Introduction to the analysis of spatial data using R

Ana I. Moreno-Monroy

13-16 June 2017

Chapter 3: Point data

- Georeferenced point data: an example

- From points to Spatial Points
- Additional operations with spatial points
- Merging point data and spatial polygons
- Mapping with ggplot2 and leaflet

Georeferenced point data: an example

Load geolocalized crime data available from **ggmap** package

```
library(ggmap)
str(crime)
```

```
## 'data.frame':    86314 obs. of  17 variables:
##  $ time    : POSIXt, format: "2010-01-01 07:00:00" "2010-01-01 07:00:00" ...
##  $ date    : chr  "1/1/2010" "1/1/2010" "1/1/2010" "1/1/2010" ...
##  $ hour    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ premise : chr  "18A" "13R" "20R" "20R" ...
##  $ offense : Factor w/ 7 levels "aggravated assault",..: 4 6 1 1 1 3 3 3 3 3 ...
##  $ beat    : chr  "15E30" "13D10" "16E20" "2A30" ...
##  $ block   : chr  "9600-9699" "4700-4799" "5000-5099" "1000-1099" ...
##  $ street  : chr  "marlive" "telephone" "wickview" "ashland" ...
##  $ type    : chr  "ln" "rd" "ln" "st" ...
##  $ suffix  : chr  "-" "-" "-" "-" ...
##  $ number  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ month   : Ord.factor w/ 8 levels "january"<"february"<..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ day     : Ord.factor w/ 7 levels "monday"<"tuesday"<..: 5 5 5 5 5 5 5 5 5 5 ...
##  $ location: chr  "apartment parking lot" "road / street / sidewalk" "residence / house" "residence /
##  $ address : chr  "9650 marlive ln" "4750 telephone rd" "5050 wickview ln" "1050 ashland st" ...
##  $ lon     : num  -95.4 -95.3 -95.5 -95.4 -95.4 ...
##  $ lat     : num  29.7 29.7 29.6 29.8 29.7 ...
```

Georeferenced point data: an example

- Subset by type of offense after checking all possible values

```
levels(crime$offense)
```

```
## [1] "aggravated assault" "auto theft"         "burglary"
## [4] "murder"             "rape"               "robbery"
## [7] "theft"
```

- Subset to keep only murders

```
murder <- subset(crime, offense == "murder")
```

- Drop unwanted columns

```
murder<-murder[,colnames(murder) %in% c("number", "lon", "lat")]
```

Assigning coordinates

- Variables **lon** and **lat** refer to geographic planar coordinates. Check the first pair of values

```
c(murder$lat[1], murder$lon[1])
```

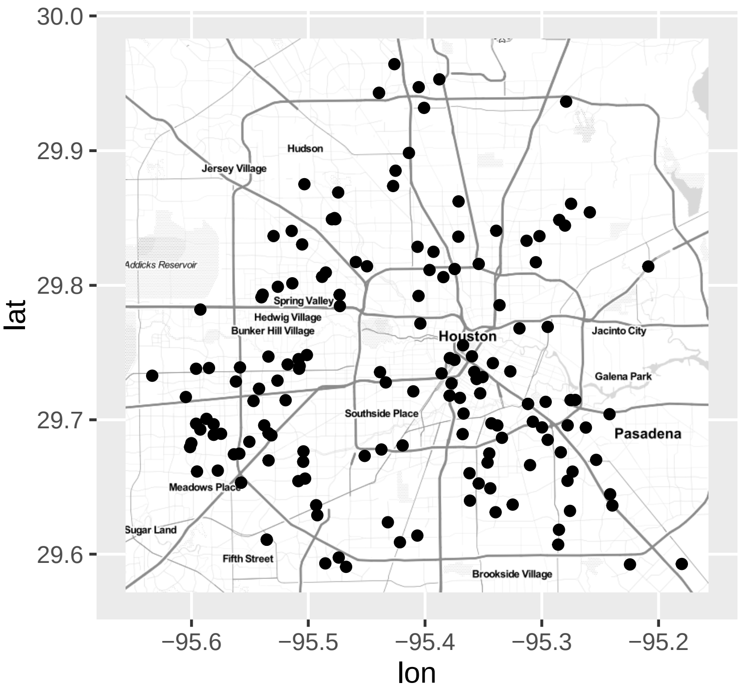
```
## [1]  29.67790 -95.43739
```

- Try searching for "29.67790, -95.43739" in Google Maps manually
- Each point refers to the location of a specific event, but where do these occur?

Plotting points in map with ggmap

If we do not have polygon data, we can use a background tile of the location to locate the points using the **qmplot** function of the **ggmap** package

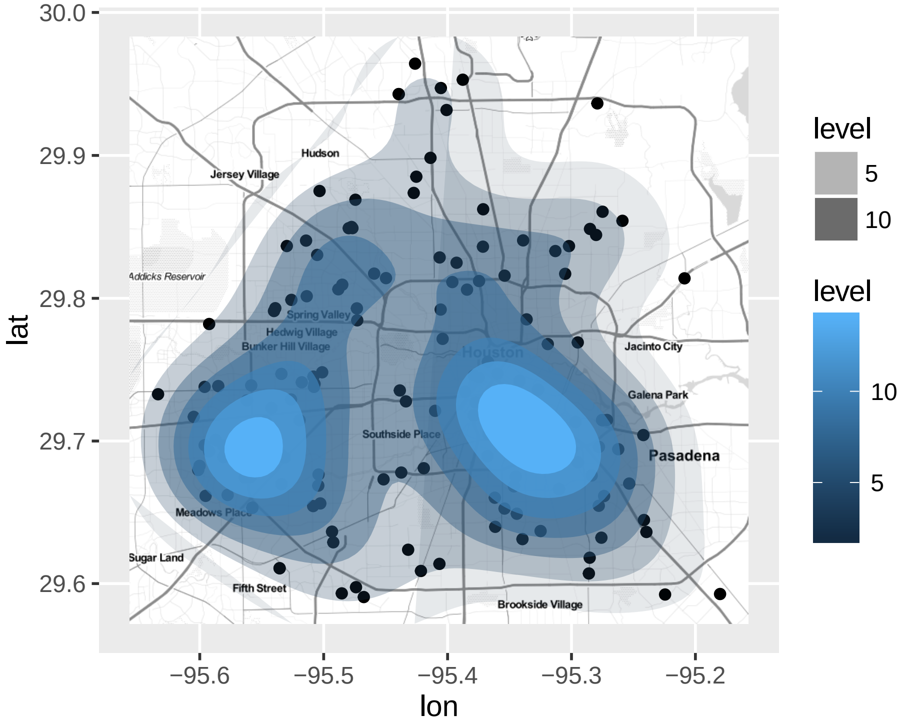
```
library(ggmap)
crime_map<-qmplot(lon, lat, data = murder, extent = "normal")
crime_map
```



Kernel density overlay

Base plot plus 2-D kernel density of murders

```
crime_map +
  stat_density2d(aes(x = lon, y = lat, fill = ..level.., alpha = ..level..), data = murder, geom = "pol
```



Conveting lat/lon into a Spatial Object

- To set coordinates and create a Spatial object, use the **coordinates** function of the **sp** package

```
library(sp)
coordinates(murder)=~lon+lat
```

- The dataframe has been converted in a SpatialPointsDataFrame

```
summary(murder)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##           min       max
## lon -95.63365 -95.18011
## lat  29.59057  29.96421
## Is projected: NA
## proj4string : [NA]
## Number of points: 157
## Data attributes:
##      number
##  Min.   :1.000
##  1st Qu.:1.000
##  Median :1.000
##  Mean   :1.057
##  3rd Qu.:1.000
##  Max.   :3.000
```

Spatial Points

- Check the slots

```
slotNames(murder)
```

```
## [1] "data"        "coords.nrs"  "coords"      "bbox"        "proj4string"
```

- bbox refers to the Bouding Box, or (geographical) extent of the data

```
bbox(murder)
```

```
##           min       max
## lon -95.63365 -95.18011
## lat  29.59057  29.96421
```

- No coordinates system assigned

```
proj4string(murder)
```

```
## [1] NA
```

Merging points and base polygons

- Import base polygons with Super Neighborhoods boundaries

```
library(rgdal)
neigh <- readOGR(dsn = "Super_Neighborhoods", layer = "Super_Neighborhoods")
```

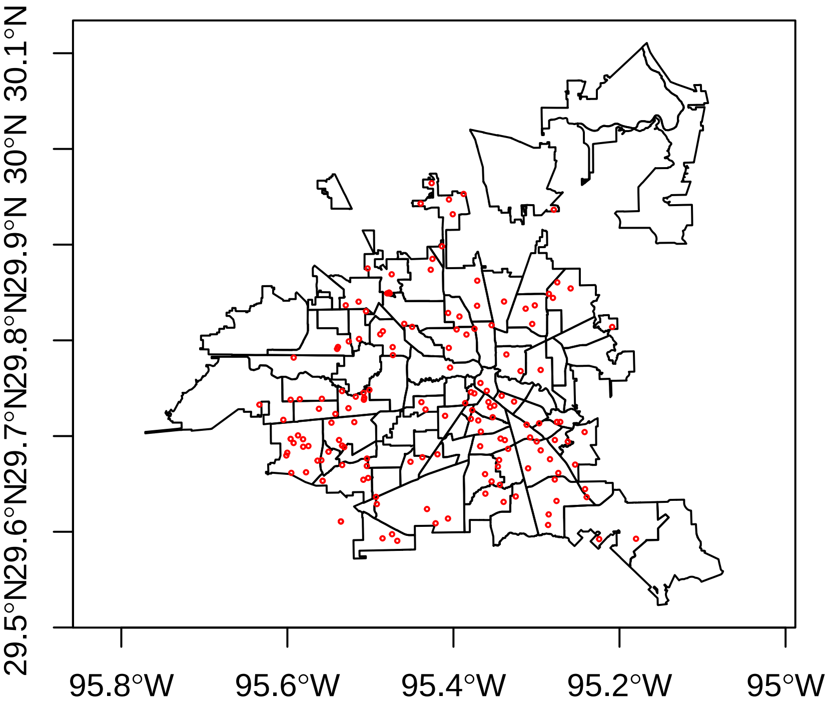
```
## OGR data source with driver: ESRI Shapefile
## Source: "Super_Neighborhoods", layer: "Super_Neighborhoods"
## with 88 features
## It has 11 fields
```

- For merging points and polygons, we must have them both in the same coordinate system. For this, we start by assigning to the spatial points the coordinates of the polygons

```
proj4string(murder)<-proj4string(neigh)
```

Basic plot to check overlay

```
plot(neigh, axes=TRUE)
points(murder, cex=0.3, col="red")
```



Merging Spatial Points and Polygons

- We need to aggregate the number of murders by Super Neighborhood
- First we subset murders happening within Super Neighborhoods boundaries

```
murder_o <- murder[neigh, ]
```

- By comparing the lengths of original points and the subset points, we can see two murders do not happen within SN Boundaires

```
length(murder)
```

```
## [1] 157
```

```
length(murder_o)
```

```
## [1] 155
```

Aggregating points

- Variable "number" refers to the number of murder represented by each point. To aggregate this variable by polygon we can use the **over** function of **sp** package

```
library(sp)
neigh@data$murders<-over(neigh, murder_o["number"], fn=sum)
```

- We added a data.frame to our @data slot. We can convert the data.frame into a variable

```
neigh@data$murders<-neigh@data$murders$number
```

- Alternatively, we can use the **aggregate** function of the **sp** package

```
neigh@data$murders <- aggregate(x = murder_o["number"], by = neigh, FUN = sum)
neigh@data$murders<-neigh@data$murders$number
```

Verification

- Check total sums of points and aggregates coincide

```
sum(neigh@data$murders, na.rm=TRUE)
```

```
## [1] 164
```

```
sum(murder_o$number)
```

```
## [1] 164
```

- Neighborhoods with no murders are assigned NA

```
table(is.na(neigh@data$murders))
```

```
##
## FALSE  TRUE
##    59    29
```

- We can assing zeros to NA values

```
neigh@data$murders[is.na(neigh@data$murders)]<-0
```

Additional operations with Spatial points

- Distance of every murder to the center of Downtown using the **distm** function of the **geosphere** package

```
library(rgeos)
library(geosphere)
center <- gCentroid(neigh[neigh$SNBNAME == "DOWNTOWN",])
dist_murder<-distm(murder_o, center)
```

- Mean distance of murders from downtown (in kilometers)

```
mean(dist_murder)/1000
```

```
## [1] 13.98999
```

More advanced plotting options

- The **ggplot2** package offers multiple possibilities for designing graphs. It is very well documented
- Interactive mapping is possible using the **leaflet** and **shiny** packages
- For some examples of beautiful mapping with ggplot2, see James Cheshire website

Interactive mapping with leaflet: an example

This is a plot of murders commited on a Monday (it can also be opened in a web browser from the right-top corner of the Viewer)

```r
library(leaflet)
murder <- subset(crime, offense == "murder")
murder_m<-murder[murder$day=="monday",]
murder_m<-murder_m[,colnames(murder_m) %in% c("time", "lon", "lat")]
map<-leaflet(data = murder_m) %>% addTiles() %>%
  addMarkers(~lon, ~lat)
map
```

Interactive mapping with shiny: an example

Now the plot also has a pop-up with the time the crime was committed

```r
library(shiny)
shinyApp(
  ui = fluidPage(leafletOutput('myMap')),
  server = function(input, output) {
      map<-leaflet(data = murder_m) %>% addTiles() %>%
      addMarkers(~lon, ~lat)  %>%
      addPopups(~lon, ~lat, popup = ~time) # popup
    output$myMap = renderLeaflet(map)
  }
)
```