Tables & Geographic data

Maithreyi Gopalan

Week 8

```
#install.packages("Cairo")
```

Agenda

- Wrap up websites
- Tables
  - {gt}
  - {kableExtra} (quickly)
  - reactable
- Geographic data
  - Vector/raster data
  - Producing basic maps

Learning objectives: Tables

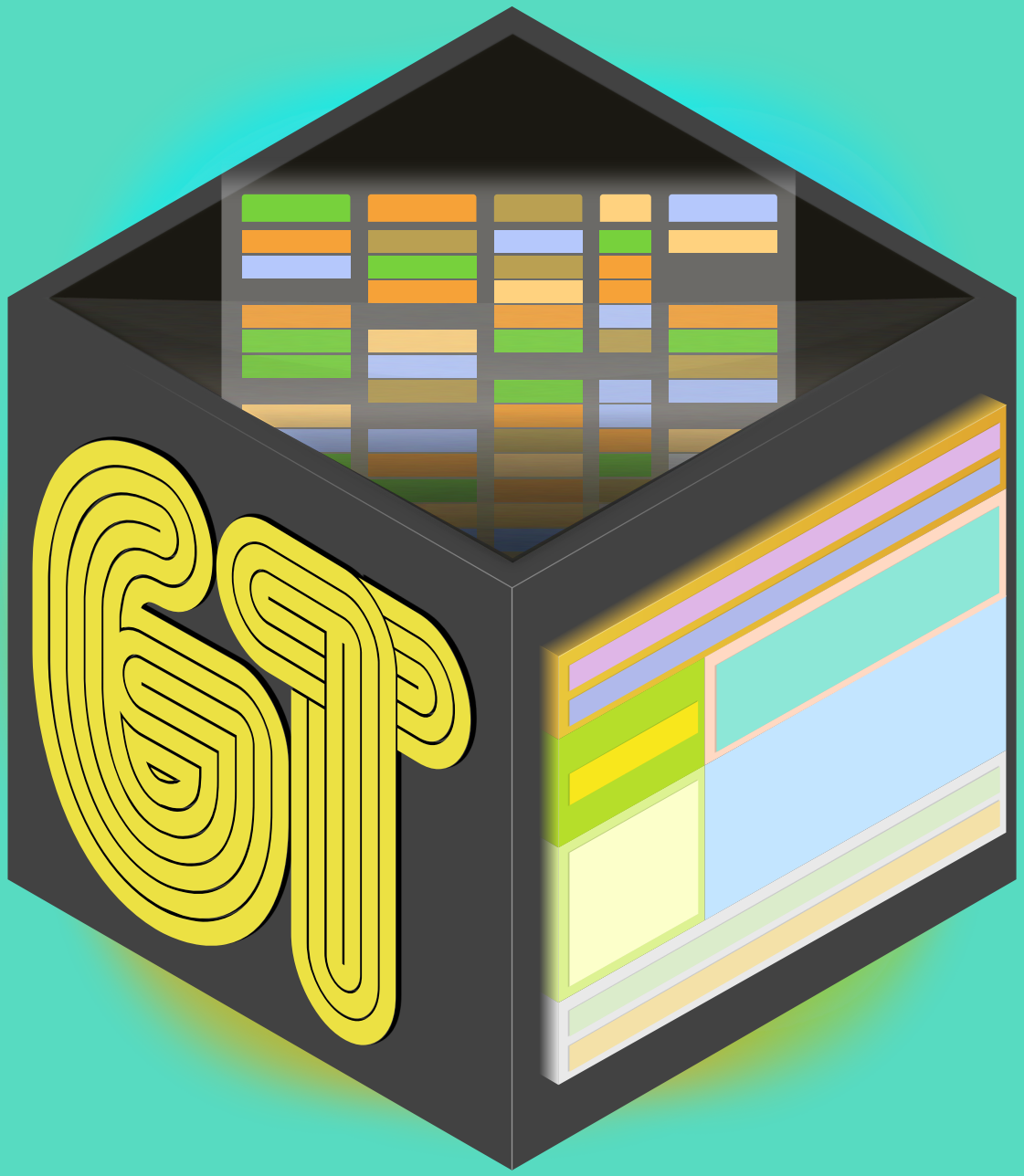
- Be comfortable with the basics of `gt`
  - create a table
  - format columns
  - create spanner heads
  - etc.

Learning objectives: geo

- Know the difference between vector and raster data
- Be able to produce basic maps using a variety of tools
- Be able to obtain different types of geographic data from a few different places
- Be able to produce basic interactive maps
- Understand the basics of the R geospatial ecosystem

Wrap up websites

Tables



Overview

- Pipe-oriented
- Beautiful tables easy
- Spanner heads/grouping used to be a total pain - not so anymore
- Renders to HTML/PDF without even thinking about it

Probably my favorite package for creating static tables, although **kableExtra** is great too.

My experience is that fewer people are generally familiar with **gt**, which is why I cover it here.

Install

```
#install.packages("gt")

# or

remotes::install_github("rstudio/gt")
```

Please follow along

01:00

The hard part

- Getting your data in the format you want a table in
- Utilize your `pivot_*` skills regularly

```
library(fivethirtyeight)
flying
```

```
## # A tibble: 1,040 × 27
##    respondent_id gender age   height   children_under_18 household_income
##            <dbl> <chr>  <ord> <ord>    <lgl>             <ord>
##  1    3436139758 <NA>   <NA>   <NA>    NA                <NA>
##  2    3434278696 Male   30-44 "6'3\""  TRUE              <NA>
##  3    3434275578 Male   30-44 "5'8\""  FALSE             $100,000 - $149,999
##  4    3434268208 Male   30-44 "5'11\"" FALSE             $0 - $24,999
##  5    3434250245 Male   30-44 "5'7\""  FALSE             $50,000 - $99,999
##  6    3434245875 Male   30-44 "5'9\""  TRUE              $25,000 - $49,999
##  7    3434235351 Male   30-44 "6'2\""  TRUE              <NA>
##  8    3434218031 Male   30-44 "6'0\""  TRUE              $0 - $24,999
##  9    3434213681 <NA>   <NA>  "6'0\""  TRUE              <NA>
## 10    3434172894 Male   30-44 "5'6\""  FALSE             $0 - $24,999
## # ℹ 1,030 more rows
## # ℹ 21 more variables: education <ord>, location <chr>, frequency <ord>,
## #   recline_frequency <ord>, recline_obligation <lgl>, recline_rude <ord>,
```

```
flying %>%
  count(gender, age, recline_frequency)
```

```
## # A tibble: 53 × 4
##    gender age   recline_frequency       n
##    <chr>  <ord> <ord>               <int>
##  1 Female 18-29 Never                  24
##  2 Female 18-29 Once in a while        36
##  3 Female 18-29 About half the time    10
##  4 Female 18-29 Usually                13
##  5 Female 18-29 Always                 10
##  6 Female 18-29 <NA>                   19
##  7 Female 30-44 Never                  21
##  8 Female 30-44 Once in a while        25
##  9 Female 30-44 About half the time    22
## 10 Female 30-44 Usually                22
## # ℹ 43 more rows
```

```r
smry <- flying %>%
  count(gender, age, recline_frequency) %>%
  drop_na(age,recline_frequency) %>%
  pivot_wider(
    names_from = "age",
    values_from = "n"
  )
smry
```

```
## # A tibble: 10 × 6
##    gender recline_frequency   `18-29` `30-44` `45-60` `> 60`
##    <chr>  <ord>                 <int>   <int>   <int>  <int>
##  1 Female Never                    24      21      19     23
##  2 Female Once in a while          36      25      30     36
##  3 Female About half the time      10      22      18     17
##  4 Female Usually                  13      22      26     28
##  5 Female Always                   10      21      29     12
##  6 Male   Never                    24      17      20     18
##  7 Male   Once in a while          19      39      40     29
##  8 Male   About half the time      11      11      16     11
##  9 Male   Usually                  14      30      15     27
## 10 Male   Always                   11      14      21     14
```

Turn into table

```
library(gt)
smry %>%
  gt()
```

Disclaimer: these all look slightly different on the slides

The way they look for you locally is how they will render in standard R Markdown files

gender	recline_frequency	18-29	30-44	45-60	> 60
Female	Never	24	21	19	23
Female	Once in a while	36	25	30	36
Female	About half the time	10	22	18	17
Female	Usually	13	22	26	28
Female	Always	10	21	29	12
Male	Never	24	17	20	18
Male	Once in a while	19	39	40	29
Male	About half the time	11	11	16	11
Male	Usually	14	30	15	27
Male	Always	11	14	21	14

Add gender as a grouping variable

```
smry %>%
  group_by(gender) %>%
  gt()
```

recline_frequency	18-29	30-44	45-60	> 60
Female				
Never	24	21	19	23
Once in a while	36	25	30	36
About half the time	10	22	18	17
Usually	13	22	26	28
Always	10	21	29	12
Male				
Never	24	17	20	18
Once in a while	19	39	40	29
About half the time	11	11	16	11
Usually	14	30	15	27
Always	11	14	21	14

This is an example of a table that looks better with the default CSS

Add a spanner head

```
smry %>%
  group_by(gender) %>%
  gt() %>%
  tab_spanner(
    label = "Age Range",
    columns = vars(`18-29`, `30-44`, `45-60`, `> 60`)
  )
```

	Age Range			
recline_frequency	18-29	30-44	45-60	> 60
Female				
Never	24	21	19	23
Once in a while	36	25	30	36
About half the time	10	22	18	17
Usually	13	22	26	28
Always	10	21	29	12
Male				
Never	24	17	20	18
Once in a while	19	39	40	29
About half the time	11	11	16	11
Usually	14	30	15	27
Always	11	14	21	14

Change column names

```
smry %>%
  group_by(gender) %>%
  gt() %>%
  tab_spanner(
    label = "Age Range",
    columns = vars(`18-29`, `30-44`, `45-60`, `> 60`)
  ) %>%
  cols_label(recline_frequency = "Recline")
```

	Age Range			
Recline	18-29	30-44	45-60	> 60
Female				
Never	24	21	19	23
Once in a while	36	25	30	36
About half the time	10	22	18	17
Usually	13	22	26	28
Always	10	21	29	12
Male				
Never	24	17	20	18
Once in a while	19	39	40	29
About half the time	11	11	16	11
Usually	14	30	15	27
Always	11	14	21	14

Align columns

```
smry %>%
  group_by(gender) %>%
  gt() %>%
  tab_spanner(
    label = "Age Range",
    columns = vars(`18-29`, `30-44`, `45-60`, `> 60`)
  ) %>%
  cols_label(recline_frequency = "Recline") %>%
  cols_align(align = "left",
             columns = vars(recline_frequency))
```

	Age Range			
Recline	18-29	30-44	45-60	> 60
Female				
Never	24	21	19	23
Once in a while	36	25	30	36
About half the time	10	22	18	17
Usually	13	22	26	28
Always	10	21	29	12
Male				
Never	24	17	20	18
Once in a while	19	39	40	29
About half the time	11	11	16	11
Usually	14	30	15	27
Always	11	14	21	14

Add a title

```
smry %>%
  group_by(gender) %>%
  gt() %>%
  tab_spanner(
    label = "Age Range",
    columns = vars(`18-29`, `30-44`, `45-60`, `> 60`)
  ) %>%
  cols_label(recline_frequency = "Recline") %>%
  cols_align(align = "left",
             columns = vars(recline_frequency)) %>%
  tab_header(
    title = "Airline Passengers",
    subtitle = "Leg space is limited, what do you do?"
  )
```

Airline Passengers

Leg space is limited, what do you do?

	Age Range			
Recline	18-29	30-44	45-60	> 60
Female				
Never	24	21	19	23
Once in a while	36	25	30	36
About half the time	10	22	18	17
Usually	13	22	26	28
Always	10	21	29	12
Male				
Never	24	17	20	18
Once in a while	19	39	40	29
About half the time	11	11	16	11
Usually	14	30	15	27
Always	11	14	21	14

Format columns

```
smry %>%
  mutate(across(c(`18-29`, `30-44`, `45-60`, `> 60`),
                ~.x/100)) %>%
  group_by(gender) %>%
  gt() %>%
  tab_spanner(
    label = "Age Range",
    columns = vars(`18-29`, `30-44`, `45-60`, `> 60`)
  ) %>%
  fmt_percent(
    vars(`18-29`, `30-44`, `45-60`, `> 60`),
    decimals = 0
  ) %>%
  cols_label(recline_frequency = "Recline") %>%
  cols_align(align = "left",
             columns = vars(recline_frequency)) %>%
  tab_header(
    title = "Airline Passengers",
    subtitle = "Leg space is limited, what do you do?"
  )
```

Airline Passengers

Leg space is limited, what do you do?

	Age Range			
Recline	18-29	30-44	45-60	> 60
Female				
Never	24%	21%	19%	23%
Once in a while	36%	25%	30%	36%
About half the time	10%	22%	18%	17%
Usually	13%	22%	26%	28%
Always	10%	21%	29%	12%
Male				
Never	24%	17%	20%	18%
Once in a while	19%	39%	40%	29%
About half the time	11%	11%	16%	11%
Usually	14%	30%	15%	27%
Always	11%	14%	21%	14%

Add a source note

```
smry %>%
  mutate(across(c(`18-29`, `30-44`, `45-60`, `> 60`),
                ~.x/100)) %>%
  group_by(gender) %>%
  gt() %>%
  tab_spanner(
    label = "Age Range",
    columns = vars(`18-29`, `30-44`, `45-60`, `> 60`)
  ) %>%
  fmt_percent(
    vars(`18-29`, `30-44`, `45-60`, `> 60`),
    decimals = 0
  ) %>%
  cols_label(recline_frequency = "Recline") %>%
  cols_align(align = "left",
             columns = vars(recline_frequency)) %>%
  tab_header(
    title = "Airline Passengers",
    subtitle = "Leg space is limited, what do you do?"
  ) %>%
  tab_source_note(
    source_note = md("Data from [fivethirtyeight](https://fiveth
  )
```

Airline Passengers

Leg space is limited, what do you do?

	Age Range			
Recline	18-29	30-44	45-60	> 60
Female				
Never	24%	21%	19%	23%
Once in a while	36%	25%	30%	36%
About half the time	10%	22%	18%	17%
Usually	13%	22%	26%	28%
Always	10%	21%	29%	12%
Male				
Never	24%	17%	20%	18%
Once in a while	19%	39%	40%	29%
About half the time	11%	11%	16%	11%
Usually	14%	30%	15%	27%
Always	11%	14%	21%	14%

Data from fivethirtyeight

Color cells

```
... %>%
  data_color(
    vars(`18-29`, `30-44`, `45-60`, `> 60`),
    colors = scales::col_numeric(
      palette = c("#FFFFFF", "#FF0000"),
      domain = NULL
    )
  ) %>%
  ...
```

Airline Passengers

Leg space is limited, what do you do?

	Age Range			
Recline	18-29	30-44	45-60	> 60
Female				
Never	24%	21%	19%	23%
Once in a while	36%	25%	30%	36%
About half the time	10%	22%	18%	17%
Usually	13%	22%	26%	28%
Always	10%	21%	29%	12%
Male				
Never	24%	17%	20%	18%
Once in a while	19%	39%	40%	29%
About half the time	11%	11%	16%	11%
Usually	14%	30%	15%	27%
Always	11%	14%	21%	14%

Data from fivethirtyeight

What else?

- Lots more it can do, see the website

Thomas Mock does a lot of great work with tables and often has tutorials showing your how to go further (e.g., see here and here and here).

A few other table options

kableExtra

A few quick examples

Make sure to specify `results = "asis"` in your chunk options.

```
library(knitr)
library(kableExtra)
dt <- mtcars[1:5, 1:6]
kable(dt) %>%
  kable_styling("striped") %>%
  column_spec(5:7, bold = TRUE
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	**110**	**3.90**	**2.620**
Mazda RX4 Wag	21.0	6	160	**110**	**3.90**	**2.875**
Datsun 710	22.8	4	108	**93**	**3.85**	**2.320**
Hornet 4 Drive	21.4	6	258	**110**	**3.08**	**3.215**
Hornet Sportabout	18.7	8	360	**175**	**3.15**	**3.440**

```
kable(dt) %>%
  kable_styling("striped") %>%
  column_spec(5:7, bold = TRUE) %>%
  row_spec(c(2, 4),
           bold = TRUE,
           color = "#EFF3F7",
           background = "#71B0DE")
```

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	**110**	**3.90**	**2.620**
Mazda RX4 Wag	**21.0**	**6**	**160**	**110**	**3.90**	**2.875**
Datsun 710	22.8	4	108	**93**	**3.85**	**2.320**
Hornet 4 Drive	**21.4**	**6**	**258**	**110**	**3.08**	**3.215**
Hornet Sportabout	18.7	8	360	**175**	**3.15**	**3.440**

```
kable(dt) %>%
  kable_styling("striped", full_width = FALSE) %>%
  pack_rows(
    "Group 1", 1, 3,
    label_row_css = "background-color: #666; color: #fff;"
  ) %>%
  pack_rows(
    "Group 2", 4, 5,
    label_row_css = "background-color: #666; color: #fff;"
  )
```

	mpg	cyl	disp	hp	drat	wt
Group 1						
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Group 2						
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440

KableExtra wrapup

Many other options, please see the documentation. Works well for PDF and HTML.

What about Microsoft Word?

flextable

flextable 0.5.8 | Overview | Selectors | Layout | Format visual properties | Format Content | Render as image | Examples | Function reference

flextable R package

build passing | BUILD PASSING | CRAN 0.5.8 | downloads 16K/month | repo status Active

The flextable package provides a framework for easily create tables for reporting and publications. Tables can be embedded within:

- R Markdown documents with support for HTML, Word and PowerPoint documents.
- Microsoft Word or PowerPoint documents.
- PDF documents with package `pagedown` (it's only HTML)

Tables can also be rendered as R plots or graphic files (png, pdf and jpeg).

Getting Started

An API is available to let R users create tables for reporting and control their formatting properties and their layout. A `flextable` object is a data.frame representation, it can be manipulated with functions that give control over:

Links

Download from CRAN at
https://cloud.r-project.org/package=flextable

Report a bug at
https://github.com/davidgohel/flextable/issues

Visit ARDATA website at
https://www.ardata.fr

License

GPL-3

Developers

David Gohel
Author, maintainer

All authors...

Many others

- huxtable
- formattable
- DT (my former favorite for shiny)
- rhandsontable

Particularly helpful for modeling

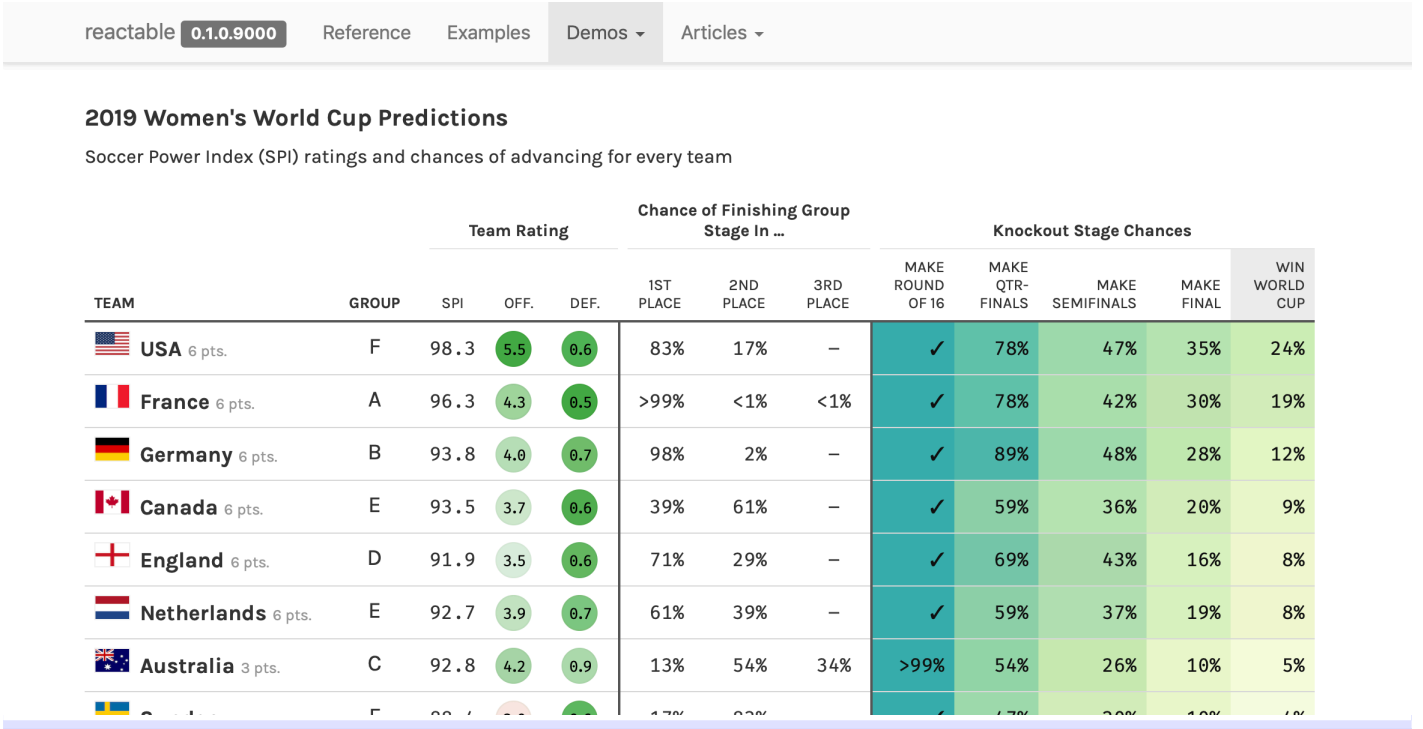
- stargazer
- pixiedust
- modelsummary

For descriptives

- gtsummary

reactable

My favorite for interactive tables



Works great with **shiny** too

Penguins data

```
library(palmerpenguins)
library(reactable)
reactable(penguins)
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
Adelie	Torgersen	39.1	18.7	181	3750	male
Adelie	Torgersen	39.5	17.4	186	3800	female
Adelie	Torgersen	40.3	18	195	3250	female
Adelie	Torgersen					
Adelie	Torgersen	36.7	19.3	193	3450	female
Adelie	Torgersen	39.3	20.6	190	3650	male
Adelie	Torgersen	38.9	17.8	181	3625	female
Adelie	Torgersen	39.2	19.6	195	4675	male
Adelie	Torgersen	34.1	18.1	193	3475	
Adelie	Torgersen	42	20.2	190	4250	

1–10 of 344 rows

Previous **1** 2 3 4 5 … 35 Next

Rename columns

```
penguins %>%
  reactable(
    columns = list(
      bill_length_mm = colDef(name = "Bill Length (mm)"),
      bill_depth_mm = colDef(name = "Bill Depth (mm)")
    )
  )
```

species	island	Bill Length (mm)	Bill Depth (mm)	flipper_length_mm	body_mass_g	sex
Adelie	Torgersen	39.1	18.7	181	3750	male
Adelie	Torgersen	39.5	17.4	186	3800	female
Adelie	Torgersen	40.3	18	195	3250	female
Adelie	Torgersen					
Adelie	Torgersen	36.7	19.3	193	3450	female
Adelie	Torgersen	39.3	20.6	190	3650	male
Adelie	Torgersen	38.9	17.8	181	3625	female
Adelie	Torgersen	39.2	19.6	195	4675	male
Adelie	Torgersen	34.1	18.1	193	3475	
Adelie	Torgersen	42	20.2	190	4250	

1–10 of 344 rows Previous 1 2 3 4 5 ... 35 Next

Or use a function

```
library(stringr)

penguins %>%
  reactable(
    defaultColDef = colDef(
      header = function(x) str_to_title(gsub("_", " ", x))
    )
  )
```

Species	Island	Bill Length Mm	Bill Depth Mm	Flipper Length Mm	Body Mass G	Sex
Adelie	Torgersen	39.1	18.7	181	3750	male
Adelie	Torgersen	39.5	17.4	186	3800	female
Adelie	Torgersen	40.3	18	195	3250	female
Adelie	Torgersen					
Adelie	Torgersen	36.7	19.3	193	3450	female
Adelie	Torgersen	39.3	20.6	190	3650	male
Adelie	Torgersen	38.9	17.8	181	3625	female
Adelie	Torgersen	39.2	19.6	195	4675	male
Adelie	Torgersen	34.1	18.1	193	3475	
Adelie	Torgersen	42	20.2	190	4250	

1–10 of 344 rows

Previous **1** 2 3 4 5 ... 35 Next

Add filter

```
reactable(penguins, filterable = TRUE)
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
Adelie	Torgersen	39.1	18.7	181	3750	male
Adelie	Torgersen	39.5	17.4	186	3800	female
Adelie	Torgersen	40.3	18	195	3250	female
Adelie	Torgersen					
Adelie	Torgersen	36.7	19.3	193	3450	female
Adelie	Torgersen	39.3	20.6	190	3650	male
Adelie	Torgersen	38.9	17.8	181	3625	female
Adelie	Torgersen	39.2	19.6	195	4675	male
Adelie	Torgersen	34.1	18.1	193	3475	
Adelie	Torgersen	42	20.2	190	4250	

1–10 of 344 rows Previous **1** 2 3 4 5 … 35 Next

Searchable

```
reactable(penguins, searchable = TRUE)
```

Search

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
Adelie	Torgersen	39.1	18.7	181	3750	male
Adelie	Torgersen	39.5	17.4	186	3800	female
Adelie	Torgersen	40.3	18	195	3250	female
Adelie	Torgersen					
Adelie	Torgersen	36.7	19.3	193	3450	female
Adelie	Torgersen	39.3	20.6	190	3650	male
Adelie	Torgersen	38.9	17.8	181	3625	female
Adelie	Torgersen	39.2	19.6	195	4675	male
Adelie	Torgersen	34.1	18.1	193	3475	
Adelie	Torgersen	42	20.2	190	4250	

1–10 of 344 rows

Previous **1** 2 3 4 5 ... 35 Next

Pagination

```
reactable(penguins, defaultPageSize = 3)
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
Adelie	Torgersen	39.1	18.7	181	3750	male
Adelie	Torgersen	39.5	17.4	186	3800	female
Adelie	Torgersen	40.3	18	195	3250	female

1–3 of 344 rows Previous **1** 2 3 4 5 ... 115 Next

Page jump

```
reactable(penguins,
          defaultPageSize = 3,
          paginationType = "jump")
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
Adelie	Torgersen	39.1	18.7	181	3750	male
Adelie	Torgersen	39.5	17.4	186	3800	female
Adelie	Torgersen	40.3	18	195	3250	female

1–3 of 344 rows Previous 1 of 115 Next

Grouping

```
reactable(penguins, groupBy = c("species", "island"))
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
▶ Adelie (3)						
▶ Gentoo (1)						
▶ Chinstrap (1)						

Aggregate

```
penguins %>%
  reactable(
    groupBy = c("species", "island"),
    columns = list(
      bill_length_mm = colDef(aggregate = "mean",
                              format = colFormat(digits = 2))
    )
  )
```

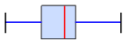
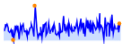
species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
▶ Adelie (3)		38.79				
▶ Gentoo (1)		47.50				
▶ Chinstrap (1)		48.83				

Sparklines

```
library(sparkline)
table_data <- penguins %>%
  group_by(species) %>%
  summarize(bill_length = list(bill_length_mm)) %>%
  mutate(boxplot = NA,
         sparkline = NA)
table_data
```

```
## # A tibble: 3 × 4
##   species   bill_length boxplot sparkline
##   <fct>     <list>      <lgl>   <lgl>
## 1 Adelie    <dbl [152]> NA      NA
## 2 Chinstrap <dbl [68]>  NA      NA
## 3 Gentoo    <dbl [124]> NA      NA
```

```
table_data %>%
  reactable(
    columns = list(
      bill_length = colDef(cell = function(value) {
        sparkline(value, type = "bar")
      }),
      boxplot = colDef(cell = function(value, index) {
        sparkline(table_data$bill_length[[index]], type = "box")
      }),
      sparkline = colDef(cell = function(value, index) {
        sparkline(table_data$bill_length[[index]])
      })
    )
  )
```

species	bill_length	boxplot	sparkline
Adelie			
Chinstrap			
Gentoo			

Lots more!

Idea of today is not to teach you everything, but to give you an idea of what's possible. Check out the documentation for more information. Also check out {reactablefmtr} for easier use and amazing extensions!

Geographic data

First - a disclaimer

- We're *only* talking about visualizing geographic data, not analyzing geographic data
- Even so, there's SO MUCH we won't get to
- Today is an intro - lots more you can do, hopefully you'll feel comfortable with the basics

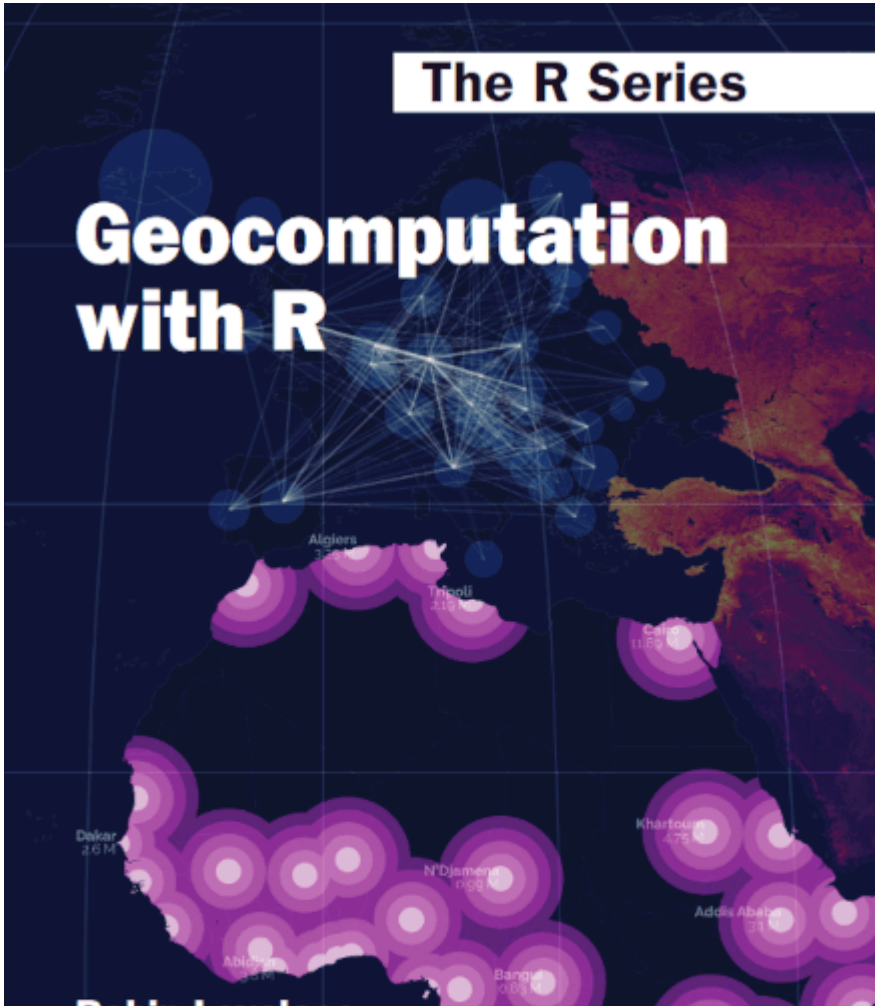
Learning objectives

- Know the difference between vector and raster data
- Be able to produce basic maps
- Be able to obtain different types of geographic data from a few different places
- Be able to produce basic interactive maps
- Understand the basics of the R geospatial ecosystem

Today is partially about content and partially about exposure

Where to learn more

Geocomputation with R



Zev Ross 2-day Workshop

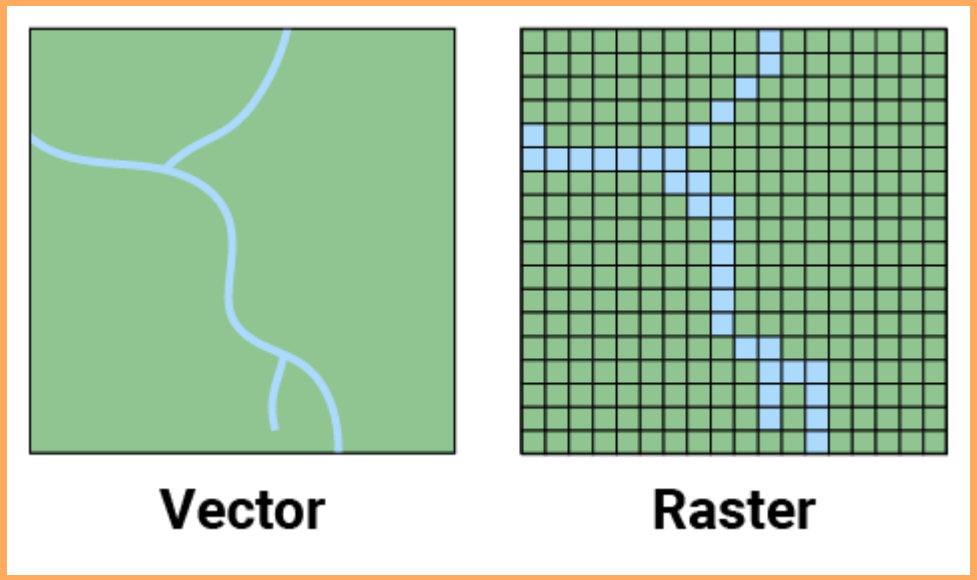
From rstudio::conf(2020)

Modern Geospatial Data Analysis with R

A workshop by Zev Ross, ZevRoss Spatial Analysis, delivered at the RStudio conference 2020

To have Zev deliver this training at your institution or learn more about training provided by ZevRoss Spatial Analysis visit our training page.

Introduction (section 1)



Vector versus

Image from Zev Ross

Vector data

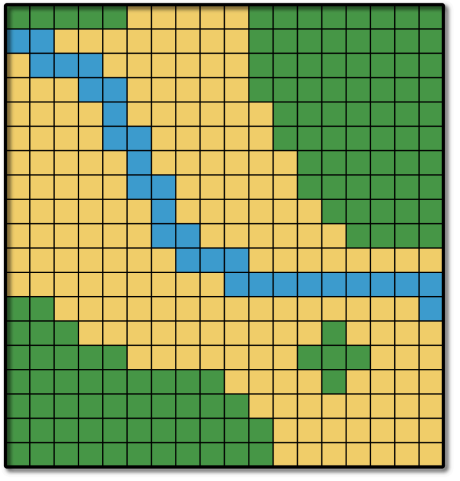
- points, lines, and polygons
- Can easily include non-spatial data (e.g., number of people living within the polygon)
- Come in the form of shapefiles (`.shp`), GeoJSON, or frequently in R packages.

This is what we'll talk about almost exclusively today

Tends to be the most relevant for social science research questions

Raster data

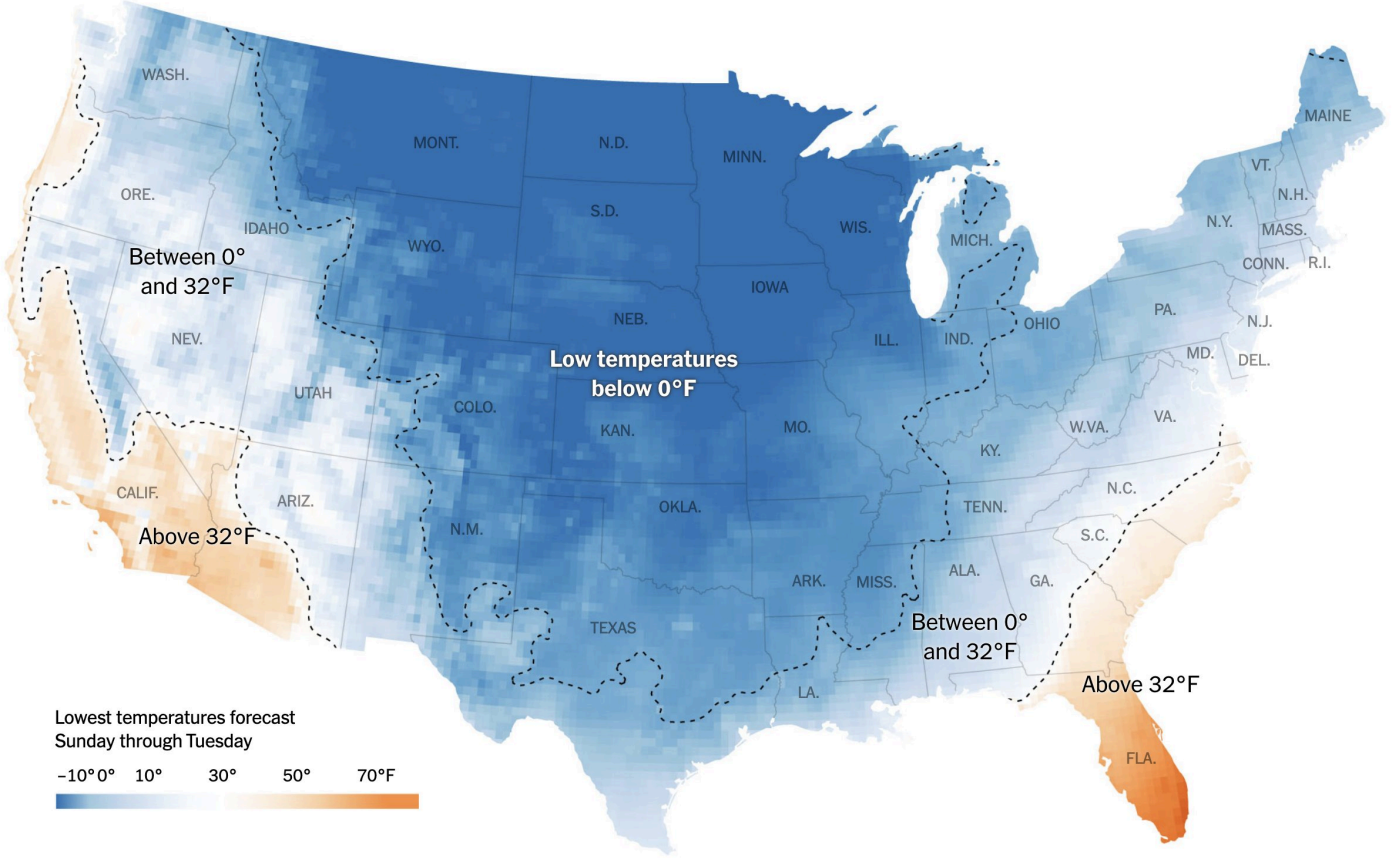
- Divide the space into a grid
- Assign each square (pixel) a value



Common formats include images and are often used in satellite and remote sensing data.

Can occasionally be helpful in social science data to show things like population density.

Example



source

Some of the #rspatial ecosystem

- {sf}
- {raster}
- {ggplot2}
- {tmap}
- {mapview}

My goal

Take you through at least a basic tour of each of these (minus {raster}, although we'll discuss raster data).

Some specific challenges with geospatial data

- Coordinate reference systems and projections (we won't have much time for this)
- List columns (specifically when working wtih {sf} objects)
- Different geometry types (lines, points, polygons)
- Vector versus raster
- Data regularly stored in data "cubes" or "bricks" to represent, e.g., longitude, latitude, and elevation, or time series, or different colors

Getting spatial data

- We'll only cover a few ways to do this
- Purposefully United States centric
- Generally reading shape files is not terrifically difficult. Reading in and manipulating raster data can be tricky at times.
- Lots of organizations out there that publish spatial data, and a fair amount are available through R packages

Working with spatial data

Two basic options

- spatial*DataFrame (from the {sp} package)
- sf data frame (simple features)
    - We'll mostly talk about this

I can show you spatial*DataFrame outside the slides (it hung things up here). Generally, I'd stick with {sf}.

Use `sf::st_as_sf` to convert `{sp}` to `{sf}`

{tigris}

```
library(tigris)
library(sf)
options(tigris_class = "sf")

roads_laneco <- roads("OR", "Lane")
roads_laneco
```

```
## Simple feature collection with 20382 features and 4 fields
## Geometry type: LINESTRING
## Dimension:     XY
## Bounding box:  xmin: -124.1536 ymin: 43.4376 xmax: -121.8131 ymax: 44.29001
## Geodetic CRS:  NAD83
## # A tibble: 20,382 × 5
##    LINEARID      FULLNAME           RTTYP MTFCC                    geometry
##    <chr>         <chr>              <chr> <chr>              <LINESTRING [°]>
##  1 1102152610459 W Lone Oak Lp      M     S1640 (-123.1256 44.10108, -123.1…
##  2 110458664549  Village Plz Lp     M     S1400 (-123.1053 44.08658, -123.1…
##  3 1102217699746 Sheldon Village Lp M     S1640 (-123.0723 44.07875, -123.0…
##  4 110458663505  Cottage Hts Lp     M     S1400 (-123.0522 43.7893, -123.05…
##  5 1102217699747 Sheldon Village Lp M     S1640 (-123.0742 44.07891, -123.0…
##  6 110458661289  River Pointe Lp    M     S1400 (-123.0864 44.10306, -123.0…
##  7 1102152615811 Village Plz Lp     M     S1640 (-123.1051 44.08716, -123.1…
##  8 1102223141058 Carpenter Byp      M     S1400 (-123.3368 43.78013, -123.3…
##  9 1104486303973 State Hwy 126 Bus  S     S1200 (-123.0319 44.04427, -123.0…
```

I/O

Let's say I want to write the file to disk.

```
# from the sf library
write_sf(roads_laneco, here::here("data", "roads_lane.shp"))
```

Then read it in later

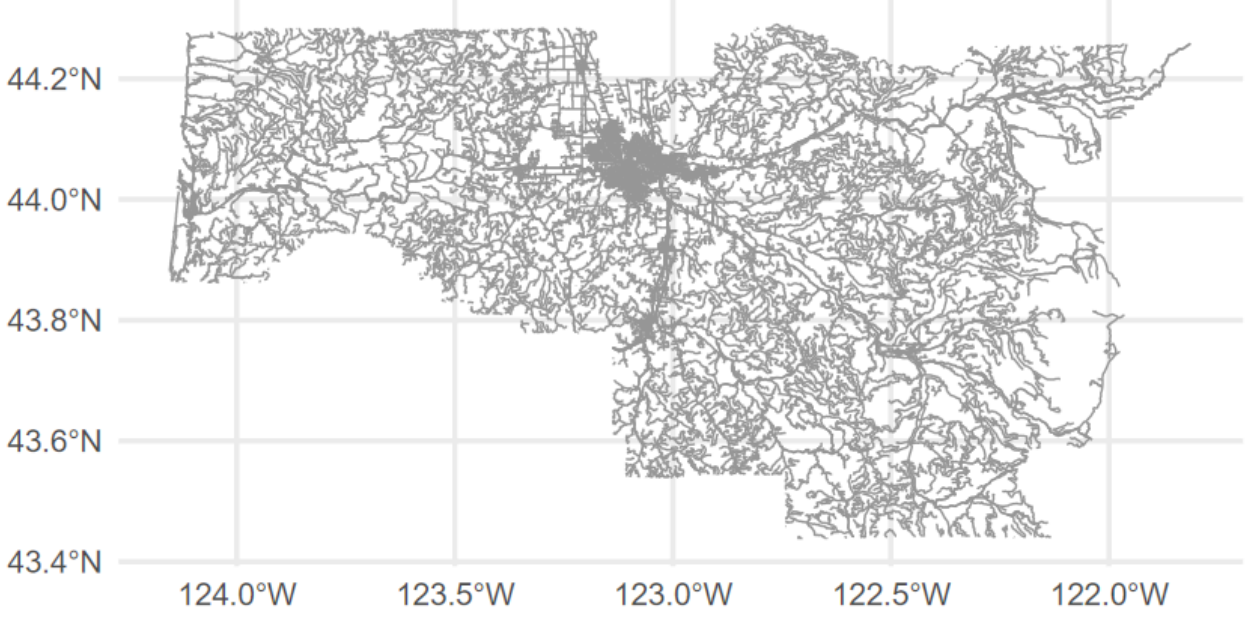
```
roads_laneco <- read_sf(here::here("data", "roads_lane.shp"))
roads_laneco
```

```
## Simple feature collection with 20382 features and 4 fields
## Geometry type: LINESTRING
## Dimension:     XY
## Bounding box:  xmin: -124.1536 ymin: 43.4376 xmax: -121.8131 ymax: 44.29001
## Geodetic CRS:  NAD83
## # A tibble: 20,382 × 5
##    LINEARID      FULLNAME           RTTYP MTFCC                    geometry
##    <chr>         <chr>              <chr> <chr>              <LINESTRING [°]>
##  1 1102152610459 W Lone Oak Lp      M     S1640 (-123.1256 44.10108, -123.1…
##  2 110458664549  Village Plz Lp     M     S1400 (-123.1053 44.08658, -123.1…
##  3 1102217699746 Sheldon Village Lp M     S1640 (-123.0723 44.07875, -123.0…
##  4 110458663505  Cottage Hts Lp     M     S1400 (-123.0522 43.7893, -123.05…
```

{sf} works with ggplot

Use `ggplot2::geom_sf`

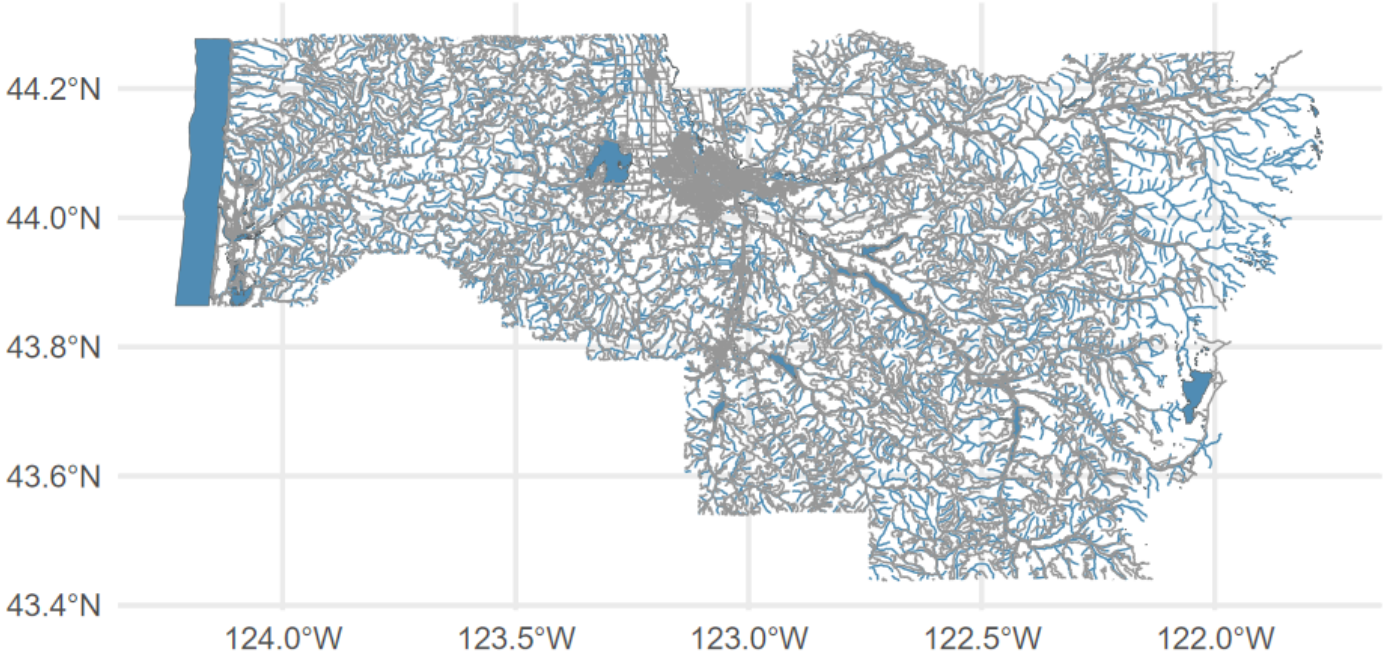
```
ggplot(roads_laneco) +
  geom_sf(color = "gray60")
```



Add water features

```
lakes <- area_water("OR", "Lane")
streams <- linear_water("OR", "Lane")

ggplot() +
  geom_sf(data = lakes, fill = "#518FB5") + # Add lakes
  geom_sf(data = streams, color = "#518FB5") + # Add streams/dra
  geom_sf(data = roads_laneco, color = "gray60") # add roads
```

Note - these functions are all from the {tigris} package.


44.2°N
44.0°N
43.8°N
43.6°N
43.4°N
124.0°W
123.5°W
123.0°W
122.5°W
122.0°W

Quick aside

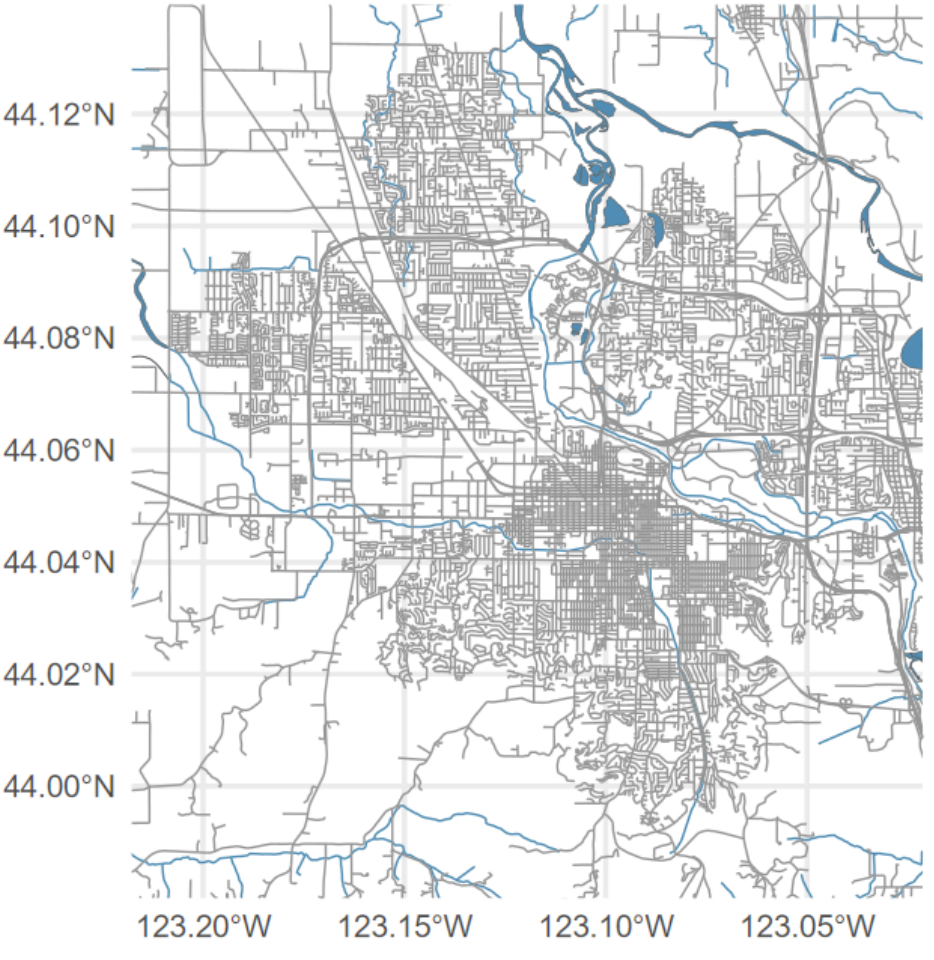
Similar package `osmdata`

- Specifically for street-level data.
- We'll just use the boundry box functionality, but you can add many of the same things (and there are other packages that will provide you with boundary boxes)

```
bb <- osmdata::getbb("Eugene")
bb
```

```
##          min        max
## x -123.20876 -123.03059
## y   43.98753   44.13227
```

```
ggplot() +
  geom_sf(data = lakes, fill = "#518FB5") + # Add lakes
  geom_sf(data = streams, color = "#518FB5", size = 1.2) + # Add
  geom_sf(data = roads_laneco, color = "gray60") + # add roads
  coord_sf(xlim = bb[1, ], ylim = bb[2, ]) # limit range
```


44.12°N
44.10°N
44.08°N
44.06°N
44.04°N
44.02°N
44.00°N
123.20°W
123.15°W
123.10°W
123.05°W

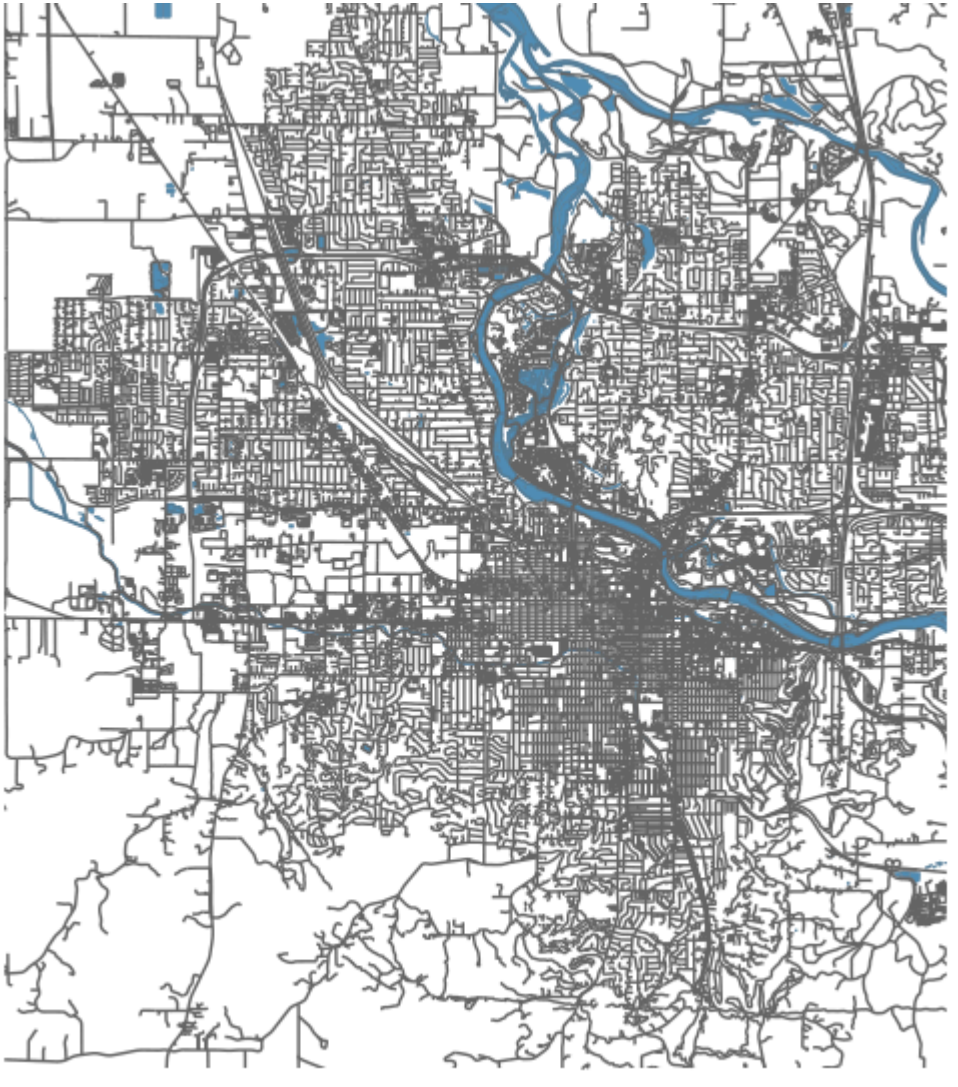
Quickly

Same thing but fully `osmdata`

```
library(osmdata)
library(colorspace)

bb <- getbb("Eugene")

roads <- bb %>%
  opq() %>% #overpass query
  add_osm_feature("highway") %>% # feature to add
  osmdata_sf() # Change it to sf

water <- bb %>%
  opq() %>%
  add_osm_feature("water") %>%
  osmdata_sf()
```

Use the data to plot

```
ggplot() +
  geom_sf(data = water$osm_multipolygons,
          fill = "#518FB5",
          color = darken("#518FB5")) +
  geom_sf(data = water$osm_polygons,
          fill = "#518FB5",
          color = darken("#518FB5")) +
  geom_sf(data = water$osm_lines,
          color = darken("#518FB5")) +
  geom_sf(data = roads$osm_lines,
          color = "gray40",
          size = 0.2)  +
  coord_sf(xlim = bb[1, ],
           ylim = bb[2, ],
           expand = FALSE) +
  labs(caption = "Eugene, OR")
```



Eugene, OR

Let's get some census data

Note

To do this, you need to first register an API key with the US Census, which you can do here. Then use `census_api_key("YOUR API KEY")`.

Alternatively, you can specify `CENSUS_API_KEY = "YOUR API KEY"` in **.Renviron**. You can do this by using `usethis::edit_r_environ()`

Getting the data

```
library(tidycensus)
# Find variable code
# v <- load_variables(2019, "acs5")
# View(v)

census_vals <- get_acs(
  geography = "tract",
  state = "OR",
  variables = c(med_income = "B06011_001",
                ed_attain = "B15003_001"),
  year = 2019,
  geometry = TRUE
)
```

```
##
  |
  |                                                                      |   0%
  |
  |=                                                                     |   1%
  |
  |==                                                                    |   3%
  |
  |====                                                                  |   6%
  |
```

Look at the data

```
census_vals
```

```
## Simple feature collection with 1668 features and 5 fields (with 12 geometries e
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -124.5662 ymin: 41.99179 xmax: -116.4635 ymax: 46.29204
## Geodetic CRS:  NAD83
## First 10 features:
##          GEOID                                                NAME   variable
## 1  41031960302 Census Tract 9603.02, Jefferson County, Oregon med_income
## 2  41031960302 Census Tract 9603.02, Jefferson County, Oregon  ed_attain
## 3  41057960100    Census Tract 9601, Tillamook County, Oregon med_income
## 4  41057960100    Census Tract 9601, Tillamook County, Oregon  ed_attain
## 5  41015950100        Census Tract 9501, Curry County, Oregon med_income
## 6  41015950100        Census Tract 9501, Curry County, Oregon  ed_attain
## 7  41039001902        Census Tract 19.02, Lane County, Oregon med_income
## 8  41039001902        Census Tract 19.02, Lane County, Oregon  ed_attain
## 9  41029000300         Census Tract 3, Jackson County, Oregon med_income
## 10 41029000300         Census Tract 3, Jackson County, Oregon  ed_attain
##    estimate  moe                       geometry
## 1     30061 2953 MULTIPOLYGON (((-121.8495 4...
## 2      3011  228 MULTIPOLYGON (((-121.8495 4...
## 3     30243 3888 MULTIPOLYGON (((-123.983 45...
## 4      2707  245 MULTIPOLYGON (((-123.983 45...
```

Remove missing geometry rows

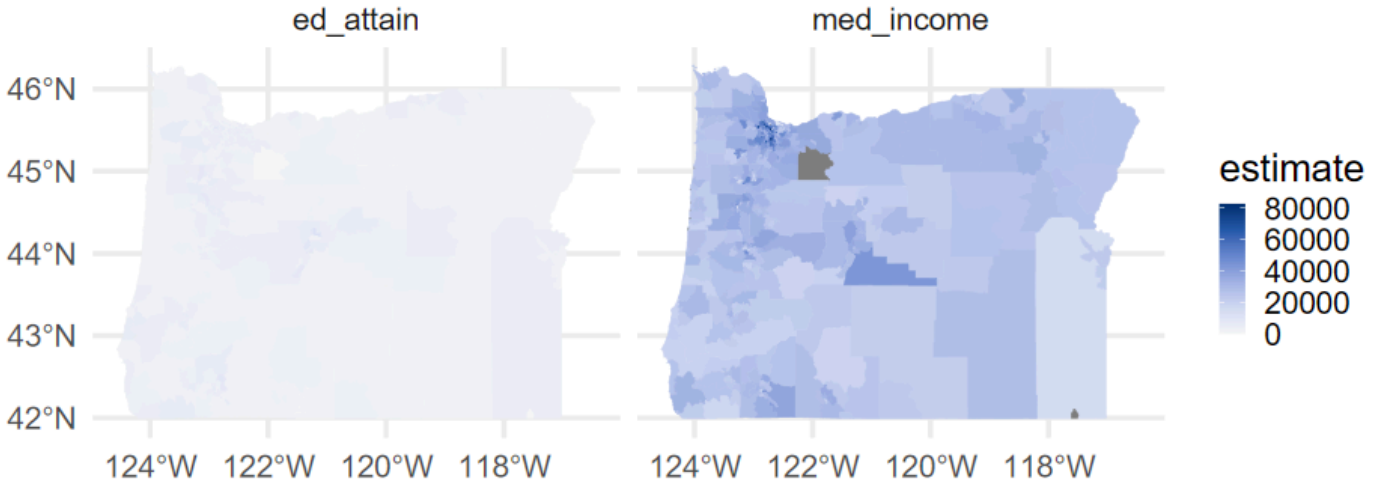
- Tidycensus is (currently) bringing in some rows with missing geometries
- This is not a big deal for ggplot, but is for other plotting systems
- Let's remove those rows

```
census_vals <- census_vals[!st_is_empty(census_vals$geometry), ,
```

Plot it

```
library(colorspace)
ggplot(census_vals) +
  geom_sf(aes(fill = estimate, color = estimate)) +
  facet_wrap(~variable) +
  guides(color = "none") +
  scale_fill_continuous_diverging("Blue-Red 3", rev = TRUE) +
  scale_color_continuous_diverging("Blue-Red 3", rev = TRUE)
```

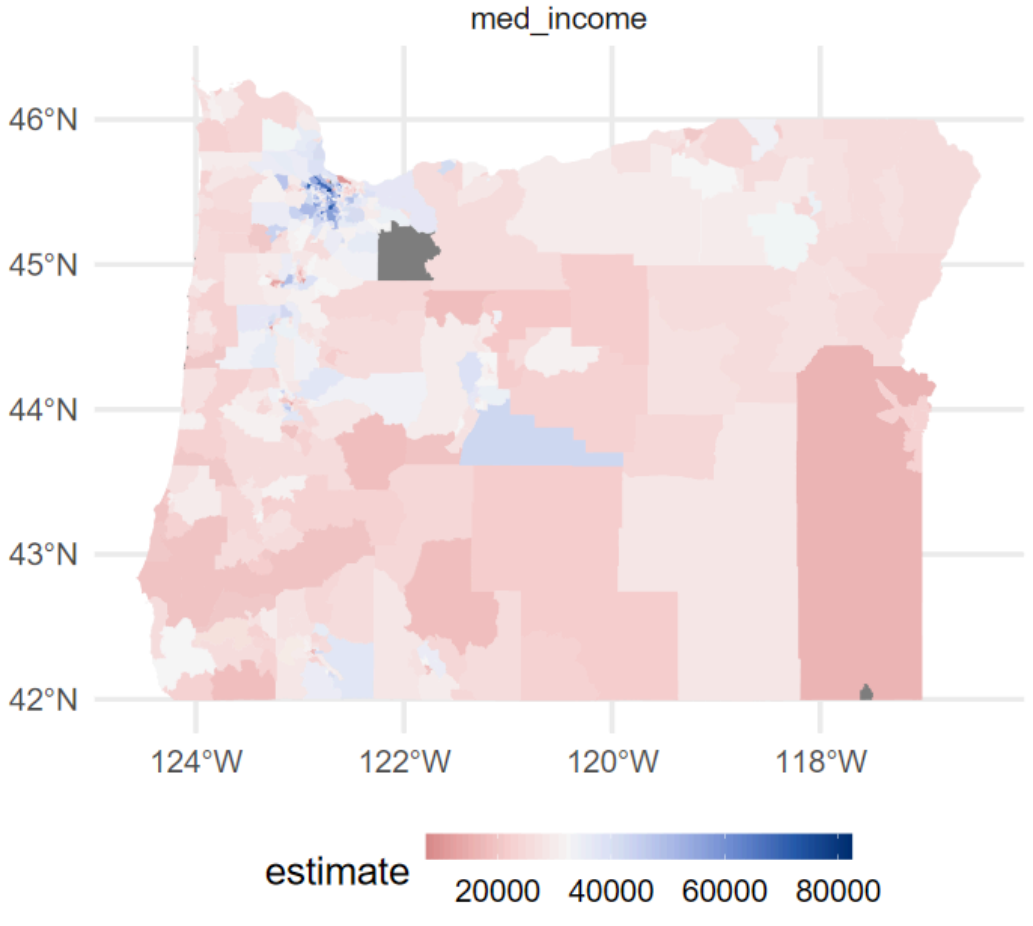
hmm...


ed_attain
med_income
46°N
45°N
44°N
43°N
42°N
124°W
122°W
120°W
118°W
estimate
80000
60000
40000
20000
0

Try again

```
library(colorspace)
income <- filter(census_vals, variable == "med_income")

income_plot <- ggplot(income) +
  geom_sf(aes(fill = estimate, color = estimate)) +
  facet_wrap(~variable) +
  guides(color = "none") +
  scale_fill_continuous_diverging(
    "Blue-Red 3",
    rev = TRUE,
    mid = mean(income$estimate, na.rm = TRUE)
  ) +
  scale_color_continuous_diverging(
    "Blue-Red 3",
    rev = TRUE,
    mid = mean(income$estimate, na.rm = TRUE)
  ) +
  theme(legend.position = "bottom",
        legend.key.width = unit(2, "cm"))
```

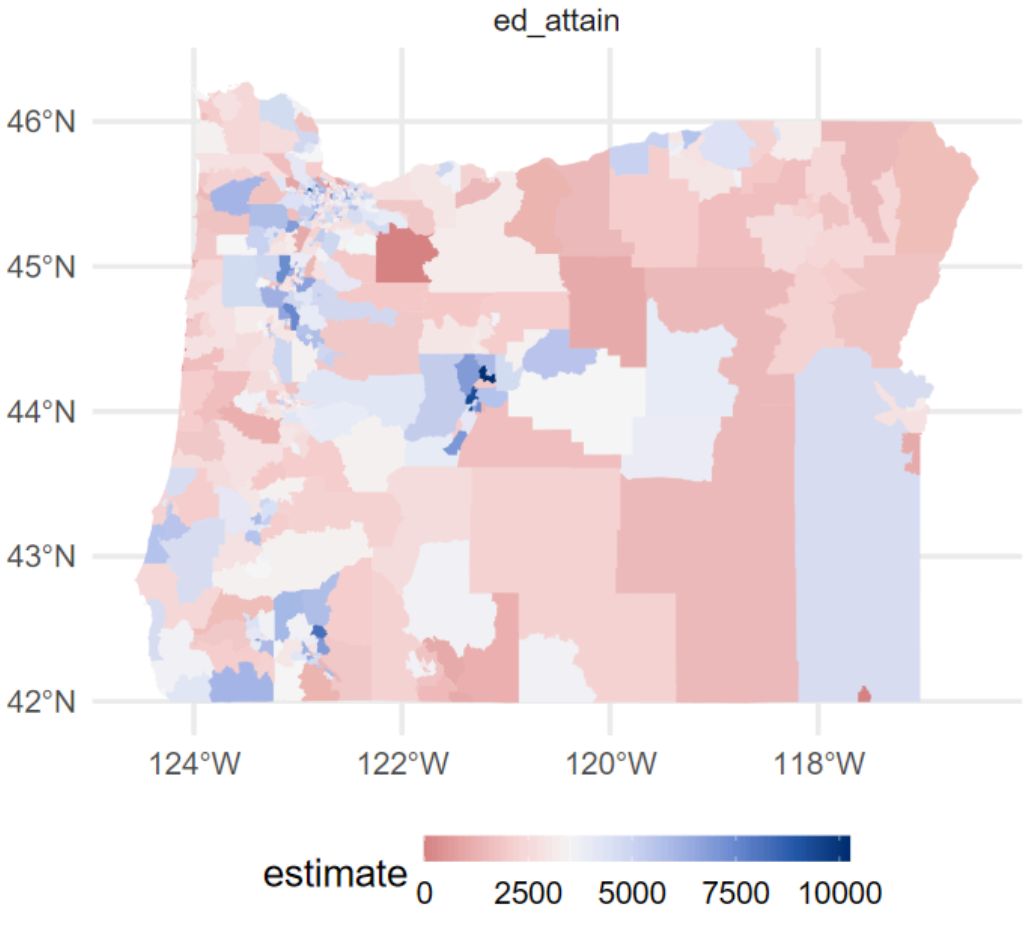
```
income_plot
```


med_income
46°N
45°N
44°N
43°N
42°N
124°W
122°W
120°W
118°W
estimate
20000
40000
60000
80000

Same thing for education

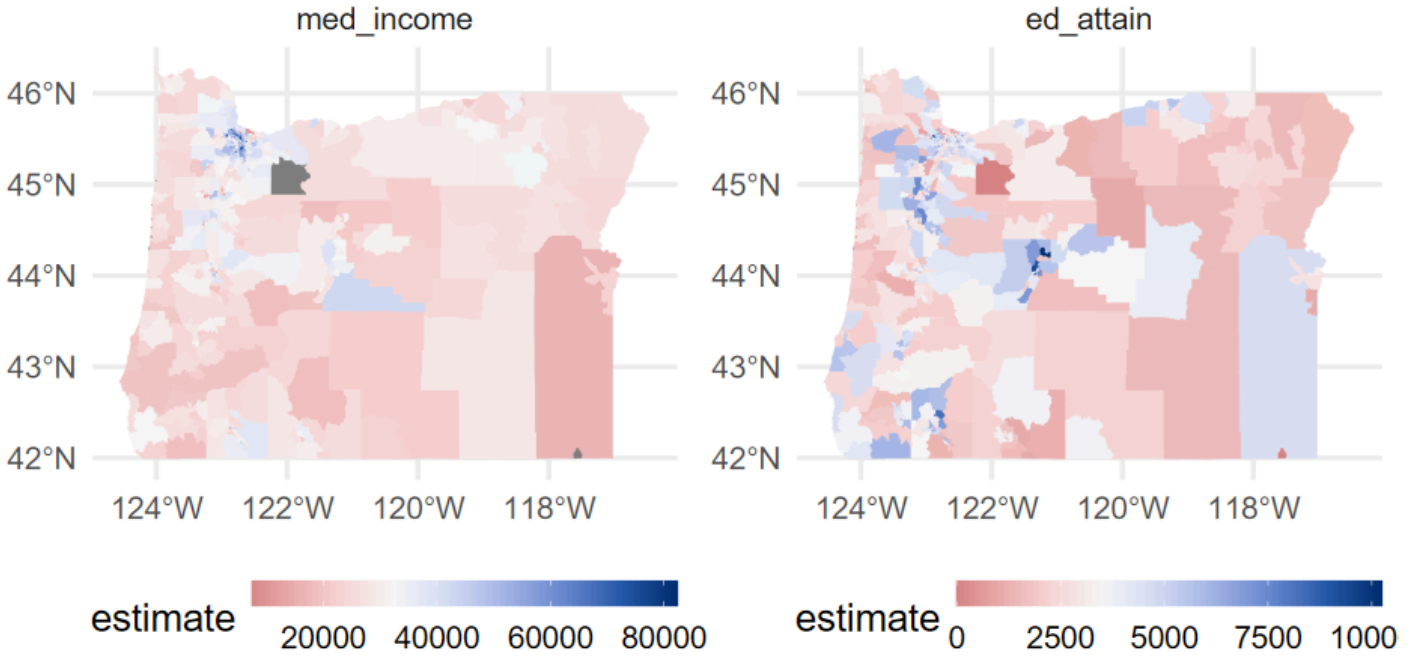
```
ed <- filter(census_vals, variable == "ed_attain")

ed_plot <- ggplot(ed) +
  geom_sf(aes(fill = estimate, color = estimate)) +
  facet_wrap(~variable) +
  guides(color = "none") +
  scale_fill_continuous_diverging(
    "Blue-Red 3",
    rev = TRUE,
    mid = mean(ed$estimate, na.rm = TRUE)
  ) +
  scale_color_continuous_diverging(
    "Blue-Red 3",
    rev = TRUE,
    mid = mean(ed$estimate, na.rm = TRUE)
  ) +
  theme(legend.position = "bottom",
        legend.key.width = unit(2, "cm"))
```

```
ed_plot
```


ed_attain
46°N
45°N
44°N
43°N
42°N
124°W
122°W
120°W
118°W
estimate
0
2500
5000
7500
10000

Put them together

```
gridExtra::grid.arrange(income_plot, ed_plot, ncol = 2)
```

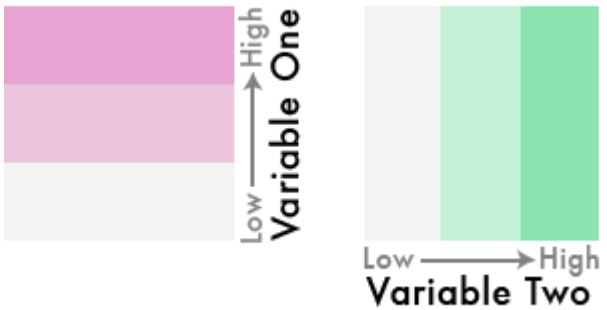


Bivariate color scales

How?

There are a few different ways. Here's one:

- Break continuous variable into categorical values
- Assign each combination of values between categorical vars a color
- Make sure the combinations of the colors make sense



gif from Joshua Stevens

Do it

Note - this will be fairly quick. I'm not expecting you to know how to do this, but I want to show you the idea and give you the breadcrumbs for the code you may need.

First - move it to wider

```
wider <- get_acs(
  geography = "tract",
  state = "OR",
  variables = c(med_income = "B06011_001",
                ed_attain = "B15003_001"),
  year = 2019,
  geometry = TRUE,
  output = "wide"
)

# remove missing geometry rows
wider <- wider[!st_is_empty(wider$geometry), , drop = FALSE]
```

Find the quartiles

```
ed_quartiles <- quantile(
  wider$ed_attainE,
  probs = seq(0, 1, length.out = 4),
  na.rm = TRUE
)

inc_quartiles <- quantile(
  wider$med_incomeE,
  probs = seq(0, 1, length.out = 4),
  na.rm = TRUE
)

ed_quartiles
```

```
##        0% 33.33333% 66.66667%      100%
##     0.000  2715.000  3949.333 10245.000
```

```
inc_quartiles
```

```
##        0% 33.33333% 66.66667%      100%
##   7449.00  26718.33  34375.00  82375.00
```

Create the cut variable

```
wider <- wider %>%
  mutate(cat_ed = cut(ed_attainE, ed_quartiles),
         cat_inc = cut(med_incomeE, inc_quartiles))
wider %>%
  select(starts_with("cat"))
```

```
## Simple feature collection with 828 features and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -124.5662 ymin: 41.99179 xmax: -116.4635 ymax: 46.29204
## Geodetic CRS:  NAD83
## First 10 features:
##                  cat_ed             cat_inc                       geometry
## 1  (2.71e+03,3.95e+03] (2.67e+04,3.44e+04] MULTIPOLYGON (((-121.8495 4...
## 2         (0,2.71e+03] (2.67e+04,3.44e+04] MULTIPOLYGON (((-123.983 45...
## 3         (0,2.71e+03] (7.45e+03,2.67e+04] MULTIPOLYGON (((-124.5646 4...
## 4  (3.95e+03,1.02e+04] (7.45e+03,2.67e+04] MULTIPOLYGON (((-122.9855 4...
## 5  (3.95e+03,1.02e+04] (7.45e+03,2.67e+04] MULTIPOLYGON (((-122.9079 4...
## 6  (2.71e+03,3.95e+03] (3.44e+04,8.24e+04] MULTIPOLYGON (((-123.5016 4...
## 7         (0,2.71e+03] (7.45e+03,2.67e+04] MULTIPOLYGON (((-123.0927 4...
## 8  (2.71e+03,3.95e+03] (7.45e+03,2.67e+04] MULTIPOLYGON (((-120.8811 4...
## 9  (2.71e+03,3.95e+03] (7.45e+03,2.67e+04] MULTIPOLYGON (((-123.5093 4...
## 10 (2.71e+03,3.95e+03] (7.45e+03,2.67e+04] MULTIPOLYGON (((-123.8179 4...
```

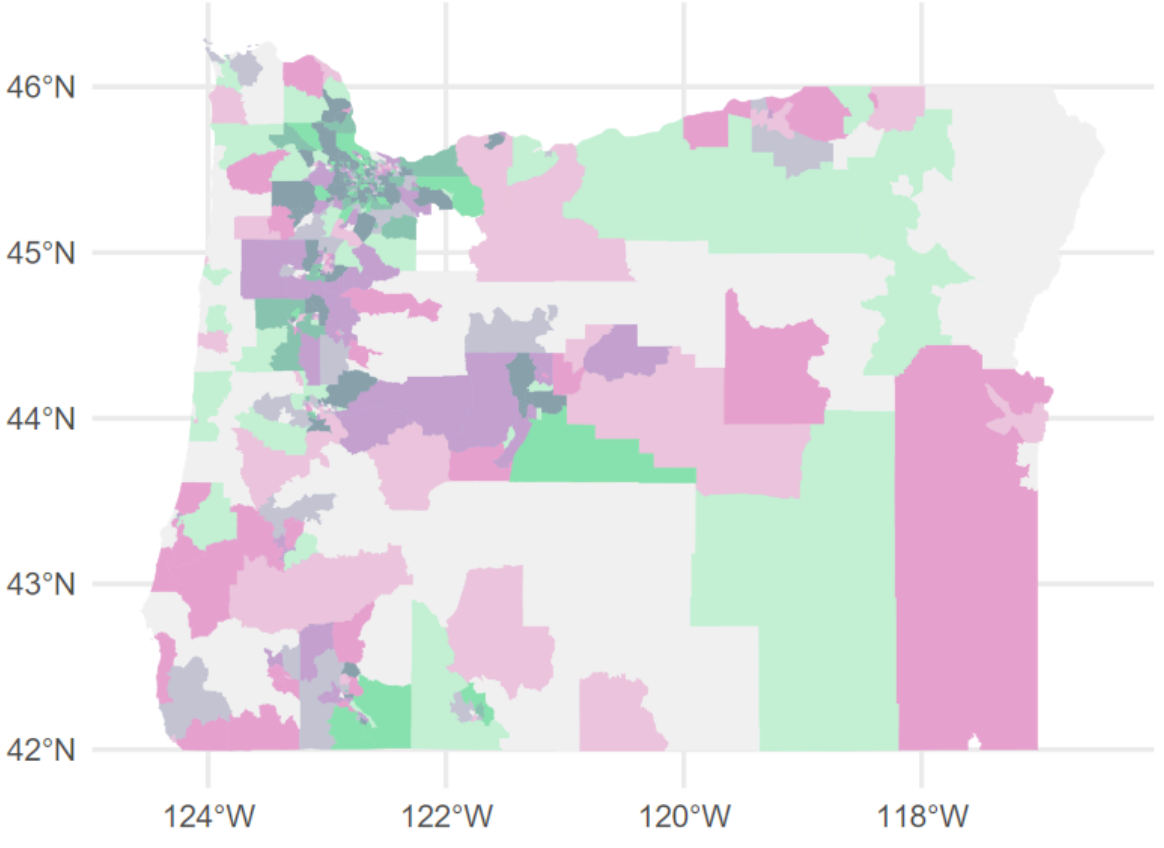
Set palette

```
# First drop geo column
pal <- st_drop_geometry(wider) %>%
  count(cat_ed, cat_inc) %>%
  arrange(cat_ed, cat_inc) %>%
  drop_na(cat_ed, cat_inc) %>%
  mutate(pal = c("#F3F3F3", "#C3F1D5", "#8BE3AF",
                 "#EBC5DD", "#C3C5D5", "#8BC5AF",
                 "#E7A3D1", "#C3A3D1", "#8BA3AE"))
pal
```

```
##                cat_ed             cat_inc   n     pal
## 1        (0,2.71e+03] (7.45e+03,2.67e+04] 116 #F3F3F3
## 2        (0,2.71e+03] (2.67e+04,3.44e+04]  85 #C3F1D5
## 3        (0,2.71e+03] (3.44e+04,8.24e+04]  70 #8BE3AF
## 4 (2.71e+03,3.95e+03] (7.45e+03,2.67e+04]  87 #EBC5DD
## 5 (2.71e+03,3.95e+03] (2.67e+04,3.44e+04]  97 #C3C5D5
## 6 (2.71e+03,3.95e+03] (3.44e+04,8.24e+04]  92 #8BC5AF
## 7 (3.95e+03,1.02e+04] (7.45e+03,2.67e+04]  71 #E7A3D1
## 8 (3.95e+03,1.02e+04] (2.67e+04,3.44e+04]  92 #C3A3D1
## 9 (3.95e+03,1.02e+04] (3.44e+04,8.24e+04] 113 #8BA3AE
```

Join & plot

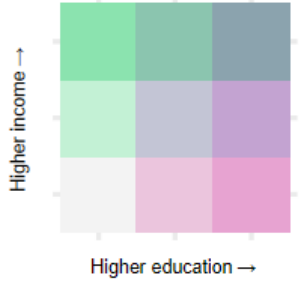
```
bivar_map <- left_join(wider, pal) %>%
  ggplot() +
  geom_sf(aes(fill = pal, color = pal)) +
  guides(fill = "none", color = "none") +
  scale_fill_identity() +
  scale_color_identity()
```


46°N
45°N
44°N
43°N
42°N
124°W
122°W
120°W
118°W

Add in legend

First create it

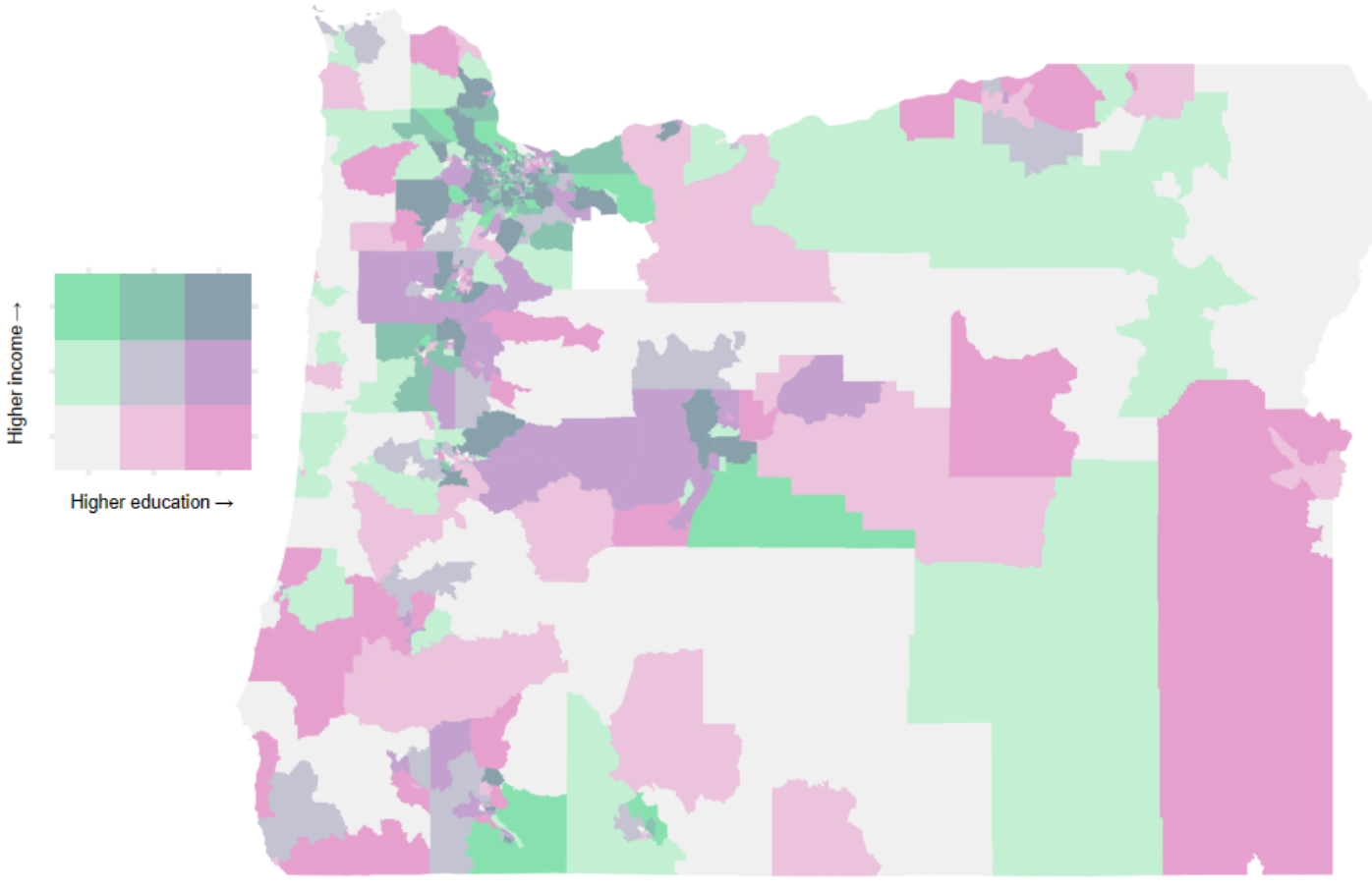
```
leg <- ggplot(pal, aes(cat_ed, cat_inc)) +
  geom_tile(aes(fill = pal)) +
  scale_fill_identity() +
  coord_fixed() +
  labs(x = expression("Higher education" %->% ""),
       y = expression("Higher income" %->% "")) +
  theme(axis.text = element_blank(),
        axis.title = element_text(size = 12))
leg
```



Put together

```
library(cowplot)
ggdraw() +
  draw_plot(bivar_map + theme_void(), 0.1, 0.1, 1, 1) +
  draw_plot(leg, -0.05, 0, 0.3, 0.3)
```

Coordinates are mostly guess/check depending on aspect ratio


Higher income →
Higher education →

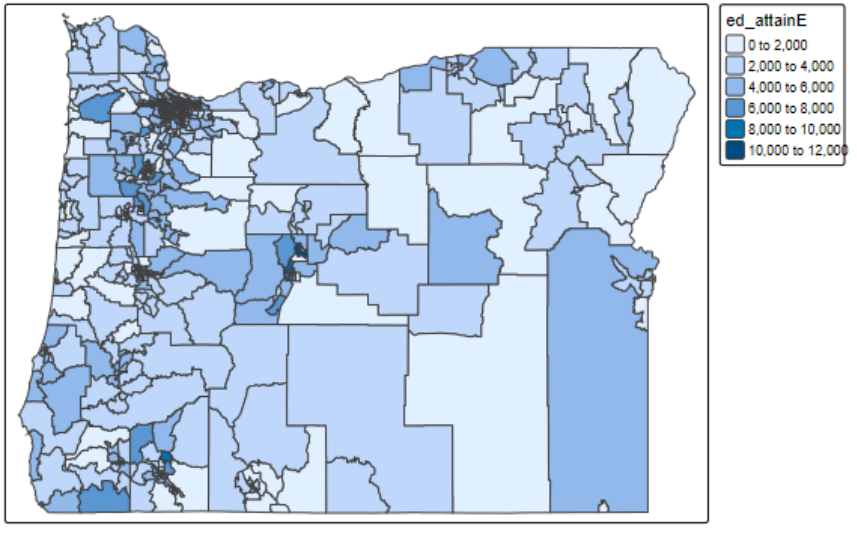
{tmap}

Back to just one variable

I mostly use `ggplot()`, but the **{tmap}** package is really powerful and the syntax is pretty straightforward, so let's have a quick overview.

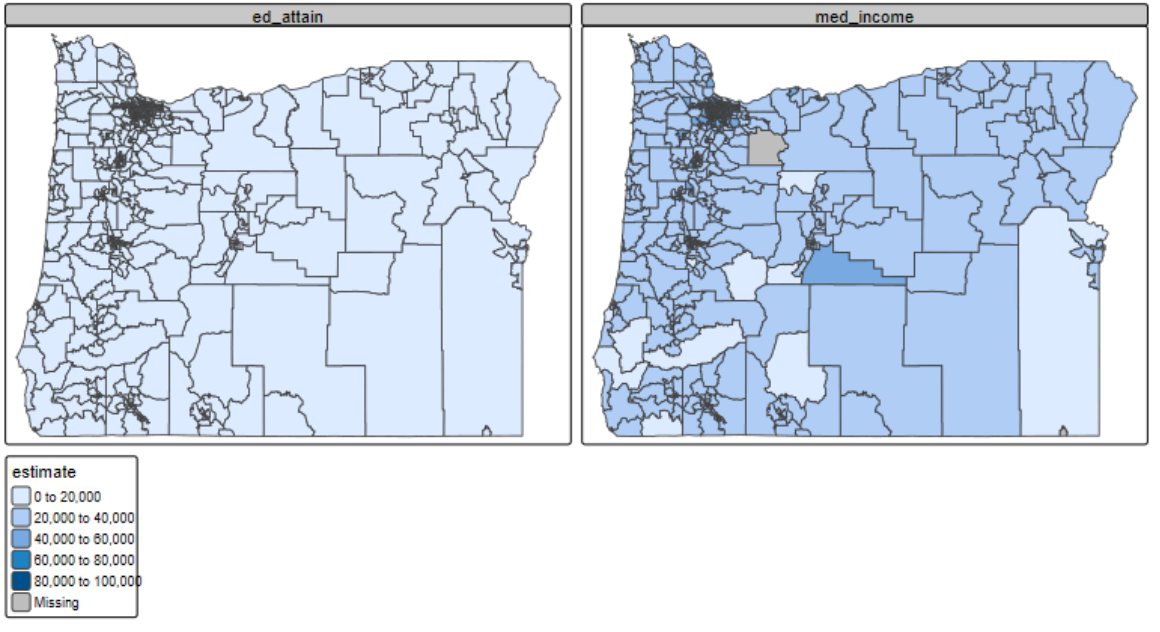
Education map with {tmap}.

```
library(tmap)
tm_shape(wider) +
  tm_polygons("ed_attainE") +
  tm_layout(legend.outside = TRUE)
```



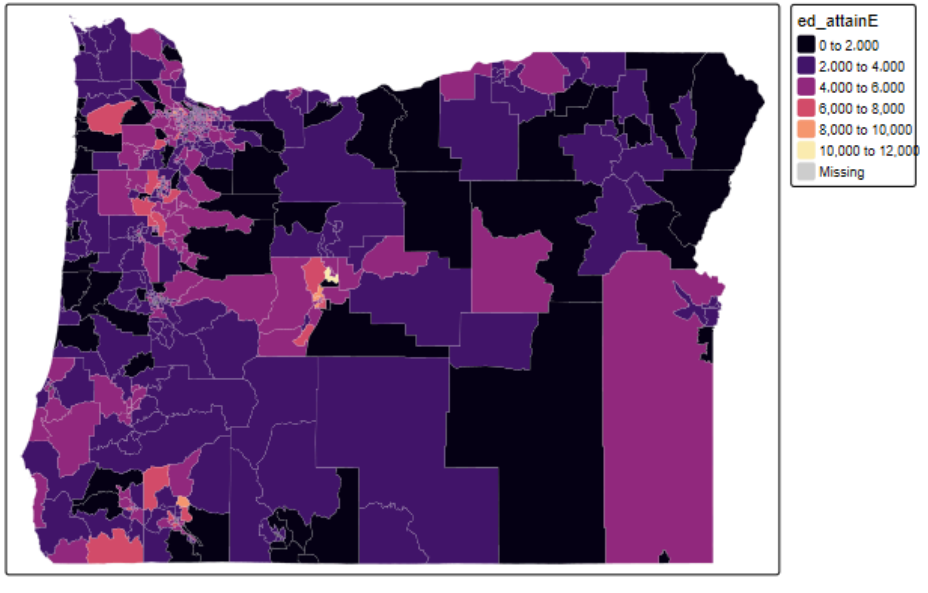
Facet

```
tm_shape(census_vals) +
  tm_polygons("estimate") +
  tm_facets("variable") +
  tm_layout(legend.outside = TRUE)
```



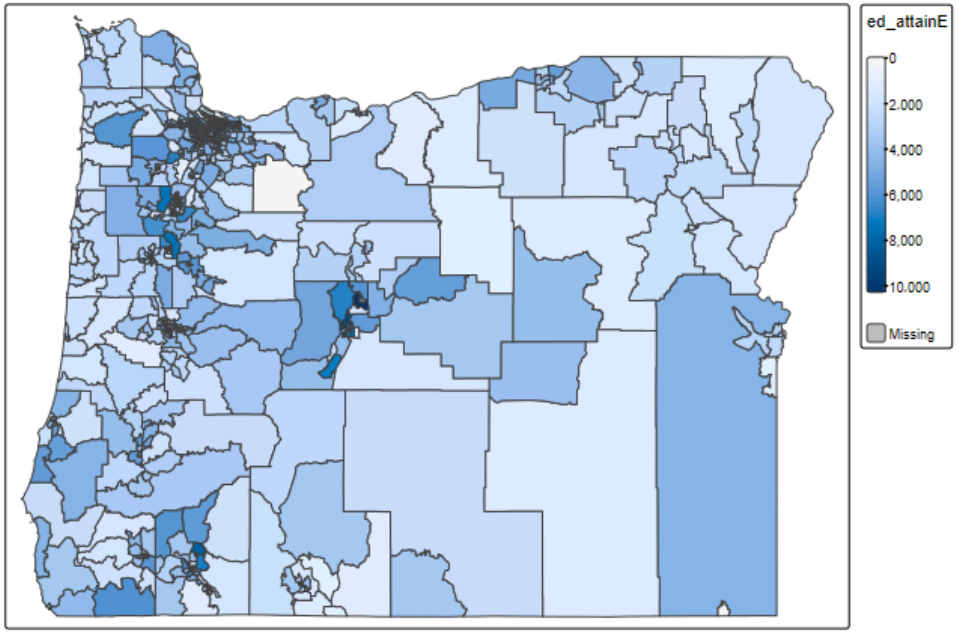
Change colors

```
tm_shape(wider) +
  tm_polygons("ed_attainE",
              palette = "magma",
              border.col = "gray90",
              lwd = 0.1) +
  tm_layout(legend.outside = TRUE)
```



Continuous legend

```
tm_shape(wider) +
  tm_polygons("ed_attainE",
              style = "cont") +
  tm_layout(legend.outside = TRUE)
```



Add text

- First, let's get data at the county level, instead of census tract level

```
cnty <- get_acs(
  geography = "county",
  state = "OR",
  variables = c(ed_attain = "B15003_001"),
  year = 2019,
  geometry = TRUE
)
```

```
##
  |
  |                                                                      |   0%
  |
  |=                                                                     |   2%
  |
  |==                                                                    |   3%
  |
  |==                                                                    |   4%
  |
  |===                                                                   |   5%
  |
```

```
cnty
```

```
## Simple feature collection with 36 features and 5 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -124.5662 ymin: 41.99179 xmax: -116.4635 ymax: 46.29204
## Geodetic CRS:  NAD83
## First 10 features:
##    GEOID                      NAME  variable estimate moe
## 1  41017 Deschutes County, Oregon ed_attain   135615 192
## 2  41003    Benton County, Oregon ed_attain    55359 143
## 3  41015     Curry County, Oregon ed_attain    18304 139
## 4  41061     Union County, Oregon ed_attain    17539  78
## 5  41055   Sherman County, Oregon ed_attain     1251  84
## 6  41051 Multnomah County, Oregon ed_attain   587290 134
## 7  41007   Clatsop County, Oregon ed_attain    28475 165
## 8  41033 Josephine County, Oregon ed_attain    63802 148
## 9  41031 Jefferson County, Oregon ed_attain    16197  24
## 10 41039      Lane County, Oregon ed_attain   256373 147
##                          geometry
## 1  MULTIPOLYGON (((-122.0019 4...
## 2  MULTIPOLYGON (((-123.8167 4...
## 3  MULTIPOLYGON (((-124.3239 4...
## 4  MULTIPOLYGON (((-118.6978 4...
## 5  MULTIPOLYGON (((-121.0312 4...
## 6  MULTIPOLYGON (((-122.9292 4...
## 7  MULTIPOLYGON (((-123.5989 4...
## 8  MULTIPOLYGON (((-124.042 42...
```

Estimate polygon centroid

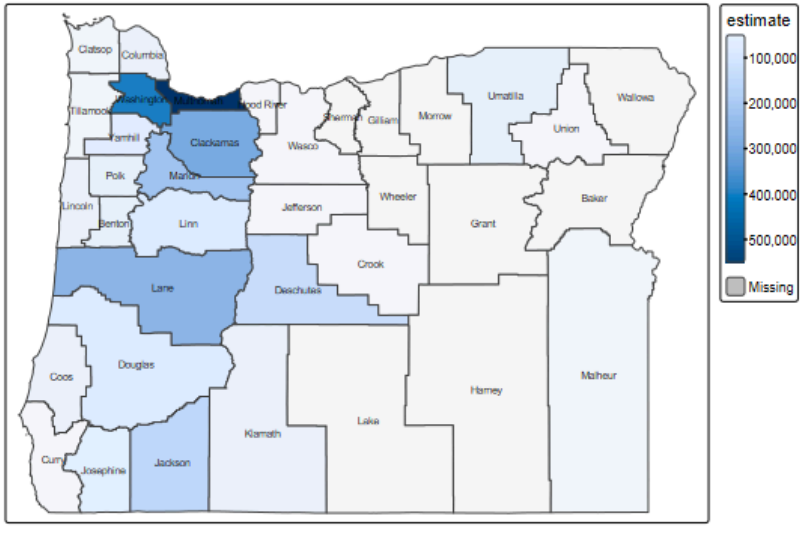
```
centroids <- st_centroid(cnty)
```

Extract just county name

```
centroids <- centroids %>%
  mutate(county = str_replace_all(NAME, " County, Oregon", ""))
```

Plot

```
tm_shape(cnty) +
  tm_polygons("estimate",
              style = "cont") +
tm_shape(centroids) +
  tm_text("county", size = 0.5) +
  tm_layout(legend.outside = TRUE)
```



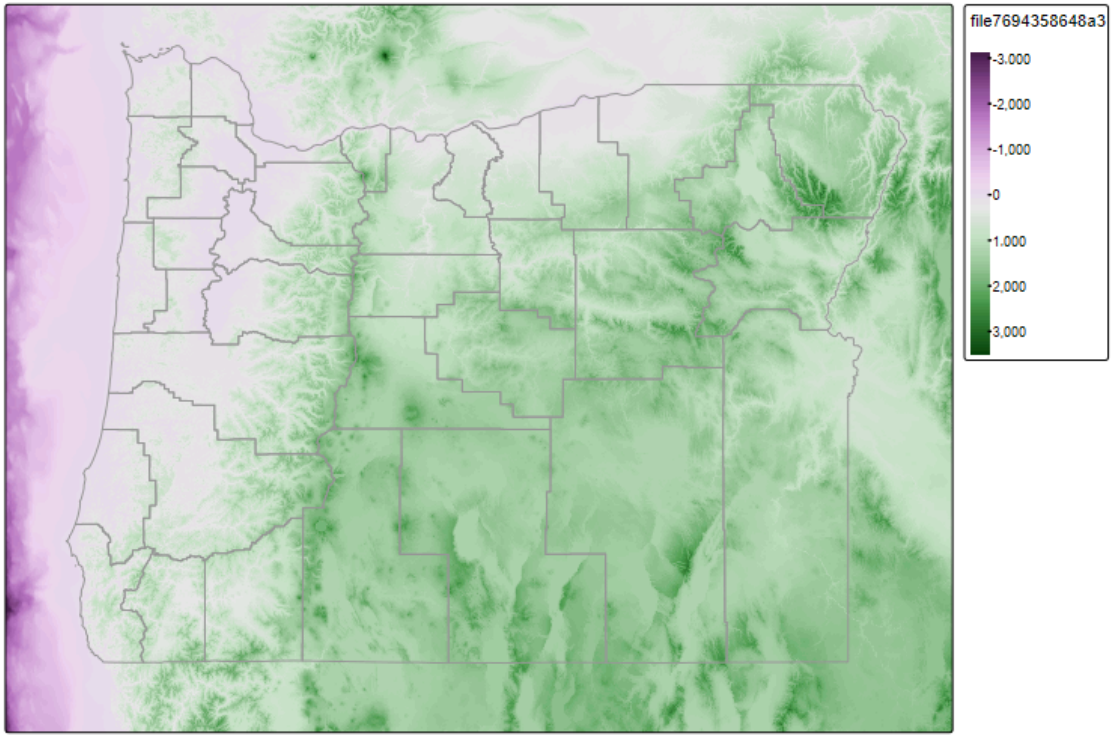
Doesn't work for me on the slides. Not sure why, but should work for you locally.

Add raster elevation data

```
states <- get_acs(
  "state",
  variables = c(ed_attain = "B15003_001"),
  year = 2019,
  geometry = TRUE
)

or <- filter(states, NAME == "Oregon")

# convert to spatial data frame
#sp <- as(or, "Spatial")

# use elevatr library to pull data
library(elevatr)
or_elev <- get_elev_raster(or, z = 9)
lane_elev <- get_elev_raster(or, z = 9)
```

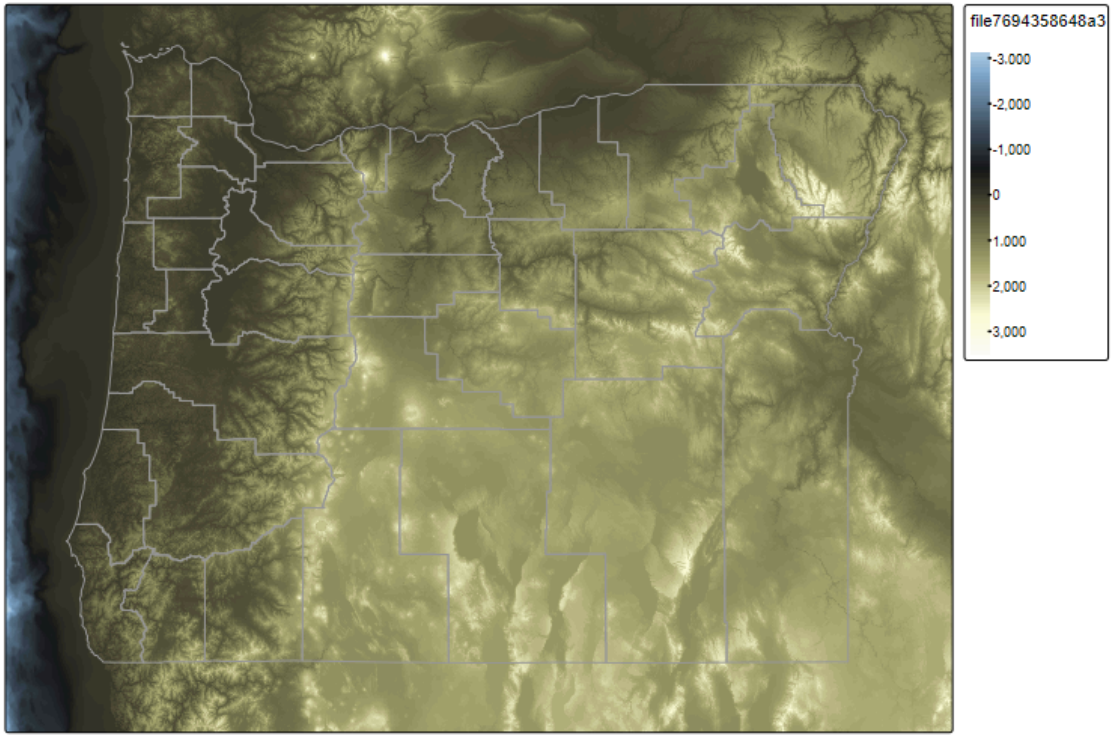
Plot

```
tm_shape(or_elev) +
  tm_raster(midpoint = NA,
            style = "cont") +
  tm_layout(legend.outside = TRUE) +
tm_shape(cnty) +
  tm_borders(col = "gray60")
```

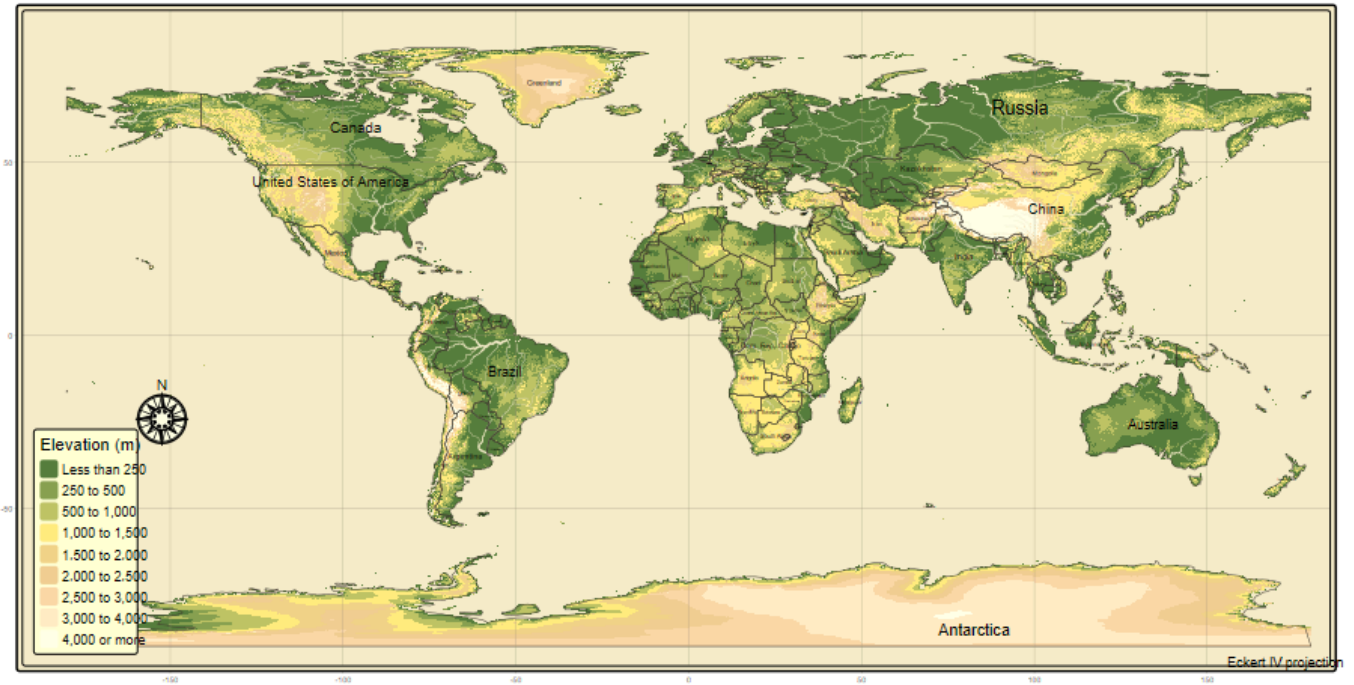


Add custom palette

```
tm_shape(or_elev) +
  tm_raster(midpoint = NA,
            style = "cont",
            palette = c("#E2FCFF", "#83A9CE", "#485C6E",
                        "#181818", "#5C5B3E", "#AAA971",
                        "#FCFCD3", "#ffffff")) +
  tm_layout(legend.outside = TRUE) +
tm_shape(cnty) +
  tm_borders(col = "gray60")
```


file7694358648a3
-3,000
-2,000
-1,000
0
1,000
2,000
3,000

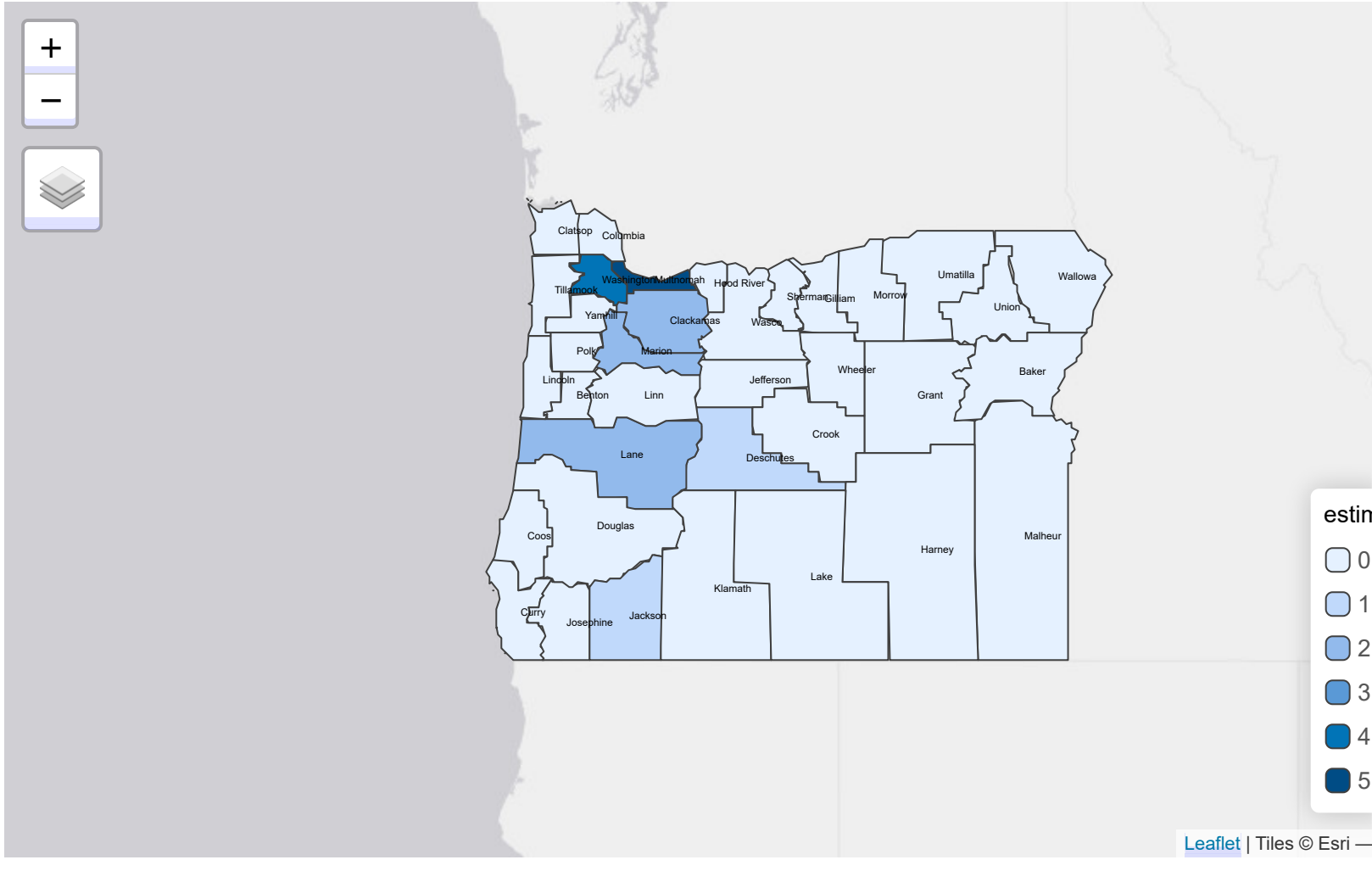
You can do some amazing things!



Create interactive maps

Just change run `tmap_mode("view)` then run the same code as before

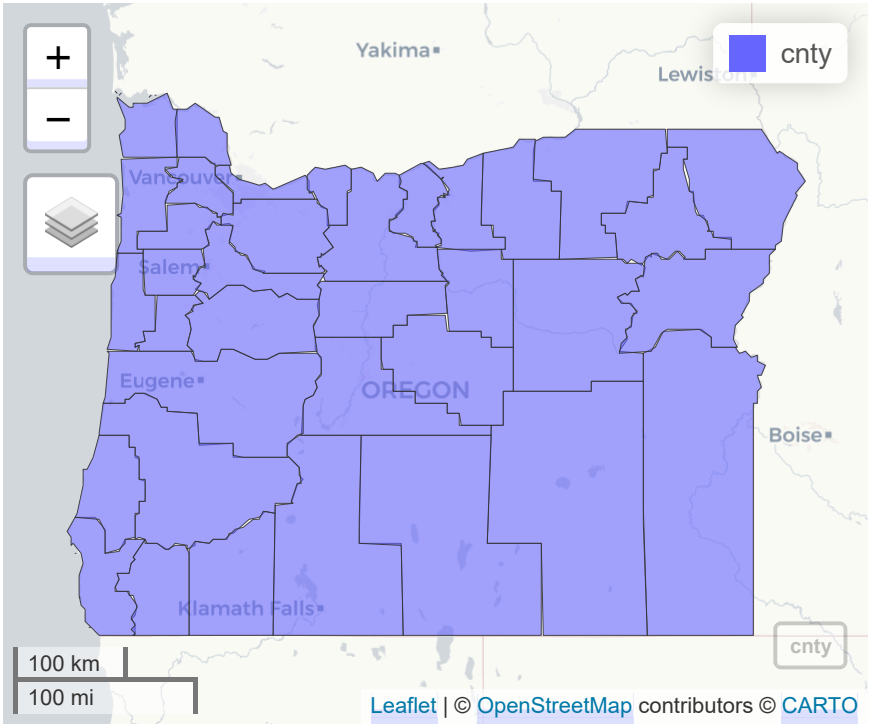
```
tmap_mode("view")

tm_shape(cnty) +
  tm_polygons("estimate") +
tm_shape(centroids) +
  tm_text("county", size = 0.5)
```


Clatsop
Columbia
Tillamook
Washington
Multnomah
Hood River
Sherman
Gilliam
Morrow
Umatilla
Wallowa
Union
Yamhill
Clackamas
Wasco
Polk
Marion
Jefferson
Wheeler
Baker
Lincoln
Benton
Linn
Grant
Crook
Lane
Deschutes
Douglas
Coos
Harney
Malheur
Lake
Klamath
Curry
Josephine
Jackson
estim
0
1
2
3
4
5
Leaflet | Tiles © Esri —

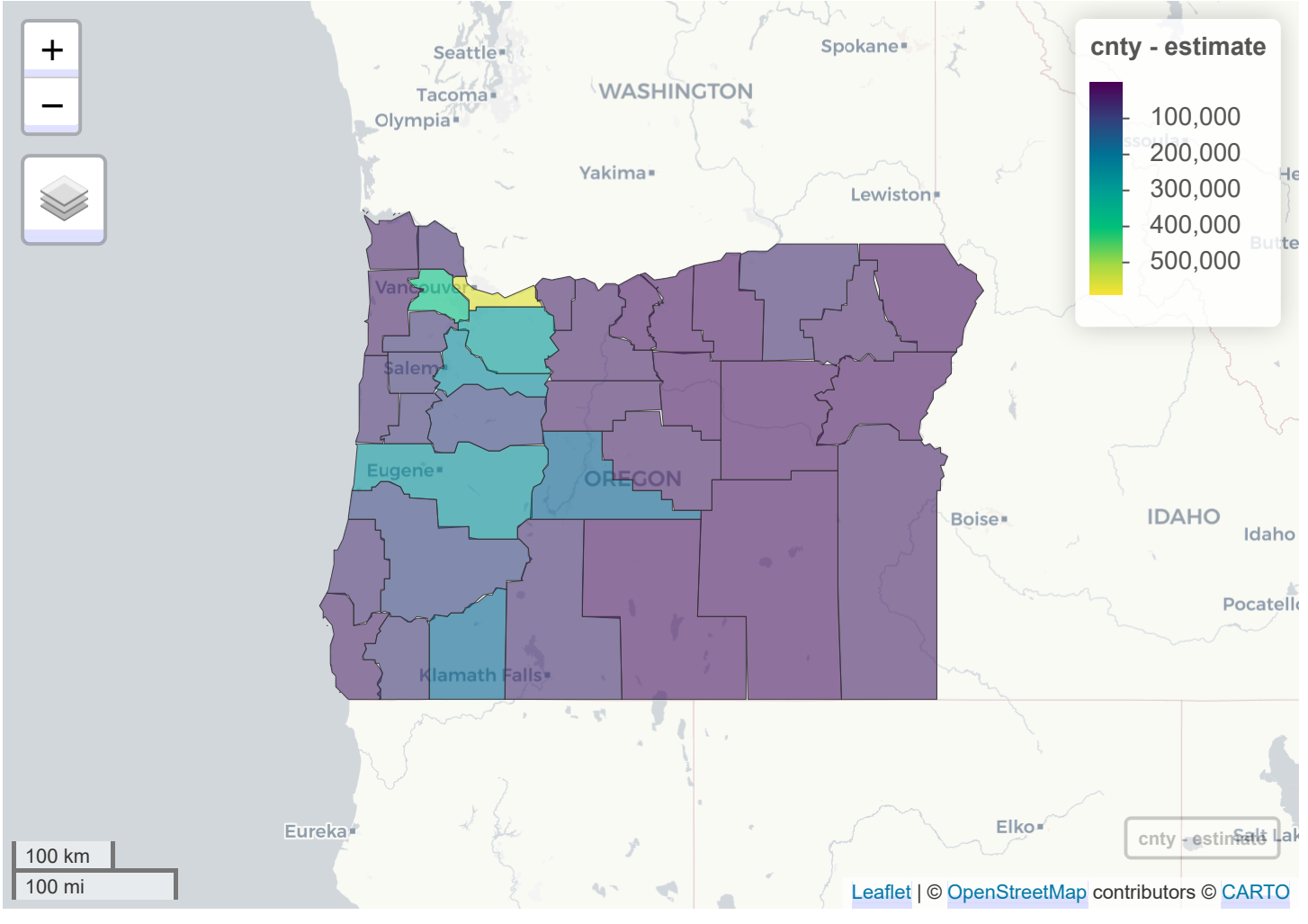
mapview

- Really quick easy interactive maps

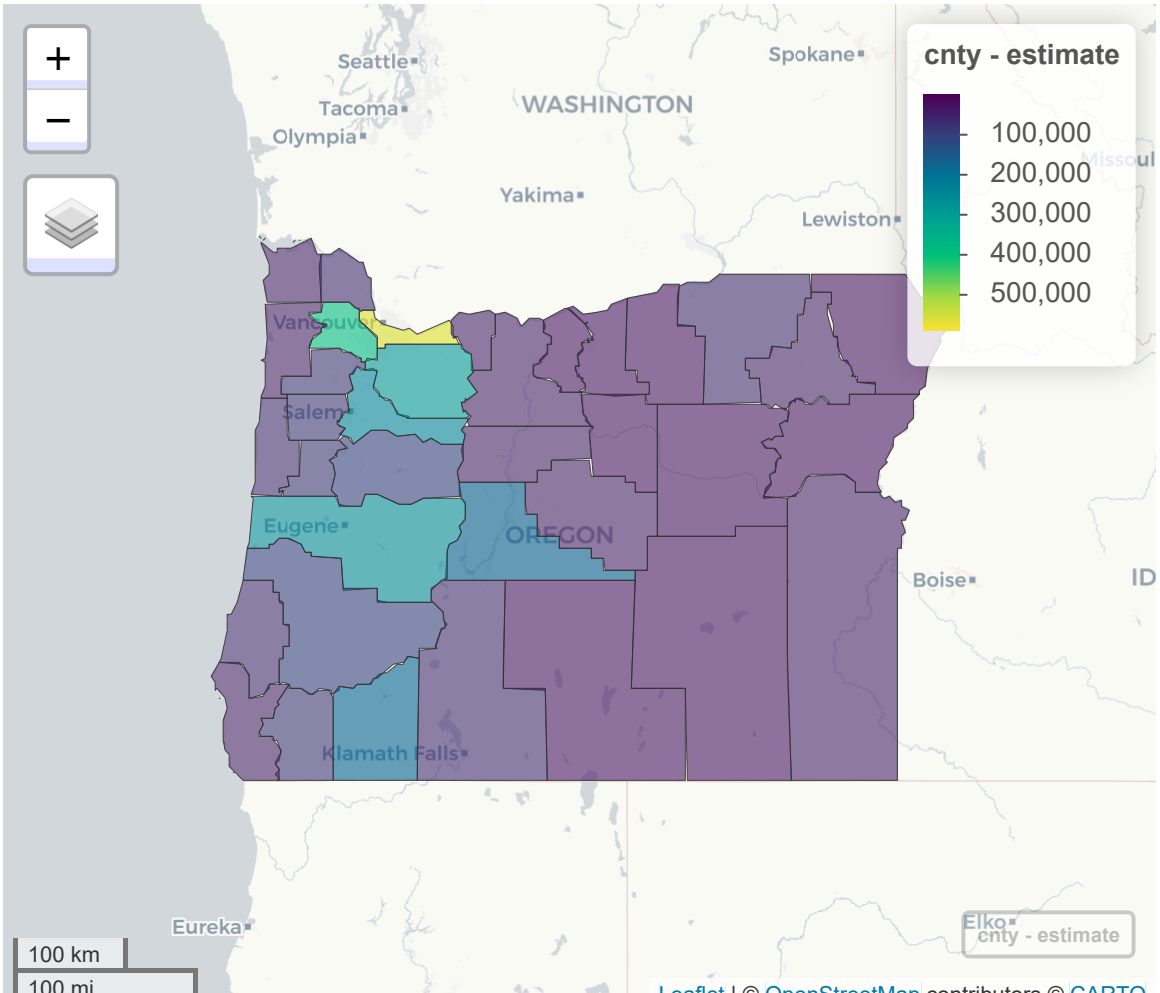
```
library(mapview)
mapview(cnty)
```



```
mapview(cnty, zcol = "estimate")
```



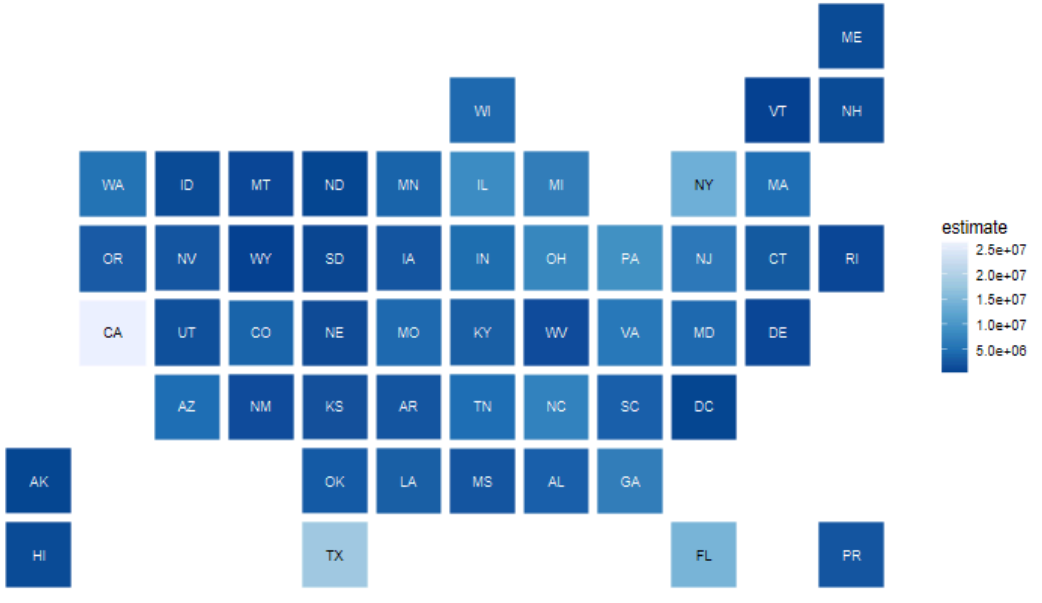
```
mapview(cnty,
        zcol = "estimate",
        popup = leafpop::popupTable(cnty,
                                    zcol = c("NAME", "estimate")))
```



A few other things of note

statebins

```
library(statebins)
statebins(states,
          state_col = "NAME",
          value_col = "estimate") +
  theme_void()
```



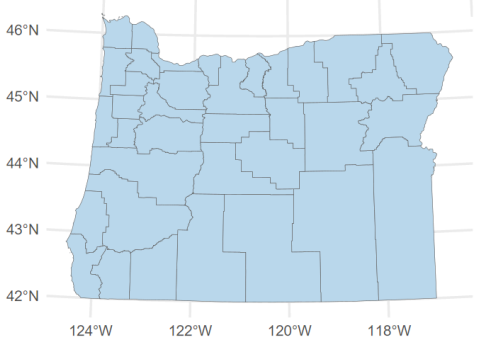
Cartograms

```
library(cartogram)
or_county_pop <- get_acs(
  geography = "county",
  state = "OR",
  variables = "B01001E_001",
  year = 2018,
  geometry = TRUE
)
```

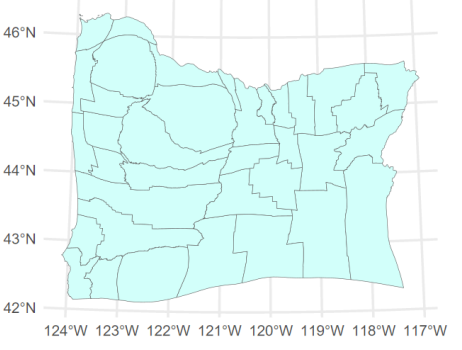
```
##
  |
  |                                                                      |   0%
  |
  |                                                                      |   1%
  |
  |=                                                                     |   1%
  |
  |=                                                                     |   2%
  |
  |==                                                                    |   4%
  |
  |===                                                                   |   4%
  |
  |====                                                                  |   7%
```

Compare

```
ggplot(or_county_pop) +
  geom_sf(fill = "#BCD8EB")
```




```
ggplot(carto_counties) +
  geom_sf(fill = "#D5FFFA")
```

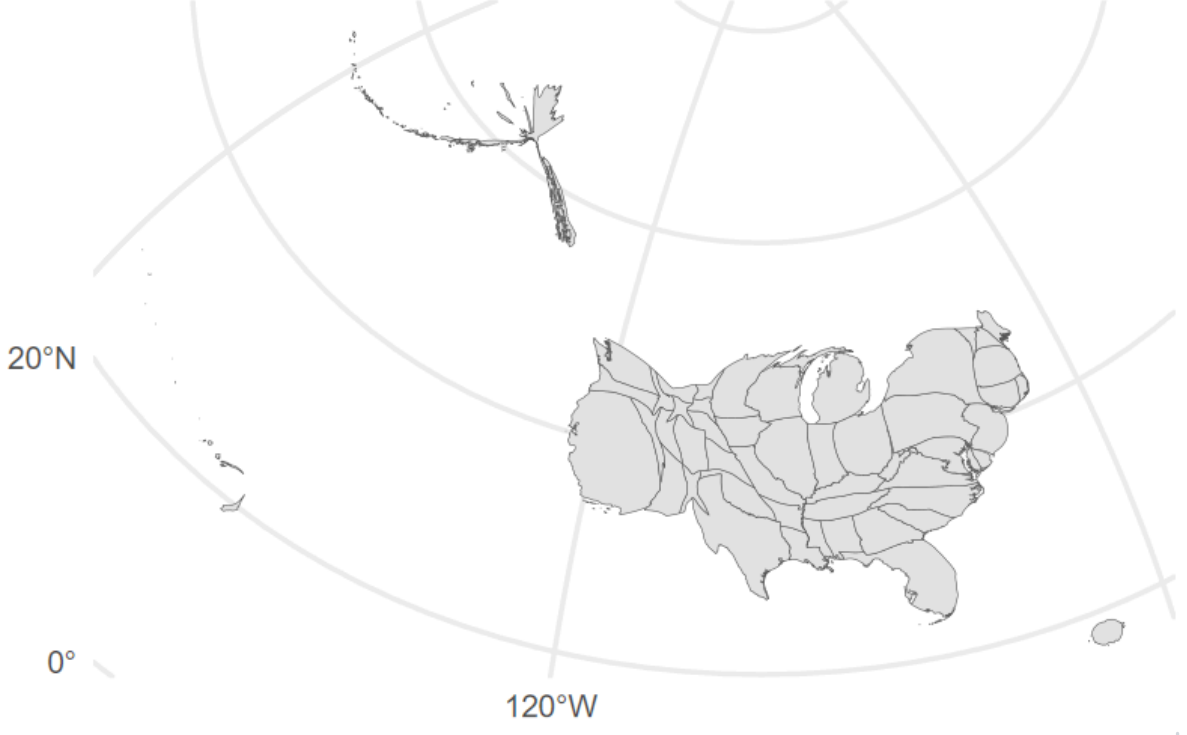


State

```
state_pop <- get_acs(
  geography = "state",
  variables = "B00001_001",
  year = 2018,
  geometry = TRUE
)

# Set projection
state_pop <- st_transform(state_pop, crs = 2163)

# found  the CRS here: https://epsg.io/transform#s_srs=3969&t_srs
carto_states <- cartogram_cont(state_pop, "estimate")
```

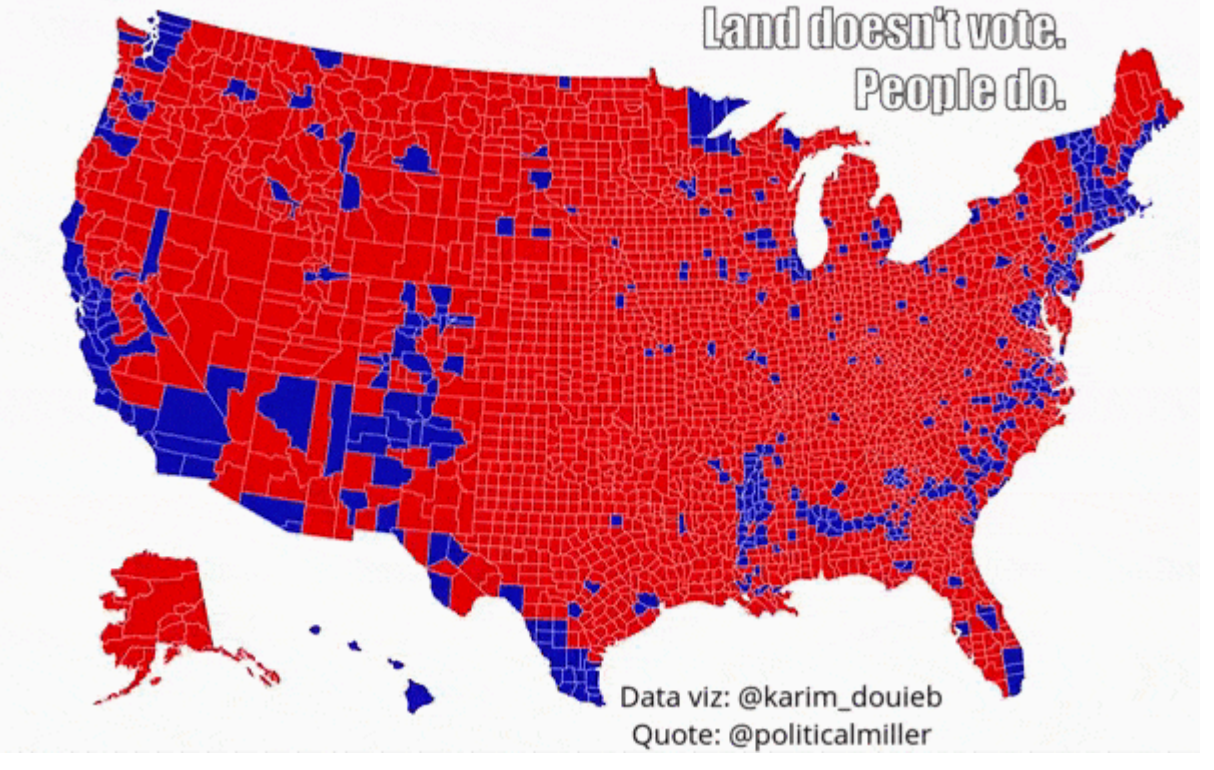
Cartogram of USA by population

```
ggplot(carto_states) +
  geom_sf()
```



Last note

You may or may not like cartograms. Just be careful not to lie with maps.



Final Project - Let's look at the rubric again!

Next time

- Wrap up maps and loose ends with HTML
- More time to work on final project and peer-review of other's visualization