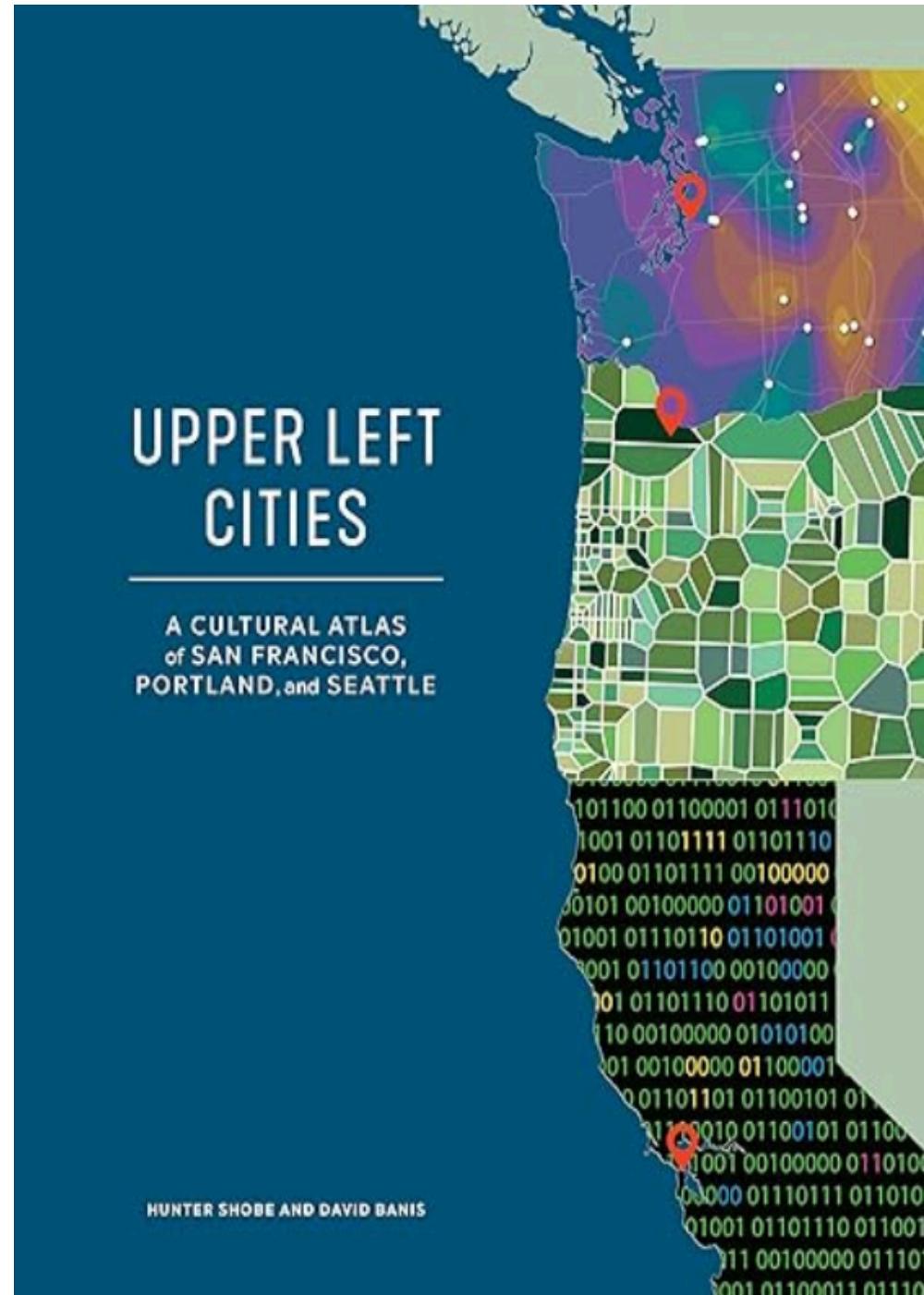


Cartographic Tour of R and `{ggplot2}`

Justin Sherrill

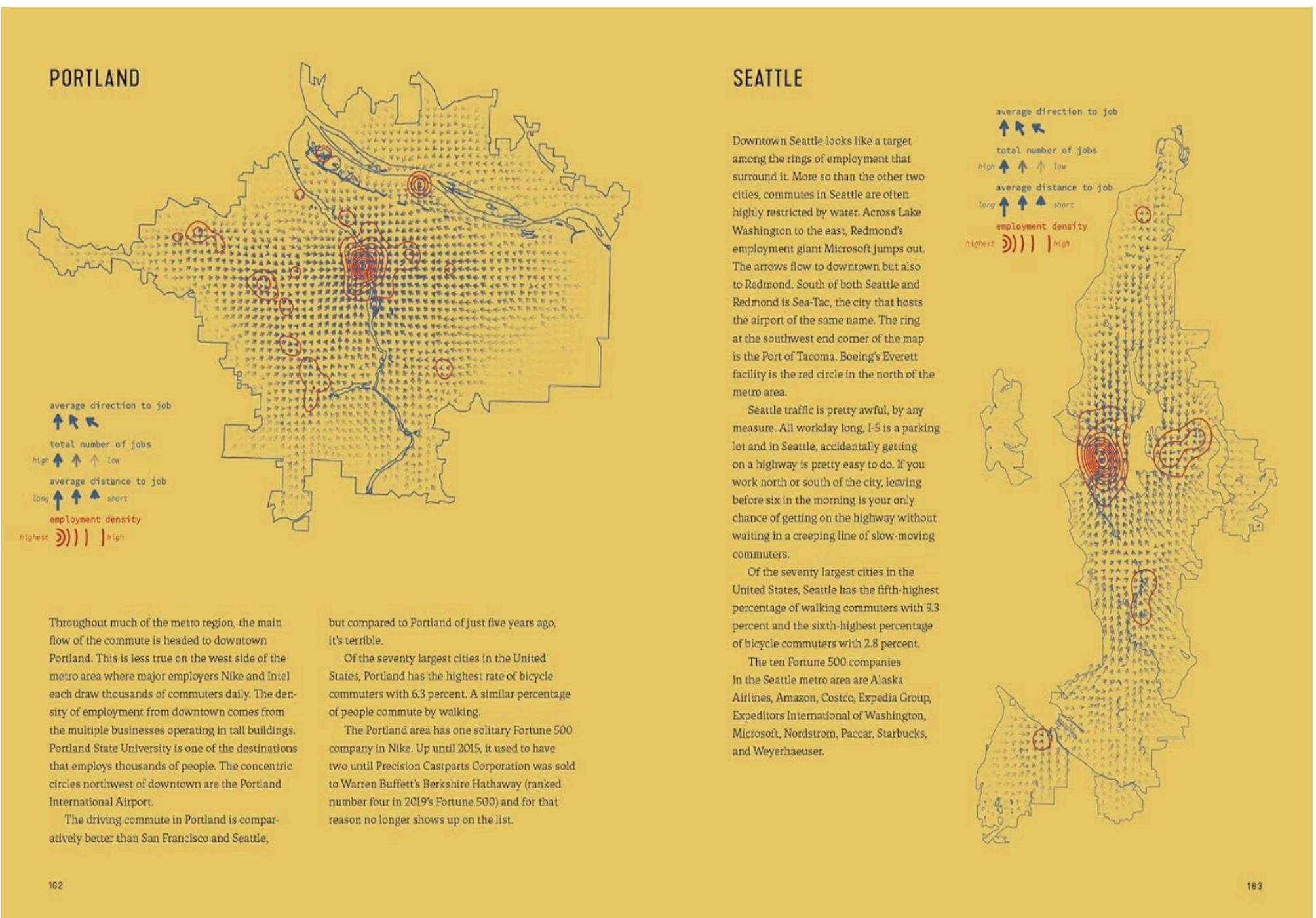
Who am I?

- Analyst with EConorthwest
- Data visualization, housing & demographic analysis, mapping
- R maps featured in *Upper Left Cities: A Cultural Atlas of San Francisco, Portland, and Seattle*



Who am I?

- Analyst with ECONorthwest
- Data visualization, housing & demographic analysis, mapping
- R maps featured in *Upper Left Cities: A Cultural Atlas of San Francisco, Portland, and Seattle*



Cartography in R

Keep it in {ggplot2}. Why?

- Reinforces learning the ins and outs of one of the strongest dataviz tools in existence
- Flexible enough for the needs of the project and your personal style
- Good chance of redundancy - many overlapping
- Meet exciting new developers learn new ways of analysis & visualization

Cartography in R

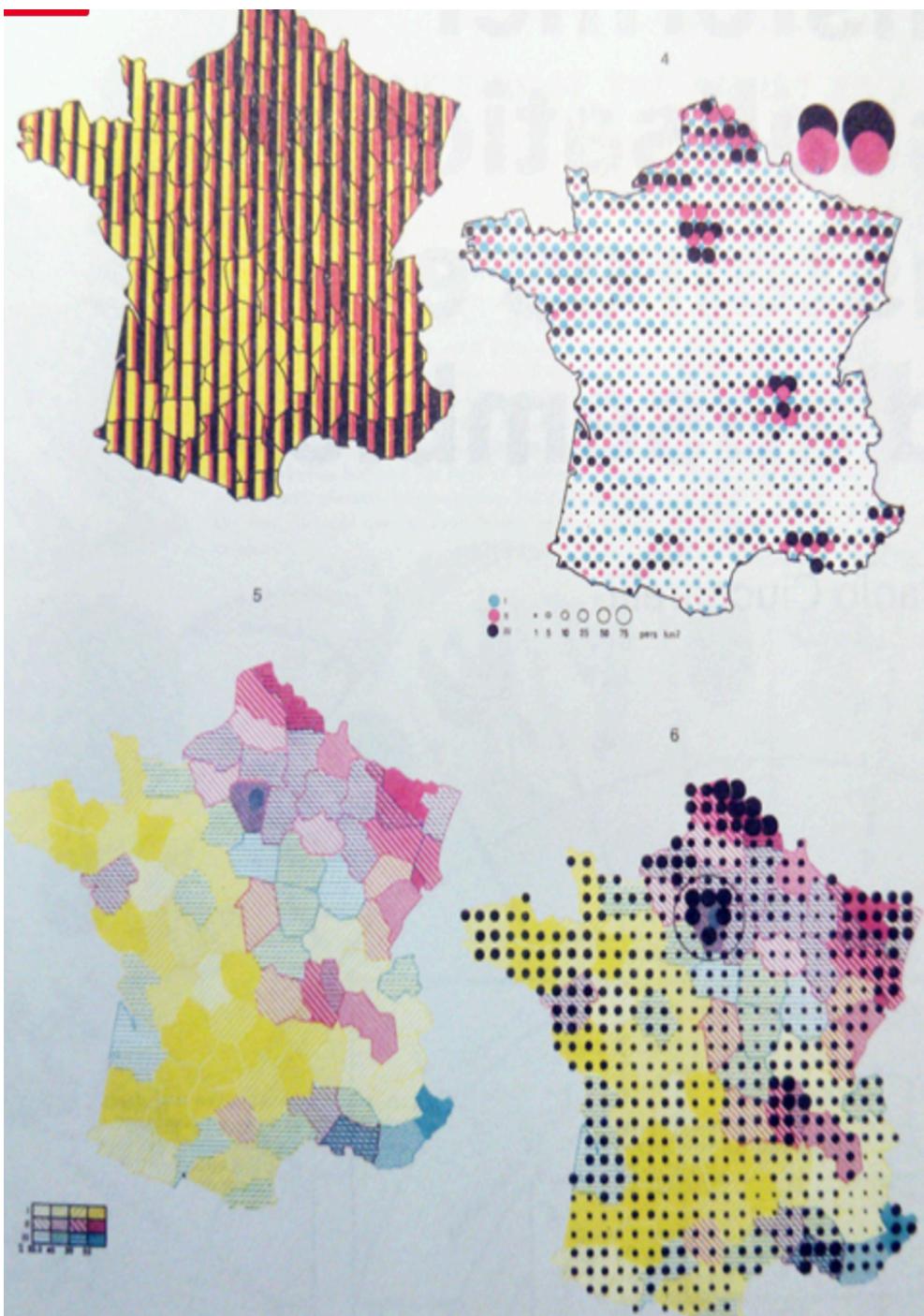
- Sometimes you need a custom package to accomplish the task at hand
- Other times, {sf} and {ggplot2} had the solution all along

What is “good” cartography?

- Visual hierarchy
- Legibility
- Figure-ground
- Balance
- Making choices

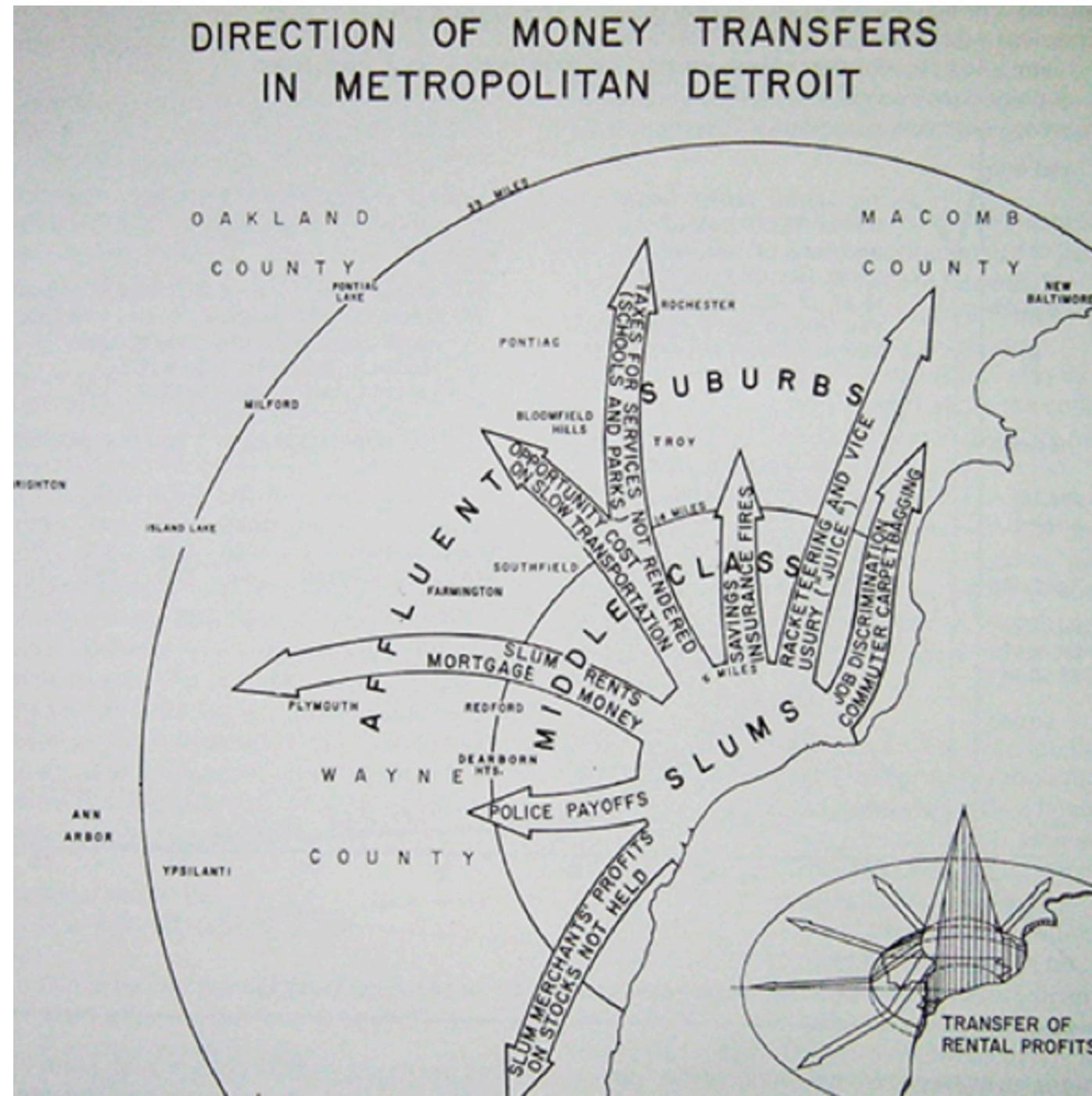
How to make a good map

- Copy a good map
- Jacques Bertin
- William Bunge
- Timo Grossenbacher



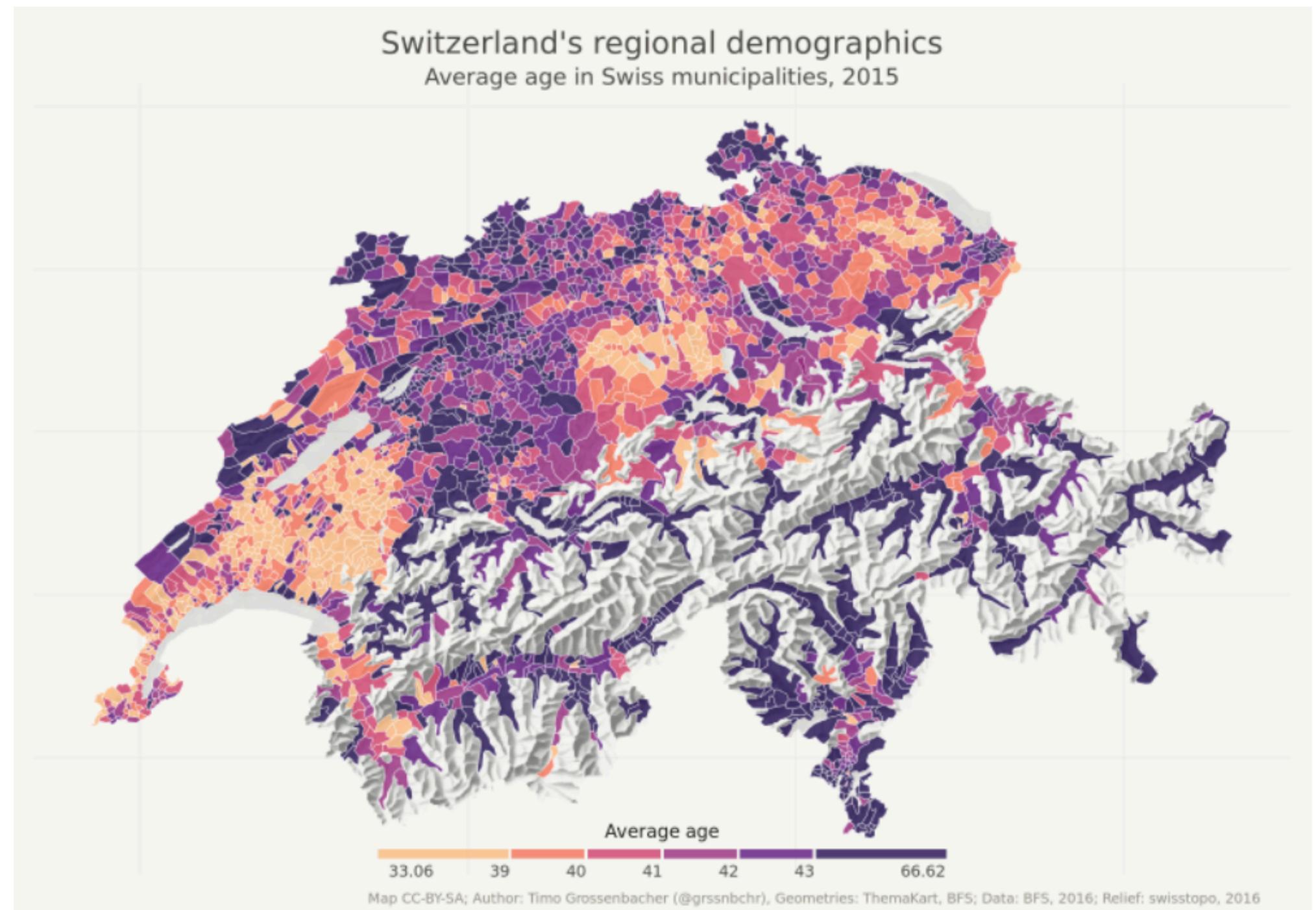
How to make a good map

- Copy a good map
- Jacques Bertin
- William Bunge
- Timo Grossenbacher



How to make a good map

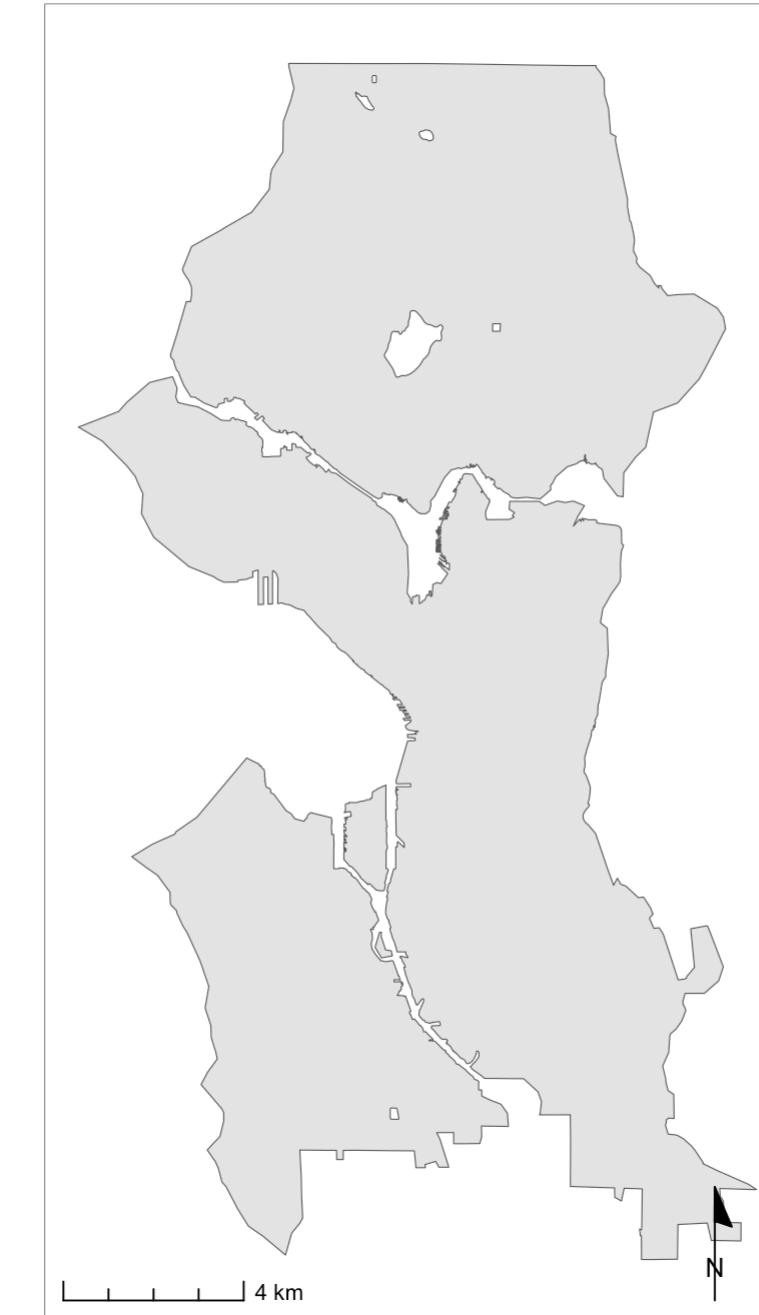
- Copy a good map
- Jacques Bertin
- William Bunge
- Timo Grossenbacher



Marginalia

📎 {ggspatial} can do so much, including easily add a north arrow and scale bar to your map

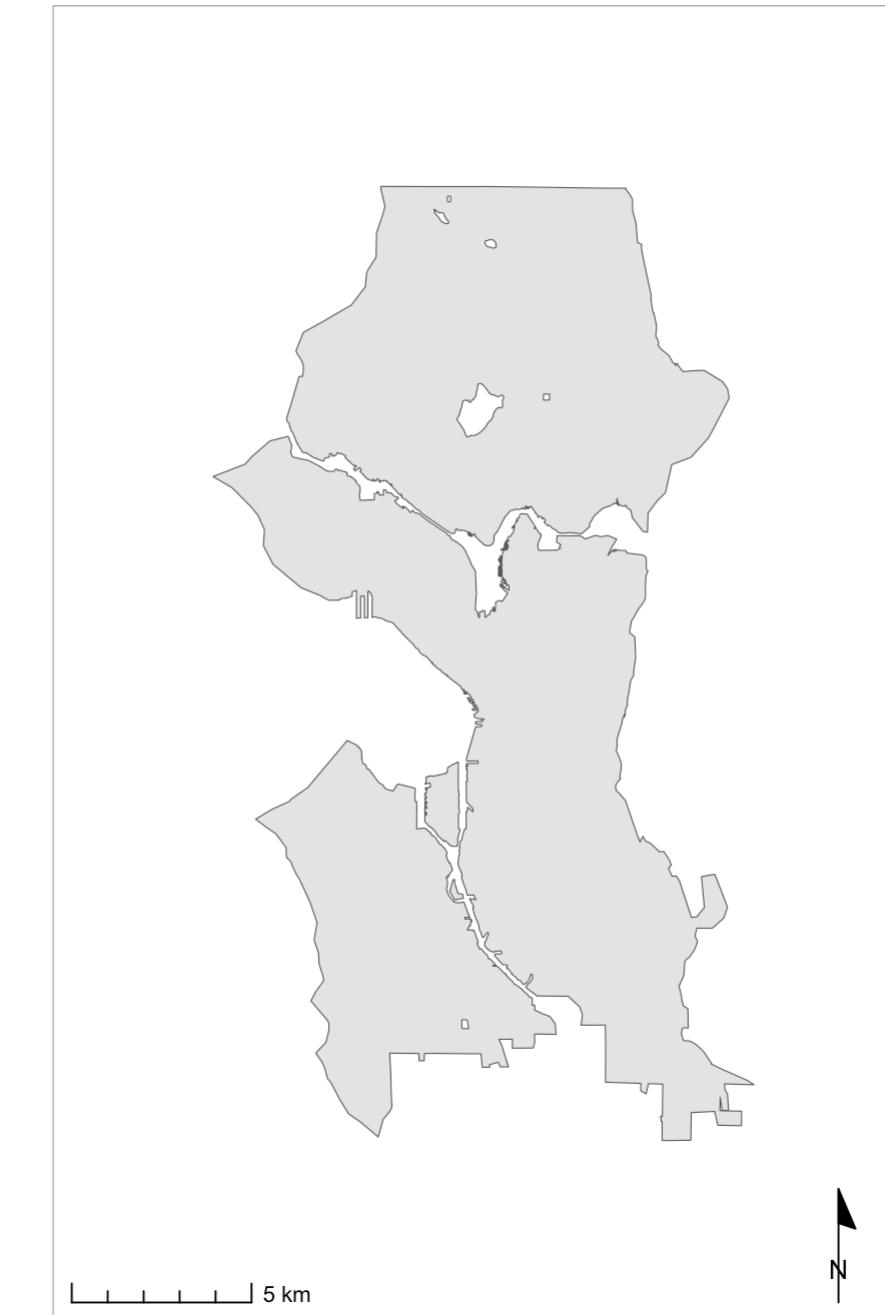
```
1 sea <- places("wa",
2                   cb = TRUE) %>%
3   filter(NAME == "Seattle") %>%
4   st_transform(3857) %>%
5   erase_water(area_threshold = .25)
6
7 ggplot() +
8   geom_sf(data = sea) +
9   ggspatial::annotation_north_arrow(location = "br",
10                                     which_north = "true",
11                                     style = north_arrow_minimal) +
12   annotation_scale(location = "bl",
13                     style = "ticks") +
14   coord_sf(crs = 3857,
15             datum = NA) +
16   theme_minimal() +
17   theme(panel.border = element_rect(linewidth = .2, fill = NA))
```



Marginalia, cont'd

📎 Giving your marginalia a little breathing room can be a good move. So expand your limits within `coord_sf()`!

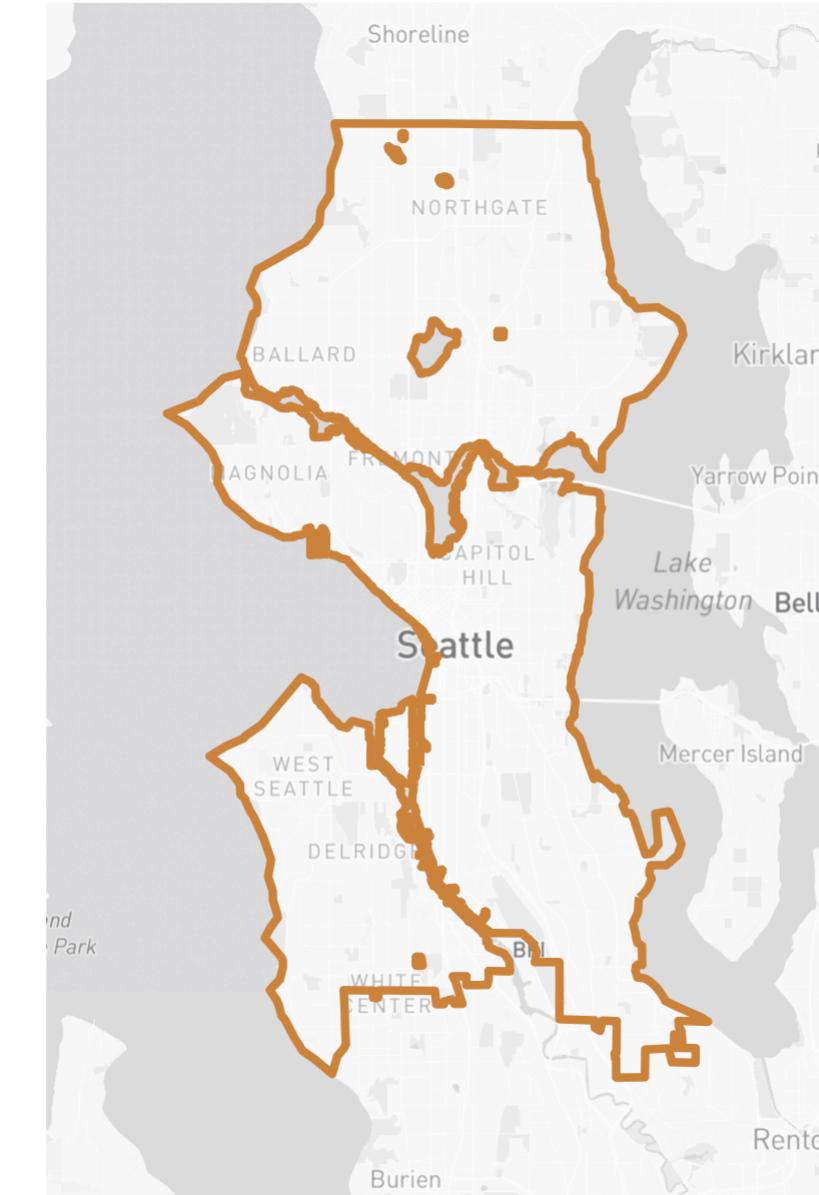
```
1 sea <- places("wa",
2                 cb = TRUE) %>%
3   filter(NAME == "Seattle") %>%
4   st_transform(3857) %>%
5   erase_water(area_threshold = .25)
6
7 xlims <- st_buffer(sea, 5000) %>%
8   st_bbox() %>%
9   .[c(1, 3)]
10
11 ylims <- st_buffer(sea, 5000) %>%
12   st_bbox() %>%
13   .[c(2, 4)]
14
15 ggplot() +
16   geom_sf(data = sea) +
17   ggspatial::annotation_north_arrow(location = "br",
18                                     which_north = "true",
19                                     style = north_arrow_minimal) +
20   annotation_scale(location = "bl",
21                     style = "ticks") +
22   coord_sf(crs = 3857,
23             datum = NA,
24             xlim = xlims,
25             ylim = ylims) +
26   theme_minimal() +
```



Basemaps

📎 Kyle Walker's {mapboxapi} + {ggspatial} = fast, easy basemaps to add context to your map

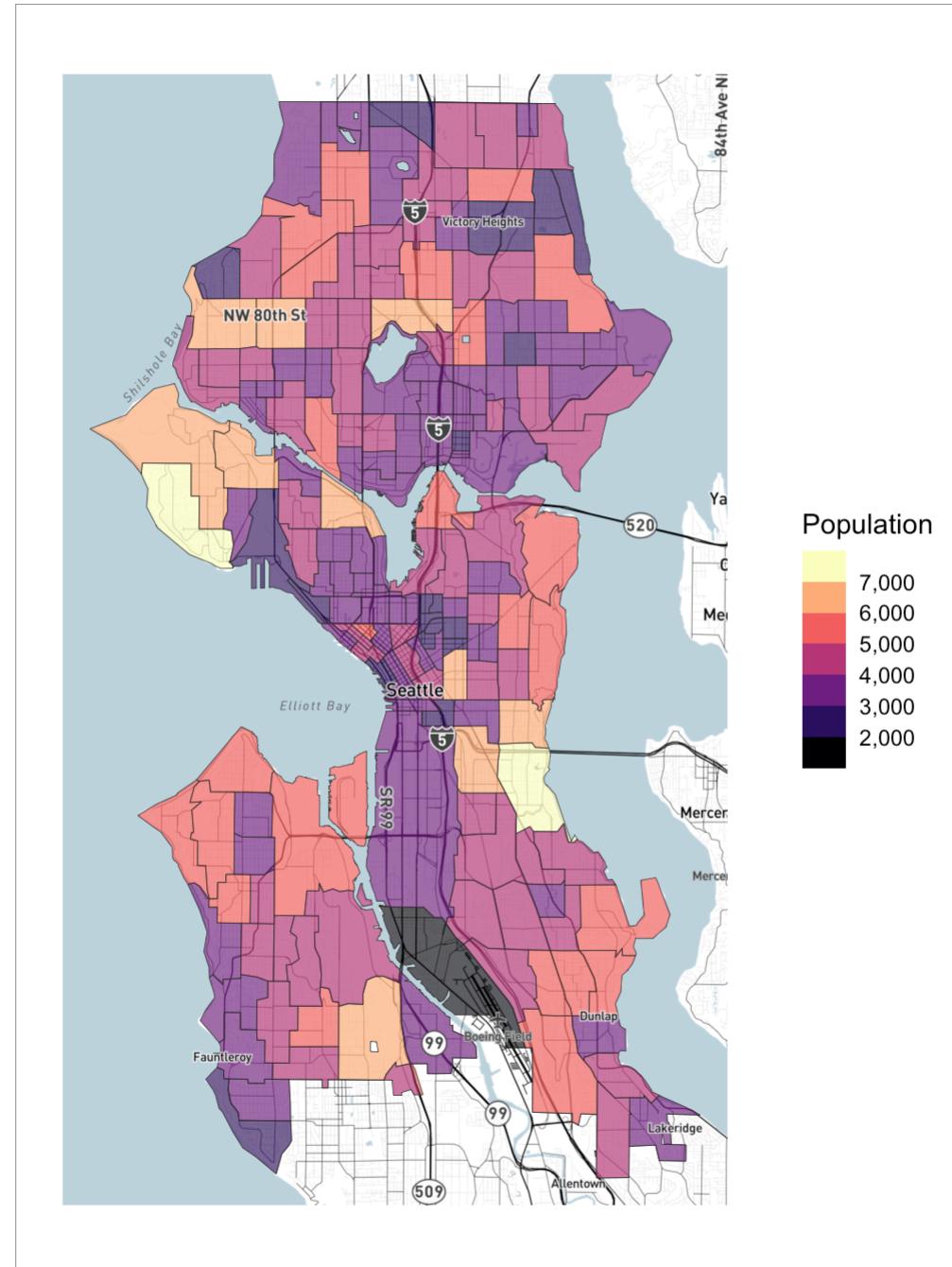
```
1 basemap <- get_static_tiles(  
2   location = st_buffer(sea, 5000),  
3   zoom = 10,  
4   buffer_dist = 0,  
5   scaling_factor = "2x",  
6   crop = TRUE,  
7   style_id = basic_id,  
8   username = me  
9 )  
10  
11 ggplot() +  
12   layer_spatial(data = basemap) +  
13   geom_sf(data = sea,  
14     fill = NA,  
15     color = "peru",  
16     linewidth = 1.5) +  
17   coord_sf(crs = 3857,  
18     datum = NA,  
19     xlim = xlims,  
20     ylim = ylims) +  
21   theme_minimal()
```



Basemaps, cont'd

Try breaking your basemap into two layers: one for the labels and one for the map itself, using Mapbox studio (or other resource), and then layer them with multiple calls to `layer_spatial()`

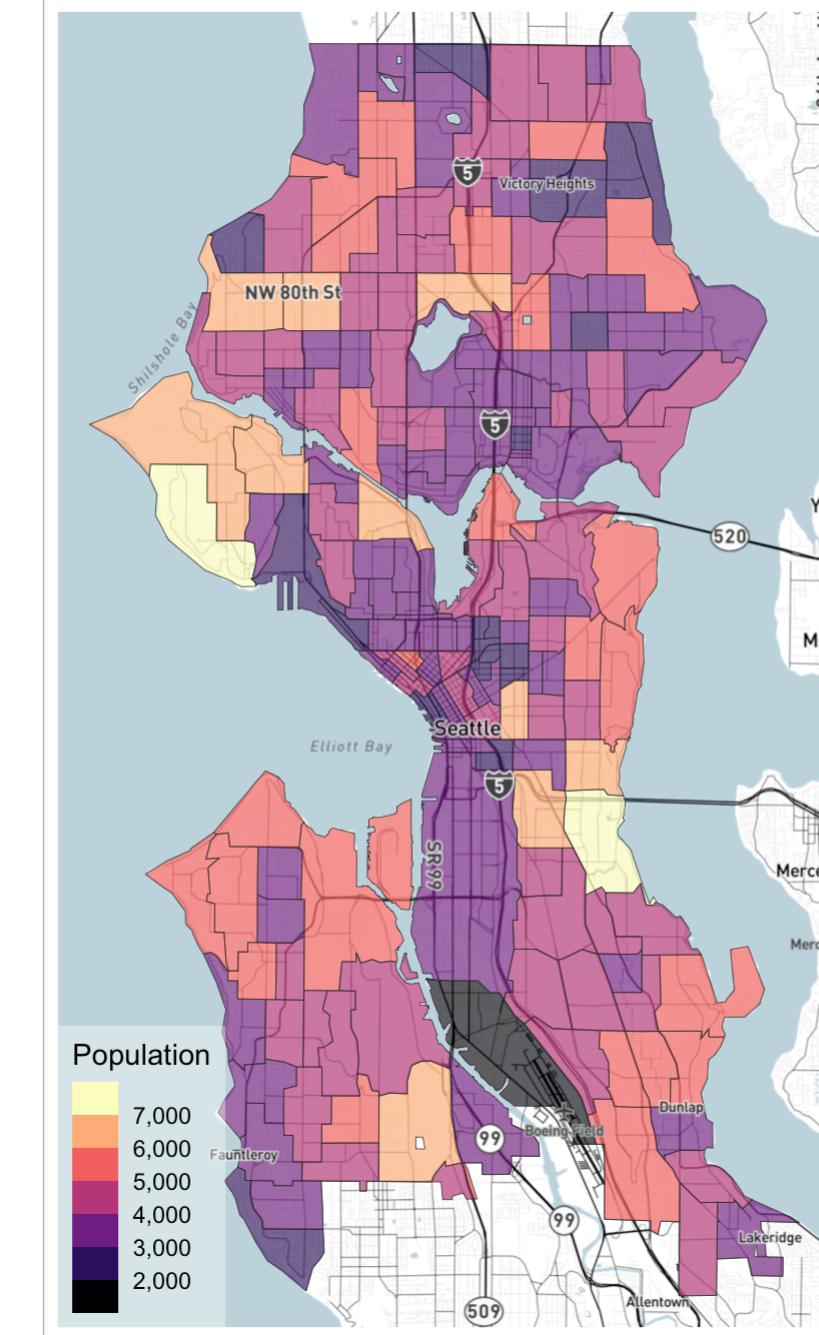
```
1 tract_pop <- get_acs("tract",
2                         state = "WA",
3                         county = "King",
4                         variables = "B01001_001",
5                         geometry = TRUE) %>%
6   st_transform(st_crs(sea)) %>%
7   st_intersection(sea)
8
9 basemap <- get_static_tiles(
10   location = sea,
11   zoom = 11,
12   buffer_dist = 1000,
13   scaling_factor = "2x",
14   crop = TRUE,
15   style_id = basemap_id,
16   username = me
17 )
18
19 labels <- get_static_tiles(
20   location = sea,
21   zoom = 11,
22   buffer_dist = 1000,
23   scaling_factor = "2x",
24   crop = TRUE,
25   style_id = labels_id,
26   username = me
27 )
```



Legends

📎 Positioning your legend inside the map can be a good move, especially if you have a lot of white space you'd prefer not to deal with

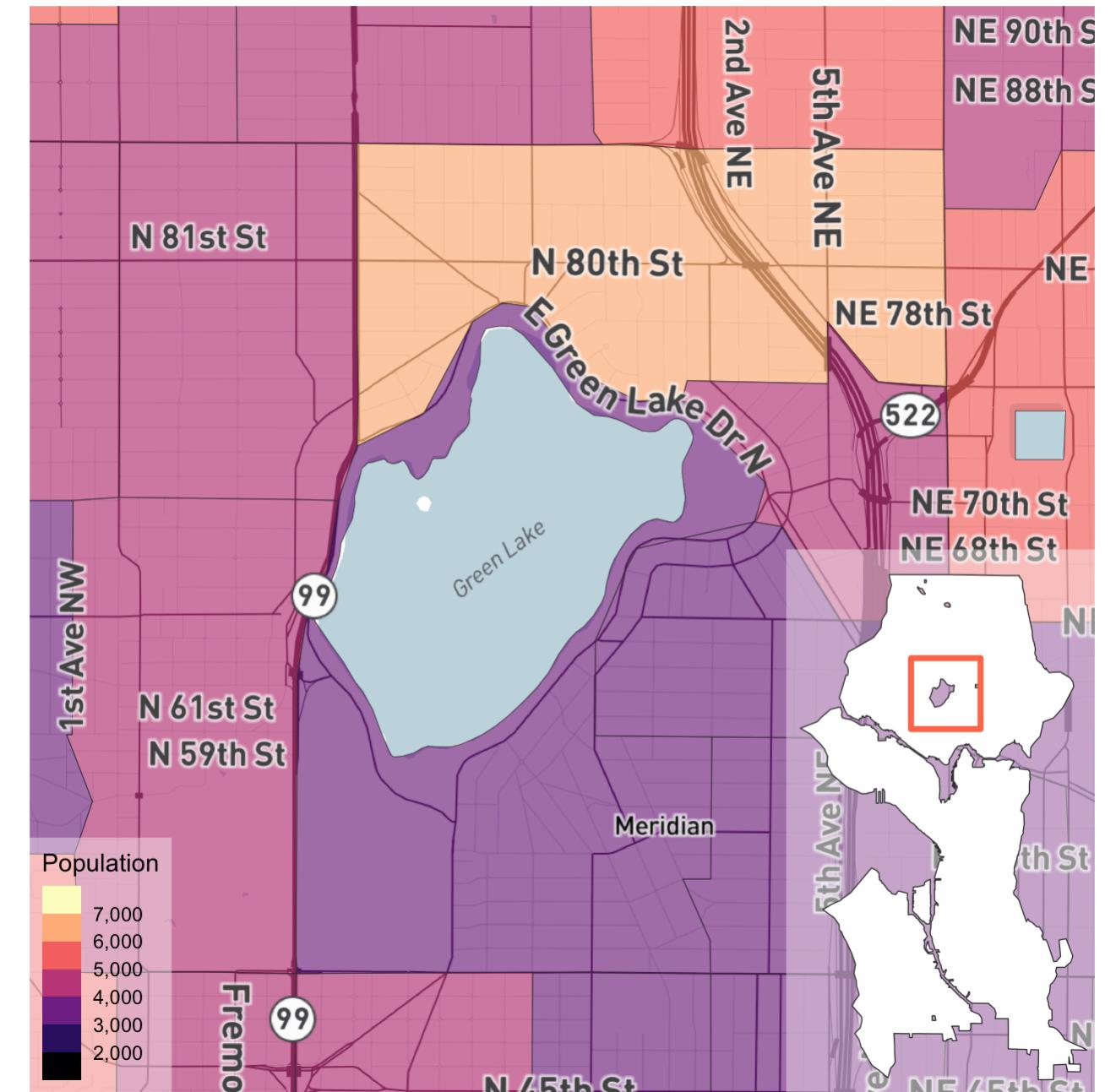
```
1 ggplot() +
2   layer_spatial(data = basemap) +
3   geom_sf(data = tract_pop,
4     aes(fill = estimate),
5     color = "gray20",
6     linewidth = .15,
7     alpha = .7) +
8   layer_spatial(data = labels) +
9   scale_fill_viridis_b(option = "A",
10    direction = 1,
11    name = "Population",
12    n.breaks = 7,
13    labels = comma) +
14   coord_sf(crs = 3857,
15    datum = NA,
16    # This one is necessary!
17    expand = 0) +
18   theme_minimal() +
19   theme(legend.position = "inside",
20     legend.location = "plot",
21     legend.position.inside = c(0, 0),
22     legend.justification.inside = c(0, 0),
23     # element_rect() doesn't have an alpha argument, but you c
24     legend.box.background = element_rect(fill = "#fffffe60", c
25   theme(plot.background = element_rect(fill = "white", color = "gr
```



Insets

📎 You can use {patchwork} to compose a simple inset map composition. Here you're really just arranging two separate plots.

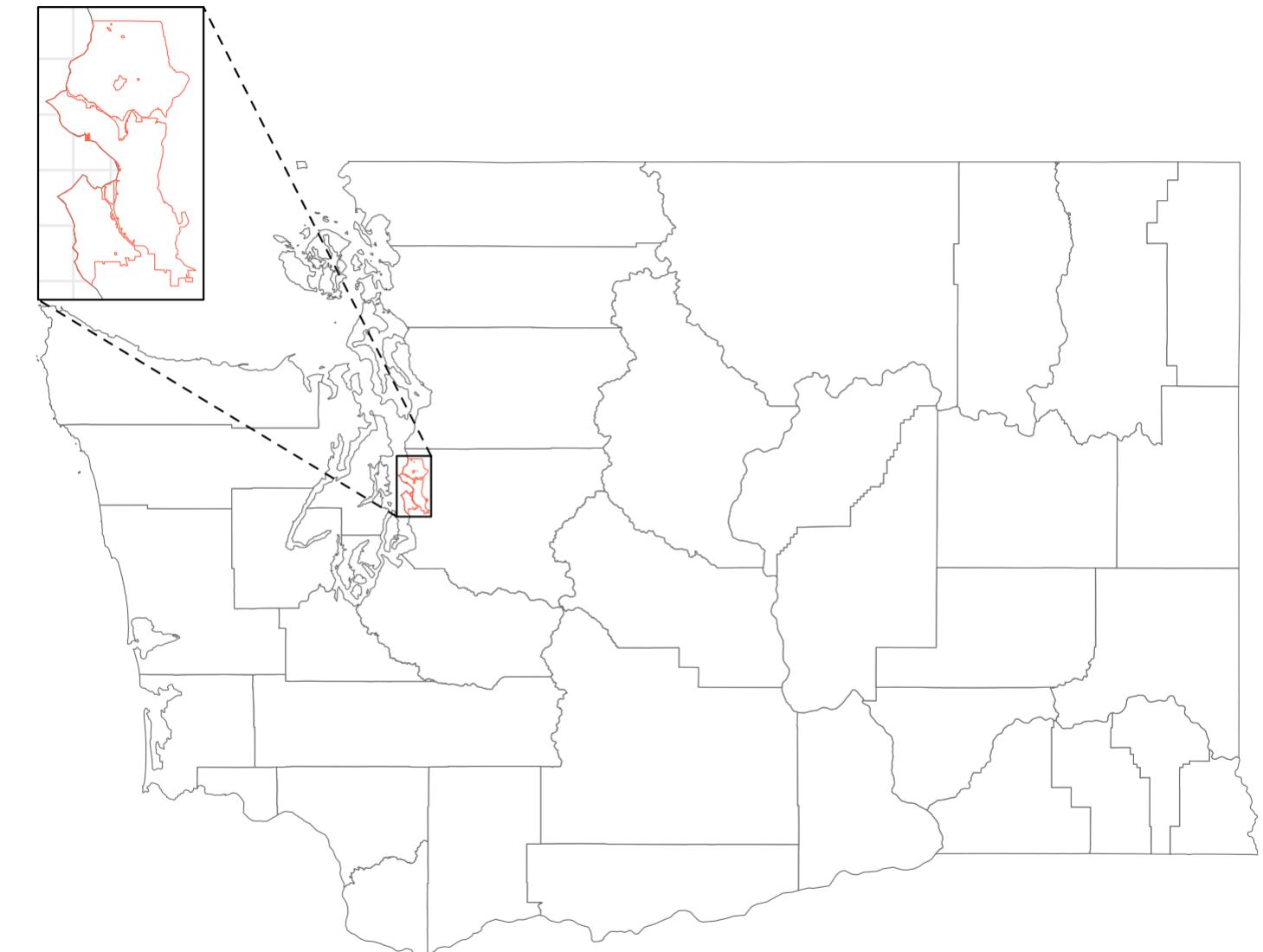
```
1 greenlake <- st_as_sfc("POLYGON ((-13620987 6050695, -13620987 605  
2   st_sf()  
3  
4 xlims <- st_bbox(greenlake)[c(1, 3)]  
5  
6 ylims <- st_bbox(greenlake)[c(2, 4)]  
7  
8 basemap <- get_static_tiles(  
9   location = greenlake,  
10  zoom = 13,  
11  buffer_dist = 1000,  
12  scaling_factor = "2x",  
13  crop = TRUE,  
14  style_id = basemap_id,  
15  username = me  
16 )  
17  
18 labels <- get_static_tiles(  
19   location = greenlake,  
20   zoom = 13,  
21   buffer_dist = 1000,  
22   scaling_factor = "2x",  
23   crop = TRUE,  
24   style_id = labels_id,  
25   username = me  
26 )  
~ ~
```



Insets

📎 Or, {ggmagnify} could let you do a more traditional inset, if that's what you're looking for

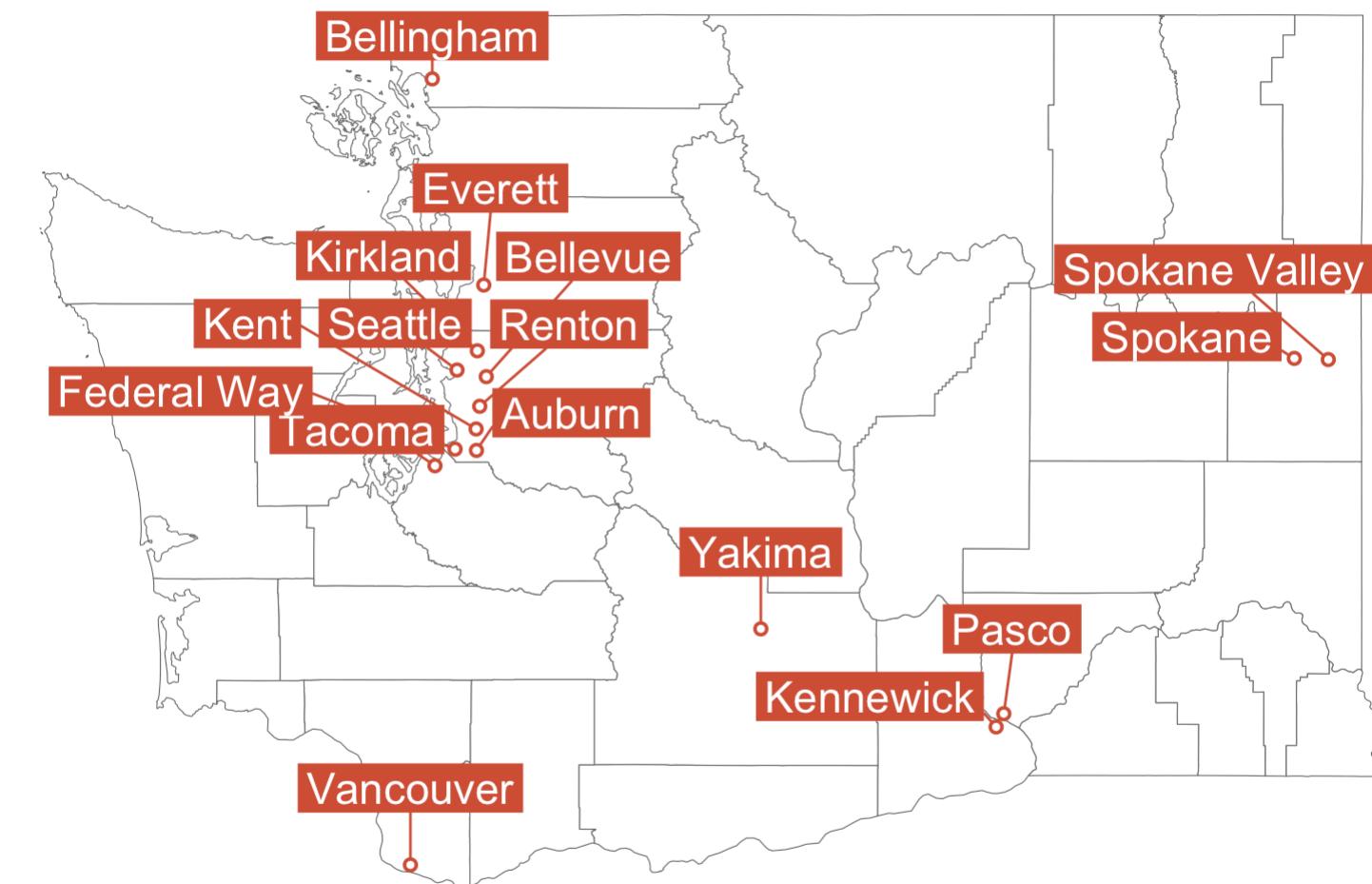
```
1 counties <- counties("wa",
2   cb = TRUE) %>%
3   st_transform(3857)
4
5 to <- st_as_sfc("POLYGON ((-13881692 6185920, -13881692 6376113, -
6   crs = 3857) %>%
7   st_sf()
8
9 xlims <- st_bbox(st_union(counties, to))[c(1, 3)]
10
11 ylims <- st_bbox(st_union(counties, to))[c(2, 4)]
12
13 county_map <- ggplot() +
14   geom_sf(data = counties,
15     fill = "white",
16     color = "gray40",
17     linewidth = .2) +
18   geom_sf(data = sea,
19     fill = NA,
20     color = "tomato",
21     linewidth = .2) +
22   coord_sf(crs = 3857,
23     datum = NA,
24     # This one is necessary!
25     expand = 0,
26     xlim = xlims,
27     ylim = ylims)
```



Labels

📎 `{ggrepel}` is a very powerful tool for labeling your map, especially when you have a lot of labels to place. It can be a bit fussy though, so it will likely take lots of trial and error. And don't sleep on the `bg.colour` argument for `geom_text_repel()`!

```
1 cities <- get_acs("place",
2                           state = "wa",
3                           variables = "B01001_001",
4                           year = 2022,
5                           geometry = TRUE) %>%
6   filter(estimate > 7.5e4) %>%
7   mutate(NAME = str_remove_all(NAME, paste0(c(" city, Washington",
8     mutate(class = santoku::chop(estimate, breaks = c(1e5, 1.5e5,
9     # filter(NAME %in% c("Seattle", "Tacoma", "Olympia", "Spokane",
10    st_transform(3857)
11
12 ggplot() +
13   geom_sf(data = counties,
14           fill = "white",
15           color = "gray40",
16           linewidth = .2) +
17   geom_label_repel(data = st_centroid(cities),
18                     aes(x = st_coordinates(geometry)[, 1],
19                           y = st_coordinates(geometry)[, 2],
20                           label = NAME,
21                           size = class),
22                     min.segment.length = 0,
23                     color = "white",
24                     fill = "tomato3",
25                     label.r = unit(0, "lines"),
26                     label.size = 0,
```



Labels, cont'd

- Using `st_inscribed_circle()` can help you find the appropriate centroid for unusual polygons, especially when `st_point_on_surface()` fails

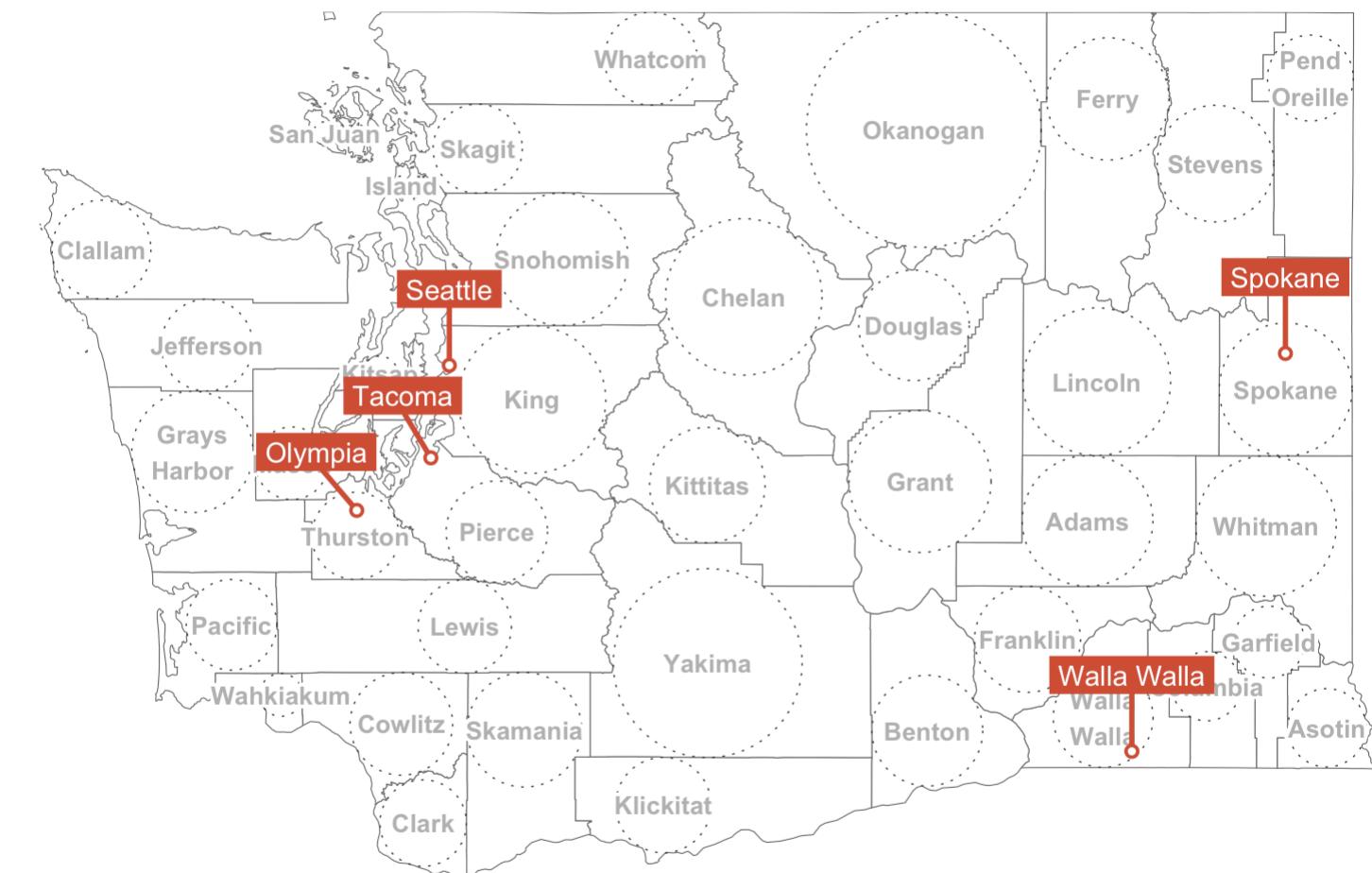
```
1 cities <- places("wa") %>%
2   filter(NAME %in% c("Seattle", "Tacoma", "Olympia", "Spokane", "W
3   st_transform(3857)
4
5 county_lbls <- st_point_on_surface(counties) %>%
6   transmute(name = NAME,
7             x = st_coordinates(.)[, 1],
8             y = st_coordinates(.)[, 2])
9
10 ggplot() +
11   geom_sf(data = counties,
12           fill = "white",
13           color = "gray40",
14           linewidth = .2) +
15   geom_text_repel(data = county_lbls,
16                   aes(x = x,
17                         y = y,
18                         label = str_wrap(name, 10)),
19                   # ggrepel functions have this bg.colour argument
20                   bg.colour = "white",
21                   bg.r = .2,
22                   force = 0,
23                   color = "gray70",
24                   fontface = "bold",
25                   size = 3.5) +
26   geom_label_repel(data = st_centroid(cities),
```



Labels, cont'd

- Using `st_inscribed_circle` can help you find the appropriate centroid for unusual polygons, especially when `st_point_on_surface` fails

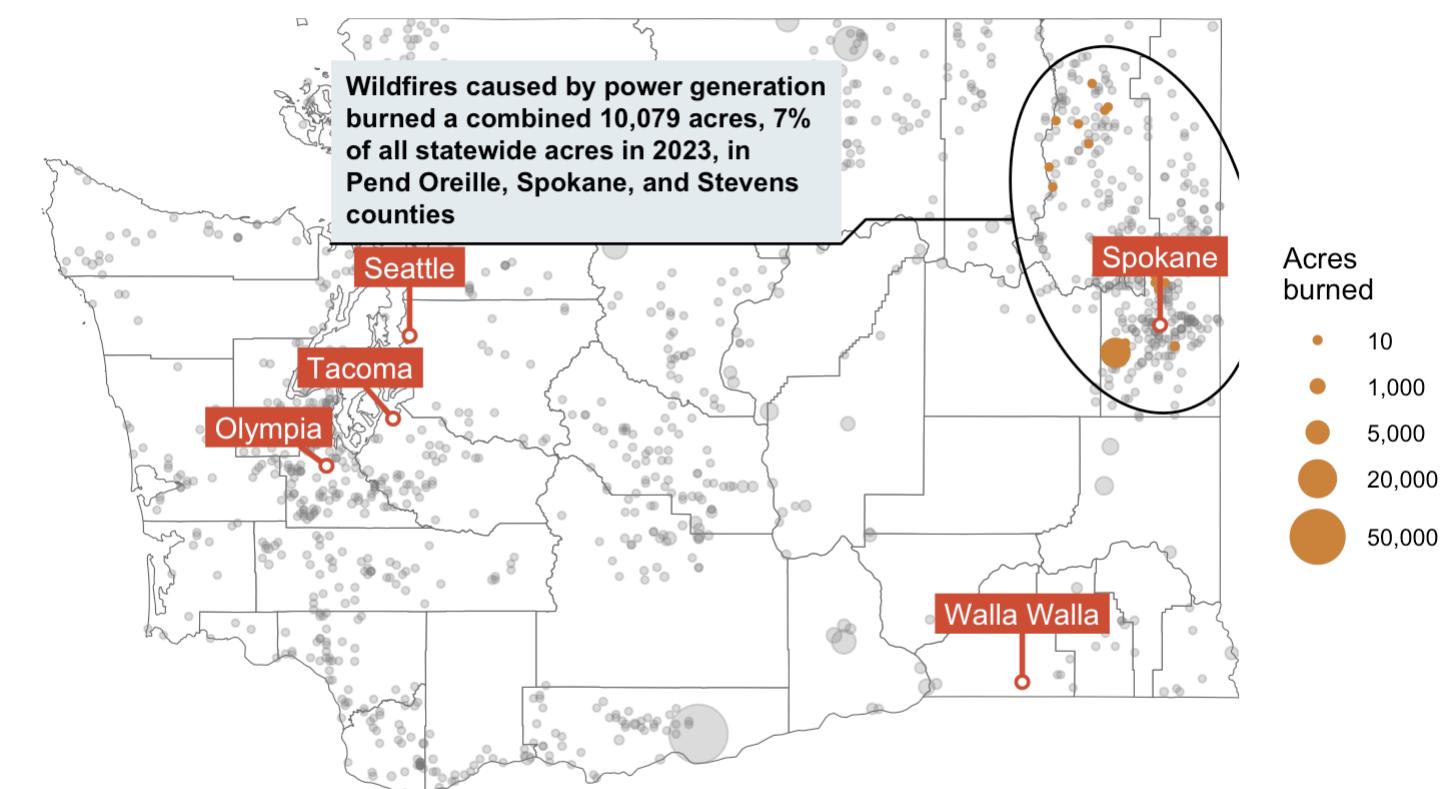
```
1  circles <- counties %>%
2    pull(geometry) %>%
3    st_inscribed_circle() %>%
4    st_sf() %>%
5    filter(!st_is_empty(geometry))
6
7  county_lbls <- circles %>%
8    st_centroid() %>%
9    transmute(name = counties$NAME,
10              x = st_coordinates(.)[, 1],
11              y = st_coordinates(.)[, 2])
12
13 ggplot() +
14   geom_sf(data = counties,
15           fill = "white",
16           color = "gray40",
17           linewidth = .2) +
18   geom_sf(data = circles,
19           fill = NA,
20           color = "gray20",
21           linewidth = .25,
22           linetype = "dotted") +
23   geom_text_repel(data = county_lbls,
24                   aes(x = x,
25                         y = y,
26                         label = str_wrap(name, 10)),
27                   ...)
```



Labels, cont'd

📎 {ggforce} has a `geom_mark_...()` family of functions that are nicely styled and can be used to label clusters of points or other geometries

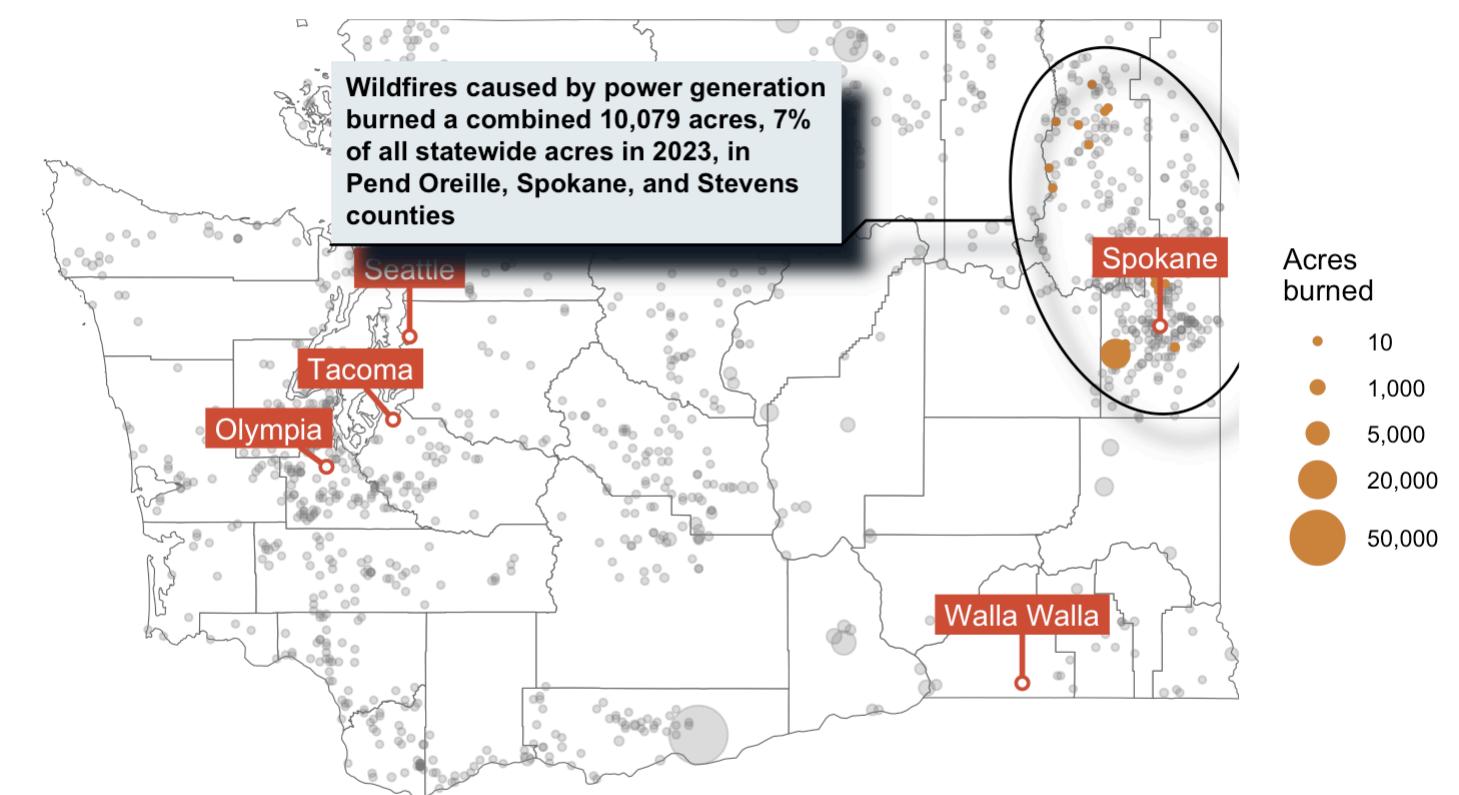
```
1 # Read in the data, in this case this CSV from WA DNR https://data
2 fires <- read_csv("/Users/sherrill/Projects/CascadiaR_Cartographic
3   st_as_sf(coords = c("LON_COORD", "LAT_COORD"), crs = 4326) %>%
4   st_transform(3857) %>%
5   transmute(acres_burned = ACRES_BURNED,
6             cause = FIRECAUSE_LABEL_NM,
7             year = str_sub(DSCVR_DT, 1, 4)) %>%
8   st_as_sf() %>%
9   filter(year == 2023,
10         acres_burned > 0)
11
12 fires_ne <- fires %>%
13   filter(year == 2023,
14         cause == "Power Gen") %>%
15   st_filter(counties %>%
16             filter(NAME %in% c("Pend Oreille", "Sp
17   mutate(x = st_coordinates(geometry)[,1],
18         y = st_coordinates(geometry)[,2])
19
20 tot <- sum(fires_ne$acres_burned)
21
22 tot_2023 <- sum(fires$acres_burned)
23
24 pct <- tot/tot_2023
25
26 anchor <- st_centroid(st_union(fires_ne)) %>%
27   st_as_sf()
```



Labels, cont'd

📎 `{ggfx}` has a number of graphical filtering functions. You could pair it with `{ggforce}` to create a poppy label box with a drop-shadow effect

```
1 all_fires <- read_csv("/Users/sherrill/Projects/CascadiaR_Cartogra-  
2     st_as_sf(coords = c("LON_COORD", "LAT_COORD"), crs = 4326) %>%  
3     st_transform(3857) %>%  
4     transmute(acres_burned = ACRES_BURNED,  
5                 cause = FIREGCAUSE_LABEL_NM,  
6                 year = str_sub(DSCVR_DT, 1, 4)) %>%  
7     st_as_sf()  
8  
9 fires <- all_fires %>%  
10    filter(year == 2023,  
11             acres_burned > 0)  
12  
13 fires_ne <- fires %>%  
14    filter(year == 2023,  
15             cause == "Power Gen") %>%  
16    st_filter(counties %>%  
17                 filter(NAME %in% c("Pend Oreille", "Sp-  
18     mutate(x = st_coordinates(geometry)[,1],  
19             y = st_coordinates(geometry)[,2])  
20  
21 tot <- sum(fires_ne$acres_burned)  
22  
23 tot_2023 <- sum(fires$acres_burned)  
24  
25 pct <- tot/tot_2023
```

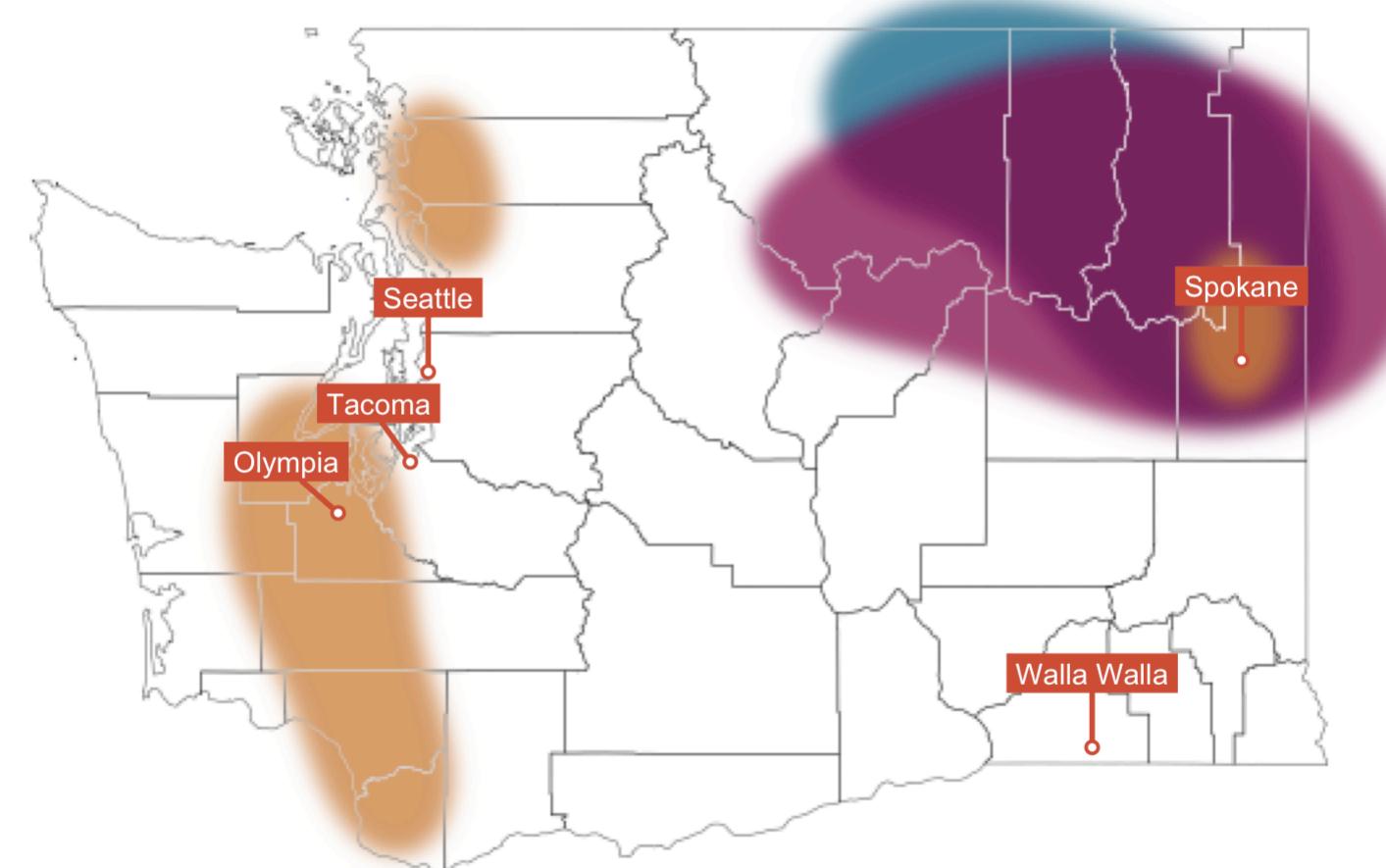


Patterns

📎 `{ggdensity}` has a `geom_hdr()` functions that provides a more visually appealing way to show point density, with a fairly easy-to-understand legend. You could even give it a blur with `{ggfx}` to make it seem more gestural than literal

```
1 fires_hdr <- fires %>%
2   mutate(x = st_coordinates(geometry)[,1],
3         y = st_coordinates(geometry)[,2]) %>%
4   st_drop_geometry() %>%
5   filter(cause %in% c("Natural", "Power Gen", "Fireworks")) %>%
6   mutate(cause = ordered(cause,
7                         levels = c("Natur
8
9
10 ggplot() +
11   as_reference(with.blur(
12     geom_hdr(
13       data = fires_hdr,
14       aes(
15         x = x,
16         y = y,
17         group = cause,
18         fill = cause
19       ),
20       probs = c(.33),
21       n = 300,
22       # Expand the limits to give the geometry some breathing room
23       xlim = st_bbox(counties)[c(1, 3)] * c(.9, 1.1),
24       ylim = st_bbox(counties)[c(2, 4)] * c(.9, 1.1),
25       alpha = .8
26     )
27   )
```

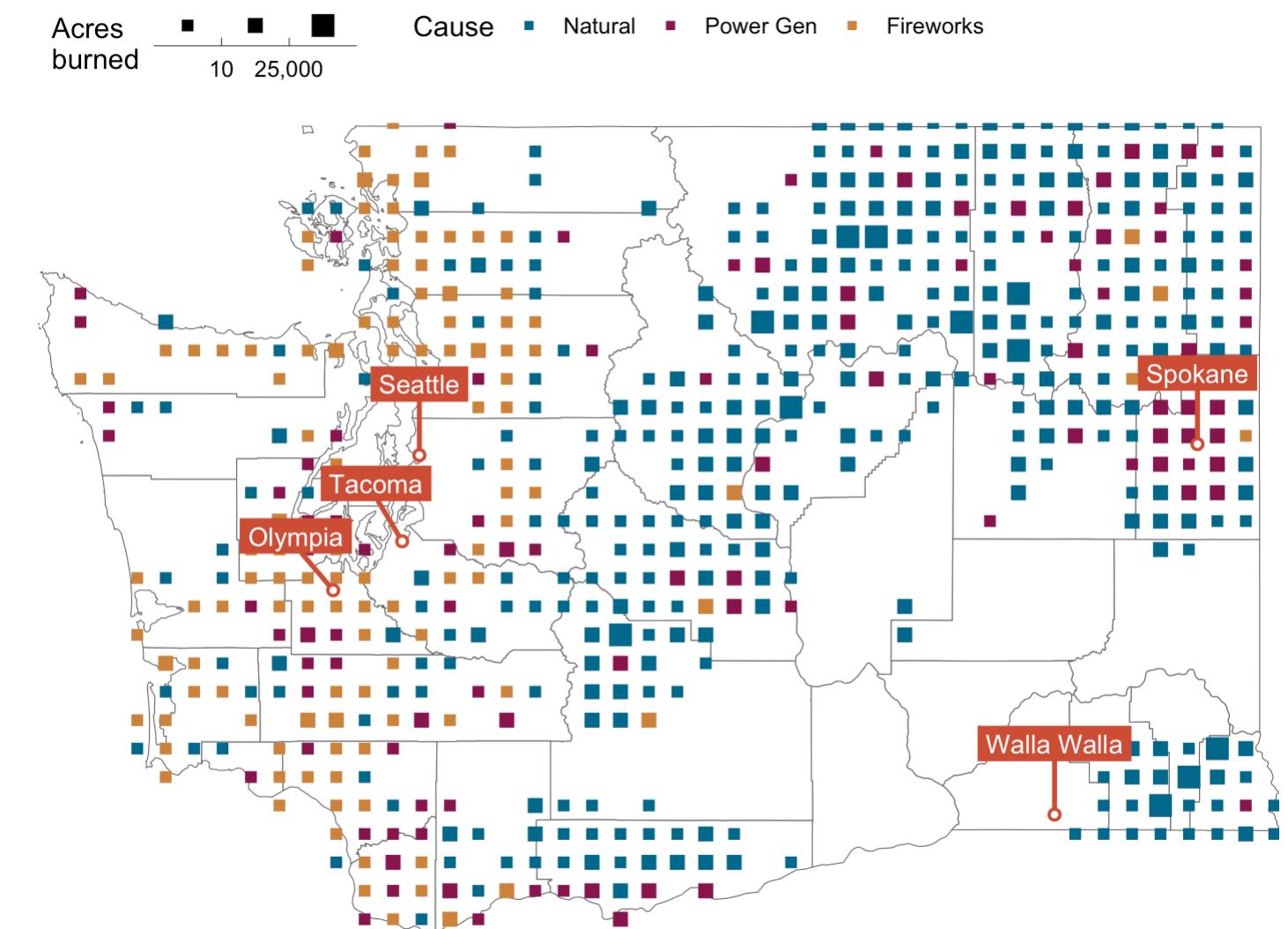
Cause Natural Power Gen Fireworks



Patterns

Then again, sometimes using a grid is an effective way of showing spatial patterns, and you can add an extra visual variable by using points that match your grid and varying their size

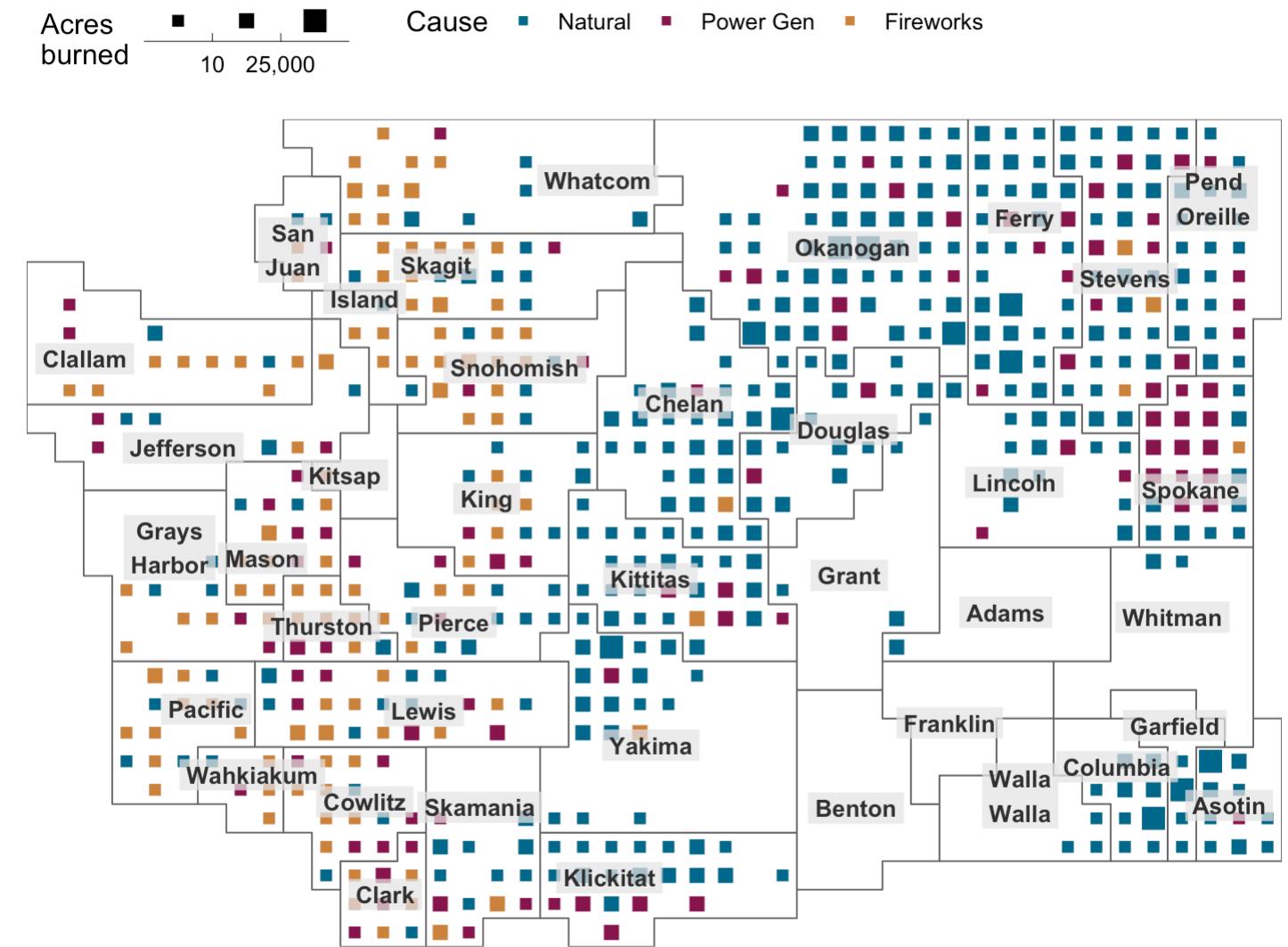
```
1 wa_grid <- st_make_grid(counties,
2                           cellsize = 20000)
3   st_sf() %>%
4     rowid_to_column("gridid")
5
6 fire_grid <- st_join(wa_grid, all_fires %>%
7   filter(acres_burned >
8         cause %in%
9       group_by(gridid, cause) %>%
10      summarise(acres_burned = sum(acres_burned)) %>%
11      group_by(gridid) %>%
12      filter(acres_burned == max(acres_burned, na.rm = TRUE)) %>%
13      ungroup() %>%
14      mutate(acres_burned = ifelse(is.na(acres_burned), 0, acres_burned))
15      st_centroid() %>%
16      mutate(cause = ordered(cause,
17                            levels = c("Natural", "Power Gen", "Fireworks"))
18
19 ggplot() +
20   geom_sf(
21     data = counties,
22     fill = "white",
23     color = "gray40",
24     linewidth = .2
25   ) +
26   geom_sf(data = fire_grid,
```



Patterns

Though you might want to make your other layers conform to this grid, too.

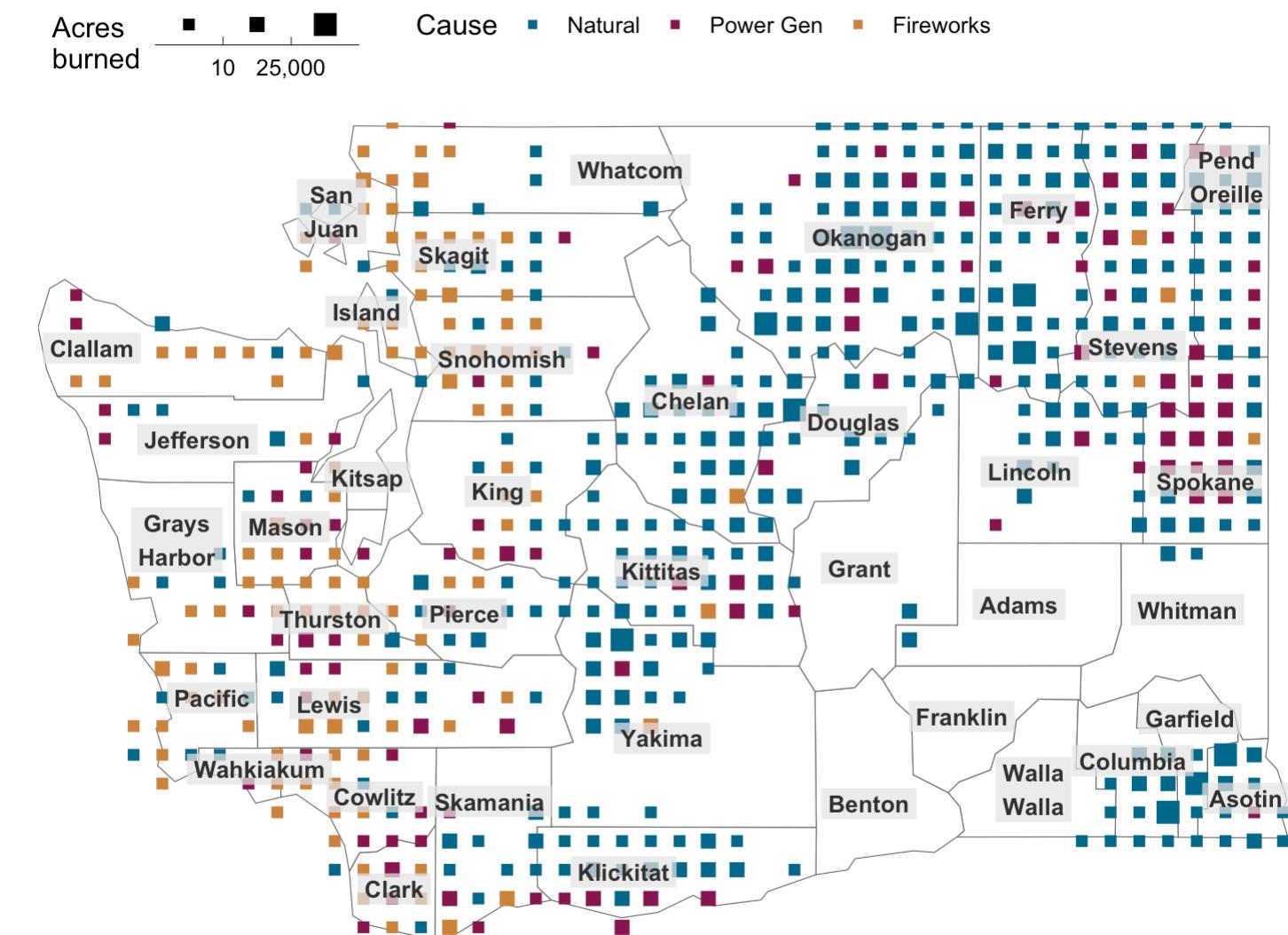
```
1 wa_grid <- st_make_grid(counties,
2                           cellsize = 20000)
3
4   st_sf() %>%
5     rowid_to_column("gridid")
6
7 county_grid <- wa_grid %>%
8   st_join(counties, largest = TRUE, left = FALSE) %>%
9   group_by(county = NAME) %>%
10  summarise()
11
12 county_grid_lbls <- county_grid %>%
13   pull(geometry) %>%
14   st_inscribed_circle() %>%
15   st_sf() %>%
16   filter(!st_is_empty(geometry)) %>%
17   st_centroid() %>%
18   transmute(name = county_grid$county,
19             x = st_coordinates(.)[, 1],
20             y = st_coordinates(.)[, 2])
21
22 fire_grid <- st_join(wa_grid, all_fires %>%
23   filter(acres_burned >
24         cause %in%
25       group_by(gridid, cause) %>%
26       summarise(acres_burned = sum(acres_burned)) %>%
27       group_by(gridid) %>%
```



Simplification

- While we're modifying geometries, {rmapshaper} is a great package for simplifying geometries, which can be especially useful for rendering large spatial datasets

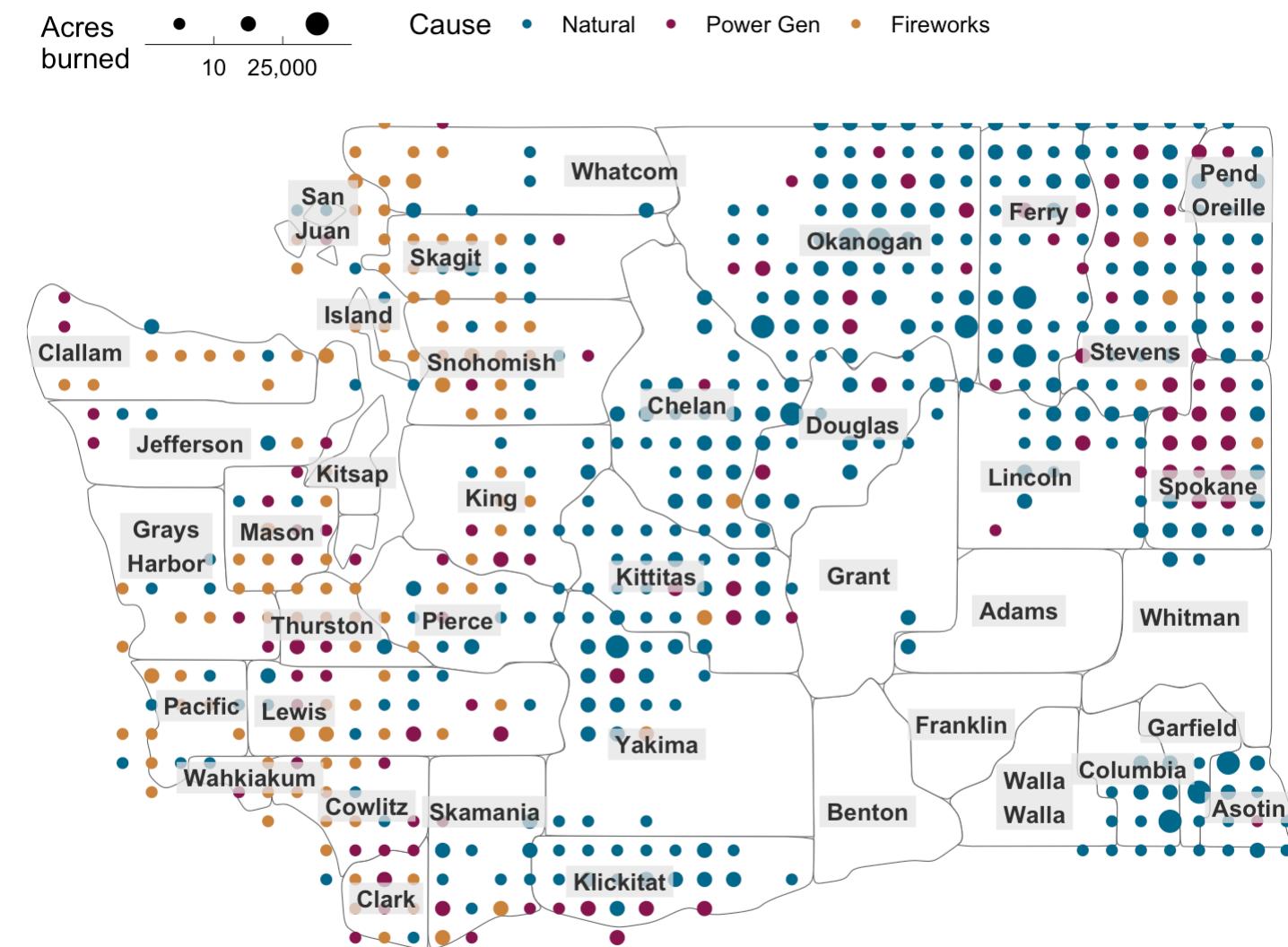
```
1 counties_simp <- ms_simplify(counties, keep = .02, keep_shapes = T
2
3 counties_simp_lbls <- counties_simp %>%
4   pull(geometry) %>%
5   st_inscribed_circle() %>%
6   st_sf() %>%
7   filter(!st_is_empty(geometry)) %>%
8   st_centroid() %>%
9   transmute(name = counties_simp$NAME,
10           x = st_coordinates(.)[, 1],
11           y = st_coordinates(.)[, 2])
12
13 ggplot() +
14   geom_sf(
15     data = counties_simp,
16     fill = "white",
17     color = "gray40",
18     linewidth = .2
19   ) +
20   geom_sf(data = fire_grid,
21           aes(color = cause, size = acres_burned),
22           shape = 15) +
23   geom_label(
24     data = counties_simp_lbls,
25     aes(
26       x = x,
```



Simplification

📎 {smoothr} can also be used to add rounded corners to your polygons or line geometries for a more stylized look

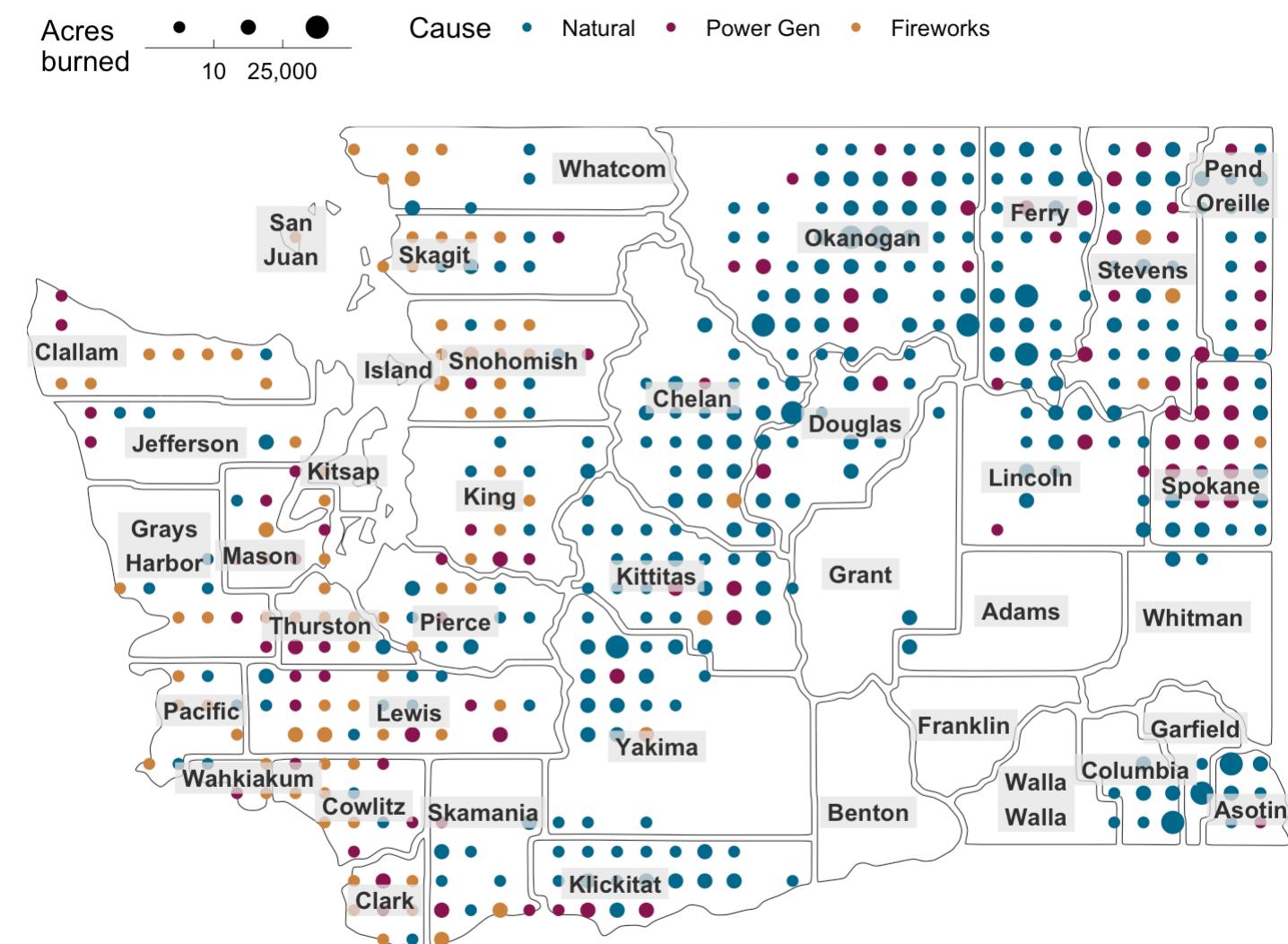
```
1 counties_simp <- counties %>%
2   ms_simplify(keep = .02, keep_shapes = TRUE) %>%
3   smooth(method = "ksmooth",
4         smoothness = .35) %>%
5   # Sometimes you can create geometry errors when smoothing, so
6   st_make_valid()
7
8 counties_simp_lbls <- counties_simp %>%
9   pull(geometry) %>%
10  st_inscribed_circle() %>%
11  st_sf() %>%
12  filter(!st_is_empty(geometry)) %>%
13  st_centroid() %>%
14  transmute(name = counties_simp$NAME,
15            x = st_coordinates(.)[, 1],
16            y = st_coordinates(.)[, 2])
17
18 ggplot() +
19   geom_sf(
20     data = counties_simp,
21     fill = "white",
22     color = "gray40",
23     linewidth = .2
24   ) +
25   geom_sf(data = fire_grid,
26           aes(color = cause, size = acres_burned),
```



Simplification

Did you know you can use negative values for `st_buffer()`?

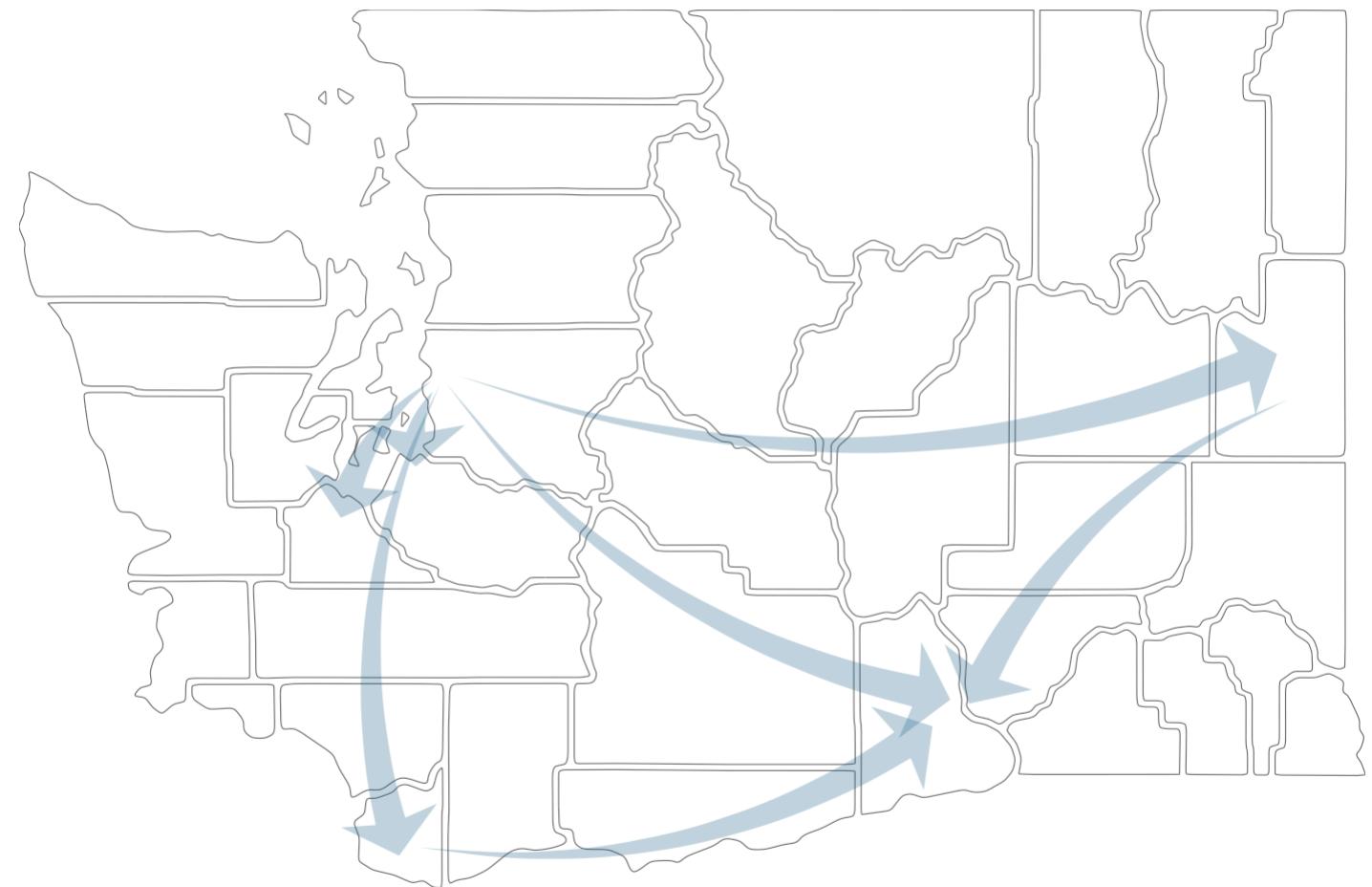
```
1 counties_simp <- counties %>%
2   st_buffer(-2000) %>%
3   ms_simplify(keep = .02, keep_shapes = TRUE) %>%
4   smooth(method = "ksmooth", smoothness = .35) %>%
5   # Sometimes you can create geometry errors when smoothing, so
6   st_make_valid()
7
8 counties_simp_lbls <- counties_simp %>%
9   pull(geometry) %>%
10  st_inscribed_circle() %>%
11  st_sf() %>%
12  filter(!st_is_empty(geometry)) %>%
13  st_centroid() %>%
14  transmute(name = counties_simp$NAME,
15            x = st_coordinates(.)[, 1],
16            y = st_coordinates(.)[, 2])
17
18 ggplot() +
19   geom_sf(
20     data = counties_simp,
21     fill = "white",
22     color = "gray20",
23     linewidth = .2
24   ) +
25   geom_sf(
26     data = fire_grid %>%
```



Symbols - Lines

📎 Brand new package `{ggarrown}` offers a completely new way of visualizing flowlines from point to point with arrows that aren't the lackluster defaults from `{ggplot2}`. Lots of customizability!

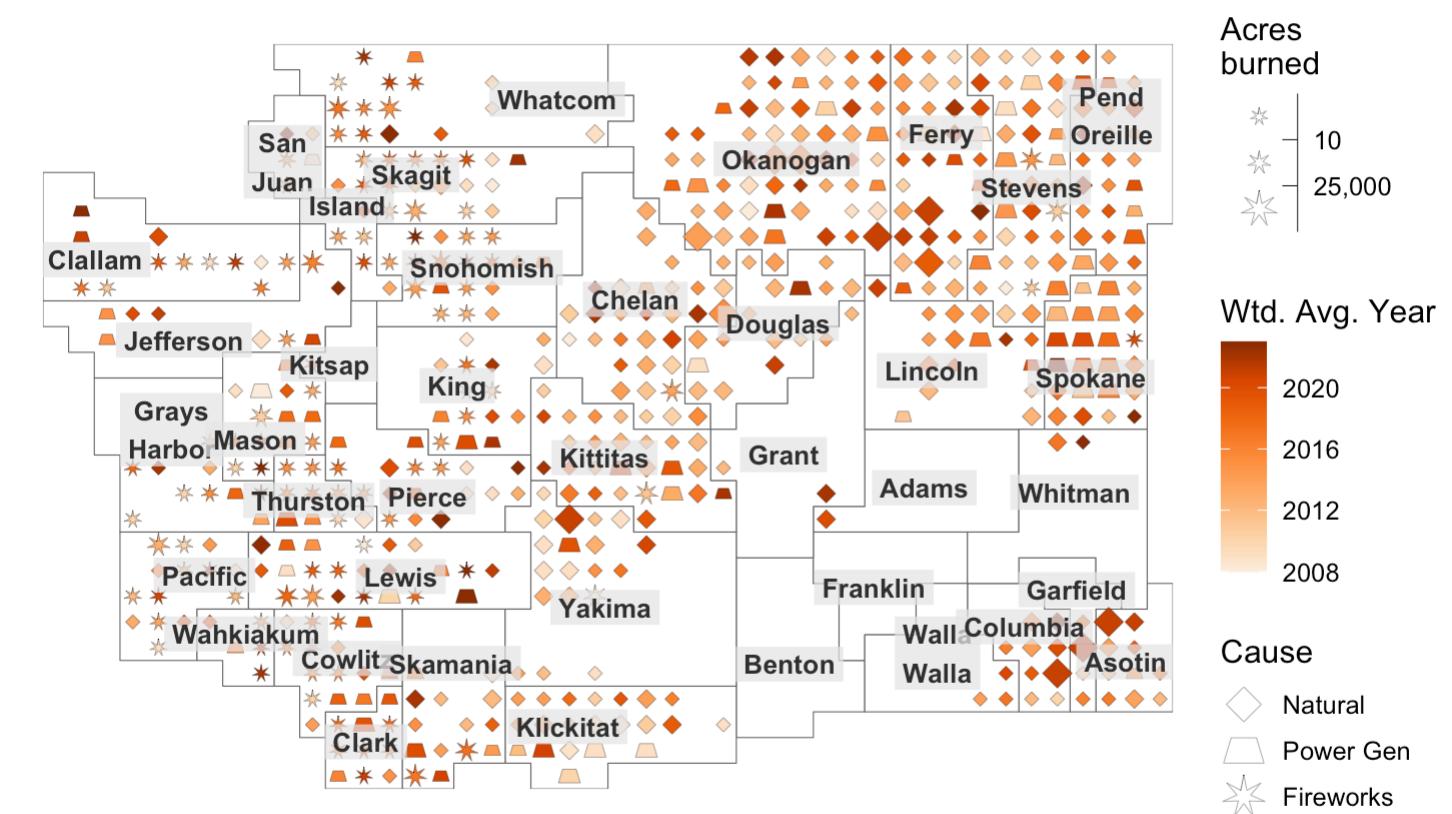
```
1 lines <- st_as_sf(c("LINESTRING (-122.3207 47.52453, -122.418 47.  
2 "LINESTRING (-122.2829 47.54782, -122.5151 45.6868)", "LINESTRING  
3 "LINESTRING (-122.2388 47.55275, -119.3356 46.312)", "LINESTRING ("  
4 "LINESTRING (-117.3921 47.49202, -119.2271 46.29595")"),  
5 crs = 4326) %>%  
6     st_transform(3857) %>%  
7     st_sf() %>%  
8     rowid_to_column("lineid") %>%  
9     st_cast("POINT") %>%  
10    mutate(x = st_coordinates(.)[,1],  
11           y = st_coordinates(.)[,2]) %>%  
12    st_drop_geometry() %>%  
13    group_by(lineid) %>%  
14    mutate(pt = 1:2) %>%  
15    pivot_wider(names_from = pt,  
16                  values_from = c(x, y))  
17  
18 ggplot() +  
19   geom_sf(  
20     data = counties_simp,  
21     fill = "white",  
22     color = "gray40",  
23     linewidth = .2  
24   ) +  
25   geom_arrow_curve(  
26     data = lines,
```



Symbols - Points

📎 {ggstar} doesn't get enough love! It's a great way to show point data with a little more flair than the standard {ggplot2} PCH shapes

```
1 star_fire_grid <- st_join(wa_grid, all_fires %>%
2                           filter(acres_burned >
3                                 year >= 2
4                                 cause %in%
5                                 !is.na(ca
6                                 left = FALSE) %>%
7                           st_drop_geometry() %>%
8                           group_by(gridid, year, cause) %>%
9                           summarise(acres_burned = sum(acres_burned)) %>%
10                          ungroup() %>%
11                          complete(year, cause, nesting(gridid), fill = list(acres_burne
12                           filter(!is.na(cause)) %>%
13                           group_by(gridid, cause) %>%
14                           mutate(cause_acres = sum(acres_burned)) %>%
15                           group_by(gridid) %>%
16                           mutate(main_cause = first(cause[which(cause_acres == max(cause
17                           group_by(gridid, main_cause, year) %>%
18                           summarise(acres_burned = ifelse(is.na(acres_burned), 0, acres_
19                           group_by(gridid, main_cause) %>%
20                           summarise(year = weighted.mean(as.numeric(year), w = acres_bur
21                               acres_burned = sum(acres_burned)) %>%
22                           ungroup() %>%
23                           inner_join(wa_grid) %>%
24                           st_as_sf() %>%
25                           st_centroid() %>%
26                           mutate(main_cause = ordered(main_cause,
```



Symbols - Points

📎 {gg(svg)} give you the ability to use SVGs as points in your plots, even our old pal Clippy!

```
1 # Read in a downloaded static SVG
2 svg_text <- paste(readLines("./01_data/01_raw/clippy.svg"), collapse = "")
3
4 ggplot() +
5   geom_sf(
6     data = counties,
7     fill = "white",
8     color = "gray40",
9     linewidth = .2
10 ) +
11   geom_point_svg(
12     data = clippy[1,],
13     aes(x = x, y = y),
14     size = 5,
15     svg = svg_text,
16     svg_width = 600,
17     svg_height = 800,
18     defaults = list(fill_inner = 'white', fill_outer = 'red')
19   ) +
20   geom_label_repel(
21     data = clippy[1,],
22     aes(
23       x = x,
24       y = y,
25       label = str_wrap(lines, 24)
26     )
27   )
```



Thank You, Creators

@hadley, @thomasp85 {gg_____}, @edzer {sf}, @clauswilke {ggplot2 and others},
@paleolimbot {ggspatial}, @ateucher {rmapshaper}, @coolbutuseless {ggsvg}, @slokow
{ggrepel}, @walkerke {tidycensus}, @jamesotto852 {ggdensity}, @xiangpin {ggstar},
@hughjonesd {ggmagnify}, @mstrimas {smoothr}

and many more!