Interactive choropleth maps

Andrew Ba Tran

2018-05-27T21:13:14-05:00

Contents

This is from the fifth chapter of learn.r-journalism.com.

Sometimes there are just too many dots on a map. If the point of your map is to know the location of every single data point because of its significance, then fine. Dots can also demonstrate the distribution of data geographically very effectively.

Choropleth or thematic maps are an effective and popular way to show geographic data.

And if it's interactive, it's useful for exploratory purposes because it can surface information that can't be expressed visually easily (Interactive maps for reader purposes need carefully considered, though, because readers will usually not click around).

Making choropleths

To make a choropleth map, you first need a shapefile or geojson of the polygons that you're filling in.

Importing shapefiles from the Census

You could download and import the shapefile into R yourself, but there's a package that brings in Census shapefiles for you called **Tigris**.

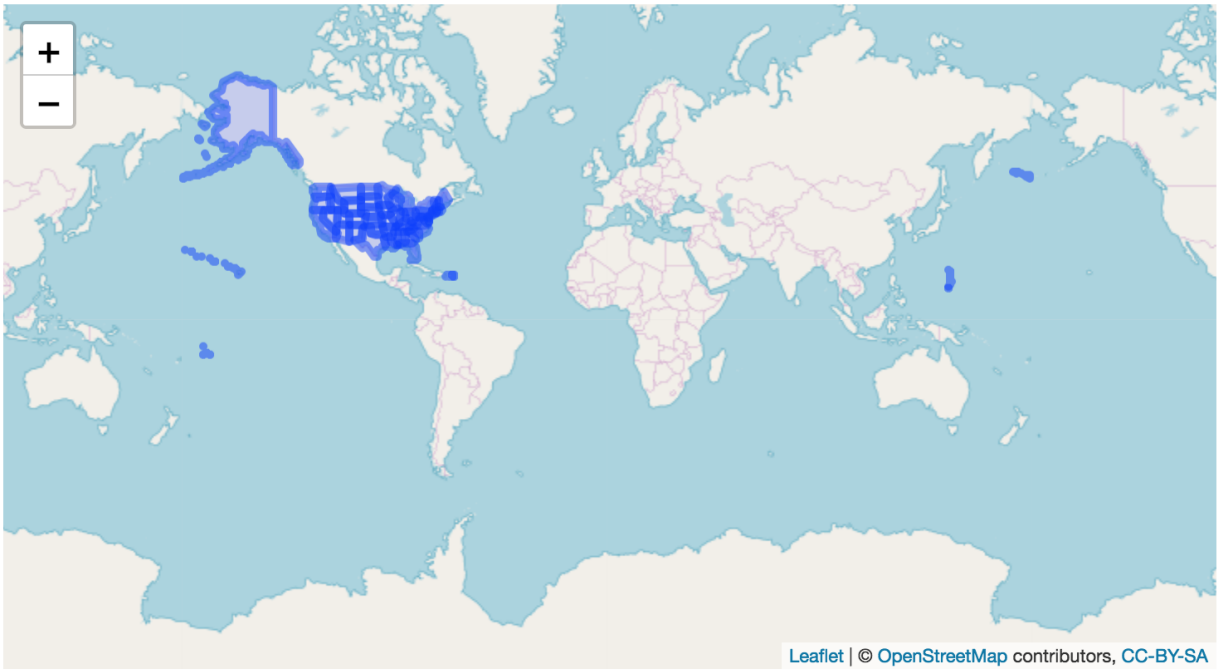
This is what the U.S. states looks like on in leaflet R.

```
# If you don't have the following packages installed yet, uncomment and run the lines below
#install.packages("tigris")
#install.packages("dplyr")
#install.packages("leaflet")

library(tigris)
library(dplyr)
library(leaflet)

# Downloading the shapefiles for states at the lowest resolution
states <- states(cb=T)
```

We can use **dplyr** `%>%` pipes because the **leaflet** package developers incorporated its function structure. We need three functions starting out:

- `leaflet()` - initializes the leaflet function
- `addTiles()` - the underlying map tiles
- `addPolygons()` - instead of dots, we're adding Polygons, or shapes
  - Passing the argument `popup` to the function with the variable *NAME* from the shapefile

```
states %>%
  leaflet() %>%
  addTiles() %>%
  addPolygons(popup=~NAME)
```



This is how it looks raw. The Census shape files also include territories, like Guam and Puerto Rico.

When mapping, we'll have to remember to exclude them if they show up.

Joining data to a shapefile

Let's make a choropleth map based on number of Starbucks per state.

We'll use the same data as the previous section where we mapped every location in Massachusetts.

```
starbucks <- read.csv("data/starbucks.csv", stringsAsFactors=F)

# First, we'll use dplyr to summarize the data
# count by state
sb_state <- starbucks %>%
  group_by(Province) %>%
  summarize(total=n()) %>%
# Some quick adjustments to the the dataframe to clean up names
  mutate(type = "Starbucks") %>%
  rename(state=Province)
```

Alright, we've summarized the the Starbucks data by state and we can now join it to the downloaded data frame.

Tigris includes a useful function called `geo_join()` to bring together shapefiles and dataframes.

We'll also use a **leaflet** function called `colorNumeric()` that will turn the continuous variable of numbers of stores into a categorical variable by dividing it into bins and assigning it a hue of color we want. In this instance: Greens. Because, obviously.

Also setting up a separate array of pop up text for the map.

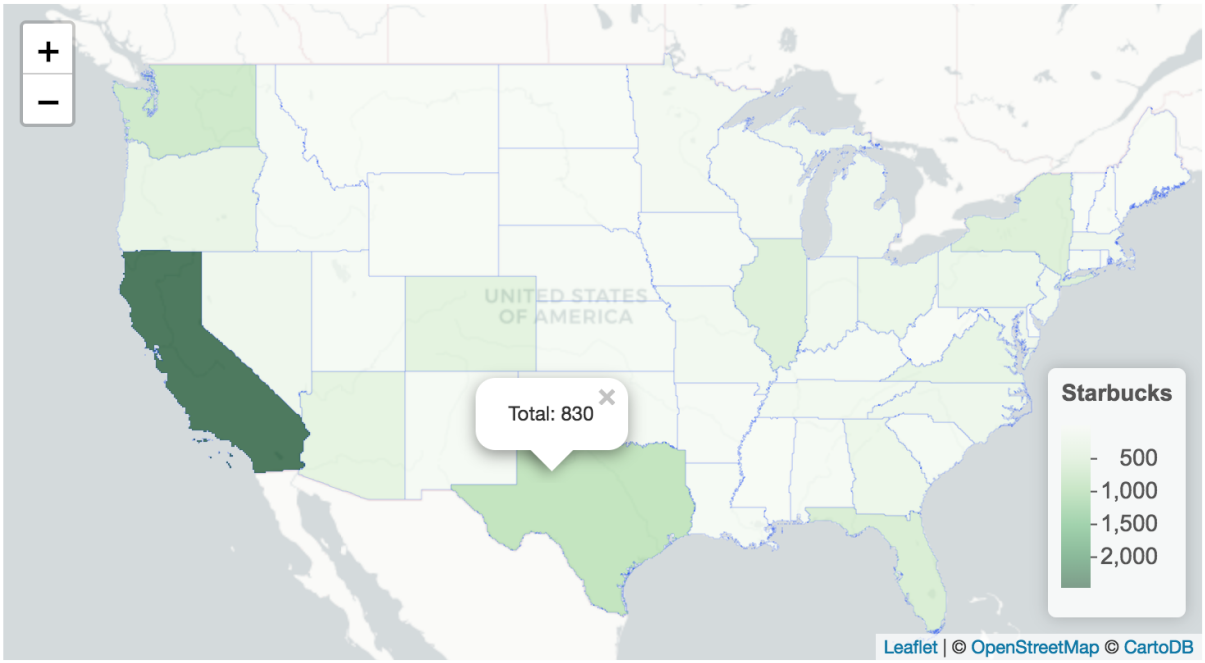
```
# Now we use the Tigris function geo_join to bring together
# the states shapefile and the sb_states dataframe -- STUSPS and state
# are the two columns they'll be joined by

states_merged_sb <- geo_join(states, sb_state, "STUSPS", "state")

# Creating a color palette based on the number range in the total column
pal <- colorNumeric("Greens", domain=states_merged_sb$total)

# Getting rid of rows with NA values
# Using the Base R method of filtering subset() because
# we're dealing with a SpatialPolygonsDataFrame and
# not a normal data frame, thus filter() wouldn't work

states_merged_sb <- subset(states_merged_sb, !is.na(total))

# Setting up the pop up text
popup_sb <- paste0("Total: ", as.character(states_merged_sb$total))
```

Next, the map.

We're adding a few more **leaflet** functions.

- `addProviderTiles()` - instead of `addTiles()`
  - Uses the Leaflet Providers plugin to add different tiles to map
- `setView()` - sets the starting position of the map
  - Centers it on defined coordinates with a specific zoom level
- Lots of arguments passed to `addPolygons()`
  - `fillColor()`
  - `fillOpacity()`
  - `weight`
  - `smoothFactor()`
- `addLegend()` - same as in the previous section but with more customization

```
# Mapping it with the new tiles CartoDB.Positron
leaflet() %>%
  addProviderTiles("CartoDB.Positron") %>%
  setView(-98.483330, 38.712046, zoom = 4) %>%
  addPolygons(data = states_merged_sb ,
              fillColor = ~pal(states_merged_sb$total),
              fillOpacity = 0.7,
              weight = 0.2,
              smoothFactor = 0.2,
              popup = ~popup_sb) %>%
  addLegend(pal = pal,
            values = states_merged_sb$total,
            position = "bottomright",
            title = "Starbucks")
```



Hmm... Not that interesting, right?

What's the problem here. *Deep down, you know what's wrong.*

This is essentially a population map.

So we need to adjust for population.

And that's easy to do using the Census API.

Bringing in Census data via API

We'll use the censusapi package created by journalist Hannah Recht.

```
# If you don't have censusapi installed yet, uncomment the line below and run
#install.packages("censusapi")

library(censusapi)
```

Don't forget your census key.

```
# Add key to .Renviron
Sys.setenv(CENSUS_KEY="YOURKEYHERE")
# Reload .Renviron
readRenviron("~/.Renviron")
# Check to see that the expected key is output in your R console
Sys.getenv("CENSUS_KEY")
```

We've already gone in some detail into how to use this package in th eprevious section.

But let's get the latest set of information on population from the ACS 5-year 2015 Census.

```
# Alright, getting total population by state from the API
state_pop <-  getCensus(name="acs5",
                        vintage=2015,
```

```
                        key=census_key,
                        vars=c("NAME", "B01003_001E"),
                        region="state:*")

head(state_pop)
```

```
##   state       NAME B01003_001E
## 1    02     Alaska      733375
## 2    01    Alabama     4830620
## 3    05   Arkansas     2958208
## 4    04    Arizona     6641928
## 5    06 California    38421464
## 6    08   Colorado     5278906
```

We now have the data set to be paired with the shapefile.

We need to prep them so they can join without issue.

Did you notice that the names of states in the downloaded data frame are fully spelled out? In the shapefile, states are abbreviated.

We need to deal with that. Did you know that R has arrays of state names and state abbreviations just built in? Just type `state.name` or `state.abb` and they just appear! That's pretty cool. We can build a dataframe of that and use that as a relationship file between the population data set and the shape file.

```
# Cleaning up the column names
colnames(state_pop) <- c("state_id", "NAME", "population")
state_pop$state_id <- as.numeric(state_pop$state_id)
# Hm, data comes in numbers of fully spelled out, not abbreviations

# Did you know R has its own built in list of State names and State abbreviations?
# Just pull it in this way to create a dataframe for reference

state_off <- data.frame(state.abb, state.name)
head(state_off)
```

```
##   state.abb state.name
## 1        AL    Alabama
## 2        AK     Alaska
## 3        AZ    Arizona
## 4        AR   Arkansas
## 5        CA California
## 6        CO   Colorado
```

```
# So I needed to create the dataframe above because the Census API data
# gave me states with full names while the Starbucks data came with abbreviated state names
# So I needed a relationship dataframe so I could join the two

# Cleaning up the names for easier joining
colnames(state_off) <- c("state", "NAME")

# Joining state population dataframe to relationship file
state_pop <- left_join(state_pop, state_off)

# The relationship dataframe didnt have DC or Puerto Rico, so I'm manually putting those in
state_pop$state <- ifelse(state_pop$NAME=="District of Columbia", "DC", as.character(state_pop$state))
state_pop$state <- ifelse(state_pop$NAME=="Puerto Rico", "PR", as.character(state_pop$state))
```

```r
# Joining Starbucks dataframe to adjusted state population dataframe
sb_state_pop <- left_join(sb_state, state_pop)

# Calculating per Starbucks stores 100,000 residents and rounding to 2 digits
sb_state_pop$per_capita <- round(sb_state_pop$total/sb_state_pop$population*100000,2)

# Eliminating rows with NA
sb_state_pop <- filter(sb_state_pop, !is.na(per_capita))
head(sb_state_pop)
```

```
## # A tibble: 6 x 7
##   state total type      state_id NAME       population per_capita
##   <chr> <int> <chr>        <dbl> <chr>           <dbl>      <dbl>
## 1 AK       42 Starbucks        2 Alaska         733375       5.73
## 2 AL       65 Starbucks        1 Alabama       4830620       1.35
## 3 AR       37 Starbucks        5 Arkansas      2958208       1.25
## 4 AZ      391 Starbucks        4 Arizona       6641928       5.89
## 5 CA     2456 Starbucks        6 California   38421464       6.39
## 6 CO      421 Starbucks        8 Colorado      5278906       7.98
```

Final map I

We're going to use the same process code as the previous map but with the `per_capita` variable we've created. Reproducibility!

We'll be changing the pop up so it shows more data. It can take in HTML, so we'll pass an HTML string based on state name, total, and per capita data.

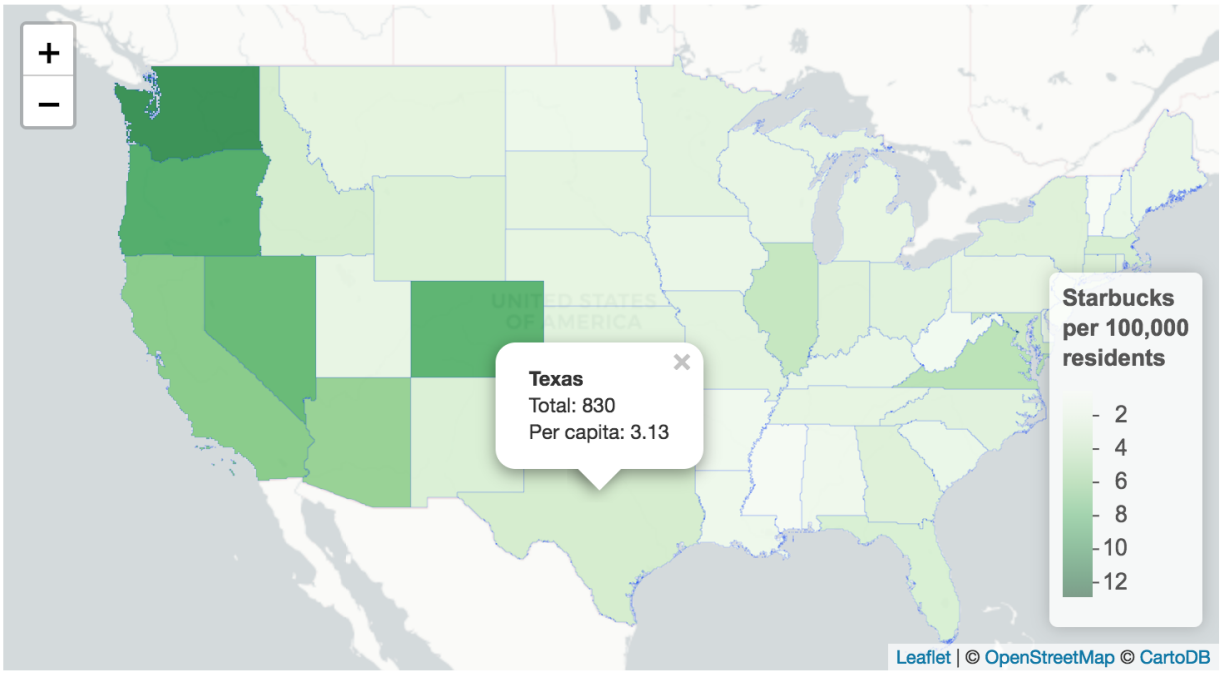
```r
states_merged_sb_pc <- geo_join(states, sb_state_pop, "STUSPS", "state")

pal_sb <- colorNumeric("Greens", domain=states_merged_sb_pc$per_capita)
states_merged_sb_pc <- subset(states_merged_sb_pc, !is.na(per_capita))

# Here's the pop up
popup_sb <- paste0("<strong>", states_merged_sb_pc$NAME,
                   "</strong><br />Total: ", states_merged_sb_pc$total,
                   "<br />Per capita: ",
                   as.character(states_merged_sb_pc$per_capita))
head(popup_sb)
```

```
## [1] "<strong>Alabama</strong><br />Total: 65<br />Per capita: 1.35"
## [2] "<strong>Alaska</strong><br />Total: 42<br />Per capita: 5.73"
## [3] "<strong>Arizona</strong><br />Total: 391<br />Per capita: 5.89"
## [4] "<strong>Arkansas</strong><br />Total: 37<br />Per capita: 1.25"
## [5] "<strong>California</strong><br />Total: 2456<br />Per capita: 6.39"
## [6] "<strong>Colorado</strong><br />Total: 421<br />Per capita: 7.98"
```

And now the map.

```r
leaflet() %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  setView(-98.483330, 38.712046, zoom = 4) %>%
  addPolygons(data = states_merged_sb_pc ,
              fillColor = ~pal_sb(states_merged_sb_pc$per_capita),
              fillOpacity = 0.9,
```

```
              weight = 0.2,
              smoothFactor = 0.2,
              popup = ~popup_sb) %>%
  addLegend(pal = pal_sb,
            values = states_merged_sb_pc$per_capita,
            position = "bottomright",
            title = "Starbucks<br />per 100,000<br/>residents")
```



Pretty nice. An easy way to display data one way and offer more data after a click.

Final map II

You can also set it up so the choropleth maps surface data on hover.

We use the `highlight()` and `labelOptions()` as sub functions to `addPolygons()`.

Read more at rstudio.

The downside is that it hover pop ups don't display HTML, so it limits the strings passed to it.

```
popup_sb <- paste0("Per capita: ", as.character(states_merged_sb_pc$per_capita))

leaflet() %>%
  addProviderTiles(providers$CartoDB.Positron) %>%
  setView(-98.483330, 38.712046, zoom = 4) %>%
  addPolygons(data = states_merged_sb_pc ,
              fillColor = ~pal_sb(states_merged_sb_pc$per_capita),
              fillOpacity = 0.9,
              weight = 0.2,
              smoothFactor = 0.2,
              highlight = highlightOptions(
```

```
                  weight = 5,
                  color = "#666",
                  dashArray = "",
                  fillOpacity = 0.7,
                   bringToFront = TRUE),
              label=popup_sb,
              labelOptions = labelOptions(
  style = list("font-weight" = "normal", padding = "3px 8px"),
  textsize = "15px",
  direction = "auto")) %>%
addLegend(pal = pal_sb,
          values = states_merged_sb_pc$per_capita,
          position = "bottomright",
          title = "Starbucks<br />per 100,000<br/>residents")
```