Intro to geospatial data in R

MAIKE HOLTHUIJZEN

AUGUST 6, 2019

Contents

1 Basic mapping with ggmap and ggplot2

Ggmap and ggplot2 are excellent R packages for plotting geospatial data. You can use ggmap to download basemaps, over which you can plot point, polygon, line, and raster data. Because the maps come from Google, you do need to get a Google Static Maps API and associate it with your Google account. Directions are here: https://www.littlemissdata.com/blog/maps. Data for this tutorial are located here: https://github.com/maikeh7/geospatialData.

Once you have your account set up, you can start downloading and plotting base maps with ggmap:

```
library(ggmap)
#you will need your own Google key!
register_google(key = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX")

boise = c(lon = -116.2023, lat = 43.615)

# Get map at zoom level 5: map_5
map_5 = get_map(boise, zoom = 5, scale = 1)

# Plot map at zoom level 5
ggmap(map_5)

# Get map at zoom level 13: corvallis_map
boise_map = get_map(boise, zoom = 15, scale = 1)
ggmap(boise_map)
```

Here are the resulting plots:

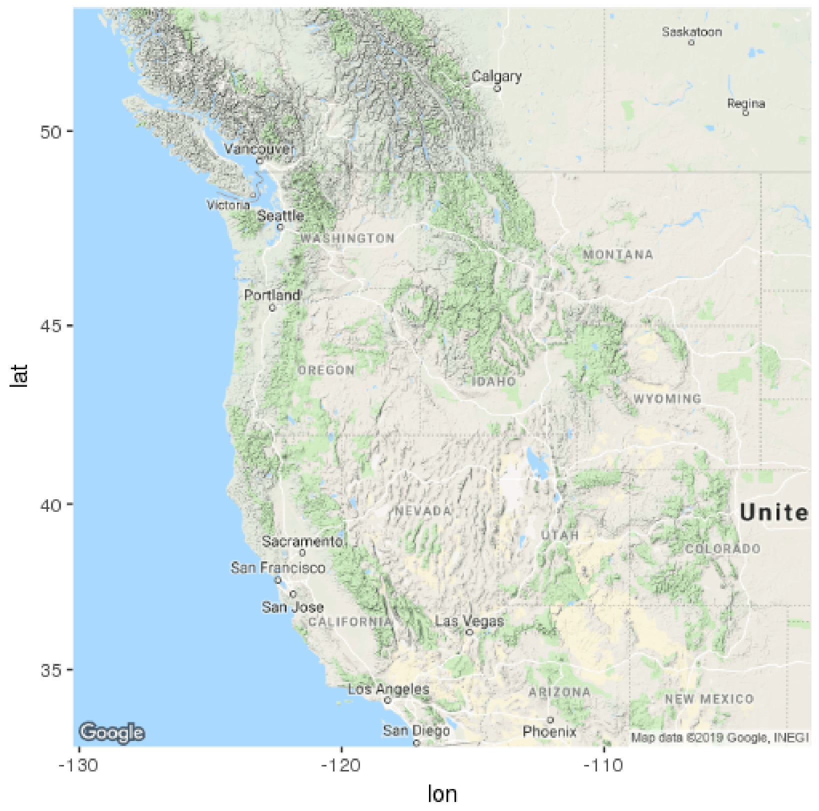

Figure 1: Basemap of Idaho with zoom level 5

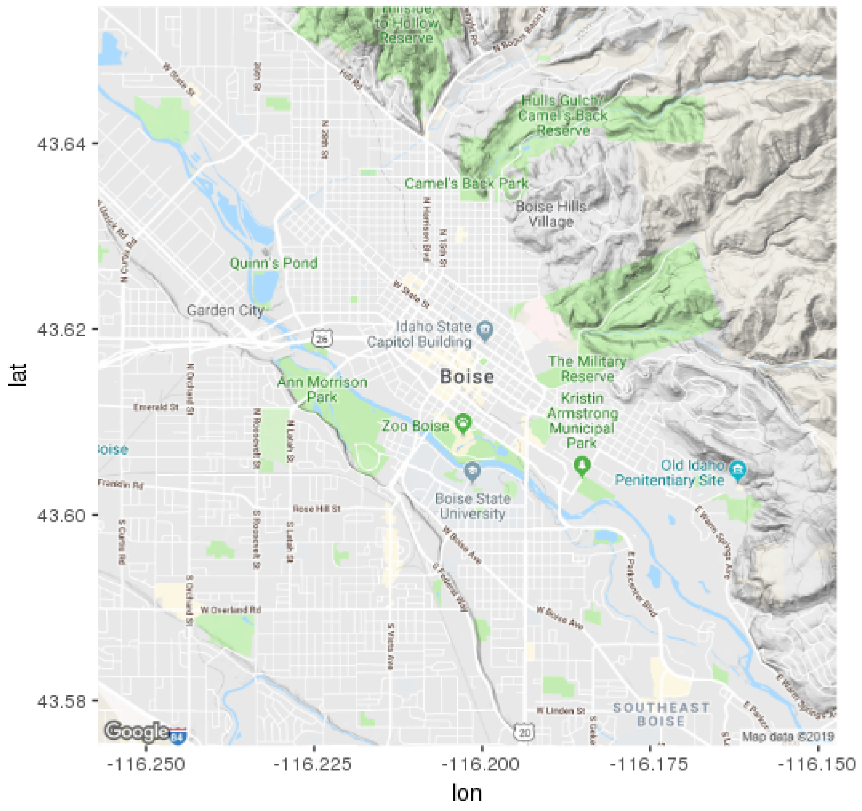

Figure 2: Basemap of Boise, Idaho with zoom level 13

You can also get lots of different map types using get_map. Just specify type =

```
    "terrain",
  "terrain-background", "satellite", "roadmap", "hybrid", "toner",
  "watercolor", "terrain-labels", "terrain-lines", "toner-2010",
  "toner-2011", "toner-background", "toner-hybrid", "toner-labels",
  "toner-lines", "toner-lite"), source = c("google", "osm", "stamen",
  "cloudmade"
```

as an argument to get_map.

Here is an example of a satellite map:

```
    boise_map <- get_map(boise, zoom = 15, scale = 1, maptype = "satellite")
ggmap(boise_map)
```

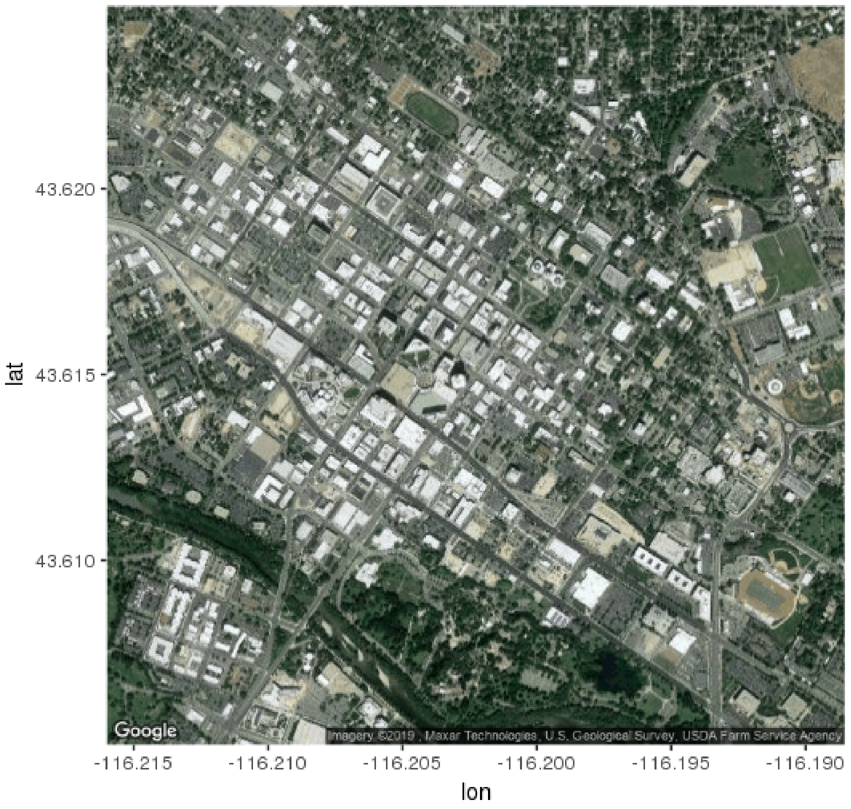
43.620
43.615
43.610
lat
-116.215
-116.210
-116.205
-116.200
-116.195
-116.190
lon
Google
Imagery ©2019, Maxar Technologies, U.S. Geological Survey, USDA Farm Service Agency

Figure 3: Basemap of Boise, Idaho with zoom level 13 and maptype = "satellite"

2 Point and polygon data

We can also add point data to a basemap with ggmap. All you need to add point data to a map is a data.frame with coordinates in lat/lon (and optionally, a variable value, as shown below). Data for this exercise can be found at: Here, we add the price of housing sales in Corvallis to a ggmap basemap using geom_point():

```
#map of corvallis, OR
corvallis <- c(lon = -123.2620, lat = 44.5646)

# Add a maptype argument to get a satellite map
corvallis_map_sat <- get_map(corvallis, zoom = 13)

sales = readRDS("01_corv_sales.rds")
head(sales)
# Add the points and color by year_built
ggmap(corvallis_map_sat) +
  geom_point(aes(lon, lat, color = year_built), data = sales)
```

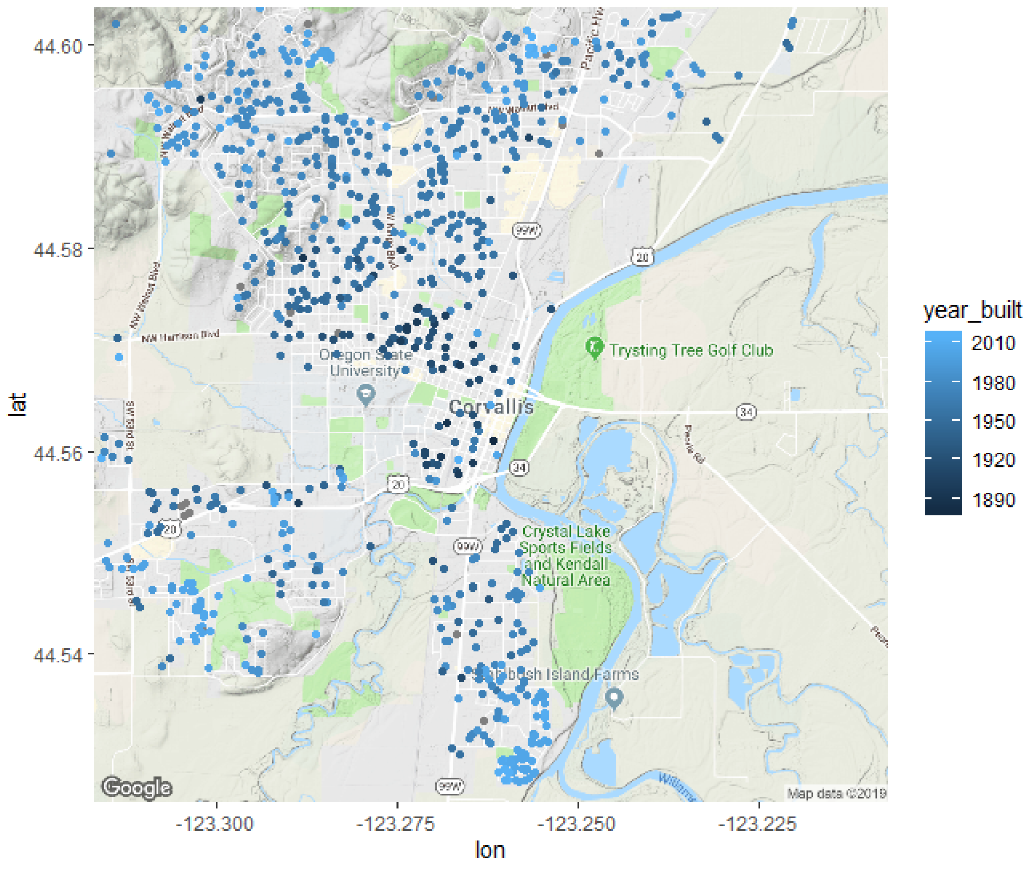

Figure 4: Map of housing sales data in Corvallis, OR

You can also plot polygon data using ggplot. In this particular example, we are going to plot polygons of housing wards in Corvallis, OR. An area may be described by more than one polygon and order matters. Group is an identifier for a single polygon, but a ward may be composed of more than one polygon, so you would see more than one value of group for such a ward. Order describes the order in which the points should be drawn to create the correct shapes.

In ggplot2, polygons are drawn with geom_polygon(). Each row of your data is one point on the boundary and points are joined up in the order in which they appear in the data frame. You specify which variables describe position using the x and y aesthetics and which points belong to a single polygon using the group aesthetic.

```
# Add a polygon layer with fill mapped to ward, and group to group
wards = readRDS("01_corv_wards.rds")
head(wards)
ggplot(wards, aes(lon, lat)) +
geom_polygon(aes(group = group, fill = ward))
```

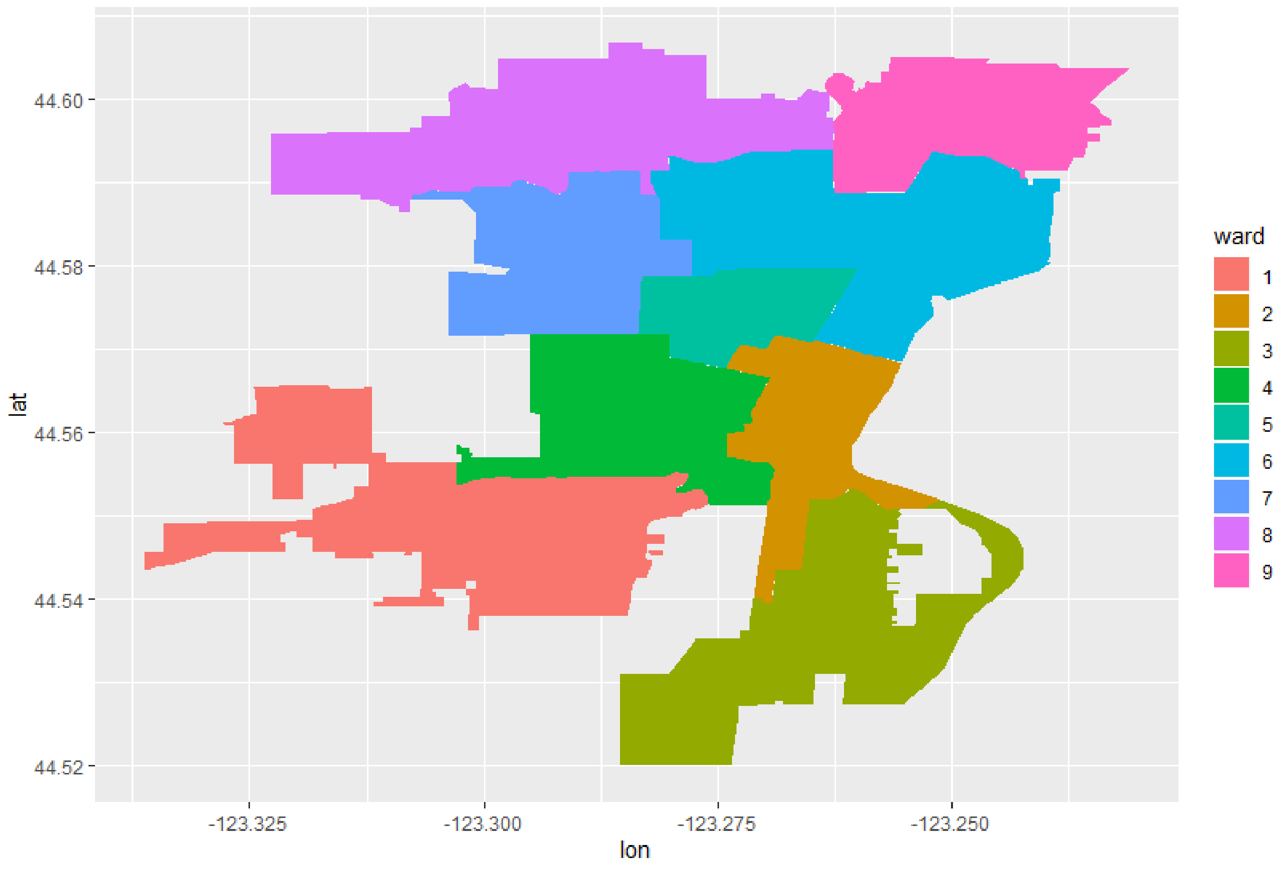

Figure 5: Polygon data plotted with ggplot

You can also plot polygon data over base maps using ggmap:

```
    corvallis_map_bw = get_map(corvallis, zoom = 12, maptype = "toner")
ggmap(corvallis_map_bw,
      base_layer = ggplot(wards, aes(lon, lat)),
      extent = "normal", maprange = FALSE) +
  geom_polygon(aes(group = group, fill = ward))
```

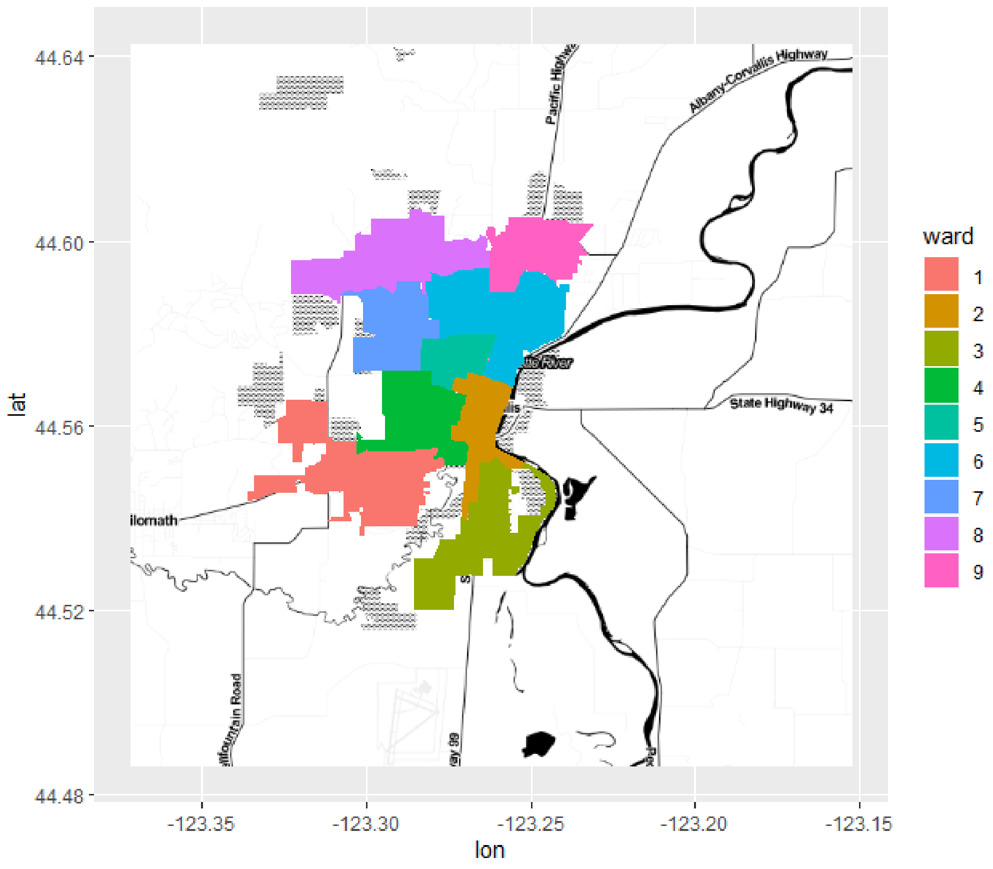

Figure 6: Polygon data plotted with ggmap

Next, we read in a shapefile and plot a subset of the polygons over the Idaho basemap.

```
library(raster)
library(sp)
counties = shapefile("tl_2013_16_cousub.shp")
#let's say we only want to plot some of the counties
idnames = as.character(0:25)
#need to fortify counties, creates a data.frame
f = fortify(counties)
#extract only the names we specified
f2  = filter(f, id %in% idnames)
ggmap(map_5) +
    geom_polygon(data = f2,
           aes(long, lat, group = group, fill = id)) +
           theme(legend.position="none")
```

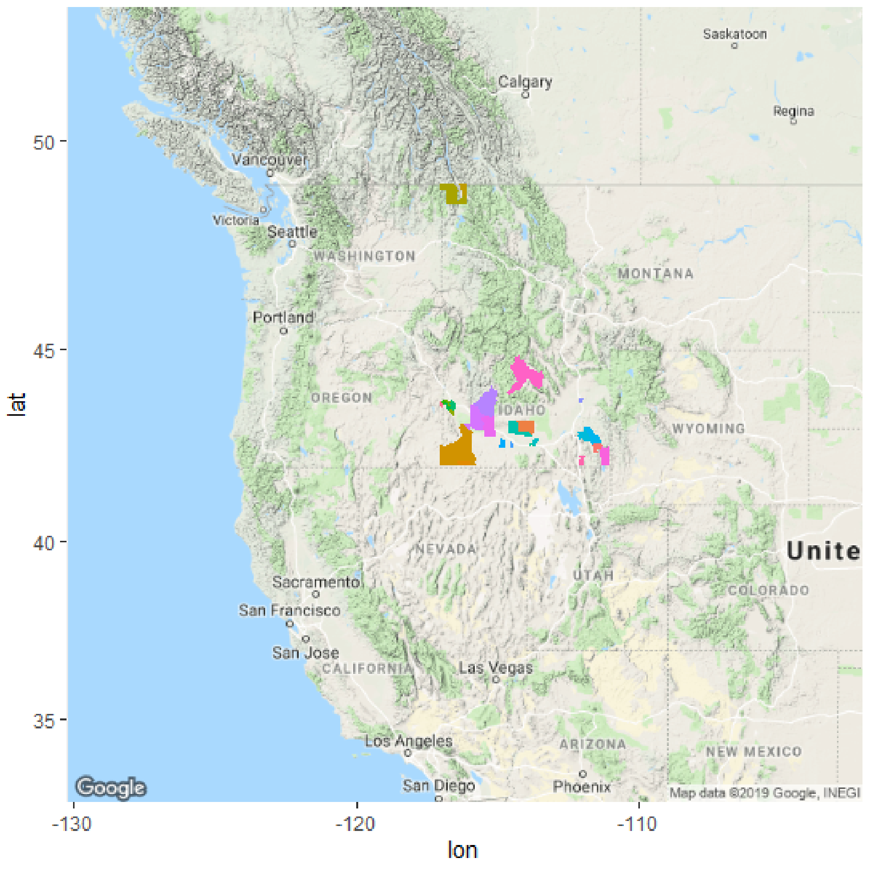

Figure 7: Some Idaho counties plotted over the basemap

Here are other useful packages for plotting and working with spatial data:

- sp. Use sp to create spatial objects for polygon, pixel, and point data. Also has good plotting capabilities with spplot()
- maptools: reading and writing spatial data, particularly shapefiles
- rgdal: R "bindings" to GDAL (Geospatial Data Abstraction Layer)
- rgeos: R interface to the GEOS "geometry engine" (overlays, etc.)

3 Raster data

The raster package in R is excellent for working with raster (gridded) data. To open a raster, you can use the function raster() from the raster package, and it will read in most formats of raster data. Here is an example. We will be working with a raster file of population in NYC.

```
library(raster)
#read in raster
pop = raster("Pop.img")
```

```
#get information about the raster layer
pop
> class        : RasterLayer
> dimensions   : 480, 660, 316800  (nrow, ncol, ncell)
> resolution   : 0.008333333, 0.008333333  (x, y)
> extent       : -75, -69.5, 39, 43  (xmin, xmax, ymin, ymax)
> coord. ref. : +proj=longlat +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +no_defs
> data source : D:\school\Climate_Analysis_Personal\ReportPlots\Pop.img
> names        : Pop
> values       : 0, 41140  (min, max)
```

You can get a simple plot of any raster by just calling plot() on your raster!

```
plot(pop)
```

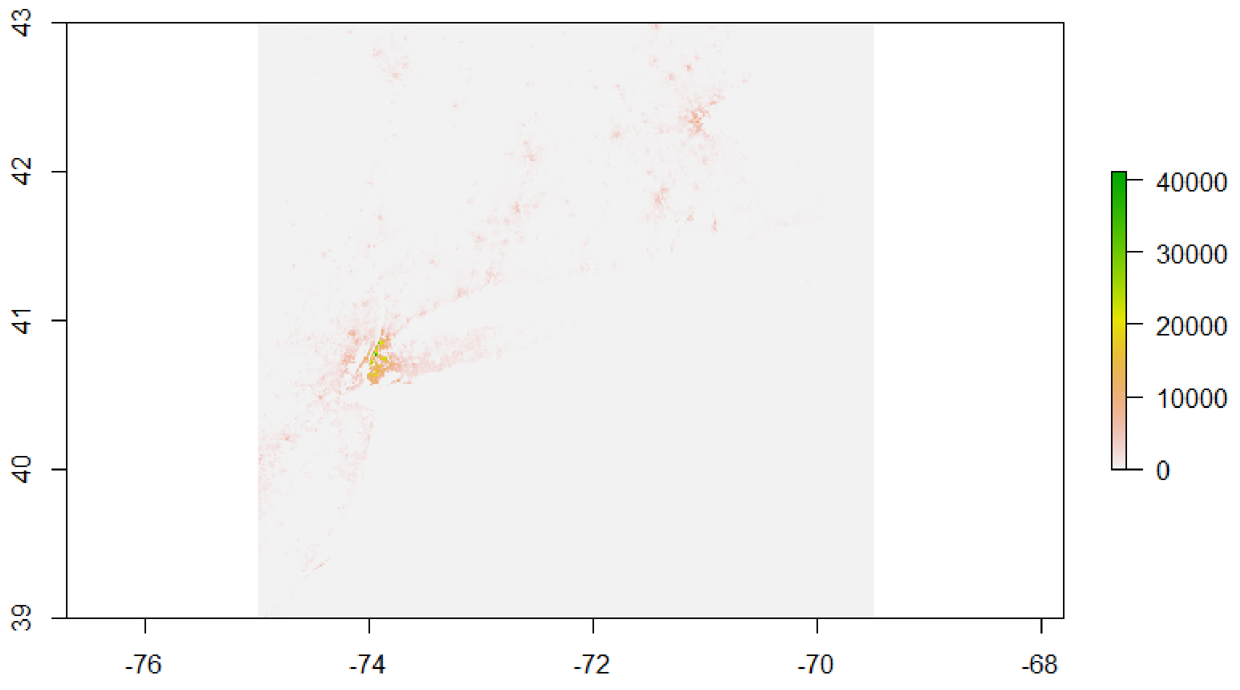

Figure 8: Plotting with the raster package

The spplot function also makes nice maps. You can easily change the map colors using the 'col.regions' argument.

```
spplot(mypop, xlab = "Lat", ylab = "Lon", col.regions = topo.colors(100))
```

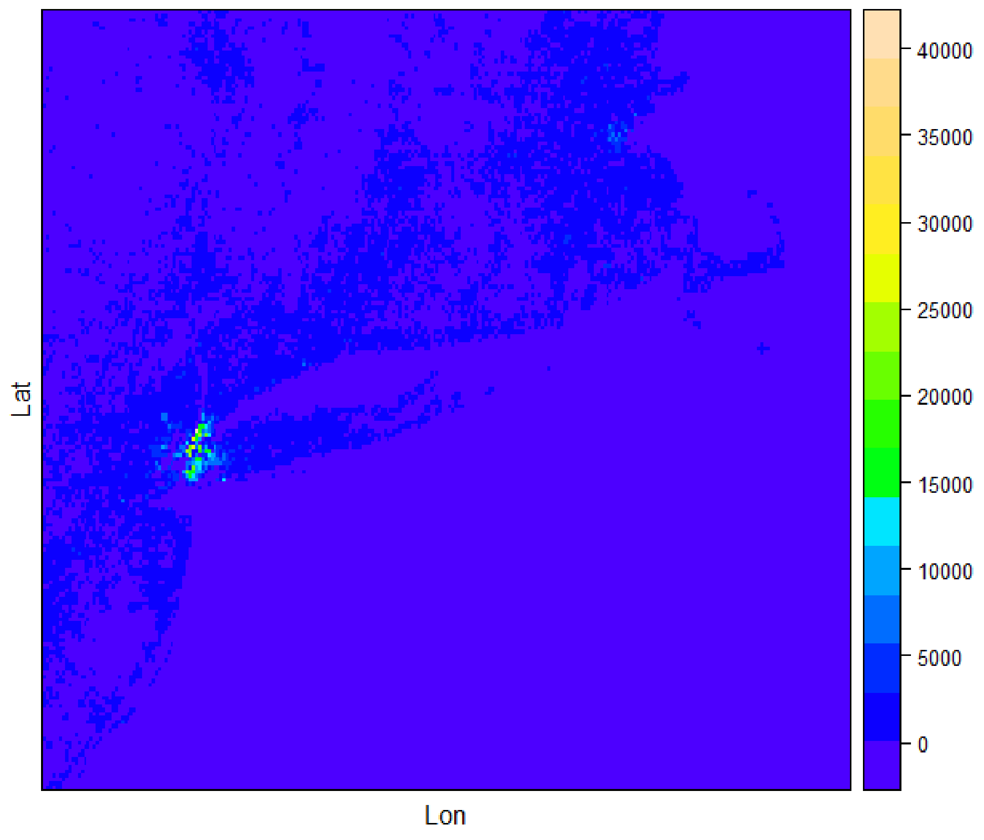

Figure 9: Plotting the population data with spplot

You can crop raster data using the crop() function. The arguments to crop are xmin, xmax, ymin, ymax, in that order. Here, we crop the population data to an extent.

```
extent(mypop)
extent(-74.758, -71.556, 43.39, 45.861)
new_extent = c(-75, -72, 37, 42)
mypop_crop = crop(mypop, new_extent)
spplot(mypop_crop)
```

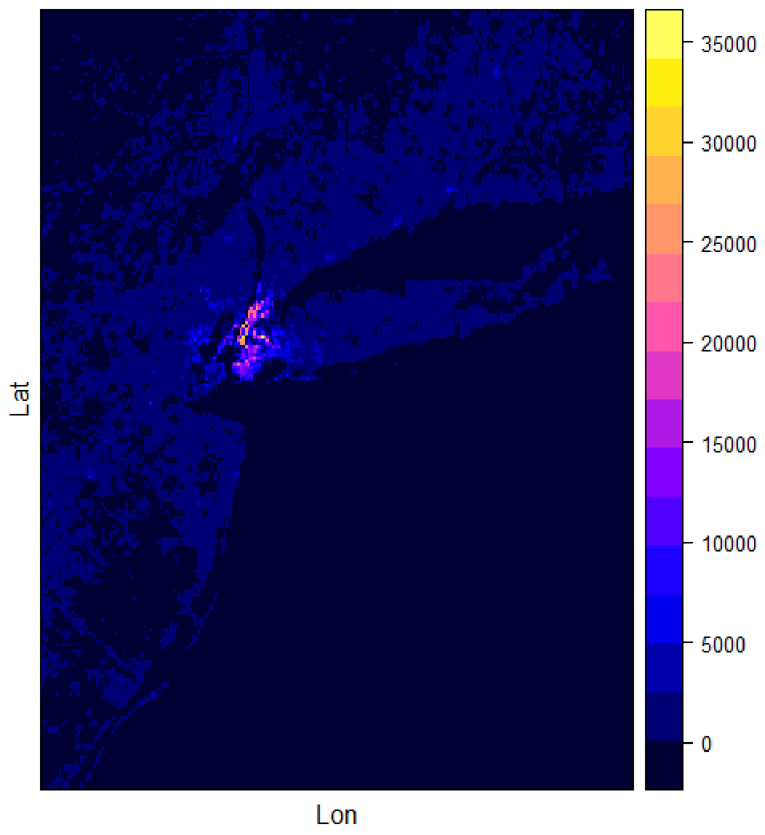

Figure 10: Plotting the cropped population data with spplot