Chapter 6 Notes

Table of contents

# Chapter 6: Mapping Census data with R

```{r setup}  
library(tidycensus)  
library(tidyverse)  
```

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
✔ dplyr 1.1.3 ✔ readr 2.1.4  
✔ forcats 1.0.0 ✔ stringr 1.5.0  
✔ ggplot2 3.4.3 ✔ tibble 3.2.1  
✔ lubridate 1.9.2 ✔ tidyr 1.3.0  
✔ purrr 1.0.2   
── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
✖ dplyr::filter() masks stats::filter()  
✖ dplyr::lag() masks stats::lag()  
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

```{r setup}  
library(tmap)  
```

The legacy packages maptools, rgdal, and rgeos, underpinning the sp package,  
which was just loaded, will retire in October 2023.  
Please refer to R-spatial evolution reports for details, especially  
https://r-spatial.org/r/2023/05/15/evolution4.html.  
It may be desirable to make the sf package available;  
package maintainers should consider adding sf to Suggests:.  
The sp package is now running under evolution status 2  
 (status 2 uses the sf package in place of rgdal)  
Breaking News: tmap 3.x is retiring. Please test v4, e.g. with  
remotes::install\_github('r-tmap/tmap')

```{r setup}  
library(mapview)  
library(leaflet)  
library(mapboxapi)  
```

Usage of the Mapbox APIs is governed by the Mapbox Terms of Service.  
Please visit https://www.mapbox.com/legal/tos/ for more information.

```{r setup}  
library(ggiraph)  
library(scales)  
```

Attaching package: 'scales'  
  
The following object is masked from 'package:purrr':  
  
 discard  
  
The following object is masked from 'package:readr':  
  
 col\_factor

```{r setup}  
library(mapdeck)  
```

Attaching package: 'mapdeck'  
  
The following object is masked from 'package:tibble':  
  
 add\_column

```{r setup}  
library(patchwork)  
library(shiny)  
library(htmlwidgets)  
library(sf)  
```

Linking to GEOS 3.11.2, GDAL 3.6.2, PROJ 9.2.0; sf\_use\_s2() is TRUE

```{r setup}  
library(tigris)  
```

To enable caching of data, set `options(tigris\_use\_cache = TRUE)`  
in your R script or .Rprofile.

* This chapter shows how to map Census data by using linked geometry parameters.
  + We review static mapping in **ggplot2** and **tmap**
  + We then review interactive mapping in **mapview** and **Leaflet**

## 6.1 Using geometry in tidycensus

* get\_acs(), get\_decennial(), and get\_estimates() have a geometry parameter
  + when set to TRUE, **tigris** is called by the functions to download associated spatial data alongside the census data

```{r dc\_med\_inc}  
options(tigris\_use\_cache = TRUE)  
  
dc\_income <- get\_acs(  
 geography = "tract",  
 variables = "B19013\_001",  
 state = "DC",  
 year = 2020,  
 geometry = TRUE  
)  
```

Getting data from the 2016-2020 5-year ACS

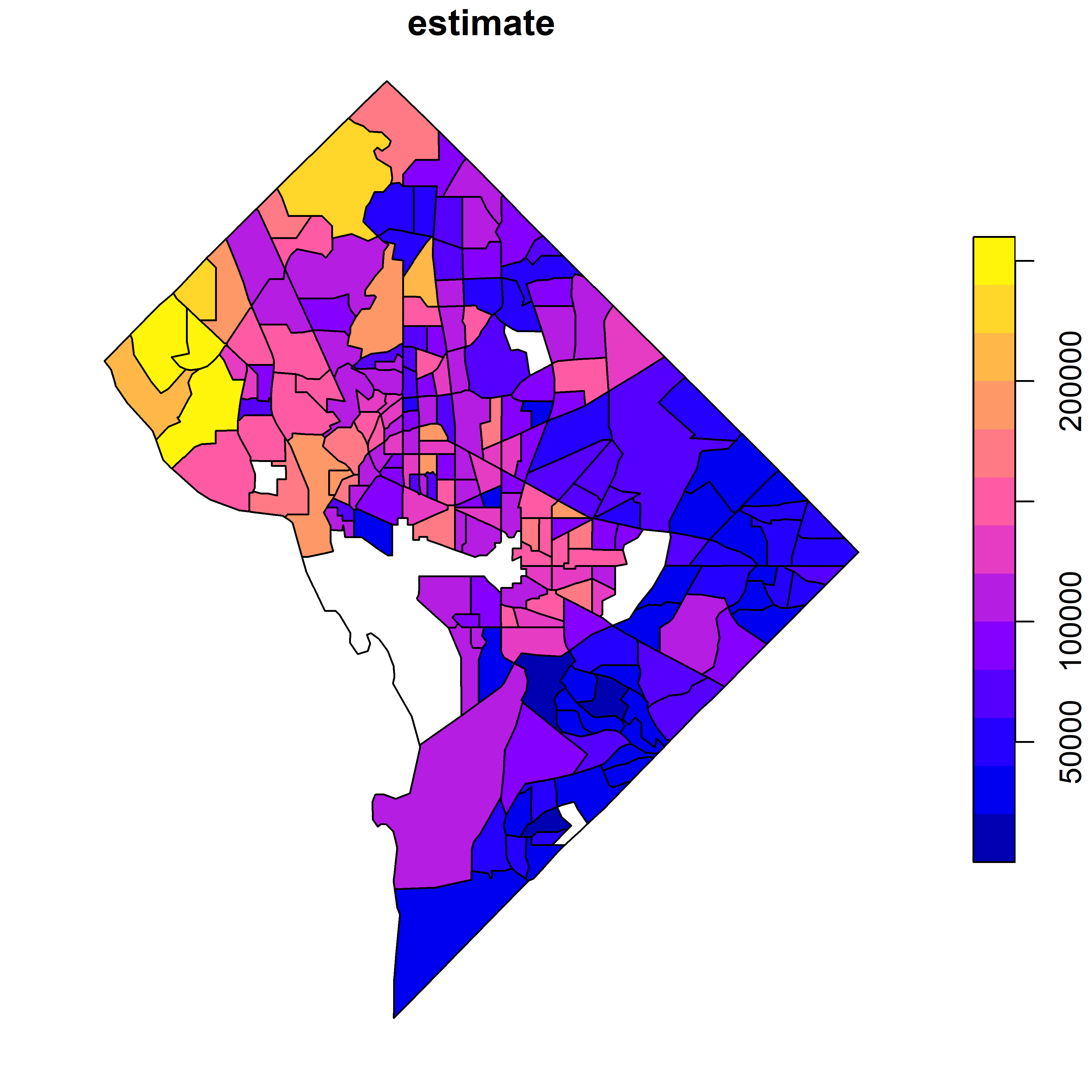
```{r dc\_med\_inc}  
dc\_income  
```

Simple feature collection with 206 features and 5 fields  
Geometry type: POLYGON  
Dimension: XY  
Bounding box: xmin: -77.11976 ymin: 38.79165 xmax: -76.9094 ymax: 38.99511  
Geodetic CRS: NAD83  
First 10 features:  
 GEOID NAME  
1 11001001002 Census Tract 10.02, District of Columbia, District of Columbia  
2 11001002801 Census Tract 28.01, District of Columbia, District of Columbia  
3 11001003100 Census Tract 31, District of Columbia, District of Columbia  
4 11001004001 Census Tract 40.01, District of Columbia, District of Columbia  
5 11001010500 Census Tract 105, District of Columbia, District of Columbia  
6 11001004600 Census Tract 46, District of Columbia, District of Columbia  
7 11001008803 Census Tract 88.03, District of Columbia, District of Columbia  
8 11001009507 Census Tract 95.07, District of Columbia, District of Columbia  
9 11001009509 Census Tract 95.09, District of Columbia, District of Columbia  
10 11001005303 Census Tract 53.03, District of Columbia, District of Columbia  
 variable estimate moe geometry  
1 B19013\_001 140772 20766 POLYGON ((-77.08563 38.9382...  
2 B19013\_001 62323 22218 POLYGON ((-77.03645 38.9349...  
3 B19013\_001 133408 18433 POLYGON ((-77.02826 38.9318...  
4 B19013\_001 153650 5886 POLYGON ((-77.05018 38.9212...  
5 B19013\_001 92083 20106 POLYGON ((-77.01756 38.8850...  
6 B19013\_001 129896 18312 POLYGON ((-77.01811 38.9146...  
7 B19013\_001 60819 22048 POLYGON ((-77.00173 38.9099...  
8 B19013\_001 72880 18411 POLYGON ((-77.00229 38.9567...  
9 B19013\_001 86042 25212 POLYGON ((-77.00201 38.9510...  
10 B19013\_001 97181 9127 POLYGON ((-77.04298 38.9101...

### Basic mapping of sf objects with plot()

* we can visualize the estimates with plot using a bracketed column name call

```{r}  
plot(dc\_income["estimate"])  
```



* this returns a simple map of income variation

## 6.2 Map-making with ggplot2 and geom\_sf

* **ggplot2**’s use of geom\_sf() allows for the development of custom spatial data visuals

### Choropleth mapping

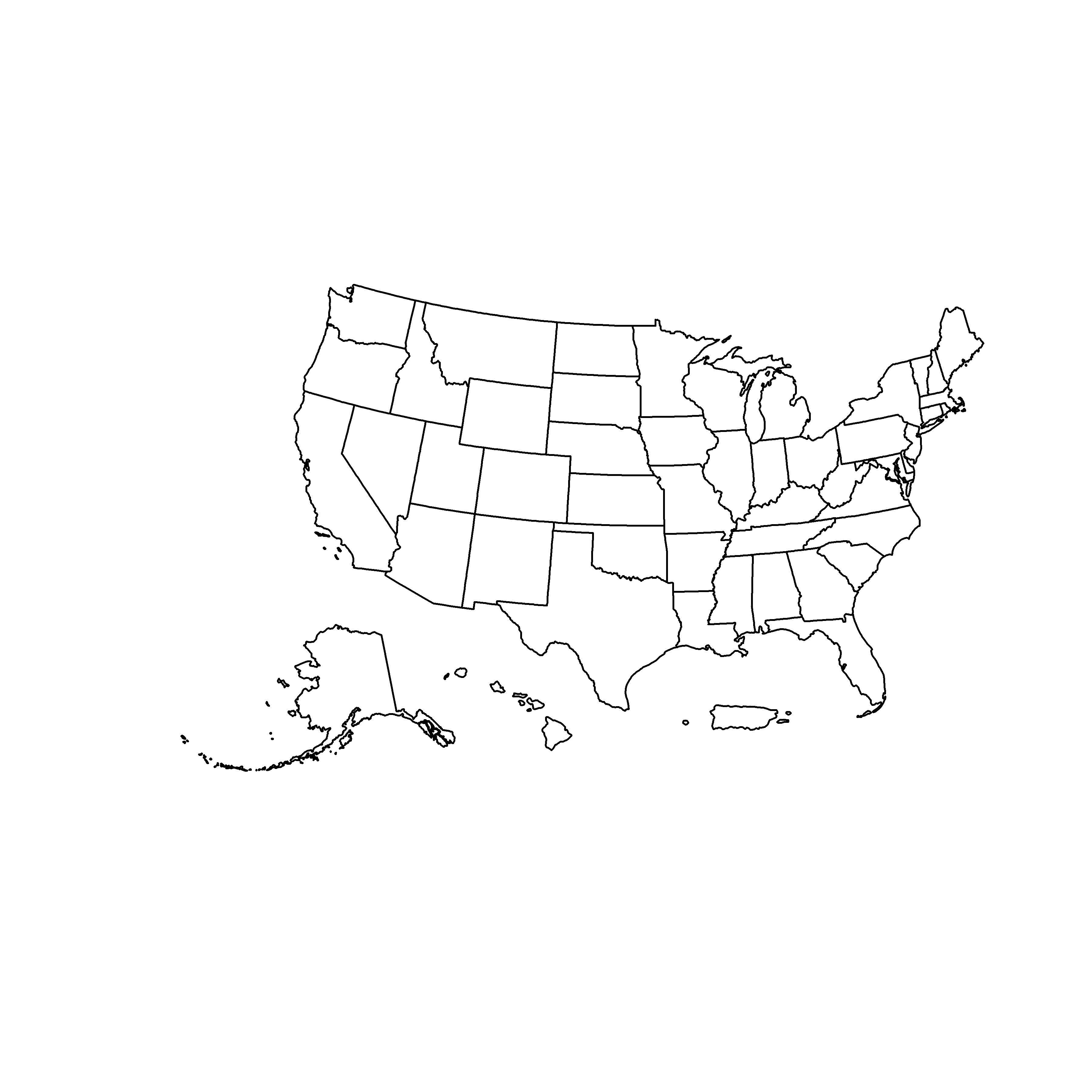
* Choropleth maps are very common in visualizing statistics over enumeration units
* Below we will use **tidycensus** and **tigris** to prepare a full national map

```{r}  
us\_median\_age <- get\_acs(  
 geography = "state",  
 variables = "B01002\_001",  
 year = 2019,  
 survey = "acs1",  
 geometry = TRUE,  
 resolution = "20m"  
) %>%   
 shift\_geometry()  
```

Getting data from the 2019 1-year ACS

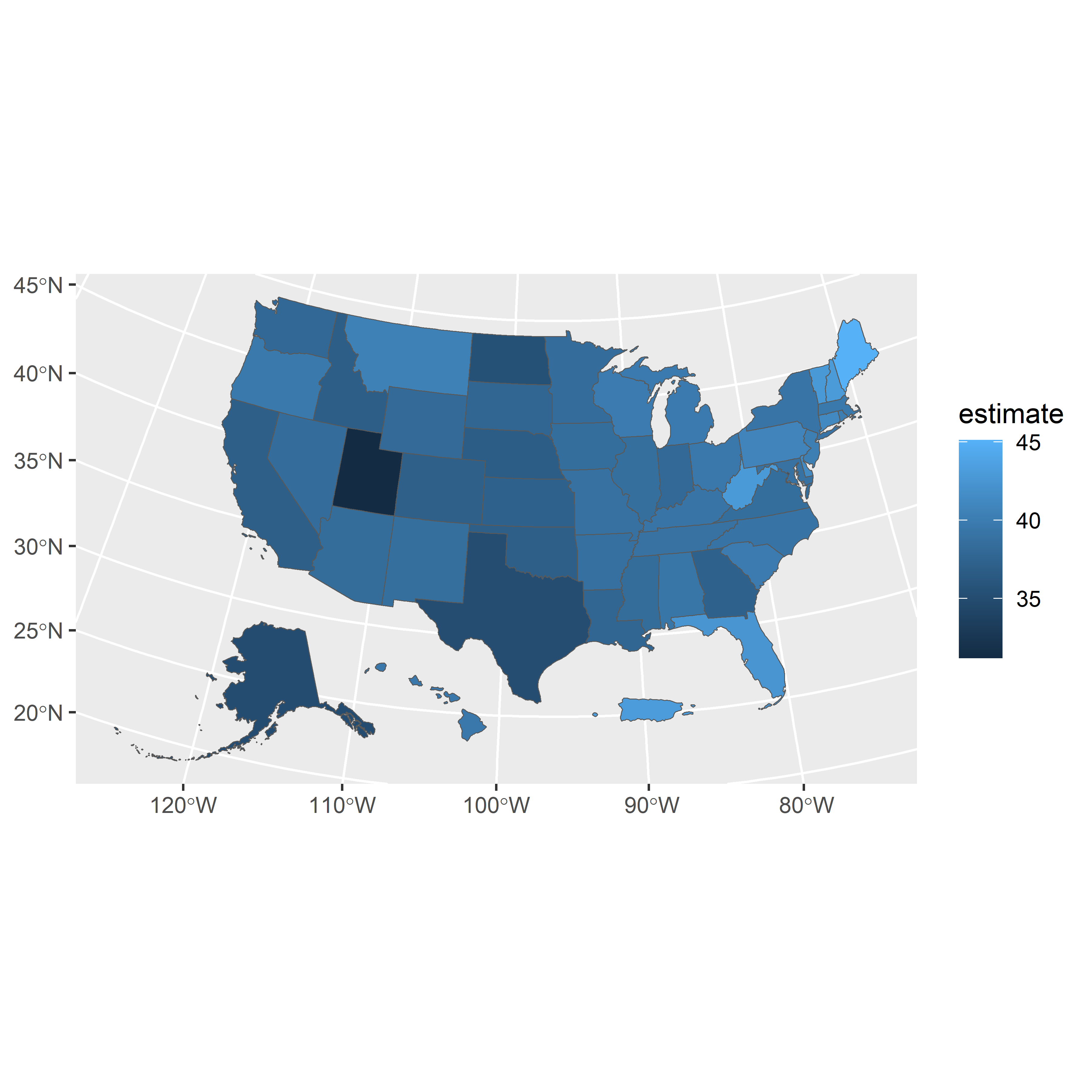
The 1-year ACS provides data for geographies with populations of 65,000 and greater.

```{r}  
plot(us\_median\_age$geometry)  
```



* we can then stylize the plot in **ggplot2**

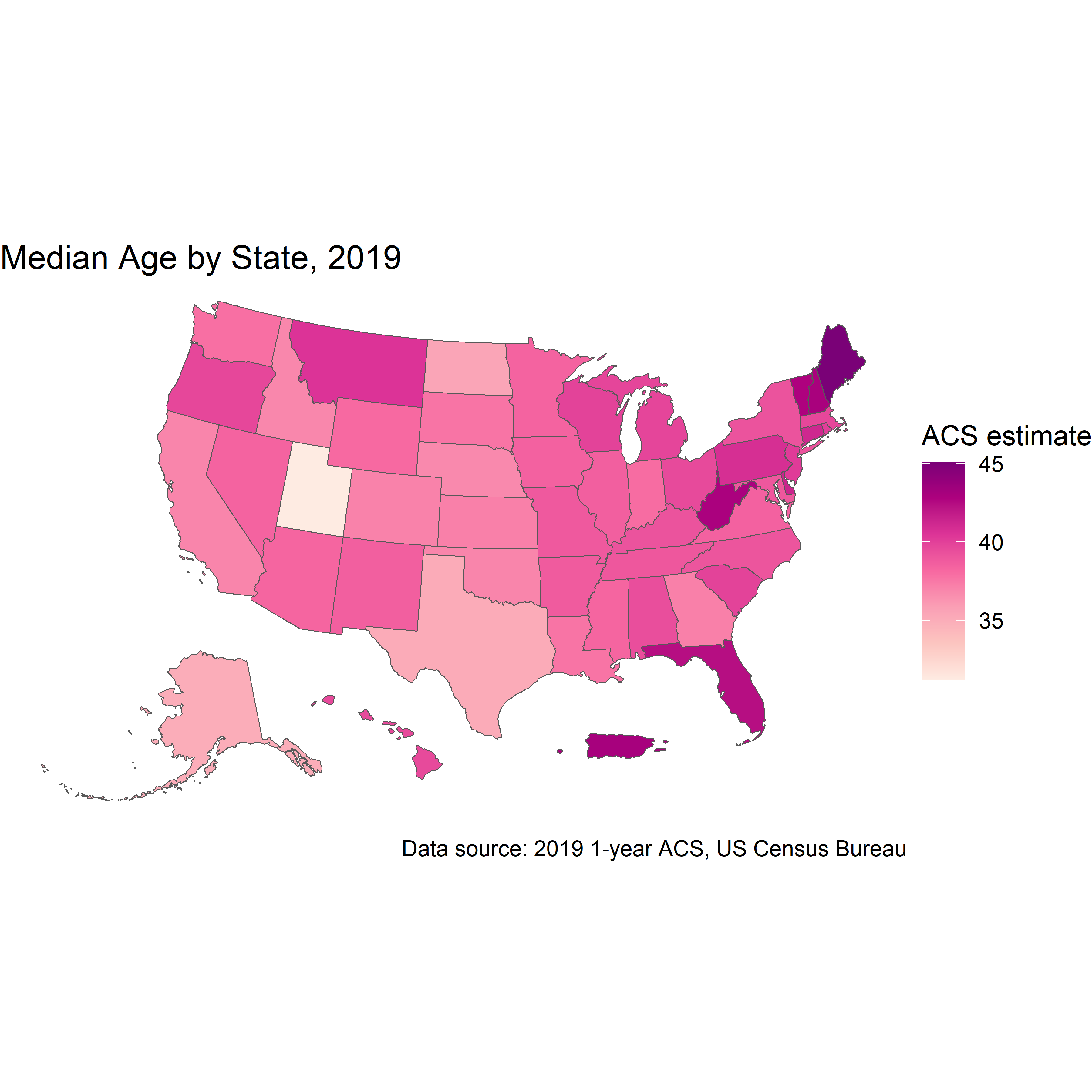
```{r}  
ggplot(data = us\_median\_age, aes(fill = estimate)) +  
 geom\_sf()  
```



### Customizing ggplot2 maps

* we have a few other tools that allow for more creative and meaningful plot construction
  + scale\_fill\_distiller() allows us to pull down palettes from ColorBrewer
  + labs() can add titles, caption(s), and a better legend label
  + theme\_void() removes background and gridlines from the plot area

```{r}  
ggplot(data = us\_median\_age, aes(fill = estimate)) +  
 geom\_sf() +  
 scale\_fill\_distiller(palette = "RdPu",  
 direction = 1) +  
 labs(title = "Median Age by State, 2019",  
 caption = "Data source: 2019 1-year ACS, US Census Bureau",  
 fill = "ACS estimate") +  
 theme\_void()  
```



## 6.3 Map-making with tmap

* **tmap** is more specifically developed for thematic mapping than **ggplot2**
* We will use get\_decennial() to map race data by Census tract in Hennepin County, Minnesota

```{r}  
hennepin\_race <- get\_decennial(  
 geography = "tract",  
 state = "MN",  
 county = "Hennepin",  
 variables = c(  
 Hispanic = "P2\_002N",  
 White = "P2\_005N",  
 Black = "P2\_006N",  
 Native = "P2\_007N",  
 Asian = "P2\_008N"  
 ),  
 summary\_var = "P2\_001N",  
 year = 2020,  
 geometry = TRUE  
) %>%   
 mutate(percent = 100 \* (value / summary\_value))  
```

Getting data from the 2020 decennial Census

Using the PL 94-171 Redistricting Data Summary File

Note: 2020 decennial Census data use differential privacy, a technique that  
introduces errors into data to preserve respondent confidentiality.  
ℹ Small counts should be interpreted with caution.  
ℹ See https://www.census.gov/library/fact-sheets/2021/protecting-the-confidentiality-of-the-2020-census-redistricting-data.html for additional guidance.  
This message is displayed once per session.

```{r}  
hennepin\_race  
```

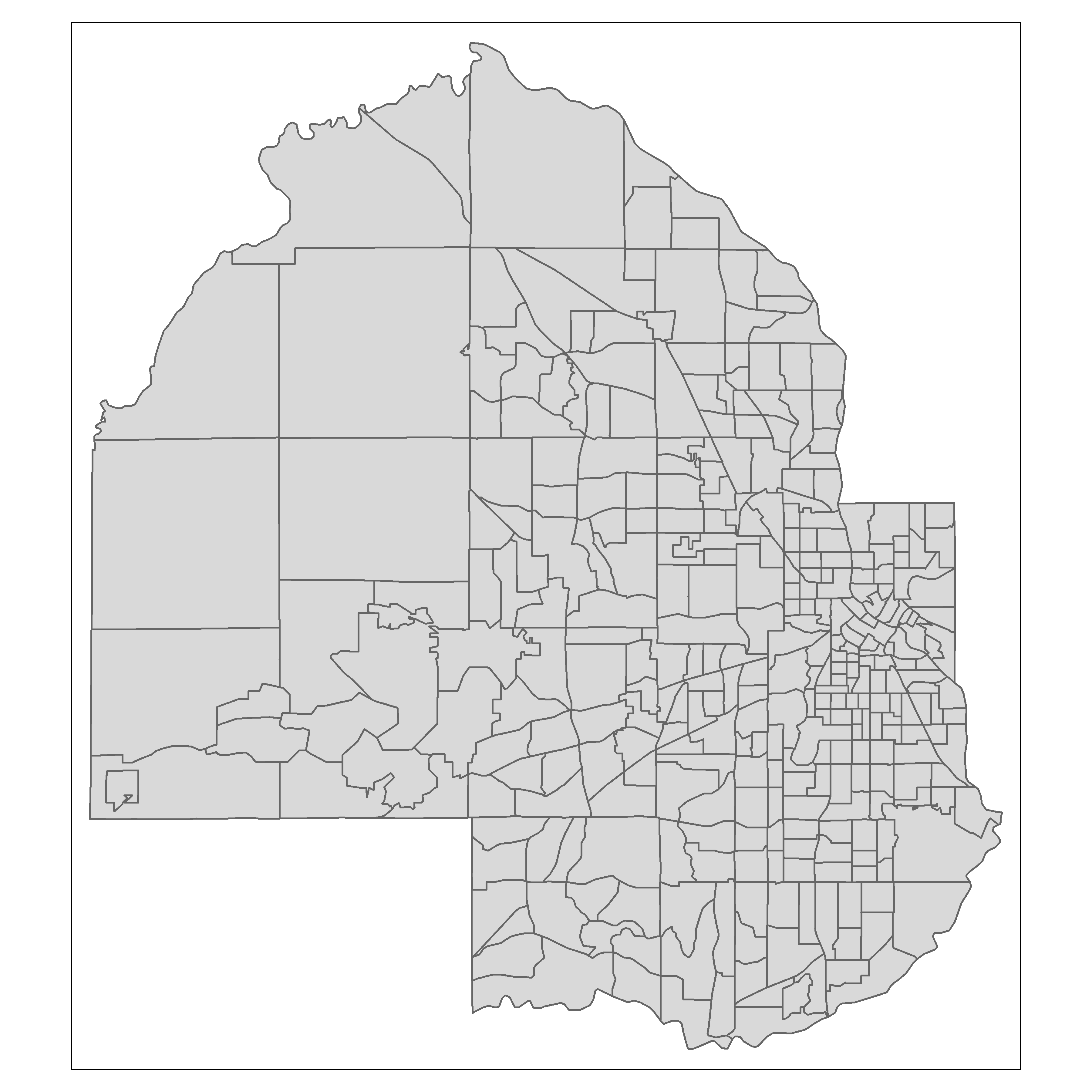
Simple feature collection with 1645 features and 6 fields  
Geometry type: MULTIPOLYGON  
Dimension: XY  
Bounding box: xmin: -93.76838 ymin: 44.78538 xmax: -93.17722 ymax: 45.24662  
Geodetic CRS: NAD83  
# A tibble: 1,645 × 7  
 GEOID NAME variable value summary\_value geometry percent  
 \* <chr> <chr> <chr> <dbl> <dbl> <MULTIPOLYGON [°]> <dbl>  
 1 2705302… Cens… Hispanic 577 4361 (((-93.35076 45.07625, -… 13.2   
 2 2705302… Cens… White 1489 4361 (((-93.35076 45.07625, -… 34.1   
 3 2705302… Cens… Black 722 4361 (((-93.35076 45.07625, -… 16.6   
 4 2705302… Cens… Native 23 4361 (((-93.35076 45.07625, -… 0.527  
 5 2705302… Cens… Asian 1327 4361 (((-93.35076 45.07625, -… 30.4   
 6 2705302… Cens… Hispanic 315 4819 (((-93.40064 45.01667, -… 6.54   
 7 2705302… Cens… White 3152 4819 (((-93.40064 45.01667, -… 65.4   
 8 2705302… Cens… Black 765 4819 (((-93.40064 45.01667, -… 15.9   
 9 2705302… Cens… Native 19 4819 (((-93.40064 45.01667, -… 0.394  
10 2705302… Cens… Asian 230 4819 (((-93.40064 45.01667, -… 4.77   
# ℹ 1,635 more rows

* note that the data is returned in long form/“tidy” form

### Choropleth maps with tmap

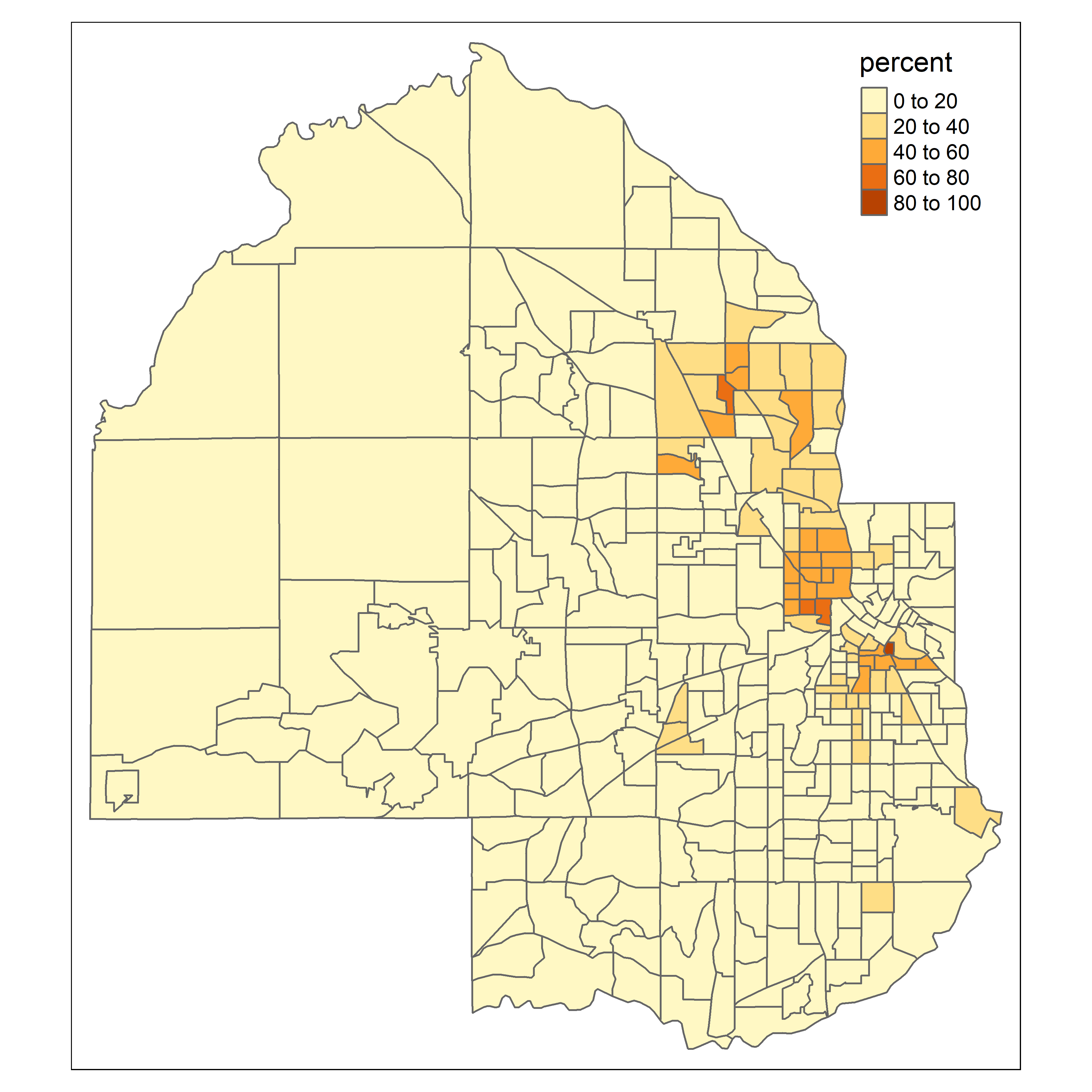
* like **ggplot2**, **tmap** uses a layer syntax appended with + signs

```{r}  
hennepin\_black <- filter(hennepin\_race,  
 variable == "Black")  
  
tm\_shape(hennepin\_black) +  
 tm\_polygons()  
```



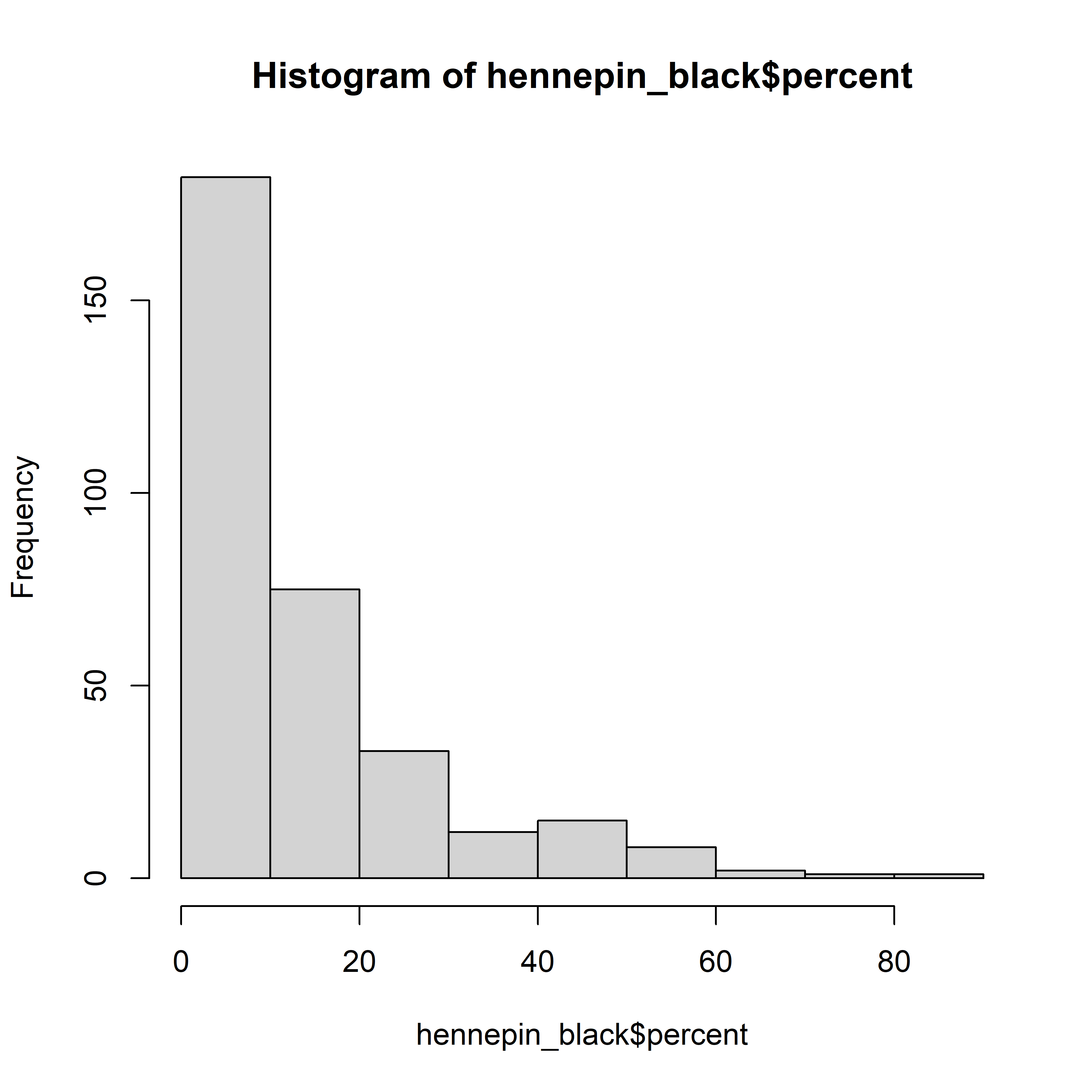
* the default view is not classified; to graduate the fill colors we can do so using tm\_fill()

```{r}  
tm\_shape(hennepin\_black) +  
 tm\_polygons(col = "percent")  
```



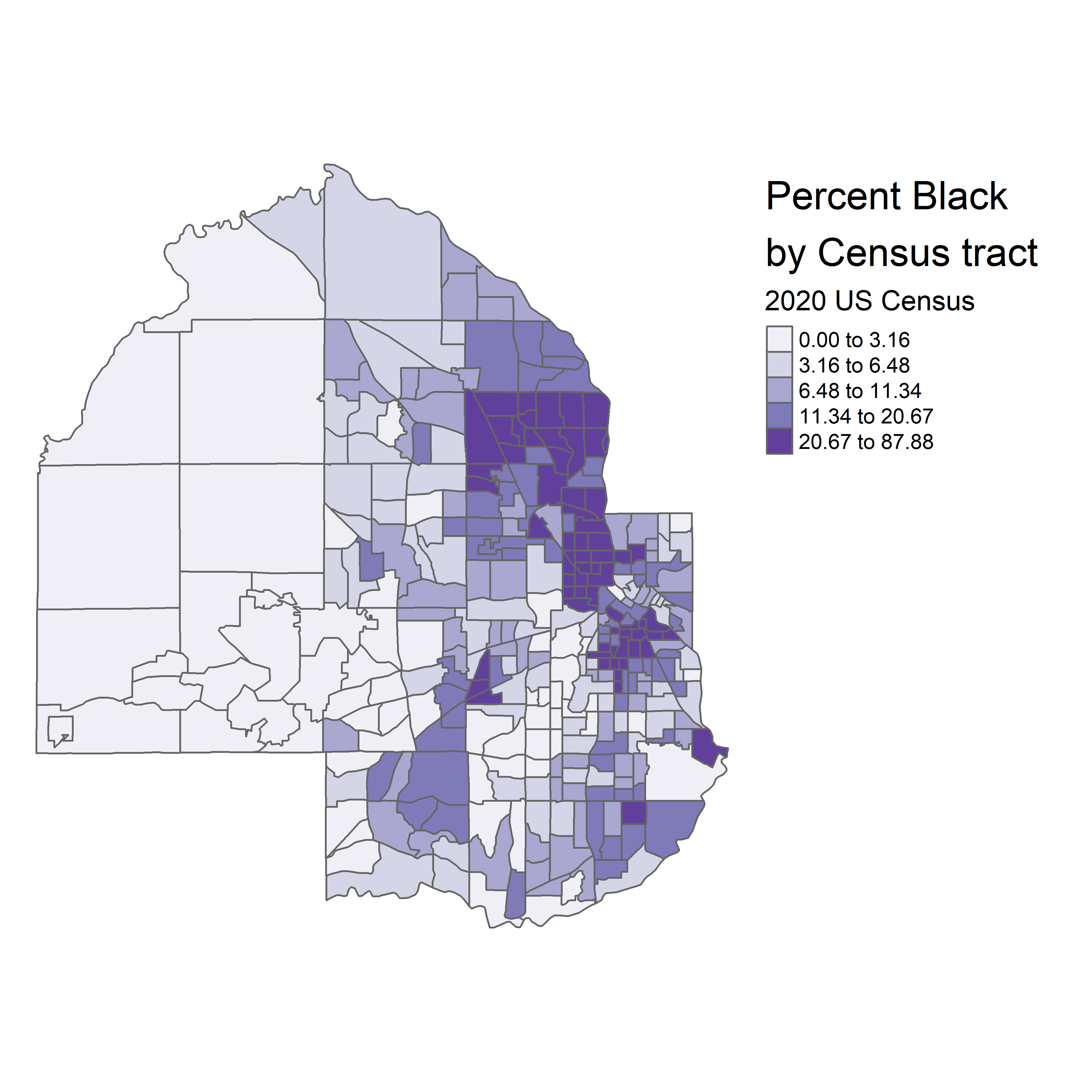
* **tmap** doesn’t create a continuous gradient, but rather breaks items into classes by a designated classification scheme
  + the default classification scheme is "pretty" which - (per ?pretty) - breaks the range of values into equally spaced classes demarcated by “equally spaced ‘round’ values”
  + this classification scheme is sensitive to the data’s underlying distribution

```{r}  
hist(hennepin\_black$percent)  
```



* We can use an alternative classification scheme like "quantile" which breaks the data down such that each class contains a (near) equal number of observations

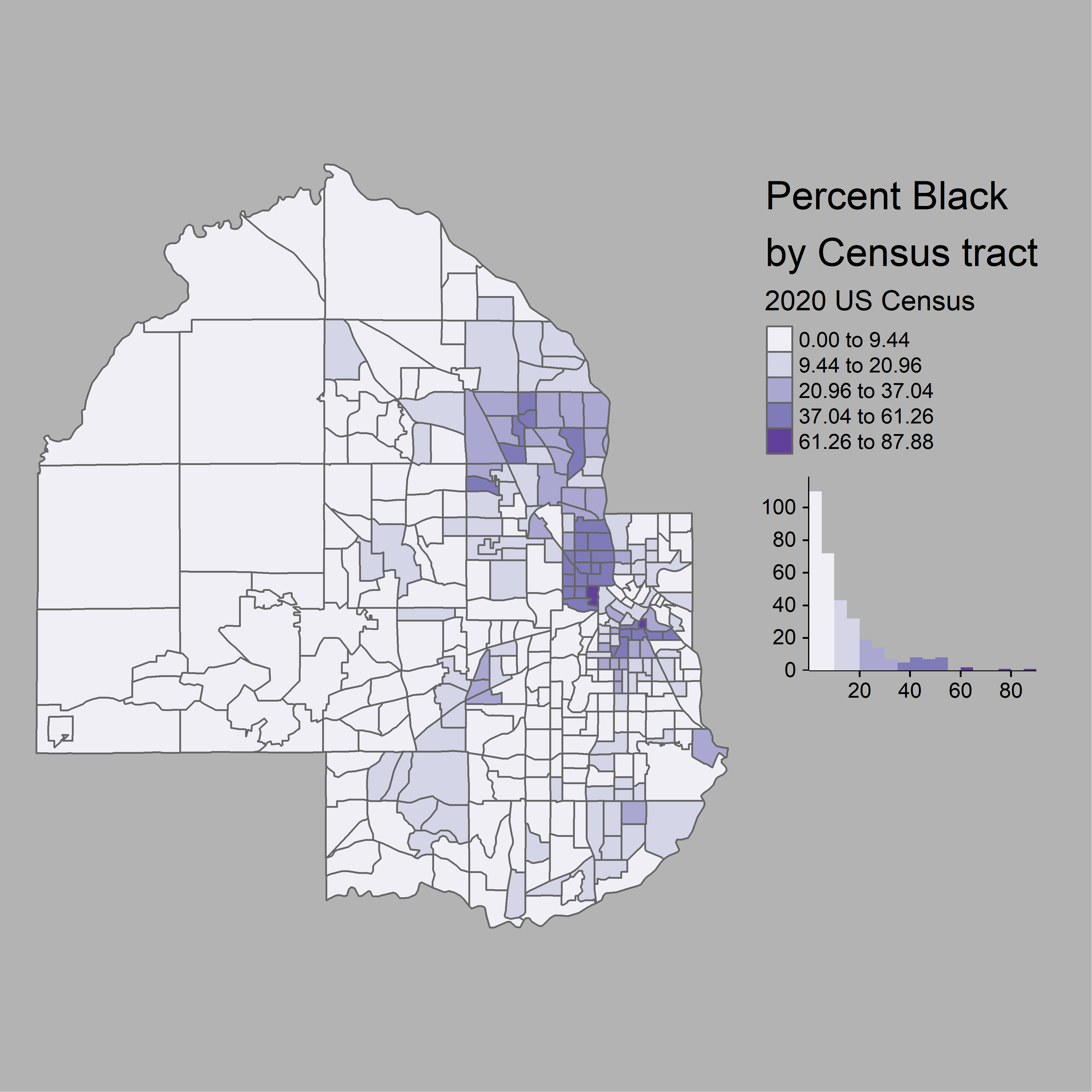
```{r}  
tm\_shape(hennepin\_black) +  
 tm\_polygons(col = "percent",  
 style = "quantile",  
 n = 5,  
 palette = "Purples",  
 title = "2020 US Census") +  
 tm\_layout(title = "Percent Black\nby Census tract",  
 frame = FALSE,  
 legend.outside = TRUE)  
```



* We may also use the commonly-selected Jenks algorithm which looks for meaningful breaks.
  + we can use legend.hist = TRUE to follow classification method changes across plots

```{r}  
tm\_shape(hennepin\_black) +  
 tm\_polygons(col = "percent",  
 style = "jenks",  
 n = 5,  
 palette = "Purples",  
 title = "2020 US Census",  
 legend.hist = TRUE) +  
 tm\_layout(title = "Percent Black\nby Census tract",  
 frame = FALSE,  
 legend.outside = TRUE,  
 bg.color = "grey70",  
 legend.hist.width = 5,  
 fontfamily = "Verdana")  
```

Warning in grid.Call.graphics(C\_text, as.graphicsAnnot(x$label), x$x, x$y, :  
font family not found in Windows font database  
  
Warning in grid.Call.graphics(C\_text, as.graphicsAnnot(x$label), x$x, x$y, :  
font family not found in Windows font database  
  
Warning in grid.Call.graphics(C\_text, as.graphicsAnnot(x$label), x$x, x$y, :  
font family not found in Windows font database



### Adding reference elements to a map

* We often include several reference elements with a map, including:
  + basemap
  + north arrow
  + scale bar
* A quick way of getting a **tmap**-compatible basemap is **rosm**
* We may also use pre-designed or custom-designed base maps from Mapbox using **mapboxapi**

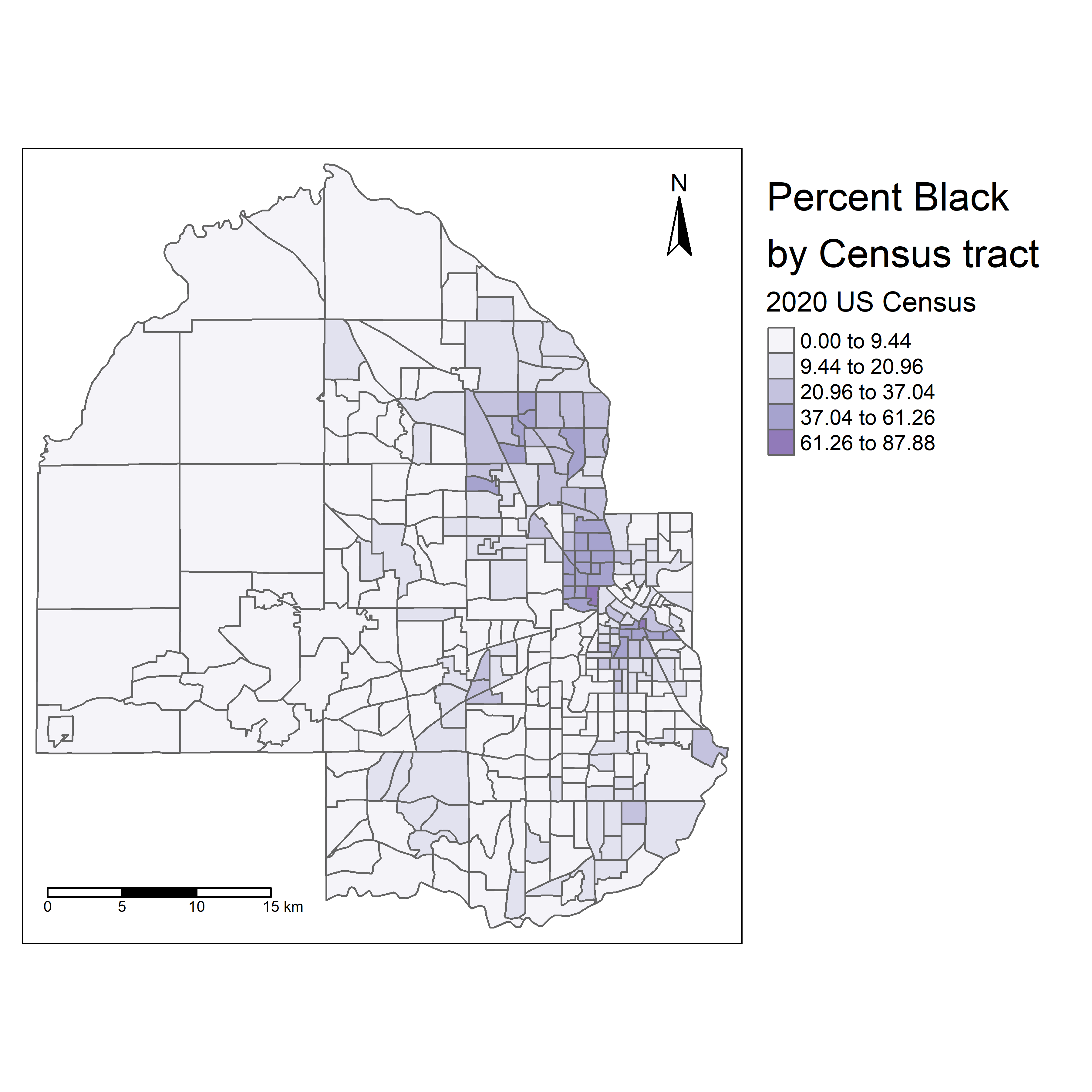
```{r}  
# Non-functional code because I don't want to give Mapbox my payment info  
# mb\_access\_token("NA")  
  
# hennepin\_tiles <- get\_static\_tiles(  
# location = hennepin\_black,  
# zoom = 10,  
# style\_id = "light-v9",  
# username = "mapbox"  
# )  
```

* when defining a basemap for choropleth maps, use muted monochrome designs to avoid drowning out choropleth colors
* basemaps are layered in tmap using tm\_rgb(), which plots a given raster
  + to see a basemap, add transparency to choropleth using the alpha argument
  + tm\_scale\_bar() adds a scale bar
  + tm\_compass() adds a north arrow
  + tm\_credits() lets one credit data sources and basemap sources

```{r}  
## Note that the basemap is commented out due to lack of a Mapbox account  
# tm\_shape(hennepin\_tiles) +  
# tm\_rgb() +  
tm\_shape(hennepin\_black) +  
 tm\_polygons(col = "percent",  
 style = "jenks",  
 n = 5,  
 palette = "Purples",  
 title = "2020 US Census",  
 alpha = 0.7) +  
 tm\_layout(title = "Percent Black\nby Census tract",  
 legend.outside = TRUE,  
 fontfamily = "Verdana") +  
 tm\_scale\_bar(position = c("left", "bottom")) +  
 tm\_compass(position = c("right", "top")) #+  
```

Warning in grid.Call.graphics(C\_text, as.graphicsAnnot(x$label), x$x, x$y, :  
font family not found in Windows font database  
  
Warning in grid.Call.graphics(C\_text, as.graphicsAnnot(x$label), x$x, x$y, :  
font family not found in Windows font database  
  
Warning in grid.Call.graphics(C\_text, as.graphicsAnnot(x$label), x$x, x$y, :  
font family not found in Windows font database  
  
Warning in grid.Call.graphics(C\_text, as.graphicsAnnot(x$label), x$x, x$y, :  
font family not found in Windows font database  
  
Warning in grid.Call.graphics(C\_text, as.graphicsAnnot(x$label), x$x, x$y, :  
font family not found in Windows font database

```{r}  
# tm\_credits("(c) Mapbox, OSM",  
 # bg.color = "white",  
 # position = c("RIGHT", "BOTTOM"))  
```



* position arguments allow for useful positioning
  + using all caps will bring elements closer to flush with the map frame

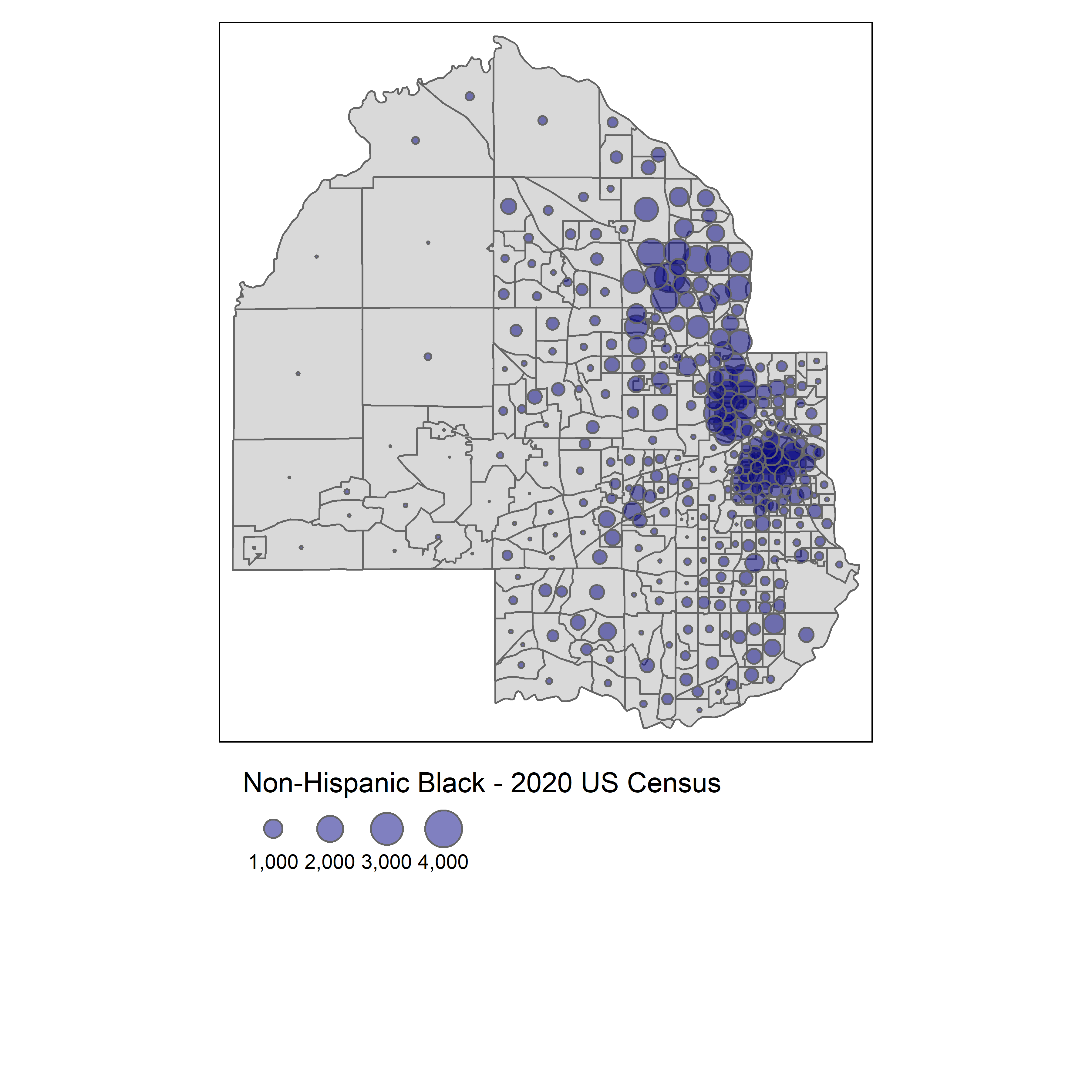
### Choosing a color palette

* Consider your data
  + quantitative data and statistical mapping means choosing palettes that show variation effectively
  + usually, this will be sequential palettes
    - Sometimes, we will go from light to dark <-> low to high, but other palettes show lighter colors more intensely, in which case we may choose the alternative.
  + Diverging palettes are useful for showing extremities along some axis with polarity
    - I.E. where there is some normal median value and extremes at both ends
    - In the context of Census mapping, diverging palettes are useful for Change over Time, with the center representing no change, and one color divergence representing a decrease over time while the other divergence represents increase over time.
  + Categorical/Qualitative data is best mapped with Qualitative palettes where colors are unique for each class, and do not imply an ordering of classes
  + Tools for palette selection include:
    - [ColorBrewer](https://colorbrewer2.org/)
    - the **viridis** library
    - tmaptools::palette\_explorer() includes selections from both of the above

### Alternative map types with tmap

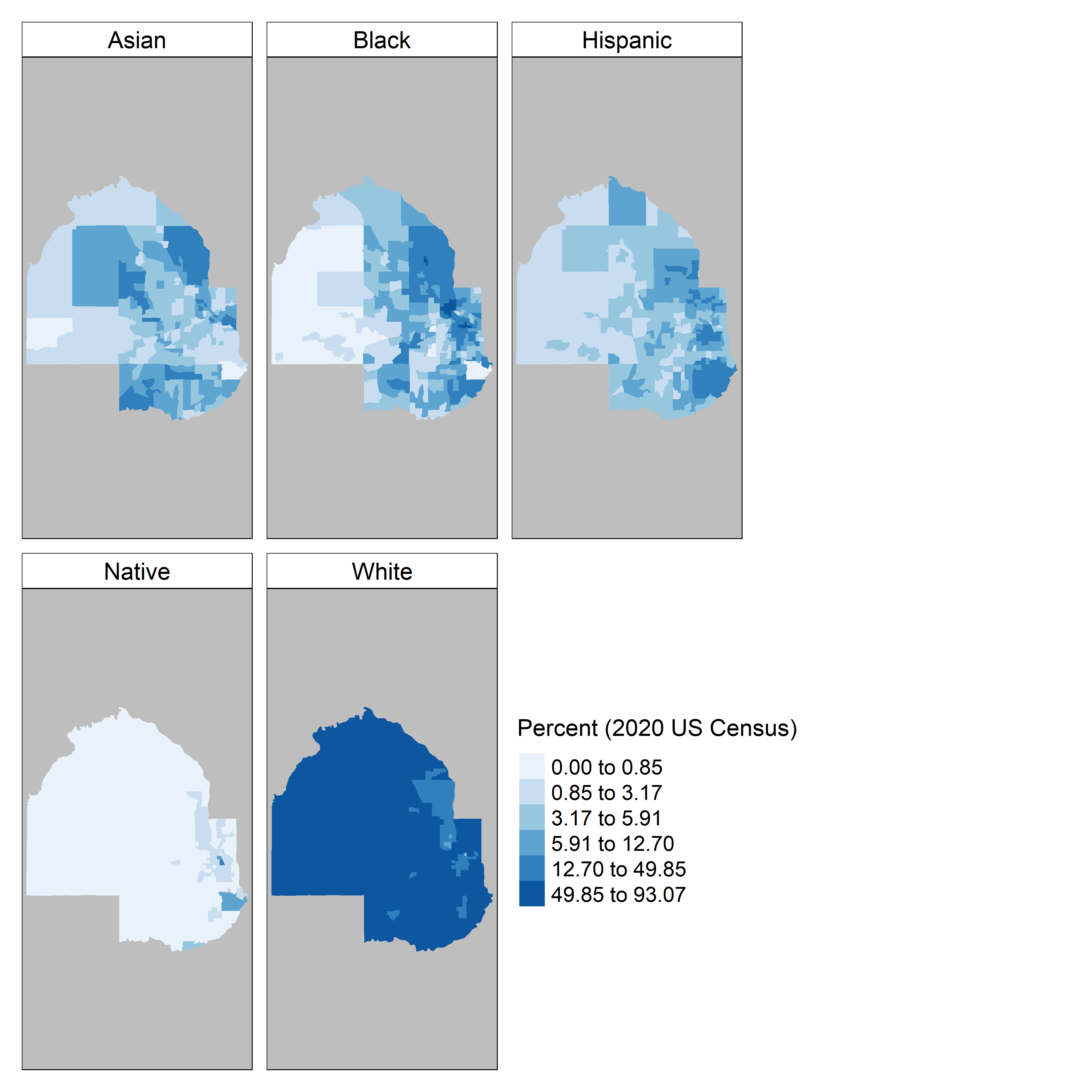
* Choropleth maps are best for “rates, percentages, or statistical values that are normalized for the population of areal units.”
  + not good for raw counts
  + can make comparison difficult
* Graduated symbols use shape symbols that scale in size relative to the data
  + bubble map below shows bubbles where size represents count

```{r bubble-hennepin}  
tm\_shape(hennepin\_black) +  
 tm\_polygons() +  
 tm\_bubbles(size = "value", alpha = 0.5,  
 col = "navy",  
 title.size = "Non-Hispanic Black - 2020 US Census") +  
 tm\_layout(legend.outside = TRUE,  
 legend.outside.position = "bottom")  
```



* Faceted maps are grids of univariate maps where each map shows a different variable, allowing comparison between variables without trying to force them all onto a single visualization.
* **tmap** allows for the creation of faceted maps using tm\_facets()
* weaknesses of faceted maps include:
  + shared legend and classification - suppresses within-group variation
  + difficult to discern diversity of areas

```{r hennepin-facet}  
tm\_shape(hennepin\_race) +  
 tm\_facets(by = "variable", scale.factor = 4) +  
 tm\_fill(col = "percent",  
 style = "quantile",  
 n = 6,  
 palette = "Blues",  
 title = "Percent (2020 US Census)",) +  
 tm\_layout(bg.color = "grey",  
 legend.position = c(-0.7, 0.15),  
 panel.label.bg.color = "white")  
```

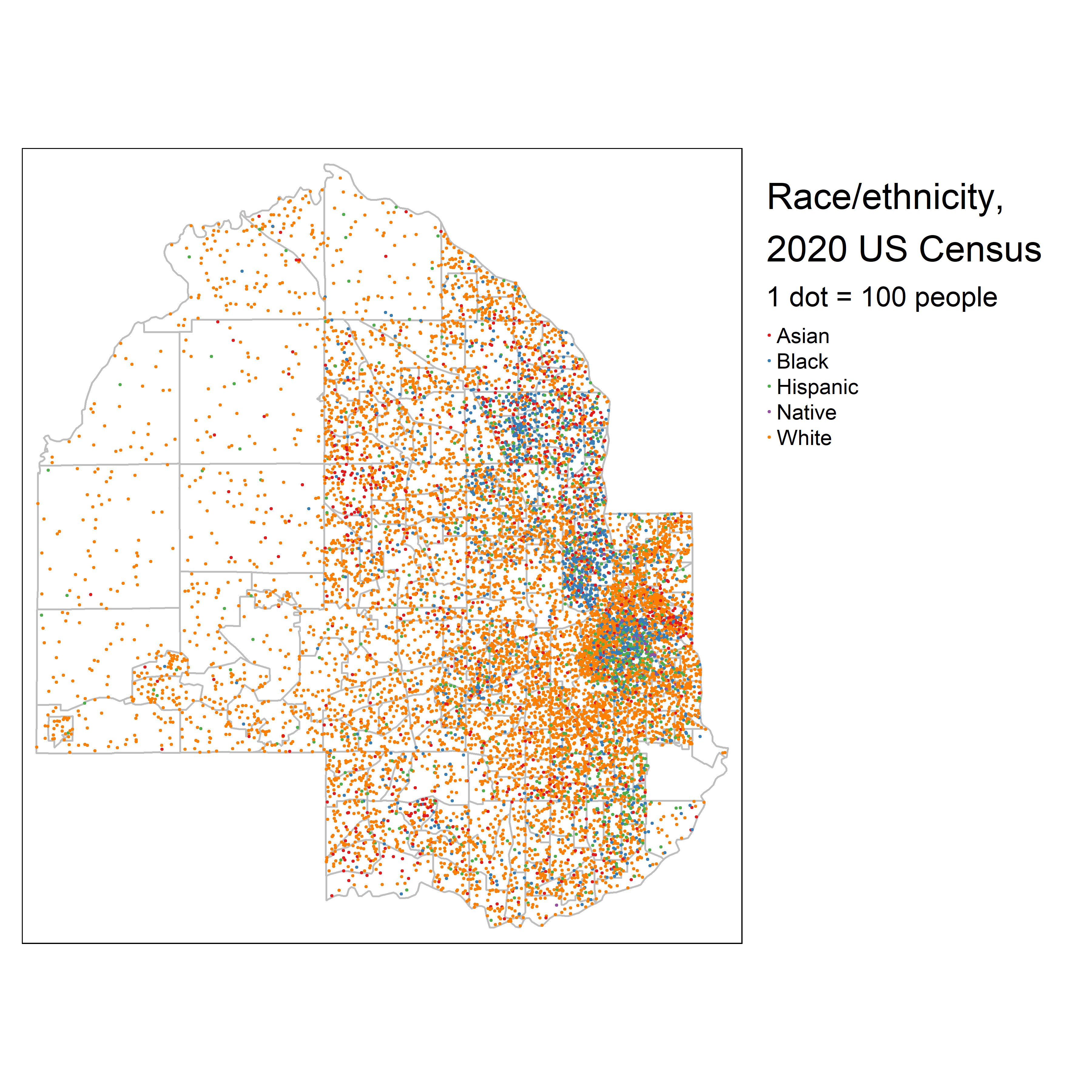


* Dot-density maps randomly place dots in polygons
  + the quantity of dots is proportional to the size of the attribute(s) plotted
  + different colored dots can represent different categories of a variable
* **tidycensus** has as\_dot\_density() to prepare Census data for dot-density mapping
  + it takes a value column as an argument for the attribute to be mapped
  + values per\_dot argument determines the proportion each dot represents of the whole
  + group argument determines how to partition dots, if subgroups exist within the data

```{r hennepin-dots}  
hennepin\_dots <- hennepin\_race %>%   
 as\_dot\_density(  
 value = "value",  
 values\_per\_dot = 100,  
 group = "variable"  
 )  
```

* the map is created with **tmap** tm\_dots() function
  + tm\_polygons() provides the background

```{r hennepin-dot-density-map}  
background\_tracts <- hennepin\_race  
  
tm\_shape(background\_tracts) +  
 tm\_polygons(col = "white",  
 border.col = "grey") +  
 tm\_shape(hennepin\_dots) +  
 tm\_dots(col = "variable",  
 palette = "Set1",  
 size = 0.005,  
 title = "1 dot = 100 people") +  
 tm\_layout(legend.outside = TRUE,  
 title = "Race/ethnicity, \n2020 US Census")  
```



* dot-density maps risk overplotting (obscuring data by dots being too close and becoming indiscernible
* dots in census tracts may be placed where people do not/cannot live, as water features, etc. are not taken into accounts
* dots follow boundaries rather than the whole map when randomly plotting; this can create the visual impression in sudden breaks in density
* one solution is *dasymetric dot-density mapping* - removing areas from polygons where people can’t live before plotting the dots.
  + **tidycensus** as\_dot\_density() function includes an erase\_water function to remove water areas

## 6.4 Cartographic workflows with non-Census data

* we may have data that is at a Census geography scale, but is not ACS or Census data proper
* with this data we cannot pull geometry down using **tidycensus** geometry = TRUE argument
* we may instead get shapes with **tigris** and join the data we have to Census geographies for visualization

### National election mapping with tigris shapes

* Election data is not provided by the Census bureau
* we will map data from the Cook Political Report for 2020 US Presidential election results
* first we acquire and import the data

```{r political-map}  
vote2020 <- read\_csv("data/us\_vote\_2020.csv")  
```

New names:  
Rows: 61 Columns: 22  
── Column specification  
──────────────────────────────────────────────────────── Delimiter: "," chr  
(10): state, called, final, dem\_percent, rep\_percent, other\_percent, dem... dbl  
(7): EV, X, Y, State\_num, Center\_X, Center\_Y, 2016 Margin num (4): dem\_votes,  
rep\_votes, other\_votes, Total 2016 Votes lgl (1): ...20  
ℹ Use `spec()` to retrieve the full column specification for this data. ℹ  
Specify the column types or set `show\_col\_types = FALSE` to quiet this message.  
• `` -> `...20`

```{r political-map}  
names(vote2020)  
```

[1] "state" "called" "final" "dem\_votes"   
 [5] "rep\_votes" "other\_votes" "dem\_percent" "rep\_percent"   
 [9] "other\_percent" "dem\_this\_margin" "margin\_shift" "vote\_change"   
[13] "stateid" "EV" "X" "Y"   
[17] "State\_num" "Center\_X" "Center\_Y" "...20"   
[21] "2016 Margin" "Total 2016 Votes"

* then we get the State-level geography we want with **tigris**

```{r states-pull}  
us\_states <- states(cb = TRUE, resolution = "20m") %>%   
 filter(NAME != "Puerto Rico") %>%   
 shift\_geometry()  
```

Retrieving data for the year 2021

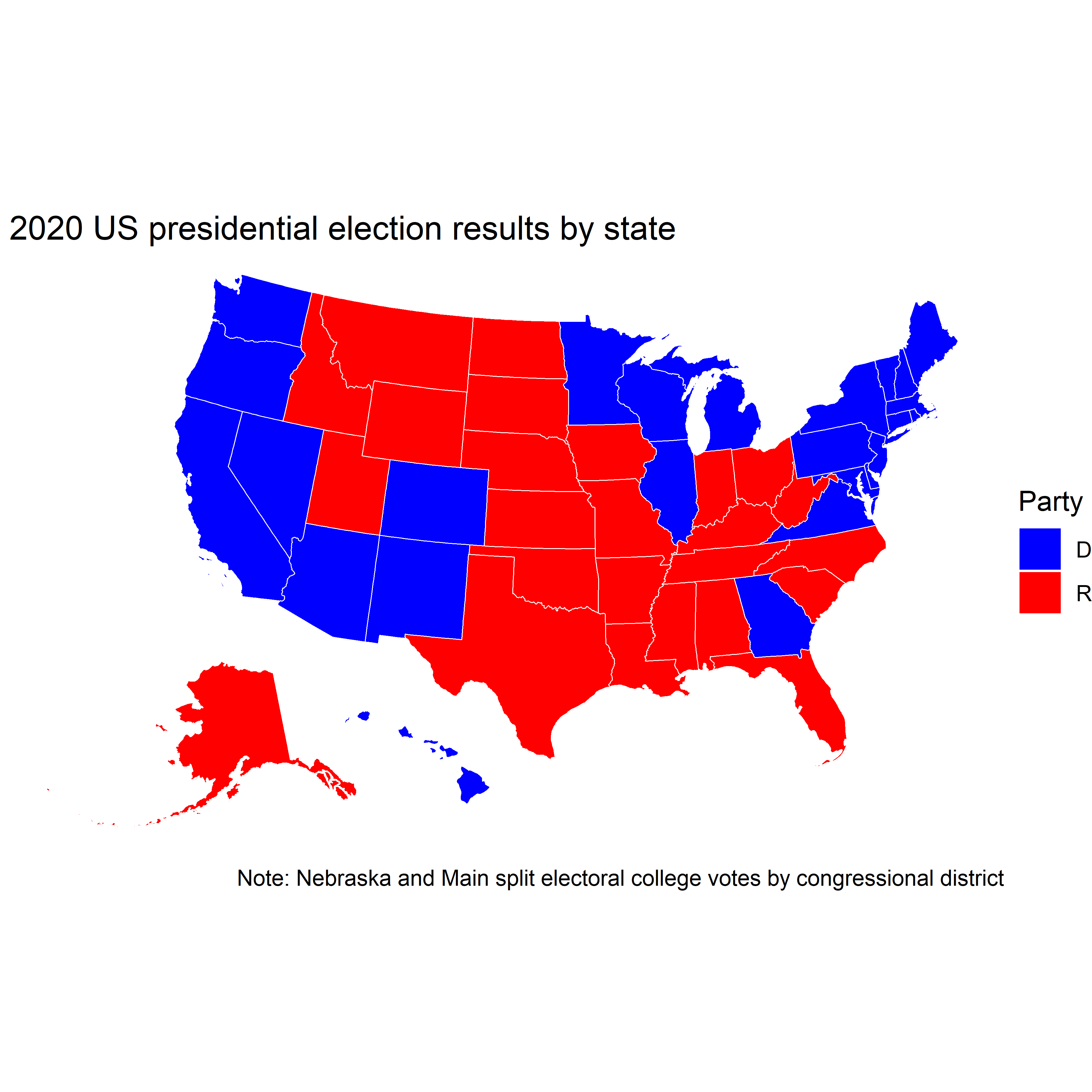
* finally, we use **dplyr** left\_join() to join the data table to the State geographies

```{r states-join}  
us\_states\_joined <- us\_states %>%   
 left\_join(vote2020, by = c("NAME" = "state"))  
  
# Quality check: did all NAME match with a state  
# we expect a false to be returned for each State and DC if the "state" name matches the "NAME" name  
table(is.na(us\_states\_joined$state))  
```

FALSE   
 51

* we used a left\_join() and started with us\_states because we are joining the data TO the states.
* now we plot

```{r election-map}  
ggplot(us\_states\_joined, aes(fill = called)) +  
 geom\_sf(color = "white", lwd = 0.2) +  
 scale\_fill\_manual(values = c("blue", "red")) +  
 theme\_void() +  
 labs(fill = "Party",  
 title = " 2020 US presidential election results by state",  
 caption = "Note: Nebraska and Main split electoral college votes by congressional district")  
```



### Understanding and working with ZCTAs

* many agencies only release data as granular as ZIP codes
  + ZIP codes represent *collections* of USPS routes, PO boxes, or individual buildings - they aren’t areal units
* US Census Bureau creates ZIP Code Tabulation Areas to approximate ZIP Code mapping
  + A ZCTAs is a shape built up by Census blocks based on their most common ZIP Code
* The IRS “Statistics of Income” data isn’t available at geographies more detailed than the ZIP Code

```{r irs-data}  
irs\_data <- read\_csv("https://www.irs.gov/pub/irs-soi/18zpallnoagi.csv")  
```

Rows: 27658 Columns: 153  
── Column specification ────────────────────────────────────────────────────────  
Delimiter: ","  
chr (3): STATEFIPS, STATE, ZIPCODE  
dbl (150): AGI\_STUB, N1, MARS1, MARS2, MARS4, ELF, CPREP, PREP, DIR\_DEP, N2,...  
  
ℹ Use `spec()` to retrieve the full column specification for this data.  
ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

```{r irs-data}  
ncol(irs\_data)  
```

[1] 153

* The data has 153 columns; one is ZIPCODE
* To map self-employment income, we can keep N09400 - the number of tax returns with self-employment tax, and N1 - which represents the total number of returns

```{r self-employment}  
self\_employment <- irs\_data %>%   
 select(ZIPCODE, self\_emp = N09400, total = N1)  
```

* we can now subset and map data in Boston, MA

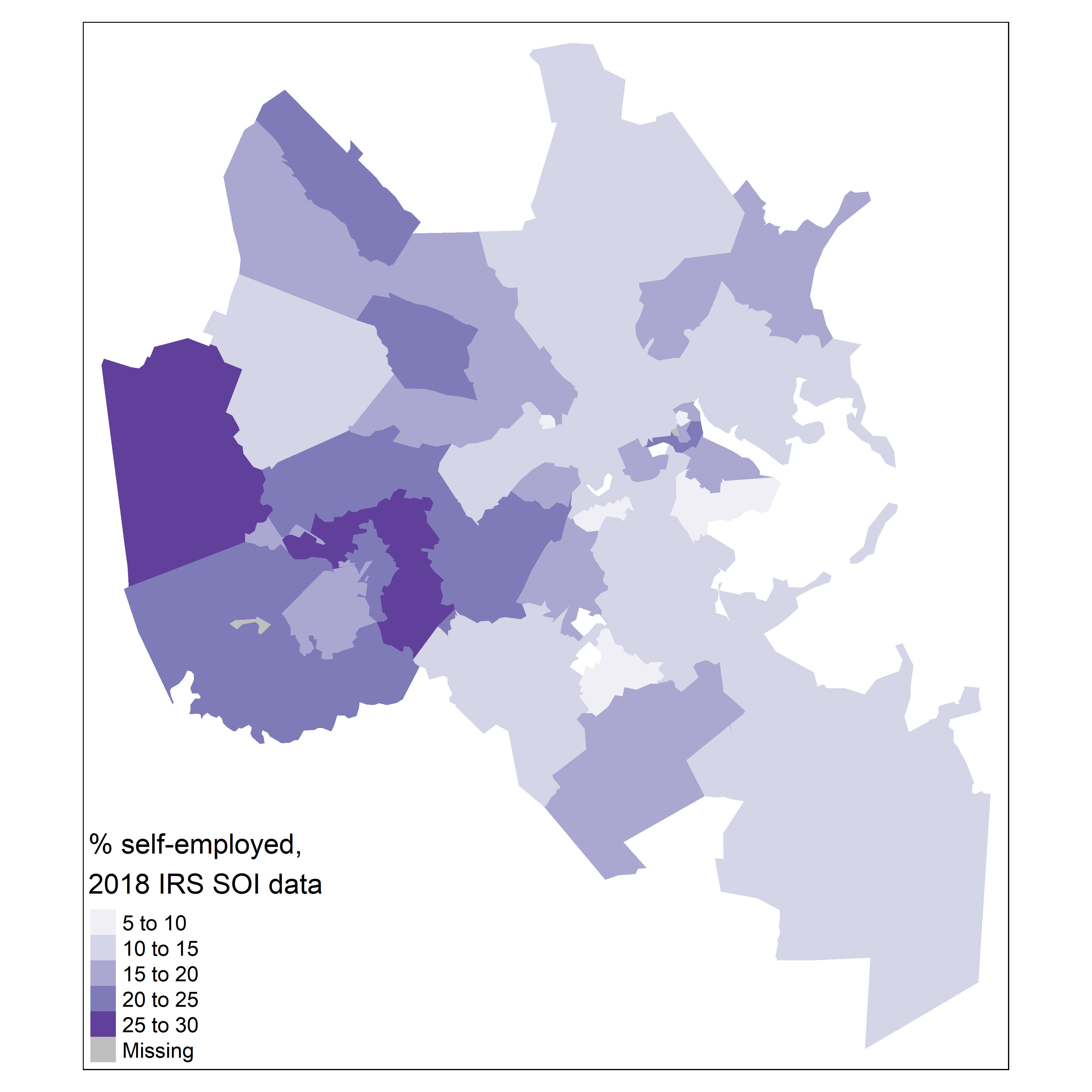
```{r}  
boston\_zctas <- zctas(  
 cb = TRUE,  
 starts\_with = c("021", "022", "024"),  
 year = 2018  
)  
  
mapview(boston\_zctas)  
```

* We can join the data-sets on "ZCTA5CE10" and "ZIPCODE"

```{r}  
boston\_se\_data <- boston\_zctas %>%   
 left\_join(self\_employment, by = c("GEOID10" = "ZIPCODE")) %>%   
 mutate(pct\_self\_emp = 100 \* (self\_emp / total)) %>%   
 select(GEOID10, self\_emp, pct\_self\_emp)  
```

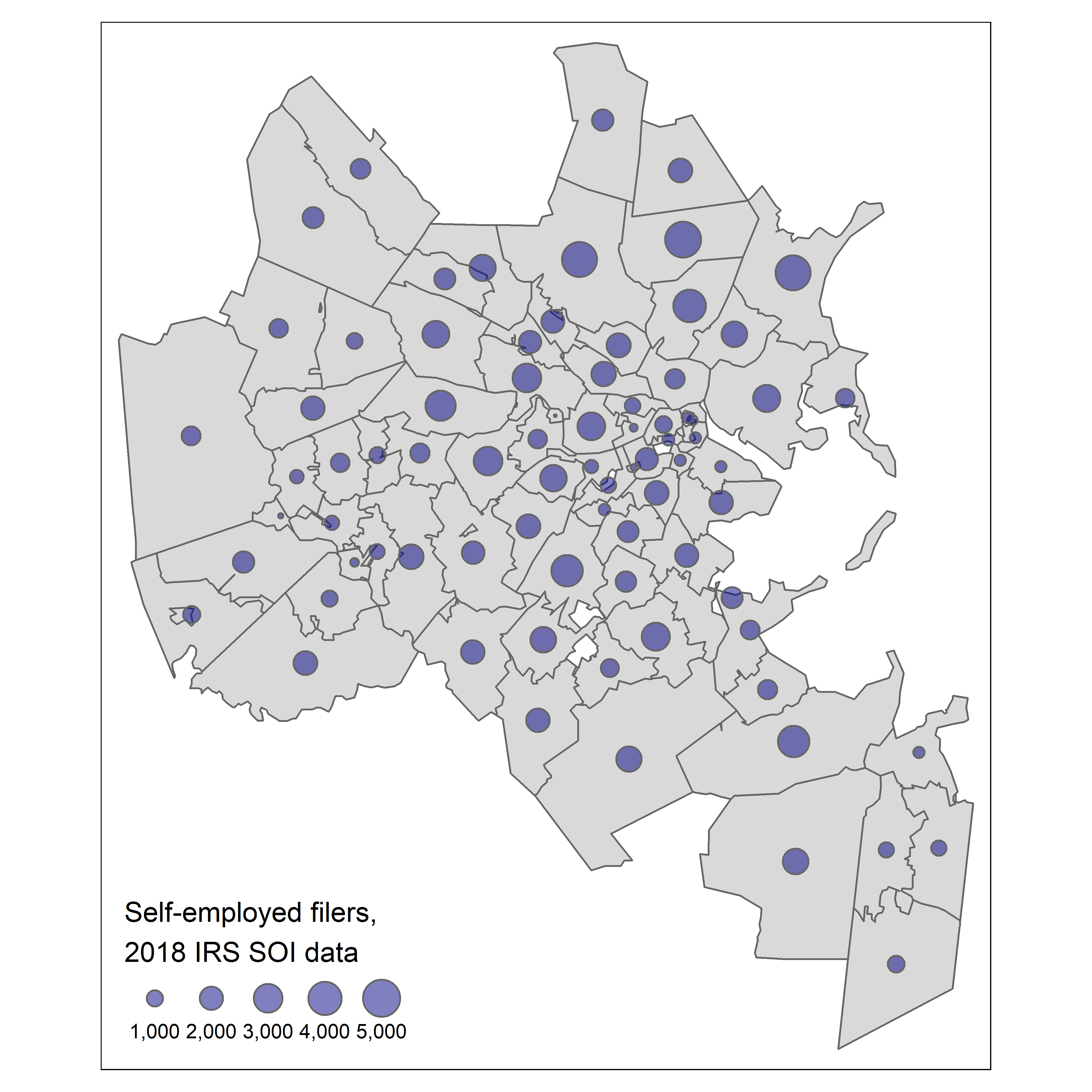
* We can finally build some plots of the data; first, we’ll make a choropleth map

```{r}  
tm\_shape(boston\_se\_data, projection = 26918) +  
 tm\_fill(col = "pct\_self\_emp",  
 palette = "Purples",  
 title = "% self-employed,\n2018 IRS SOI data") +  
 tm\_layout(legend.position = c("LEFT", "BOTTOM"))  
```



* the choropleth map shows higher rates of self-employment in suburbs
* If we want to see total numbers of filings, we would use self\_emp column and visualize it as a bubble map

```{r}  
tm\_shape(boston\_se\_data) +  
 tm\_polygons() +  
 tm\_bubbles(size = "self\_emp",  
 alpha = 0.5,  
 col = "navy",  
 title.size = "Self-employed filers,\n2018 IRS SOI data")  
```



## 6.5 Interactive mapping

* The book discusses several approaches to producing interactive, non-static maps

### Interactive mapping with Leaflet

* A popular webmap framework is the Leaflet Javascript Library
  + in r, we can use the **leaflet package**
* We will get data on percent of 25+ yo with bachelor’s degree or higher
  + we will look at the 2020 5Y ACS data
  + We will pull by Census Tract
  + We will pull for Dallas County, Texas

```{r dallas-BAs}  
dallas\_bachelors <- get\_acs(  
 geography = "tract",  
 variables = "DP02\_0068P",  
 year = 2020,  
 state = "TX",  
 county = "Dallas",  
 geometry = TRUE  
)  
```

Getting data from the 2016-2020 5-year ACS

Using the ACS Data Profile

* Mapview gives us the ability to use zcol to define a variable for choropleth mapping

```{r}  
# mapview(dallas\_bachelors, zcol = "estimate")  
```

* **tmap** also has a view mode that acts as an interactive Leaflet plot

```{r}  
# tmap\_mode("view")  
#   
# tm\_shape(dallas\_bachelors) +  
# tm\_fill(col = "estimate", palette = "magma",  
# alpha = 0.5)  
#   
# tmap\_mode("plot")  
```

* the **leaflet** package gives more fine-grained control
  + to reproduce the above examples with **leaflet**, run the following code chunks

```{r}  
# # define the color palette for 'magma'  
# pal <- colorNumeric(  
# palette = "magma",  
# domain = dallas\_bachelors$estimate  
# )  
#   
# pal(c(10, 20, 30, 40, 50))  
#   
# leaflet() %>% # initialize map  
# addProviderTiles(providers$Stamen.TonerLite) %>% # add basemap  
# addPolygons(data = dallas\_bachelors, # add census tracts  
# color = ~pal(estimate),  
# weight = 0.5,  
# smoothFactor = 0.2,  
# fillOpacity = 0.5,  
# label = ~estimate) %>%  
# addLegend( # add legend  
# position = "bottomright",  
# pal = pal,  
# values = dallas\_bachelors$estimate,  
# title = "% with bachelor's<br/>degree"  
# )  
```

### Alternative approaches to interactive mapping

* **leaflet** uses tile maps projected to Web Mercator
  + this distorts area near poles
* note the distortion of Alaska and near-invisibility of Alaska below

```{r}  
# us\_value <- get\_acs(  
# geography = "state",  
# variables = "B25077\_001",  
# year = 2019,  
# survey = "acs1",  
# geometry = TRUE,  
# resolution = "20m"  
# )  
#   
# us\_pal <- colorNumeric(  
# palette = "plasma",  
# domain = us\_value$estimate  
# )  
#   
# leaflet() %>%  
# addProviderTiles(providers$Stamen.TonerLite) %>%  
# addPolygons(data = us\_value,  
# color = ~us\_pal(estimate),  
# weight = 0.5,  
# smoothFactor = 0.2,  
# fillOpacity = 0.5,  
# label = ~estimate) %>%  
# addLegend(  
# position = "bottomright",  
# pal = us\_pal,  
# values = us\_value$estimate,  
# title = "Median home value"  
# )  
```

* If we want to make an interactive map for the whole of the US States, we can use packages such as **ggiraph**
  + This adds interactivity to a ggplot of the maps, rather than projecting onto a Leaflet webmap

```{r}  
# us\_value\_shifted <- us\_value %>%  
# shift\_geometry(position = "outside") %>%  
# mutate(tooltip = paste(NAME, estimate, sep = ": "))  
#   
# gg <- ggplot(us\_value\_shifted, aes(fill = estimate)) +   
# geom\_sf\_interactive(aes(tooltip = tooltip, data\_id = NAME),   
# size = 0.1) +   
# scale\_fill\_viridis\_c(option = "plasma", labels = label\_dollar()) +   
# labs(title = "Median housing value by State, 2019",  
# caption = "Data source: 2019 1-year ACS, US Census Bureau",  
# fill = "ACS estimate") +   
# theme\_void()   
#   
# girafe(ggobj = gg) %>%  
# girafe\_options(opts\_hover(css = "fill:cyan;"),   
# opts\_zoom(max = 10))  
```

## 6.6 Advanced examples

### Mapping migration flows

* **tidycensus** can call the ACS Migration Flows API
  + we can map migration flows by setting geometry = TRUE in the get\_flows() function
  + this returns two point columns linking locations in the flow
* Below we pull data for Travis County, Texas, a site of significant in-migration of late

```{r}  
travis\_inflow <- get\_flows(  
 geography = "county",  
 state = "TX",  
 county = "Travis",  
 geometry = TRUE  
) %>%   
 filter(variable == "MOVEDIN") %>%   
 na.omit() %>%   
 arrange(desc(estimate))  
```

* Now we can map the migration flows, IF we use **mapdeck** and are willing to set up API calls from MapBox, which requires providing credit card payment information

```{r}  
# NOTE THAT NO TOKEN IS ASSIGNED  
# travis\_inflow %>%  
# slice\_max(estimate, n = 30) %>%  
# mutate(weight = estimate / 500) %>%  
# mapdeck(token = token) %>%  
# add\_arc(origin = "centroid2",  
# destination = "centroid1",  
# stroke\_width = "weight",  
# update\_view = FALSE)   
```

### Linking maps and charts

* It is often useful to link visuals in an interactive manner
* below we link a map produced with **ggplot2** to a plot of error bars for the related margins of error
  + note that as one hovers over a polygon in the map, its affiliated error bar is highlighted

```{r}  
vt\_income <- get\_acs(  
 geography = "county",  
 variables = "B19013\_001",  
 state = "VT",  
 year = 2020,  
 geometry = TRUE  
) %>%   
 mutate(NAME = str\_remove(NAME, " County, Vermont"))  
```

Getting data from the 2016-2020 5-year ACS

```{r}  
vt\_map <- ggplot(vt\_income, aes(fill = estimate)) +  
 geom\_sf\_interactive(aes(data\_id = GEOID)) +  
 scale\_fill\_distiller(palette = "Greens",  
 direction = 1,  
 guide = "none") +  
 theme\_void()  
  
vt\_plot <- ggplot(vt\_income, aes(x = estimate, y = reorder(NAME, estimate),  
 fill = estimate)) +  
 geom\_errorbar(aes(xmin = estimate - moe, xmax = estimate + moe)) +  
 geom\_point\_interactive(color = "black", size = 4, shape = 21,  
 aes(data\_id = GEOID)) +  
 scale\_fill\_distiller(palette = "Greens", direction = 1,  
 labels = label\_dollar()) +  
 scale\_x\_continuous(labels = label\_dollar()) +  
 labs(title = "Household income by county in Vermont",  
 subtitle = "2016-2020 American Community Survey",  
 y = "",  
 x = "ACS estimate (bars represent margin of error)",  
 fill = "ACS estimate") +  
 theme\_minimal(base\_size = 14)  
  
girafe(ggobj = vt\_map + vt\_plot, width\_svg = 10, height\_svg = 5) %>%   
 girafe\_options(opts\_hover(css = "fill:cyan;"))  
```

### Reactive mapping with Shiny

* We can build interactive dashboard applications using Shiny
* Below, the author created a Shiny application to look at Minneapolis race data interactively

```{r}  
# # app.R  
# library(tidycensus)  
# library(shiny)  
# library(leaflet)  
# library(tidyverse)  
#   
# twin\_cities\_race <- get\_acs(  
# geography = "tract",  
# variables = c(  
# hispanic = "DP05\_0071P",  
# white = "DP05\_0077P",  
# black = "DP05\_0078P",  
# native = "DP05\_0079P",  
# asian = "DP05\_0080P",  
# year = 2019  
# ),  
# state = "MN",  
# county = c("Hennepin", "Ramsey", "Anoka", "Washington",  
# "Dakota", "Carver", "Scott"),  
# geometry = TRUE  
# )   
#   
# groups <- c("Hispanic" = "hispanic",  
# "White" = "white",  
# "Black" = "black",  
# "Native American" = "native",  
# "Asian" = "asian")  
#   
# ui <- fluidPage(  
# sidebarLayout(  
# sidebarPanel(  
# selectInput(  
# inputId = "group",  
# label = "Select a group to map",  
# choices = groups  
# )  
# ),  
# mainPanel(  
# leafletOutput("map", height = "600")  
# )  
# )  
# )  
#   
# server <- function(input, output) {  
#   
# # Reactive function that filters for the selected group in the drop-down menu  
# group\_to\_map <- reactive({  
# filter(twin\_cities\_race, variable == input$group)  
# })  
#   
# # Initialize the map object, centered on the Minneapolis-St. Paul area  
# output$map <- renderLeaflet({  
#   
# leaflet(options = leafletOptions(zoomControl = FALSE)) %>%  
# addProviderTiles(providers$Stamen.TonerLite) %>%  
# setView(lng = -93.21,  
# lat = 44.98,  
# zoom = 8.5)  
#   
# })  
#   
# observeEvent(input$group, {  
#   
# pal <- colorNumeric("viridis", group\_to\_map()$estimate)  
#   
# leafletProxy("map") %>%  
# clearShapes() %>%  
# clearControls() %>%  
# addPolygons(data = group\_to\_map(),  
# color = ~pal(estimate),  
# weight = 0.5,  
# fillOpacity = 0.5,  
# smoothFactor = 0.2,  
# label = ~estimate) %>%  
# addLegend(  
# position = "bottomright",  
# pal = pal,  
# values = group\_to\_map()$estimate,  
# title = "% of population"  
# )  
# })  
#   
# }  
#   
# shinyApp(ui = ui, server = server)  
```

## 6.7 Working with software outside of R for cartographic projects

### Exporting maps from R

* you can export **ggplot2** maps using ggsave()
  + when using theme\_void(), set a bg color or the map will be transparent
* tmap\_save() exports **tmap** maps
  + tmap\_save() requires an object to export

```{r}  
hennepin\_map <- tm\_shape(hennepin\_black) +  
 tm\_polygons(col = "percent",  
 style = "jenks",  
 n = 5,  
 palette = "Purples",  
 title = "ACS estimate",  
 legend.hist = TRUE) +  
 tm\_layout(title = "Percent Black\nby Census tract",  
 frame = FALSE,  
 legend.outside = TRUE,  
 bg.color = "grey70",  
 legend.hist.width = 5,  
 fontfamily = "Verdana")  
```

* Once you have an object to export, tmap\_save() gives you a lot of control over specifications

```{r}  
tmap\_save(  
 tm = hennepin\_map,  
 filename = "./images/hennepin\_map.png",  
 height = 5.5,  
 width = 8,  
 dpi = 300  
)  
```

Warning in grid.Call.graphics(C\_text, as.graphicsAnnot(x$label), x$x, x$y, :  
font family not found in Windows font database  
  
Warning in grid.Call.graphics(C\_text, as.graphicsAnnot(x$label), x$x, x$y, :  
font family not found in Windows font database  
  
Warning in grid.Call.graphics(C\_text, as.graphicsAnnot(x$label), x$x, x$y, :  
font family not found in Windows font database

Map saved to C:\Users\dacarva\Documents\R\_Mats\census\_tidy\images\hennepin\_map.png

Resolution: 2400 by 1650 pixels

Size: 8 by 5.5 inches (300 dpi)

* interactive maps can be written to HTML using htmlwidgets::saveWidget()
  + assign the Leaflet map to a variable
  + pass the variable to saveWidget()
  + **mapview** maps can be exported the same way, but require a call to the map slot using @
    - this is demonstrated below
  + selfcontained = TRUE bundles assets in the html file
    - setting it to FALSE creates a separate directory for assets which is then referenced by the HTML
  + **tmap** interactive maps are saved the same way, only using tmap\_save() instead of saveWidget()

```{r}  
dallas\_map <- mapview(dallas\_bachelors, zcol = "estimate")  
  
saveWidget(dallas\_map@map, "dallas\_mapview\_map.html", selfcontained = TRUE)  
```

### Interoperability with other visualization software

* data pulled down using **tidycensus** can be exported to a shapefile for importing into other GIS software
  + this is done with **sf** package’s st\_write() function

```{r dc-export}  
dc\_income <- get\_acs(  
 geography = "tract",  
 variables = "B19013\_001",  
 state = "DC",  
 year = 2020,  
 geometry = TRUE  
)  
```

Getting data from the 2016-2020 5-year ACS

```{r dc-export}  
st\_write(dc\_income, "dc\_income.shp")  
```

Writing layer `dc\_income' to data source `dc\_income.shp' using driver `ESRI Shapefile'  
Writing 206 features with 5 fields and geometry type Polygon.

You can also develop R scripts that function as QGIS plugins using QGIS’s Processing R Provider plugin.

1. Install the plugin in QGIS
2. click Processing > Toolbox
3. Click the R icon
4. Click Create New R Script…
5. Write your script in the R editor
   * When writing the script, define the parameters at the beginning of the script with ##
6. Once you’ve finished the script, save it with an appropriate tool name and the extension .rsx
7. You may now open it in QGIS as a tool using the Processing Toolbox, GUI and everything

## 6.8 Exercises

* Exercise 1: make a race/ethnicity map for a different county

```{r la-dot-map}  
los\_angeles\_race <- get\_decennial(  
 geography = "tract",  
 state = "CA",  
 county = "Los Angeles",  
 variables = c(  
 Hispanic = "P2\_002N",  
 White = "P2\_005N",  
 Black = "P2\_006N",  
 Native = "P2\_007N",  
 Asian = "P2\_008N"  
 ),  
 summary\_var = "P2\_001N",  
 year = 2020,  
 geometry = TRUE  
) %>%   
 mutate(percent = 100 \* (value / summary\_value))  
```

Getting data from the 2020 decennial Census

Using the PL 94-171 Redistricting Data Summary File

```{r la-dot-map}  
los\_angeles\_dots <- los\_angeles\_race %>%   
 as\_dot\_density(  
 value = "value",  
 values\_per\_dot = 100,  
 group = "variable"  
)  
  
background\_tracts <- filter(los\_angeles\_race, variable == "White")  
  
# tmap\_mode("view")  
  
tm\_shape(st\_make\_valid(background\_tracts)) +   
 tm\_polygons(col = "white",   
 border.col = "grey") +   
 tm\_shape(los\_angeles\_dots) +  
 tm\_dots(col = "variable",   
 palette = "Set1",  
 size = 0.005,   
 title = "1 dot = 100 people") +   
 tm\_layout(legend.outside = TRUE,  
 title = "Race/ethnicity,\n2020 US Census")  
```

Warning: The shape st\_make\_valid(background\_tracts) contains empty units.

```{r la-dot-map}  
# tmap\_mode("plot")  
```

* Exercise 2: find and map a different variable using tidycensus::load\_variables()

```{r multiracial-la-census-tracts}  
multir\_la <- get\_decennial("tract", variables = "P1\_009N", year = 2020, summary\_var = "P1\_001N", state = "CA", county = "Los Angeles", geometry = TRUE) %>% mutate(percent = 100 \* (value / summary\_value))  
```

Getting data from the 2020 decennial Census

Using the PL 94-171 Redistricting Data Summary File

```{r multiracial-la-census-tracts}  
multir\_la\_clean <- multir\_la %>% filter(summary\_value >= 100)  
  
ggplot() +  
 geom\_sf(data = multir\_la, fill = "grey", color = NA) +  
 geom\_sf(data = multir\_la\_clean, aes(fill = percent), color = NA) +  
 theme\_gray()  
```

