Data Visualization with ggplot2 (Part 3)

Plots for Specific Data Types (Chapter 3)

Seun Odeyemi

2019-04-18

Contents

Load Libraries

```
library(readr)
library(dplyr)
library(ggplot2)
# library(ggplot2movies)
library(tidyr)
library(skimr)
library(knitr)
library(kableExtra)
library(RColorBrewer)
library(grid)
library(ggthemes)
library(forcats)
library(GGally)
library(here)
library(hexbin)
```

Choropleths

In this chapter we'll wrap up our discussion of specialty plots by considering **maps** and **animations** plus we'll see some concepts from the previous chapter coming to play. Let's begin with maps. Many people who work with maps are turning toward R as a Geographic Information System (GIS). This is because of its capabilities for spatial statistics and mapping are steadily improving. Using R as a full-flegded GIS is a course onto itself. Here my goal is to introduce you to two commonly used map types, which can both be easily produced in `ggplot2`: Choropleths and Carthographic Maps.

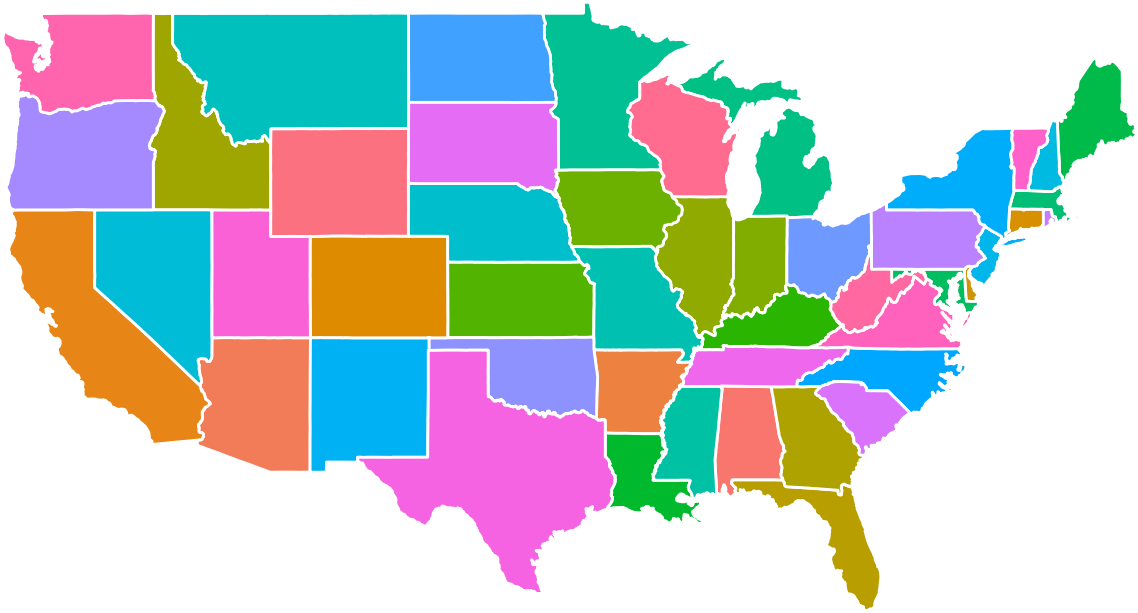
Let's start with Choropleths. You have likely encountered this type of map in popular media, in particular whenever elections are held. If you've completed the Kaggle challenge course, you would have also seen an example of the Choropleths R package. The thing to remember about Choropleths is that basically we are just drawing a bunch of polygons (ok we could also drwa points and lines, but we'll just stick with polygons for the moment). You can imagine that when we will draw a map, like the outline of the United States shown below, it's basically a large polygon with many sides. All we need is a file which tells us the latitude

or longitude of the point of the polygon. In some cases, you can find this included in a R package, like the example below. But, in the exercises we will explore how to use special shape files which can contain information about geograhical and political boundaries. Notice that, this is just a basic `ggplot2` plot. We can event adjust the coordinate system to a different projection (as shown below).

```
usa <- map_data("usa")

ggplot(usa, aes(long, lat, group = group)) +
  geom_polygon() +
  coord_map() +
  theme_nothing()
#> Error in theme_nothing(): could not find function "theme_nothing"

library(ggalt)
ggplot(usa, aes(long, lat, group = group)) +
  geom_polygon() +
  coord_proj("+proj=wintri") +
  theme_nothing()
#> Error in theme_nothing(): could not find function "theme_nothing"
```

Additionally, we may have information on many polygons such as individual states. This will allow us to visualize each individual polygon. Here, we've done that by mapping `region` onto the `fill` aesthetic.

```
states <- map_data("state")

usa <- map_data("usa")

library(ggmap)
#> Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
#> Please cite ggmap if you use it! See citation("ggmap") for details.
ggplot(states,
       aes(long, lat,
           fill = region, group = group)) +
  geom_polygon(color = "white") +
  coord_map() +
  #theme(legend.position = "none")
  theme_nothing()
```



Now, if our polygons had names such as the states' names, then we could map that value associated with a polygon. For example, a dataset which you may have seen elsewhere on datacamp is the average price of weed in each US state. You can probably imagine that the only difference between this and previous plots was merging the weed prices data set with the polygon data set – matching up the state names and then mapping the price onto the fill aesthetic. Again, we can take advantage of all the tools that are provided by `ggplot2` so we may decide to use a more appropriate pallete and we make the darker colors the higher values. **I want to stress at this point that just because you have geographic data and you can make a choropleth does not mean you have to. It is not the only or a necessary visualization.** Many times people with geographic data default to maps, but recall that a continuous color scale is not actually an efficient encoder of continuous data. Choropleths excel when we have a few number of polygons or, in the case when we have many, a clear trend emerges.

An alternative is the classic **Cleveland Dot Plot**. The advantage here is that we can plot additional variables as we saw in previous lessons. We can also order the states alphabetically or, more revealingly, in order of a continuous variable. Now, we notice trends like most trips are between $ $275 $ to $350. North Dakota is the highest price and by relatively wide margin. It is not that this type of plot is better it just allows us to answer a different questions than a choropleth. It is also very unsexy, which is why you are likely to see choropleth instead of a dot plot in the popular press. Another advantage here is that we can facet the data grouping states in line with US Census defined regions. Here, we now see that the lowest prices are predominantly in the West. We can go further defining regions and divisions, but does not necessarily tell us anything new.

Working with Maps from the Maps Package: Adding Points

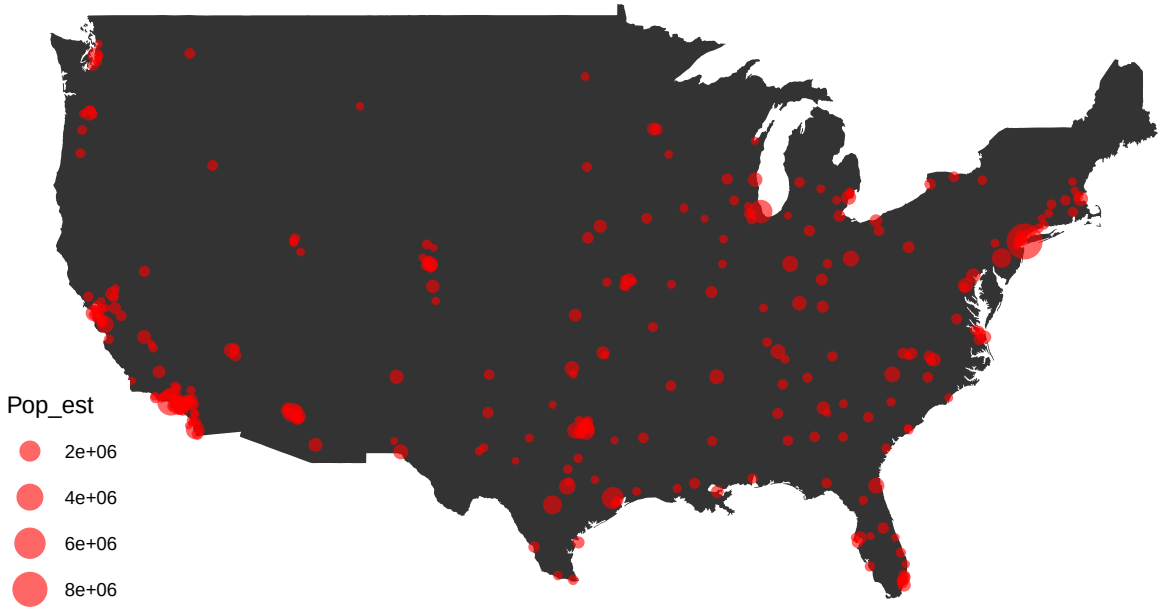
Now that you have some polygons, there are a number of things you can do. Here you'll add some data points, namely the location of US cities with a population over 100,000 (population estimation as of 2015). Since you're only looking at the continental US, Honolulu, Hawaii and Anchorage, Alaska are not included.

The data is stored in the `cities` data frame. You'll begin by drawing points of varying sizes, relative to the estimated population. An alternative is to use color instead of size, and in this case a nice trick is to order the data frame, so that the largest cities are drawn on *top* of the smaller cities. This is so that they will stand out against the background, which is particularly effective when using the viridis color palette.

```
# Finish plot 1

cities <- read_delim("datasets/US_Cities.txt", delim = "\t")
#> Parsed with column specification:
#> cols(
#>   City = col_character(),
#>   State = col_character(),
#>   Pop_est = col_double(),
#>   lat = col_double(),
#>   long = col_double()
#> )

cities %>%
  head() %>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"))
```

City	State	Pop_est	lat	long
Eugene	Oregon	163460	44.0567	-123.1162
Salem	Oregon	164549	44.9237	-123.0231
Hillsboro	Oregon	102347	45.5167	-122.9833
Santa Rosa	California	174972	38.4468	-122.7061
Portland	Oregon	632309	45.5370	-122.6500
Vancouver	Washington	172860	45.6372	-122.5965

```
ggplot(usa, aes(x = long, y = lat, group = group)) +
  geom_polygon() +
  geom_point(data = cities, aes(group = State, size = Pop_est),
             col = "red", shape = 16, alpha = 0.6) +
  coord_map() +
  theme_map()
```

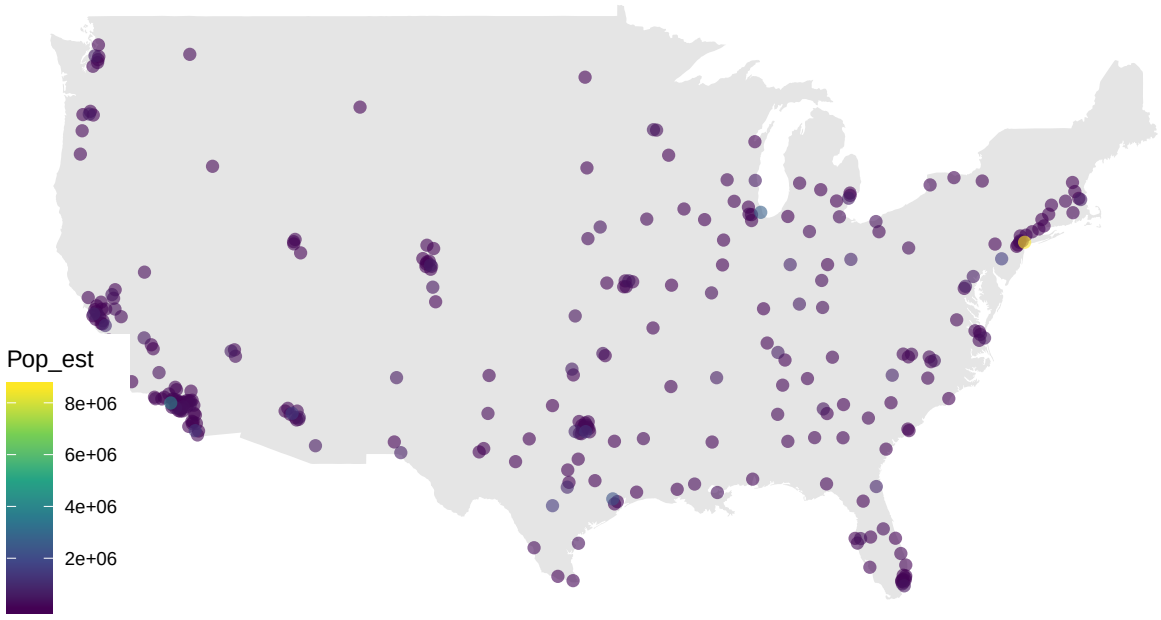



```
# Arrange cities
library(dplyr)
cities_arr <- arrange(cities, Pop_est)

cities_arr %>%
  head() %>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"))
```

City	State	Pop_est	lat	long
Renton	Washington	100242	47.4867	-122.1953
Jurupa Valley	California	100314	33.0011	-117.4706
San Angelo	Texas	100450	31.4500	-100.4500
Davie	Florida	100882	26.0814	-80.2803
Greeley	Colorado	100883	40.4167	-104.7167
Vista	California	100890	33.1936	-117.2411

```
# Copy-paste plot 1 and adapt
ggplot(usa, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "grey90") +
  geom_point(data = cities_arr, aes(group = State, col = Pop_est),
             shape = 16, alpha = 0.6, size = 2) +
  coord_map() +
  theme_map() +
```

```
  scale_color_viridis_c()
```



Great! New York appears as a bright yellow anomly in a sea of darker points. If you didn't set the order of the data, this point would have been obscured. You can also see LA, Chicago and Houston as lighter blue points.

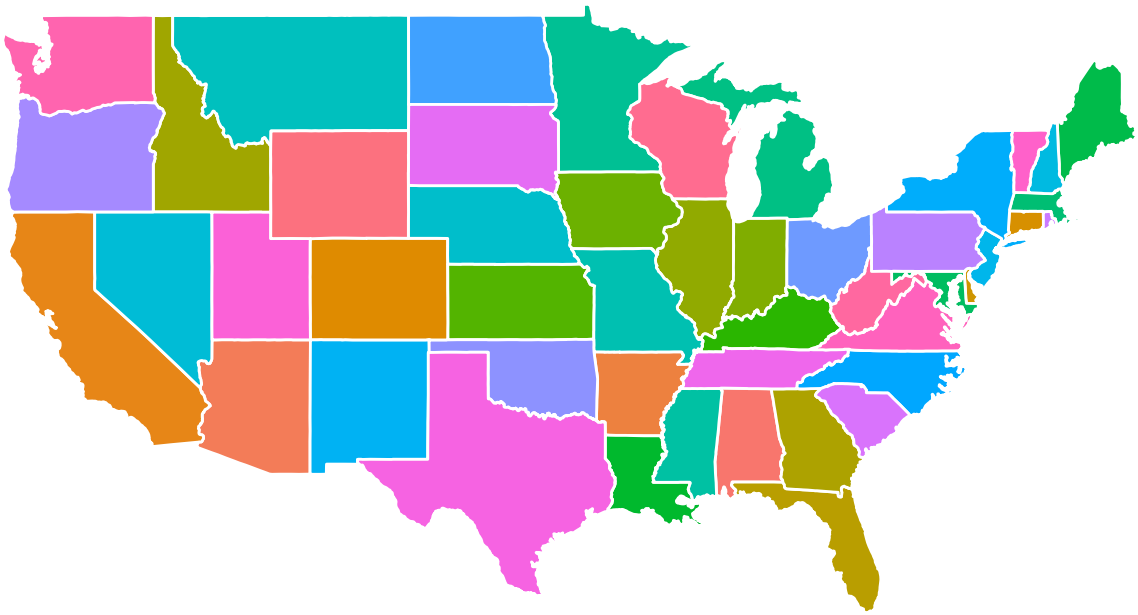
State Choropleth

To make a choropleth (a map in which areas are shaded according to some measure) you'll need information on state boundaries, which you can find in the `maps` package. Once the map information is converted into a data frame, you can merge this with another data frame containing some quantitative information, like the estimated population, and use that variable in our aesthetic mappings.

```
# Use map_data() to create state
state <- map_data("state") %>% as_tibble()

head(state)
```

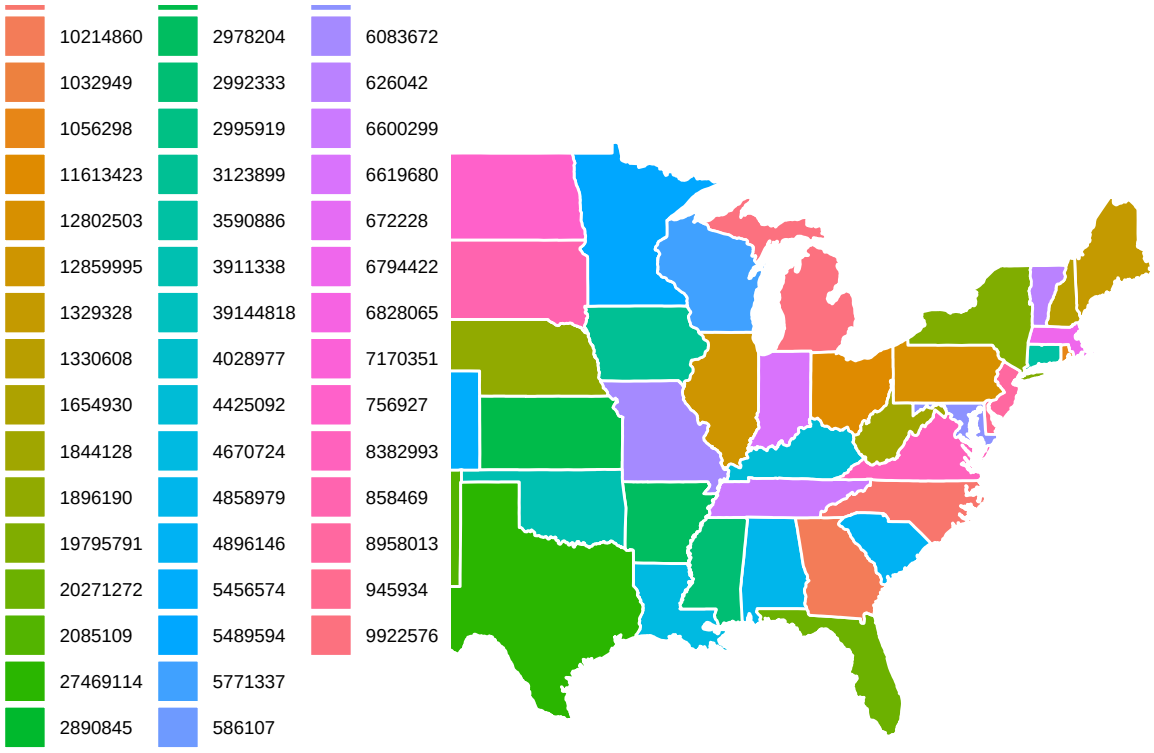
long	lat	group	order	region	subregion
-87.46201	30.38968	1	1	alabama	NA
-87.48493	30.37249	1	2	alabama	NA
-87.52503	30.37249	1	3	alabama	NA
-87.53076	30.33239	1	4	alabama	NA
-87.57087	30.32665	1	5	alabama	NA
-87.58806	30.32665	1	6	alabama	NA

```
state <- state %>% dplyr::rename ("state" = "region")

# Map of states
ggplot(state, aes(x = long, y = lat, fill = state, group = group)) +
  geom_polygon(col = "white") +
  coord_map() +
  theme_nothing()
```



```
# Merge state and pop: state2
library(janitor)

pop <- read_delim("datasets/US_States.txt", delim = "\t") %>% mutate_all(funs(tolower))
#> Parsed with column specification:
#> cols(
#>   State = col_character(),
#>   Pop_est = col_double()
#> )
#> Warning: funs() is soft deprecated as of dplyr 0.8.0
#> please use list() instead
#>
#> # Before:
#> funs(name = f(.)
#>
#> # After:
#> list(name = ~f(.))
```

```
#> This warning is displayed once per session.

pop <- pop %>% clean_names(case = "snake")

state2 <- merge(state, pop)


# state %>% inner_join(pop, by = c("state" = "State"))

# state2 <- dplyr::select(state2, everything(), -state, -State)

# Map of states with populations
ggplot(state2, aes(x = long, y = lat, fill = pop_est, group = group)) +
  geom_polygon(col = "white") +
  coord_map() +
  theme_map()
```



Maps from Shapefiles

Although the built-in maps from the `maps` package are very convenient, using shapefiles is a more flexible way of accessing geographic and political boundaries.

Shapefiles can be used to describe points, polylines or polygons - here you'll focus on polygons for drawing maps.

A single shapefile actually consists of several files, each describing a specific aspect of the overall geometry. The three core file types are:

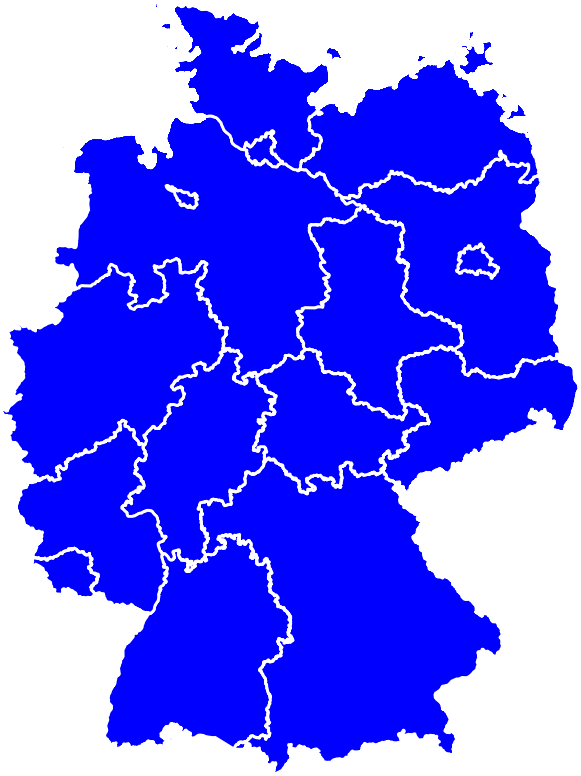
- .shp: the shape, the feature geometry itself.
- .shx: the shape index, a positional index.
- .dbf: the attribute, attributes for each shape arranged in columns.

The prefix name of these files must be consistent and they must be kept in the same directory. The files you'll use are for Germany, and all begin with DEU. The suffix specifies the level of organization:

DEU_adm0 is the administrative (political) boundaries of the entire country (like the usa object before) DEU_adm1 is the administrative boundaries for each of the 16 states (like the state object before) Let's start by importing the shapefile and creating a map of Germany. All shapefiles you need are in the shapes folder of your working directory; you can check it out with dir(). In the next exercise, you'll take things one step further.

NOTE: Building these maps is computationally heavy, so it can take some time before you see your results.

```
library(rgdal)
#> Loading required package: sp
#> rgdal: version: 1.4-3, (SVN revision 828)
#>  Geospatial Data Abstraction Library extensions to R successfully loaded
#>  Loaded GDAL runtime: GDAL 2.2.3, released 2017/11/20
#>  Path to GDAL shared files: C:/Users/USER/Documents/R/win-library/3.5/rgdal/gdal
#>  GDAL binary built with GEOS: TRUE
#>  Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
#>  Path to PROJ.4 shared files: C:/Users/USER/Documents/R/win-library/3.5/rgdal/proj
#>  Linking to sp version: 1.3-1
germany <- readOGR(dsn = "shapes", layer = "DEU_adm1")
#> OGR data source with driver: ESRI Shapefile
#> Source: "C:\Users\USER\repos\datacamp_courses\shapes", layer: "DEU_adm1"
#> with 16 features
#> It has 16 fields

bundes <- fortify(germany)
#> Regions defined for each Polygons

ggplot(data = bundes, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "blue", col = "white") +
  coord_map() +
  theme_nothing()
```



Great! You can also imagine that it would be possible to draw single layers, like only the shape for Berlin.

Choropleth from Shapefiles

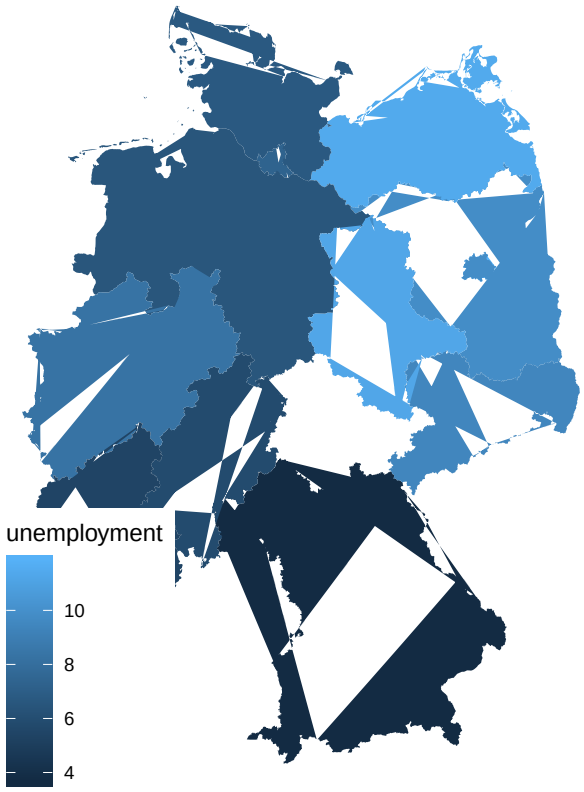
Now that you have the shape data a neatly formatted data frame called `bundes`, you can easily merge it with state-level data. `germany`, `bundes` and 'unemp" (a data frame on unemployment in Germany) is available in your workspace. Can you do some data munging and then make a fancy choropleth?

If you check out `bundes`, you'll see that you've lost the state names, so merging isn't going to work out of box. That's why we've added two lines of code in the editor, to re-add state names.

NOTE: Building these choropleths is computationally heavy, so it can take some time before you see your results

```
unemp <- read_delim("datasets/germany_unemployment.txt", delim = "\t")
#> Parsed with column specification:
#> cols(
#>   state = col_character(),
#>   unemployment = col_double()
#> )

# re-add state names to bundes
bundes$state <- factor(as.numeric(bundes$id))
levels(bundes$state) <- germany$NAME_1

# Merge bundes and unemp: bundes_unemp
bundes_unemp <- merge(bundes, unemp)
```

```
# Update the ggplot call
p <- ggplot(bundes_unemp, aes(x = long, y = lat, group = group, fill = unemployment)) +
  geom_polygon() +
  coord_map() +
  theme_map()

p + geom_polygon(data = subset(bundes_unemp, id == "Berlin"))
```



At the outset this looks fine, but if you count the states, you'll see you only have 14! The reason is that Berlin and Bremen are city-states located within the shape definitions of other states. You can solve this by reording the data (like you did before), or by adding two specific layers, for example, `geom_polygon(data = subset(bundes_unemp, id == "Berlin"))`.

Cartograhic Maps

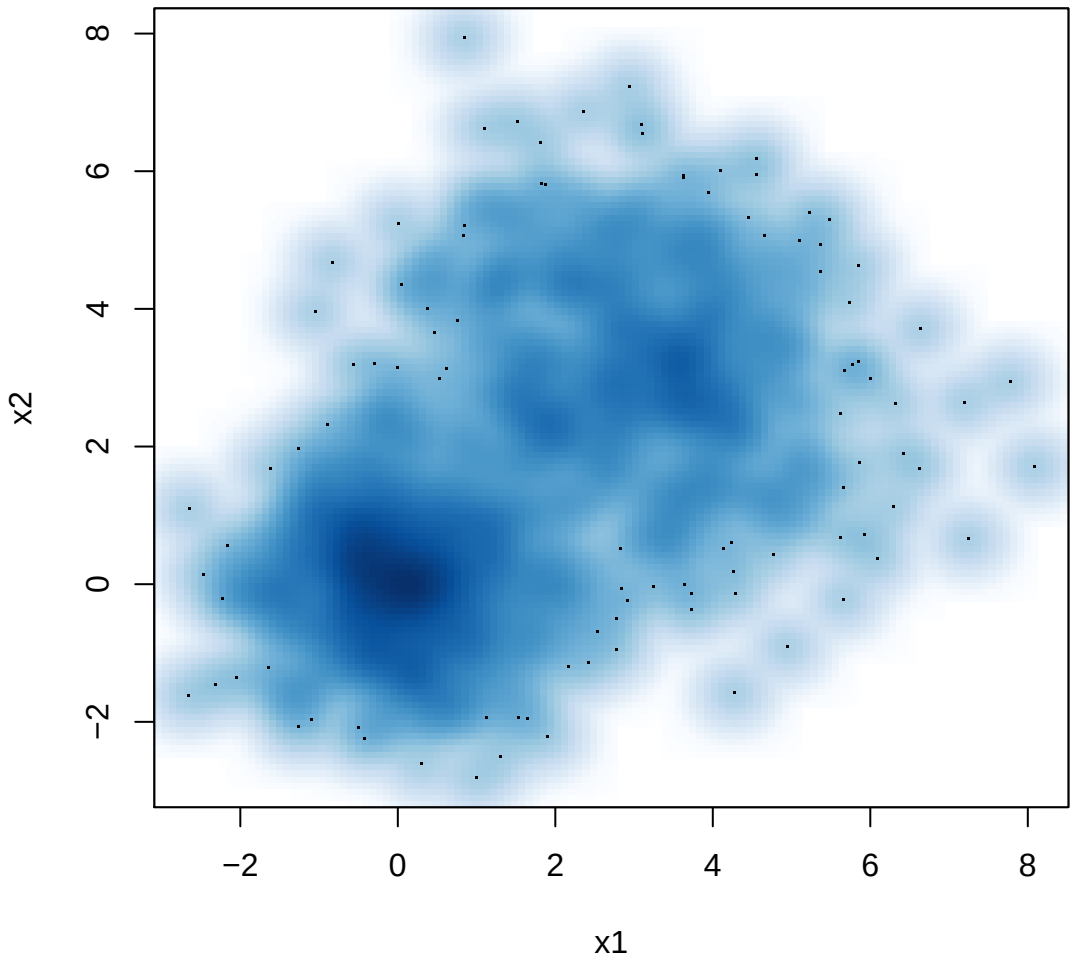
`html_pretty` currently supports three page themes (`cayman`, `tactile` and `architect`), and two syntax highlight styles (`github` and `vignette`). The theme and highlight styles can be specified in the document metadata, for example:

```
output:
  prettydoc::html_pretty:
    theme: architect
    highlight: github
```

Animations

Feel free to use the `knitr` infrastructure with dozens of tunable options in your document.

```r
set.seed(123)
n <- 1000
x1  <- matrix(rnorm(n), ncol = 2)
x2  <- matrix(rnorm(n, mean = 3, sd = 1.5), ncol = 2)
x   <- rbind(x1, x2)
head(x)
#>             [,1]        [,2]
#> [1,] -0.56047565 -0.60189285
#> [2,] -0.23017749 -0.99369859
#> [3,]  1.55870831  1.02678506
#> [4,]  0.07050839  0.75106130
#> [5,]  0.12928774 -1.50916654
#> [6,]  1.71506499 -0.09514745
smoothScatter(x, xlab = "x1", ylab = "x2")
```



You can include code snippets of languages other than R, but note that the block header has no curly brackets around the language name.

```
// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {
    return x * 2;
}
```

You can also write math expressions, e.g. $Y = X\beta + \epsilon$, footnotes[1], and tables, e.g. using `knitr::kable()`.

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

Stay Tuned

Please visit the development page of the `prettydoc` package for latest updates and news. Comments, bug reports and pull requests are always welcome.

[1] A footnote here.