Week 11: Geography as Data

Ben Schmidt

4/8/2015

For our final week, we'll be considering *mapping* elements.

Computerized treatments of geography have so extensive a provenance that they've spurned their own class of software–Geographic Information Systems (GIS).

```
#install.packages("ggmap")
library(ggmap)
library(maptools)
library(gpclib)
library(sp)
```

We'll start by downloading a simple file from the census: the United States.

This is a complicated type of object–don't look it if your computer is short on memory.

Like many R objects, you can run the function `plot` on it to get an idea of what's in it.

You can also use `as.data.frame`: here, that will pull out the metadata about each state in the set.

```
plot(states)
stateMetadata = states %>% as.data.frame
stateMetadata$id=rownames(stateMetadata)
```

To plot shapefile data in R, we use the function `fortify` from ggplot to create a new object

```
polygons = fortify(states)
head(polygons)
ggplot(polygons) + geom_polygon(aes(x=long,y=lat,group=group,fill=id))
```

This is ugly! For lots of reasons. What makes a map of the USA look decent?

1. First, we'll simply drop out Alaska, Hawaii, and Puerto Rico by performing a join on the metadata we already created.
2. Then, in the plot, instead of "x" and "y" we'll use "coord_map" to indicate we're using a map *projection*. The standard projection for the lower 48 is the Albers projection.
3. Finally, we'll use a "black and white" theme from ggplot–the grey background is great for plots, but looks a little strange on maps.

```
polygons %>%
  inner_join(stateMetadata) %>%
  filter(!STUSPS%in% c("AK","HI","PR")) %>%
ggplot() +
  geom_polygon(aes(x=long,y=lat,group=group,color=id),fill="white",alpha=.4) +
  coord_map(proj="albers",y0=29,y1=45) + theme_bw()
```

This isn't yet a thematic map of any sort. But we can add in some information to make it better.

You can save a ggplot object as a base and then add more layers to it. That's what we'll do here, adding on top some of the largest cities.

```
baseplot = polygons %>%
  inner_join(stateMetadata) %>%
  filter(!STUSPS%in% c("AK","HI","PR")) %>%
ggplot() +
  geom_polygon(aes(x=long,y=lat,group=group),fill="white",alpha=.4,color="grey") +
  coord_map(proj="albers",y0=29,y1=45) + theme_bw()

cities = read.csv("data/CESTACityData.csv")

bigCities = cities %>% filter(X2010>500000)

baseplot + geom_point(data=bigCities,aes(x=LON,y=LAT)) + labs(title="All cities over 500,000 people") +
```

Good luck getting Dallas and Fort Worth not to overlap–that's one of the things that tends to be very hard in a framework like this.

There are other obvious approaches to take: for example, plotting the cities that lost the most population between 1910 and 2010.

```
shrinkingCities = cities %>% filter(as.numeric(X2010)<as.numeric(X1910)) %>% ungroup %>% arrange(X2010-

baseplot + geom_point(data=shrinkingCities,aes(x=LON,y=LAT,size=X1910-X2010)) + labs(title="The twelve
```

Again, this is obviously bad: you might want to, limit the area of the plot. You can do this by specifying the xlim and ylim parameters in `coord_map`.

```
shrinkingCities = cities %>% filter(as.numeric(X2010)<as.numeric(X1910)) %>% ungroup %>% arrange(X2010-

baseplot + geom_point(data=shrinkingCities,aes(x=LON,y=LAT,size=X1910-X2010)) + labs(title="The twelve
```

As a rule, making a map for publication or slide this way will be an iterative process–you'll notice something wrong, and then figure out how best to fix it.

This might be useful for a slide, but isn't really an exploratory visualization of any important sort. Maybe more useful would be to plot change for every city, without labels, to see if there are geographical patterns. (Of course, you know there are. . .)

```
populationChange = cities %>% mutate(change=X2010-X1950)

baseplot + geom_point(data=populationChange %>% arrange(abs(change)),aes(x=LON,y=LAT,color=change>0,size
```

This same process can apply to any shapefiles you can find. The census has railway lines, for example.

Perhaps the most frequently used form of maps after simple points in space is the **choropleth** map. In this case, we want to actually *join together* data from two different sets: make the states, for example, be colorized by their change in urban populations.

```
# Here's what the data looks like: 2010-1950, divided by the 1950 population. And then multiplied by 10
cities %>% group_by(ST) %>% summarize(urbanChange=
                                        100*(
                                          sum(X2010,na.rm=T)-sum(X1950,na.rm=T))/
                                        sum(X1950,na.rm=T))

# Here's we can join it in
popChanges = cities %>% group_by(ST) %>% summarize(urbanChange=
                                        100*(
                                          sum(X2010,na.rm=T)-sum(X1950,na.rm=T))/
                                        sum(X1950,na.rm=T))


#Two lines from earlier
polygons %>%
  inner_join(stateMetadata) %>%
  filter(!STUSPS%in% c("AK","HI","PR")) %>%
# Here's the new line
  inner_join(popChanges,by = c("STUSPS"="ST")) %>%
ggplot() +
  geom_polygon(aes(x=long,y=lat,group=group,fill=urbanChange)) +
  coord_map(proj="albers",y0=29,y1=45) + theme_bw() + scale_fill_gradientn(colours=c("red","white","blu
```

Geocoding.

One last, important feature is **geocoding.** There are many different ways to geocode.

One easy one is to use the `geocode` function in ggmap, the ggplot package for maps.

Note that you must technically abide by the Google Maps Terms of Service to use the data, which is by my reading extraordinarily restrictive: at least one of the authors this week seems to me to have violated it!

```
library(ggmap)
crews = read.csv("data/CrewlistCleaned.csv")

locations = crews %>% group_by(Residence) %>% summarize(count=n()) %>% arrange(-count) %>% head(30)

locations = locations %>% filter(Residence!="") %>% group_by(Residence,count) %>% do(.$Residence %>% as

ggplot(crews %>% inner_join(locations)) + geom_point(aes(x=lon,y=lat))
```

Once again, the lack of context is devasting here: so let's download a map of the world. Free from the contraints of Google's copyrights, for the time being the best world map is from Natural Earth data online.

```
#download.file("http://www.naturalearthdata.com/http//www.naturalearthdata.com/download/110m/physical/n
#unzip("coastline.zip")
#coasts = "ne_110m_coastline.shp" %>% readShapeSpatial() %>% fortify

ggplot(coasts) + geom_path(aes(x=long,y=lat,group=group),fill=NA) + geom_point(data = locations, aes(x=
```

Named Entity Extraction

We're not going to look at the process of named entity extraction, but it resembles topic modeling in that doing it in R requires deploying external Java libraries in R.

Lincoln Mullen has a guide online for performing these actions.; go down to "named entity extraction." It will require you to install a few extra packages.