Week 8 Lab

Maps, Functions, and Loops

Micaela Wood
02/24/2022

Today

- Spatial Data.
- Preparing Maps.
- Functions
- For Loops

Spatial Data

`maps` allows you to draw lines and polygons as specified by a map database, calculate their areas and comes with a set of prepared databases to work with. Two Types of Spatial Data; **Spatial Point Data**: Represent the locations of events as points on a map. **Spatial Polygon Data**: Represent geographic areas by connecting points on a map.

```
p_load(maps)
data(us.cities)
head(us.cities)
```

name	country.etc	pop	lat	long	capital
Abilene TX	TX	113888	32.5	-99.7	0
Akron OH	OH	206634	41.1	-81.5	0
Alameda CA	CA	70069	37.8	-122	0

Spatial Data

Lets prepare a map of the US and plot its capitals.

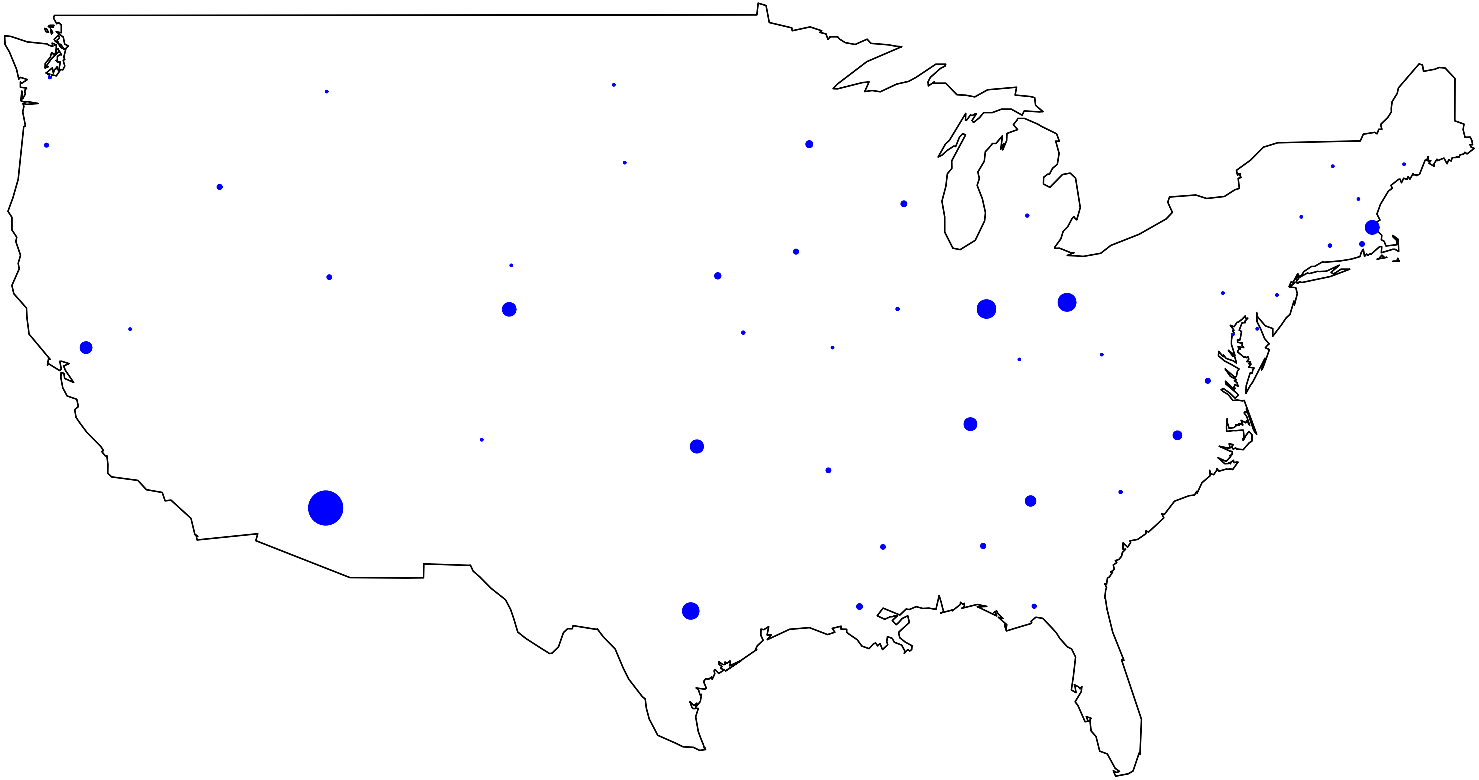
We can scale the size of the points for capitals based on population, set the points to be blue and choose solid circles (pch=19). If you wanted a filed circle instead, try (..., pch=21, bg="red").

```r
map(database = "usa")
# subset state capitals
capitals <- subset(us.cities, capital == 2)

points(x = capitals$long, y= capitals$lat, col = "blue",
       cex = capitals$pop / 500000, pch=19)
title("US state capitals")
```

Spatial Data

US state capitals



Spatial Data

Lets load in state borders.

```r
map(database = "state")
# subset state capitals
capitals <- subset(us.cities, capital == 2)

points(x = capitals$long, y= capitals$lat, col = "blue",
       cex = capitals$pop / 500000, pch=19)
title("US state capitals")
```

Spatial Data

```
map(database = "county")
capitals <- subset(us.cities, capital == 2)
points(x=capitals$long,y=capitals$lat,col ="blue",cex=capitals$pop/500000,pch=19
title("US state capitals")
```

Spatial Data

```r
map("world", "Ireland")
map.cities(country = "Ireland", capitals = 0,
      pch=21, bg="green", cex=capitals$pop/500000, col ="black" )
```

Spatial Data

```r
map("world", "France")
map.cities(country = "France", capitals = 0,
      pch=21, bg="green", cex=capitals$pop/500000, col ="black" )
```

Spatial Data

```
map("world", "Italy")
map.cities(country = "Italy", capitals = 0,
      pch=21, bg="green", cex=capitals$pop/500000, col ="black" )
```

Spatial Data

An Alternative for Maps: `usmap`

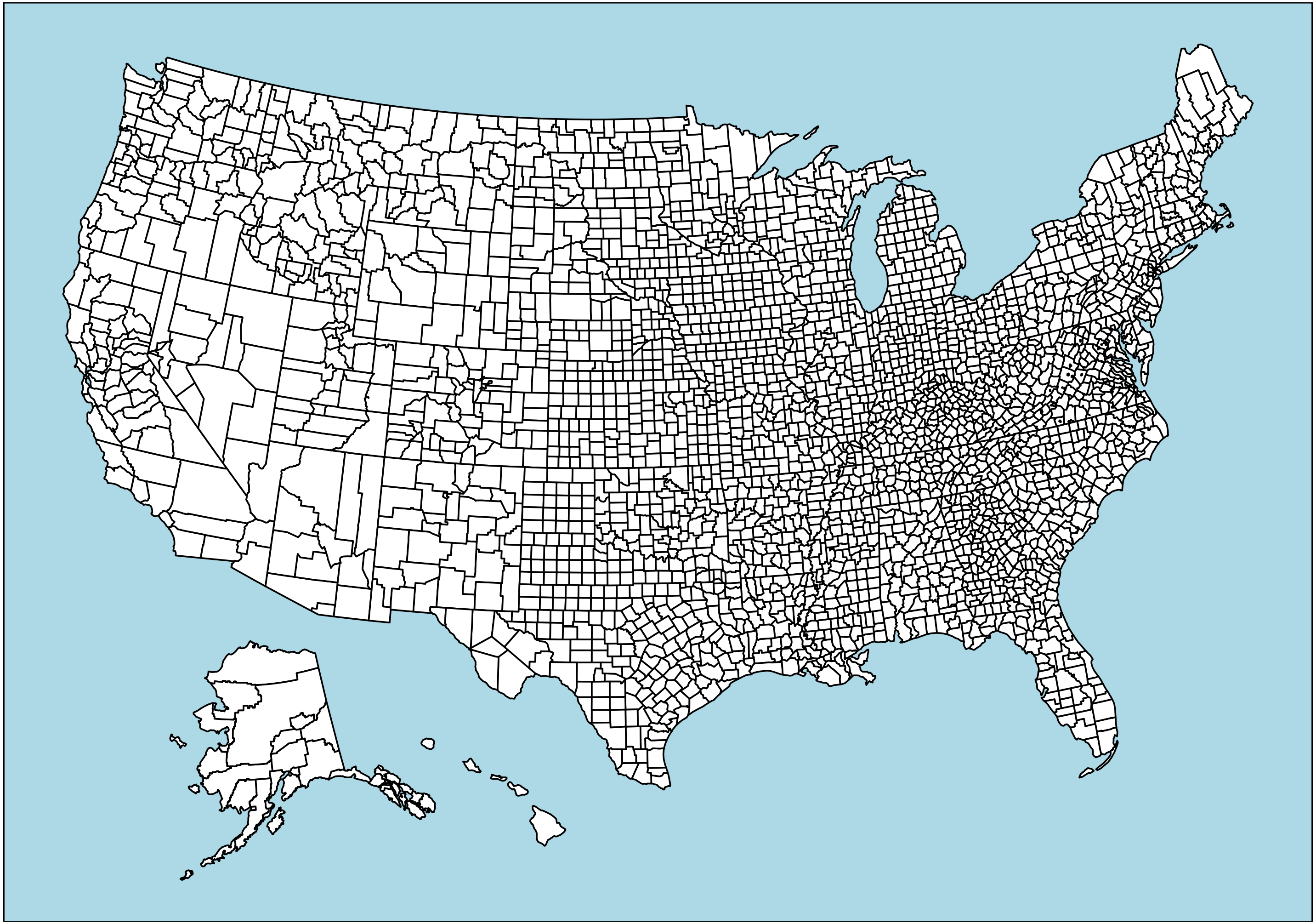
```r
p_load(usmap, ggplot2)

plot_usmap(regions = "counties") +
  labs(title = "US Counties",
       subtitle = "This is a blank map of the counties of the United States.") +
  theme(panel.background = element_rect(color = "black", fill = "lightblue"))
```

Spatial Data

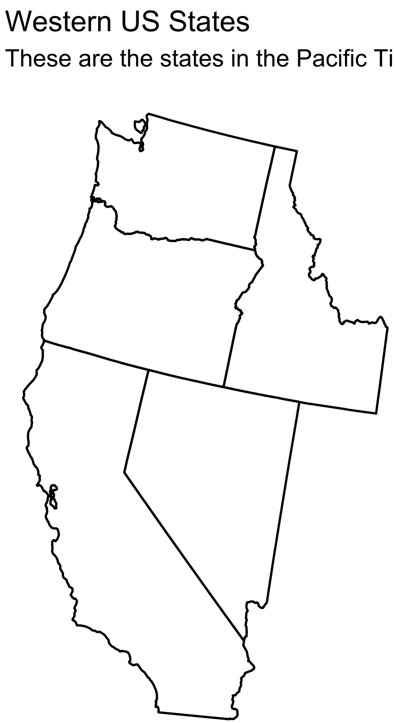
US Counties

This is a blank map of the counties of the United States.



Spatial Data

```r
plot_usmap(include = c("CA", "ID", "NV", "OR", "WA")) +
  labs(title = "Western US States",
       subtitle = "These are the states in the Pacific Timezone.")
```



Spatial Data

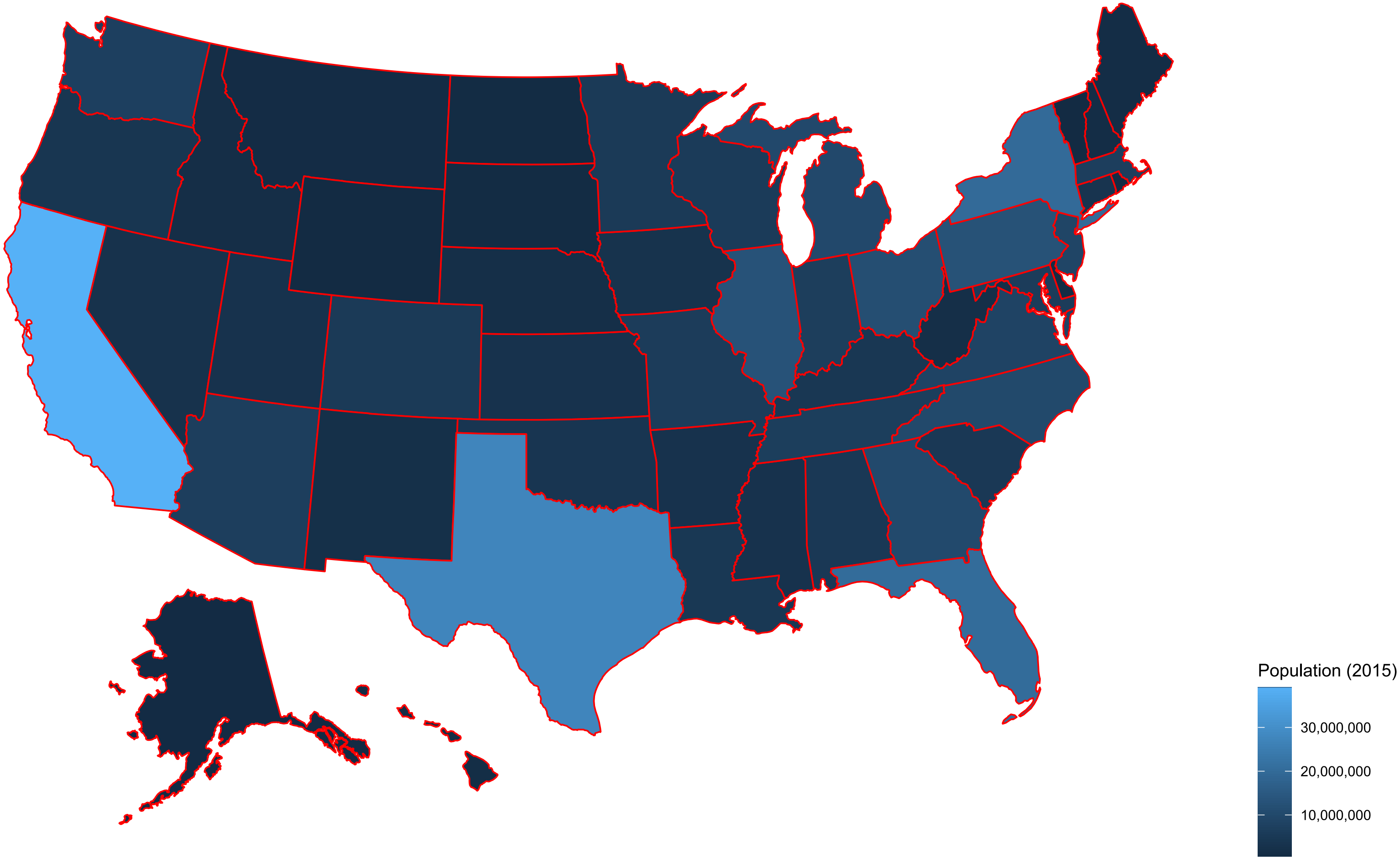
Lets start colouring in these states given the data that we have. This is normally the most common means to which spatial data is used.

The code below is using state population data from the `usmaps` package so we don't need to load it separately. We will colour in the states letting it differ by populations in 2015. Red will be the colour of the borders.

The continuous fill to the legend and allowing the scaling values to have commas to them presents an especially pleasing aesthetic.

```r
plot_usmap(data = statepop, values = "pop_2015", color = "red") +
  scale_fill_continuous(name = "Population (2015)", label = scales::comma) +
  theme(legend.position = "right")
```

Spatial Data

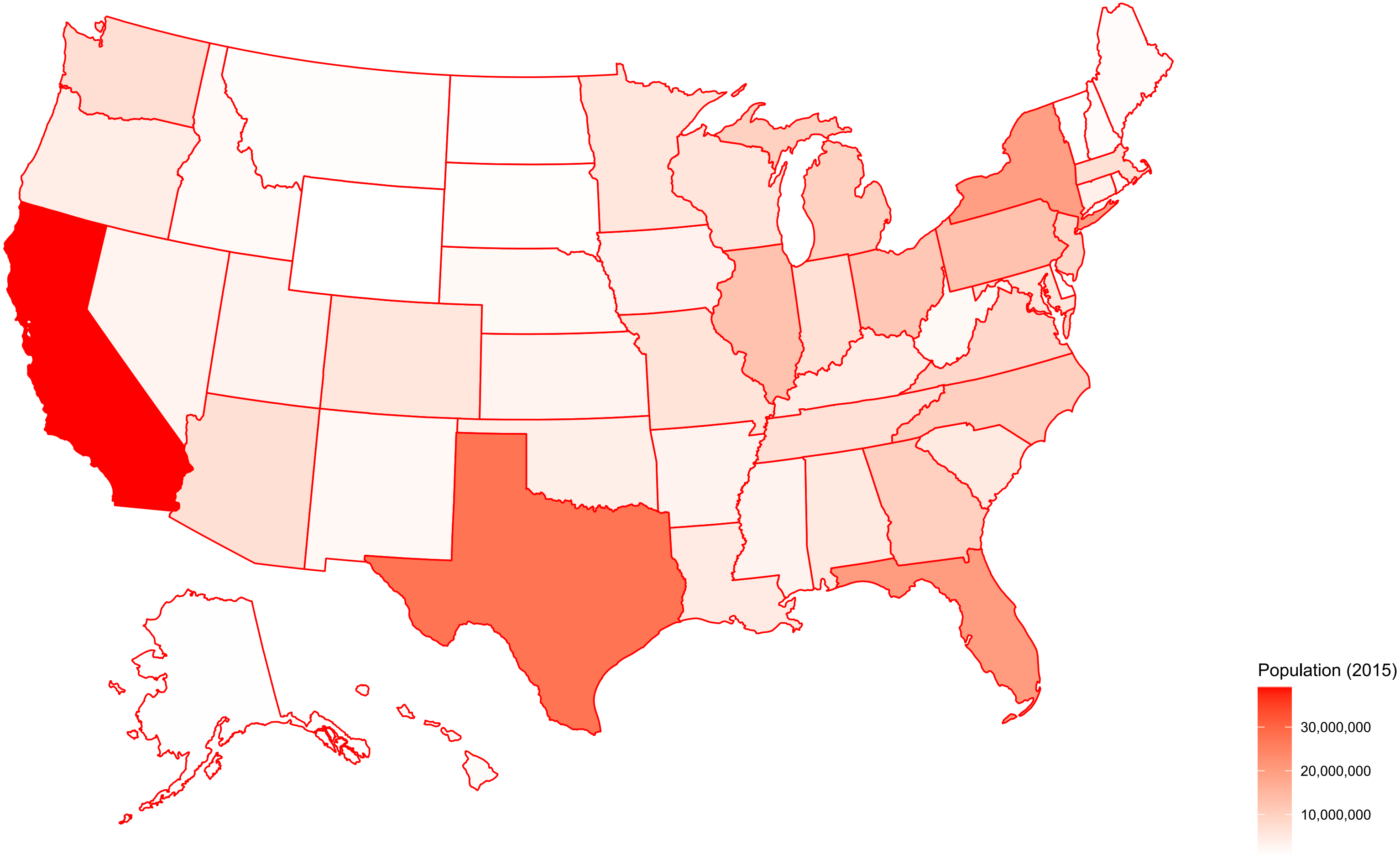


Spatial Data

The theme can be very easily adjusted too. Though we're looking at population, this would be a great template for studies of air pollution or avergate temperatures per annum.

```
plot_usmap(data = statepop, values = "pop_2015", color = "red") +
  scale_fill_continuous(
    low = "white", high = "red", name = "Population (2015)", label = scales::com
  ) + theme(legend.position = "right")
```

Spatial Data

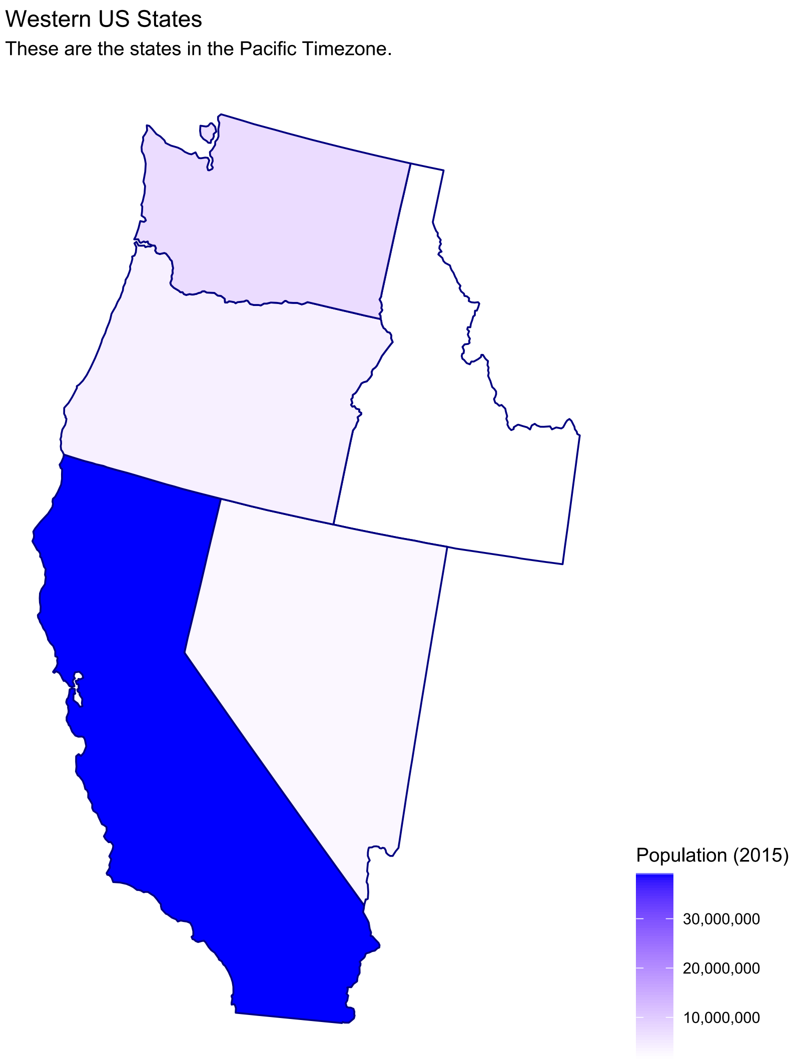


Spatial Data

This is easily applicable to state maps too.

```
plot_usmap(
    data = statepop, values = "pop_2015", include = c("CA", "ID", "NV", "OR", "W
    color = "navy") + scale_fill_continuous(
    low = "white", high = "blue", name = "Population (2015)", label = scales::co
  ) +
  labs(title = "Western US States",
       subtitle = "These are the states in the Pacific Timezone.") +
  theme(legend.position = "right")
```

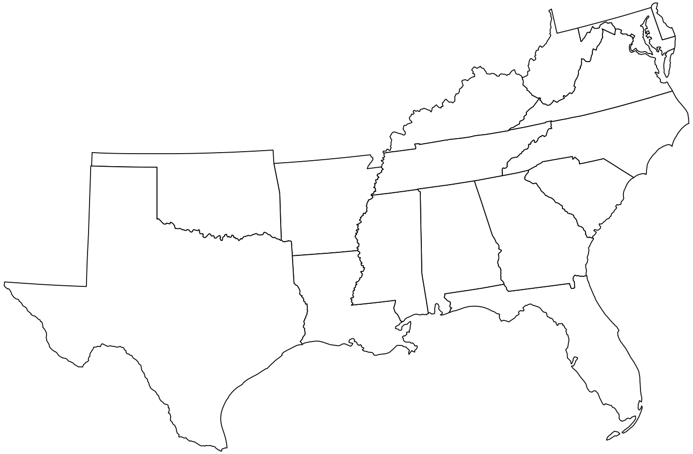
Spatial Data



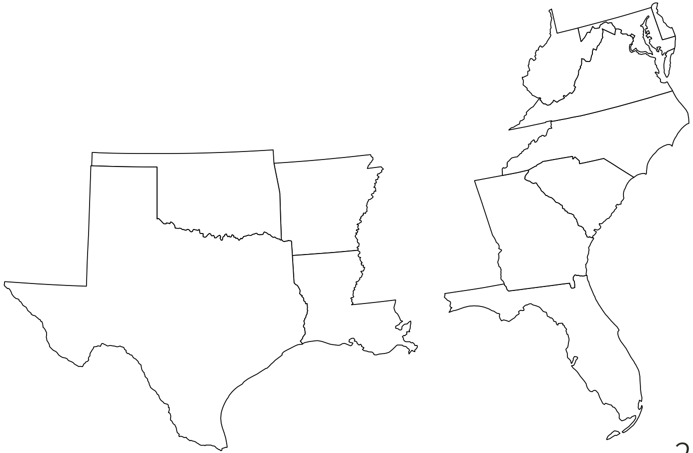
Spatial Data

Built-in Regions `usmap` provides some built-in regions based on the US Census Bureau Regions and Divisions. These can be used in place of the include/exclude parameters when using us_map or plot_usmap and start with a . (dot). For example:

```r
usmap::plot_usmap(
  include = .south_region
  )
```



```r
usmap::plot_usmap(
  include = .south_region,
  exclude = .east_south_central)
```



For Loops

Sometime you may find yourself copying and pasting code just to change one part each time.

This can be tedious and leaves a lot of room for errors.

One way we can simplify this and avoid errors is with a **for** loop.

For Loops

Let's start with some basics

```
for(i in sequence){
  expression
}
```

For example, we can take a character vector and use a `for` loop and `nchar()` to print the number of characters in each string.

```
cities <- c("New York", "Paris", "London")
# The copy-paste version of this would be:
nchar(cities[1])
nchar(cities[2])
nchar(cities[3])
# The for loop version is:
for(city in cities){
  print(nchar(city))
}
```

For Loops

```
cities ← c("New York", "Paris", "London")

# The for loop version is:
for(city in cities){
  print(nchar(city))
}
```

```
#> [1] 8
#> [1] 5
#> [1] 6
```

We can also use for loops for mathematical expressions.

```
numbers = c(10, 2, 23, 15)
```

Let's print the results when we add 5 to each element of the vector

```
for(i in numbers){
  print(i + 5)
```

For Loops

We can also do more complicated expressions

Suppose you wanted to solve this sum

$$\sum_{n=1}^{100} n^2$$

We could write this out by hand

or we can use a loop

For Loops

$$\sum_{n=1}^{100} n^2$$

To do this we can make a vector of the n's and then do the sum.

```r
#This makes a vector of values 1,2,3, ... ,99,100
sum = seq(1, 100, 1)
square = c()
for(i in sum){
  #This makes a vector of values 1,4,9, ... 10000
  square[i] = i^2
  result = sum(square)
}

result
```

```
#> [1] 338350
```

For Loops

We can also combine for loops with if statements

```
numbers = c(10, 2, 23, 15)
```

Let's have R add 5 to the numbers less than 15 and print the results

```
for(i in numbers){
  ifelse(i < 15, print(i + 5), print(i))
}
```

```
#> [1] 15
#> [1] 7
#> [1] 23
#> [1] 15
```

Functions

Sometimes we may want to get results and be able to change certain entries

We have been using built in functions from R, but we can also make our own.

Functions

Let's make a function that adds 3 to any number we input

```
plus = function(x){
  x + 3
}
```

Functions

Now we can test it out

```r
plus(100)
```

```
#> [1] 103
```

```r
plus(25)
```

```
#> [1] 28
```

```r
plus(3)
```

```
#> [1] 6
```

Functions

We could also use a for loop with our function to add 3 to every number between 1 and 5

```
numbers = c(1, 2, 3, 4, 5)
answer = c()

for(i in numbers){
  answer[i] = plus(i)
}

answer
```

```
#> [1] 4 5 6 7 8
```