Lead Testing Results for the City of St. Louis

Christopher Prener, Ph.D.

December 22, 2016

Introduction

This tutorial introduces a number of skills related to mapping data in `R`: data cleaning, table joins, and map production. `R` is not a replacement for a GIS like ArcGIS or QGIS. However, its mapping tools are useful for previewing data and creating quick maps, particularly of quantitative data. `R` is also particularly useful for its spatial statistics tools.

Packages

The following tutorial requires a number of packages from the `tidyverse` as well as other packages that facilitate mapping. If you need any of these packages, use the `install.packages("packageName")` function.

Note that you also need to have the package `rgeos` installed. It does not have to be loaded for this tutorial to work, but it must be present. Re-start RStudio after installing this package to ensure that you do not get any errors.

```r
# load required packages
library(dplyr) # data cleaning
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(rgdal) # read shapefiles
```

```
## Warning: package 'rgdal' was built under R version 3.3.2

## Loading required package: sp

## rgdal: version: 1.2-5, (SVN revision 648)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.1.2, released 2016/10/24
##  Path to GDAL shared files: /usr/local/share/gdal
##  Loaded PROJ.4 runtime: Rel. 4.9.1, 04 March 2015, [PJ_VERSION: 491]
##  Path to PROJ.4 shared files: (autodetected)
##  Linking to sp version: 1.2-3
```

```r
library(ggplot2) # create plots
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```r
library(maptools) # tools for mapping
```

```
## Warning: package 'maptools' was built under R version 3.3.2
## Checking rgeos availability: TRUE
```

```
library(ggmap) # tools for mapping
library(RColorBrewer) # pre-defined color ramps
library(Cairo) # high quality output
```

Data

The `csv` file included with this tutorial was created by exporting an attribute table from a U.S. Census TIGER/Line shapefile containing all current Census Tracts in the City of St. Louis.

The data (`countTested` and `pctElevated`) were obtained from the Reuters report on lead poisoning that was published on 19 Dec 2016. Each tract's data was copied from Reuters' map to a `csv` file manually.

`countTested` refers to the number of children screened for blood lead levels in the tract between 2010 and 2015. `pctElevated` is the percentage of children tested who had blood lead levels greater than or equal to 5 micrograms per deciliter. These were the original two values obtained from Reuters' reporting.

```
lead <- read.csv("STLLead.csv", stringsAsFactors = FALSE)
head(lead)
```

```
##   STATEFP COUNTYFP TRACTCE       GEOID    NAME             NAMELSAD
## 1      29      510  118100 29510118100 1181.00    Census Tract 1181
## 2      29      510  117400 29510117400 1174.00    Census Tract 1174
## 3      29      510  126700 29510126700 1267.00    Census Tract 1267
## 4      29      510  119102 29510119102 1191.02 Census Tract 1191.02
## 5      29      510  126800 29510126800 1268.00    Census Tract 1268
## 6      29      510  126900 29510126900 1269.00    Census Tract 1269
##   countTested pctElevated
## 1         345        9.57
## 2         871       12.06
## 3         458       18.12
## 4         182        2.20
## 5         486        4.73
## 6        1296       15.66
```

Data Cleaning

There are a number of geography variables that we do not need. We'll start by creating a variable named `id` that contains the `GEOID` data stored as character values. This will be important for mapping our lead level data. We'll then retain that new variable plus the `NAMELSAD` variable (useful for double-checking data) and our two data variables `countTested` and `pctElevated`.

```
lead$id <- as.character(lead$GEOID) # reformat and rename GEOID
lead$name <- lead$NAMELSAD # rename NAMELSAD

# drop all variables except those listed
lead <- select(lead, id, name, countTested, pctElevated)
head(lead)
```

```
##            id                 name countTested pctElevated
## 1 29510118100    Census Tract 1181         345        9.57
## 2 29510117400    Census Tract 1174         871       12.06
## 3 29510126700    Census Tract 1267         458       18.12
## 4 29510119102 Census Tract 1191.02         182        2.20
## 5 29510126800    Census Tract 1268         486        4.73
## 6 29510126900    Census Tract 1269        1296       15.66
```

Next, we want to get the count of children with elevated blood lead levels rather than just the percentage:

```
# create estimate of number of children with elevated blood lead levels
lead$estElevated <- round((lead$pctElevated*.01)*lead$countTested, digits = 0)
head(lead)
```

```
##            id                 name countTested pctElevated estElevated
## 1 29510118100    Census Tract 1181         345        9.57          33
## 2 29510117400    Census Tract 1174         871       12.06         105
## 3 29510126700    Census Tract 1267         458       18.12          83
## 4 29510119102 Census Tract 1191.02         182        2.20           4
## 5 29510126800    Census Tract 1268         486        4.73          23
## 6 29510126900    Census Tract 1269        1296       15.66         203
```

Basic Mapping in `R`

The first step for mapping lead levels is to import a shapefile that contains Census Tract boundaries for the City of St. Louis. This is done using the `rgdal` package. If you want to reference a shapefile located in another directory, you can specificy the file path using `dsn = "filePath"`. Otherwise, include the period as below to reference a shapefile stored in your working directory.

In the layer option, include the name of the shapefile *without the extension*.

After loading the shapefile, use the `ggplot2` function `fortify()` to convert the data from a shapefile to a dataframe that we can plot using `ggplot2`. This will dramatically increase the number of observations, and will create a variable named `id` that includes the `GEOID` data specified under the `region` option below. This variable will be important for mapping our lead level data.

```
# import shapefile
tract <- readOGR(dsn = ".", layer = "STLTracts")
```

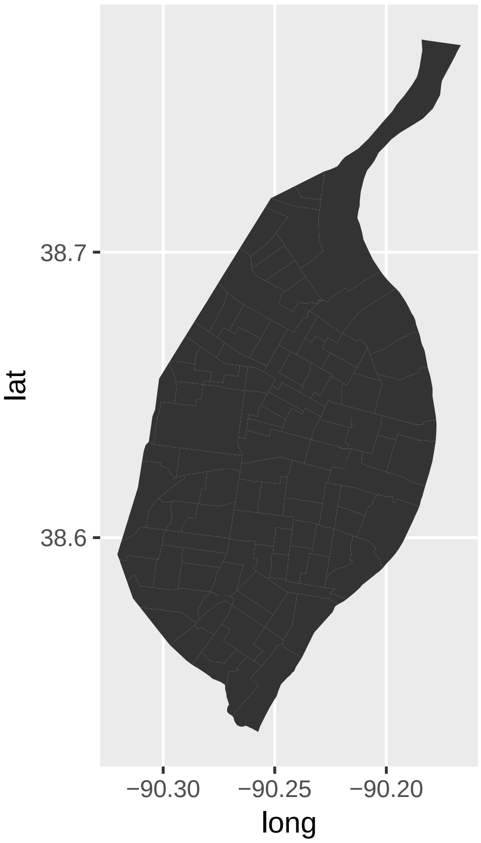
```
## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "STLTracts"
## with 106 features
## It has 12 fields
## Integer64 fields read as strings:  ALAND AWATER
```

```
# convert shapefile to dataframe
tract <- fortify(tract, region="GEOID")
```

You can get a preview of the tract boundaries by using `ggplot2` to map them quickly. The following example also provides a basic example for how to generate maps. A major difference between the following code and a typical ggplot is that the initial `ggplot()` function does not have a data source or asthetic specified. Instead, those details are supplied within `geom_polygon()`. This geom is ideal for mapping polygon spatial data. As shown below, at a minimum you need to specify a data source, the x and y variables (typically `x = long, y = lat`), and a group variable that uniquely identifies individual polygons.

The `coord_map()` function is a tool included with `ggplot2` that gives you access to projected coordinate systems included in the package `mapproj`. If nothing is included in the function, it will use the Mercator projection by default.

```
ggplot() +
  geom_polygon(data = tract, aes(x = long, y = lat, group = id)) +
  coord_map()
```



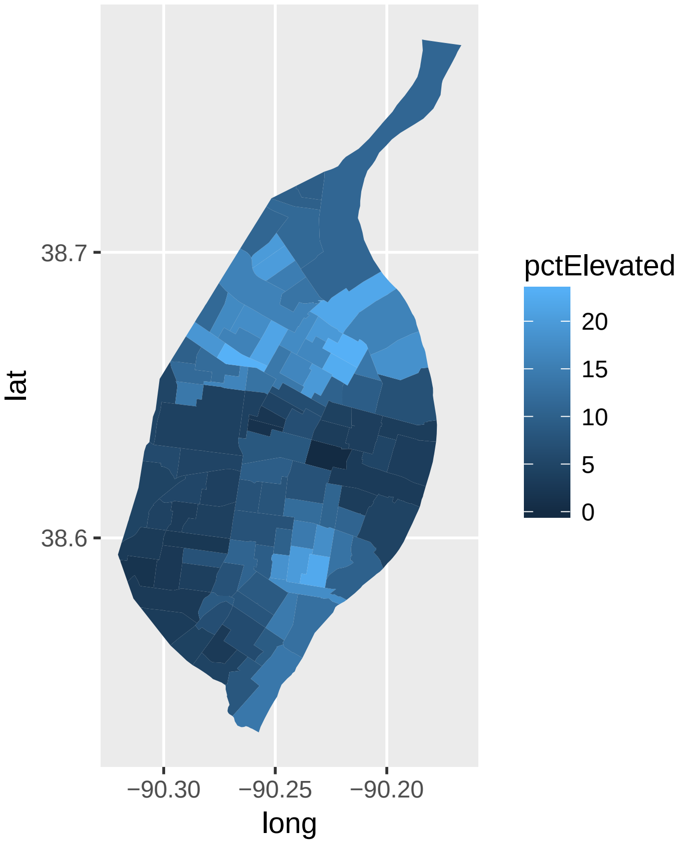
Mapping the Joined the Lead and Tract Data

To map the blood lead level data, we must first join it to the dataframe containing the tract polygons. We can do this using tools from `dplyr`, including the `left_join(dataSource, by "joinVar")` function. The `joinVar` should be identically named and stored in both dataframes. The pipe operator (`%>%`) allows us to join the data and immediately apply the combined dataset to a new dataframe.

```
# join tables into new dataframe
leadTracts <- tract %>%
  left_join(lead, by = "id")
```

With the data joined, we can now map the perctage of children in each Census Tract that have elevated blood lead levels. We add the `fill = variable` option to map the quanities stored in a specific variable:

```
ggplot() +
    geom_polygon(data = leadTracts, aes(x = long, y = lat, group = id, fill = pctElevated)) +
    coord_map()
```



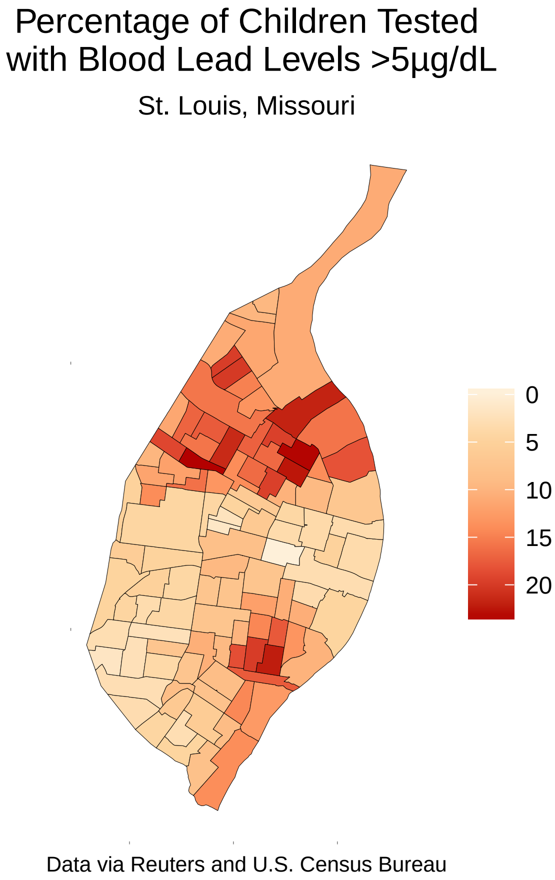
This map has some issues, however. The color ramp is not intuitive - the darkest colors are low values, not high ones. We can use options included in `ggplot2` and `RColorBrewer` to adjust the fill. You can preview all of the available palattes using the `display.brewer.all()`. The `scale_fill_distiller()` is the function for continuous color scales. I've selected the `OrRd` palette and used `trans = "reverse"` to flip the color ramp so that the darkest colors correspond with the highest percentages of children with elevated blood lead levels.

The plot has some other issues as well. The traditional ggplot background is distracting, and the latitude and longitude values are not helpful for a lay viewer. It is also missing helpful labels, like a title, subtitle, and caption. We can use the `theme()` function to adjust the appearance of all of these text elements. `hjust = 0.5` centers each text element. `size = rel(.75)` shrinks the relative size of the caption's text by 25%.

```
# save final plot to object leadPlot
leadPlot <- ggplot() +
    geom_polygon(
      data = leadTracts,
      aes(x = long, y = lat, group = id, fill = pctElevated),
      color = "black", size = 0.05) +
    scale_fill_distiller(palette = "OrRd", trans = "reverse") +
    coord_map() +
    theme_nothing(legend = TRUE) +
    labs(
      title = "Percentage of Children Tested\n with Blood Lead Levels >5µg/dL",
      subtitle = "St. Louis, Missouri",
      fill = "",
      caption = "Data via Reuters and U.S. Census Bureau") +
    theme(
      plot.title = element_text(hjust = 0.5),
      plot.subtitle = element_text(hjust = 0.5),
      plot.caption=element_text(size = rel(.75), hjust=0.5))
```

```
## Warning: `panel.margin` is deprecated. Please use `panel.spacing` property
## instead
```

```
leadPlot # display plot
```


Percentage of Children Tested
with Blood Lead Levels >5μg/dL
St. Louis, Missouri
0
5
10
15
20
Data via Reuters and U.S. Census Bureau

Saving Map Output

You can use the `ggsave()` function to save an already created plot:

```
ggsave(leadPlot, file = "STLLead.png", width = 5, height = 5, type = "cairo-png")
```