Lecture 5. Maps

PUBH 6199: Visualizing Data with R, Summer 2025

Xindi (Cindy) Hu, ScD

2025-06-17

Outline for today

- Types of maps for spatial data
- Map design considerations (color, projection, legend)
- R packages for mapping
- Static maps with `tmap`
- Interactive maps with `leaflet`
- Accessing spatial data with `tigris` and `tidycensus`

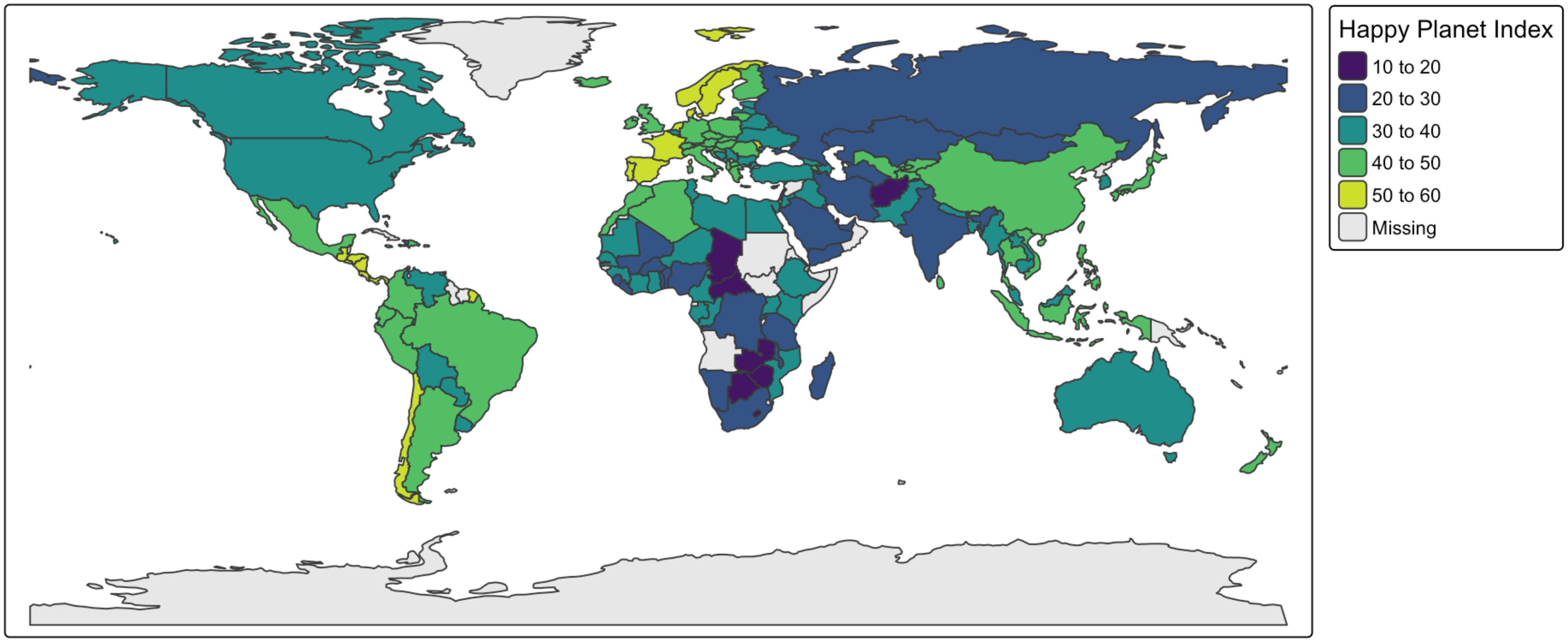
Types of Maps for Spatial Data

1. Choropleth Map

- Display the spatial distribution of a variable across divided geographical areas
- Best for **normalized** data (e.g., rates)
- Variable encoding: **color** (intensity or hue)

Example: Choropleth

```r
library(tmap)
data("World")
tm_shape(World) +
  tm_polygons("HPI", palette = "viridis", title = "Happy Planet Index")
```

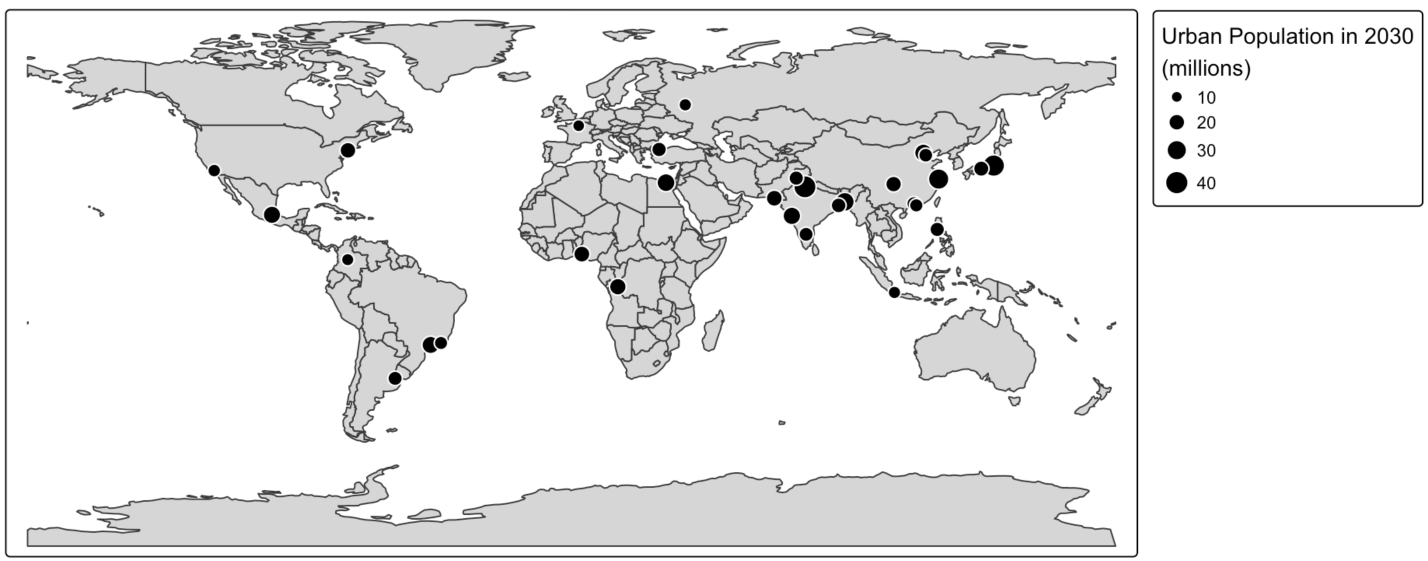


2. Point Map

- Each point represents a location
- Can show locations of events, facilities, cases
- Variable encoding: **color**, **shape**, **size**

Example: Point Map

```r
library(tidyverse)
library(spData)
library(tmap)
data(urban_agglomerations)
urb_2030 <- urban_agglomerations |> filter(year == 2030)
tm_shape(World) +
  tm_polygons() +
  tm_shape(urb_2030) +
  tm_symbols(fill = "black", col = "white", size = "population_millions",
             size.legend = tm_legend(title = "Urban Population in 2030\n(millions)"))
```

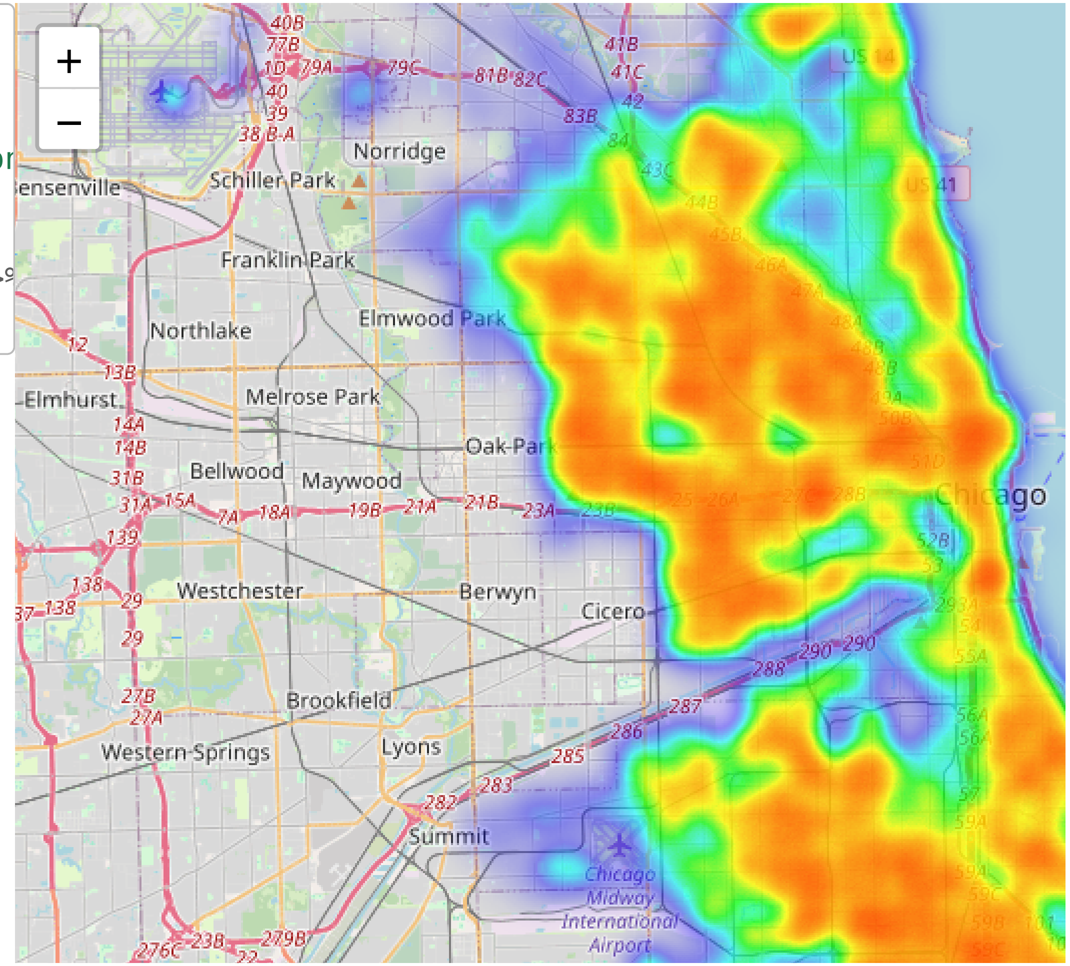


3. Heat Map (Density)

- Visualizes concentration of events
- Uses kernel density estimation or hex bins
- Variable encoding: **color** (intensity)

Example: Heat map with `leaflet.extras`

```r
library(leaflet)
library(leaflet.extras)
thefts_coords <- read_csv("data/thefts_coords.csv")
thefts_sf <- st_as_sf(thefts_coords, coords = c("lon
leaflet(thefts_sf) %>%
  addProviderTiles("OpenStreetMap") %>%
  addHeatmap(blur = 15, max = 0.05, radius = 10) %>%
  setView(lng = -87.6298, lat = 41.8781, zoom = 11)
```

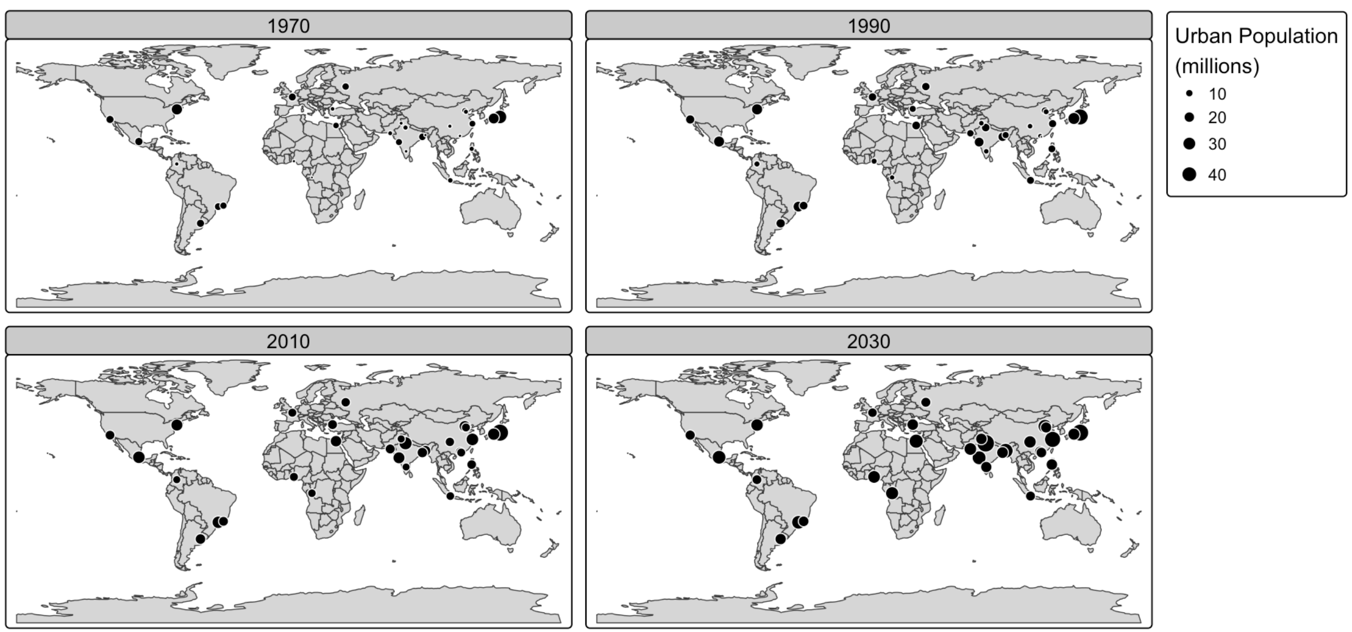


4. Faceted Map

- Create small multiples to compare across categories
- Great for time series or group comparisons

Example: Faceted Map

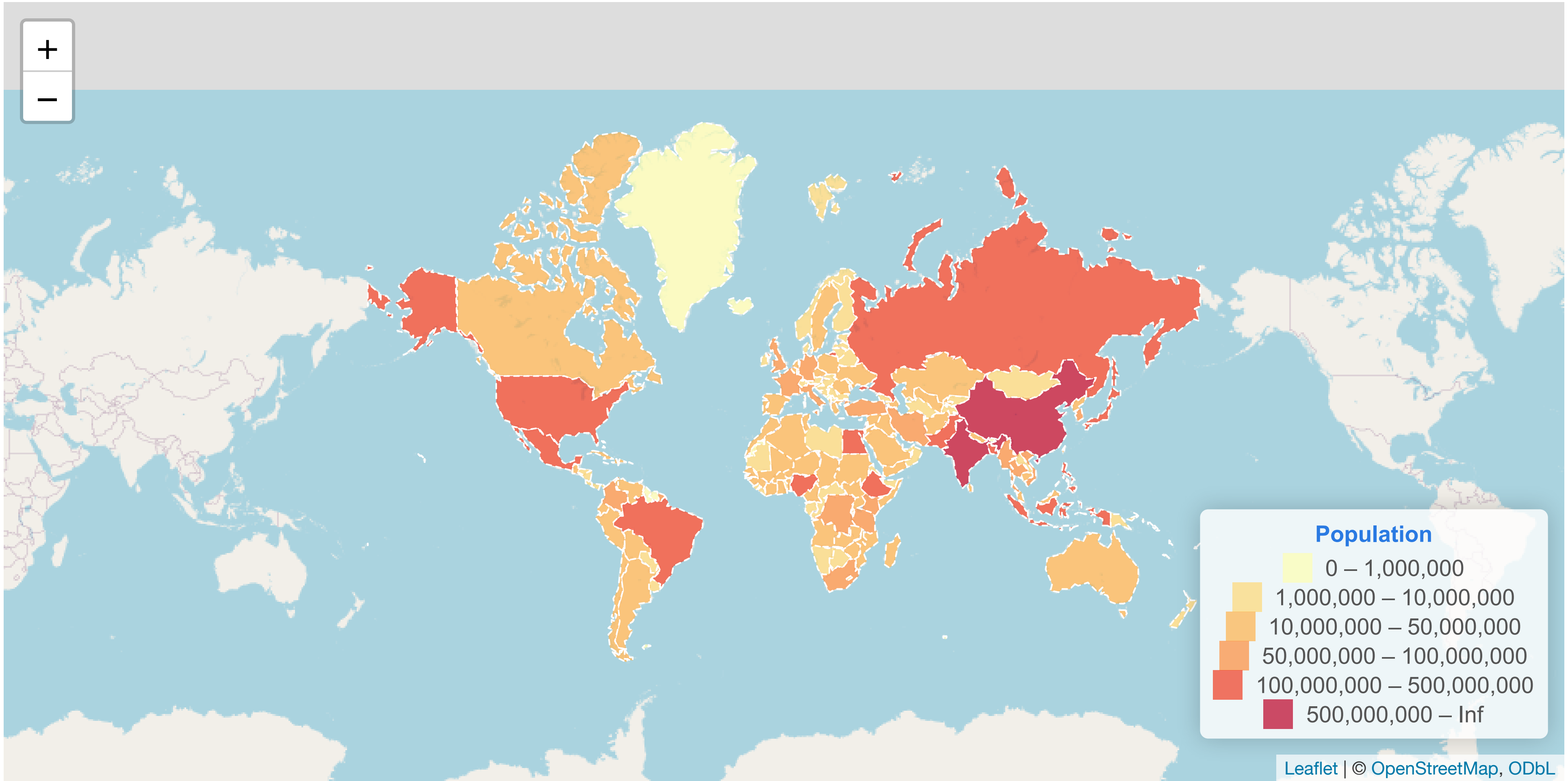
```r
library(tidyverse)
library(spData)
library(tmap)
data(urban_agglomerations)
urb_1970_2030 <- urban_agglomerations |> filter(year %in% c(1970, 1990, 2010, 2030))
tm_shape(World) +
  tm_polygons() +
  tm_shape(urb_1970_2030) +
  tm_symbols(fill = "black", col = "white", size = "population_millions",
             size.legend = tm_legend(title = "Urban Population\n(millions)")) +
  tm_facets(by = "year", ncol = 2)
```



5. Interactive Map

- Enable zooming, hovering, filtering
- Useful for dashboards, web apps

Example: Interactive Map

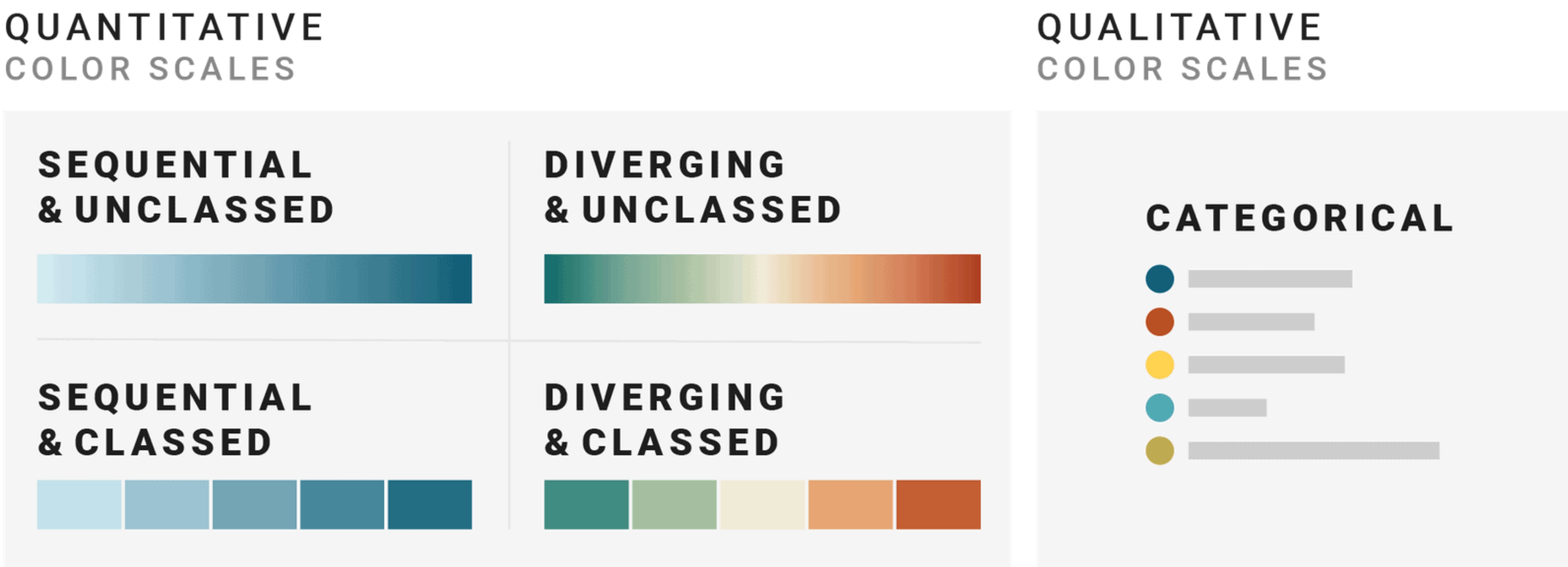


Outline for today

- Types of maps for spatial data
- **Map design considerations (color, scale, projection, legend)**
- R packages for mapping
- Static maps with `tmap`
- Interactive maps with `leaflet`
- Accessing spatial data with `tigris` and `tidycensus`

Color Choices for Maps

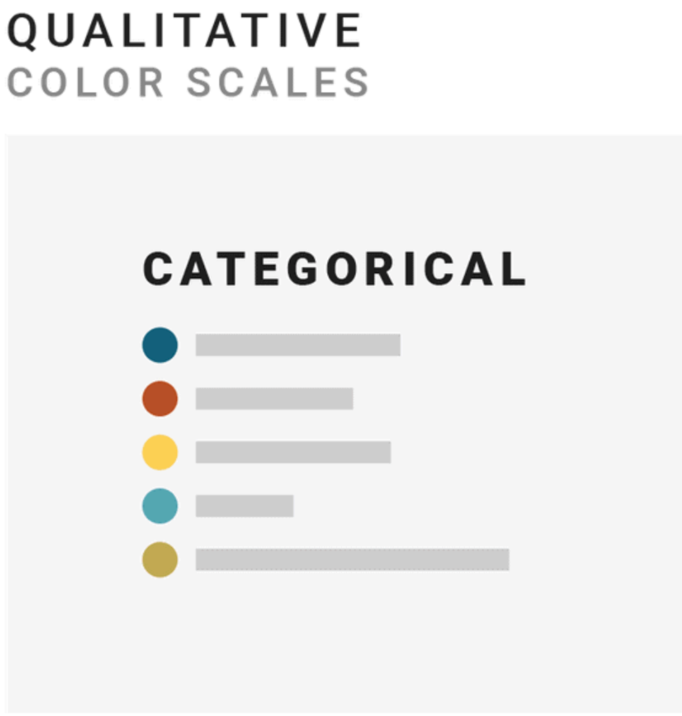
- **Sequential** palettes for ordered values
- **Diverging** palettes for above/below mean
- **Qualitative** for categories

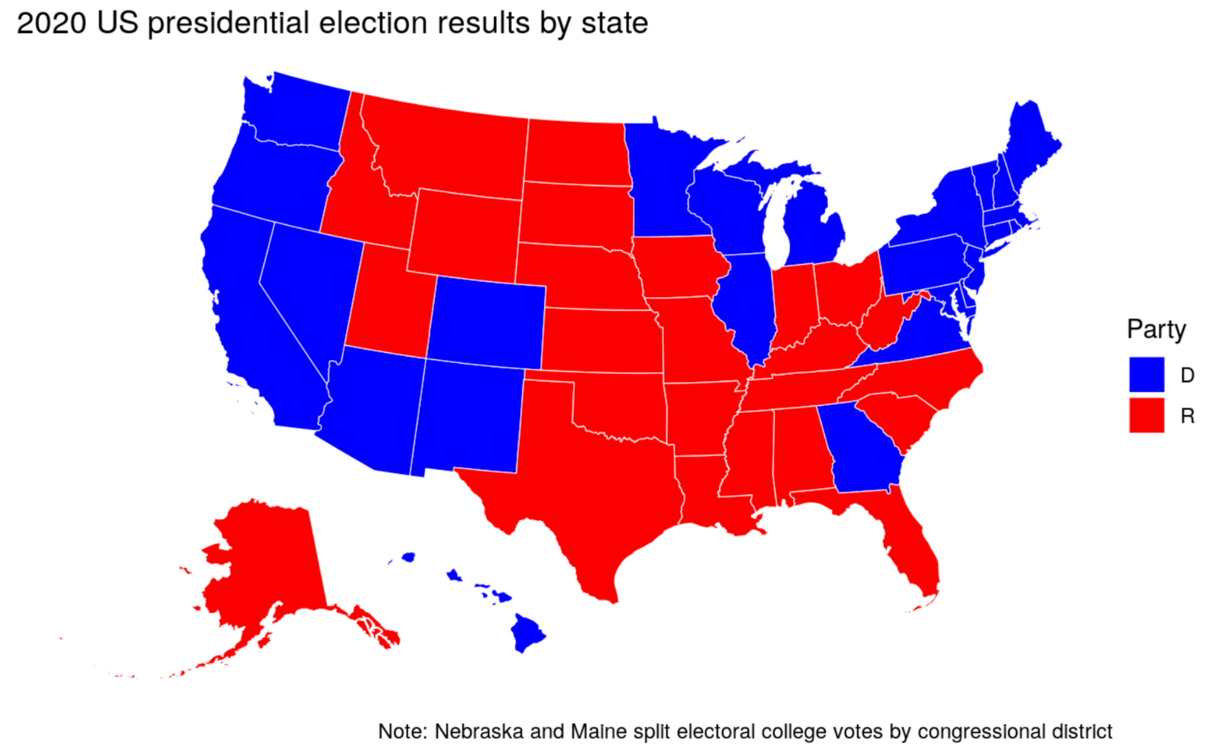


Source: [Which color scale to use when visualizing data](), by Lisa Charlottte Muth.

Categorical scales

- Use distinct **hues** for different categories
- Limit to **no more than 7 hues**

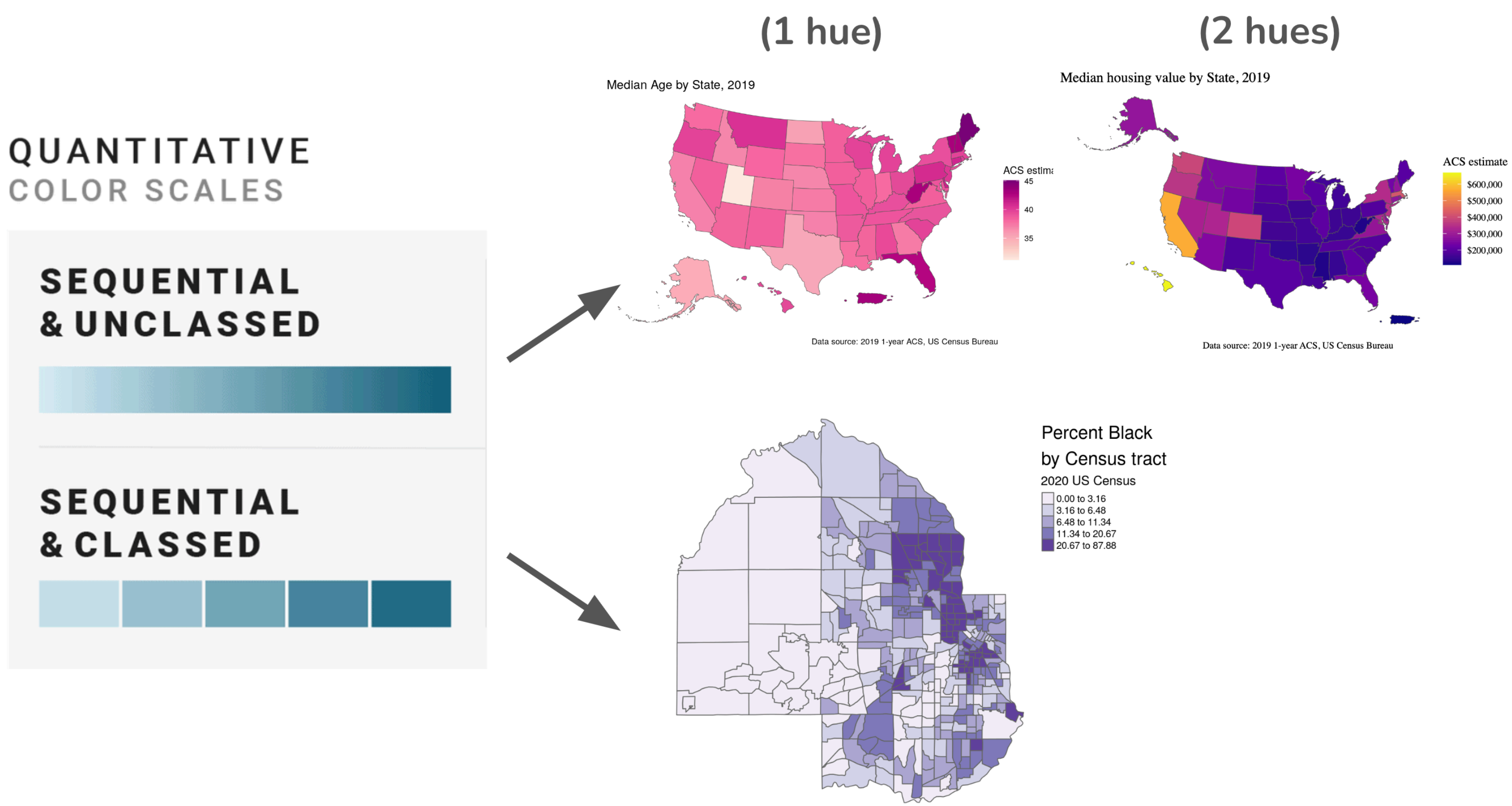




Source: Analyzing US Census Data, by Kyle Walker

Sequential Scales

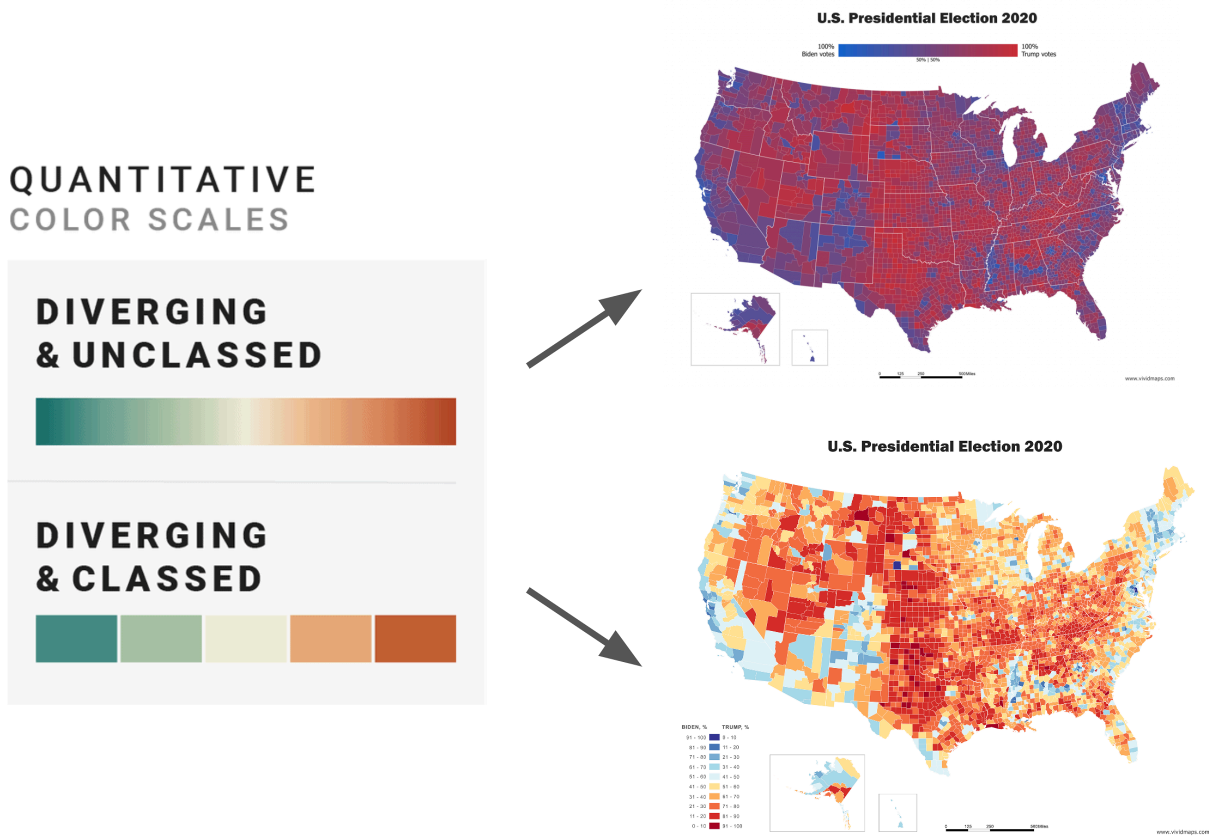
- Map value to color on a continuum, based on both **intensity** and **hue**
- Use for ordered data (e.g., population, income)



Source: Analyzing US Census Data, by Kyle Walker

Diverging Scales

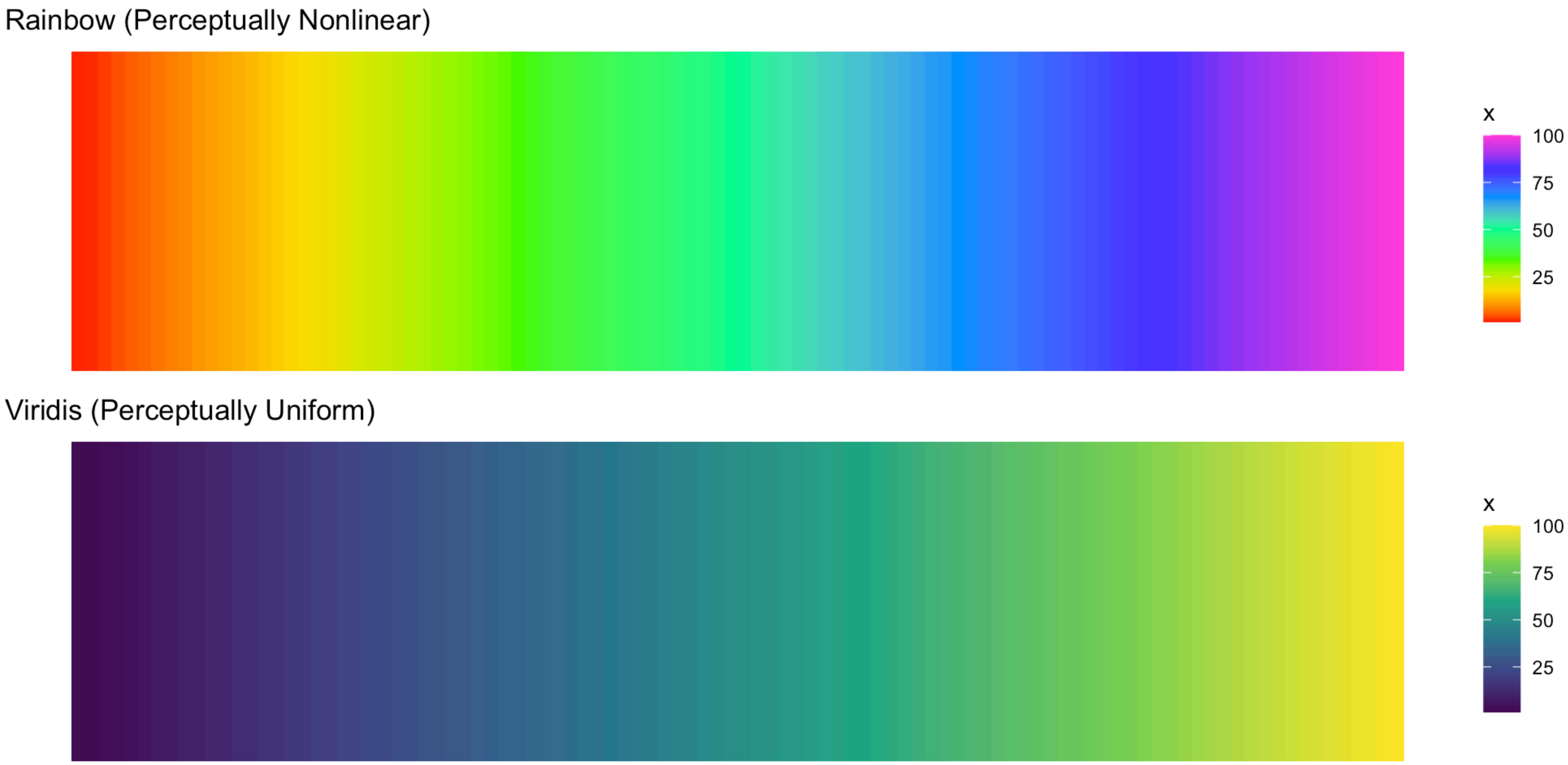
- Use for data with a **meaningful midpoint** (e.g., above/below average)
- Two contrasting colors with a **neutral midpoint** (e.g., white/light gray)



Source: 2020 U.S. Election Mapped, by Vivid Maps

Avoid Misleading Colors

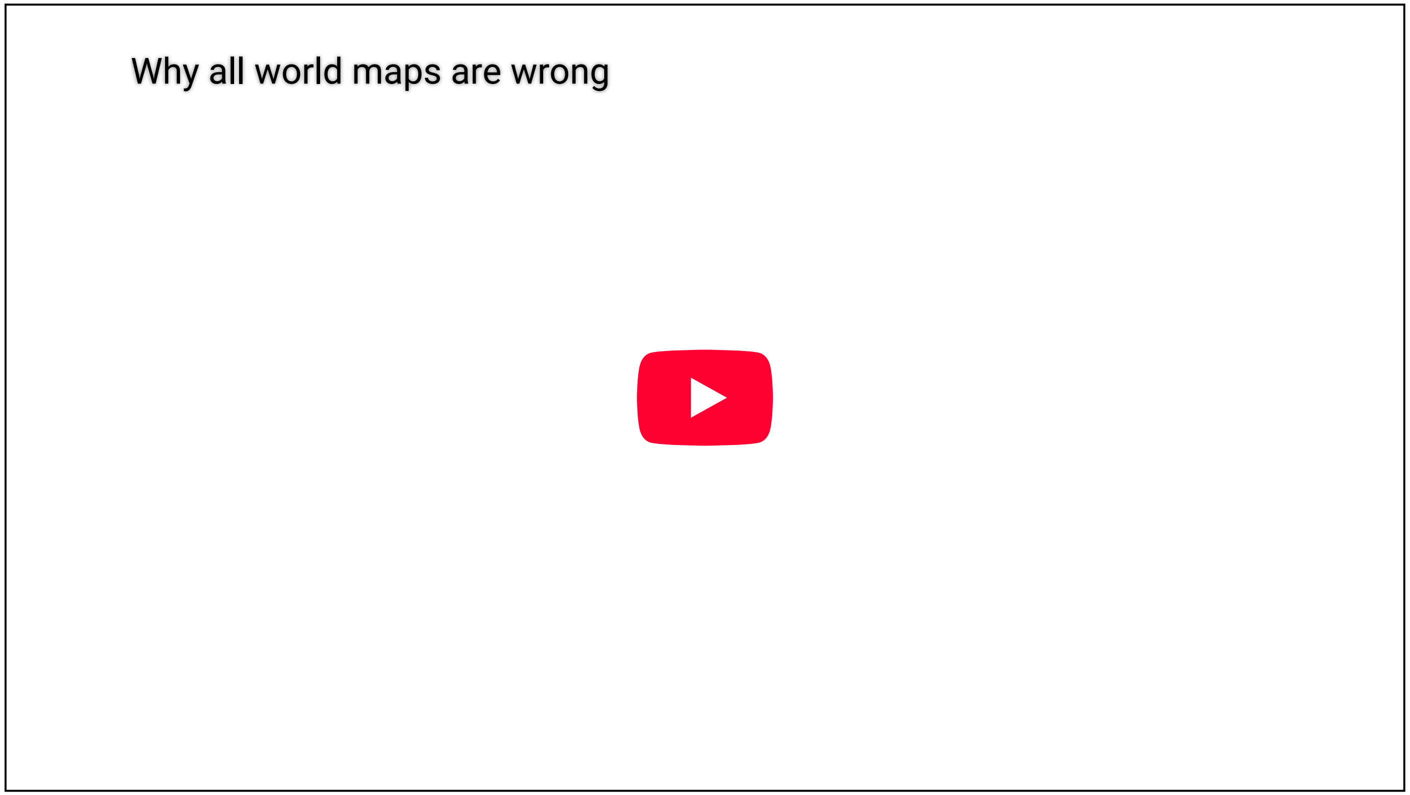
- Don't use rainbow: not perceptually uniform



- Consider accessibility (color-blind safe palettes)
- Avoid encoding meaning with non-intuitive colors

Map Projections

- A projection distorts shape, area, distance, or direction
- Use equal-area projections for choropleths



Common Projections in R

Use Case	Recommended Projection	EPSG Code
Equal-area choropleths	Albers Equal Area	5070
Interactive maps	Web Mercator	3857
Global perspective	Robinson or Winkel Tripel	54030 / 54042
Local detail (U.S.)	NAD83 / State Plane	varies

Use `st_transform()` to convert:

```
my_data <- sf::st_transform(my_data, crs = 5070)
```

In `tmap`:

- In static mode: all layers are reprojected to match the first layer.
- In interactive mode: all layers are projected to EPSG:3857.

Outline for today

- Types of maps for spatial data
- Map design considerations (color, scale, projection, legend)
- **R packages for mapping**
- Static maps with `tmap`
- Interactive maps with `leaflet`
- Accessing spatial data with `tigris` and `tidycensus`

R Ecosystem for Mapping

- **Data handling**: `sf`, `sp`
- **Thematic mapping**: `tmap`, `ggplot2`, `cartography`
- **Basemaps & interactivity**: `leaflet`, `mapview`, `ggmap`
- **Shapefiles**: `rgdal`, `rmapshaper`
- **Data access**: `tigris`, `tidycensus`

{sf}: simple features

The {sf} package is the **standard way to work with vector spatial data in R**. It replaces older tools like {sp} with a **simple, tidy-friendly** interface.

Key Features of {sf}

- Stores **geometry + attributes** in a single `data.frame`-like object
- Built on **simple features** standard (ISO 19125-1)
- Fully compatible with **dplyr**, **ggplot2**, **tmap**
- Uses `sfc` column to store spatial information (e.g., points, polygons)

```
Simple feature collection with 100 features and 6 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
Geodetic CRS:  NAD27
First 3 features:
  BIR74 SID74 NWBIR74 BIR79 SID79 NWBIR79                       geometry
1  1091     1      10  1364     0      19 MULTIPOLYGON (((-81.47276 3...
2   487     0      10   542     3      12 MULTIPOLYGON (((-81.23989 3...
3  3188     5     208  3616     6     260 MULTIPOLYGON (((-80.45634 3...
```

Simple feature

Simple feature geometry list-colum (sfc)

Simple feature geometry (sfg)

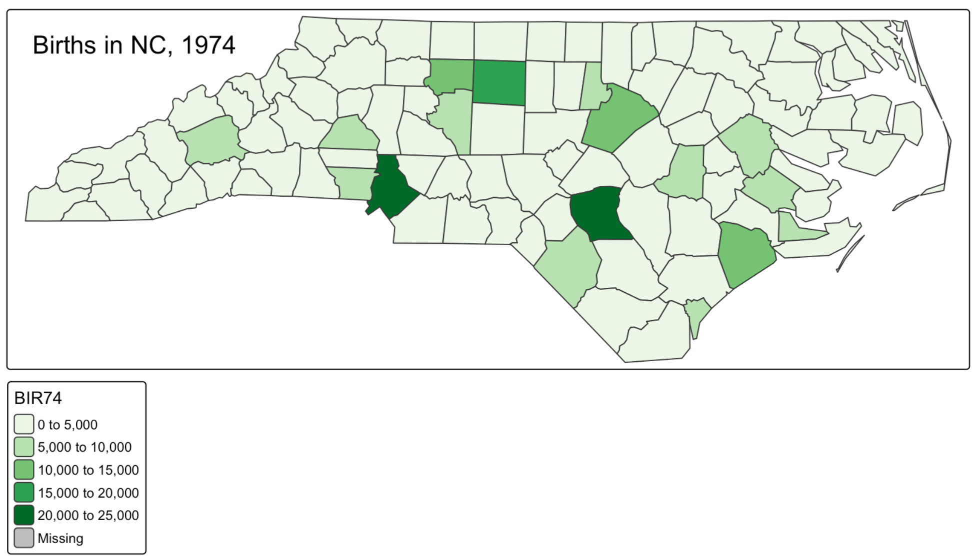
Outline for today

- Types of maps for spatial data
- Map design considerations (color, scale, projection, legend)
- R packages for mapping
- **Static maps with `tmap`**
- Interactive maps with `leaflet`
- Accessing spatial data with `tigris` and `tidycensus`

Static Mapping with `tmap`

- Similar to `ggplot2`, based on **"the grammar of graphics"**
- Supports both static and interactive modes
- Excellent for quick, polished maps, sensible defaults

```r
library(tmap)
tmap_mode("plot")
nc <- st_read("data/nc.shp", quiet = TRUE) # nc is an `sf` object
tm_shape(nc) + # defines input data
  tm_polygons("BIR74", palette = "Greens") + # mapping data to aesthetics
  tm_layout(title = "Births in NC, 1974")
```



How `{tmap}` works?

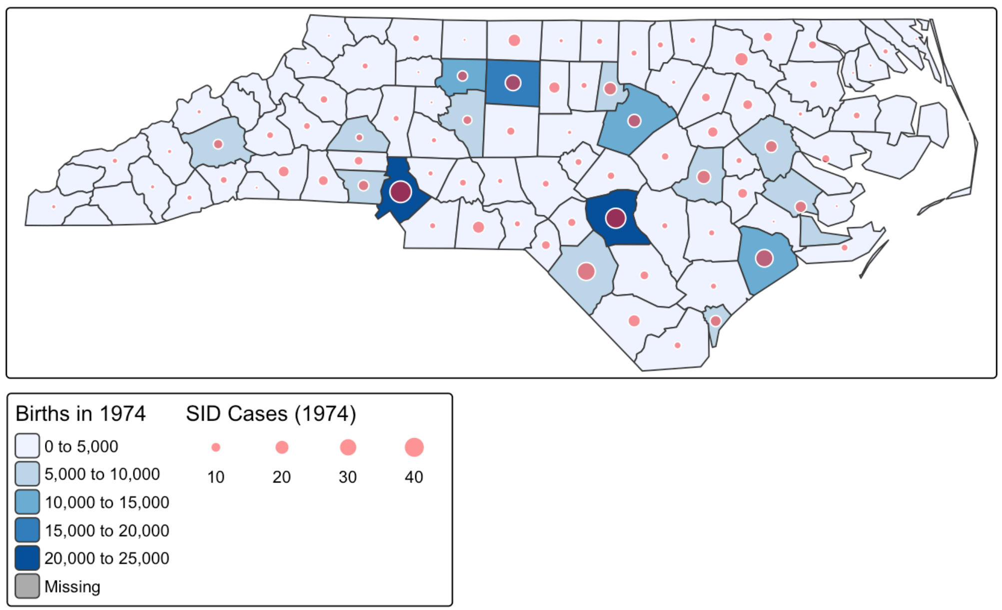
> `{tmap}` adopts an intuitive approach to map-making: the addition operator + adds a new layer, followed by tm_*():

- `tm_fill()`: shaded areas for (multi)polygons
- `tm_borders()`: border outlines for (multi)polygons
- `tm_polygons()`: both, shaded areas and border outlines for (multi)polygons
- `tm_lines()`: lines for (multi)linestrings
- `tm_symbols()`: symbols for (multi)points, (multi)linestrings, and (multi)polygons
- `tm_raster()`: colored cells of raster data (there is also tm_rgb() for rasters with three layers)
- `tm_text()`: text information for (multi)points, (multi)linestrings, and (multi)polygons

Adding layers in `{tmap}`

- `tm_polygons()`: for choropleth maps
- `tm_symbols()`: for point data, size and color can represent different variables

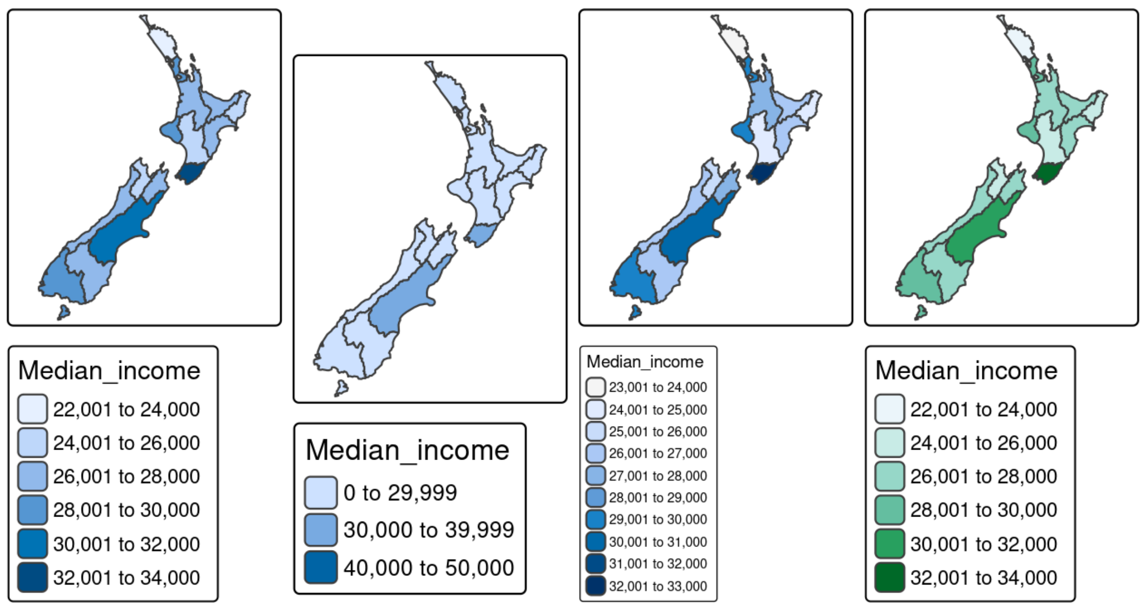
```r
# Create the map: choropleth + bubbles
tm_shape(nc) +
  tm_polygons("BIR74", palette = "brewer.blues", title = "Births in 1974") +
  tm_symbols(size = "SID74", col = "red", alpha = 0.5, border.col = "white",
             title.size = "SID Cases (1974)")
```



Scale

> Scales control how the values are represented on the map and in the legend, and can have a major impact on how spatial variability is portrayed

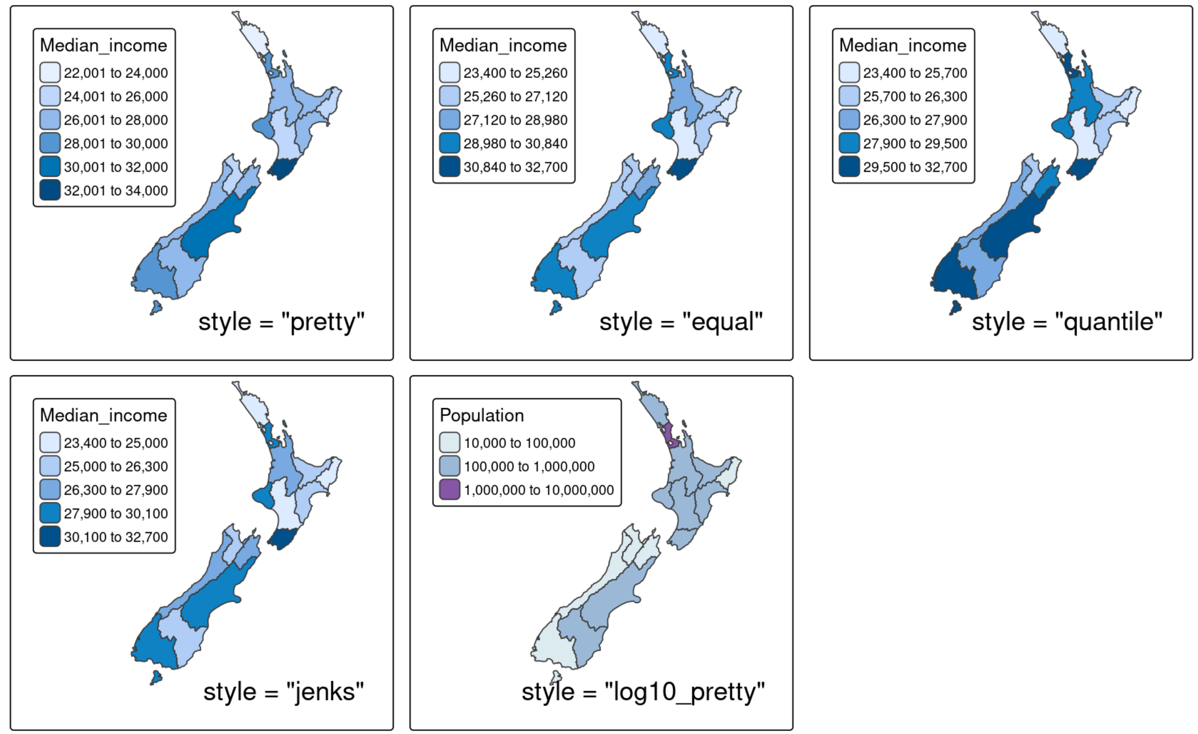
```r
tm_shape(nz) + tm_polygons(fill = "Median_income")
tm_shape(nz) + tm_polygons(fill = "Median_income",
                           fill.scale = tm_scale(breaks = c(0, 30000, 40000, 50000)))
tm_shape(nz) + tm_polygons(fill = "Median_income",
                              fill.scale = tm_scale(n = 10))
tm_shape(nz) + tm_polygons(fill = "Median_income",
                              fill.scale = tm_scale(values = "BuGn"))
```



Style options for classifying map data

`tm_scale_intervals(style = "pretty"):`

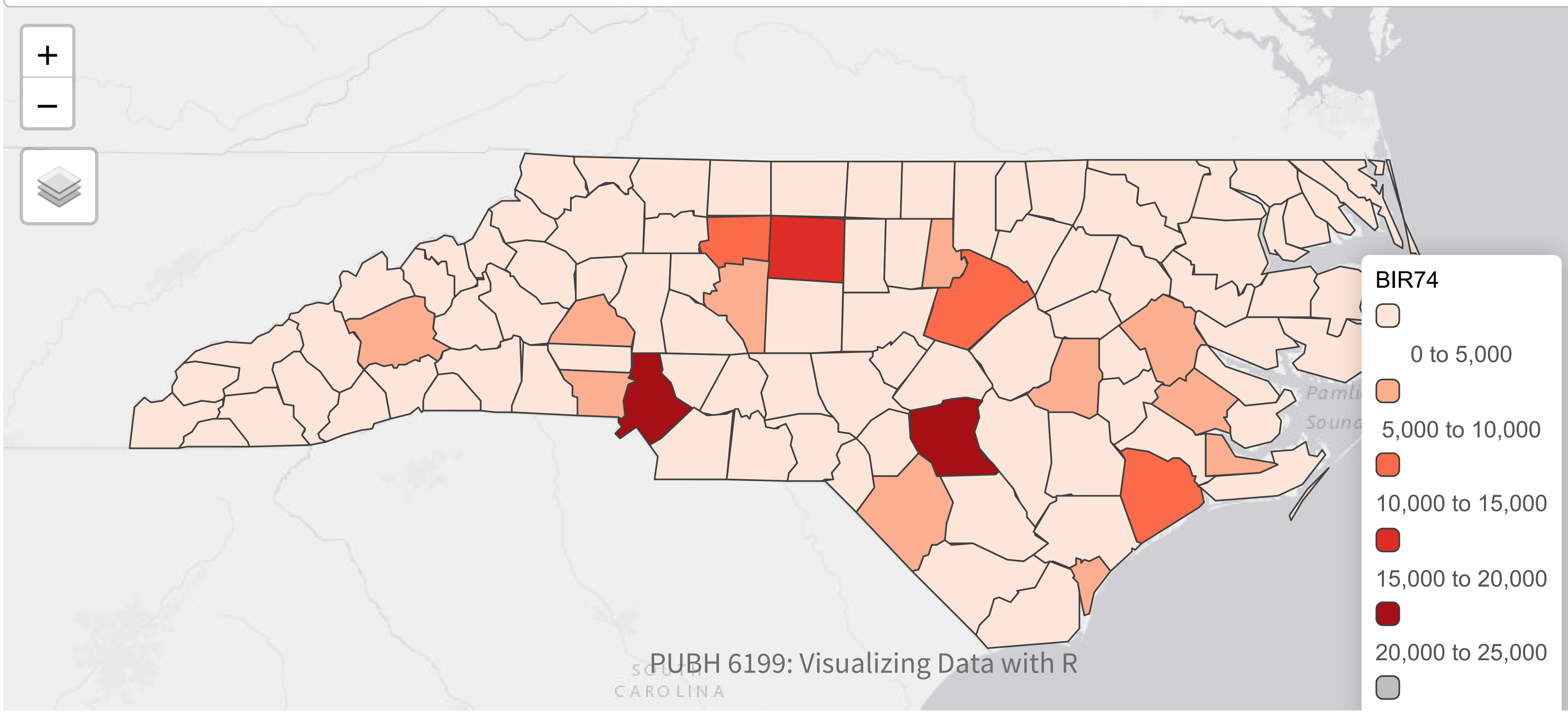
- **“pretty”**: Rounded, evenly spaced breaks **(default)**.
- **“equal”**: Equal-width bins; poor fit for skewed data — may hide variation.
- **“quantile”**: Equal count per bin; be careful with wide bin ranges.
- **“jenks”**: Optimizes natural groupings; can be slow with large datasets.
- **“log10_pretty”**: Log-scaled breaks; only appropriate for right-skewed, positive values.



Switch to interactive `{tmap}`

> A unique feature of `{tmap}` is its ability to create static and interactive maps using the same code. Maps can be viewed interactively at any point by switching to view mode, using the command `tmap_mode("view")`

```r
tmap_mode("view")
tm_shape(nc) +
  tm_polygons("BIR74", palette = "brewer.reds")
```



Outline for today

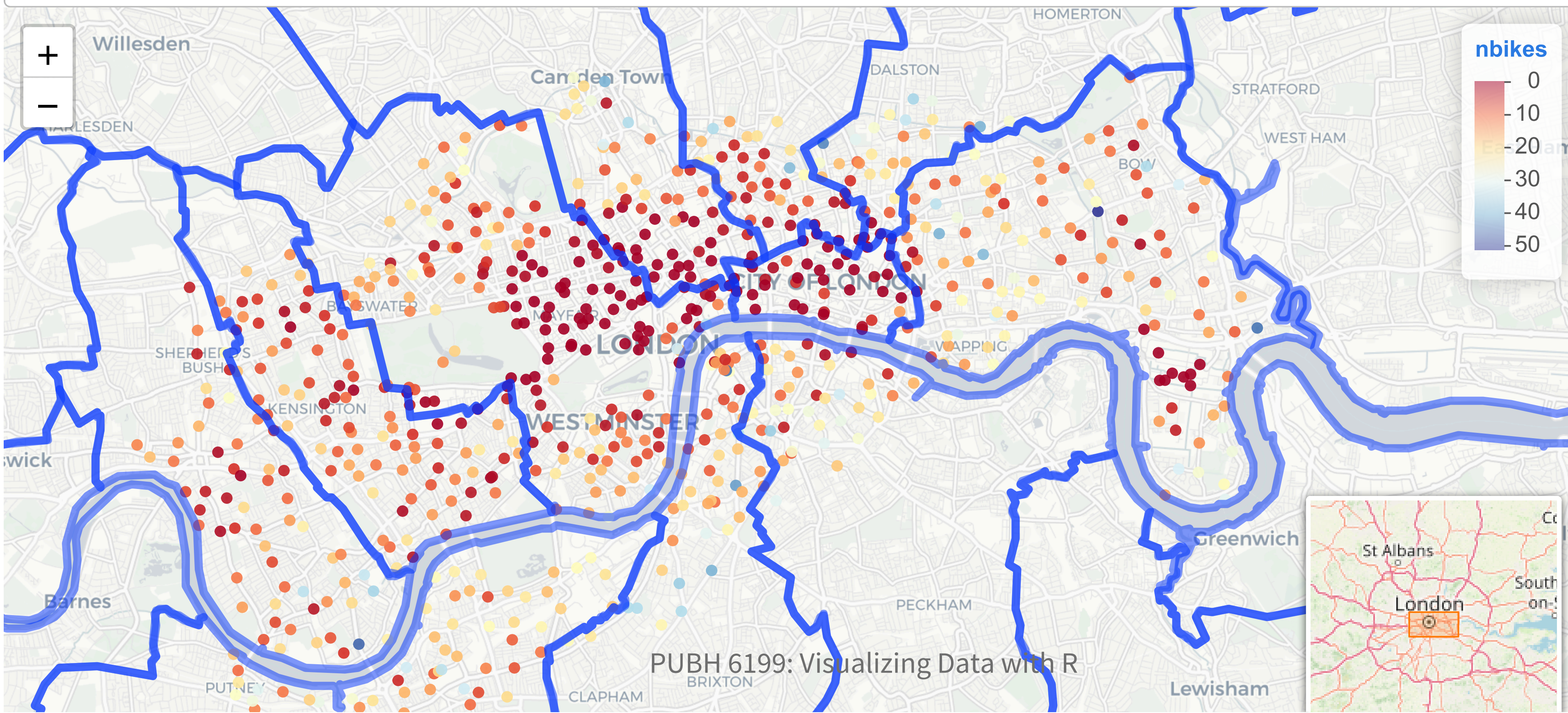
- Types of maps for spatial data
- Map design considerations (color, scale, projection, legend)
- R packages for mapping
- Static maps with `tmap`
- **Interactive maps with `leaflet`**
- Accessing spatial data with `tigris` and `tidycensus`

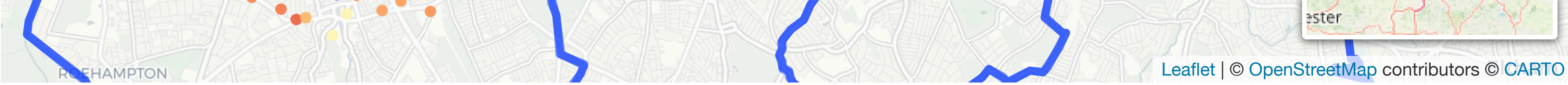
Interactive Mapping with `{leaflet}` in R

- `{leaflet}` is the most widely used interactive mapping package in R.
- It provides a relatively low-level interface to the Leaflet.js JavaScript library leafletjs.com.
- Maps start with leaflet() and use pipeable layers like addTiles(), addCircles(), and addPolygons().

Example `leaflet` Map

```r
pal <- colorNumeric("RdYlBu", domain = cycle_hire$nbikes)
# cycle_hire is an `sf` object with columns: name, nbikes, geometry, built-in data from spData
leaflet(cycle_hire) |>
  addProviderTiles(providers$CartoDB.Positron) |>
  addCircles(col = ~pal(nbikes), opacity = 0.9) |>
  # lnd is a London boroughs shapefile
  addPolygons(data = lnd, fill = FALSE) |>
  addLegend(pal = pal, values = ~nbikes) |>
  setView(lng = -0.1, lat = 51.5, zoom = 12) |>
  addMiniMap()
```



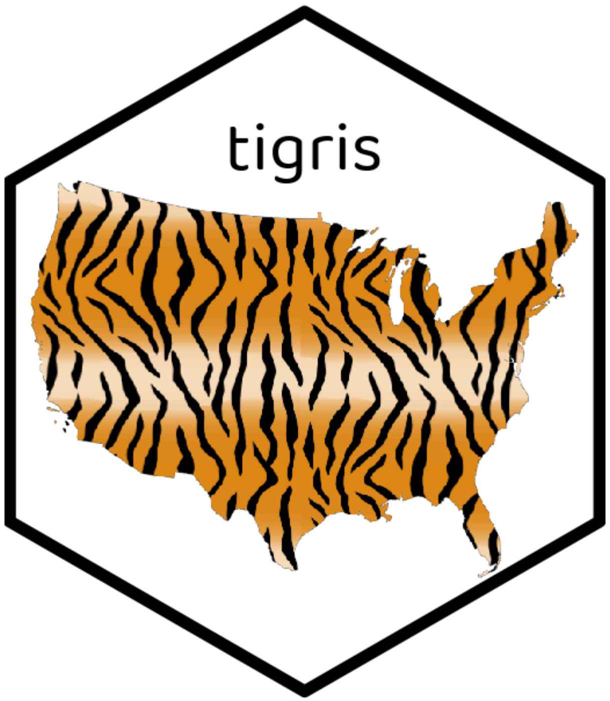

Leaflet | © OpenStreetMap contributors © CARTO

Outline for today

- Types of maps for spatial data
- Map design considerations (color, scale, projection, legend)
- R packages for mapping
- Static maps with `tmap`
- Interactive maps with `leaflet`
- **Accessing spatial data with `tigris` and `tidycensus`**

Getting Data with `tigris`

> The `{tigris}` package provides access to U.S. Census Bureau geographic data. Shapefiles downloaded using {tigris} will be loaded as a simple features (sf) object with geometries.



- A **shapefile** is a vector data file format commonly used for geospatial analysis.
- Shapefiles contain information for spatially describing features (e.g. points, lines, polygons), as well as any associated attribute information.
- You can find / download shapefiles online (e.g. from the US Census Bureau), or depending on the tools available, access them via packages (like we're doing today).

Getting U.S. County Shapefiles

Entire US

```r
library(tigris)
library(sf)
counties <- counties(state = NULL, cb = TRUE, progre
# Use `cb = TRUE` for simplified geometries
glimpse(counties)
```

```
Rows: 3,235
Columns: 13
$ STATEFP    <chr> "01", "01", "01", "01", "01", "04",
"04", "05", "05", "06",…
$ COUNTYFP   <chr> "069", "023", "005", "107", "033",
"012", "001", "081", "12…
$ COUNTYNS   <chr> "00161560", "00161537", "00161528",
"00161580", "00161542",…
$ AFFGEOID   <chr> "0500000US01069", "0500000US01023",
"0500000US01005", "0500…
$ GEOID      <chr> "01069", "01023", "01005", "01107",
"01033", "04012", "0400…
$ NAME       <chr> "Houston", "Choctaw", "Barbour",
"Pickens", "Colbert", "La …
$ NAMELSAD   <chr> "Houston County", "Choctaw County",
"Barbour County", "Pick…
$ STUSPS     <chr> "AL", "AL", "AL", "AL", "AL", "AZ",
"AZ", "AR", "AR", "CA",…
$ STATE_NAME <chr> "Alabama", "Alabama", "Alabama",
"Alabama", "Alabama", "Ari…
```

One state

```r
library(tigris)
library(sf)
counties_md <- counties(state = "Maryland", cb = TRU
# Use `cb = TRUE` for simplified geometries
glimpse(counties_md)
```

```
Rows: 24
Columns: 13
$ STATEFP    <chr> "24", "24", "24", "24", "24", "24",
"24", "24", "24", "24",…
$ COUNTYFP   <chr> "005", "047", "031", "037", "017",
"021", "043", "027", "03…
$ COUNTYNS   <chr> "01695314", "01668802", "01712500",
"01697853", "01676992",…
$ AFFGEOID   <chr> "0500000US24005", "0500000US24047",
"0500000US24031", "0500…
$ GEOID      <chr> "24005", "24047", "24031", "24037",
"24017", "24021", "2404…
$ NAME       <chr> "Baltimore", "Worcester",
"Montgomery", "St. Mary's", "Char…
$ NAMELSAD   <chr> "Baltimore County", "Worcester
County", "Montgomery County"…
$ STUSPS     <chr> "MD", "MD", "MD", "MD", "MD", "MD",
"MD", "MD", "MD", "MD",…
$ STATE_NAME <chr> "Maryland", "Maryland", "Maryland",
"Maryland", "Maryland",…
```

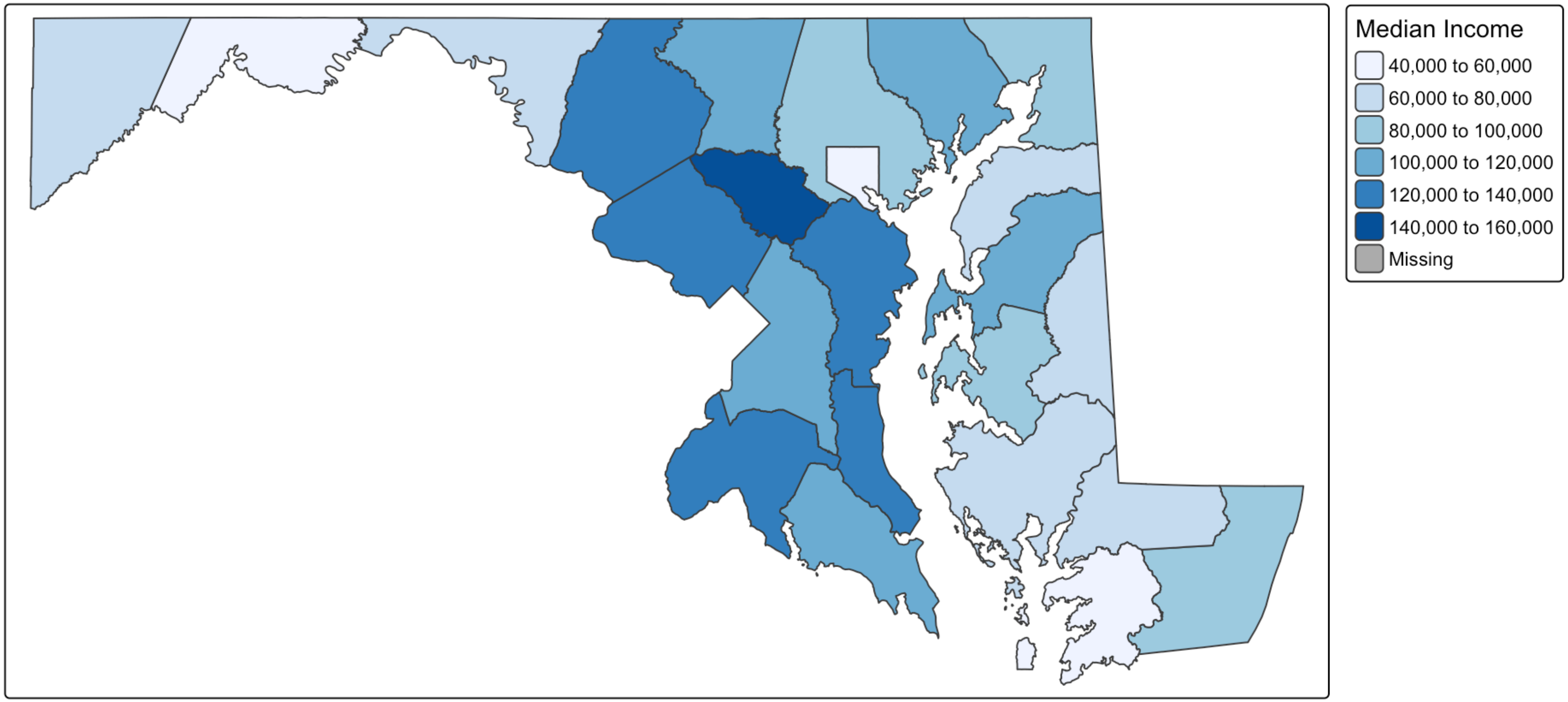
Getting Census Data with `tidycensus`

```r
library(tidycensus)
invisible(
  census_api_key(Sys.getenv("CENSUS_API_KEY"), install = TRUE, overwrite = TRUE)
)
options(tigris_use_cache = TRUE)
income_md <- get_acs(geography = "county",
  state = "MD",
  variables = "B19013_001",
  geometry = FALSE,
  show_progress = FALSE)
```

Lab 5 has a tutorial on how to get started with `{tidycensus}` packages. Follow it carefully. Remember to add your `.Renviron` to `.gitignore` so that you do not share your API keys.

Plotting Census Data

```
tmap_mode("plot")
income_md <- counties_md |>
  left_join(income_md, by = c("GEOID" = "GEOID")) # join the income data to the counties
tm_shape(income_md) +
  tm_polygons("estimate", palette = "Blues", title = "Median Income")
```



Your turn in HW 5:

- Choose a U.S. state
- Download county shapefiles with `tigris` or `tidycensus`
- Plot a choropleth using `tmap`
- Add labels and legends

15:00

Summary

- Choose the right map for the data and audience
- Make thoughtful color and projection choices
- Use `tmap` for quick static/interactive maps
- Use `leaflet` for rich interactivity
- Access geographic data via `tigris` and `tidycensus`

End-of-Class Survey

Fill out the end-of-class survey

~ This is the end of Lecture 5 ~

10:00