Convert .shp file for Tableau or R

Laura Hughes

April 20, 2016

Contents

Converting an ESRI Shapefile into a format for Tableau or R

To be able to plot polygons within Tableau or R, you have to convert a shapefile into latitude and longitude coordinates.

Fortunately, there are some nifty packages within R that allow you to do that very easily. To learn more about adding polygon data to plots in R and Tableau, see:

- Useful help document on plotting shapefiles in R using ggplot2
- Useful help document on plotting polygons in Tableau

In this overview, I'll walk you through the process of manipulating a shapefile for use in R and Tableau. The steps are:

1. Import R packages to deal with spatial data
2. Load in an ESRI shapefile
3. Reproject the data
4. Manipulate the shapefile to pull out the needed info
5. Check that it looks right by plotting
6. Save the output as a .csv file so you can use it later.

1. Import necessary R packages. This includes:

- **dplyr**: general functions for manipulating data. My second favorite package in R.
- **rgdal**: functions to import and manipulate spatial data
- **maptools**: functions to import and manipulate spatial data
- **rgeos**: functions to manipulate spatial data
- **ggplot**: contains function to convert shapefiles into lat/lon coordinates. Also an amazingly powerful plotting package for R, and my favourite R package.

- **RColorBrewer**: library of color palettes developed by Cynthia Brewer

```
pkgs = c('dplyr', 'rgdal', 'maptools', 'rgeos', 'ggplot2', 'RColorBrewer')

# Check if the packages have been installed.`
alreadyInstalled = installed.packages()[,'Package']

toInstall = pkgs[!pkgs %in% alreadyInstalled]

# Install anything that isn't already installed.
if(length(toInstall > 0)) {
  print(paste0('Installing these packages: ', toInstall))

  install.packages(toInstall)
}

library(rgdal)
```

```
## Loading required package: sp

## rgdal: version: 1.1-8, (SVN revision 616)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.0.2, released 2016/01/26
##  Path to GDAL shared files: /opt/local/share/gdal
##  Loaded PROJ.4 runtime: Rel. 4.9.2, 08 September 2015, [PJ_VERSION: 492]
##  Path to PROJ.4 shared files: (autodetected)
##  Linking to sp version: 1.2-2
```

```
library(maptools)
```

```
## Checking rgeos availability: TRUE
```

```
library(rgeos)
```

```
## rgeos version: 0.3-19, (SVN revision 524)
##  GEOS runtime version: 3.5.0-CAPI-1.9.0 r4084
##  Linking to sp version: 1.2-2
##  Polygon checking: TRUE
```

```
library(dplyr)
```

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

## The following objects are masked from 'package:rgeos':
##
##     intersect, setdiff, union

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

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

```
library(ggplot2)
library(RColorBrewer)
```

2. Import your data

In this example, we're going to pull out the data for the Administrative 1 units for Cambodia and a layer containing all the water bodies.

```
# --- CAMBODIA ADM1 ---
# Set your working directory to the folder that contains your data
setwd('~/Documents/USAID/mini projects/tableaupolygonsfromshapefiles/khm_admbnda_adm1_gov')

# the dsn argument of '.' says to look for the layer in the current directory.
rawAdm1 = rgdal::readOGR(dsn=".", layer="khm_admbnda_adm1_gov")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "khm_admbnda_adm1_gov"
## with 25 features
## It has 6 fields
```

```
# --- CAMBODIA BODIES OF WATER ---
setwd('~/Documents/USAID/mini projects/tableaupolygonsfromshapefiles/khm_WatrcrsA_wfp/')

# the dsn argument of '.' says to look for the layer in the current directory.
rawLakes = rgdal::readOGR(dsn=".", layer="khm_WatrcrsA_wfp")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "khm_WatrcrsA_wfp"
## with 875 features
## It has 11 fields
```

Shapefiles have five components:

- **data**: a table containing the associated data with the polygons, so things like the name of the polygons and any data you've added to the file.
- **polygons**: a list of the coordinates of the polygons
- **plotOrder**: the order in which to plot the polygons. If your polygons overlap each other, obviously becomes very important.
- **bbox**: the bounding box for the data (minimum and maximum lat and lon of the data)
- **proj4string**: the type of projection for the data. This'll be important.

3. The obnoxious part: reprojecting the data

So... sadly, before we pull out the lat/lon coordiantes from the files, there's a bit more work that needs to be done, unless you've set things up in ArcGIS before hand. R and ArcGIS can deal with any projection system/units, but Tableau is more limited. Tableau needs the lat/lon coordinates to be in a **geographic coordinate system using decimal latitude and longitude units, such as GCS NAD 1983.**

Projections are complicated, but a for a decent overview see here.

First, let's check the projection of our data.

```
is.projected(rawAdm1)
```

```
## [1] TRUE
```

```
summary(rawAdm1)
```

```
## Object of class SpatialPolygonsDataFrame
## Coordinates:
##         min     max
## x  211837.2  785034
## y 1144269.1 1625281
## Is projected: TRUE
## proj4string :
## [+proj=utm +zone=48 +a=6377276.345 +b=6356075.41314024 +units=m
## +no_defs]
## Data attributes:
##     OBJECTID         Level                HRName      HRPCode   HRParent
##  Min.   : 1.00   Min.   :1   Banteay Meanchey: 1   KH01   : 1   KH:25
##  1st Qu.: 7.00   1st Qu.:1   Battambang       : 1   KH02   : 1
##  Median :13.00   Median :1   Kampong Cham     : 1   KH03   : 1
##  Mean   :12.72   Mean   :1   Kampong Chhnang : 1   KH04   : 1
##  3rd Qu.:18.00   3rd Qu.:1   Kampong Speu     : 1   KH05   : 1
##  Max.   :24.00   Max.   :1   Kampong Thom     : 1   KH06   : 1
##                              (Other)          :19   (Other):19
##     PRO_CODE
##  Min.   : 1
##  1st Qu.: 7
##  Median :13
##  Mean   :13
##  3rd Qu.:19
##  Max.   :25
##
```

```
is.projected(rawLakes)
```

```
## [1] FALSE
```

```
summary(rawLakes)
```

```
## Object of class SpatialPolygonsDataFrame
## Coordinates:
##         min       max
## x 102.47064 107.61803
## y  10.37245  14.59647
## Is projected: FALSE
## proj4string :
## [+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0]
## Data attributes:
##       AREA             PERIMETER             CODE         FCODE
```

```
##  Min.   :0.000e+00   Min.   :       4   Min.   :51.00   BH140:875
##  1st Qu.:1.811e+04   1st Qu.:     947   1st Qu.:51.00
##  Median :5.530e+04   Median :    2157   Median :51.00
##  Mean   :4.981e+06   Mean   :   25917   Mean   :51.26
##  3rd Qu.:1.650e+05   3rd Qu.:    6273   3rd Qu.:52.00
##  Max.   :3.971e+09   Max.   :13342023   Max.   :52.00
##       HYC             WSC     NAM          ACRES
##  Min.   :6.000   Min.   :2   UNK:875   Min.   :     0.0
##  1st Qu.:6.000   1st Qu.:2            1st Qu.:     4.5
##  Median :8.000   Median :2            Median :    13.7
##  Mean   :7.477   Mean   :2            Mean   :  1230.7
##  3rd Qu.:8.000   3rd Qu.:2            3rd Qu.:    40.8
##  Max.   :8.000   Max.   :2            Max.   :981170.7
##     HECTARES          Shape_Leng         Shape_Area
##  Min.   :     0.0   Min.   :  0.00004   Min.   :0.0000000
##  1st Qu.:     1.8   1st Qu.:  0.00862   1st Qu.:0.0000015
##  Median :     5.5   Median :  0.01962   Median :0.0000046
##  Mean   :   498.1   Mean   :  0.23627   Mean   :0.0004148
##  3rd Qu.:    16.5   3rd Qu.:  0.05739   3rd Qu.:0.0000137
##  Max.   :397067.3   Max.   :121.66989   Max.   :0.3307649
```

rawAdm1 is projected ('Is projected: TRUE') and is in units of 'm'. This won't work for Tableau, so we will fix this using the `rgdal` package and transform it into a geographic coordinate system.

```
adm1_decimal = spTransform(rawAdm1, CRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,

lakes_decimal = spTransform(rawLakes, CRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,
```

4. Converting polygon points to lat/lon coordinates.

Now that is settled, we can pull out the rest of the data we need: the polygon coordinates, which we will merge to the original underlying data.

```
# --- CAMBODIA ---
# pull out the row names from the data and save it as a new column called 'id'
adm1_decimal@data$id = rownames(adm1_decimal@data)

# Convert the shape polygons into a series of lat/lon coordinates.
adm1_points = ggplot2::fortify(adm1_decimal, region="id")

# Merge the polygon lat/lon points with the original data
adm1_df = dplyr::left_join(adm1_points, adm1_decimal@data, by="id")

# --- LAKES ---
# pull out the row names from the data and save it as a new column called 'id'
lakes_decimal@data$id = rownames(lakes_decimal@data)


lakes_points = ggplot2::fortify(lakes_decimal, region="id")
lakes_df = dplyr::left_join(lakes_points, lakes_decimal@data, by="id")
```

Now our data is in a nice tabular format with `long` and `lat` as separate columns:

```
knitr::kable(head(adm1_df))
```

long	lat	order	hole	piece	id	group	OBJECTID	Level	HRName	HRPCode	HRP
104.6436	12.60018	1	FALSE	1	0	0.1	4	1	Kampong Chhnang	KH04	KH
104.6454	12.59949	2	FALSE	1	0	0.1	4	1	Kampong Chhnang	KH04	KH
104.6478	12.59843	3	FALSE	1	0	0.1	4	1	Kampong Chhnang	KH04	KH
104.6510	12.59689	4	FALSE	1	0	0.1	4	1	Kampong Chhnang	KH04	KH
104.6533	12.59597	5	FALSE	1	0	0.1	4	1	Kampong Chhnang	KH04	KH
104.6548	12.59483	6	FALSE	1	0	0.1	4	1	Kampong Chhnang	KH04	KH

```
knitr::kable(head(lakes_df))
```

long	lat	order	hole	piece	id	group	AREA	PERIMETER	CODE	FCODE	HYC	WSC
106.5021	14.56412	1	FALSE	1	0	0.1	22673.15	1557.019	51	BH140	8	2
106.5018	14.56448	2	FALSE	1	0	0.1	22673.15	1557.019	51	BH140	8	2
106.5019	14.56467	3	FALSE	1	0	0.1	22673.15	1557.019	51	BH140	8	2
106.5024	14.56428	4	FALSE	1	0	0.1	22673.15	1557.019	51	BH140	8	2
106.5035	14.56365	5	FALSE	1	0	0.1	22673.15	1557.019	51	BH140	8	2
106.5043	14.56326	6	FALSE	1	0	0.1	22673.15	1557.019	51	BH140	8	2

Pulling out the names of each of the polygons.

We also might want to label each polygon with its name. That data is located in two places: the name is within the @data part of the raw shapefile, and we can use the function `coordinates` to find out the centroid of each polygon.

```
# Find the coordinates of the centroids, put it into a data frame, and rename the names of the columns.
adm1_centroids = data.frame(coordinates(adm1_decimal)) %>%
  rename(long = X1, lat = X2)

adm1_centroids = cbind(adm1_centroids,
                       HRName = adm1_decimal@data$HRName) # The name of the province is in the HRName v
```

Adding extra data to the dataframe

You might have additional data that you want to bind to your dataframe, for instance to create a choropleth where each polygon is colored by another value. For example, you might want to plot the malaria incidence rate of each of the 25 provinces. You can easily merge data together using the function `left_join`. I don't have real data, so I'll make some fake data.

To merge the data, you'll need to have a common ID. In this case, I'll use the names of the provinces, which is located in the raw shapefile in the variable `HRName`.

```
# Extract the names of the provinces
fakeData = data.frame(HRName = adm1_decimal@data$HRName)

# generate some random data
fakeData = fakeData %>%
```

```
  rowwise() %>%
  mutate(fakeIndicator = sample(1:100,1))

# merge the data together
adm1_df = left_join(adm1_df, fakeData, by = 'HRName')
```

5. Checking the imported and manipulated data looks correct

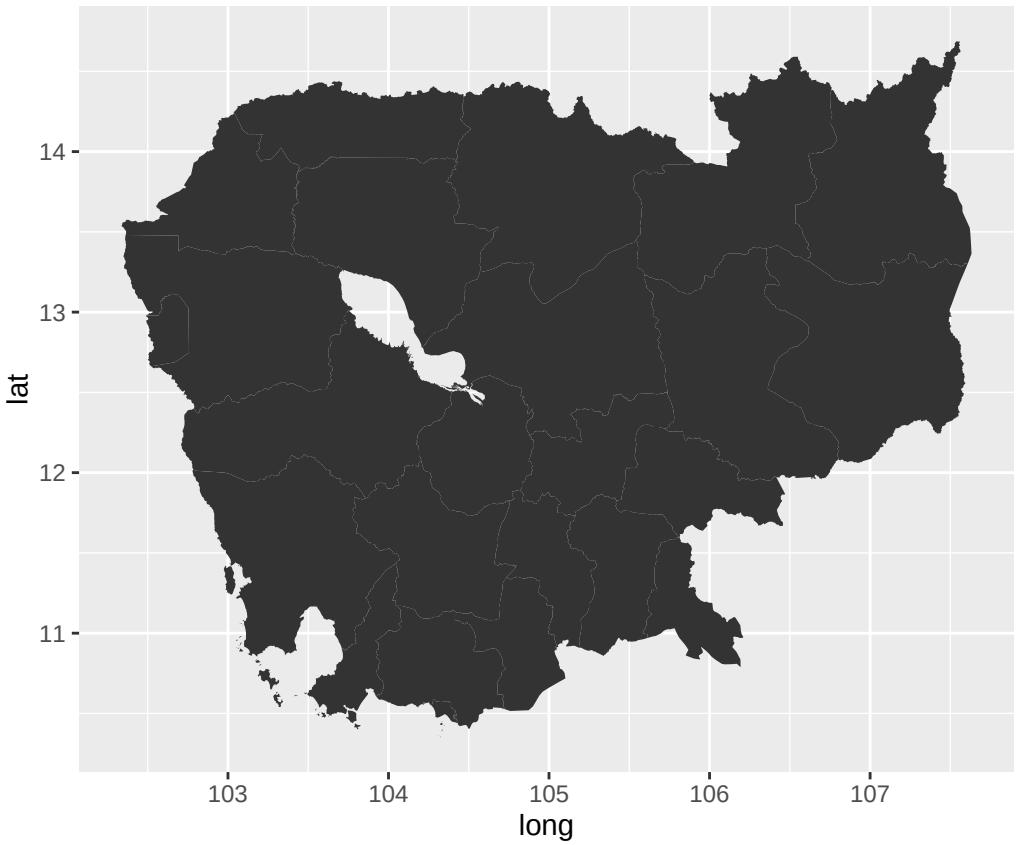
To check that we have what we want, we can plot the file in R using ggplot, quite possibly the most amazing plotting package ever built.

ggplot is built using the Grammar of Graphics, where every element of a plot is built up using a series of marks– basic elements that can be combined to greater effect. So, for example, a bar graph is a series of rectangles that combine to form a bar graph. In the case of a shapefile of polygon data, as you might expect, we'll tell ggplot that we have a group of polygons to plot.

These polygons will be made up of a series of coordinates, with the x being longitude, y being latitude, and will be grouped according to the 'group' variable in our new data frame. The group variable is just a running numbered list of which polygon those coordinates belong to.

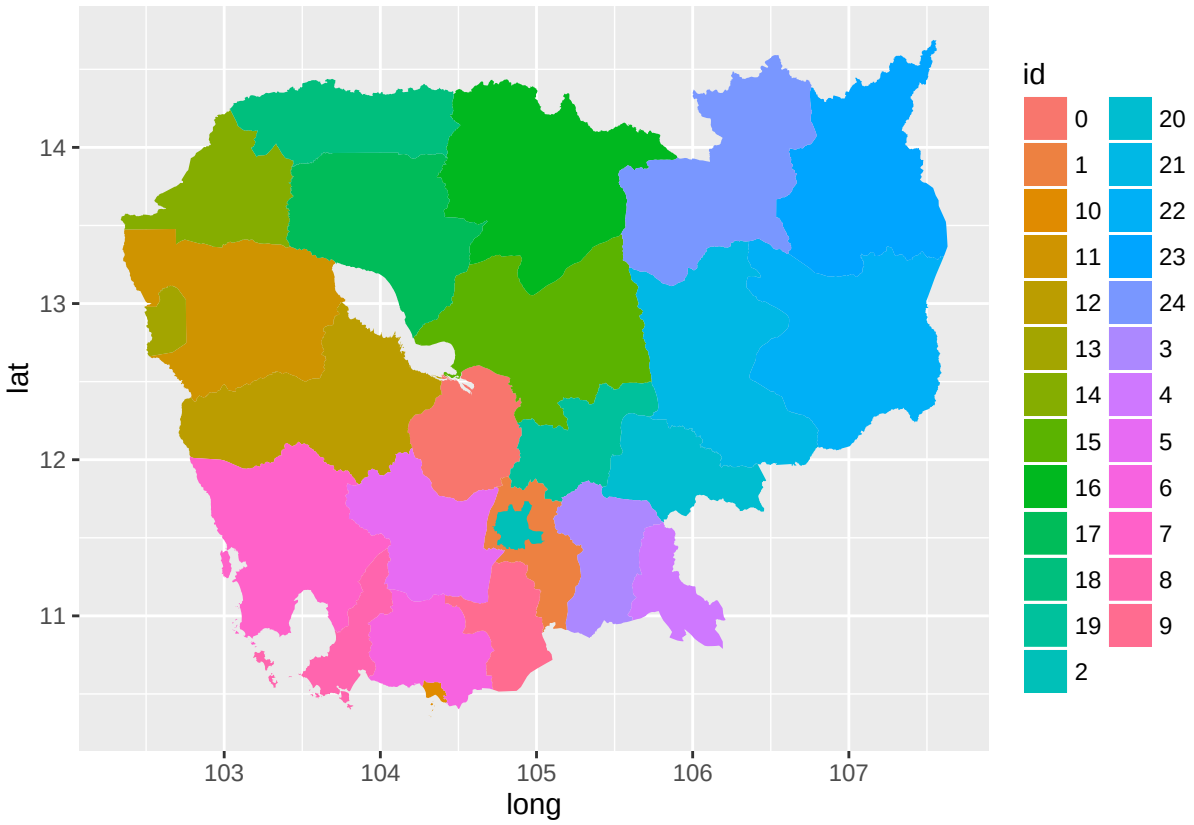
In ggplot, those are given to the function by defining the aesthetics within `aes()`. What's great about ggplot is that every component is built up on top of each other, simply by stringing functions together using `+`. It makes it a really powerful way to add complexity. Here, we define where the data is using `ggplot` and `aes`, tell it that we want to plot them as polygons using `geom_polygon` with equal coordinates using `coord_equal`.

```
ggplot(data = adm1_df, mapping = aes(x = long,
                                     y = lat,
                                     group = group)) +
  geom_polygon() +
  coord_equal()
```



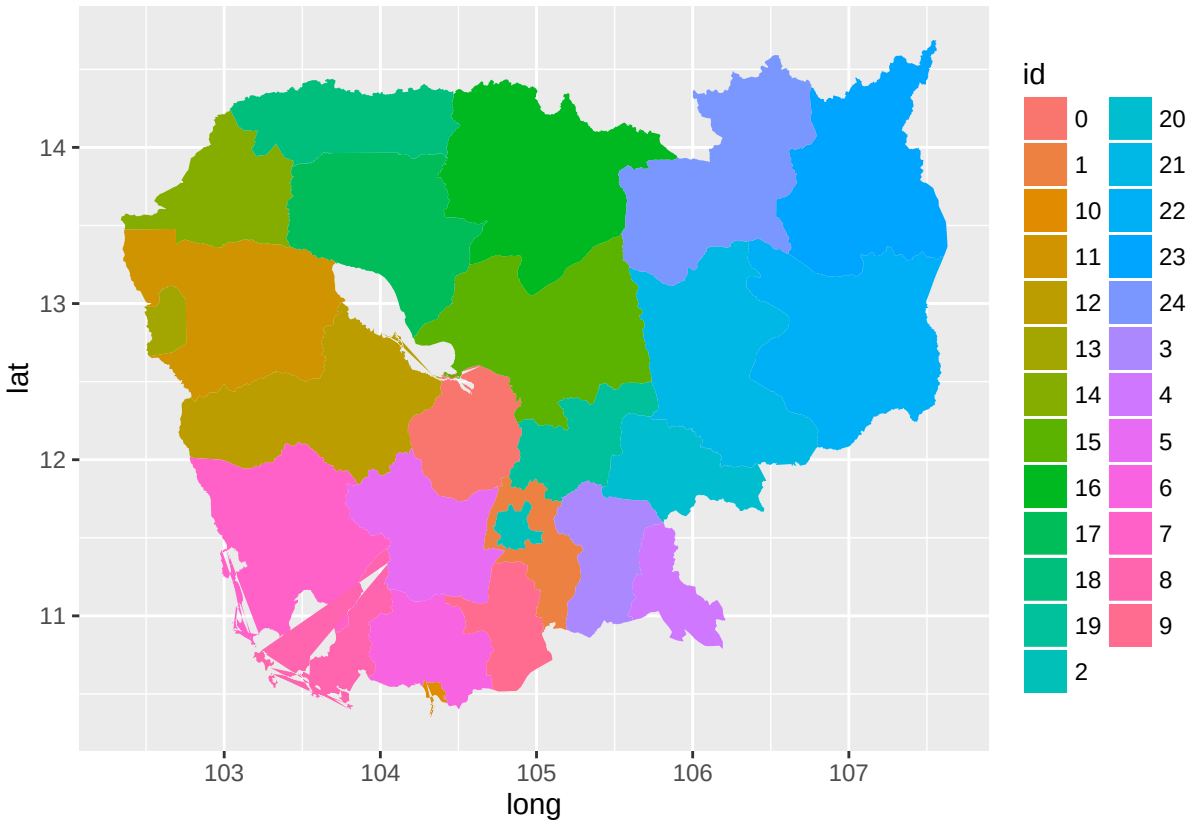
To better see the different administrative units, we can color the `fill` by the id for each different polygon:

```
ggplot(adm1_df, aes(x = long,
                    y = lat,
                    group = group,
                    fill = id)) +
  geom_polygon() +
  coord_equal()
```



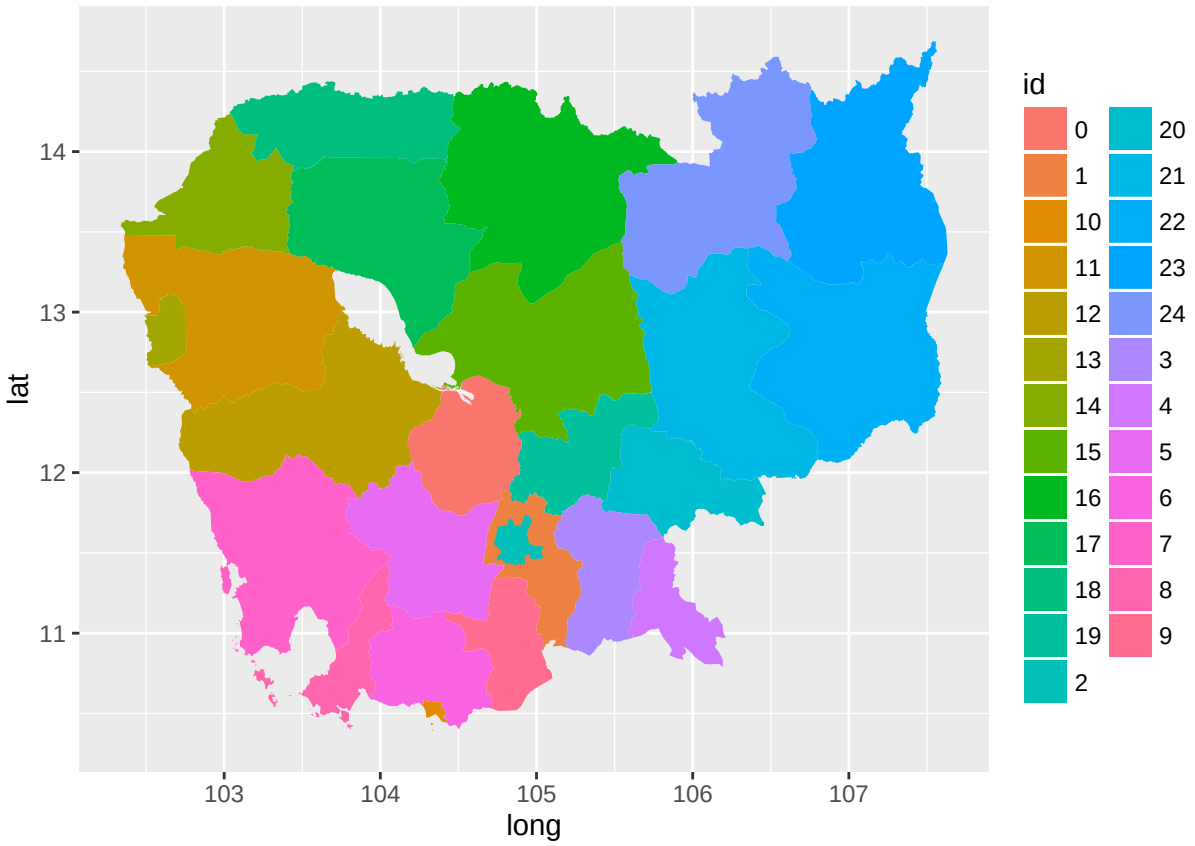
The aesthetic value `group` is the key to making the coordinates plot in the right order and be connected correctly. If we delete that argument, bad things happen:

```
ggplot(adm1_df, aes(x = long,
                    y = lat,
                    fill = id)) +
  geom_polygon() +
  coord_equal()
```



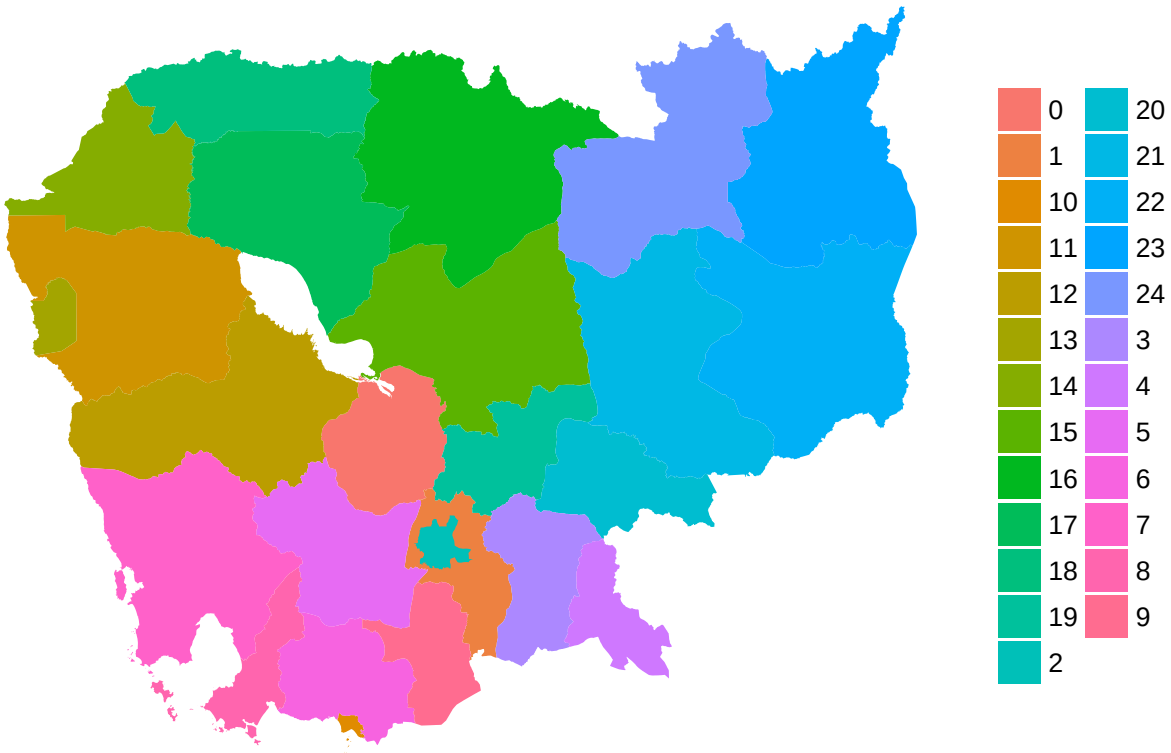
You'll also notice that we've set the coordinates to be equal using `coord_equal()`. If you don't do that, the coordinates will look distorted and screwy:

```
ggplot(adm1_df, aes(x = long,
                    y = lat,
                    group = group,
                    fill = id)) +
  geom_polygon()
```



Now that we have a decent plot, we can start to do some fun things and clean it up. First, we'll get rid of the coordinate background. ggplot has a bunch of themes that determine the aestethics of how things are plotted: what's the format of grid lines, axes, backgrounds, legends, text, ... There are also preset themes that you can call; here, we'll use `theme_void()`, which is essentially a blank background with a legend. The legend is somewhat meaningless here since we're just coloring the different regions, but would be useful in a traditional choropleth (think heatmap for a map). You can customize all the settings using the `theme` function.

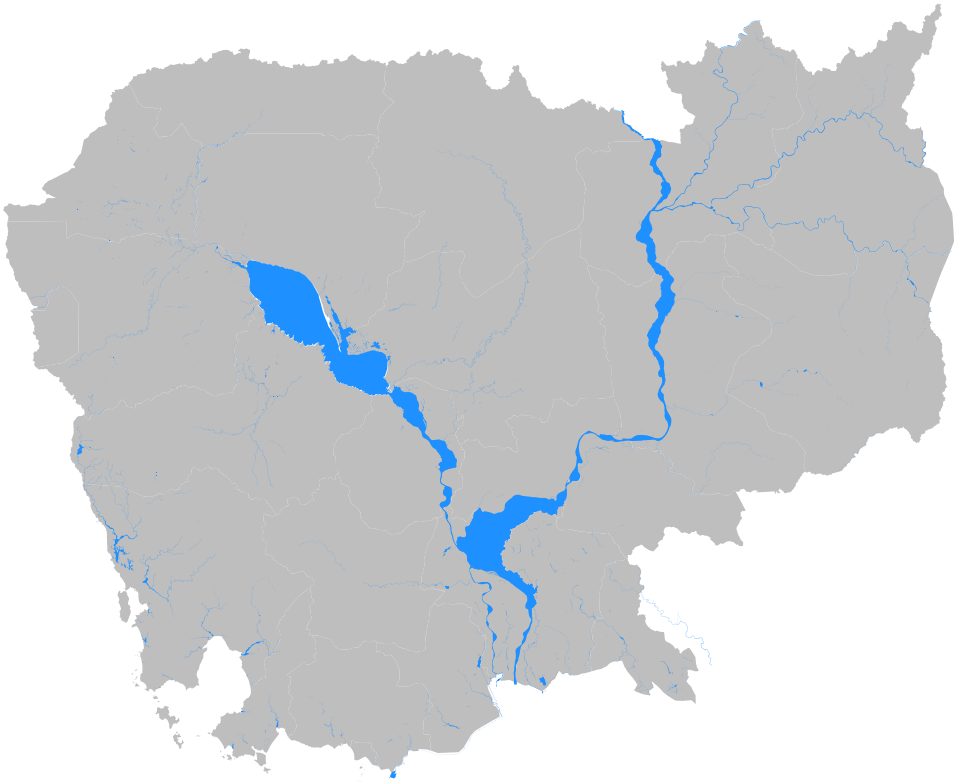
```
ggplot(adm1_df, aes(x = long,
                    y = lat,
                    group = group,
                    fill = id)) +
  geom_polygon() +
  coord_equal() +
  theme_void()
```



Next, we can start to add in other layers of complexity. If we have another GIS layer with the lakes and rivers, we can add it on top of the original plot. To do that, we add another `geom_polygon` layer, but we'll need to specify that the underlying data source is different (`lakes_df`). ggplot will assume the aesthetics are the same as the main plot unless you say otherwise. We'll also make all the admin units grey so the map is less busy.

Note: the lakes layer is complicated so it'll take awhile to plot.

```
ggplot(adm1_df, aes(x = long,
                    y = lat,
                    group = group)) +
  geom_polygon(fill = 'grey') +
  coord_equal() +
  theme_void() +
  geom_polygon(data = lakes_df, fill = 'dodgerblue') # water should be blue, huh?
```

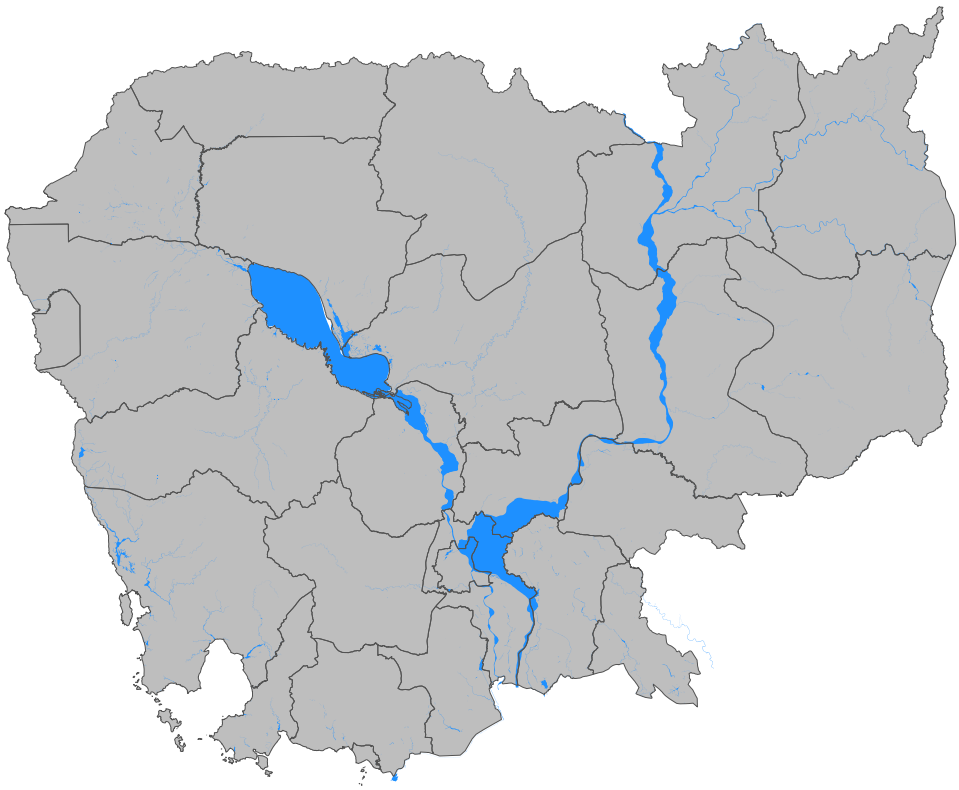


In ggplot, the items are plotted in the order they appear, so if we switch the lakes layer and the Adm1 layer, we'd hide the rivers.

```
ggplot(adm1_df, aes(x = long,
                    y = lat,
                    group = group)) +
  geom_polygon(data = lakes_df, fill = 'dodgerblue') +
  geom_polygon(fill = 'grey') +
  coord_equal() +
  theme_void()
```

We can also add a border around the provinces by adding a `geom_path` layer:

```
ggplot(adm1_df, aes(x = long,
                    y = lat,
                    group = group)) +
  geom_polygon(fill = 'grey') +
  coord_equal() +
  theme_void() +
  geom_polygon(data = lakes_df, fill = 'dodgerblue') +
  geom_path(colour = '#525252', size = 0.1)  # size is in points, colour is most conveniently given as
```

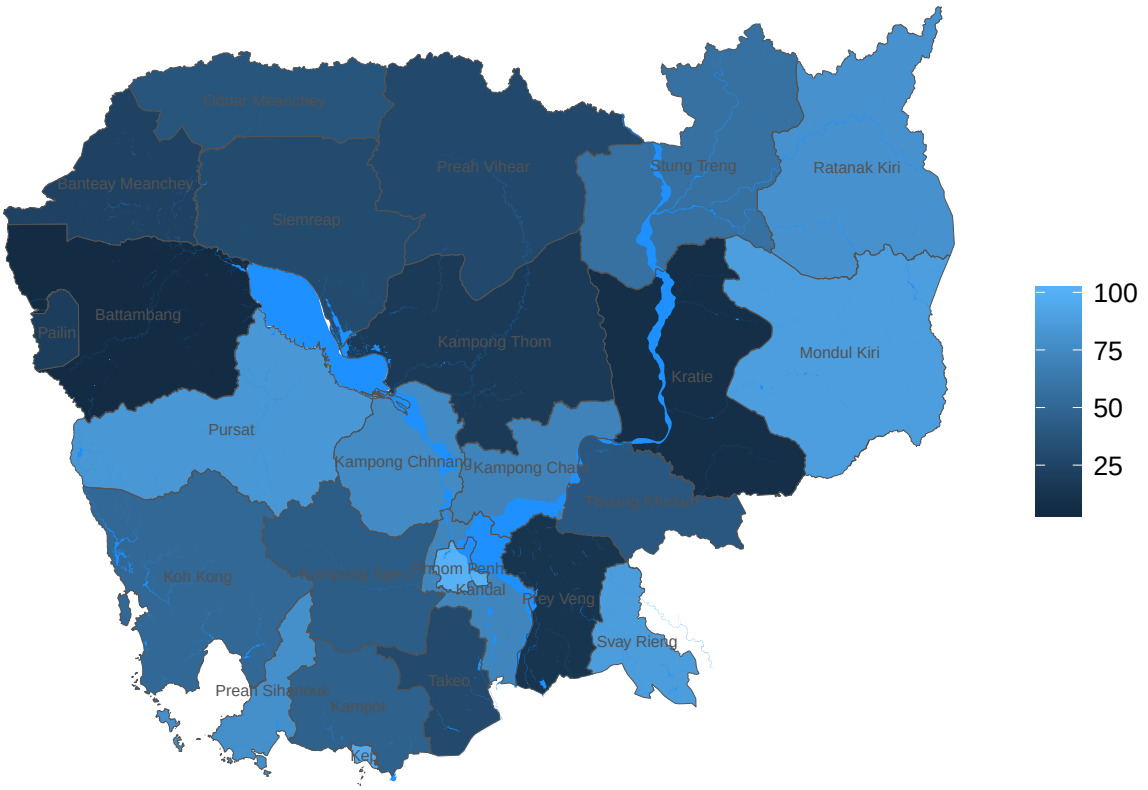


And lastly, we'll add the names of the provinces by adding a `geom_text` layer. `geom_text` has an additional required argument within `aes`: `label`, which specifies what variable contains the text we want to display.

```
ggplot(adm1_df, aes(x = long,
                    y = lat)) +
  geom_polygon(aes(group = group), fill = 'grey') +
  coord_equal() +
  theme_void() +
  geom_polygon(mapping = aes(group = group),
               data = lakes_df, fill = 'dodgerblue') +
  geom_path(aes(group = group), colour = '#525252', size = 0.1) +
  geom_text(aes(label = HRName),
            data = adm1_centroids,
            size = 2,
            colour = '#525252') # set label to be the 'label' variable within adm1_ centroids; size is
```

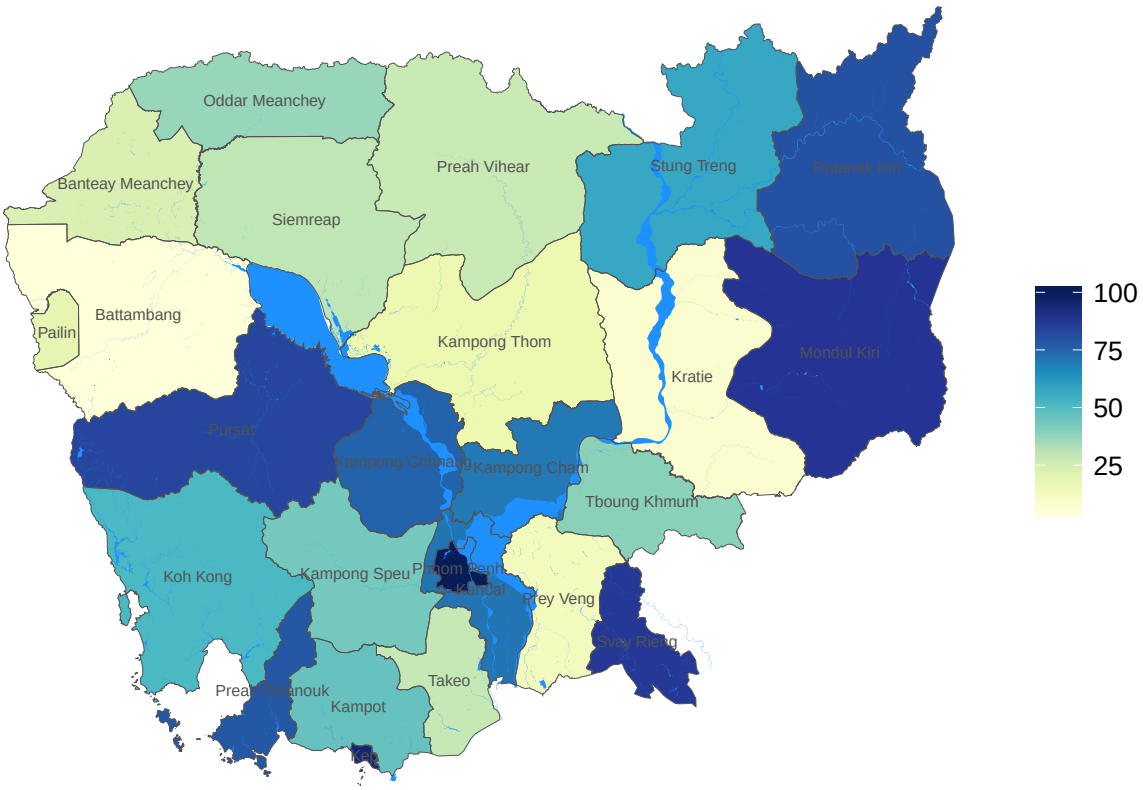
Tying it all together, we can make a choropleth using the random data we generated. We will change the `fill` argument within the aesthetics to use our `fakeIndicator` data.

```
ggplot(adm1_df, aes(x = long,
                    y = lat)) +
  geom_polygon(aes(group = group,
                   fill = fakeIndicator)) +
  coord_equal() +
  theme_void() +
  geom_polygon(aes(group = group),
               data = lakes_df, fill = 'dodgerblue') +
  geom_path(aes(group = group),
            colour = '#525252', size = 0.1) +
  geom_text(aes(label = HRName),
            data = adm1_centroids,
            size = 2,
            colour = '#525252') # set label to be the 'HRName' variable within adm1_ centroids; size is
```



You can then adjust the scale of the indicator data using one of the functions that start with `scale_fill_...`. We'll use `scale_fill_gradeientn` and feed in one of Cynthia Brewer's Color Scales.

```
ggplot(adm1_df, aes(x = long,
                    y = lat)) +
  geom_polygon(aes(group = group,
                   fill = fakeIndicator)) +
  coord_equal() +
  theme_void() +
  geom_polygon(aes(group = group),
               data = lakes_df, fill = 'dodgerblue') +
  geom_path(aes(group = group), colour = '#525252', size = 0.1) +
  geom_text(aes(label = HRName),
            data = adm1_centroids,
            size = 2,
            colour = '#525252') + # set label to be the 'label' variable within adm1_ centroids; size i
  scale_fill_gradientn(colours = brewer.pal(9, 'YlGnBu'))
```



Obviously, we can clean this up and make it better by changing the font color so it's different based on the underlying fill color, but it's a decent start.

6. Save everything!

```
# Save the coordinates as a .csv file
write.csv(adm1_df, '~/Documents/USAID/mini projects/tableaupolygonsfromshapefiles/khm_admbnda_adm1_gov_

write.csv(lakes_df, '~/Documents/USAID/mini projects/tableaupolygonsfromshapefiles/khm_admbnda_adm1_gov
```