proj4string(MyCityB))
MyCityA.reprojected <- spTransform(MyCityA, common.crs)
```

So far we have used a common unique identifier to merge, as with the regular datasets. We can also use the geolocation to merge. We are going to work with Boston municipal service requests for graffiti removal.

We already know the geolocation of graffitis but we don't know in which district of Boston they are located. So we will just overlay the graffitis with our Boston map using the over() command, which will match each graffiti with a district. After the match we'll combine the two data using the spCbind() command.

Start by loading graffiti data.frame and turning it into a SpatialPointsDataFrame so that R understands that it is a geospatial file.

Hide

```
boston.requests <- read.csv("~/Dropbox/Priya-PhD- Documents/Courses/Data Analysis and Visualization-Spring 2019/Datasets/Bos
tonData/graffiti.csv")
boston.graf <- SpatialPointsDataFrame(coords = cbind(boston.requests$LONGITUDE,
                                                     boston.requests$LATITUDE),
                                      data = boston.requests,
                                      proj4string = CRS("+proj=longlat"))
```

Next we need to make sure the Boston map has the same projection as our spatial point data frame:

Hide

```
boston <- spTransform(boston, CRS("+proj=longlat"))
```

Now for every graffiti, we want to get information about their district. We will use the over() command. The command basically asks R to return the row number of the observation in Data 2 that intersects with any given observation in Data 1.

Hide

```
boston.districts <- over(boston.graf, boston)
boston.districts[1:5, ]
```

```
  DISTRICT OBJECTID           Councillor SHAPE_area SHAPE_len Winning_Perc
1        3        3     Councillor Baker  133669338  121914.9    0.5578289
2        3        3     Councillor Baker  133669338  121914.9    0.5578289
3        6        6  Councillor O'Malley  255919050  119570.0    0.9754722
4        7        9   Councillor Jackson  114280199   65404.6    0.8434874
5        1        1 Councillor LaMattina  188299817  220959.1    0.9690193
  Runner_Perc       MOV
1  0.43676455 0.1210643
2  0.43676455 0.1210643
3  0.02452781 0.9509444
4  0.13988095 0.7036064
5  0.03098067 0.9380387
```

Note that over only returns the relevant row of Data 2 for each point. If boston did not have any data, we could just get the index of the intersecting polygon for each graffiti. If we just want to know the polygon index, we can use the geometry() command to remove the rest of the boston.graf data:

Hide

```
boston.districts2 <- over(boston.graf, geometry(boston))
boston.districts2[1:5]
```

```
1 2 3 4 5
3 3 6 9 1
```

Finally we use the spCbind() command to merge the two spatial objects.

Hide

```
boston2 <- spCbind(boston.graf, boston.districts)
names(boston2)
```

```
 [1] "CASE_ENQUIRY_ID"                "OPEN_DT"
 [3] "CLOSED_DT"                      "TIME_DIFF"
 [5] "CASE_STATUS"                    "CLOSURE_REASON"
 [7] "CASE_TITLE"                     "SUBJECT"
 [9] "REASON"                         "TYPE"
[11] "QUEUE"                          "Department"
```

```
[13] "Location"                    "neighborhood"
[15] "neighborhood_services_district" "LOCATION_STREET_NAME"
[17] "LOCATION_ZIPCODE"            "LATITUDE"
[19] "LONGITUDE"                   "Source"
[21] "completion.time"             "DISTRICT"
[23] "OBJECTID"                    "Councillor"
[25] "SHAPE_area"                  "SHAPE_len"
[27] "Winning_Perc"                "Runner_Perc"
[29] "MOV"
```

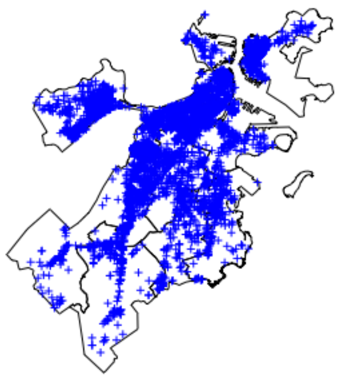
Making Maps

Making Maps with plot()

Create a very simple plot where we overlay the Boston districts and graffiti locations:

Hide

```
plot(boston)
plot(boston.graf, col="blue", cex=0.5, add=T)
```



Choropleth maps

Another useful plotting method is coloring the polygons based on the values in one of the attributes.Example, we can color the districts based on the margin of victories in the city council elections.

For this we will require to: i) choose n - the number of cuts we want to use, ii) obtain the quantile values that indicate the cutoff points, iii) pick a palette we like from amongst the options available in the brewer.pal() function. Then obtain our color codes using the findColours() function.

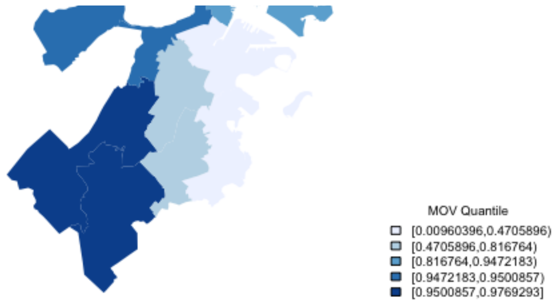
Hide

```
#determine the color codes
var <- as.numeric(boston$MOV)
nclr <- 5
plotclr <- brewer.pal(nclr,"Blues")
class <- classIntervals(var, nclr, style="quantile")
colcode <- findColours(class, plotclr)
```

Now you can just pass these color codes to plot():

Hide

```
plot(boston, col = colcode, border=NA) #remove the borders since looks prettier
legend(x = "bottomright",
       fill = attr(colcode,"palette"), #set the legend colors
       bty = "n", #remove the around the legend
       legend = names(attr(colcode, "table")), #show the quantile values
       title = "MOV Quantile",
       cex = 0.6) #adjust legend size
```



You can choose another color from the RColorBrewer package. You can see a list of all the color pallets that come with RColorBrewer with the following command:

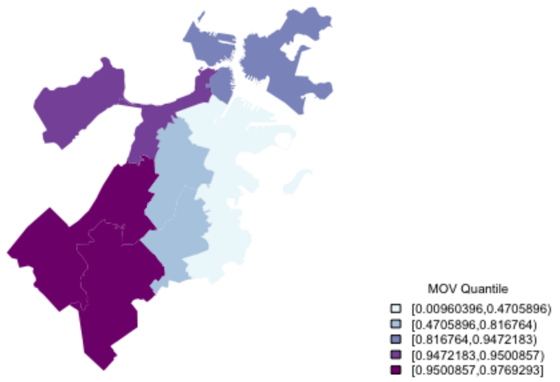
Hide

```
display.brewer.all()
```

We can also add labels to our map using the text() command.et's make the same plot above with a different color pallet.

Hide

```
#obtain the new color codes:
plotclr <- brewer.pal(nclr,"BuPu")
colcode <- findColours(class, plotclr)
#plot:
plot(boston, col = colcode, border=NA) #remove the borders since looks prettier
legend(x = "bottomright",
       fill = attr(colcode,"palette"), #set the legend colors
       bty = "n", #remove the around the legend
       legend = names(attr(colcode, "table")), #show the quantile values
       title="MOV Quantile",
       cex = 0.6)
```



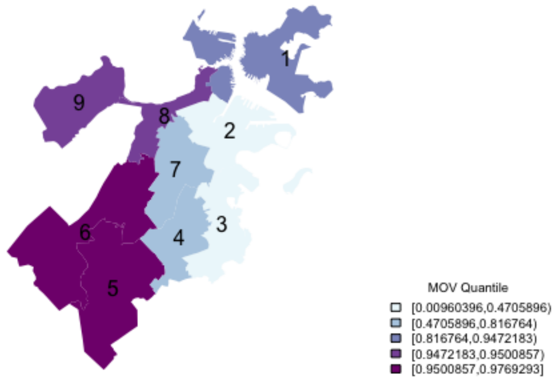
We can also add labels to our map using the text() command and indicating the coordinates where we want R to put our labels.

Hide

```
#store the central point of each polygon, where our labels will appear
centers = coordinates(boston)
# plot:
plot(boston, col = colcode, border=NA) #remove the borders for a prettier asthetic.
legend(x = "bottomright",
       fill = attr(colcode,"palette"), #set the legend colors
       bty = "n", #remove the around the legend
       legend = names(attr(colcode, "table")), #show the quantile values
       title="MOV Quantile",
       cex = 0.6)
```

Hide

```
text(centers,label=boston$DISTRICT)
```

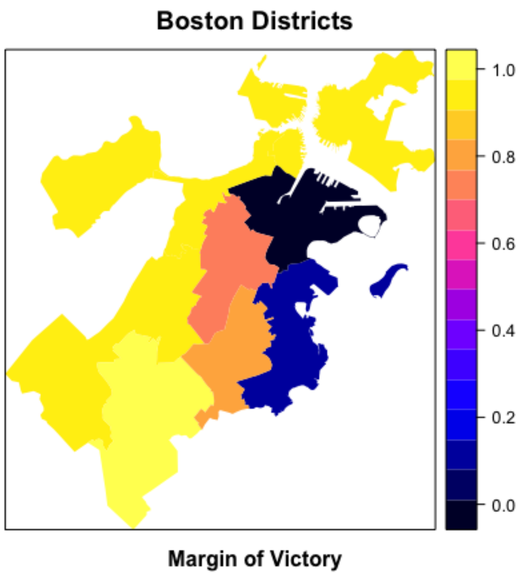


Making Maps with spplot()

spplot() is an extension of the plot() function in R specifically for making maps of spatial objects. It’s more convenient for filling in polygon colors based on attributes in the SpatialDataFrame object. You just need to pass the object and the name of columns you want to plot to the function. If you don’t pass specific column names, a separate figure will be created for each column.

Hide

```
spplot(boston,
       "MOV",
       main = "Boston Districts",
       sub = "Margin of Victory",
       col = "transparent") #remove the borders for a prettier asthetic.
```



Extract data from a raster

Hide

```
# load packages
library(raster)
library(rasterVis)
```

User the raster package to read the vegetation data.

Hide

```
# read potential natural vegetation data sage_veg30.nc:
vegtype_path <- "~/Downloads/"
```

```
vegtype_name <- "sage_veg30.nc"
vegtype_file <- paste(vegtype_path, vegtype_name, sep="")
vegtype <- raster(vegtype_file, varname="vegtype")
```

Hide

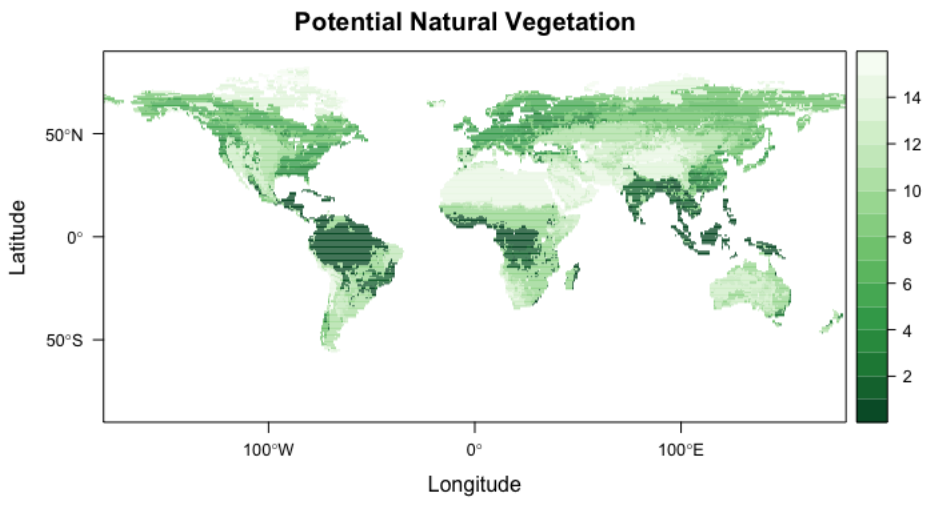
```
vegtype
```

```
class       : RasterLayer
dimensions  : 360, 720, 259200  (nrow, ncol, ncell)
resolution  : 0.5, 0.5  (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 : /Users/priya/Downloads/sage_veg30.nc
names       : vegetation.type
zvar        : vegtype
```

Plot the vegetation data using a rasterVis levelplot:

Hide

```
mapTheme <- rasterTheme(region=rev(brewer.pal(8,"Greens")))
levelplot(vegtype, margin=FALSE, par.settings=mapTheme,
                 main="Potential Natural Vegetation")
```



Spatial Data Analysis

Here, we will focus on how we detect and deal with spatial dependence in regressions models and on basic regression models.

Assessing Spatial Clustering

To detect spatial autocorrelation, we will use one informal and one formal method. For this part we will use a new SpatialPolygonsDataFrame and name it boston_n, which not only includes coordinates and unique ids for polygon units, but also extra columns - some demographic variables from 2000. We can start by loading our data and take a look at the content of the dataset as well as the simple plot.

Hide

```
boston_n <- readOGR(dsn = "~/Dropbox/Priya-PhD- Documents/Courses/Data Analysis and Visualization-Spring 2019/Datasets/Bosto
nData",
                   layer = "Boston_N")
```

```
OGR data source with driver: ESRI Shapefile
Source: "/Users/priya/Dropbox/Priya-PhD- Documents/Courses/Data Analysis and Visualization-Spring 2019/Datasets/BostonData",
layer: "Boston_N"
with 26 features
It has 8 fields
```

Hide

```
head(boston_n@data)
```

```
              Name OBJECTI      Acres SqMiles      ShpSTAr   ShpSTLn mnbchlr
0       Roslindale       1 1605.56824    2.51  69938272.9 53563.913   32.74
1    Jamaica Plain       2 2519.24539    3.94 109737890.8 56349.937   44.07
```

```
1    Jamaica Plain       2 2519.24539    3.94 109737890.8 56349.937   44.97
2     Mission Hill       3  350.85356    0.55  15283120.0 17918.724   40.24
3         Longwood       4  188.61195    0.29   8215903.5 11908.757   34.86
4      Bay Village       5   26.53984    0.04   1156070.8  4650.635   41.90
5 Leather District       6   15.63991    0.02    681271.7  3237.141   41.90
  blck_l_
0   33.06
1   43.52
2   35.93
3   26.51
4    7.72
5    7.72
```

Hide

```
plot(boston_n)
```

We are interested in is whether the share of Black and Hispanic population in a neighborhood,a variable named blck_l_ is associated with education attainment, as measured by the percentage of people with at least a bachelor degre, mnmmbchlr.

First we will check how our dataset looks like and change the column names to clearer names. We can manipulate the data in SpatialPolygonsDataFrame objects exactly as we do in the typical data.frame objects.

Hide

```
names(boston_n)
```

```
[1] "Name"    "OBJECTI" "Acres"   "SqMiles" "ShpSTAr" "ShpSTLn" "mnbchlr"
[8] "blck_l_"
```

Hide

```
names(boston_n)[7:8] <- c("minimumbachelor", "black_lat")
```

An informal approach to check for spatial autocorrelation. We will run a regression model where we regress education attainment on the share of hispanic and black population, and store the residuals.The idea is if neighboring residuals are similar in size, our model has some spatial autocorrelation. Then we will create a choroplet map, in which neighborhoods with similar-sized residuals will appear in similar shades.

Hide

```
#save residuals
lm.model <- lm(minimumbachelor ~ black_lat, data = boston_n)
boston_n$resid_ols <- lm.model$residuals

#get the color codes:
var <- as.numeric(boston_n$resid_ols)
nclr <- 5
plotclr <- brewer.pal(nclr,"Blues")
class <- classIntervals(var, nclr, style = "quantile")
colcode <- findColours(class, plotclr)

plot(boston_n, col = colcode, border = "grey")
legend(x = "bottomleft",
       fill = attr(colcode,"palette"),
       bty = "n",
       legend = names(attr(colcode, "table")),
       title = "OLS Residuals Quantile",
       cex = 0.6)
```

Normally the colors in the map should look arbitrarily distributed. But the autocorrelation map shows similar residual values for neighborhoods spatially close to each other, suggesting that there might be some spatial factors we haven't specified in our model.

We can do more formal tests of spatial autocorrelation by computing the spatial weight matrix. To compute the spatial weight matrix,first identify the neighborhoods that share edges using the poly2nb() command, and convert it to a matrix using the nb2listw() command. This matrix will also enable us to model the spatial autocorrelation among neighborhoods in our spatial regressions.

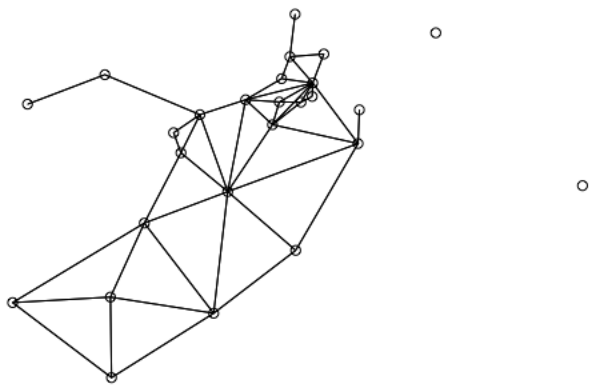
Hide

```
#make continuity NB (neighbors share edges)
continuity.nb <- poly2nb(boston_n,
                         queen = TRUE) #if FALSE, two units are neighbors only if they share a side (not an edge)
summary(continuity.nb) # shows how many links each polygon has, mean connections=5.2
```

```
Neighbour list object:
Number of regions: 26
Number of nonzero links: 92
Percentage nonzero weights: 13.60947
Average number of links: 3.538462
2 regions with no links:
11 25
Link number distribution:

0 1 2 3 4 5 6 8 9
2 3 4 4 5 4 2 1 1
3 least connected regions:
12 17 22 with 1 link
1 most connected region:
15 with 9 links
```

Hide

```
#plot neighbors
plot(continuity.nb, coordinates(boston_n))
```



Hide

```
#convert to a weight-matrix
continuity.listw <- nb2listw(continuity.nb,
                             style = "W", #for standardized rows
                             zero.policy = TRUE)
#if TRUE, assigns zero to the lags of zones without neighbours
summary(continuity.listw, zero.policy = TRUE) #shows more info with row-standardized weights
```

```
Characteristics of weights list object:
Neighbour list object:
Number of regions: 26
Number of nonzero links: 92
Percentage nonzero weights: 13.60947
Average number of links: 3.538462
2 regions with no links:
11 25
Link number distribution:
```

```
0 1 2 3 4 5 6 8 9
2 3 4 4 5 4 2 1 1
3 least connected regions:
12 17 22 with 1 link
1 most connected region:
15 with 9 links

Weights style: W
Weights constants summary:
   n  nn S0       S1       S2
W 24 576 24 15.42398 104.2362
```

Hide

```
head(continuity.listw$weights) #shows the actual weights for each polygon
```

```
[[1]]
[1] 0.25 0.25 0.25 0.25

[[2]]
[1] 0.2 0.2 0.2 0.2 0.2

[[3]]
[1] 0.25 0.25 0.25 0.25

[[4]]
[1] 0.5 0.5

[[5]]
[1] 0.25 0.25 0.25 0.25

[[6]]
[1] 0.5 0.5
```

One formal test of spatial dependence is the Moran's I, which is easily executed using the moran.test() function and passing the weights created above, as well as the dependent variable we are interested in, to it. We can also observe the spatial dependence on a plot relying on the moran.plot() command.

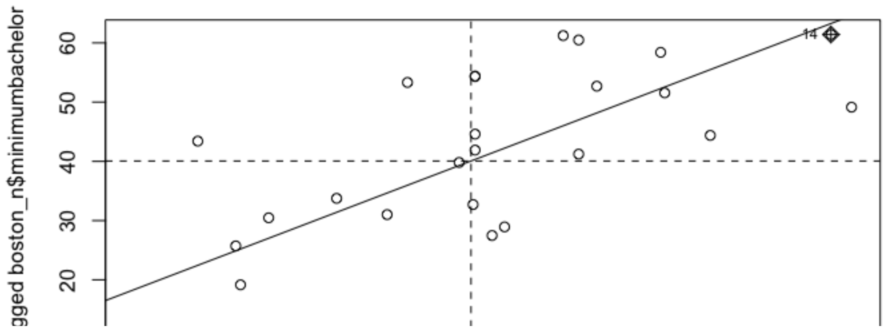
Hide

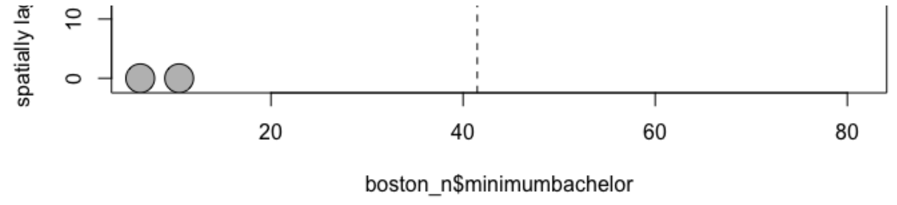
```
moran.test(boston_n$minimumbachelor,
           listw = continuity.listw,
           zero.policy = TRUE, #if TRUE, assigns zero to the lagged value of zones without neighbours
           adjust.n = TRUE)#if TRUE, the number of observations is adjusted for no-neighbour observations
```

```


    Moran I test under randomisation

data:  boston_n$minimumbachelor
weights: continuity.listw  n reduced by no-neighbour observations


Moran I statistic standard deviate = 2.5398, p-value = 0.005545
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
       0.34111187       -0.04347826        0.02292917
```

Hide

```
moran.plot(boston_n$minimumbachelor,
           continuity.listw,
           zero.policy = TRUE)
```





The statistic ranges from -1 to 1, and the stat of 0.798 means it's significantly spatially correlated.

Calculate local Moran's I statistics for each polygon rather than just global values. For local Moran's I statistics, we use the localmoran() command.

Hide

```
locmoran.boston_n <- localmoran(boston_n$minimumbachelor,
                          continuity.listw,
                          zero.policy = TRUE)
head(locmoran.boston_n)
```

```
             Ii  E.Ii    Var.Ii        Z.Ii Pr(z > 0)
0  0.2408182217 -0.04 0.2073821  0.61665129 0.2687324
1 -0.1154786299 -0.04 0.1585651 -0.18954846 0.5751685
2  0.0054051954 -0.04 0.2073821  0.09970568 0.4602890
3 -0.2063822346 -0.04 0.4514674 -0.24762459 0.5977876
4  0.0144825088 -0.04 0.2073821  0.11963864 0.4523847
5  0.0004766308 -0.04 0.4514674  0.06024086 0.4759819
```

Hide

```
boston_n <- spCbind(boston_n, as.data.frame(locmoran.boston_n))
```

Instead of mapping the residuals as above, we can now map the Z scores in the local Moran's I test output.

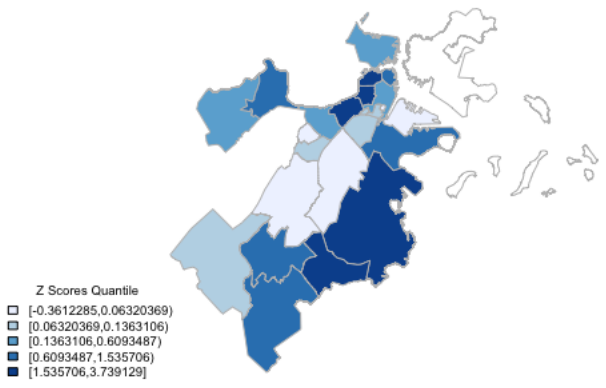
Hide

```
#get the color codes:
var <- as.numeric(boston_n$Z.Ii)
nclr <- 5
plotclr <- brewer.pal(nclr,"Blues")
class <- classIntervals(var, nclr, style="quantile")
```

```
var has missing values, omitted in finding classes
```

Hide

```
colcode <- findColours(class, plotclr)
plot(boston_n, col = colcode, border = "grey")
legend(x = "bottomleft",
       fill = attr(colcode,"palette"),
       bty = "n",
       legend = names(attr(colcode, "table")),
       title = "Z Scores Quantile",
       cex = 0.6)
```



Now we can explicitly see that there is spatial dependence among neighborhoods.

Spatial Regression

Having detected spatial dependence, we can now take it into account in our regression models. Spatial dependence in a regression setting can be modeled similarly to the autocorrelation in time series. The presence of spatial autocorrelation means a nonzero correlation with the error term, which implies that OLS estimates in the non-spatial model will be biased and inconsistent.

The estimation of the spatial autoregressive model (SAR) can be done with the function lagsarl(). Here we assume normality of the error term and use maximum likelihood. This model will give us the value of rho parameter and its p-value, and allow us comparing it with standard OLS.

Hide

```
summary(my.model.sar <- lagsarlm(minimumbachelor ~ black_lat,
                                 data = boston_n,
                                 continuity.listw,
                                 zero.policy = TRUE))
```

```

Call:lagsarlm(formula = minimumbachelor ~ black_lat, data = boston_n,
    listw = continuity.listw, zero.policy = TRUE)

Residuals:
     Min       1Q   Median       3Q      Max
-17.8753  -5.8921  -1.9245   5.8259  27.5723

Type: lag
Regions with no neighbours included:
 11 25
Coefficients: (asymptotic standard errors)
            Estimate Std. Error z value  Pr(>|z|)
(Intercept) 33.59490    7.54598  4.4520 8.506e-06
black_lat   -0.41208    0.10759 -3.8301 0.0001281

Rho: 0.47336, LR test value: 8.8257, p-value: 0.0029702
Asymptotic standard error: 0.13514
    z-value: 3.5027, p-value: 0.00046063
Wald statistic: 12.269, p-value: 0.00046063

Log likelihood: -100.1283 for lag model
ML residual variance (sigma squared): 121.54, (sigma: 11.025)
Number of observations: 26
Number of parameters estimated: 4
AIC: 208.26, (AIC for lm: 215.08)
LM test for residual autocorrelation
test value: 0.0048895, p-value: 0.94425
```

We can map the new residuals to see if the new model helped with autocorrelation.

Hide

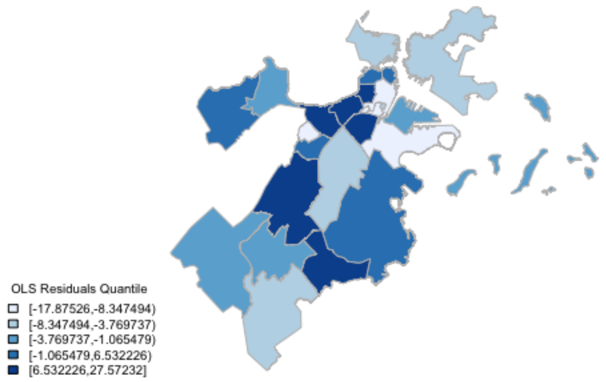
```
boston_n <- spCbind(boston_n, my.model.sar$residuals)
head(boston_n@data)
```

```
              Name OBJECTI      Acres SqMiles     ShpSTAr   ShpSTLn
0       Roslindale       1 1605.56824    2.51  69938272.9 53563.913
1    Jamaica Plain       2 2519.24539    3.94 109737890.8 56349.937
2     Mission Hill       3  350.85356    0.55  15283120.0 17918.724
3         Longwood       4  188.61195    0.29   8215903.5 11908.757
4      Bay Village       5   26.53984    0.04   1156070.8  4650.635
5 Leather District       6   15.63991    0.02    681271.7  3237.141
  minimumbachelor black_lat  resid_ols            Ii  E.Ii    Var.Ii
0           32.74     33.06  -5.099691  0.2408182217 -0.04 0.2073821
1           44.97     43.52  13.296055 -0.1154786299 -0.04 0.1585651
2           40.24     35.93   4.092058  0.0054051954 -0.04 0.2073821
3           34.86     26.51  -6.840650 -0.2063822346 -0.04 0.4514674
4           41.90      7.72 -10.876593  0.0144825088 -0.04 0.2073821
5           41.90      7.72 -10.876593  0.0004766308 -0.04 0.4514674
         Z.Ii Pr.z...0. my.model.sar.residuals
0  0.61665129 0.2687324              -1.909212
1 -0.18954846 0.5751685              15.613640
2  0.09970568 0.4602890               2.605512
3 -0.24762459 0.5977876             -13.050250
4  0.11963864 0.4523847             -14.264522
5  0.06024086 0.4759819              -8.347494
```

Hide

```
#get the color codes:
var <- as.numeric(boston_n$my.model.sar.residuals)
```

```
nclr <- 5
plotclr <- brewer.pal(nclr,"Blues")
class <- classIntervals(var, nclr, style="quantile")
colcode <- findColours(class, plotclr)
plot(boston_n, col = colcode, border = "grey")
legend(x = "bottomleft",
       fill = attr(colcode,"palette"),
       bty = "n",
       legend = names(attr(colcode, "table")),
       title = "OLS Residuals Quantile",
       cex = 0.6)
```



It looks a bit better.

Alternatively we can model spatial autocorrelation by specifing the autoregressive process in the error term relying on the errorsarlm function:

Hide

```
#error auto-Regresive
summary(my.model.ear <- errorsarlm(minimumbachelor ~ black_lat,
                                   data = boston_n,
                                   continuity.listw,
                                   zero.policy = TRUE))
```

```


Call:errorsarlm(formula = minimumbachelor ~ black_lat, data = boston_n,
    listw = continuity.listw, zero.policy = TRUE)

Residuals:
      Min        1Q    Median        3Q       Max
-20.50404  -6.95151   0.27443  10.75028  27.16799

Type: error
Regions with no neighbours included:
 11 25
Coefficients: (asymptotic standard errors)
            Estimate Std. Error z value  Pr(>|z|)
(Intercept) 51.75193    6.03055  8.5816 < 2.2e-16
black_lat   -0.58032    0.12462 -4.6567 3.214e-06

Lambda: 0.56342, LR test value: 3.5274, p-value: 0.060362
Asymptotic standard error: 0.17381
    z-value: 3.2416, p-value: 0.0011886
Wald statistic: 10.508, p-value: 0.0011886

Log likelihood: -102.7775 for error model
ML residual variance (sigma squared): 144.44, (sigma: 12.018)
Number of observations: 26
Number of parameters estimated: 4
AIC: 213.55, (AIC for lm: 215.08)
```