Introduction to R, Day 4 Handout

In this handout, we cover the following topics and R commands:

Topics

- Repeating Operations: Loops & tapply
- Introduction to Geospatial Analysis
- Discussion of Text Analysis and Network Analysis
- Using `dplyr` for data manipulation
- Introduction to R Markdown

R Commands (update)

- Using loops `for(i in X){}` to repeat operations
- Generating "container" vectors using `rep(NA , n.obs)` or `NULL`
- Applying functions by indexes using `tapply()`
- Using `cbind()` function to combine vector, matrix or data frame by columns.

Loops

- Sometimes, our analysis would require us to repeat the same operation over and over again where only small changes occur to the operations each time. The main goal of using a loop is to avoid writing many similar code chunks.
- For example, suppose that we have a dataset on protests across countries (N= 93) over time (N= 93*number of years for each country). And, suppose that we would like to find the number of protests for the earliest year the countries are in the dataset. If I only had 7 countries, we can consider writing nearly identical code chunks by only changing the name of the country 7 times. But, we would not want to the same thing 93 times!
- Loops allow us to perform the same operation over and over, through various subsets of the data. Loops are not the only function in R that would allow us to achieve this task. We will shortly see that `tapply()` will also help us to do a task that requires repetition differently.

```
container variable <- rep(NA, n.observations)

for (i.counter in X) {

command1...

command2...

...

}
```

- By loop, we refer to several distinct pieces of a code chunk that are wrapped up in parantheses with the expression `for`:

- The expression `for`. establishes the loop
- A `counter` controls the flow of the loop
- A `container` varaible. collects and stores output from the loop

Let's explore the structure of the loop with a simple example:

```
for (i.count in 1:10) {
    print(ifelse(i.count >= 3 & i.count <= 7, "true", "false"))
}
```

```
## [1] "false"
## [1] "false"
## [1] "true"
## [1] "true"
## [1] "true"
## [1] "true"
## [1] "true"
## [1] "false"
## [1] "false"
## [1] "false"
```

- The expression `for` in this command establishes the loop. The counter is `i.count`. Under the hood, R thinks about the loop as follows:

1. Set `i.count` to 1.
2. Do whatever is in the brackets, i.e. test the logical statement stated, when it is TRUE assign "true, otherwise assign"false".
3. Set `i.count` to the next value.
4. If `i.count` is less than or equal to 10, go back to (2).
5. When `i.count` is greater than 10, complete the task.

⊙ Note: In loops, we generally reserve the letter `i` for the counter in a loop. To make it more intuitive, for example, if you are using a loop to do something for countries, you can name your counter as `i.country`.

- For our next example, we will use a subset of the Freedom House dataset, which includes freedom house democracy scores of countries over time. We will use a loop to explore the max freedom score of each country.

Below are the variables in the dataset:

Table 1: Freedom House Scores of Countries

- `country.` Country name - `region.` Region the country belongs to - `year.` year the report is released - `status.` country's freedom status: Free, Partly Free, or Not Free - `total.` country's freedom score ranges from 0 to 100

= Let's load the data and then ensure that we loaded it properly

```
fhdat <- read.csv("day4data/freedomhouse.csv")
head(fhdat)
```

```
##   X     country  region year status total
## 1 1    Abkhazia Eurasia 2020     PF    40
## 2 2 Afghanistan    Asia 2020     NF    27
## 3 3     Albania  Europe 2020     PF    67
## 4 4     Algeria    MENA 2020     NF    34
## 5 5     Andorra  Europe 2020      F    94
## 6 6      Angola     SSA 2020     NF    32
```

```
dim(fhdat)
```

```
## [1] 3129    6
```

```
summary(fhdat)
```

```
##        X              country          region         year      status
##  Min.   :   1   Abkhazia   :  15   Americas:536   Min.   :2006   F :1344
##  1st Qu.: 783   Afghanistan:  15   Asia    :645   1st Qu.:2009   NF: 826
##  Median :1565   Albania    :  15   Eurasia :250   Median :2013   PF: 959
##  Mean   :1565   Algeria    :  15   Europe  :642   Mean   :2013
##  3rd Qu.:2347   Andorra    :  15   MENA    :315   3rd Qu.:2017
##  Max.   :3129   Angola     :  15   SSA     :741   Max.   :2020
```

```
##                 (Other)    :3039
##      total
##  Min.   : -1.00
##  1st Qu.: 33.00
##  Median : 62.00
##  Mean   : 59.09
##  3rd Qu.: 88.00
##  Max.   :100.00
##
```

Declaring a New Variable inside a Loop

- Now we will create a loop to explore the max freedom score of each country. We will go over the distinct pieces of the loop step by step.

```
for (i.country in unique(fhdat$country)) {
    fhdat$maxfh[fhdat$country == i.country] <- max(fhdat$total[fhdat$country ==
        i.country])
}
fhdat$maxfh[1:10]
```

```
##  [1] 42 35 68 36 96 32 85 85 53 98
```

- Let's check whether our loop is working properly.

```
fhtable <- fhdat$maxfh
names(fhtable) <- unique(fhdat$country)
fhtable[1:10]
```

```
##            Abkhazia         Afghanistan             Albania             Algeria
##                  42                  35                  68                  36
##             Andorra              Angola Antigua and Barbuda           Argentina
##                  96                  32                  85                  85
##             Armenia           Australia
##                  53                  98
```

- It seems so, using a loop, we created a new variable called `maxfh`. And for each country, we calculated the max freedom house score that country achieved during the time period 2003 - 2020.

Declaring a New Variable with a Container

- It will not be always possible for us to create a new variable inside a loop. Instead of creating a new variable to store the values, we can save the output from the loop in a container. To do so, we must create the `container variable`. We need to declare the container *outside* the loop that is generating the output.

```
maxfh <- NULL
for (i.country in unique(fhdat$country)) {
    maxfh[fhdat$country == i.country] <- max(fhdat$total[fhdat$country ==
        i.country])
}
maxfh[1:10]
```

```
##  [1] 42 35 68 36 96 32 85 85 53 98
```

```
maxfh2 <- rep(NA, length(unique(fhdat$country)))
for (i.country in unique(fhdat$country)) {
    maxfh2[fhdat$country == i.country] <- max(fhdat$total[fhdat$country ==
        i.country])
}
maxfh2[1:10]
```

```
##  [1] 42 35 68 36 96 32 85 85 53 98
```

- We didn't have to worry about adding the information created by the loop to our data frame when we created a new variable. What about the container? How can we add the information stored by the container to our data frame? We will use `cbind()` to achieve this task, which will allow us to combine the information stored by the container (vector) and our data frame by columns.

```
fhdatnew <- cbind(fhdat, maxfh2)
fhdatnew[1:5, ]
```

```
##   X     country  region year status total maxfh maxfh2
## 1 1    Abkhazia Eurasia 2020     PF    40    42     42
## 2 2 Afghanistan    Asia 2020     NF    27    35     35
## 3 3     Albania  Europe 2020     PF    67    68     68
## 4 4     Algeria    MENA 2020     NF    34    36     36
## 5 5     Andorra  Europe 2020      F    94    96     96
```

- How to debug the code that involves loops? Since a loop simply executes the same commands many times, one should first check whether the commands that go inside of the loop can be executed without any error. We can check this for Argentina. In our loop, we find that Argentina's maximum freedom house score over the years is 85. Let's compute this for Argentina without a loop

```
argentina <- fhdat[fhdat$country == "Argentina", ]
argentinafh <- argentina$total
names(argentinafh) <- argentina$year
argentinafh
```

```
## 2020 2019 2018 2017 2016 2015 2014 2013 2012 2011 2010 2009 2008 2007 2006
##   85   84   83   82   79   80   80   80   81   80   80   80   80   81   84
```

```
max(argentinafh)
```

```
## [1] 85
```

- If there is something wrong with few observations in the dataset, the loop will stop working (I know this is frustrating...). We need to debug the problem by looking at the specific count that produces the error.

Figure A1: Source: Allison Horst

⊙ Note: I will be honest with you. When I first tried to do a loop for a problem set, I strugglet a lot. And, it took some time for me to digest what's going on. Please don't worry. It will make sense when you become comfortable with the structure.

Applying Functions by Indexes

- The function `tapply()` (`table apply`) is another way we can can consider when we would like to do a task that repeats itself with small changes.

  tapply(X, INDEX, FUN)

- Specifically, `tapply` function:

1. takes a vector X defined by a vector INDEX
2. applies the function FUN to X for each of the groups defined by a vector INDEX.

- You can replace the FUN with mean, median, sd, etc. to generate desired quantity.
- Let's do the same thing we did with the loop, but this time with tapply:

```
maxfhscores <- tapply(fhdat$total, fhdat$country, max)
maxfhscores[1:10]
```

```
##           Abkhazia        Afghanistan            Albania            Algeria
##                 42                 35                 68                 36
```

```
##             Andorra              Angola Antigua and Barbuda           Argentina
##                  96                  32                  85                  85
##             Armenia           Australia
##                  53                  98
```

☋ Short Practice Exercises

1. Calculate the mean of total Freedom House scores by country using `tapply()`
2. Calculate the mean of total Freedom House scores using a loop. Store the result by declaring an empty container variable outside the loop.

Geospatial Analysis

Many important aspects of politics are *inherently* spatial. Governing jurisdictions within some countries (e.g. states, counties, school districts etc.) are defined by clear geographical boundries. Governors, mayors, state representatives, and city council members represent geographically and spatially defined areas. These domestic boundries have political implications, which in turn led researchers to explore and explain a wide range of topics using spatial data: voting patterns and behaviour, the spatial distribution of political donations, the coordination of strategies in political campaigns, the unevenness of democratization, diffusion of protests, the spatial distribution of insurgent attacks. Moving beyond domestic politics, countries are defined by clear international borders. Despite the globalization of politics and economics, politics across borders and territorial conflicts continue to be a salient aspect of contemporary politics and pollicy. [1]

- So far, we focused on learning R using traditional data sets and models. R can also be used to analyze and explore many kinds of new data including spatial data.
- A very effective way to analyze spatial data is by visualization through maps. [2]

Spatial data contain information about patterns over space and can be visualized through maps.

- `spatial point data`. represent the locations of events as points on a map
- `spatial polygon data`. represent geographical areas as polygons by connecting points on a map

`spatial temporal data`. represent a sequence of connected points on a map corresponding to the boundaries of certain areas such as counties, districts, and provinces.

- Similar to yesterday, we will go beyond `base R` and use some powerful R packages that will allow us to do spatial analysis.
- Before we start, let's install and load the following packages.

- To install a package:
> install.packages("packagename")
- To load the package
> library("packagename")

- Install and load the following packages

```
library(maps)
library(sf)
```

```
## Linking to GEOS 3.7.2, GDAL 2.4.2, PROJ 5.2.0
```

[1] Simmons, B.A. and Kenwick, M., 2019. Border Orientation in a Globalizing World: Concept and Measurement. Good news! If you are interested in border politics, Prof. Kenwick is in our department!

[2] Our introduction is limited to using spatial analysis for *exploration*. Researchers also use spatial analysis for statistical analysis (spatial statistics) and making causal inferences.

```
library(sp)
library(tmap)
library(ggplot2)
```

- Let's check whether we were succesful in loading our packages

```
search()
```

```
##  [1] ".GlobalEnv"        "package:ggplot2"   "package:tmap"
##  [4] "package:sp"        "package:sf"        "package:maps"
##  [7] "package:formatR"   "package:knitr"     "package:foreign"
## [10] "package:stats"     "package:graphics"  "package:grDevices"
## [13] "package:utils"     "package:datasets"  "package:methods"
## [16] "Autoloads"         "package:base"
```

Getting Familiar with Maps: U.S. cities

- We will use a built-in dataset from the `maps package` named `us.cities`. Let's load the data:

```
data(us.cities)
```

- Let's whether dataset is loaded properly. Specifically, check the first few rows, obtain a summary of the variables, and check the dimension of the dataset.

```
head(us.cities)
```

```
##         name country.etc    pop   lat    long capital
## 1 Abilene TX          TX 113888 32.45  -99.74       0
## 2   Akron OH          OH 206634 41.08  -81.52       0
## 3 Alameda CA          CA  70069 37.77 -122.26       0
## 4  Albany GA          GA  75510 31.58  -84.18       0
## 5  Albany NY          NY  93576 42.67  -73.80       2
## 6  Albany OR          OR  45535 44.62 -123.09       0
```

```
summary(us.cities)
```

```
##      name           country.etc             pop               lat
##  Length:1005        Length:1005        Min.   :   8003   Min.   :19.70
##  Class :character   Class :character   1st Qu.:  50116   1st Qu.:33.87
##  Mode  :character   Mode  :character   Median :  66628   Median :38.51
##                                        Mean   : 125548   Mean   :37.50
##                                        3rd Qu.: 103782   3rd Qu.:41.60
##                                        Max.   :8124427   Max.   :61.18
##       long            capital
##  Min.   :-157.80   Min.   :0.0000
##  1st Qu.:-116.40   1st Qu.:0.0000
##  Median : -90.21   Median :0.0000
##  Mean   : -95.10   Mean   :0.0995
##  3rd Qu.: -80.30   3rd Qu.:0.0000
##  Max.   : -69.77   Max.   :2.0000
```

```r
dim(us.cities)
```

```
## [1] 1005    6
```

- Subset U.S. capitals

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

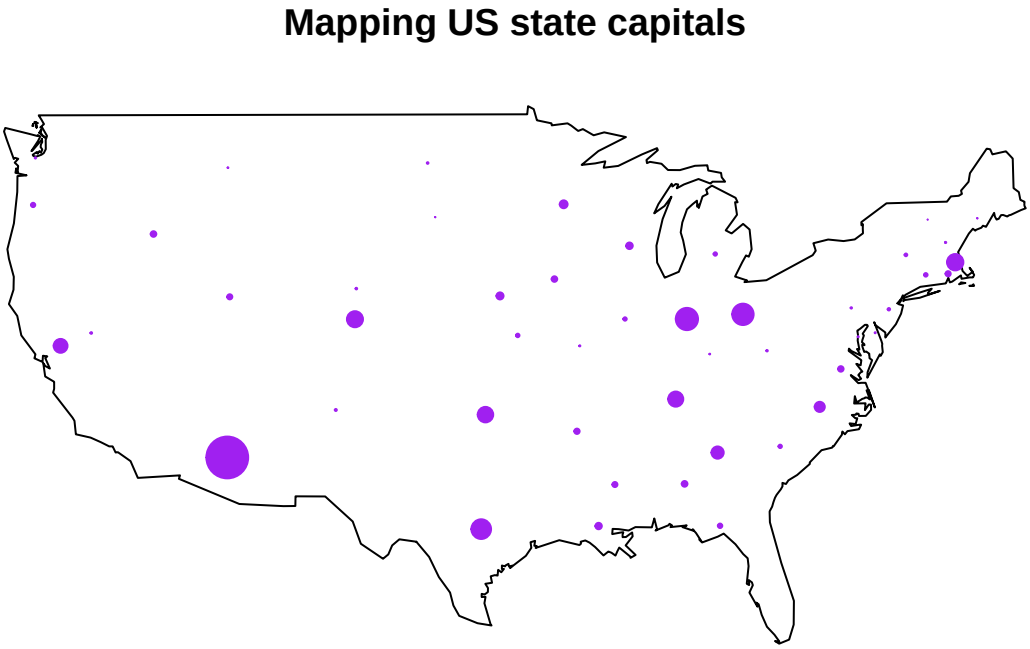
```
## [1] 50  6
```

- Now, we have information about the states and their coordinates. We need a map to visualize the capital cities! We will use the `map()` function to access a spatial database and visualize the data using this map.

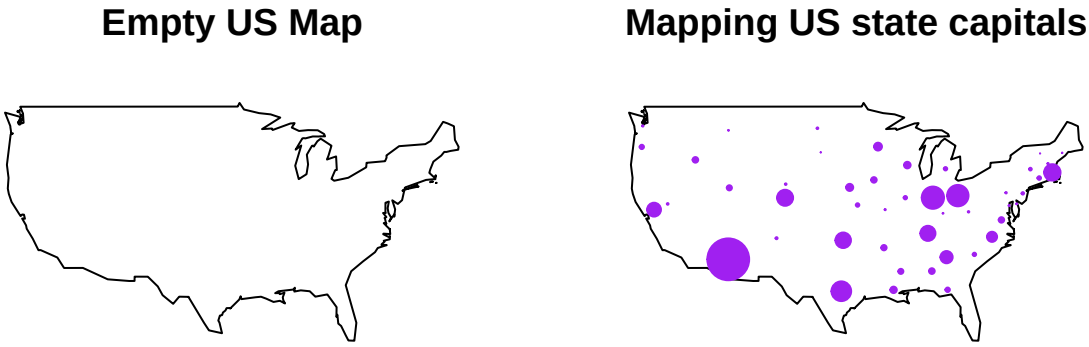
```r
map(database = "usa")
title("Empty US Map")
```

- We will visualize state capitals proportional to their population on the map of the US. To do so, we will tell R that we would like to add spatial points to the map using the `points()` function. We will feed the lattitude and longitude information of cities into the function as our input.

```r
## add points proportional to population using latitude
## and longitude
map(database = "usa")
points(x = capitals$long, y = capitals$lat, cex = capitals$pop/5e+05,
    pch = 19, col = "purple")
title("Mapping US state capitals")  # add a title
```




```
par(mfrow = c(1, 2))
map(database = "usa")
title("Empty US Map")
map(database = "usa")
points(x = capitals$long, y = capitals$lat, cex = capitals$pop/5e+05,
    pch = 19, col = "purple")
title("Mapping US state capitals")  # add a title
```



Geospatial Distribution of Republican/Democratic Vote-shares

data set: pres08.csv

- To explore how R can handle spatial data, our first running example will be an analysis of the geospatial distribution of Republican/Democratic vote-shares over the last century [3]
- Our analysis is an example of `spatial polygon data`, where each state represents a polygon whose boundaries can be constructed by connecting a series of points.

[3] Example adapted from Kosuke Imai, QSS, Chapter 5

Table 1: 2008 US Presidential Election Data

- `country`. Country
- `union`. Percentage of workers who belong to a union
- `left`. Extent to which parties of the left have controlled government
- `size`. Size of the labor force
-

`concent` Measure of economic concentration in top four industries

- Let's load the data

```
pres08 <- read.csv("day4data/pres08.csv")
head(pres08)
```

```
##   state.name state Obama McCain EV
## 1    Alabama    AL    39     60  9
## 2     Alaska    AK    38     59  3
## 3    Arizona    AZ    45     54 10
## 4   Arkansas    AR    39     59  6
## 5 California    CA    61     37 55
## 6   Colorado    CO    54     45  9
```

- Let's compute the two-party vote share for Democrats and Republicans and store this information by creating two new variables

```
## two-party vote share
pres08$Dem <- pres08$Obama/(pres08$Obama + pres08$McCain)
pres08$Rep <- pres08$McCain/(pres08$Obama + pres08$McCain)
head(pres08)
```

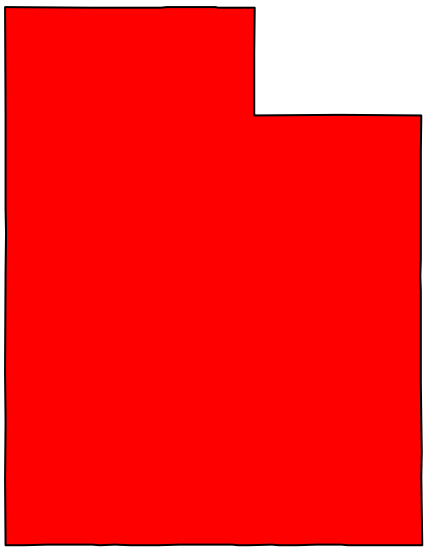
```
##   state.name state Obama McCain EV       Dem       Rep
## 1    Alabama    AL    39     60  9 0.3939394 0.6060606
## 2     Alaska    AK    38     59  3 0.3917526 0.6082474
## 3    Arizona    AZ    45     54 10 0.4545455 0.5454545
## 4   Arkansas    AR    39     59  6 0.3979592 0.6020408
## 5 California    CA    61     37 55 0.6224490 0.3775510
## 6   Colorado    CO    54     45  9 0.5454545 0.4545455
```

- Let's decide on how which colors we will use to color our map. For the time being, we will focus on only two states.
- We will color California as blue.

```
## California as a blue state
map(database = "state", regions = "California", col = "blue",
    fill = TRUE)
title("California Map")
```

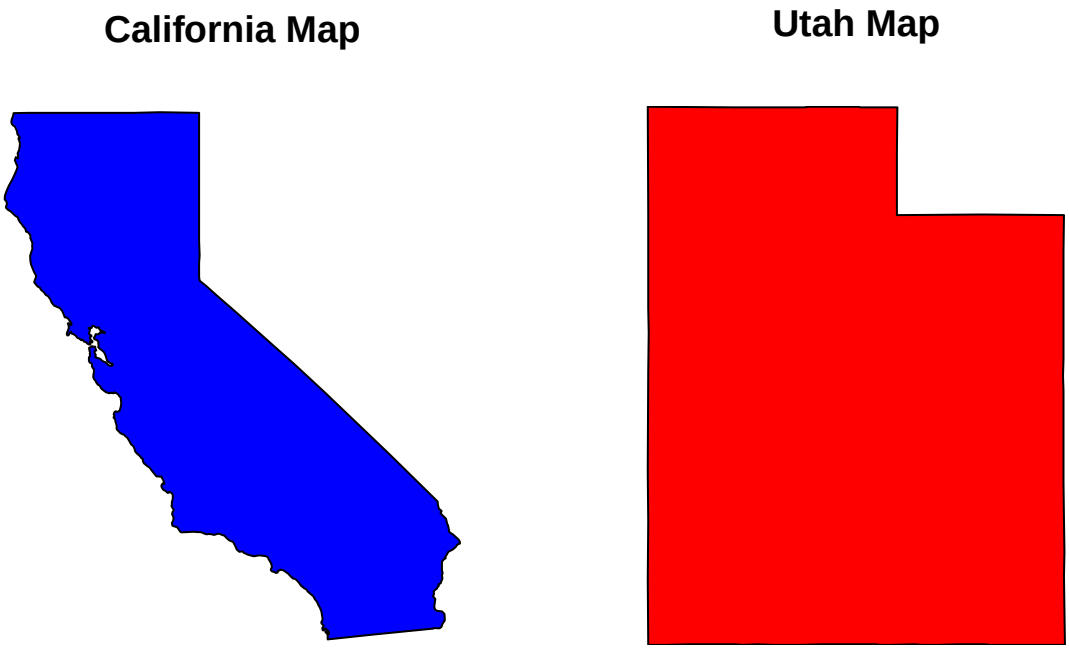
- We will color Utah as red.

```
## Utah as a red state
mapcol <- map(database = "state", regions = "Utah", col = "red",
    fill = TRUE)
title("Utah Map")
```



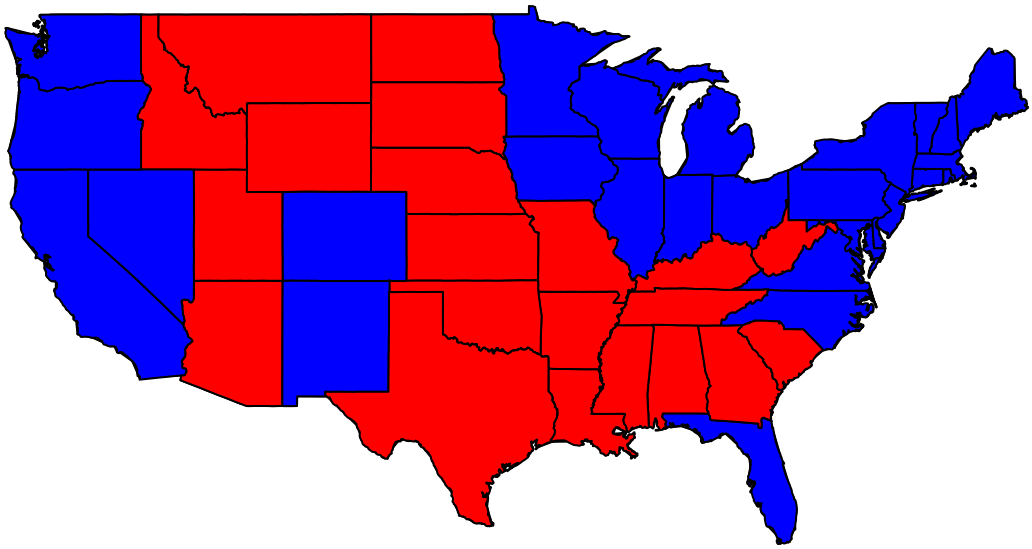
- Let's put these two maps side by side using the `par(mfrow)` function we learned yesterday

```r
par(mfrow = c(1, 2))
## California as a blue state
map(database = "state", regions = "California", col = "blue",
    fill = TRUE)
title("California Map")
## Utah as a red state
mapcol <- map(database = "state", regions = "Utah", col = "red",
    fill = TRUE)
title("Utah Map")
```



- Now let's generalize what we did for California and Utah for all states.
- We will use a loop to repeat this task over and over again. . . for all US states. Otherwise, we need to write a similar code code for each states. That would be a pain. The map does not include Hawaii, Alaska, and Washington DC, so we will skip those states. If we don't add this to the loop, we will hit an error.
- We first use a `dichotomized color scheme` (blue vs. red) where the states Obama won appear blue and those won by McCain are shown as red.

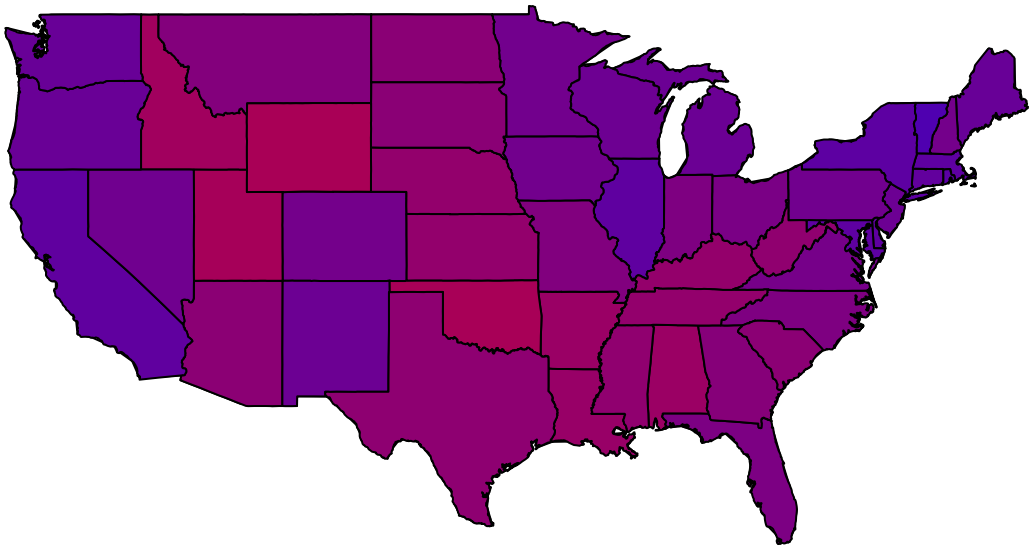
```r
## Create a map
map(database = "state")
## Run a loop to categorize each state by color
for (i in 1:nrow(pres08)) {
    if ((pres08$state[i] != "HI") & (pres08$state[i] !=
        "AK") & (pres08$state[i] != "DC")) {
        maps::map(database = "state", regions = pres08$state.name[i],
            col = ifelse(pres08$Rep[i] > pres08$Dem[i],
                "red", "blue"), fill = TRUE, add = TRUE)
    }
}
title("2008 US Presidential Election Outcomes")
```



- Next, we will use the `RGB color scheme` based on the two-party vote share for each state. The loop will almost the same. The only difference is the way in which color is chosen for each state.

⊙ In R, the `rgb()` function helps create hexadecimal color codes from numerical values. The three arguments, red, green, and blue, take the intensity of each color, ranging from 0 to 1, which gets translated into an integer value from 0 to 255 and then represented as a hexadecimal numeral.

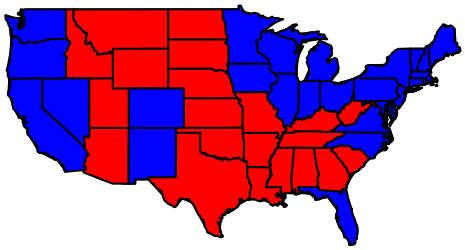
```
map(database = "state")  # create a map
for (i in 1:nrow(pres08)) {
    if ((pres08$state[i] != "HI") & (pres08$state[i] !=
        "AK") & (pres08$state[i] != "DC")) {
        map(database = "state", regions = pres08$state.name[i],
            col = rgb(red = pres08$Rep[i], blue = pres08$Dem[i],
                green = 0), fill = TRUE, add = TRUE)
    }
}
title("2008 US Presidential Election Vote-Shares")
```



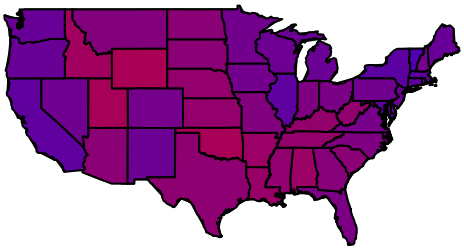
- Let's put these two maps side by side the `par(mfrow())` function we learned yesterday.

```
par(mfrow = c(1, 2))
map(database = "state")  # create a map
## Run a loop to categorize each state by color
for (i in 1:nrow(pres08)) {
    if ((pres08$state[i] != "HI") & (pres08$state[i] !=
        "AK") & (pres08$state[i] != "DC")) {
        maps::map(database = "state", regions = pres08$state.name[i],
            col = ifelse(pres08$Rep[i] > pres08$Dem[i],
                "red", "blue"), fill = TRUE, add = TRUE)
    }
}
title("2008 US Presidential Election Outcomes", cex.main = 0.8)
map(database = "state")  # create a map
for (i in 1:nrow(pres08)) {
    if ((pres08$state[i] != "HI") & (pres08$state[i] !=
        "AK") & (pres08$state[i] != "DC")) {
        maps::map(database = "state", regions = pres08$state.name[i],
            col = rgb(red = pres08$Rep[i], blue = pres08$Dem[i],
                green = 0), fill = TRUE, add = TRUE)
    }
}
title("2008 US Presidential Election Vote-Shares", cex.main = 0.8)
```

2008 US Presidential Election Outcomes



2008 US Presidential Election Vote−Shares



- The left-hand map shows that Obama won many states on the West and East Coasts whereas McCain was particularly strong in the Midwest. However, the right-hand map illustrates that no state is completely dominated by either Democrats or Republicans. Each state has both types of voters, and it is the winner-take-all electoral system that is responsible for characterizing each state as either a blue or a red state.

Geo-referencing Ethnic Power Relations (GeoEPR)

- For the previous example, we access a map using R's `maps package` which includes spatial datasets. But, most of the time we won't be able to find what we are looking for in this package.
- To explore how datasets with spatial attributes looks like, we will use a dataset called `GeoEPR 2019`. GeoEPR dataset codes the settlement patterns of politically relevant ethnic groups in independent states from 1946-2017.
- In our dataset folder for Day 4, we need to put a folder that includes 5 different data files with extensions: `.dbf`, `.fix`, `.prj`, `.shp` and `.shx`. We don't need to worry about the details of these datafiles. We only need to know that we need to have all of these data files together when we are working with spatial datasets. The data file we will load to R will be the one with the extension `.shp`. This is a data file in a `shapefile` format. This format allows us to store the geometric location and attribute information of geographic features.
- Let's load our spatial dataset. To read a shape file, we will use the `st_read()` function.

```
epr <- st_read("day4data/GEOEPR-2019/GeoEPR.shp")
```

```
## Reading layer 'GeoEPR' from data source '/Users/burcukolcak/Desktop/RWorkshop2021/RWDay4/day4data/Ge
## replacing null geometries with empty geometries
## Simple feature collection with 1470 features and 10 fields (with 134 geometries empty)
## geometry type:  GEOMETRY
## dimension:      XY
## bbox:           xmin: -180 ymin: -55.31195 xmax: 180 ymax: 76.99887
## CRS:            4326
```

Class of datasets with geometry features

- Recall our discussion of types of objects in R. Previously, we covered various objects including `data.frame`, `numeric`, `character`, `logical`.
- To be able to conduct spatial analysis, R will treat datasets with geospatial analysis differently by classifying them under a different type of object. The class of a geospatial dataset will not be a `data frame`.

```r
class(epr)
```

```
## [1] "sf"         "data.frame"
```

- `sf` is the abbreviation of `shape file`. The class of this dataset tells us that this is a dataset with geometry features. Many people refer to such datasets as `spatial data frames`.
- Previously, we learned that spatial data can be represented in various ways. We can check the type of our geometry by using `st_geometry()`

Spatial Data Frames and Geometry Types

```r
st_geometry(epr)
```

```
## Geometry set for 1470 features  (with 134 geometries empty)
## geometry type:  GEOMETRY
## dimension:      XY
## bbox:           xmin: -180 ymin: -55.31195 xmax: 180 ymax: 76.99887
## CRS:            4326
## First 5 geometries:

## MULTIPOLYGON (((-95.08186 49.35959, -95.02584 4...

## MULTIPOLYGON (((-73.17778 41.17083, -73.15684 4...

## MULTIPOLYGON (((-73.91165 45, -73.8639 44.81884...

## MULTIPOLYGON (((-95.08186 49.35959, -95.02584 4...

## MULTIPOLYGON (((-73.91165 45, -73.8639 44.81884...
```

- When we look at the first 5 geometries, we can tell that this countries in this dataset are stored as `spatial polygon data` representing geographical areas as polygons by connecting points on a map

Spatial Data Frames and Projection

- Map projections provide methods for representing the three-dimensional surface of the earth as two-dimensional plots.
- Projection is an issue that researchers who use spatial data constantly think about. For example, when we want to do operations to calculate distances, or densities, we want a projection that uses an `equal area projection`. Otherwise, our findings will be not accurate.

- The function `st_crs()` allows us to deal with the coordinate system, which includes both projection and extent of the map

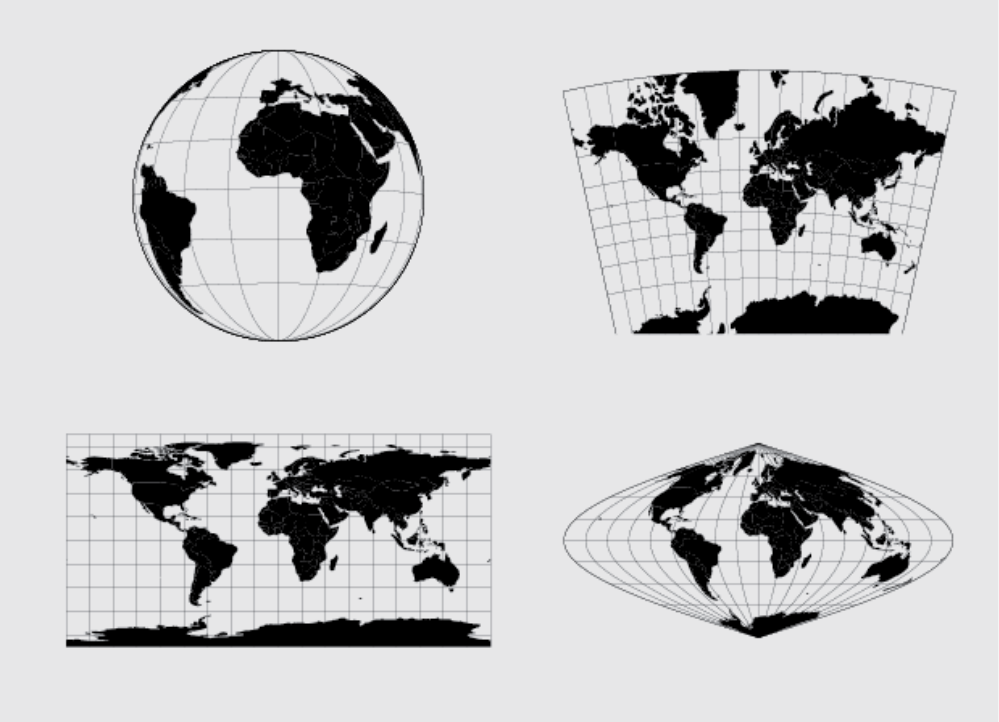


Figure A2: Source: Allison Horst

```
st_crs(epr)
```

```
## Coordinate Reference System:
##   User input: 4326
##   wkt:
## GEOGCS["WGS84(DD)",
##     DATUM["WGS84",
##         SPHEROID["WGS84",6378137.0,298.257223563]],
##     PRIMEM["Greenwich",0.0],
##     UNIT["degree",0.0174532925199433295],
##     AXIS["Geodetic longitude",EAST],
##     AXIS["Geodetic latitude",NORTH]]
```

- Let's play around with the geoepr dataset. Our aim will be to map settlement patterns of ethnic groups for a specific country for a specific time period.
- First, let's check how many unique states are included in this dataset.

```
states <- unique(epr$statename)
```

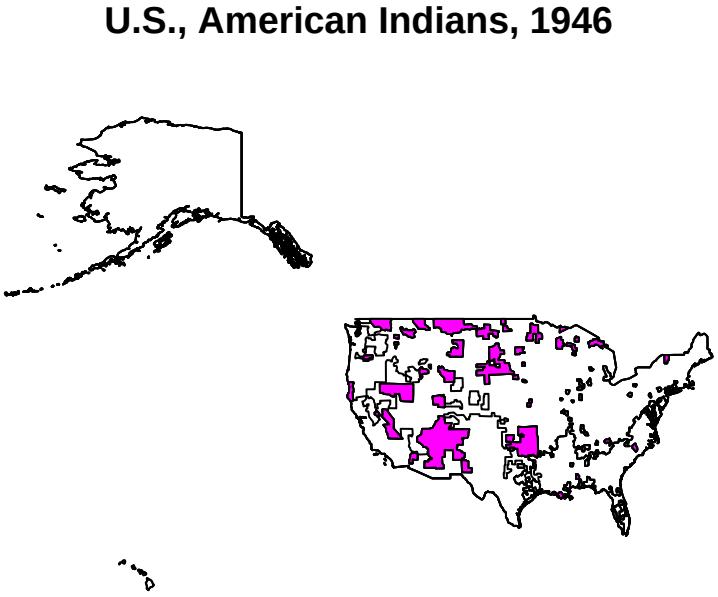
- We will map a African Americans' settlement pattern in the United States over different time periods. To do so, we need to subset the data.

```
epr_us <- epr[epr$statename == "United States", ]
epr_us
```

```
## Simple feature collection with 12 features and 10 fields (with 2 geometries empty)
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -178.2165 ymin: 18.92548 xmax: -66.97084 ymax: 71.35144
## CRS:            4326
## First 10 features:
##    gwid     statename from   to             group groupid gwgroupid umbrella
## 1     2 United States 1946 1958            Whites    1000    201000       NA
## 2     2 United States 1946 1958 African Americans    3000    203000       NA
## 3     2 United States 1946 1958  American Indians    5000    205000       NA
## 4     2 United States 1959 1959            Whites    1000    201000       NA
## 5     2 United States 1959 1959  American Indians    5000    205000       NA
## 6     2 United States 1959 1959 African Americans    3000    203000       NA
## 7     2 United States 1960 2017            Whites    1000    201000       NA
## 8     2 United States 1960 2017  American Indians    5000    205000       NA
## 9     2 United States 1960 2017 African Americans    3000    203000       NA
## 10    2 United States 1966 2017   Asian Americans    4000    204000       NA
##       sqkm             type                       geometry
## 1  7940038        Statewide MULTIPOLYGON (((-95.08186 4...
## 2  1037020        Statewide MULTIPOLYGON (((-73.17778 4...
## 3   939761 Regionally based MULTIPOLYGON (((-73.91165 4...
## 4  9462968        Statewide MULTIPOLYGON (((-95.08186 4...
## 5  2444822 Regionally based MULTIPOLYGON (((-73.91165 4...
## 6  1037020        Statewide MULTIPOLYGON (((-73.17778 4...
## 7  9462968        Statewide MULTIPOLYGON (((-95.08186 4...
## 8  2444822 Regionally based MULTIPOLYGON (((-73.91165 4...
## 9  1037020        Statewide MULTIPOLYGON (((-73.17778 4...
## 10      NA            Urban             MULTIPOLYGON EMPTY
```

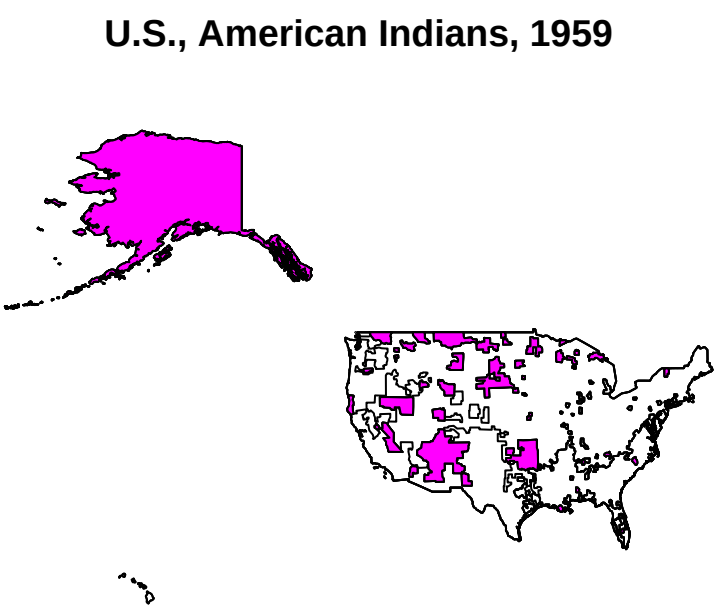
- First let's map the settlement of African Americans in 1946-1958.

```
epr_us1 <- epr_us[epr_us$from >= 1946 & epr_us$to <= 1958 &
    epr_us$group == "American Indians", ]
plot(epr_us$geometry, main = "U.S., American Indians, 1946")
plot(epr_us1$geometry, col = "magenta", add = T)
```



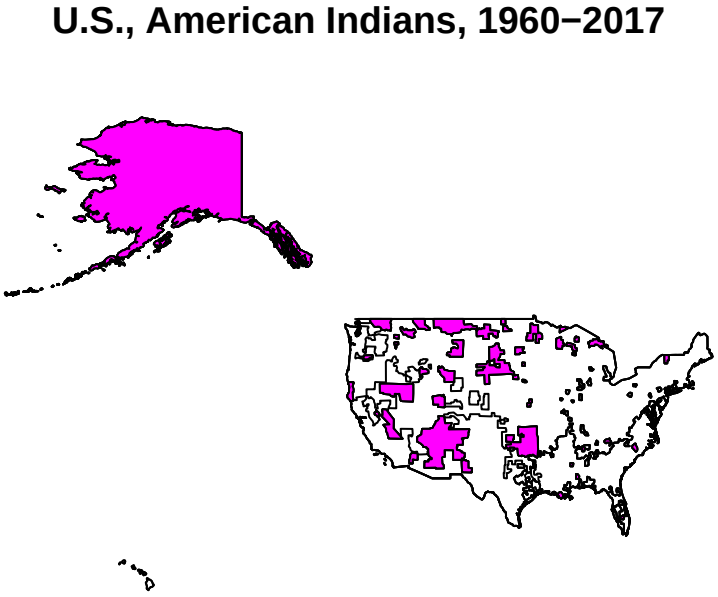
- Second, let's map the settlement of African Americans in 1959

```
epr_us2 <- epr_us[epr_us$from >= 1959 & epr_us$to <= 1959 &
    epr_us$group == "American Indians", ]
plot(epr_us$geometry, main = "U.S., American Indians, 1959")
plot(epr_us2$geometry, col = "magenta", add = T)
```



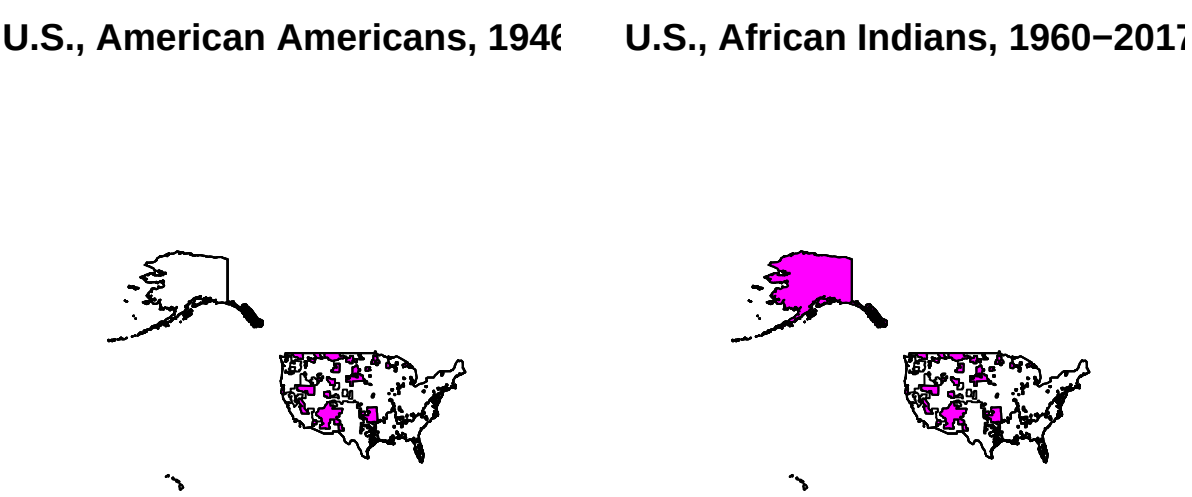
- Lastly, let's map the settlement of African Americans in 1960-2017.

```
epr_us3 <- epr_us[epr_us$from >= 1960 & epr_us$group ==
    "American Indians", ]
plot(epr_us$geometry, main = "U.S., American Indians, 1960-2017")
plot(epr_us3$geometry, col = "magenta", add = T)
```



- Let's put the first and last figures into one figure using `par(mfrow())`.

```
par(mfrow = c(1, 2))
plot(epr_us$geometry, main = "U.S., American Americans, 1946")
plot(epr_us1$geometry, col = "magenta", add = T)
plot(epr_us$geometry, main = "U.S., African Indians, 1960-2017")
plot(epr_us3$geometry, col = "magenta", add = T)
```

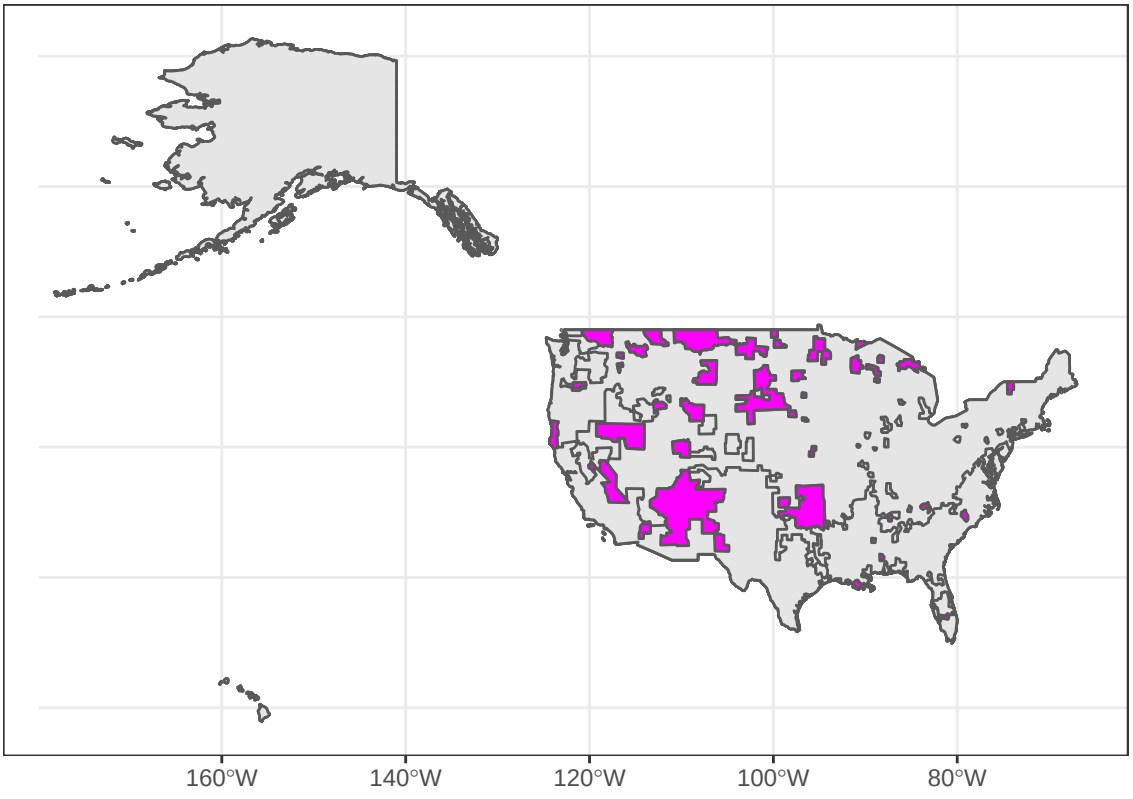


- Good news! 'ggplot() is also compatible with data frames with spatial attributes. If you liked it more, you can use that as well.

Yesterday, we went over the main steps for creating visualizations with `ggplot()`. The structure of ggplot will differ when we use it to create maps. For example, unlike histograms or scatterplots, we won't write feed into variables into the main part.

```
ggplot(epr_us) + geom_sf() + geom_sf(data = epr_us1, fill = "magenta") +
    labs(title = "US American Indians, 1946") + theme_bw()
```

US American Indians, 1946



℧ Short Practice Exercises

1. Let's pick another state and map the ethnic settlement of a specific ethnic group over time. Alternatively, you can map the settlement of ethnic groups for all ethnic groups at one point. You can either use `plot()` or `ggplot()`

R Markdown

Figure A3: Source: Allison Horst

Discussion of Text Data and Network Data and R Resources

Data Manipulation with `dplyr` verbs

- If time permits

Practice Exercises

- Might not have for the last day