Mapping in R

Alton Barbehenn

December 3, 2015

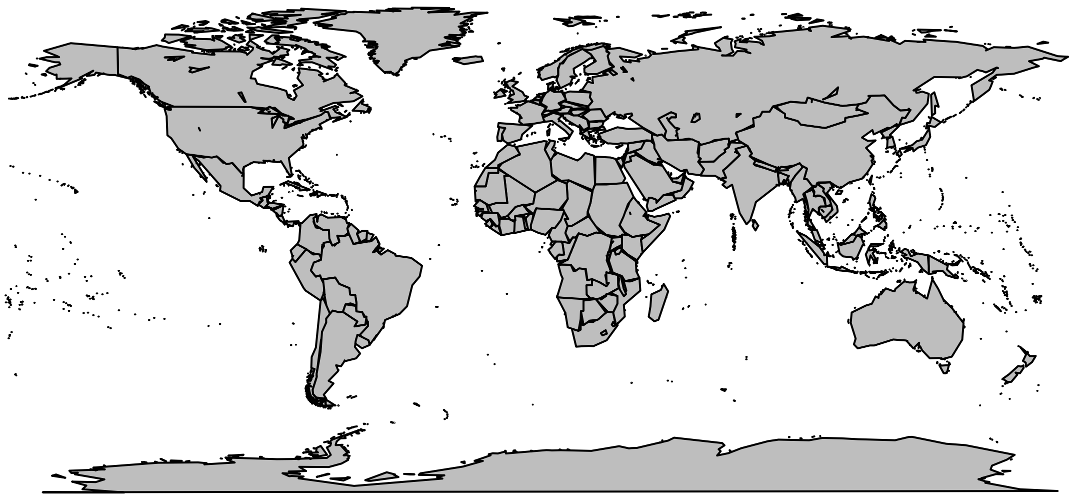
Basic Mapping

The basic mapping package that we can use is `maps`. This package allow us to make some basic plots that have geographical coordinates, making them easy to map onto. The most basic map we can produce is the world with the countries outlined.

```
require(maps)
```

```
## Loading required package: maps
```

```
map('world', xlim = c(), ylim = c(), fill = TRUE, col = "grey")
```



There are many optional parameters to utilize, the main ones are shown. If we add limits we can narrow the range of the map down to a box. For example, `xlim = c(-175,-35), ylim = c(-70,90)` gives you a box that displays most of the Americas. Note that this is a Mercator Projection which means the relative areas on the map are not to scale.

Another package you can use is `dismo`. This package allows us to query Google maps easily and so we can produce more complex maps. It is important to note that the maps produced in dismo are also Mercator Projections, but this can be fixed with the parameter `lonlat=TRUE`. The main advantage to mapping with `dismo` is that you can query Google Maps almost as freely as when using Google's online interface; in the first field you can enter a textual locality description or an object with lon/lat coordinates. However, in order to plot onto a map produced using this package, there is a slightly cumbersome process to convert longitude and latitude points into the correct projection (more on this later). To produce a basic map in this package we use a slightly different procedure:

```
require(dismo)
```

```
## Loading required package: dismo
## Loading required package: raster
## Loading required package: sp
```

```r
mymap <- gmap("Germany", type = "hybrid", zoom = 6, size=c(640, 640))
```

```
## Loading required namespace: rgdal
## Loading required namespace: XML
```

```r
plot(mymap)
```

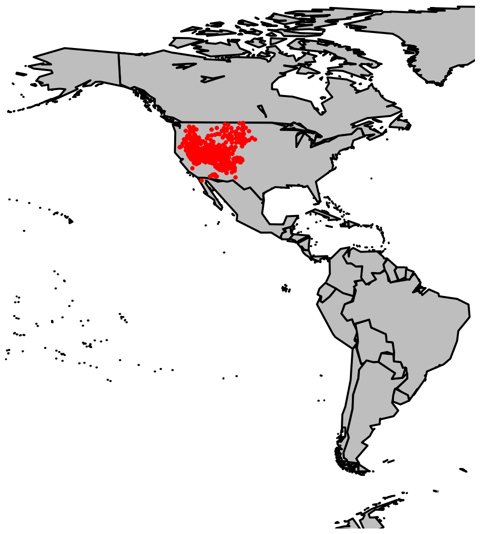
Fyn
Kiel
Rostock
Hamburg
Szczecin
Bremen
Berlin
Hanover
Potsdam
Brunswick
erlands
Dortmund
Essen
Leipzig
Düsseldorf
Germany
Dresden
Cologne
Frankfurt
Prague
xembourg
Czech Re
Mannheim
Nuremberg
Stuttgart
Strasbourg
Augsburg
Munich
Salzburg
Basel
Zürich
Austria
Innsbruck
Liechtenstein
Switzerland
Map data ©2015 GeoBasis-DE/BKG (©2009), Google Imagery ©2015 TerraMetrics

This code chunk demonstrates the main parameters of gmap. You can write any country or geographic location, for example "Lake Michigan" or "world", in the first parameter, capitalization of first letter does not matter. The second parameter is the type of map; this defaults to political map, however, you can choose from these formats as well 'roadmap', 'satellite', 'hybrid', and 'terrain'. Finally, zoom is an integer from 0 to 21 that specifies how zoomed in you would like your map (1 being the whole world).

Plotting Points (and Shapes)

To plot points on either of these map types we need to use the `points` function in the `graphics` package. This function adds points to the map using longitude and latitude as coordinates. This can be used in conjunction with either mapping tool described above. I will be using a data frame derived earlier, `df`, containing a three columns: latitude, longitude, and color. Note that the pch and cex refer to the shape and size of each data point respectively. Examples:

```r
require(maps)
map('world', xlim = c(-175,-35), ylim = c(-70,90), fill = TRUE, col = "grey")
points(df$longitude, df$latitude, pch=20, col="red", cex=0.25)
```



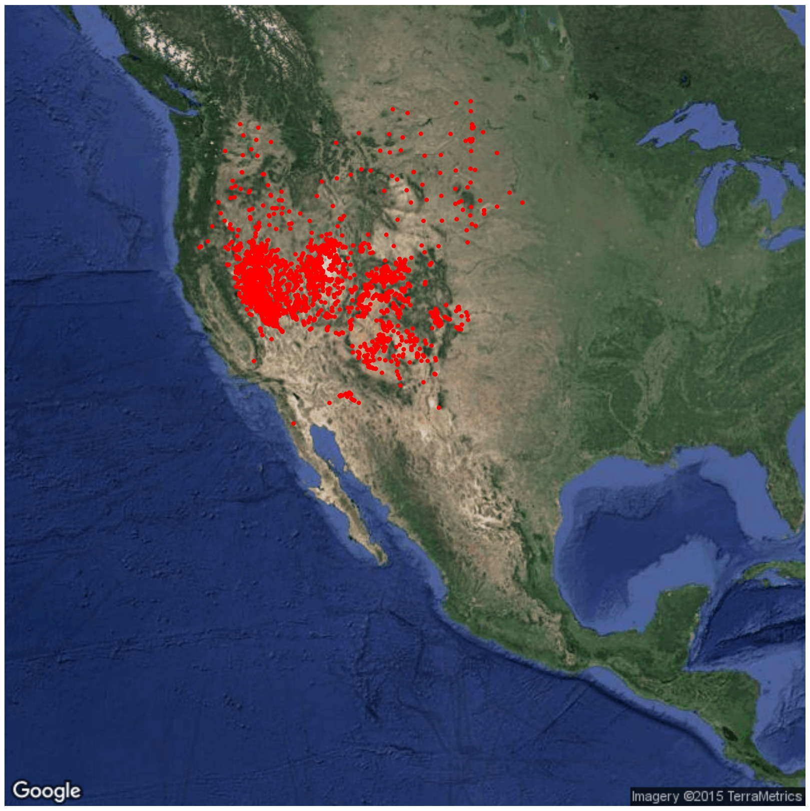
This also works with maps generated in with the `dismo` package, however, we need to convert our coordinates in `df` into a usable form.

In order to plot shapes, such as range polygons, you can simply plot the shape with the parameter `add=TRUE`. If we need to adjust the projection to plot points onto the map, you also need to adjust the projection of the shape using the `proj4string()`, `CRS()`, and `spTransform()` functions as applicable.

```r
require(dismo)
require(sp)
mymap <- gmap("Central North America", type = "satellite", size=c(640, 640), zoom = 4)
plot(mymap)
```

```
#correct the projection of points
#(details can be found in the CRAN r-project dismo package under gmap)
coordinates(df) <- ~longitude + latitude
proj4string(df) <-"+proj=longlat +datum=WGS84 +ellps=WGS84"
pts <- spTransform(df, CRS("+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0
                           +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +no_defs"))

#plot points onto map
points(pts, col='red', pch = 20, cex=.25)
```



Google Maps

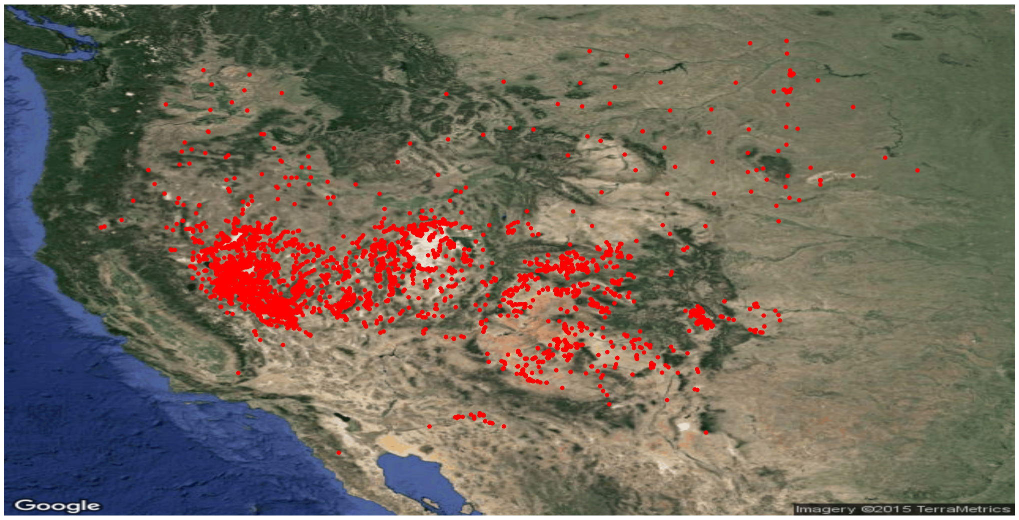
As you may have noticed, when you use the `dismo` package, you are querying Google Maps. We can do this directly using the `RgoogleMaps` package. For example:

```
require(RgoogleMaps)
```

```
## Loading required package: RgoogleMaps
```

```
#get bounds of plot
lon_range <- c(min(df$longitude), max(df$longitude))
lat_range <- c(min(df$latitude), max(df$latitude))

#get plot
mymap <- GetMap.bbox(lon_range, lat_range, destfile = "TestGoogleMap.png",
                     maptype="satellite", size=c(640,640))
PlotOnStaticMap(mymap, lon=df$longitude, lat=df$latitude, pch=20, cex = .25, col=df$color)
```



Note that when using `RgoogleMaps` you don't need to make the transformation of coordinate system that we used with the `dismo` maps.

When using `GetMap.bbox()` you have the choice to either use latitude and longitude range vectors, a center location, a zoom level, or some combination of these to fix the location you wish to retrieve. Using the `PlotOnStaticMap` function in this package, we plot the Google map and any points we want in the normal way. We can add a legend to make things look pretty using the `legend()` function in the package `graphics`.

There is a lot more examples of different mapping tools in R found at https://github.com/Pakillo/R-GIS-tutorial/blob/master/R-GIS_tutorial.md.